Spring 3-1-2009

A Hierarchical Approach for Useable and Consistent CAPEC-based Attack Patterns

Patrick H. Engebretson
Dakota State University

A HIERARCHICAL APPROACH FOR USEABLE AND CONSISTENT CAPEC-BASED ATTACK PATTERNS

A dissertation submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctorate of Science

in

Information Systems

March, 2009

By
Patrick H. Engebretson

DAKOTA STATE dsu UNIVERSITY

DISSERTATION APPROVAL FORM

We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Doctorate of Science in Information Systems.

Student Name: ____

Dissertation Title: ____

Dissertation Chair:_________________________________Date: _____________

Committee member:________________________________Date: _____________

Committee member:________________________________Date: _____________

Committee member:________________________________Date: _____________

ACKNOWLEDGEMENTS

This dissertation is dedicated to all of those people who have played a part in its creation. I may have written the words, but I couldn’t have completed it without the countless sacrifices of those around me.

To my parents Marge and Larry Engebretson, both teachers, who taught me the value of hard work and worthiness of dedicating yourself to the education of others. To my mother and father-in-law, who supported our family and encouraged us to never take our “eye off the prize”. A special thanks to “Grammy Joyce” for the many hours of babysitting while I worked to complete this project. To my daughter Maggie, who ultimately inspired me to leave my job at the bank and pursue higher-education. To my daughter Molly, whose smile and excitement make the long hours and hard work worthwhile. To my brother, for understanding the many cancelled trips and hurried phone conversations when my dissertation called.

Of all those who played a part in my success, no one deserves more credit than my wife. To Lori Ann Engebretson, the love of my life. She encouraged me to walk away from an easy-going, well paying job and pursue my dream of teaching. She encouraged this path knowing that money would be tight and all roads would lead away from Watertown. Her support has never faltered or wavered. She is the one who has had to sacrifice more than anyone else. When I was crabby, stressed, tired, and unpleasant or when our girls were unruly and she was exhausted, she continued to hold our family together through unmatched love, dedication, and faith. I couldn’t have asked for a more perfect wife, friend, and soul mate. This dissertation belongs as much to her as to me.

To each of my committee members, thanks for being part of my DSc journey. With all of your other commitments to the university, you dedicated time to take part in my dissertation research and provided valuable insight into the completion of my degree.

To Dr. Sarnikar for teaching me the meaning of evaluation and helping me to overcome my fear of "validation". You pushed me to validate my work and I'm grateful for the confidence it has inspired.

To Dr. Rich Avery, thank you for agreeing to serve as the external member, for always providing timely feedback, and for your openness and willingness to work with the committee.

To Dr. Kevin Streff, who worked extremely hard on my behalf to secure my position and acceptance into the D.Sc. program, for showing me the ropes when I first arrived on campus, for being both a mentor and a friend over the past three years, and for providing an excellent example of what can be accomplished through hard work and dedication.

To Dr. Tom Halverson, Dean of the College of Business and Information Systems, for taking a chance on a new teacher, for trusting me enough to give me classes which pushed me outside of my comfort zone, and for giving me the freedom and power to teach my classes in my own way. Because of the opportunity you have given me, I have found my true calling in life; teaching.

To Dr. Josh Pauli, my friend and mentor, thank you for your countless hours of guidance, at the office and at home, dissertation related, and other. For your quick turn-arounds and willingness to drop everything to talk "diss". Thanks for always reminding me that "It's all worth it in the end!". It's hard to believe how far we've come from an idea that was hatched in your garage on a hot August day. There is little doubt in my mind that you’ve read this dissertation more times than the rest of the world will read combined. You’ve taught me how to research and how to write. I’m truly thankful for all of your efforts.

And finally; to wrap up on a lighter note and keep people guessing, I want to thank (in no particular order and for reasons too varied to mention): my students, McDonalds, Subway, and One Stop (all fast food actually), Mickeys, the Minnesota Vikings, the Tour de France, DefCon, 2600 magazine, Linus Torvalds, Hoglund & McGraw, the Matrix (just the first one), bicycles, XBOX 360 and PSP, Pandora Radio, and Lyle Lovett.

ABSTRACT

The inability to gather, analyze and share various aspects of an attack has made it difficult to effectively counter real-world information system attacks. The lack of a formally defined vocabulary which can express an “attacker’s-perspective” makes collaboration of academic research difficult. These problems lead to significant confusion by security managers and decision makers who are constantly bombarded by the media and security vendors attempting to describe or prevent the latest attack (Hoglund & McGraw, 2004).

The Common Attack Pattern Enumeration Classification (CAPEC) Release 1 Dictionary defines attack patterns as a formalized representation of a computer attacker’s tools, methodologies, and perspective (capec.mitre.org, 2007). CAPEC provides a formal definition of each attack by providing descriptive textual fields. These fields, defined as elements, provide explicit details for each identified attack pattern. The current CAPEC release includes a list of 101 specific information system attacks. Each attack pattern may include up to 30 elements to describe attack details.

While CAPEC has addressed the need to create a standard for representing and defining attacks from an attacker’s perspective, issues pertaining to usability and consistency exist. The goal of this research is to further refine and extend the CAPEC framework in order to provide usability and consistency. Issues of usability arise when CAPEC adopters attempt to leverage the Release 1 dictionary because of the sheer amount of information presented (Engebretson, Pauli, & Streff, 2008). Furthermore, while the

details of each attack pattern are extremely valuable, CAPEC does not provide a consistent level of documentation for each element among the 101 attack patterns.

Our approach includes three distinct processes to take the vast repository of CAPEC information and create a usable and consistent model for leveraging attack pattern details in system security configurations.

Process one creates a framework for general parent mitigations for each attack pattern. Parent mitigations are abstracted directly from the “solutions and mitigation” element in CAPEC and adds the appropriate National Institute of Standards and Technology (NIST) based Parent Mitigation element (Engebretson et al., 2008). These solutions and mitigations improve the resistance of the target software and reduce the likelihood of the attack’s success. They also improve the resilience of the target software and reduce the impact of the attack if it is successful.

Process two re-includes a Parent level Threat as an attack pattern element. The Parent Threat element places all 101 of the attack patterns into context without having to manually interact with both the full Release 1 dictionary and the CAPEC Classification Tree, thus ridding our approach of this manual research. We also use the Parent Threat element to provide structure in our hierarchy-based graphical models. Textual attack descriptions for viewing attack patterns are created to provide additional details about each attack pattern in a consistent manner.

Process three creates two security metrics, Knock-Out Effect (KOE) and Parent Mitigation Power (PMP), to provide usability to CAPEC. The addition of security metrics to our approach allows adopters to quickly and accurately leverage the vast amount of

information provided by the CAPEC standard from both the individual attack pattern and parent mitigation perspectives.

The result of this dissertation is an approach for increasing the usability and consistency of the CAPEC standard. The use of a taxonomy for cataloging and organizing attacks can increase awareness and communication about attacks as well as provide a framework for collecting consistent data about each attack (Hansman & Hunt, 2005).

Process one abstracts nearly 400 unique mitigation strategies into one of 17 commonly accepted, Parent Mitigations. Process two re-includes the “Parent Threat” element into the dictionary to provide consistency and context to each attack pattern. The creation of graphical hierarchies and textual attack descriptions are used to provide CAPEC with visual and textual representations for each attack without becoming overwhelming to the user. The introduction of a defined hierarchy between descriptive elements assists with learning and processing attack patterns. The significance of this process is a much clearer and less convoluted picture of the attack, resulting in a more usable and appropriate element set.

Process three creates security metrics derived from defined mitigation strategies, which creates a measurable numeric value which can allow security personnel to make more informed security decisions, play "what-if" security scenarios, and quickly analyze the cost-benefit for mitigation strategies.

DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

____

<Student name>

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Background

In the United States and around the globe, information systems make up a critical component of communication, commerce, and control of the physical infrastructure (Benioff & Lazowska, 2005). Along with data, infrastructure components include networks, computers, routers, domain servers, switches, and transmission lines (Bishop, 2003). Taken together, these systems allow for the exchange and flow of information. These connections can be tied directly to one another through dedicated paths or indirectly through the ubiquity of interlaced, non-centralized networks. Unbounded networks, such as the Internet, represent a growing collection of interconnected systems, devices and organizations (Ellison et al., 1999). Because of their distributed nature and lack of central control, unbounded networks increase both risk and exposure to abuse. It is not possible for any system connected to an unbounded network to be completely immune from attack (Ellison et al., 1997). The digital infrastructure of unbound networks provides new areas and avenues for malicious exploitation leaving governments, corporations, and private citizens vulnerable to such attacks. The protection and securing of this infrastructure is vital, as their destruction would have an immediate impact on the economy, livelihood, and psychology of the nation (Chakrabarti & Manimaran, 2002; Lewis, 2006).

The US Department of Homeland Security defines critical infrastructure into 11 sectors (Lewis, 2006).

1. Agriculture
2. Water
3. Public Health
4. Emergency Services
5. Defense Industrial Base
6. Telecommunications
7. Energy
8. Transportation
9. Banking and Finance
10. Chemical and Hazardous Materials
11. Postal and Shipping

Each of the 11 critical infrastructures relies heavily on the use of information technology and interconnected systems through the use of unbounded networks (Ellison et al., ; Rinaldi, Peerenboom, & Kelly, 2001).

Integrating security throughout the entire organization has long been understood as being very important (Hoglund & McGraw, 2004).

A wide variety of standards and technologies have emerged to address the rise of security risks. Generally, these standards and technologies are grouped into one of four categories, which include: 1) standards and policies, 2) library and tools, 3) administrative and system management, and 4) physical tools (Wang & Wang, 2003). Standards and policies are a series of best practices that work to alleviate specific security issues. Libraries and tools are integrated directly into the software development process and have the ability to provide protection from the planning phase. Administrative and management technologies include any tool that a system administrator would use to guard against security attacks. Physical tools include physical and external hardware designed for the specific purpose of security protection (Wang & Wang, 2003). Most security managers attempt to provide system security by using a combination of these standards and technologies. Unfortunately, these standards and technologies alone are not enough to fully prevent all attacks from executing and causing harm to the software system.

The loss of confidentiality, integrity and availability of information systems due to security problems such as Trojan horses, backdoors, denial of services, viruses, worms, misuse, buffer overflows, and configuration errors continues to rise. Even though these attacks have been studied, the appropriate mitigation strategies and solutions are not well understood.

Each year, the number, severity and sophistication of computer, network, and software security attacks continues to increase at an alarming rate (Hansman & Hunt, 2005). The ability to organize, comprehend and disseminate these attacks is a critical component in defending against them. System administrators, managers, and security experts must be able to understand the individual characteristics of each attack as well as how the attacks relate to one another (Jajodia, 2007). As the complexity of systems, networks and software continues to grow, the ability to keep track of attack specific details and relationships becomes increasingly difficult.

The process of learning, dissecting and understanding computer, network, and software security attacks requires an extra ordinary amount of effort. The need for a standard which addresses multiple audiences is important as security depends on people in many different capacities, such as requirement specifiers, designers, coders, users, maintenance personnel, managers, and administrators (Neumann, 2004). The use of a formal language and defined structure provides a modular approach which eases the inclusion and discovery of new attacks as well as giving users an increased ability to predict new attacks (DeLooze, 2004).

In March of 2007, the National Cyber Security Division of the Department of Homeland Security in conjunction with Cigital and MITRE Corporation released an official dictionary of 101 attack patterns. The Common Attack Pattern Enumeration and Classification (CAPEC) Release 1 Dictionary provides an official schema and formal representation for defining attack patterns (Barnum & Amit, 2006a; capec.mitre.org, 2007). CAPEC further organizes attack patterns by gathering and displaying both primary and supporting data elements for each identified attack (Sean Barnum, 2007) .

1.2. Problem Definition

The inability to gather, analyze and share various aspects of an attack has made it difficult to effectively counter real-world information system attacks. The lack of a formally defined vocabulary which can express an “attacker’s-perspective” makes collaboration of research difficult. Simultaneously, this problem leads to significant confusion by security managers and decision makers who are constantly bombarded by the media and security vendors attempting to describe or prevent the latest attack (Hoglund & McGraw, 2004).

A taxonomy is needed in order to facilitate a comprehensive understanding of information system attacks (Chakrabarti & Manimaran, 2002). While a wide variety of network, computer and software security attack classifications have been suggested, very few have attempted to address more than one specific audience (Lindqvist & Jonsson, 1997). CAPEC provides a useful framework for classifying attacks, but each of the 101 attack patterns provides approximately 30 descriptive fields, thus making it difficult to implement. (Pauli & Engebretson, 2008a, 2008b).

While CAPEC has addressed the need to create an industry standard for representing attacks from an attacker's perspective, several issues pertaining to usability and consistency remain as introduced below.

1. **CAPEC's Release 1 Dictionary is inconsistent level of information for "Solutions and Mitigation" element.** CAPEC includes nearly 400 individually prescribed controls in the "Solutions and Mitigations" element. These controls can be used to mitigate or reduce the effects of the defined 101 attack patterns. The current level of detail documented in the "Solutions and Mitigations" element is inconsistent. Some attack patterns provide an extremely granular level of detail. For example, one of the prescribed mitigations for attack pattern 42 (MIME Conversion) calls for disabling "the 7 to 8 bit conversion by removing the F=9 flag from all Mailer specifications in the sendmail.cf file." (capec.mitre.org, 2007). This level of detail may lead CAPEC adopters to believe that they need not be concerned with MIME Conversion attacks if they implement a Microsoft Exchange server rather than a Sendmail-based email server. Such a mistake could lead to an increased attack exposure and a false sense of security. The reverse is also true; some attack patterns provide only a high level overview of potential mitigation strategies. Attack pattern 9 (Buffer Overflow in Local Command-Line Utilities)

includes the “Do not unnecessarily expose services” mitigation (capec.mitre.org, 2007). This is too vague and undefined to be of use. The Solutions and Mitigations is also inconsistent in its specificity of mitigation. As demonstrated in the example above some solutions are presented at the architectural level while others are presented at the system or product level.

2. **CAPEC’s Release 1 Dictionary is inconsistent use of elements to describe attack patterns.** In many cases attack pattern elements are missing completely. CAPEC’s disjointed structure leads to confusion and frustration when attempting to make use of the current CAPEC Dictionary (Engebretson et al., 2008). The inconsistent use of elements makes it problematic to discern the relationship, if any, between the descriptive fields. The lack of a defined and consistent structure makes it difficult for new adopters to fully understand the context of each attack. This problem is exacerbated when descriptive elements are missing. The current inconsistent use of elements and presentation of information represents a significant challenge to increased adoption of CAPEC (Engebretson et al., 2008; J Pauli & Engebretson, 2008a, 2008b).

3. **The volume of information presented to users is overwhelming**. CAPEC defines 101 unique attack patterns. Each attack pattern can make use of up to 28 elements to describe attack details. Given the number of attacks and volume of information presented about each attack, deep understanding of the CAPEC library is difficult (Pauli & Engebretson, 2008a). This issue is further complicated by the inability to

quickly and accurately discern related attack patterns. Ideally, a user interested in CAPEC attack patterns should be able to quickly and accurately identify the threat family that the particular attack pattern belongs to. The lack of a formally defined “Parent Threat” element results in a disjointed presentation. The parent threat data is currently available via the CAPEC website, but it is not part of the 101 formal attack pattern definitions. This structure leads to confusion and frustration when attempting to make use of the current CAPEC Dictionary (Engebretson & Pauli, 2008).

4. **CAPEC R1 does not include associated metrics to measure the effectiveness of chosen mitigation strategies.** The CAPEC Release 1 Dictionary does not include any metrics which can be used to measure the effectiveness of prescribed mitigation strategies. Metrics are a critical component in aiding security related decisions. The lack of a defined metric remains a significant hurdle to the widespread adoption of CAPEC outside of academia. The creation of a metric would provide value for many potential CAPEC adopters including software designers, administrators, managers and researchers (Engebretson & Pauli, 2008).

1.3. Objectives and Approach

Our objective is to develop and demonstrate an approach that meets the needs of the problem definition.

1. Create a Parent Mitigation element for inclusion into the CAPEC standard to provide consistency to the currently given child mitigations. This objective will

simultaneously create a manageable and serviceable list of accepted mitigation strategies.

2. Creation of an enhanced CAPEC view to augment the existing CAPEC standard by re-include the Parent Threat element into the view to provide logical grouping of the 101 Attack Patterns at the Parent Threat level.
3. Further refine the enhanced CAPEC view by trimming the element set. Only descriptive elements which have an entry in each of the 101 attack patterns will be considered for inclusion into the view. This will provide a consistent framework for viewing the details of each attack pattern.
4. Create a graphical representation and textual description of each attack pattern for purpose of viewing information in a condensed and meaningful way. This will provide contextual information for each attack.
5. Create security metrics from the CAPEC standard which can be used to make security related decisions. These metrics provide a numeric value to help make security decisions for different situations that include specific threats.

Our objectives are accomplished through the creation of an approach that includes three processes which provide a level of consistency and standardization to the CAPEC library that it had previously lacked. Our models specify which information needs to be documented for each attack and how that information is documented. We also provide context through the use of standardized threats and mitigations. These threats and mitigations frame each attack and provide relationship data between each attack element.

Our approach is best understood when broken down into three distinct processes which provide a level of consistency to make the CAPEC library more useable for multiple audiences including requirement specifiers, designers, coders, users, maintenance personnel, managers, and administrators. A breakdown of our approach is introduced in Figure 1.

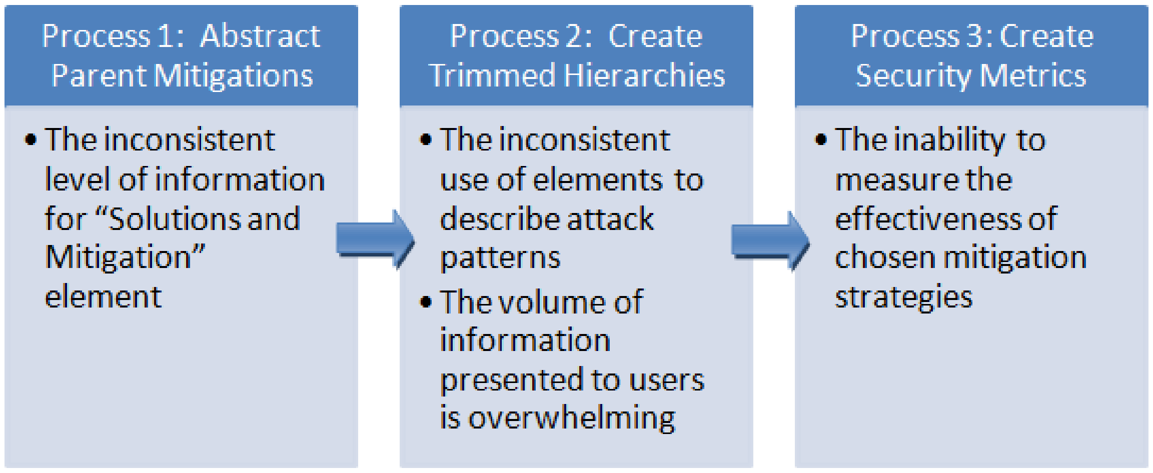

Process 1: Abstract Parent Mitigations
• The inconsistent level of information for "Solutions and Mitigation" element
Process 2: Create Trimmed Hierarchies
• The inconsistent use of elements to describe attack patterns
• The volume of information presented to users is overwhelming
Process 3: Create Security Metrics
• The inability to measure the effectiveness of chosen mitigation strategies

Figure 1. High-Level Overview of Our Approach with Problem Addressed.

Process one creates a framework for introducing a series of general Parent Mitigations for each attack pattern. Attack patterns can be defined as a formalized representation of an attacker's perspective including specific and clear terminology (Barnum, 2008). Parent mitigations are abstracted directly from the "solutions and mitigation" element currently defined in the CAPEC Release 1 Dictionary. CAPEC provides the following definition for the "Solutions and Mitigations" element: "the actions or approaches that can potentially prevent or mitigate the risk of this type of attack. These solutions and mitigations are targeted to improve the resistance of the target software and thereby reduce the likelihood of the attack's success or to improve the resilience of the target software and thereby reduce the impact of the attack if it is successful" (Barnum, 2008).

This element is a required field in order to make the standard effective for mitigating attacks (Engebretson et al., 2008). Ideally, a user concerned with a given attack pattern must be able to review the CAPEC standard for the particular attack and formulate a plan for reducing exposure to the attack. However, as previously highlighted some attack patterns provide details that are too granular while others provide information that are too vague. The objective of this process is to leverage this vast repository of attack pattern information and add an addition layer of information thus providing a uniform standard for mitigation strategies for each attack pattern.

Process one adds the appropriate NIST-based Parent Mitigation element. In the first step, mitigations are listed individually from the CAPEC "Solutions and Mitigation" element to create a list. Each mitigation is then matched up to a corresponding NIST Child Element from the NIST 800-53r2 control list (NIST, 2007). The final step in Process 1 is to abstract the NIST Child level control to its corresponding Parent level control. The Parent level control is then documented as a new mitigation element. This process is repeated for each control listed under the current CAPEC "Solutions and Mitigation" element.

Just as adding a level of consistency to the mitigation element is an important step in increasing usability, another benefit of this process is the creation of a unifying Parent Threat element (Engebretson et al., 2008). While this information is available via the CAPEC website, it is currently separated from the formal attack pattern definitions (capec.mitre.org, 2007). Process 2 re-includes a Parent level threat as an attack pattern element. The goal of adding the Parent Threat element to the formal definition set is to assist in placing all 101 of the attack patterns into context without having to manually interact with both the full Release 1 dictionary and the CAPEC Classification Tree. Adding Parent Threat as a formal element increases usability by simplifying the process of identifying threat families. We also use the Parent Threat element to provide structure and introduce the top node in our hierarchy-based model for viewing attack patterns. The purpose of this hierarchy is to logically group each attack pattern with related attack patterns from the same Parent Threat.

An illustration of this point can be seen by examining attack pattern 101, “Server Side Includes”. The CAPEC website provides the following elements to describe Attack Pattern 101: Attack Pattern ID, Typical Severity, Description, Attack Pattern Prerequisites, Typical Likelihood of Exploit, Methods of Attack, Examples-Instances, Attacker Skill or Knowledge Required, Resources Required, Probing Techniques, Solutions and Mitigations, Attack Motivation Consequences, Context Description, Injection Vector, Payload, Activation Zone, Payload Activation Impact, Related Weaknesses, Related Security Principles, Related Guidelines, Purpose, CIA Impact, Technical Context, and Source.

In order to determine the general threat classification, a CAPEC user is forced to navigate away from the “Full CAPEC Dictionary” on the CAPEC web site and search the “CAPEC Classification Tree”. The user must then wade through three levels of detail to uncover “Server Side Includes” (attack pattern 101) as a member of the “Injection” threat family.

Our hierarchy structure also increases usability by documenting relationships between the descriptive elements. In order to facilitate learning and foster a deeper understanding of attack patterns, our model reduces the number of descriptive elements displayed. Using a smaller number of elements presents adopters with a more manageable and usable dataset. Trimming the current CAPEC dataset and presenting the elements in a hierarchical fashion was a technique previously used to introduce students to the concept of attack patterns without overwhelming the audience (Pauli & Engebretson, 2008a). The trimmed element set provides usability, consistency, structure, and logical organization to the model. The top of this model will include the 11 Parent Threats and be tied together at the bottom of the hierarchy by 17 Parent Mitigations which were introduced in Process 1. An example of this model is shown in Figure 2.

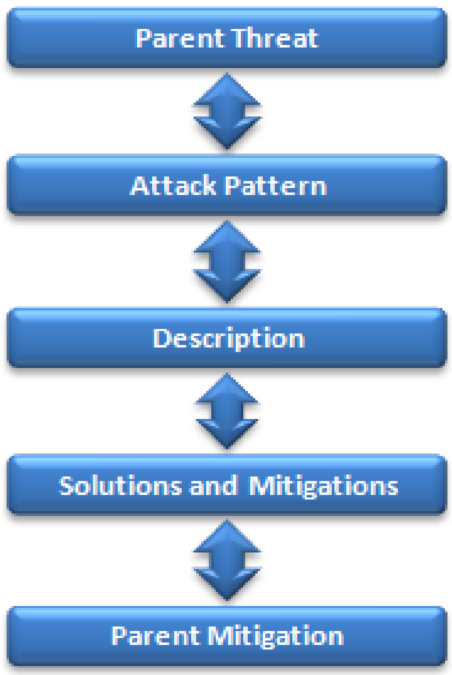


Figure 2. Trimmed Hierarchical Model for Viewing Attack Patterns.

The hierarchy can be traversed in either direction. Each attack pattern is framed by the use of Parent Threats at the top of the hierarchy and Parent Mitigations at the bottm. These elements serve to provide natural grouping and context. Process 3 creates two security metrics as part of each hierarchy and textual attack description. Our first metric, Knock-Out Effect (KOE), is the total number of Parent Mitigations abstracted in Process 1 for each attack pattern. KOE provides a metric for quickly determining the number of Parent Mitigations needed to fully mitigate an individual attack pattern. This metric remains the same for each attack pattern no matter what the system configuration is.

Our second metric, Parent Mitigation Power (PMP), is calculated at the conclusion of Process 3. PMP is a numeric summary expressing two types of mitigation in a “X.Y” format, where:

- X = Number of unique attacks that the parent mitigation helped to mitigate.
- Y = Total number of child mitigations that can be traced back to the parent mitigation.

It is important to note the goal of our approach is not to challenge or advocate replacement of the current CAPEC standard. Original element details will always be readily available in addition to the hierarchy and textual attack descriptions that our approach creates.

Our approach will make use of the design science research methodology. Specifically use the seminal work which formalized these concepts for the IS world to ensure that our methodology is appropriately applied (Hevner, March, Park, & Ram, 2004). Design science was chosen because of its natural fit with our approach. The goal of design science is to extend human and organizational capabilities through the creation of artifacts and models. Our artifact is a model which combines two federally funded standards, NIST and CAPEC, into a singular consistent framework.

Our work can be evaluated by examining each of the seven guidelines prescribed by Hevner et al., (2004).

- Guideline 1: Design as an Artifact
    - Requirement: "Design-science must produce a viable artifact in the form of a construct, a model, a method, or an instantiation." (Hevner et al., 2004)
    - How our work meets Guideline 1: Our work provides an innovative solution, in the form of a model, which solves a previously identified and unsolved problem.
- Guideline 2: Problem Relevance
    - Requirement: "The objective of design-science research is to develop technology based solutions to important and relevant business problems" (Hevner et al., 2004)
    - How our work meets Guideline 2: Our work is based on problems which have been identified, discussed, and accepted into the knowledgebase. Specifically, CAPEC is too large and inconsistent to be useful outside of a theoretical context. (Pauli & Engebretson, 2008a, 2008b).
- Guideline 3: Design Evaluation
    - Requirement: "The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods." (Hevner et al., 2004)
    - How our work meets Guideline 3: Our work can be viewed as functional, complete, and consistent. Furthermore our work was completed in an

iterative sequence which allowed several cycles of incremental activity and evaluation while the model was being developed. Our model solves each of the identified problem statements. We make use of informed argument, experimental and analytical validation techniques. We provide details of the validation techniques in the Discussion section of each chapter. We provide further validation through the execution and simulation of our model by use of a case study comprised of 11 attack patterns.

- Guideline 4: Research Contributions
    - Requirement: “Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.” (Hevner et al., 2004)
    - How our work meets Guideline 4: Our approach provides new and interesting contributions by providing an artifact which solves a heretofore unsolved problem. Our contribution is a model. This design artifact applies existing knowledge in new and innovative ways.
- Guideline 5: Research Rigor
    - Requirement: “Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.” (Hevner et al., 2004)
    - How our work meets Guideline 5: “The artifact itself must be rigorously defined, formally represented, coherent, and internally consistent” (Hevner et al., 2004). Our work clearly follows this guideline through the creation of a defined, represented, consistent model which is presented in Figure 1. We

provide further rigor through application of the knowledgebase. Both CAPEC and NIST are well established, highly respected standards. Our work relies on the use and application of these bodies to provide rigor.

- Guideline 6: Design as a Search Process
    - Requirement: “The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.” (Hevner et al., 2004)
    - How our work meets Guideline 6: Our artifact was created through an iterative process. Our development cycle consisted of construction, feedback, and incorporation of feedback into a new model. This process was repeated over a half a dozen times.
- Guideline 7: Communication of Research
    - Requirement: “Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.” (Hevner et al., 2004)
    - How our work meets Guideline 7: This dissertation and the subsequent academic publications serve to provide communication to technical audiences. Our approach as provided in this dissertation is well documented and can be used to establish repeatability for further research. The scenarios provided in each chapter, future grant applications, and whitepapers will provide communication to business oriented audiences. Research communication is also being achieved by incorporating the research results into teaching.

1.4. Results and Significance

The result of this dissertation is an approach for increasing the usability and consistency of the CAPEC standard. The use of a taxonomy for cataloging and organizing attacks can increase awareness and communication about attacks as well as provide a framework for collecting consistent data about each attack (Hansman & Hunt, 2005). While the current CAPEC standard provides a significant amount of information, there are tremendous variations in the depth and breadth of the “Mitigations and Solutions” currently outlined for each attack pattern. The result of our approach is the abstraction of nearly 400 unique mitigation strategies into one of 17 commonly accepted and federally standardized Parent Mitigations.

The introduction of a “Parent Mitigation” element into the dictionary provides consistency to the CAPEC Release 1 Dictionary. Because the current “Mitigation and Solutions” element provides valuable information, we are not advocating its removal. One intention of our approach is to add the “Parent Mitigation” element to provide a more manageable number of mitigations. This is a valuable step to the increased adoption and wide spread acceptance of the CAPEC Release 1 Dictionary.

The re-inclusion of a “Parent Threat” element into the dictionary provides consistency and context to the CAPEC Release 1 Dictionary. We present a new model for presenting CAPEC attack patterns by refining nearly 30 descriptive elements to provide a standardized set of useable and consistent elements. The creation of a graphical hierarchy provides CAPEC with a new visual representation for each attack without becoming overwhelming to the user. The introduction of a defined hierarchy between descriptive elements assists with learning and processing attack patterns. The significance of this process is a much clearer and less convoluted picture of the attack, resulting in a more usable element set.

The creation of security metrics derived from defined mitigation strategies increases the usability of CAPEC for several audiences. This process creates measurable numeric values which can allow security personnel to make more informed security decisions and play "what-if" scenarios.

A deep understanding of attack patterns can lead to the permeation of security throughout an organization, as well as heighten awareness of known exploits, vulnerabilities and weaknesses (Gegick & Williams, 2005). Integrating attack pattern knowledge into managerial level IT security decisions can result in a higher level of security by creating less exposure to identified bugs and known flaws (Hoglund & McGraw, 2004). Attack patterns can be used by developers, administrators and managers to provide a deeper understanding of security (S. Barnum, 2007).

1.5. Outline

The study is structured where Chapter 2 covers related work. Chapter 3 covers Process 1 of our approach for abstracting Parent Mitigations from the CAPEC attack pattern dictionary. Chapter 4 covers Process 2 of our approach of formally re-including the Parent Threat element into the attack dictionary. Chapter 4 also covers the new models created for viewing and using CAPEC attack patterns. This process includes trimming the element set, defining a hierarchy, and creating a graphical representation and textual attack description for each attack. Chapter 5 covers the creation, explanation, and use of our

Knock-Out Effect and Parent Mitigation Power security metrics. This is Process 3 of our approach. Chapter 6 is the Conclusions reached from this study.

2. LITERATURE REVIEW

2.1. Risk Assessment

The identification and mitigation of risks to information systems are paramount to the sustainability and survival of organizations (Rowe, 1977; Stoneburner, Goguen, & Feringa, 2002). The study and analysis of risk has become common practice throughout several industries including medical, insurance, earth science, financial, investment, public health, environmental, engineering and economics. The concept of studying, analyzing and scientifically framing the risk assessment procedure specifically for use in protecting and safeguarding information systems has been grossly under-managed and underutilized (Coleman & Jamieson, 1991; Farbey, Land, & Targett, 1992; Willcocks, 1992).

Information technology risks can be defined as the probability of a threat to a system, the probability of a vulnerability being discovered, or the probability of equipment or software malfunctioning (Whitman & Mattord, 2003). Risk assessment is an analysis that identifies the risks and protection requirements for the system through a formal process. It is also a key component of risk management that brings together important information for officials regarding the identification of threats and vulnerabilities and includes the potential impact on an organization’s operations, assets or individuals which can result in the loss of confidentiality, integrity, or availability (Grance, Hash, & Stevens, 2003; McCumber, 2004). By identifying and computing the probability of a threat occurring and separately determining the ramifications of the particular threat, an organization can begin to determine its risk level (Blakley, McDermott, & Geer, 2001).

Early information system risk assessment models can be traced back to system security modeling. Security modeling allowed for the definition of relationships. In this model, users were defined as subjects and data was defined as objects. This process allowed for enforcing the state of information within a system (Bell, 1996).

Attackers are constantly evolving their attacks and technologies through the creation of new tools and the discovery of new vulnerabilities (Recipes). In order to be effective against such attackers, the risk assessment process must be updated regularly and allow for flexibility in dealing with these new threats and vulnerabilities (Myerson, 2002). The risk assessment process defines threats as that which could cause potential harm to resources or the organization; while a vulnerability is defined as weakness in the asset which could be exploited by a threat (Ciampa, 2005; Hansche, Berti, & Hare, 2003; Hoglund & McGraw, 2004).

The keys to completing a viable and accurate risk assessment are clear and complete documentation of the information system, its relationship to other systems, and the information system’s relationship to the business itself (P. Fung & Longley, 2003). The accurate documentation of each system and its contents naturally leads to a more precise risk assessment. Knowing where your critical information is stored and who has access to it is equally important as knowing the probability and impact of a particular threat to a system. Often times this documentation process is overlooked or simply not addressed. Because media outlets tend to sensationalize hacker activity and malicious code such as viruses and worms, many companies disproportionately invest in attempting to mitigate these types of risks(P. Fung & Longley, 2003). The blending of these two points can lead to disastrous results. A clear illustration of this problem was brought to light recently when a laptop containing the confidential records of 26.5 million retired veterans was stolen from the home of a Veterans Affairs employee. The largest security breach in the history of the United States Government was not the result of nefarious hacker activity or the use of some exotic code exploit, rather it was simple theft (Burger, 2006). Proper documentation and risk assessment would have prevented the employee from leaving the government facility with such a valuable asset.

As businesses continue to grow and become more dependent on large scale computing systems, managers and organizations must learn to effectively identify and assess risks to these systems. Organizations have several choices and methodologies for attempting to quantify risk. Bayesian Probabilistic Risk Analysis is the process of risk management which includes identifying system weaknesses and reducing the probability of the particular system from being impacted by the exposed weaknesses (Ali, Hilton, & Peter, 1985). Bayesian risk analysis was originally developed for use in the nuclear power industry. A measurement of risk can be determined by answering four fundamental questions (Ali et al., 1985; Bedford & Cooke, 2001).

- What can go wrong?
- How frequently can it be expected to happen?
- What would be its consequences?
- How certain are we about the answers to the first three questions?

While much has changed through the use of advanced computer modeling and the creation of complex risk assessment software, the answers to these four questions can still provide a highly useful and accurate level of information system risk analysis.

The ability to defend an information system depends upon fully understanding the risks associated with that system and applying controls commensurate with the defined level of risk (Holden, 2003). This process of risk assessment helps organizations and managers appropriately spend time and money defending and protecting assets which need it most. In this way, risk assessment can be seen as a productivity tool that saves the organization time, money and reputation.

While several common underlying themes are often found in the risk assessment process such as, risk = impact x probability, there are often many different and widely accepted models used to complete the actual risk assessment (Woerner, 2007). Some methodologies focus on system failure to help identify risk (Gautam, Kenneth, & Kazuhiko, 1989). These models present a qualitative modeling technique to enhance the risk assessment process and facilitate the design of a risk assessment system. This approach helps overcome uncertainties associated with the unpredictability of human behavior and the failure rate of information systems, which must be factored into an overall risk rating (Gautam et al., 1989).

Other approaches call for the combined use of a knowledge based system and qualitative problem solving which can result in the creation of a generic and portable risk assessment tool (Gautam et al., 1989). A prevalent theme in the use of such knowledge based systems is the incorporation of event and fault trees. Event and fault tree analysis involves identifying unique potential failure as individual “tree-roots or trunks”, then properly identifying each of the potentially impacted system as a branch on the tree. The result of this concept is that given a particular failure, a detailed list of all potentially impacted systems can be accurately generated (Haasl, Roberts, Vesely, & Goldberg, 1981).

One of the primary advantages of developing a knowledge based system using fault tree analysis is that it provides for an excellent tool to model “what-if” scenarios. By examining the potential system failures, organizations and managers can get a broad and accurate picture of potential risk.

Another popular method for measuring risk is through the concept of Annualized Loss Expectation. Annualized Loss Expectation helps to quantify risk in terms of a financial definition where companies predict a specific value or cost associated with the occurrence of a particular risk (Blakley et al., 2001). Using this model, an organization calculates risk by multiplying a specific dollar amount against the probability of the risk’s occurrence. Cost is estimated by totaling both the direct and indirect dollar amounts over the course of one year, which are related to the occurrence of the risk. Examples of direct and indirect dollar amounts include physical damage, equipment replacement, labor costs to repair, decreased employee productivity, lost sales, reputation damage, and legal costs. Probability is determined by weighing the likelihood of a risk event on a 1 to “x” scale. This probability is then multiplied by the cost associated with the annual loss resulting in a final dollar value which is representative of risk for the particular system (Visintine, 2003).

Others methodologies have taken a different approach to defining the risk assessment process. One model defines risk assessment in six distinct steps (Ye, Barry, & Betsy, 2006). This approach begins with identifying a cost factor rating system. Once the rating system has been defined, risks are identified. The next step is assigning risk probability. This is followed by analyzing risk severity where an overall risk can be normalized on a scale from 1-100. The scale of 1-100 can then be disseminated into the following categories. Systems with an overall risk from 0-5 are considered “low risk”, 5-15 are marked as “moderate risk”, 15-50 are said to be “high risk” while 50-100 should be labeled as “very high risk”. The final step is to offer ways of reducing the presented risk (Ye et al., 2006).

Not every framework for assessing risk is concerned with both impact and probability. Some risk assessments focus solely on the probability of the risk occurring (Benoit, Michel, & Suzanne, 2005). This type of risk assessment can be especially useful when the impact or occurrence of a particular risk results in an irreversible state. The medical community provides several examples of this type of risk assessment. Often times medical risk assessments will focus solely on the probability of a particular disease because the resulting impact is death. In these cases, because the impact is irreversible, it is no longer given consideration (Benoit et al., 2005).

Many organizations mistakenly assume that increased spending on security investments will lead to a direct decrease in overall information system risk. The level of risk obtained from an organization’s completed risk assessment often determines the organization’s willingness to invest in appropriate security controls (Cavusoglu, Mishra, & Raghunathan, 2004). This type of organizational philosophy illustrates the importance of an appropriate and accurate risk assessment as there are clear implications to an organization’s financial health and bottom line.

The process of assessing risk is often too difficult to perform accurately without the use of automated software. Because of the complexity involved in accurate risk assessment, there is a need for the creation of an automated system (Hamdi & Boudriga, 2003).

Several standards have been introduced which can help organizations understand and complete the risk assessment process. ISO 27001, COBIT and NIST each provide guidance to ensure that risk assessment is handled appropriately (Brenner, 2007; NIST, 2002; von Solms, 2005).

Completing an accurate risk assessment is both valuable and necessary for an organization and its ability to properly protect its information system assets. Upon completion of the risk assessment process the organization and management staff will be ready to make precise and informed decisions with regard to budgeting, staffing and resource management. A well defined risk assessment leads to a deeper and more complete understanding of both the overall level of risk associated with the implemented technology and the risks associated with each individual system.

Upon completion of the risk assessment process, organizations have four options when addressing each risk (Blakley et al., 2001).

1. Liability Transfer: This occurs when a business is able to convey the risk to another party outside of the organization, effectively removing the responsibility or accountability for the particular risk. Most often this is accomplished through use of a disclaimer or other type of binding agreement.
2. Indemnification: Indemnifying risks is effectively insuring the organization against the occurrence of a particular risk.
3. Mitigation: This is the process of reducing identified risks through procedure, processes, or controls. Mitigations can be used to specifically reduced the impact, probability, or both impact and probability of a risk.
4. Retention: This is an organization's acceptance of a given risk. The specific risk is acknowledged and documented during the risk assessment process but no further steps are taken to reduce the current level of risk. This path is typically chosen

when the probability or impact of a risk occurring is very small. Retention is also a viable option when the “return on risk reduction spending” does not produce a meaningful return.

Accurate, complete, and meaningful risk assessment of a business’s information systems is a vital function for every organization across all industries. As standards continue to mature, processes continue to evolve, and new forms of risk assessment are introduced, organizations must find way to make sense of it all. A thorough risk assessment process gives companies a greater degree of power by ensuring risks have been accounted for and accurate, meaningful controls are in place (Peltier, 2005).

2.2. Attack Modeling

Modeling is a technique for organizing and viewing the details of a system or process. Models can provide relevant information through the process of abstraction and demonstration of relationships (Booch, Rumbaugh, & Jacobson, 1999). The goal of modeling is to better understand the systems or processes we are studying; modeling accomplishes this goal by providing the following (Booch et al., 1999; Scheer & Habermann, 2000).

- Aiding in the visualization of a system or process
- Specifying the structure or behavior of a system or process
- Providing a template which can be used to further advance, create, or study a system or process

- Providing documentation

Attack modeling is an approach for documenting commonly occurring computer, hardware, software, or network attack details while providing information in a structured and reusable form (Moore, Ellison, & Linger, 2001). Attack models can be used by system administrators, security analysts, system developers and managers. Attacks on information systems are often described via a single vulnerability or exploit and therefore lack the descriptive depth needed to fully capture the complexity and detail of most attacks (Templeton & Levitt, 2001). Utilizing modeling to describe attacks can help to fill in the appropriate level of detail.

Proper techniques for avoiding and mitigating information system attacks require an awareness of the risks associated with a particular system. Knowledge sharing through modeling can be useful for increasing awareness and collaboration of information system attack details (Steffan & Schumacher, 2002). Analysis, prediction and collaboration of attacks are valuable tools in the effort to protect information systems. The use of models to describe attacks can be extremely helpful in providing these tools (Daley, Larson, & Dawkins, 2002). A coherent model of exploits and vulnerabilities provides a solid foundation which can be used to educate system administrators as well as offering valuable details for appropriately responding to such attacks (Tidwell, Larson, Fitch, & Hale, 2001).

Attack Trees and Threat Models are two examples of common techniques used to organize and present details of attacks. Attack Trees offer a goal-oriented perspective for modeling the behavior and effects of an attack while Threat Modeling is often used to provide descriptions of threats at the code level (Schneier, 2000; Swiderski & Snyder, 2004).

2.2.1. Attack Trees

Attack trees provide a formal and systematic way of describing threats and counter measures to threats for a given information system (Schneier, 1999). Attack trees provide users with an ability to make calculations and compare various types of attacks. These graphical representations also allow us to visualize, enumerate and weigh information system attacks (Salter, Saydjari, Schneier, & Wallner, 1998). Each attack tree consists of a root and leaf structure. The end goal of the attack is represented as the tree's root while the various ways of achieving that goal are represented by its leaves. Despite this apparent simplicity, attack trees can be extremely useful in threat analysis (C. Fung et al., 2005). It is important to note that some leaves have sub-nodes (child-leaves). This structure indicates there are multiple steps needed to accomplish the goal. Each leaf node can be either conjunctive or disjunctive in nature (Tidwell et al., 2001). Conjunctive leaves are represented using an "AND" and inform the user that all of the child nodes must be completed in order to satisfy their parent node. Disjunctive leaf nodes are considered stand-alone alternatives and do not require other leaves to be satisfied before accomplishing its parent node. Disjunctive leaf nodes are represented using the "OR" designation. Upon completion of the attack tree each node can be evaluated and assigned a value of either "I" for impossible or "P" for possible depending on the probability of the attack. An example of a simple attack tree is introduced in Figure 3 (Schneier, 2000).

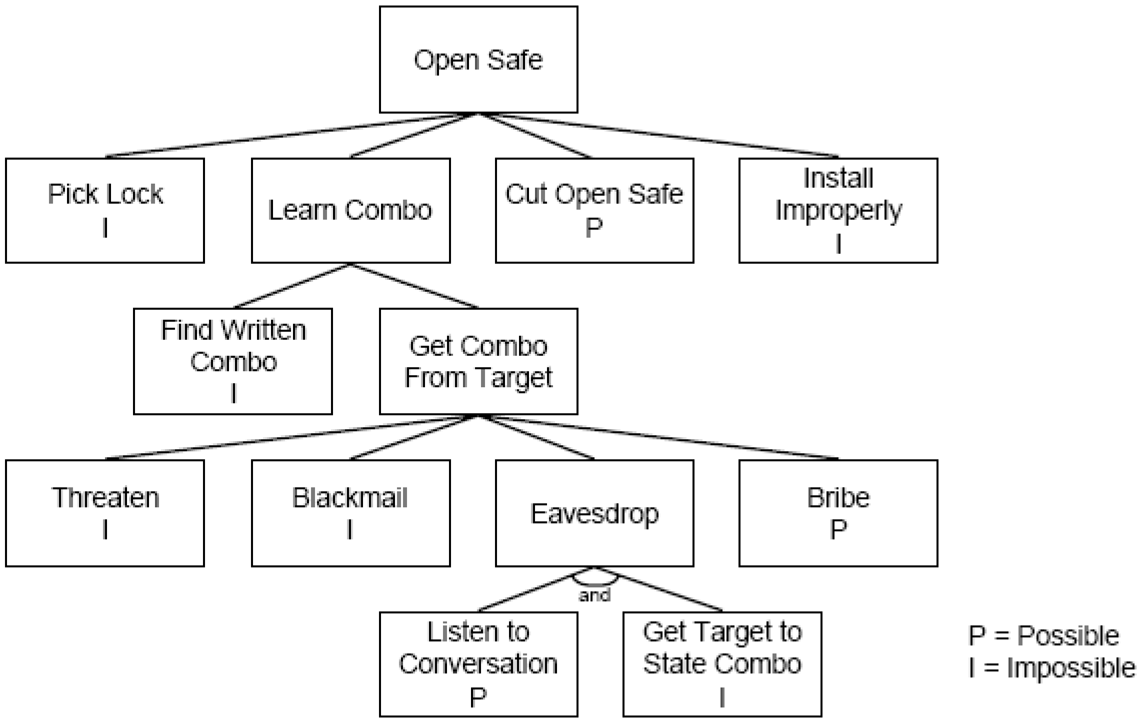


Figure 3. Attack Tree for Accessing a Physical Safe adopted from "Secrets and Lies" (Schneier, 2000).

Figure 3 illustrates the classic example of an attack against a physical safe (Schneier, 2000). The goal, represented by the root, is to gain access to a physical safe (open safe). The leaves, listed individually below the goal, represent different approaches for achieving the goal.

Upon completion of the attack tree, it is possible to assign a cost to each node. Doing so allows for further analysis and comparison of the various attack costs. Evaluating the costs of cyber and network attacks is an integral part of understanding both the risks and their mitigating countermeasures (Futoransky, Notarfrancesco, Richarte, & Sarraute). Attackers often demonstrate a negative correlation between the use of an attack and its cost. The insight gained from this process can be extremely helpful in determining which specific attacks an information system may face (Schechter, 2005). Figure 4, introduces the costs associated with each node of the “Open Safe” attack tree (Schneier, 1999).

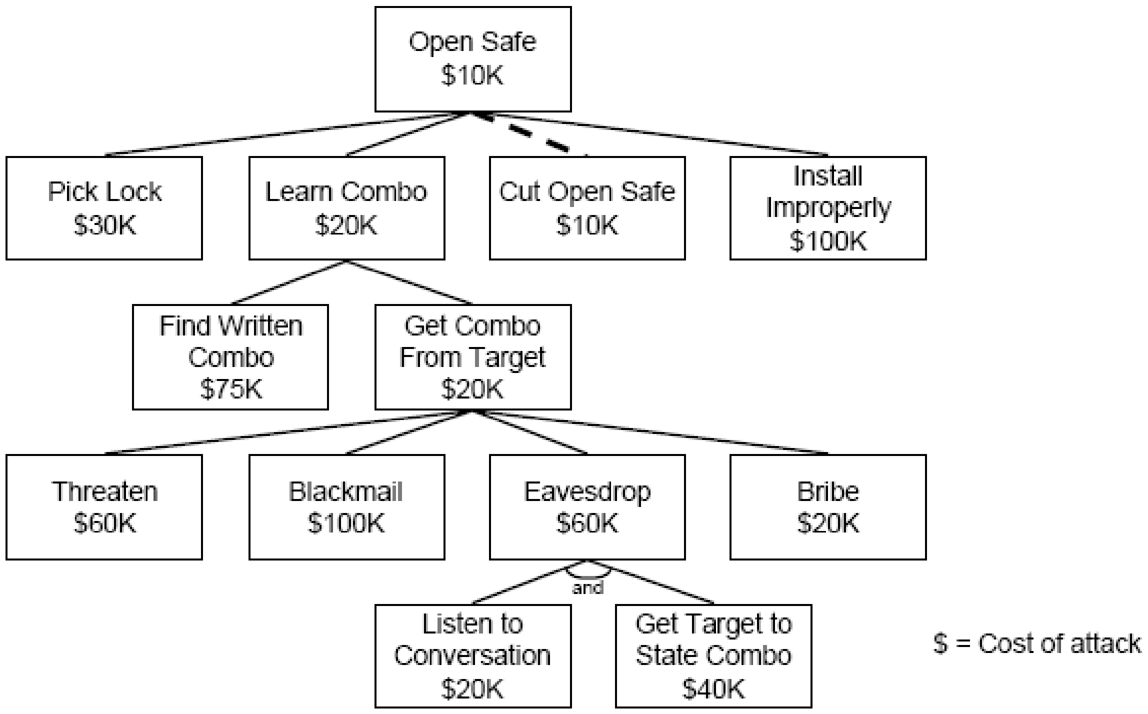


Figure 4. Attack Tree with Cost-Per-Node Included adopted from “Attack Trees” (Schneier, 1999).

As shown in Figure 4, it is possible to “Cut Open the Safe” for $10,000 while “Learning Combo” through eavesdropping would cost the attacker $60,000 (Listening to Conversation + Get Target to State Combo). This type of analysis can be helpful in determining which specific attacks you are likely to encounter.

Attack trees can also be useful for examining technical attacks and environments. Consider the various scenarios in which an attacker could gain root (administrative) access to a web server. Figure 5, introduces a partially completed attack tree for completing this attack (Tidwell et al., 2001).

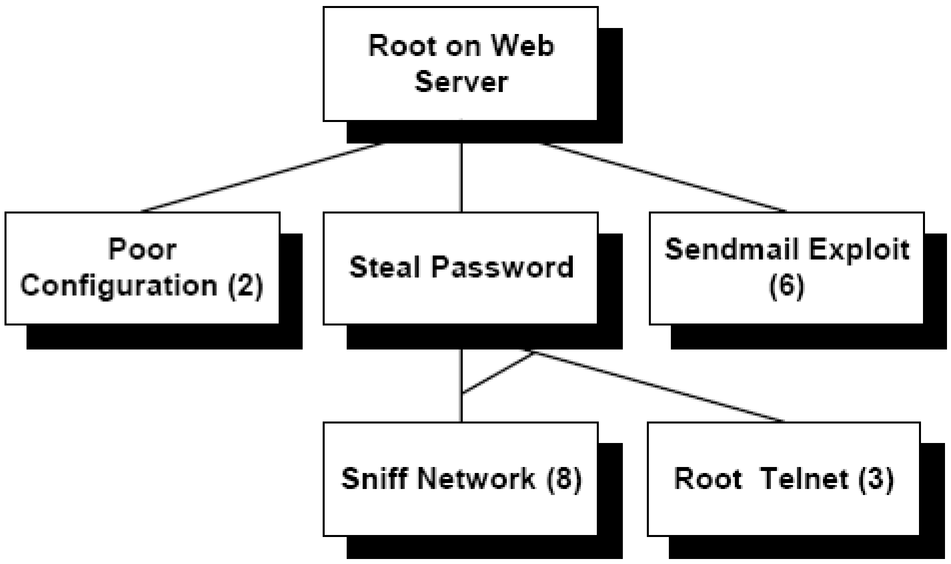


Figure 5. Partial Attack Tree for Gaining Root Access to a Web Server adopted from “Modeling Internet Attacks” (Tidwell et al., 2001).

In this example, attack tree nodes are assigned weighted values to represent the likelihood of success in achieving the root goal. Assigned values range from 1 (Least Likely) to 10 (Most Likely). The “Steal Password” leaf is made up from the children nodes “Sniff Network” and “Root Telnet”. The lowest child score is inherited by the parent to signal the path of least resistance. As a result of this process, “Steal Password” would be assigned a value of 3. Ranking the listed attacks would result in the following (From “Most Likely” to occur to “Least Likely” to occur).

- Sendmail Exploit (6)
- Steal Password (3)
- Poor Configuration (2)

Attack trees provide an effective aid for modeling threats (Mauw & Oostdijk, 2005). The ability to clearly model and understand threats is vital to today’s security professionals. Carnegie Mellon CERT shows a dramatic growth in the number of new vulnerabilities reported each year. 262 new vulnerabilities were catalogedin 1998, while 7236 new vulnerabilities were recorded in 2007 (CERT, 2007). As the number of reported vulnerabilities continues to rise, the need for additional ways to manage and visualize the complexity of such attacks grows as well. Attack trees can be an effective methodology for understanding threat-based inter-relationships and ranking threats according to risk (Byres, Franz, & Miller, 2004).

2.3. Attack Patterns

An attack is a specific action carried out to exploit a vulnerability (Fong, Gaucher, Okun, Black, & Dalci, 2008). The Common Attack Pattern Enumeration and Classification (CAPEC) framework is a model for identifying, classifying, cataloging, sharing and refining various types of information about attacks (Barnum & Amit, 2006a). The CAPEC framework provides this information through descriptive schema or elements used to specify the various components which make up an attack. Each attack pattern is a generalized outline of the attack which has been developed by reviewing large sets of exploits (McGraw, 2006). Attack patterns also detail the approach and methodology used by attackers to generate an exploit (Barnum & Sethi).

Like attack trees, attack patterns represent the objective of the attacker and the techniques which may be used by attackers to achieve their goals and provide an organized way to analyze the details of a specific attack (Barnum & Amit, 2006b; Viega & McGraw, 2002). The ability to view threats from an attackers perspective is a vital component in protecting information systems (Arce, 2004). Security research is often slowed because of the level of secrecy surrounding attacks, vulnerabilities and exploits (Barnum & Amit, 2006b; Logan & Clarkson, 2005). Attack patterns can be used to expose the details of such attacks. In the past, security experts have been hesitant to create and share the details of exploits, fearing such data could be used to further malicious attacker's knowledge (Russell, 2002).

Creating a deeper understanding of attackers, attacks, and countermeasures can lead to a more effective ability to combat and counter these threats (Schneier, 1999). Fostering this deep understanding of attack patterns can also lead to the permeation of security throughout the software development life cycle and heighten the awareness of known exploits, vulnerabilities and weaknesses (Gegick & Williams, 2005). Integrating and increasing attack pattern knowledge can result in adding security by creating less exposure to identified bugs and known flaws (Hoglund & McGraw, 2004). Attack patterns can also be used to create a security checklist, which in turn can lead to a higher level of security (S. Barnum, 2007).

The origins of attack patterns can be traced back to the 1960's when the foundation for today's attack patterns were established as the concept of a general and repeatable solution to identified system development problems (Gamma, Helm, Johnson, & Vlissides, 1995). More recently the concept of presenting from an attacker's perspective was done on an individual basis, with no agreed upon formula, structure, or common language for consistently presenting such a viewpoint (Hoglund & McGraw, 2004).

The lack of a common vocabulary makes it difficult to gather, analyze, and share pertinent information which could be used to advance the discipline of software security(Hoglund & McGraw, 2004). The term "attack pattern" was introduced in 2001 to describe the concept of combining various types of malicious attacks and present the attacker's perspective within a specified framework (Gamma et al., 1995; Moore et al., 2001). Further research was done to formally define descriptive attack pattern elements and the create 48 original and complete attack patterns (Hoglund & McGraw, 2004).

The National Cyber Security Division of the Department of Homeland Security in conjunction with Cigital and MITRE Corporation agreed to sponsor CAPEC (S. Barnum, 2007; Barnum & Amit, 2006b). The final result of this collective effort was published in March of 2007 as the CAPEC Release 1 Dictionary and included a formalized attack-driven perspective of software security with 101 different attack patterns outlined (Barnum & Amit, 2006a).

The Common Attack Pattern Enumeration and Classification (CAPEC) list provides an official schema and formal representation for defining individual attack patterns (Barnum, 2008; Barnum & Amit, 2006a). CAPEC formally organizes and presents each attack pattern by gathering and displaying both primary and supporting data elements (Sean Barnum, 2007). Primary elements include the following list (Barnum, 2008; capec.mitre.org, 2007).

- Attack Pattern ID
- Attack Pattern Name
- Description
- Related Weaknesses
- Related Vulnerabilities
- Methods of Attack
- Examples-Instances

- References
- Solutions and Mitigations
- Typical Severity
- Typical Likelihood of Exploit
- Attack Prerequisites
- Attacker Skill or Knowledge Requirements
- Resources Required
- Attack Motivation-Consequences
- Context Description

Supporting elements include the following list (Barnum, 2008; capec.mitre.org, 2007).

- Injection Vector
- Payload
- Activation Zone
- Payload Activation Impact
- Probing Techniques
- Indicators/Warnings of Attack
- Obfuscation Techniques
- Related Attack Patterns
- Relevant Security Requirements
- Relevant Design Patterns
- Relevant Security Principles
- Related Guidelines

Exploration and examination of the various techniques used by malicious attackers are important steps in providing better security for our technology resources (Skoudis & Liston, 2006). “Know thy enemy” is a classic adage amongst security researchers which suggests that security professionals need the ability to understand system vulnerability from the perspective of a potential attacker (Fadia, 2002; Jones, Shema, & Johnson, 2002; Koziol et al., 2004; McClure, Scambray, & Kurtz, 2005). The best penetration tests are built on a solid understanding of both design and risks (McGraw, 2006). This type of understanding can only be achieved when we have a formal set of definitions to build and share knowledge. CAPEC attack patterns provide such a framework.

3. ABSTRACTING PARENT MITIGATIONS

The CAPEC Release 1 Dictionary includes nearly 400 individually prescribed controls which can be used to mitigate or reduce the effects of the defined attack patterns. This current level of detail in the “Solutions and Mitigations” element tends to be too inconsistent (Engebretson & Pauli, 2008). Some attack patterns provide an extremely granular level of detail. For example, one of the prescribed mitigations for attack pattern 1 (Accessing Functionality Not Proper Constrained by ACLs) calls for changing a Java setting. Specifically the Solutions and Mitigations element prescribes, “In a J2EE setting, deployers can associate a role that is impossible for authenticator to grant users, such as ‘NoAccess’, with all Servlets to which access is guarded by a limited number of servlets visible to, and accessible by, the user”. This level of detail can lead CAPEC adopters to assume that attacks based off accessing functionality not properly constrained by ACL’s are confined only to environments where Java or J2EE are deployed. Such a belief could lead to an increased attack exposure and a false sense of security because attacks that focus on “Accessing Functionality Not Properly Constrained by ACLs” include a much broader attack vector than just the Java environments.

The reverse is also true. Some attack patterns provide only a brief overview of potential mitigation strategies. Attack pattern 5 (Analog In-Band Switching Signals (aka Blue Boxing)) includes “Upgrade phone lines” as a mitigation strategy. This generalized strategy is too open-ended to be of use to many users. This type of vagueness leaves many basic questions unanswered related to infrastructure, physical design, layout, speed, and quality issues.

In order to increase the effectiveness and consistency of mitigation strategies, we propose the inclusion of a new element to the CAPEC standard. Our Parent Mitigation element is directly abstracted from the currently prescribed CAPEC “Solutions and Mitigations” element.

We examined several standards when looking for a complete set of parent mitigation strategies to complement the CAPEC Dictionary. It is vital to make use of a predefined, currently accepted and standardized list of controls to remove the heuristic tone of an ad-hoc approach. Our approach is both detailed and specific to ensure individuals following our prescribed processes will reach the same findings.

We reviewed COBIT 4.1, ISO 27002:2005, and NIST SP 800-53 for an acceptable list of controls to use as Parent Mitigations in our approach (ISACA, 2008; ISO, 2005; NIST, 2007). After reviewing the controls outlined in each of these standards, we choose to make use of NIST 800-53 (revision 2). Both NIST and CAPEC have strong ties to the United States Federal government. NIST is a non-regulatory federal agency funded through the U.S. Department of Commerce, while CAPEC is the direct result of funding from the Department of Homeland Security (NIST, 2006). CAPEC is a federally funded classification of attacks and NIST is a federally funded list of controls.

During the selection process, we were able to reject the controls outlined in the COBIT standard, because it is less specific to Information Systems or Information Technology details than the controls outlined in ISO (Flowerday & Von Solms, 2005). Because of the technical nature of attack patterns, we focus on controls which provide the most technical details. ISO was rejected because of its emphasis on being a management system, rather than a technology specification (Calder, 2006). We are providing a technical specification for mitigations as part of our approach. We view NIST as a stronger match than the business process-oriented ISO standard.

Additionally, we chose to use NIST because the controls provide a ready-made hierarchy which fits within our Parent-Child model. This additional level of detail and structure not only correlates directly with our work, but will also be used in future work to further extend the relationship between NIST and CAPEC.

NIST 800-53 provides an established and usable control-based hierarchy. At the top level this hierarchy consists of Family controls which are general and wide-reaching. The final draft of 800-53-r2 includes a total of 17 Family controls which are presented in a well-defined and organized structure. A two character identifier is used to uniquely identify individual family controls. NIST Family level controls and their corresponding identifiers are introduced in Table 1 (NIST, 2007).

Table 1. NIST 800-53 17 Family Level Controls and Their Unique Identifier.

IDENTIFIER	**FAMILY**
AC	Access Control
AT	Awareness and Training
AU	Audit and Accountability
CA	Certification, Accreditation, and Security Assessments
CM	Configuration Management
CP	Contingency Planning
IA	Identification and Authentication
IR	Incident Response
MA	Maintenance
MP	Media Protection
PE	Physical and Environmental Protection
PL	Planning
PS	Personnel Security
RA	Risk Assessment
SA	System and Services Acquisition
SC	System and Communications Protection
SI	System and Information Integrity

Each of the 17 Family level controls is further broken down into individual controls identified by NIST. In order to identify individual NIST controls, a number is appended to the family identifier. This combination of "Family identifier – control number" is used to uniquely identify each control outlined in the NIST 800-53r2 (NIST, 2007). For example, CM-8 corresponds to the 8th control listed under the "Configuration Management" Family control. Our approach introduces the appropriate NIST control into the existing CAPEC dictionary as a "Parent Mitigation" to provide a more generalized mitigation strategy for each of the 400 CAPEC attack patterns. Our process groups all 400 mitigations into 17 standardized Parent Mitigations.

3.1. Abstracting Parent Mitigations from the CAPEC Dictionary

To illustrate our approach we completed a case study utilizing the CAPEC attack pattern dictionary. This case study consists of 11 unique attack patterns. In order to provide adequate sampling, we've chosen one attack pattern from each of the 11 Parent Threats outlined on the CAPEC classification tree. Parent Threats are as follows (Engebretson & Pauli, 2008):

- Abuse of Functionality
- Spoofing
- Probabilistic Techniques
- Exploration of Authentication
- Resource Depletion

- Exploitation of Privilege/Trust
- Injection (Injecting Control Plane content through the Data Plane)
- Data Structure Attacks
- Data Leakage Attacks
- Resource Manipulation
- Time and State Attacks

The same 11 attack patterns were used to demonstrate the three processes that make up our approach. The entire approach was carried out for all 101 attack patterns and a complete listing of these results can be found in Appendix 1. The chosen attack patterns for the case study and corresponding Parent Threat are introduced in Table 2.

Table 2. Selected Parent Threats and Corresponding Attack Patterns for Case Study.

Parent Threat	**Attack Patter Name (Attack Pattern Number)**
Abuse of Functionality	Forceful Browsing (87)
Spoofing	Man in the Middle Attack (94)
Probabilistic Techniques	Rainbow Table Password Cracking (55)
Exploration of Authentication	Reusing Session IDs (Session Replay) (60)
Resource Depletion	XML Denial of Service (XDoS) (82)
Exploitation of Privilege/Trust	Manipulating Writeable Configuration Files (75)
Injection (Injecting Control Plane content through the Data Plane)	Server Side Includes (SSI) Injection (101)
Data Structure Attacks	Buffer Overflow via Environment Variables (10)
Data Leakage Attacks	Passively Sniff and Capture Application Code Bound for Authorize Client (65)
Resource Manipulation	Using Leading 'Ghost' Character Sequences to Bypass Input Filters (3)
Time and State Attacks	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions (29)

NIST provides significant detail for each child control including unique control number, name, brief control description, and supplemental guidance. The control description provides a concise description of the control. The supplemental guidance provides additional examples and requirements (NIST, 2007). Both the control description and the supplemental guidance are useful in order to accurately match the NIST and CAPEC controls. The process matches a CAPEC Solutions and Mitigations element and one of the NIST details.

The process of abstracting Parent Mitigations from the CAPEC Attack Pattern Release 1 Dictionary is made up of 4 steps as introduced in Figure 6.

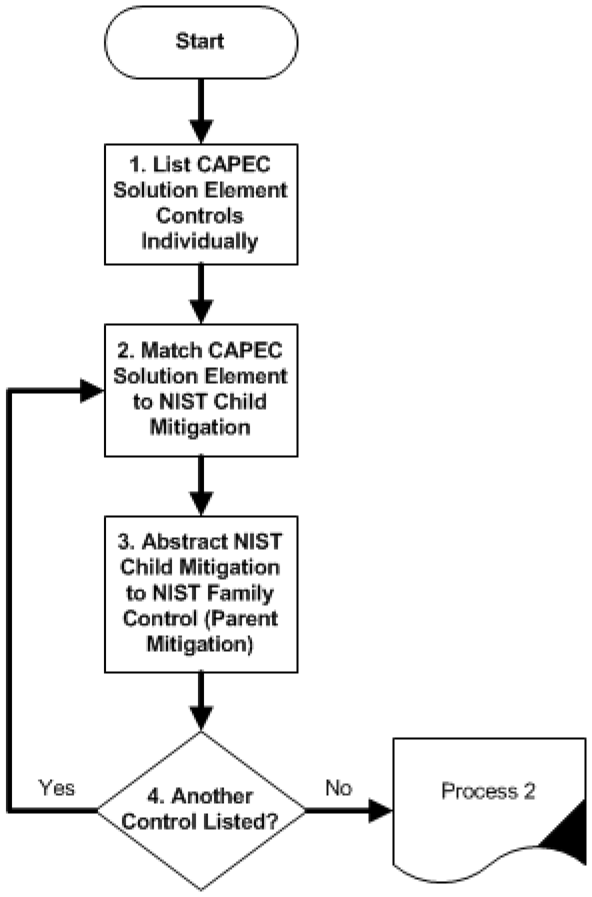

Start
1. List CAPEC Solution Element Controls Individually
2. Match CAPEC Solution Element to NIST Child Mitigation
3. Abstract NIST Child Mitigation to NIST Family Control (Parent Mitigation)
4. Another Control Listed?
Yes
No
Process 2

Figure 6. Steps Required to Abstract Parent Mitigations from the CAPEC Release 1 Dictionary.

The process of abstracting Parent Mitigations starts by breaking down the attack pattern's Solutions and Mitigation element into a list of individual controls as shown in step 1. Step 2 introduces a line item review of each mitigation strategy. Using the control definitions outlined in NIST 800-53, we match each CAPEC control to a corresponding NIST control. Although we are only interested in the NIST Family control, we map each of the current CAPEC mitigations to the detailed controls in NIST 800-53 to ensure accuracy. Step 3 allows us to determine the appropriate Family level controls for inclusion into the CAPEC standard. The abstracted NIST Family controls are then added to the CAPEC

Dictionary as a Parent Mitigation element. Step 4 checks for the repeating of this process until each of the Solutions and Mitigations listed in step 1 have been abstracted.

The detailed steps in Process 1 are listed below.

1. Create a table to create a list of individual controls taken directly from the attack pattern's Solutions and Mitigations element. Controls should be listed 1 per row under the column heading "Solutions and Mitigations".
2. Select an individual control from the table created in step 1 and match the CAPEC Solutions and Mitigations element to the appropriate 800-53r2 NIST Child Mitigation(s). It is possible that individual controls from step 1 will match up with more than one NIST Child control. For this reason, it is important to review individual CAPEC controls against all of the 800-53r2 NIST controls. When a definition match is found, record the NIST Child Mitigation abbreviation under the column heading "NIST Child Mitigation". When multiple matches for a single control are found, they should be recorded in the same cell and separated by a comma.
3. Abstract the individual NIST Child Mitigation(s) to its corresponding NIST Family Control by removing the specific control number from the recorded Child Mitigation. It is important to review the table to verify if this Parent Threat has been previously recorded. If not, record the NIST Family Control under the Parent Mitigation column heading in the table.
4. If another Solutions and Mitigations control is listed, repeat steps 2-3. Continue this process until all controls for the attack pattern have been abstracted.

Table 3 introduces the table which is required to complete this process.

Table 3. Table Used to Abstract the Parent Mitigations.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**

Our case study begins with a detailed analysis of attack pattern ID 3: “Using Leading 'Ghost' Character Sequences to Bypass Input Filters”. Attack pattern 3 belongs to the Resource Manipulation Threat Family as first introduced in Table 2. Step 1 requires that we create a table to list individual controls from the CAPEC definition for attack pattern 3. We utilize the first two columns presented in Table 3 to complete step 1. Examination of the CAPEC Dictionary provides three individual mitigations for this attack as introduced in Table 4.

Table 4. Individually Listed Controls for Attack Pattern 3.

Attack Pattern	**Solutions and Mitigations**
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.
	Cononicalize all data prior to validation.
	Take an iterative approach to input validation (defense in depth)

Step 2 of Process 1 requires that we select an individual control from step 1 and match the control to the appropriate NIST 800-53r2 Child Mitigation(s). Careful review and examination is needed to align this control with the appropriate control descriptions provided as part of the NIST 800-53r2 standard. The first individual Solution and Mitigation listed in Table 3 is “Perform white list, rather than black list, input validation”. The NIST child mitigation definitions which relate to this control are listed below:

- AC-3 ACCESS ENFORCEMENT
    - Access Enforcement (AC-3) was chosen because “white list” is a type of access control enforcement (Chow, Hui, Yiu, Chow, & Lui, 2005). Furthermore, examination of the NIST AC-3 Supplemental Guidance provides the following detail which aligns closely with the CAPEC control, “Access control policies and associated access enforcement mechanisms are employed by the organization to control access between users (or processes) and objects (e.g., devices, files, records, processes, programs, domains) in the information system. In addition to controlling access at the information system level, access enforcement mechanisms are employed at the application level, when necessary, to provide increased information security for the organization.” (NIST, 2007).
- AC-4 INFORMATION FLOW ENFORCEMENT
    - Selection of Information Flow Enforcement (AC-4) can be justified by examination of the NIST child control description, “The information system enforces assigned authorizations for controlling the flow of information within the system and between interconnected systems in accordance with applicable policy.” (NIST, 2007) as well as the supplemental guidance which provides the following information, “Flow control is based on the characteristics of the information and/or the information path. Specific

examples of flow control enforcement can be found in boundary protection devices (e.g., proxies, gateways, guards, encrypted tunnels, firewalls, and routers) that employ rule sets or establish configuration settings that restrict information system services or provide a packet filtering capability" (NIST, 2007). White list input validation is an effective means of controlling the flow of information.

- CM-7 LEAST FUNCTIONALITY
  - Least functionality (CM-7) was chosen as a result of CAPEC's use of the terms "white list rather than black list". White lists are more restrictive in nature and employ the concept of least functionality by allowing denying any services not explicitly allowed. Black lists are less restrictive by allowing any service not explicitly blocked (Emmanuel & Yu).
- SI-9 INFORMATION INPUT RESTRICTIONS
  - Information Input Restrictions (SI-9) present a natural fit with the given CAPEC control as the NIST control description provides the following definition, "The organization restricts the capability to input information to the information system" (NIST, 2007). Both the NIST and CAPEC controls are describing an input validation process.
- SI-10 INFORMATION ACCURACY, COMPLETENESS, VALIDITY, AND AUTHENTICITY
  - Information Accuracy, Completeness, Validity, and Authenticity (SI-10) provides the following information in the supplemental guidance, "Checks for accuracy, completeness, validity, and authenticity of information are

accomplished as close to the point of origin as possible. Rules for checking the valid syntax of information system inputs (e.g., character set, length, numerical range, acceptable values) are in place to verify that inputs match specified definitions for format and content. Inputs passed to interpreters are prescreened to prevent the content from being unintentionally interpreted as commands." (NIST, 2007). This description presents another clear example of input validation and is therefore included as a match for the prescribed CAPEC mitigation.

The completed second step for the first control in attack pattern 3 is introduced in Table 5.

Table 5. Matching NIST Child Mitigations for the First CAPEC Control for Attack Pattern 3.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10

Step 3 requires that we abstract the individual Child Mitigations chosen in step 2. Step 3 also necessitates that each Parent Mitigation be recorded only one time. Parent Mitigations are abstracted by recording a single entry for each unique NIST Family mitigation shown under the NIST Child Mitigations column. Table 6 introduces the completed table for the first CAPEC mitigation including the abstracted Parent Mitigation column.

Table 6. Addition of Parent Mitigations for the First CAPEC Control for Attack Pattern 3.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10	AC, SI, CM

Step 4 of Process 1 mandates that we repeat step 2 if another CAPEC control exists. The second control outlined for attack pattern 3 by CAPEC is “Canonicalize all data prior to validation”. Using the NIST 800-53 guidelines, we correlate this with the following NIST controls:

- SI-9 INFORMATION INPUT RESTRICTIONS
    - Canonicalization is the process by which a potentially flexible data type can be altered into one that has guaranteed characteristics. Canonicalization is a frequent technique for input data validation and therefore relates well to the NIST standard SI-9 (Fithen, 2005). An example of canonicalization is seen when the same input data can be encoded in many ways, such as ASCII or Unicode. This transformation of data into a known and expected type is a useful form or input validation.
- SI-10 INFORMATION ACCURACY, COMPLETENESS, VALIDITY, AND AUTHENTICITY
    - Canonicalization is a frequent technique for input data validation and therefore relates well to the NIST standard SI-10 (Fithen, 2005). An example of canonicalization is seen when the same input data can be

encoded in many ways, such as ASCII or Unicode. This transformation of data into a known and expected type is a useful form or input validation.

The first two controls for attack pattern 3 and the corresponding NIST Child Mitigations are introduced in Table 7.

Table 7. Matching NIST Child Mitigations for the First and Second CAPEC Control for Attack Pattern 3.

Attack Pattern	Solutions and Mitigations	NIST Child Mitigation(s)
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10
	Canonicalize all data prior to validation	SI-9, SI-10

Repeating step 3 requires that we abstract the individual Child Mitigations chosen in step 2. Table 8 introduces the completed table for the second CAPEC mitigation including the addition of the abstracted Parent Mitigation column and values. Because System and Information Integrity (SI) has already been listed in first row, it is not necessary to repeat the Parent Mitigation.

Table 8. Addition of Parent Mitigations for the First CAPEC Control for Attack Pattern 3.

Attack Pattern	Solutions and Mitigations	NIST Child Mitigation(s)	Parent Mitigation(s)
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10	AC, SI, CM
	Canonicalize all data prior to validation	SI-9, SI-10	

The final control for attack pattern 3 is listed as: “Take an iterative approach to input validation (defense in depth)”. We correlate this CAPEC mitigation with the following NIST control.

- SI-10 INFORMATION ACCURACY, COMPLETENESS, VALIDITY, AND AUTHENTICITY
    - Justification for the selection of Information Accuracy, Completeness, Validity and Authenticity (SI-10) can be found in both the control description, “The information system checks information for accuracy, completeness, validity, and authenticity.” as well as the supplemental guidance “Rules for checking the valid syntax of information system inputs (e.g., character set, length, numerical range, acceptable values) are in place to verify that inputs match specified definitions for format and content. Inputs passed to interpreters are prescreened to prevent the content from being unintentionally interpreted as commands. The extent to which the information system is able to check the accuracy, completeness, validity, and authenticity of information is guided by organizational policy and operational requirements.” (NIST, 2007) Both of these definitions pertain directly with input validation.

The three original CAPEC controls and the justified NIST Child Mitigations are introduced in Table 9.

Table 9. Matching NIST Child Mitigations for All CAPEC Controls Assigned for Attack Pattern 3.

Attack Pattern	Solutions and Mitigations	NIST Child Mitigation(s)
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10
	Canonicalize all data prior to validation	SI-9, SI-10
	Take an iterative approach to input validation (defense in depth)	SI-10

Because System and Information Integrity (SI) has already been listed in the Parent Mitigation column, we do not relist this information again.

Table 10. Process 1 Results for Attack Pattern 3.

Attack Pattern	Solutions and Mitigations	NIST Child Mitigation(s)	Parent Mitigation(s)
Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10	AC, SI, CM
	Canonicalize all data prior to validation	SI-9, SI-10	
	Take an iterative approach to input validation (defense in depth)	SI-10	

The original CAPEC Solutions and Mitigations element provides three controls for attack pattern 3. Our process of abstraction results in the same number of controls needed to mitigate the risk. We are not concerned with reducing the number of controls for each attack pattern. Our approach reduces the total mitigations from nearly 400 (from CAPEC) to no more than 17 (from the NIST “Family”). Adding the “Parent” mitigation into the CAPEC dictionary brings a level of consistency. The CAPEC Dictionary’s mitigation strategies are now standardized into 17 "Parents" at the same level of abstraction. Users are less likely to dismiss a particular attack pattern because the mitigation is too detailed or too specific. This is currently a risk for CAPEC adopters who believe that they are not at risk for a given attack because they do not have the specific technology mentioned in the CAPEC mitigation.

This same process was followed for attack pattern 75: "Manipulating Writable Configuration Files". Attack pattern 75 belongs to the Exploitation of Privilege/Trust family. The CAPEC Dictionary provides five individual mitigations for this attack as introduced in Table 11.

Table 11. Individually Listed Controls for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege
	Design: Backup copies of all configuration files
	Implementation: Integrity monitoring for configuration files
	Implementation: Enforce audit logging on code and configuration promotion procedures.
	Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD

The first Solution and Mitigation listed in Table 11 is "Design: Enforce principle of least privilege". NIST provides a clear match with this control.

- AC-6 LEAST PRIVILEGE
    - Least Privilege was chosen as a result of a direct match between the CAPEC and NIST controls. The AC-6 NIST control definition provides the following definition: "The information system enforces the most restrictive

set of rights/privileges or accesses needed by users (or processes acting on behalf of users) for the performance of specified tasks.” (NIST, 2007).

The completed process for the first control in attack pattern 75 is introduced in Table 12.

Table 12. Matching NIST Child Mitigations for the First CAPEC Control for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6

Next we abstract the Parent Mitigations from the individual Child Mitigations. Table 13 introduces the completed table for the first CAPEC mitigation, including the abstracted Parent Mitigation column.

Table 13. Addition of Parent Mitigation Column for the First CAPEC Control for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6	AC

The second control outlined for attack pattern 75 by CAPEC is “Design: Backup copies of all configuration files”. Using the NIST 800-53 guidelines, we correlate this with the following NIST controls:

- CP-9 INFORMATION SYSTEM BACKUP
    - Information System Backup (CP-9) was chosen as a counterpart for the CAPEC solution because of the direct match between the CAPEC and NIST controls. Both mitigations are directly concerned with backups. Specifically, NIST provides the following information as part of the control description, “The organization conducts backups of user-level and system-level information (including system state information) contained in the information system” (NIST, 2007). Because configuration files are an important component of system backups, the CAPEC and NIST controls present a natural fit.
- CP-10 INFORMATION SYSTEM RECOVERY AND RECONSTITUTION
    - Information system recovery and reconstitution (CP-10) is included as a result of both the NIST control definition as well as the supplemental guidance. The NIST control provides the following description,” The organization employs mechanisms with supporting procedures to allow the information system to be recovered and reconstituted to a known secure state after a disruption or failure.” (NIST, 2007). Backup of the configuration files is a crucial component of a “mechanism to allow the information system to be recovered and reconstituted to a known secure state after a disruption or failure.” Without a backup of the current configuration file, a complete system restore would result in the loading of a default configuration file.

- CM-2 BASELINE CONFIGURATION
  - Baseline Configuration (CM-2) was selected as a result of the NIST control definition, “The organization develops, documents, and maintains a current baseline configuration of the information system.” (NIST, 2007). One component of maintaining a baseline configuration is through the backup of the configuration.

The first two controls for attack pattern 75, and corresponding NIST Child Mitigations are introduced in Table 14.

Table 14. Matching NIST Child Mitigations for the First and Second CAPEC Control for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2

Table 15 introduces the completed table for the second CAPEC mitigation including the abstracted Parent Mitigation column and values. Even though “Contingency Planning” has two entries (CP-9, CP-10) in the NIST Child Mitigation(s) column, the Parent Mitigation is listed only once.

Table 15. Addition of Parent Mitigations for the First CAPEC Control for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6	AC
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM

The next CAPEC mitigation for attack pattern 75 is "Implementation: Integrity Monitoring for Configuration Files". We correlate this CAPEC mitigation with the following NIST controls.

- AU-6 AUDIT MONITORING, ANALYSIS, AND REPORTING
  - Audit Monitoring, Analysis, and Reporting (AU-6) was chosen because of its close natural match to the CAPEC solution "monitoring for configuration files". Both mitigations pertain directly to monitoring. Specifically NIST uses the following definition, "The organization regularly reviews/analyzes information system audit records for indications of inappropriate or unusual activity, investigates suspicious activity or suspected violations, reports findings to appropriate officials, and takes necessary actions." (NIST, 2007).
- CA-7 CONTINUOUS MONITORING
  - Justification for Continuous Monitoring (CA-7) again stems from the key mitigation strategy of monitoring. Moreover, the NIST control definition provides the following information, "The organization monitors the security controls in the information system on an ongoing basis." and the

supplemental guidance for CA-7 offers this insight, "This control is closely related to and mutually supportive of the activities required in monitoring configuration changes to the information system." (NIST, 2007).

- CM-4 MONITORING CONFIGURATION CHANGES
    - Monitoring Configuration Changes (CM-4) is included as a direct match between CAPEC's solution and NIST's control definition, "The organization monitors changes to the information system" (NIST, 2007).
- CM-6 CONFIGURATION SETTINGS
    - Configuration Settings (CM-6) is added to the list of applicable child controls because of the NIST supplemental guidance which states "Organizations monitor and control changes to the configuration settings in accordance with organizational policies and procedures" (NIST, 2007).
- SI-4 INFORMATION SYSTEM MONITORING TOOLS AND TECHNIQUES
    - Information systems monitoring tools and techniques (SI-4) is justified through examination of the following control definition "The organization employs tools and techniques to monitor events on the information system" (NIST, 2007). Both the CAPEC and NIST controls make use of information system monitoring for protection purposes and are therefore directly connected.
- SI-7 SOFTWARE AND INFORMATION INTEGRITY
    - The final NIST control selected as a match for the CAPEC solution and mitigation element is Software and Information Integrity (SI-7). This control was selected based off the NIST supplemental guidance, which directly

addresses monitoring the integrity of the information system. Specifically NIST provides the following information, "The organization employs integrity verification applications on the information system to look for evidence of information tampering, errors, and omissions." (NIST, 2007).

The first three original CAPEC controls and the justified NIST Child Mitigations are introduced in Table 16.

Table 16. Matching NIST Child Mitigations for Three CAPEC Control Assigned for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7

Table 17 introduces the complete abstracted table including the Parent Mitigation column for CAPEC attack pattern 75.

Table 17. Abstracted Parent Mitigation Table for Three CAPEC Controls for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6	AC
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7	AU, CA, SI

The fourth control listed in the CAPEC Solutions and Mitigations element is "Implementation: Enforce audit logging on code and configuration promotion procedures". We correlate this CAPEC mitigation with the following NIST controls:

- AU-1 AUDIT AND ACCOUNTABILITY POLICY AND PROCEDURES
    - Audit and Accountability Policy and Procedures (AU-1) was selected as a result of the NIST control definition, "The organization develops, disseminates, and periodically reviews/updates: (i) a formal, documented, audit and accountability policy that addresses purpose, scope, roles, responsibilities, management commitment, coordination among organizational entities, and compliance; and (ii) formal, documented procedures to facilitate the implementation of the audit and accountability policy and associated audit and accountability controls." (NIST, 2007). Both of these controls deal directly with auditing and their subsequent procedures.
- AU-2 AUDITABLE EVENTS
    - Auditable Events (AU-2) was selected as a match because of the NIST supplemental guidance which states the following, "The purpose of this control is to identify important events which need to be audited as significant and relevant to the security of the information system. The organization specifies which information system components carry out auditing activities." (NIST, 2007). Both the CAPEC and NIST controls are focused on the auditing process.

- CM-3 CONFIGURATION CHANGE CONTROL
  - Configuration Change Control (CM-3) can be justified by examination of both the control definition which states, “The purpose of this control is to identify important events which need to be audited as significant and relevant to the security of the information system. The organization specifies which information system components carry out auditing activities.” (NIST, 2007). Additionally, the supplemental guidance offers the following information, “Configuration change control includes changes to the configuration settings for information technology products (e.g., operating systems, firewalls, routers).” and “The organization audits activities associated with configuration changes to the information system.” (NIST, 2007). Each of these statements lines up with the currently selected CAPEC Solution and Mitigation.
- CM-4 MONITORING CONFIGURATION CHANGES
  - Monitoring Configuration Changes was (CM-4) was selected because of the control definition which states, “The organization monitors changes to the information system” (NIST, 2007). This correlates well with the CAPEC mitigation of logging configuration changes.
- CM-5 ACCESS RESTRICTIONS FOR CHANGE
  - Access Restrictions for Change (CM-5) was chosen as a result of the following information provided by NIST for CM-5, “The organization: (i) approves individual access privileges and enforces physical and logical access restrictions associated with changes to the information system; and

(ii) generates, retains, and reviews records reflecting all such changes." (NIST, 2007). This definition clearly aligns itself to the monitoring and logging of configuration changes.

- CM-6 CONFIGURATION SETTINGS
    - NIST's Configuration Settings (CM-6) control was selected because of the supplemental guidance which provides the following definition, "Organizations monitor and control changes to the configuration settings in accordance with organizational policies and procedures." (NIST, 2007). Again, clear parallels between the two controls are easily identified. Both controls pertain directly with logging and monitoring of configuration settings.

The first three original CAPEC controls and the justified NIST Child Mitigations are introduced in Table 18.

Table 18. Matching NIST Child Mitigations for Four CAPEC Controls Assigned for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7
	Implementation: Enforce audit logging on code and configuration promotion procedures.	AU1, AU2, CM3, CM4, CM5, CM6

Audit and Accountability (AU) and Configuration Management (CM) have already been listed under the Parent Mitigation column, so there is no need to fill in this information again. Table 19 introduces the complete abstracted table including the Parent Mitigation column for CAPEC attack pattern 75.

Table 19. Abstracted Parent Mitigation Table for Four Attack Pattern 75 Solutions.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6	AC
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7	AU, CA, SI
	Implementation: Enforce audit logging on code and configuration promotion procedures.	AU1, AU2, CM3, CM4, CM5, CM6	

The final Solution and Mitigation listed in Table 10 is “Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD”. NIST Child Mitigation definitions which relate to this control are as follows:

- AC-5 SEPARATION OF DUTIES
    - Separation of Duties (AC-5) was chosen as a match for the CAPEC solution because of the link between the CAPEC and NIST controls. Specifically

NIST provides the following detail in the control definition, “The information system enforces separation of duties through assigned access authorizations.” (NIST, 2007). Loading the configuration from a separate space is an example of separation of process duties.

- SC-2 APPLICATION PARTITIONING
    - Justification for the selection of Application Partitioning (SC-2) stems from the correlation between the CAPEC control and the NIST supplemental guidance, “The information system separates user functionality (including user interface services) from information system management functionality.” (NIST, 2007). Loading the configuration is a clear example of application management functionality.
- SC-3 SECURITY FUNCTION ISOLATION
    - Security and Function Isolation (SC-3) is included as a result of the NIST supplemental guidance which states, “The information system isolates security functions from nonsecurity functions by means of partitions, domains, etc., including control of access to and integrity of, the hardware, software, and firmware that perform those security functions. The information system maintains a separate execution domain (e.g., address space) for each executing process” (NIST, 2007). Loading the configuration file from a separate space is clearly aligned with this NIST control.

All five of the original CAPEC controls and the justified NIST Child Mitigations are introduced in Table 20.

Table 20. Matching NIST Child Mitigations for all CAPEC Controls Assigned for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7
	Implementation: Enforce audit logging on code and configuration promotion procedures.	AU-1, AU-2, CM-3, CM-4, CM-5, CM-6
	Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD	AC-5, SC-2, SC-3

Because Access Control (AC) has already been listed under the Parent Mitigation column, we are only required to list Systems and Communication Protection (SC) as a new entry. Table 21 introduces the complete abstracted table including the Parent Mitigation column for CAPEC attack pattern 75.

Table 21. Complete Abstracted Parent Mitigation Table for Attack Pattern 75.

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Manipulating Writable Configuration Files	Design: Enforce principle of least privilege	AC-6	AC
	Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM
	Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7	AU, CA, SI
	Implementation: Enforce audit logging on code and configuration promotion procedures.	AU1, AU2, CM3, CM4, CM5, CM6	
	Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD	AC-5, SC-2, SC-3	SC

3.2. Results of Case Study

In addition to the two attack patterns shown in section 3.1, our case study followed each of the required four steps in Process 1 for the nine remaining attack patterns identified in Table 2. Table 22 introduces the complete results for attack pattern 87 (“Forceful Browsing”).

Table 22. Process 1 Results for Attack Pattern 87 ("Forceful Browsing").

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Forceful Browsing: **87**	Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.	AC17, IA2, IA3, MA4, SC8, SC23, SI10	AC, IA, MA, SC, SI
	Forceful browsing can also be made difficult to a large extent by not hard coding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.	SC18, AT3, CA2, CA4, PL2, SA3, SA8, SA10	AT, CA, PL,SA

Table 23 introduces the results of Process 1 for attack pattern 94 ("Man in the Middle").

Table 23. Process 1 Results for Attack Pattern 94 (“Man in the Middle”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Man in the Middle: **94**	Get your Public Key signed by a Certificate Authority	CA4, IA5, IA7, SC13, SC17	CA, IA, SC
	Encrypt your communication using cryptography (SSL,...)	AC3, AC4, SC7, AC17, IA7, SC8, SC9, SC12, SC13, SI7	AC ,SI
	Use Strong mutual authentication to always fully authenticate both ends of any communications channel.	AC17, IA1, IA2, IA3, IA4, IA5, SC8, SC11, SC23, SI10	
	Exchange public keys using a secure channel	SC17, SC12, SC13	

Table 24 introduces the results of Process 1 for attack pattern 55 (“Rainbow Table Password Cracking”).

Table 24. Process 1 Results for Attack Pattern 55 (“Rainbow Table Password Cracking”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Rainbow Table Pswd Cracking: **55**	Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.	SI7, SC13, IA5	SI, SC, IA

Table 25 introduces the results of Process 1 for attack pattern 60 (“Reusing Session ID’s”).

Table 25. Process 1 Results for Attack Pattern 60 ("Reusing Session ID's").

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Reusing Session ID's: **60**	Always invalidate a session ID after the user logout.	AC3, IA5, SC10, SC23, IA4	AC, IA, SC
	Setup a session time out for the session IDs.	AC11, AC12, SC23, IA4	
	Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.	AC4, IA2, IA3, IA7, SC8, SC9, SC11, SC12, SC13, SC16, SC17, SC20, SC21, SC22, SC23	SA
	Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.	SC9, SC4, SC14, SC16, SA8	
	Encrypt the session data associated with the session ID.	AC3, SC4, SC7, SC23	
	Use multifactor authentication.	IA2	

Table 26 introduces the results of Process 1 for attack pattern 82 ("XML Denial of Service").

Table 26. Process 1 Results for Attack Pattern 82 (“XMLDoS”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
XMLDoS (XDoS): **82**	Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.	AC4, SI9, SI10, AC3, CM6	AC, SI, CM
	Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers	AC4, CA3, SC6, SI4, CP10, SC5, SC22	CA,SC, CP
	Implementation: Check size of XML message before parsing	SI7, SI9, SI10	

Table 27 introduces the results of Process 1 for attack pattern 65 (“Passive Sniffing”).

Table 27. Process 1 Results for Attack Pattern 65 (“Passive Sniffing”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Passive Sniffing: **65**	Do not store secrets in client code	CM6, PE19, RA3, SA8, PL4	CM, PE, RA, SA, PL
	Use Well-Known Cryptography Appropriately and Correctly	AC3, AC17, IA7, MA4, SC8, SC9, SC12, SC13	AC, IA, MA, SC
	Use Authentication Mechanisms, Where Appropriate, Correctly	IA2, IA7, SC23, SI10	SI

Table 28 introduces the results of Process 1 for attack pattern 101 ("Server Side Includes").

Table 28. Process 1 Results for Attack Pattern 101 ("Server Side Includes").

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Server Side Includes (SSI): **101**	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them	CM1, CM6, CM7, SI6, SC3, AC6	CM, SI, SC, AC
	All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive	SI7, SI9, SI10	
	Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead	AC6	

Table 29 introduces the results of Process 1 for attack pattern 10 ("Buffer Overflow via Environment Variables").

Table 29. Process 1 Results for Attack Pattern 10 (“Buffer Overflow via Environment Variables”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Buffer Overflow via Environment Variables: **10**	Do not expose environment variable to the user.	AC6, CM6, RA3, RA5, SA10, SA11, SC4, SI10	AC, CM, RA, SA, SC, SI
	Do not use untrusted data in your environment variables.	AC3, CM6, IA2, SC23, SI17, SI19, SI10	IA,
	Use a language or compiler that performs automatic bounds checking	SA8, PL2	PL
	There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.	MA3, PL6 RA5, SA10, SA11, SI2, SI4	MA

Table 30 introduces the results of Process 1 for attack pattern 29 (“Race Conditions, Time of Check and Time of Use”).

Table 30. Process 1Results for Attack Pattern 29 (“Race Conditions, Time of Check and Time of Use”).

Attack Pattern	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Race Conditions (TOCTOU): **29**	Use safe libraries to access resources such as files.	SI7, SC18,	SI, SC
	Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.	AT2, AC3, IA2	AT, AC, IA
	Use synchronization to control the flow of execution.	SC3, AC4	
	Use static analysis tools to find race conditions.	SA11,SI10	
	Pay attention to concurrency problems related to the access of resources.	SA8, SC4	SA

3.3. Discussion and Validation

While CAPEC’s Release 1 Dictionary provides a solid framework, the current format and presentation of information provided in the Solutions and Mitigations element is inconsistent. There are tremendous variations in the depth and breadth of the “Mitigations and Solutions” currently outlined for each attack pattern. Some attack patterns provide detail that is too granular while others provide information that is vague. This chapter introduced the process to add a new Parent Mitigation element to provide consistency and mitigation strategies to be used by CAPEC adopters.

Our approach injects a Parent Mitigation element into the dictionary to provide consistency to the CAPEC Release 1 Dictionary. Because the current Mitigation and Solutions element provides valuable information, we are not advocating its removal. Our intention is to add a Parent Mitigation element to provide a manageable and consistent number of more abstracted mitigations.

There is significant value in completing this abstraction process. Adding the Parent Mitigation into the CAPEC dictionary provides a needed level of consistency and standardization. The CAPEC Dictionary's mitigation strategies are now standardized into 17 "Parents" (down from the nearly 400) each at the same level of abstraction. By abstracting these mitigations into 17 categories, users are less likely to dismiss a particular attack pattern because the mitigation is too detailed or too vague. This is currently a risk for CAPEC adopters who believe that they are not at risk for a given attack because they do not have the specific technology mentioned in the CAPEC mitigation.

Validation for Process 1 can be found by connecting our work to a strong theoretical basis. Overcoming usability issues associated with the organization and presentation of large amounts of information is a difficult task (English, Hearst, Sinha, Swearingen, & Yee, 2002). Faceted classification analysis can be used to create common categories from large amounts of data. Research has shown that these categories can be used to organize, manage, and aid in the meaningful classification of large data collections (Hearst, 2006). Similar to the use of faceted classification theory, the inclusion of the Parent Mitigation element allows for the categorization and classification of the CAPEC standard via common mitigation strategies. The Parent Mitigation element can be viewed as a facet by which the CAPEC standard can be examined and organized.

Faceted theories make use of classification systems which are organized according to specific disciplines. In this regard each facet is unique to the discipline that will utilize the classification (Hong). The inclusion of a Parent Mitigation element can be viewed as a facet which is specific to, and accepted by, the information assurance discipline. The NIST 800-53r2 makes use of a similar classification structure through the use of a Family level mitigation. Prior classification based on the Solutions and Mitigations element was not possible. Individual mitigations were unique to their corresponding attack pattern. The inclusion of a Parent Mitigation element allows for creation of classification system based off of 17 common mitigations.

The use of hierarchical faceted theory allows users to more intuitively access subcategories and underlying data (Hearst, 2006). The inclusion of the Parent Mitigation element provides similar results. CAPEC users can now access a broad category of common mitigation strategies. These strategies can be further drilled down to find the detailed and specific mitigations.

The use of faceted analysis allows for multiple perspectives of the same unit (Kwasnik, 1999). The inclusion of the Parent Mitigation element allows CAPEC users to view attack information by attack pattern name as well as by common attack pattern mitigation strategies.

The results from Process 1 are used in the completion of Process 2 and Process 3. The creation of a Parent Mitigation element is used by Process 2 to provide attack pattern context and serve as the root view in our new model for viewing attack pattern information. Parent Mitigations also provide the means for building security metrics which are presented in Process 3.

4. MODELING HIERARCHY-BASED ATTACK PATTERNS

Process 2 presents a new attack pattern model which focuses on the re-inclusion of the Parent Threat element and the inclusion of the Parent Mitigation element to logically group each of the 101 attack patterns. This model creates a graphical hierarchy for each of the attack patterns and groups them not only by Parent Threats (such as “Spoofing” and “Injection”), but also by Parent Mitigations (such as “Access Control” and “Configuration Management”). We also provide individual textual attack descriptions for each of the 101 attack patterns to provide a stand-alone, perspective of each attack pattern. Process 2 allows individual attack patterns to be traced upward to its Parent Threat and downward to its Parent Mitigation in a hierarchical tree. The traceability from the top of the tree (Parent Threat), through the selected elements of the attack patterns, to the roots of the tree (Parent Mitigation) eases the introduction of the CAPEC standard to audiences who are not familiar with attack patterns. This grouping also allows experienced users to leverage the attack information from a standardized set of elements. There is a great amount of information in the CAPEC dictionary that we are capturing and documenting with this fan-in/fan-out approach.

Process 2 includes four steps as introduced in Figure 7.

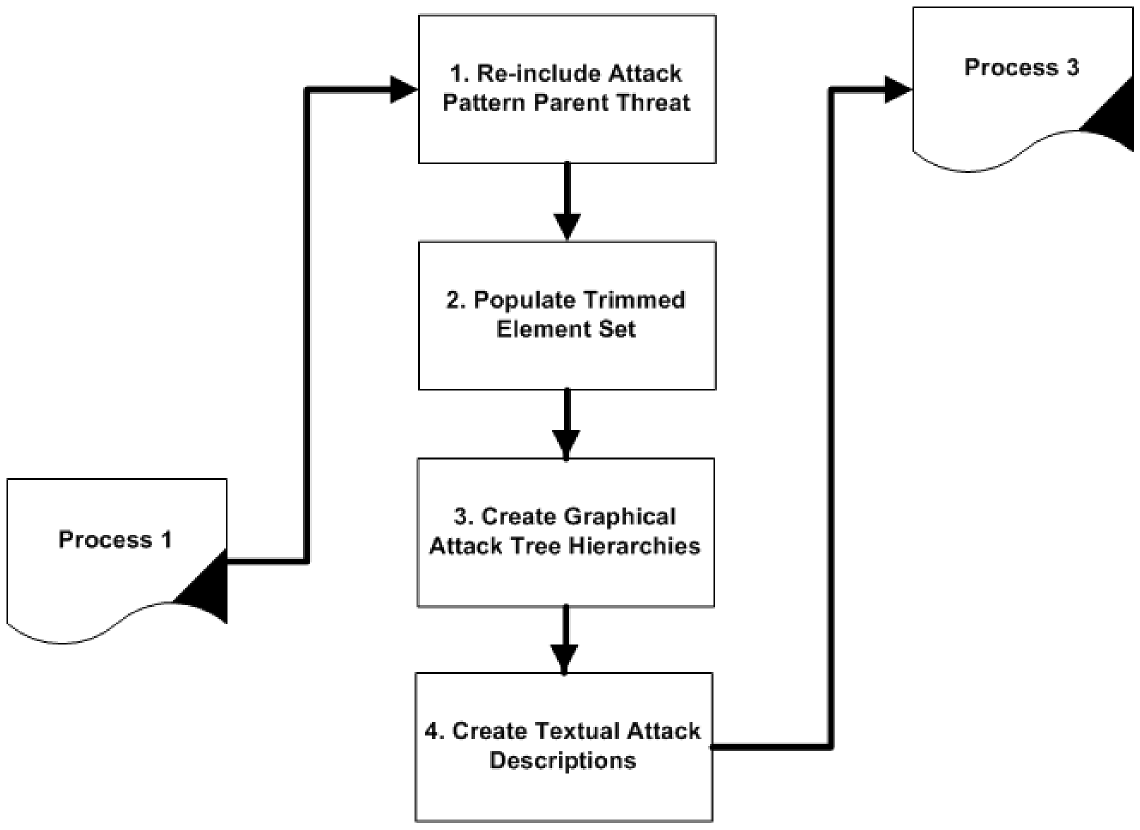

1. Re-include Attack Pattern Parent Threat
2. Populate Trimmed Element Set
3. Create Graphical Attack Tree Hierarchies
4. Create Textual Attack Descriptions
Process 1
Process 3

Figure 7. The Required Steps to Complete Process 2.

Step 1 includes Parent Threat information as a required element for each attack pattern in the CAPEC Release 1 dictionary. Step 2 reduces the number of the descriptive elements used to document each attack pattern. Step 2 is completed by populating each of the selected elements and ensuring that each element has at least one entry. The purpose of reducing the element set is to create a user-friendly model for viewing the most critical information about each of the 101 attack patterns without overwhelming the user. Justification for the selected elements is presented later in this chapter. Step 3 creates a graphical hierarchy tree which is used to model each attack pattern. The elements selected in step 2 will be used in our new model. Step 4 creates a textual attack description which is based on the trimmed element set.

Process 2 begins with the re-inclusion of a Parent Threat element into the CAPEC Dictionary to increase the usability of the standard. Currently each attack pattern can be traced to one of 11 Parent Threats via a Classification Tree which is available on the CAPEC website (capec.mitre.org, 2007). The Parent Threat information is not officially included in the CAPEC Dictionary as one of the formally defined attack pattern descriptive elements. As a result, finding the Parent Threat and related CAPEC attack patterns is a time-consuming and error prone task. This disjointed structure leads to confusion and frustration when attempting to make use of the current CAPEC Dictionary because this vital element is not included (Engebretson & Pauli, 2008). By including a Parent Threat element into the tree, we provide contextual information for the attack patterns and each related attack pattern.

The number of elements used to describe each attack pattern can present a significant problem when attempting to make use of the current CAPEC dictionary in an applied setting (Pauli & Engebretson, 2008a, 2008b). Step 2 trims the element set to provide only meaningful information of the attack pattern without overwhelming the user. The full dictionary with all descriptive elements will continue to be available for review.

Step 3 and 4 utilize the trimmed element set created in Step 2 to build hierarchies for presenting attack pattern information and viewing relationships among attack patterns. These hierarchies are derived directly from the 11 Parent Threats and are tied together by 17 Parent Mitigations introduced in chapter 3.

4.1. Re-including Parent Threats

Step 1 of Process 2 is completed by executing the following steps:

1. Open the completed table for the given attack pattern which was created in Process 1. Insert a Parent Threat column to the left of the Attack Pattern column. The new Parent Threat column will become the first column in the table.
2. Navigate to the CAPEC Classification Tree and select the "Expand All" link.
3. Locate the required attack pattern in the Classification Tree, which is listed under the attack pattern column of the chosen table.
4. Trace up the expanded classification tree to find the top level Threat Family.
5. Record this top level Family Threat in the Parent Threat column.

A template for this new table is introduced in Table 31.

Table 31. Sample Table Used to Abstract the Parent Threat.

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**

CAPEC provides the following top level threats in the Classification Tree:

- Abuse of Functionality
- Spoofing
- Probabilistic Techniques
- Exploration of Authentication
- Resource Depletion

- Exploitation of Privilege/Trust
- Injection
- Data Structure Attacks
- Data Leakage Attacks
- Resource Manipulation
- Time/State Attacks

The results of our case study for step 1 are presented below where Table 32 introduces the results of step 1 for Process 2 for attack pattern 3.

Table 32. Addition of Parent Threat Column for Attack Pattern 3 (“Using ‘Ghost’ Characters to Bypass Input Filters”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Resource Manipulation	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10	AC, CM, SI
		Canonicalize all data prior to validation	SI-9, SI-10	
		Take an iterative approach to input validation (defense in depth)	SI-10	

Table 33 introduces the results of Step 1 for Process 2 for attack pattern 75.

Table 33. Addition of Parent Threat Column for Attack Pattern 75 ("Manipulating Writable Configuration Files").

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Exploitation of Privilege / Trust	Manipulating Writable Configuration Files **75**	Design: Enforce principle of least privilege	AC-6	AC
		Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM
		Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7	AU, CA, SI
		Implementation: Enforce audit logging on code and configuration promotion procedures.	AU1, AU2, CM3, CM4, CM5, CM6	
		Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD	AC-5, SC-2, SC-3	SC

We use attack pattern 101 "Server Side Includes" as an example to introduce one benefit of this Process. CAPEC provides the following elements to describe Attack Pattern 101:

- Attack Pattern ID, Typical Severity, Description, Attack Pattern Prerequisites, Typical Likelihood of Exploit, Methods of Attack, Examples-Instances, Attacker

Skill or Knowledge Required, Resources Required, Probing Techniques, Solutions and Mitigations, Attack Motivation Consequences, Context Description, Injection Vector, Payload, Activation Zone, Payload Activation Impact, Related Weaknesses, Related Security Principles, Related Guidelines, Purpose, CIA Impact, Technical Context, and Source.

Parent Threat is not listed among the elements currently used to describe the attack. In order to determine the general threat classification that attack pattern 101 is derived from, a CAPEC user is currently required to search the expanded CAPEC Classification Tree for the given attack pattern by Attack Pattern ID or Attack Pattern Title. The Classification Tree document is separate from the CAPEC Release 1 dictionary forcing the user to move away from the descriptive elements provided within the dictionary. Within the Classification Tree, the user must wade through three levels of detail to uncover "Server Side Includes" (attack pattern 101) as a member of the "Injection" Threat Family. Step 1 in Process 2 will eliminate this manual process.

Adding the Parent Threat element to the formal definition set for all 101 of the attack patterns provides a context for viewing attack pattern information without having to manually interact with both the Full Dictionary and the Classification Tree. Adding Parent Threat as a formal element to our hierarchy increases usability by removing this manual search.

Step 1 allows users to quickly and accurately locate related threats. Figure 8 introduces a side-by-side comparison of the current and proposed steps which are required to locate threat-related CAPEC attack patterns.

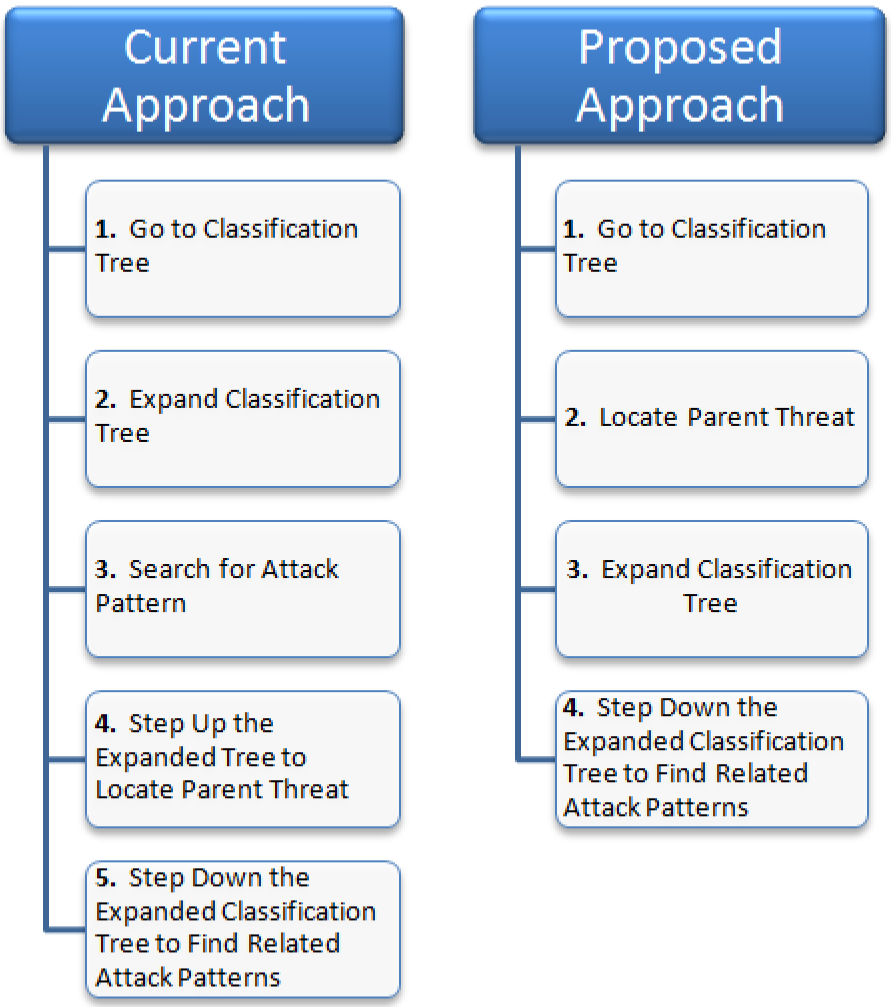

Current Approach
1. Go to Classification Tree
2. Expand Classification Tree
3. Search for Attack Pattern
4. Step Up the Expanded Tree to Locate Parent Threat
5. Step Down the Expanded Classification Tree to Find Related Attack Patterns
Proposed Approach
1. Go to Classification Tree
2. Locate Parent Threat
3. Expand Classification Tree
4. Step Down the Expanded Classification Tree to Find Related Attack Patterns

Figure 8. Benefit of Adding Parent Threat Element for Locating Related Threats.

Both approaches in Figure 8 assume the user is currently reviewing the details (elements) of a specific attack pattern. Process 2 not only reduces the number of required steps, but includes the necessary information needed to know from what Parent Threat the chosen attack pattern is derived. There is no heuristic nature to this process as the relationship between attack pattern and Parent Threat is already documented in the CAPEC Classification Tree.

In addition to the Parent Threat element, we considered proposing a "Related CAPEC Attack Pattern" element. However due to the criticism of the large number of descriptive elements currently used for each attack pattern, we chose not to create an additional element (Pauli & Engebretson, 2008a, 2008b). Adopters who are interested in finding related CAPEC attack patterns will be aided by the creation of our new Parent Threat element.

To demonstrate how this process makes the CAPEC dictionary more usable, we introduce attack pattern 98 "Phishing". It is not possible to find related attacks or a general threat for attack pattern 98 in CAPEC's current format. By leveraging our process, it is now explicitly known that "Phishing" is a type of "Spoofing" attack. Furthermore, CAPEC lists the following attack patterns under "Spoofing".

- Leveraging/Manipulating Configuration File Search Paths (Attack Pattern 38)
- Man in the Middle Attack (Attack Pattern 94)
- Utilizing Rest's Trust in the System Resources to Register Man in the Middle (Attack Pattern 57)
- Reflection Attack in Authentication Protocol (Attack Pattern 90)
- Pharming (Attack Pattern 89)

Because the Parent Threat is included as one of the attack pattern elements, it is now known what related threats should also be considered in addition to "Phishing" when concerned with "Spoofing".

4.2. Trimming the Element Set

The current CAPEC Release 1 dictionary includes 101 attack patterns with each attack pattern including up to 31 descriptive elements. The volume of information presented in the current CAPEC dictionary presents a major obstacle for the learning and application of the standard (Pauli & Engebretson, 2008a, 2008b). Furthermore, the currently prescribed element set is inconsistent. For example, attack patterns 17 (Accessing, Modifying or Executing Executable Files), 33 (HTTP Request Smuggling) and 67 (String Format Overflow in syslog()) each include the Injection Vector, Payload, and Activation Zone elements while attack patterns 1 (Accessing Functionality Not Properly Constrained by ACLs), 22 (Exploiting Trust in Client), and 44 (Overflow Binary Resource File) do not. Reducing the element set provides a level of consistency and usability by leveraging our newly added Parent Threat and Parent Mitigation elements.

Step 2 of Process 2 is the refining of the CAPEC descriptive elements. The outcome of this process is a new model for viewing relevant information about each attack pattern. We trim the CAPEC elements by focusing on elements which can be used to portray the attack fully and provide a meaningful representation for the user. It is important that our new model makes use of a reduced number of elements while still describing the attack pattern in totality. We selected the following attack pattern elements:

- Parent Threat
- Attack Pattern ID
- Attack Pattern Name
- Description
- Solutions and Mitigations

- Parent Mitigation

To complete step 2, we ensure that each element (listed above) for each attack pattern has at least one entry. We avoid the large number of descriptive elements which lead to information overload when attempting to make use of the CAPEC dictionary (Pauli & Engebretson, 2008a, 2008b). The process of trimming the element set from 31 to 6 is justified by examining each of the selected elements. The Parent Threat element is included because it increases usability by grouping related attack patterns as introduced in step 1. Leveraging the significant work done in creating the Parent Threat element requires the use of the Attack Pattern ID. As a result, we include both the Attack Pattern ID and the Parent Threat. Parent Mitigation is added because it documents a consistent and usable mitigation strategy for each attack pattern. Parent Mitigation is based on the Solutions and Mitigations element so we include both of these elements into our model. Finally, Attack Pattern Name and Description are included as they are essential to complete the description of the attack pattern. The use of a hierarchy capped by Parent Threat and Parent Mitigation elements allows for the tracing of attack patterns from individual Parent Mitigations up to Parent Threats and vice versa.

Previously accepted models have made use of a reduced CAPEC element set for the purpose of introducing new audiences to the CAPEC standard without overwhelming them (Pauli & Engebretson, 2008b). However, like the current CAPEC dictionary, previous models were inconsistent in their use of the descriptive elements. Our model requires that each of the selected elements is used for all 101 attack patterns.

The goal of trimming the element set is not to advocate element set replacement, but rather to present CAPEC adopters with a simple, easy-to-use, organized, and uniform presentation of all 101 attack patterns. The original CAPEC library will be available for further detail review.

The new hierarchy model will be used to complete steps 3 and 4 of Process 2. The use of a hierarchy makes the graphical trees more usable because it demonstrates previously undefined relationships. The new model allows us to view relationship between elements, attack patterns, Parent Threats, and Parent Mitigations. The use of a graphical tree allows users to view details from either the Parent Threat or Parent Mitigation point of view. The model will be used to make the textual attack descriptions more usable by providing a stand-alone view of each attack pattern. The use of a textual attack description is beneficial for allowing users to view details from an attack pattern-driven point of view.

Our usable model is consistent because all hierarchy trees and textual descriptions are completely populated.

4.3. Building Hierarchy-Based Graphical Trees and Textual Attack Descriptions

Hierarchy-based representations of attack patterns have been previously used to facilitate learning of the CAPEC standard (Pauli & Engebretson, 2008a). One of the major categories for learning strategies is the creation or use of a hierarchy (Weinstein & Mayer, 1986). Information presented in a hierarchical fashion is easier to learn and recall than information presented in a format with not clear connection between details (Weinstein & Mayer, 1986). Presenting attack pattern elements via a hierarchy allows users to see the connections between each of the elements. This knowledge can then be leveraged when analyzing and designing secure software, building networks, or making security related decisions.

Our model includes both a defined graphical tree hierarchy for describing element organization, structure and relationships as well as textual attack descriptions for presenting readable attack pattern details. The use of a hierarchy can also help to define relationships among the 101 attack patterns, 17 Parent Mitigations, and 11 Parent Threats.

We apply our hierarchy in a fan-in-fan-out manner to the trimmed element set selected in section 4.3. The highest level of our hierarchy provides general and wide-ranging information. The Parent Threat element is used at the top level because it is broad. There are only 11 possible Parent Threats. Subsequent hierarchy levels become specific in nature and scope. Attack pattern ID, Attack Pattern Name, Description, and the Solutions and Mitigations elements are each specific to a single attack pattern and cannot be generalized. The model concludes with a fan-out approach as it abstracts back out to more general information in the Parent Mitigation element. This element represents a direct abstraction of the 400+ Solutions and Mitigations into 17 possible mitigations. Our hierarchy model for attack pattern elements is introduced in Figure 9.

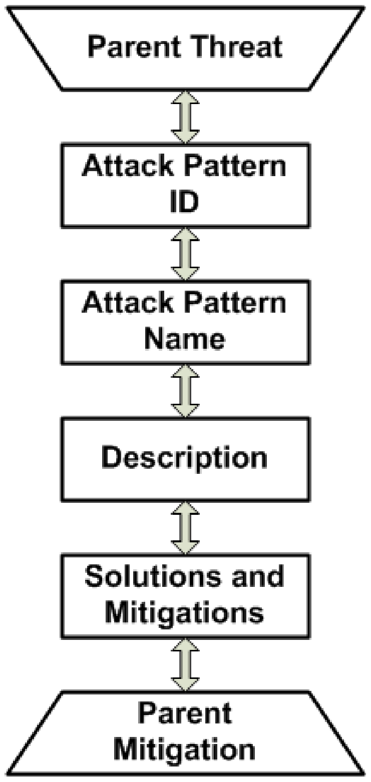


Figure 9. Hierarchy Model for Attack Patterns and Elements.

The hierarchical tree is completed by filling in each of the elements described in Figure 9. The purpose of this representation is to show relationships between elements and attack patterns. This representation can also be used to group attack patterns by related Parent Threats or Parent Mitigations. The textual attack descriptions, which are presented in a tabular format for each attack pattern, aid in documenting all of the selected elements and information for each attack pattern. Textual attack descriptions also provide consistency and usability for each attack pattern.

Step 3 of Process 2 creates both the graphical hierarchies and the textual attack descriptions for the attack pattern. We create the graphical hierarchies by filling in the required elements outlined in figure 9. The case study results for step 3 for Process 2 for attack pattern 3 and 75 are presented below. The remaining graphical hierarchies for our case study are shown in section 4.4.

The completed graphical hierarchy for attack patterns 3 is introduced in Figure 10.

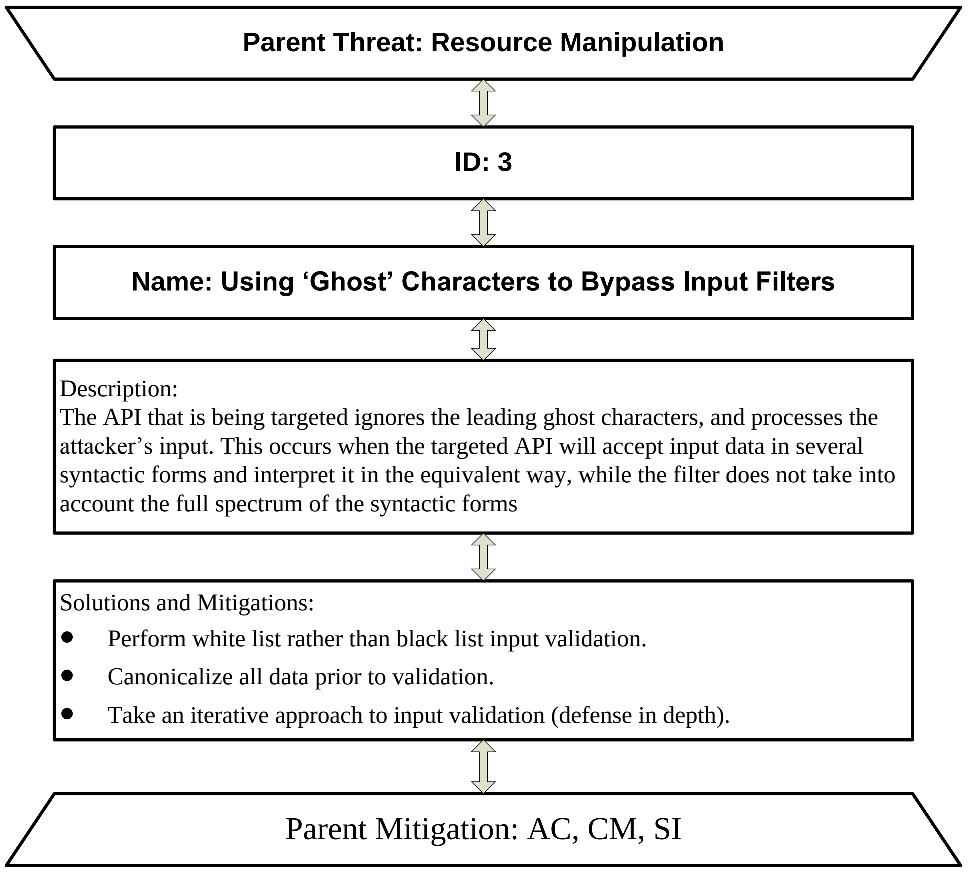

Parent Threat: Resource Manipulation
ID: 3
Name: Using 'Ghost' Characters to Bypass Input Filters
Description:
The API that is being targeted ignores the leading ghost characters, and processes the attacker's input. This occurs when the targeted API will accept input data in several syntactic forms and interpret it in the equivalent way, while the filter does not take into account the full spectrum of the syntactic forms
Solutions and Mitigations:
Perform white list rather than black list input validation.
Canonicalize all data prior to validation.
Take an iterative approach to input validation (defense in depth).
Parent Mitigation: AC, CM, SI

Figure 10. Graphical Hierarchy Tree for Attack Pattern 3 ("Using 'Ghost' Characters to Bypass Input Filters").

The completed graphical hierarchy for attack patterns 75 is introduced in Figure 11.

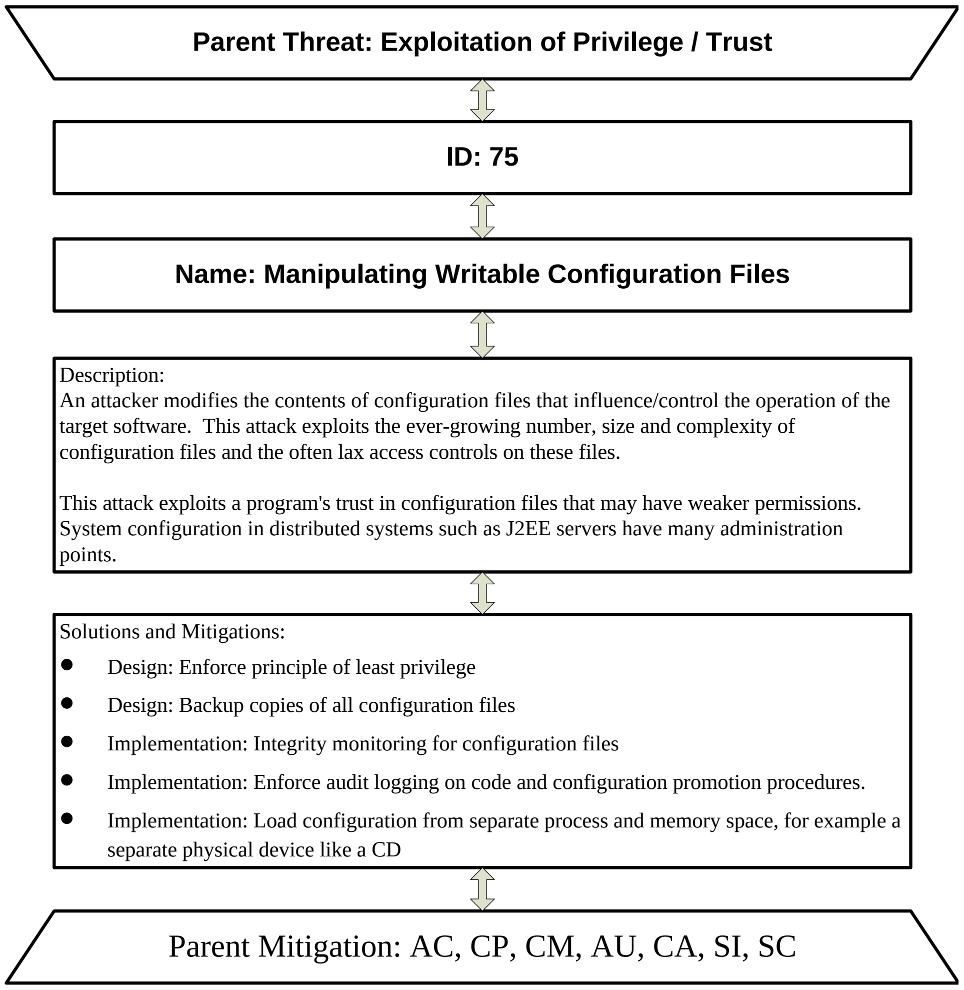

Parent Threat: Exploitation of Privilege / Trust
ID: 75
Name: Manipulating Writable Configuration Files
Description:
An attacker modifies the contents of configuration files that influence/control the operation of the target software. This attack exploits the ever-growing number, size and complexity of configuration files and the often lax access controls on these files.
This attack exploits a program's trust in configuration files that may have weaker permissions. System configuration in distributed systems such as J2EE servers have many administration points.
Solutions and Mitigations:
Design: Enforce principle of least privilege
Design: Backup copies of all configuration files
Implementation: Integrity monitoring for configuration files
Implementation: Enforce audit logging on code and configuration promotion procedures.
Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD
Parent Mitigation: AC, CP, CM, AU, CA, SI, SC

Figure 11. Graphical Hierarchy Tree for Attack Pattern 75 ("Manipulating Writable Configuration Files").

Step 4 creates a textual attack description which presents attack pattern information in a tabular format. This process is completed by extracting information from the hierarchies into the textual template provided in table 34.

Table 34. Example of Textual Attack Description.

Attack Pattern ID	
Attack Pattern Name	
Description	
Parent Threat	
Solutions and Mitigations	
Parent Mitigation	

The outcome of step 4 provides a trimmed element set which is usable, readable, and presents information from the individual attack pattern perspective for each of the 101 CAPEC attacks. The textual attack descriptions for attack patterns 3 and 75 are introduced below. The outcome of step 4 for the remaining attack patterns can be found in section 4.4. Table 35 introduces the completed textual attack description for attack pattern 3.

Table 35. Textual Attack Description for Attack Pattern 3 (“Using ‘Ghost’ Characters to Bypass Input Filters”).

Attack Pattern ID	3
Attack Pattern Name	Using ‘Ghost’ Characters to Bypass Input Filters
Description	The API that is being targeted ignores the leading ghost characters, and processes the attacker’s input. This occurs when the targeted API will accept input data in several syntactic forms and interpret it in the equivalent way, while the filter does not take into account the full spectrum of the syntactic forms acceptable to the targeted API.
Parent Threat	Resource Manipulation
Solutions and Mitigations	Perform white list rather than black list input validation. Canonicalize all data prior to validation. Take an iterative approach to input validation (defense in depth).
Parent Mitigation	AC, CM, SI

Table 36 introduces the textual description for attack pattern 75.

Table 36. Textual Attack Description for Attack Pattern 75 ("Manipulating Writable Configuration Files").

Attack Pattern ID	75
Attack Pattern Name	Manipulating Writable Configuration Files
Description	An attacker modifies the contents of configuration files that influence/control the operation of the target software. This attack exploits the ever-growing number, size and complexity of configuration files and the often lax access controls on these files. This attack exploits a program's trust in configuration files that may have weaker permissions. System configuration in distributed systems such as J2EE servers have many administration points. For example, permissions may be set on the administrative GUI, the configuration file for the server as a whole, configuration files for specific domains and applications, special jar and other class files used to load resources at runtime, and even policy specific in .war and .ear files. A mistake in permissions setting in either the file acl or the content is an opening an attacker can use to elevate privilege.
Parent Threat	Exploitation of Privilege / Trust
Solutions and Mitigations	Design: Enforce principle of least privilege Design: Backup copies of all configuration files Implementation: Integrity monitoring for configuration files Implementation: Enforce audit logging on code and configuration promotion procedures. Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD
Parent Mitigation	AC, CP, CM, AU, CA, SI, SC

4.4. Results of Case Study

Our case study presents one attack pattern from each Parent Threat. The results of our case study for Step 1 are presented below where Table 37 introduces the results of Step 1 for Process 2 for attack pattern 87.

Table 37. Addition of Parent Threat Column for Attack Pattern 87 ("Forceful Browsing").

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Abuse of Functionality	Forceful Browsing: **87**	Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.	AC17, IA2, IA3, MA4, SC8, SC23, SI10	AC, IA, MA, SC, SI
		Forceful browsing can also be made difficult to a large extent by not hard coding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.	SC18, AT3, CA2, CA4, PL2, SA3, SA8, SA10	AT, CA, PL,SA

Table 38 introduces the results of Step 1 for Process 2 for attack pattern 94.

Table 38. Addition of Parent Threat Column for Attack Pattern 94 (“Man in the Middle”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Spoofing	Man the Middle: **94**	Get your Public Key signed by a Certificate Authority	CA4, IA5, IA7, SC13, SC17	CA, IA, SC
		Encrypt your communication using cryptography (SSL,...)	AC3, AC4, SC7, AC17, IA7, SC8, SC9, SC12, SC13, SI7	AC ,SI
		Use Strong mutual authentication to always fully authenticate both ends of any communications channel.	AC17, IA1, IA2, IA3, IA4, IA5, SC8, SC11, SC23, SI10	
		Exchange public keys using a secure channel	SC17, SC12, SC13	

Table 39 introduces the results of Step 1 for Process 2 for attack pattern 55.

Table 39. Addition of Parent Threat Column for Attack Pattern 55 (“Rainbow Table Password Cracking”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Probabilistic Techniques	Rainbow Table Pswd Cracking: **55**	Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.	SI7, SC13, IA5	SI, SC, IA

Table 40 introduces the results of Step 1 for Process 2 for attack pattern 60.

Table 40. Addition of Parent Threat Column for Attack Pattern 60 (“Reusing Session Id’s”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Exploitation of Authorization	Reusing Session ID's: **60**	Always invalidate a session ID after the user logout.	AC3, IA5, SC10, SC23, IA4	AC, IA, SC
		Setup a session time out for the session IDs.	AC11, AC12, SC23, IA4	
		Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.	AC4, IA2, IA3, IA7, SC8, SC9, SC11, SC12, SC13, SC16, SC17, SC20, SC21, SC22, SC23	SA
		Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.	SC9, SC4, SC14, SC16, SA8	
		Encrypt the session data associated with the session ID.	AC3, SC4, SC7, SC23	
		Use multifactor authentication.	IA2	

Table 41 introduces the results of Step 1 for Process 2 for attack pattern 82.

Table 41. Addition of Parent Threat Column for Attack Pattern 82 (XML Denial of Service”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Resource Depletion	XMLDoS (XDoS): **82**	Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.	AC4, SI9, SI10, AC3, CM6	AC, SI, CM
		Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers	AC4, CA3, SC6, SI4, CP10, SC5, SC22	CA,SC, CP
		Implementation: Check size of XML message before parsing	SI7, SI9, SI10	

Table 42 introduces the results of Step 1 for Process 2 for attack pattern 65.

Table 42. Addition of Parent Threat Column for Attack Pattern 65 (“Passive Sniffing”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Data Leakage Attacks	Passive Sniffing: **65**	Do not store secrets in client code	CM6, PE19, RA3, SA8, PL4	CM, PE, RA, SA, PL
		Use Well-Known Cryptography Appropriately and Correctly	AC3, AC17, IA7, MA4, SC8, SC9, SC12, SC13	AC, IA, MA, SC
		Use Authentication Mechanisms, Where Appropriate, Correctly	IA2, IA7, SC23, SI10	SI

Table 43 introduces the results of Step 1 for Process 2 for attack pattern 101.

Table 43. Addition of Parent Threat Column for Attack pattern 101 (“Server Side Includes”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Injection	Server Side Includes (SSI): **101**	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them	CM1, CM6, CM7, SI6, SC3, AC6	CM, SI, SC, AC
		All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive	SI7, SI9, SI10	
		Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead	AC6	

Table 44 introduces the results of Step 1 for Process 2 for attack pattern 10.

Table 44. Addition of Parent Threat Column for Attack Pattern 10 (“Buffer Overflow via Environment Variables”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Data Structure Attacks	Buffer Overflow via Environment Variables: **10**	Do not expose environment variable to the user.	AC6, CM6, RA3, RA5, SA10, SA11, SC4, SI10	AC, CM, RA, SA, SC, SI
		Do not use untrusted data in your environment variables.	AC3, CM6, IA2, SC23, SI17, SI19, SI10	IA,
		Use a language or compiler that performs automatic bounds checking	SA8, PL2	PL
		There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.	MA3, PL6 RA5, SA10, SA11, SI2, SI4	MA

Table 45 introduces the results of Step 1 for Process 2 for attack pattern 29.

Table 45. Addition of Parent Threat Column for Attack Pattern 29 (“Race Conditions, Time of Check Time of Use”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**
Time and State Attacks	Race Conditions (TOCTOU): **29**	Use safe libraries to access resources such as files.	SI7, SC18,	SI, SC
		Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.	AT2, AC3, IA2	AT, AC, IA
		Use synchronization to control the flow of execution.	SC3, AC4	
		Use static analysis tools to find race conditions.	SA11,SI10	
		Pay attention to concurrency problems related to the access of resources.	SA8, SC4	SA

Step 3 of Process 2 requires us to create both the graphical hierarchies and the textual attack descriptions for the attack pattern. We create the graphical hierarchies by filling in the required elements outlined in Figure 9 for each attack pattern. The completed graphical hierarchy for attack pattern 87 is introduced in Figure 12.

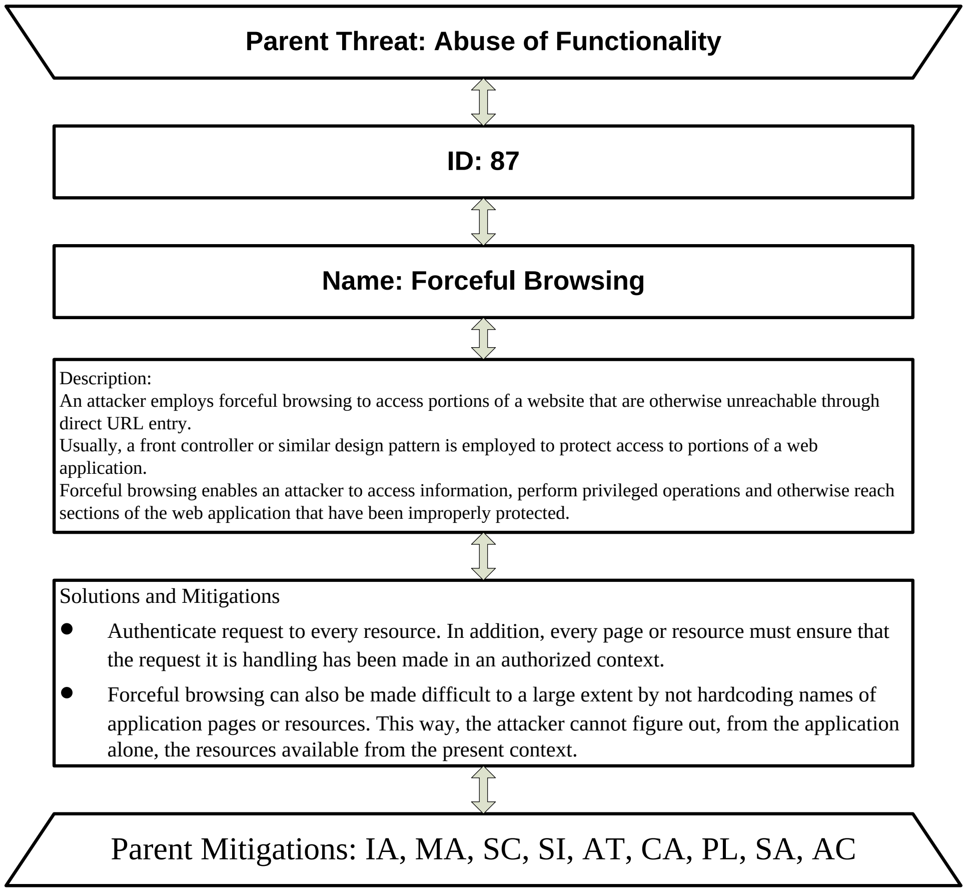
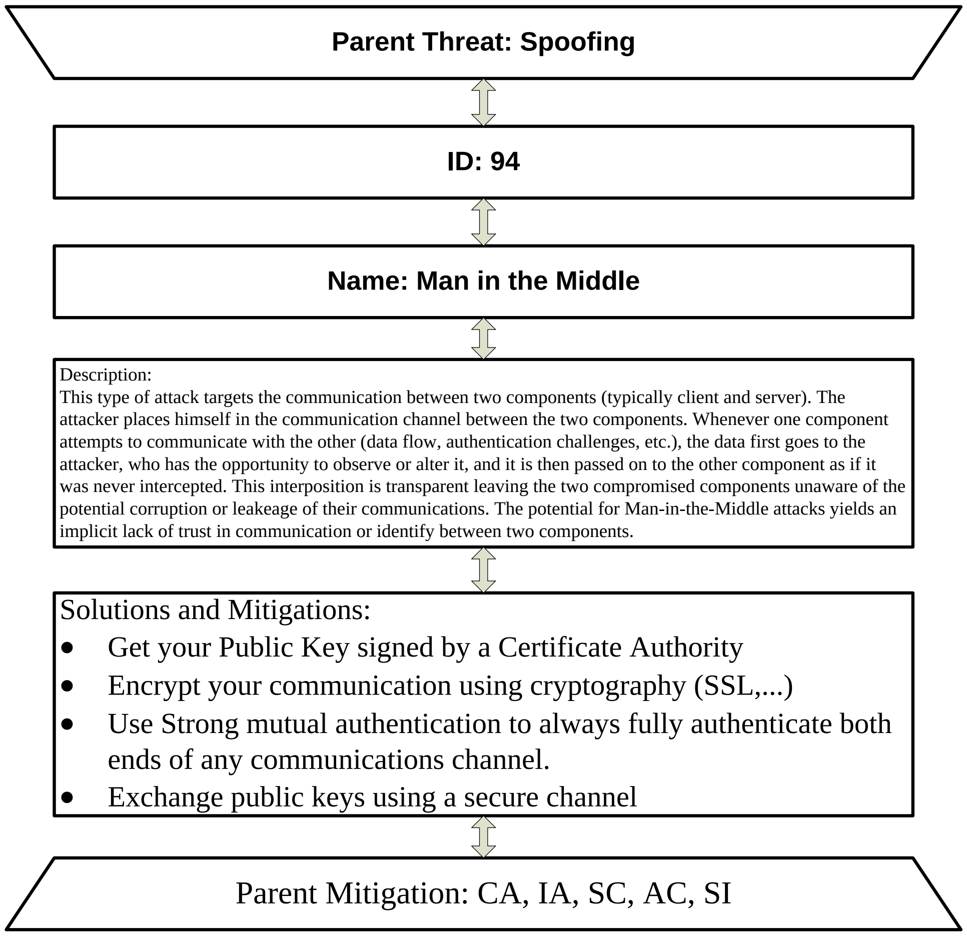

Parent Threat: Abuse of Functionality
ID: 87
Name: Forceful Browsing
Description:
An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry.
Usually, a front controller or similar design pattern is employed to protect access to portions of a web application.
Forceful browsing enables an attacker to access information, perform privileged operations and otherwise reach sections of the web application that have been improperly protected.
Solutions and Mitigations
● Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.
● Forceful browsing can also be made difficult to a large extent by not hardcoding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.
Parent Mitigations: IA, MA, SC, SI, AT, CA, PL, SA, AC

Figure 12. Graphical Hierarchy Tree for Attack Pattern 87 (“Forceful Browsing”).

The completed graphical hierarchy for attack pattern 94 is introduced in Figure 13.

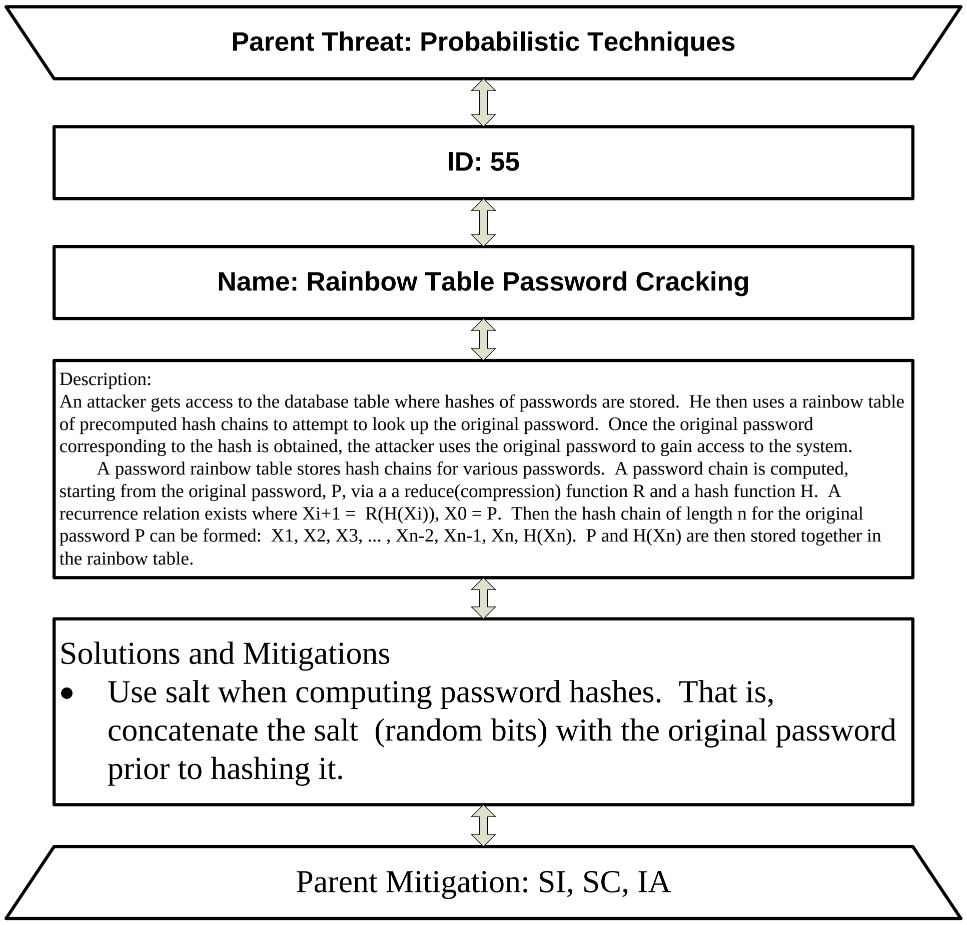


Description:
This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.

Solutions and Mitigations:
- Get your Public Key signed by a Certificate Authority
- Encrypt your communication using cryptography (SSL,...)
- Use Strong mutual authentication to always fully authenticate both ends of any communications channel.
- Exchange public keys using a secure channel

Figure 13. Graphical Hierarchy Tree for Attack Pattern 94 (“Man in the Middle”).

The completed graphical hierarchy for attack pattern 55 is introduced in Figure 14.


Parent Threat: Probabilistic Techniques
ID: 55
Name: Rainbow Table Password Cracking

Description:
An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system.

A password rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table.

Solutions and Mitigations

- Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.

Figure 14. Graphical Hierarchy Tree for Attack Pattern 94 (“Rainbow Table Password Cracking”).

The completed graphical hierarchy for attack pattern 60 is introduced in Figure 15.

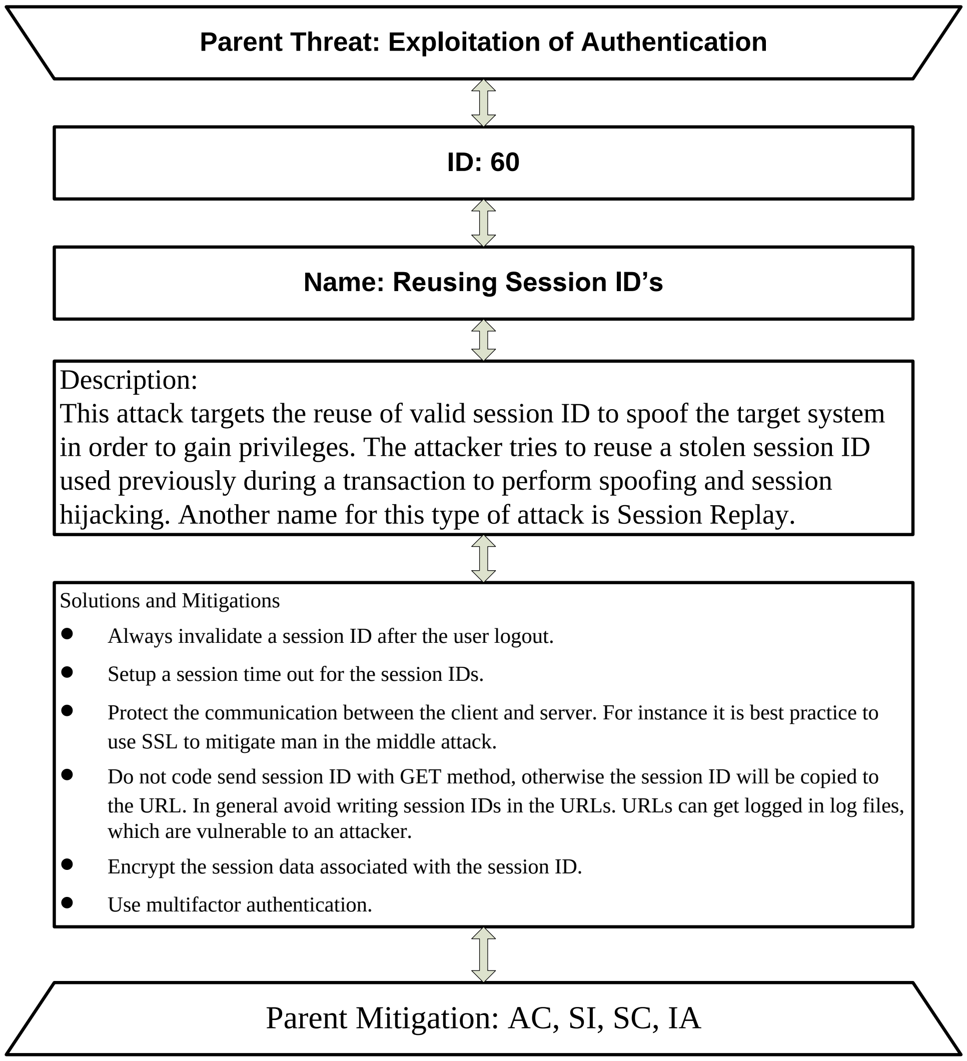
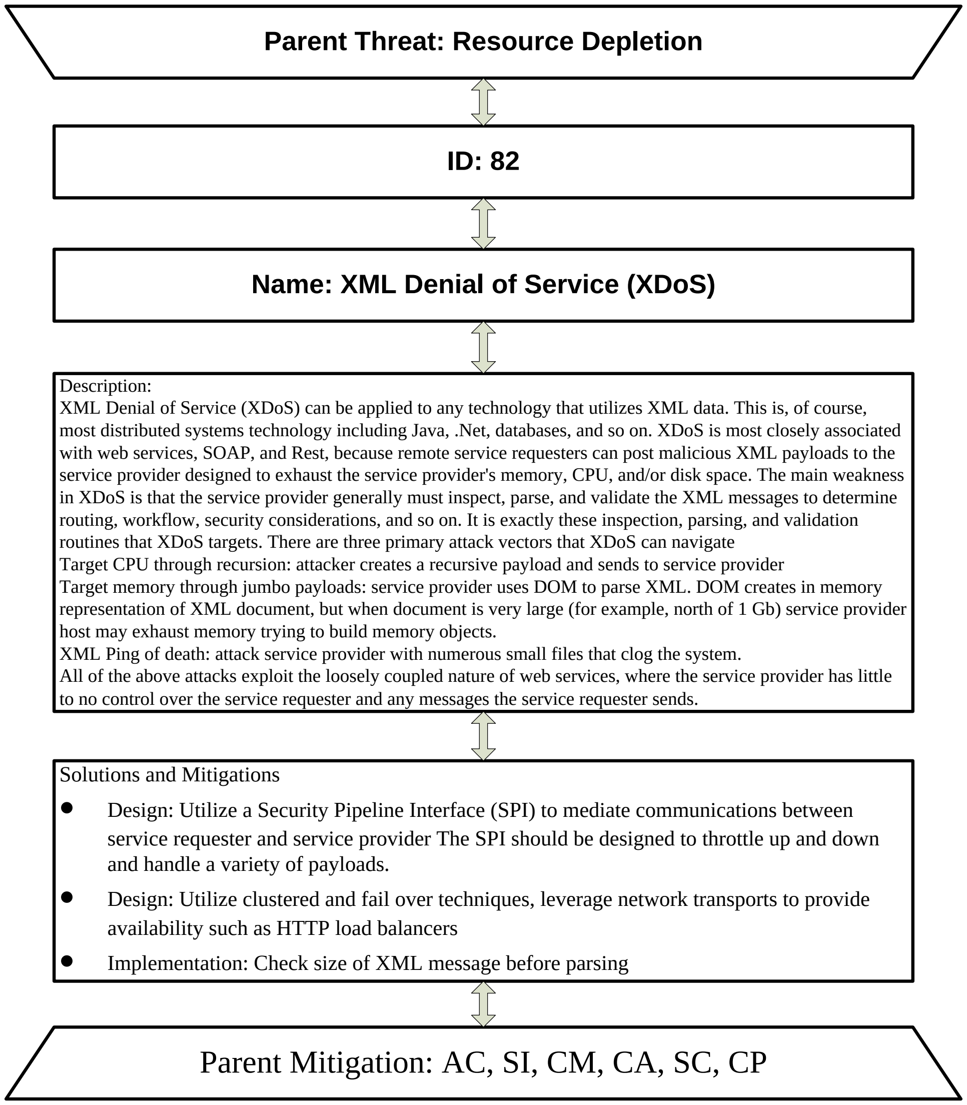

Parent Threat: Exploitation of Authentication
ID: 60
Name: Reusing Session ID’s
Description:
This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.
Solutions and Mitigations
Always invalidate a session ID after the user logout.
Setup a session time out for the session IDs.
Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.
Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.
Encrypt the session data associated with the session ID.
Use multifactor authentication.
Parent Mitigation: AC, SI, SC, IA

Figure 15. Graphical Hierarchy Tree for Attack Pattern 60 (“Reusing Session ID’s”).

The completed graphical hierarchy for attack pattern 82 is introduced in Figure 16.


Parent Threat: Resource Depletion
ID: 82
Name: XML Denial of Service (XDoS)

Description:
XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space. The main weakness in XDoS is that the service provider generally must inspect, parse, and validate the XML messages to determine routing, workflow, security considerations, and so on. It is exactly these inspection, parsing, and validation routines that XDoS targets. There are three primary attack vectors that XDoS can navigate
Target CPU through recursion: attacker creates a recursive payload and sends to service provider
Target memory through jumbo payloads: service provider uses DOM to parse XML. DOM creates in memory representation of XML document, but when document is very large (for example, north of 1 Gb) service provider host may exhaust memory trying to build memory objects.
XML Ping of death: attack service provider with numerous small files that clog the system.
All of the above attacks exploit the loosely coupled nature of web services, where the service provider has little to no control over the service requester and any messages the service requester sends.

Solutions and Mitigations

- Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.
- Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers
- Implementation: Check size of XML message before parsing

Figure 16. Graphical Hierarchy Tree for Attack Pattern 82 (“XML Denial of Service”).

The completed graphical hierarchy for attack pattern 101 is introduced in Figure 17.

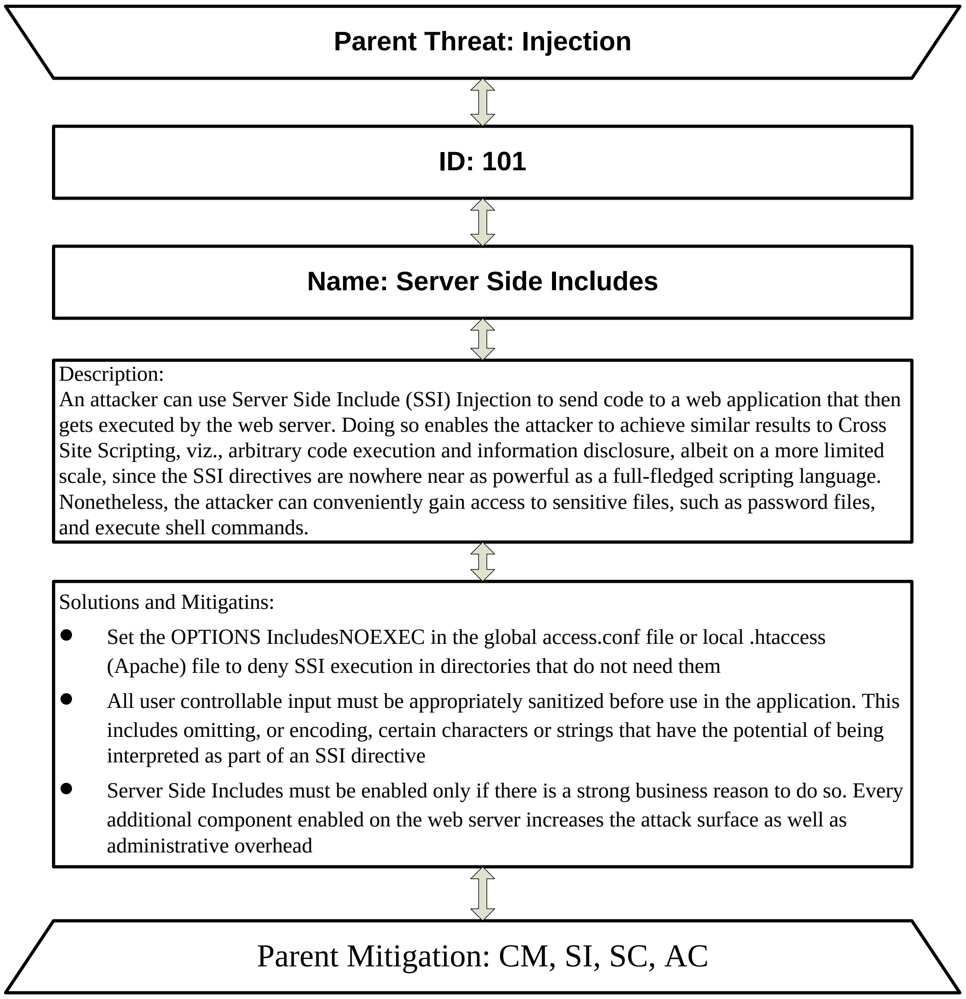

Parent Threat: Injection
ID: 101
Name: Server Side Includes
Description:
An attacker can use Server Side Include (SSI) Injection to send code to a web application that then gets executed by the web server. Doing so enables the attacker to achieve similar results to Cross Site Scripting, viz., arbitrary code execution and information disclosure, albeit on a more limited scale, since the SSI directives are nowhere near as powerful as a full-fledged scripting language. Nonetheless, the attacker can conveniently gain access to sensitive files, such as password files, and execute shell commands.
Solutions and Mitigatins:
● Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them
● All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive
● Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead
Parent Mitigation: CM, SI, SC, AC

Figure 17. Graphical Hierarchy Tree for Attack Pattern 101 (“Server Side Includes”).

The completed graphical hierarchy for attack pattern 10 is introduced in Figure 18.

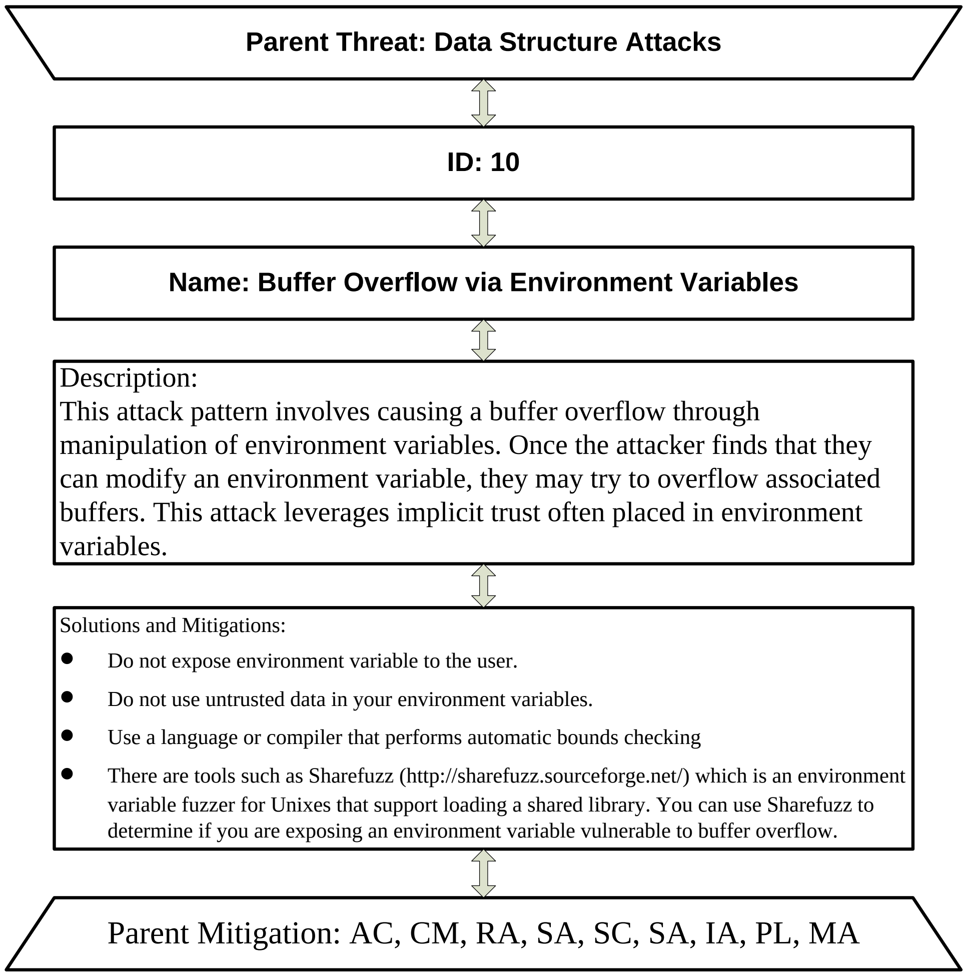

Parent Threat: Data Structure Attacks
ID: 10
Name: Buffer Overflow via Environment Variables
Description:
This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.
Solutions and Mitigations:
Do not expose environment variable to the user.
Do not use untrusted data in your environment variables.
Use a language or compiler that performs automatic bounds checking
There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.
Parent Mitigation: AC, CM, RA, SA, SC, SA, IA, PL, MA

Figure 18. Graphical Hierarchy Tree for Attack Pattern 10 (“Buffer Overflow via Environment Variables”).

The completed graphical hierarchy for attack pattern 65 is introduced in Figure 19.

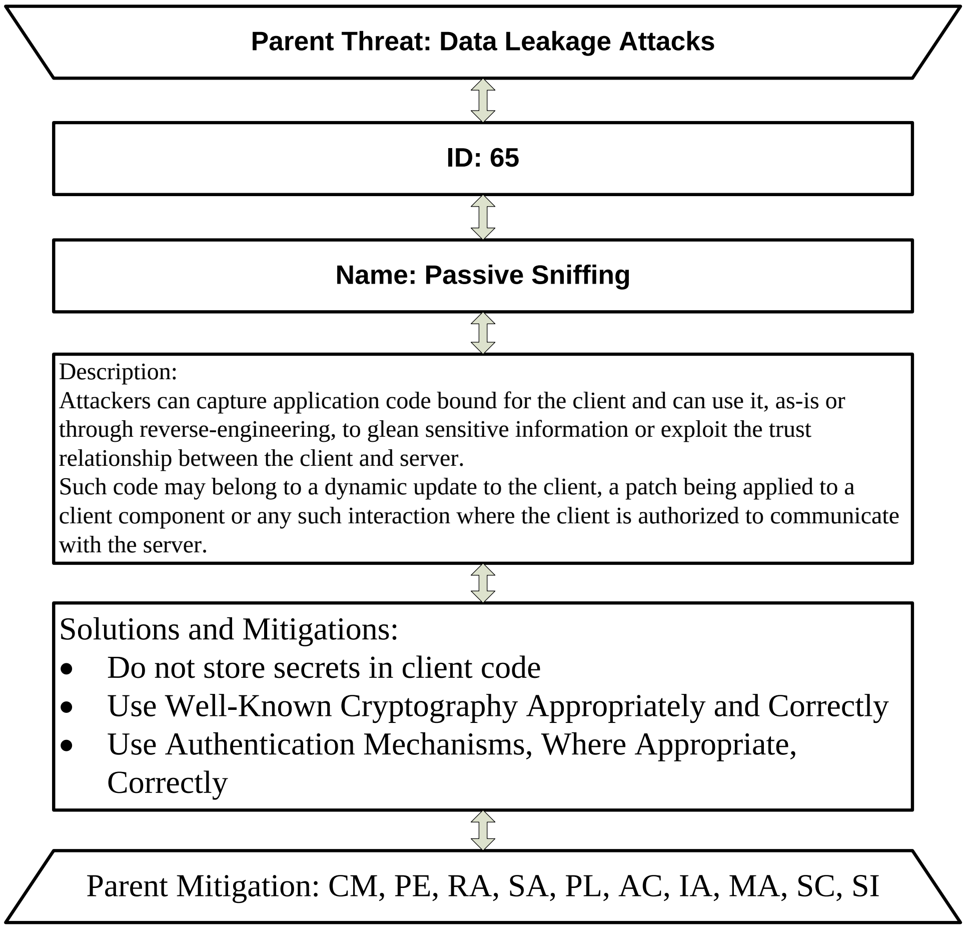

Parent Threat: Data Leakage Attacks
ID: 65
Name: Passive Sniffing
Description:
Attackers can capture application code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server.
Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.
Solutions and Mitigations:
• Do not store secrets in client code
• Use Well-Known Cryptography Appropriately and Correctly
• Use Authentication Mechanisms, Where Appropriate, Correctly
Parent Mitigation: CM, PE, RA, SA, PL, AC, IA, MA, SC, SI

Figure 19. Graphical Hierarchy Tree for Attack Pattern 65 (“Passive Sniffing”).

The completed graphical hierarchy for attack pattern 29 is introduced in Figure 20.

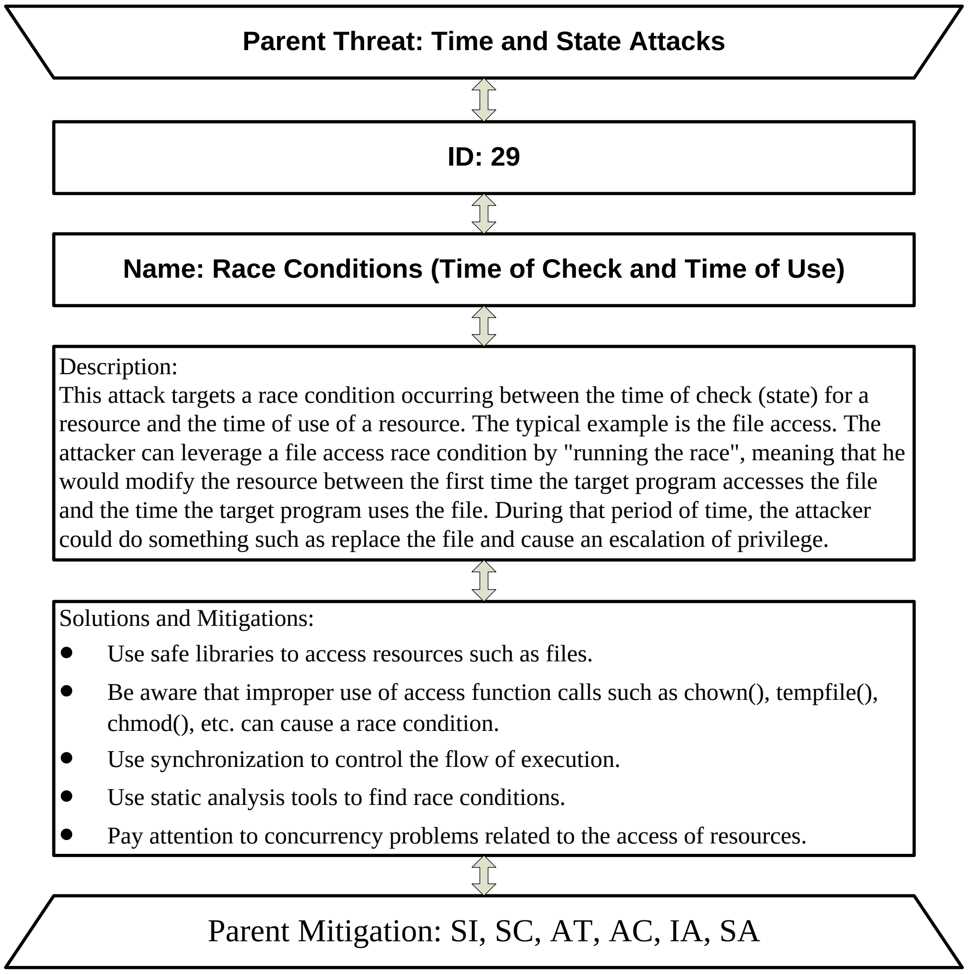

Parent Threat: Time and State Attacks
ID: 29
Name: Race Conditions (Time of Check and Time of Use)
Description:
This attack targets a race condition occurring between the time of check (state) for a resource and the time of use of a resource. The typical example is the file access. The attacker can leverage a file access race condition by "running the race", meaning that he would modify the resource between the first time the target program accesses the file and the time the target program uses the file. During that period of time, the attacker could do something such as replace the file and cause an escalation of privilege.
Solutions and Mitigations:
• Use safe libraries to access resources such as files.
• Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.
• Use synchronization to control the flow of execution.
• Use static analysis tools to find race conditions.
• Pay attention to concurrency problems related to the access of resources.
Parent Mitigation: SI, SC, AT, AC, IA, SA

Figure 20. Graphical Hierarchy Tree for Attack Pattern 65 (“Race Conditions Time of Check and Time of Use”).

Step 4 calls for the creation of a textual attack description and is completed by extracting information from the hierarchies into the textual template provided in Table 35. Table 46 introduces the textual attack description for attack pattern 87.

Table 46. Textual Attack Description for Attack Pattern 87 (“Forceful Browsing”).

Attack Pattern ID	87
Attack Pattern Name	Forceful Browsing
Description	An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry. Usually, a front controller or similar design pattern is employed to protect access to portions of a web application. Forceful browsing enables an attacker to access information, perform privileged operations and otherwise reach sections of the web application that have been improperly protected.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context. Forceful browsing can also be made difficult to a large extent by not hardcoding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.
Parent Mitigation	IA, MA, SC, SI, AT, CA, PL, SA, AC

Table 47 introduces the textual attack description for attack pattern 94.

Table 47. Textual Attack Description for Attack Pattern 94 ("Man in the Middle").

Attack Pattern ID	94
Attack Pattern Name	Man in the Middle
Description	This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.
Parent Threat	Spoofing
Solutions and Mitigations	Get your Public Key signed by a Certificate Authority Encrypt your communication using cryptography (SSL,...) Use Strong mutual authentication to always fully authenticate both ends of any communications channel. Exchange public keys using a secure channel
Parent Mitigation	CA, IA, SC, AC, SI

Table 48 introduces the textual attack description for attack pattern 55.

Table 48. Textual Attack Description for Attack Pattern 55 (“Rainbow Table Password Cracking”).

Attack Pattern ID	55
Attack Pattern Name	Rainbow Table Password Cracking
Description	An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system. A pasword rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.
Parent Mitigation	SI, SC, IA

Table 49 introduces the textual attack description for attack pattern 60.

Table 49. Textual Attack Description for Attack Pattern 60 ("Reusing Session ID's").

Attack Pattern ID	60
Attack Pattern Name	Reusing Session ID's
Description	This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	Always invalidate a session ID after the user logout. Setup a session time out for the session IDs. Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack. Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker. Encrypt the session data associated with the session ID. Use multifactor authentication.
Parent Mitigation	AC, IA, SC, SA

Table 50 introduces the textual attack description for attack pattern 82.

Table 50. Textual Attack Description for Attack Pattern 82 ("XML Denial of Service").

Attack Pattern ID	82
Attack Pattern Name	XMLDoS (XDoS)
Description	XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space. The main weakness in XDoS is that the service provider generally must inspect, parse, and validate the XML messages to determine routing, workflow, security considerations, and so on. It is exactly these inspection, parsing, and validation routines that XDoS targets. There are three primary attack vectors that XDoS can navigate Target CPU through recursion: attacker creates a recursive payload and sends to service provider Target memory through jumbo payloads: service provider uses DOM to parse XML. DOM creates in memory representation of XML document, but when document is very large (for example, north of 1 Gb) service provider host may exhaust memory trying to build memory objects. XML Ping of death: attack service provider with numerous small files that clog the system. All of the above attacks exploit the loosely coupled nature of web services, where the service provider has little to no control over the service requester and any messages the service requester sends.
Parent Threat	Resource Depletion
Solutions and Mitigations	Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads. Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers Implementation: Check size of XML message before parsing
Parent Mitigation	AC, SI, CM, CA, SC, CP

Table 51 introduces the textual attack description for attack pattern 101.

Table 51. Textual Attack Description for Attack Pattern 101 (“Server Side Includes”).

Attack Pattern ID	101
Attack Pattern Name	Server Side Includes (SSI)
Description	An attacker can use Server Side Include (SSI) Injection to send code to a web application that then gets executed by the web server. Doing so enables the attacker to achieve similar results to Cross Site Scripting, viz., arbitrary code execution and information disclosure, albeit on a more limited scale, since the SSI directives are nowhere near as powerful as a full-fledged scripting language. Nonetheless, the attacker can conveniently gain access to sensitive files, such as password files, and execute shell commands.
Parent Threat	Injection
Solutions and Mitigations	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead
Parent Mitigation	CM, SI, SC, AC

Table 52 introduces the textual attack description for attack pattern 10.

Table 52. Textual Description for Attack Pattern 10 (“Buffer Overflow via Environment Variable”).

Attack Pattern ID	10
Attack Pattern Name	Buffer Overflow via Environment Variables
Description	This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	Do not expose environment variable to the user. Do not use untrusted data in your environment variables. Use a language or compiler that performs automatic bounds checking There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.
Parent Mitigation	AC, CM, RA, SA, SC, SA, IA, PL, MA

Table 53 introduces the textual attack description for attack pattern 65.

Table 53. Textual Attack Description for Attack Pattern 65 (“Passive Sniffing”).

Attack Pattern ID	65
Attack Pattern Name	Passive Sniffing
Description	Attackers can capture appplication code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server. Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.
Parent Threat	Data Leakage Attack
Solutions and Mitigations	Do not store secrets in client code Use Well-Known Cryptography Appropriately and Correctly Use Authentication Mechanisms, Where Appropriate, Correctly
Parent Mitigation	CM, PE, RA, SA, PL, AC, IA, MA, SC, SI

Table 54 introduces the textual attack description for attack pattern 29.

Table 54. Textual Description for Attack Pattern 29 (“Race Conditions Time of Check and Time of Use”).

Attack Pattern ID	29
Attack Pattern Name	Race Conditions (Time of Check and Time of Use)
Description	This attack targets a race condition occurring between the time of check (state) for a resource and the time of use of a resource. The typical example is the file access. The attacker can leverage a file access race condition by "running the race", meaning that he would modify the resource between the first time the target program accesses the file and the time the target program uses the file. During that period of time, the attacker could do something such as replace the file and cause an escalation of privilege.
Parent Threat	Time and State Attacks
Solutions and Mitigations	Use safe libraries to access resources such as files. Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition. Use synchronization to control the flow of execution. Use static analysis tools to find race conditions. Pay attention to concurrency problems related to the access of resources.
Parent Mitigation	SI, SC, AT, AC, IA, SA

4.5. Discussion and Validation

While the current CAPEC standard provides a significant amount of valuable information, there are tremendous variations in the depth and breadth of the defined element set currently outlined for each attack pattern. Users who are presented with vast amounts of information need assistance and a proper plan to avoid feeling overwhelmed, lost, or even frustrated (Rockland, 2000). Process 2 includes a Parent Threat element into the dictionary to provide consistency and usability to the CAPEC Release 1 Dictionary. We proposed a new model for presenting CAPEC attack patterns which provides a standardized element set to provide context and describe how the elements are related. Because the current elements provide valuable information, we are not advocating their removal. Our intention is to present the data in a manageable and consistent manner. Full details of the Release 1 dictionary will be readily available. This is a valuable step to the increased adoption and wide spread acceptance of the CAPEC Release 1 Dictionary.

Our hierarchies allow CAPEC attack pattern information to be viewed from two distinct points of view. Graphical hierarchies allow for viewing element, attack pattern, Parent Threat, and Parent Mitigation relationships. This allows users to trace both up from individual Parent Mitigations or down from individual Parent Threats. The textual attack descriptions present an attack pattern point of view. The creation of this model provides a stand-alone description for understanding the attack pattern.

Validation for Process 2 was provided through the execution of a controlled experiment which produced positive preliminary results to support our claims of increased usability and consistency. Because utilizing subjects who demonstrate an interest in the topic being examined is important for usability studies (Borlund & Ingwersen, 1997), we included a total of ten participants from the undergraduate Computer and Network Security majors at Dakota State University in our experiment. The data from all ten participants was used for analysis. Subjects volunteered for participation and were not compensated for their time. All participants were tested in a computer lab environment with identical computer hardware and software.

The ten participants were divided into two groups where the first group was asked to examine attack pattern details using the current CAPEC Release 1 Dictionary. The second group was asked to examine attack pattern details utilizing the trimmed element set created as a result of Process 2. Group assignment was based on a coin toss; the first five participants to flip “heads” were assigned to the CAPEC group. The remaining five participants were assigned to the trimmed element set group.

The experiment covered a total of six randomly selected attack patterns. For each attack pattern, participants were given ten minutes to complete an assigned task using their predetermined dictionary. All participants worked individually on the same attack pattern at the same time and were asked to complete two tasks per attack pattern. The first task was to locate the selected attack pattern using their assigned dictionary. The second task was to count the number of individual prescribed Solutions and Mitigations for the given attack pattern. There was a master clock to ensure accurate timing. The master clock was set to 0:00 at the beginning of each new attack pattern. Once the attack pattern was revealed to the group, the master clock was started. Participants were asked to record the number of mitigations they found and the time it took to complete both tasks.

Each participant was assigned a number which could be used to track which attack dictionary the subject was using. No other identifying information was collected. Data was recorded by individual participants utilizing Microsoft Notepad. Participants were instructed to create a single Microsoft Notepad log; data from each of the attack pattern experiments was recorded on a single line.

At the conclusion of the experiment participants were asked to answer two questions.

1. “In the process of attempting to locate specific information about specific attack patterns, did you think there was too much data, not enough data, or just the right amount of data about each attack pattern?” Answers for the first question ranged from a low score of, “1 = The dictionary did not contain enough data / elements”, a medium score of “5 = The amount of information in the dictionary was appropriate”, and a high score of “10 = The dictionary contained too much data”.
2. "Concerning usability (structure, organization, ease of use, format, and ability to locate information quickly), how likely are you to use this dictionary again?" Answers for the second question ranged from a low score of “1 = Not Useable / Will Never Use Again”, a medium score of “5 = Indifferent”, and a high score of “10 = Very Useable / Will Definitely Use Again”.

The results of our experiment demonstrate positive preliminary results for both usability and consistency as introduced in Tables 55-60 where “Average Time to Complete Task” is the mean number of seconds it took to participants to complete the experiment. “Average Mitigation Count” is a measurement of the mean number of mitigations found by each participant. “High Mitigation Count” represents the highest number of mitigations recorded by any single participant in the group. “Low Mitigation Count” represents the lowest number of mitigations recorded by any single participant in the group. “% of Users Who Found the Same” represents the percentage of the group member who agreed on the number of mitigations listed in dictionary. CAPEC represents the group who utilized the

CAPEC dictionary and TAD represents the group who utilized our textual attack descriptions.

Table 55. Results of Usability and Consistency Study for Attack Pattern 43.

	CAPEC	TAD
Average Time to Complete Task	61.4	13
Average Mitigation Count	3.2	3
High Mitigation Count	5	3
Low Mitigation Count	2	3
% of Users Who Found the Same	60	100

Table 56. Results of Usability and Consistency Study for Attack Pattern 67.

	CAPEC	TAD
Average Time to Complete Task	57.4	15.4
Average Mitigation Count	1.4	1
High Mitigation Count	2	1
Low Mitigation Count	1	1
% of Users Who Found the Same	60	100

Table 57. Results of Usability and Consistency Study for Attack Pattern 34.

	CAPEC	TAD
Average Time to Complete Task	41.8	13.8
Average Mitigation Count	1.2	1
High Mitigation Count	2	1
Low Mitigation Count	1	1
% of Users Who Found the Same	80	100

Table 58. Results of Usability and Consistency Study for Attack Pattern 86.

	CAPEC	TAD
Average Time to Complete Task	50	19.6
Average Mitigation Count	8.6	9
High Mitigation Count	9	9
Low Mitigation Count	7	9
% of Users Who Found the Same	80	100

Table 59. Results of Usability and Consistency Study for Attack Pattern 9

	CAPEC	TAD
Average Time to Complete Task	45.2	9.6
Average Mitigation Count	6.4	7
High Mitigation Count	7	7
Low Mitigation Count	4	7
% of Users Who Found the Same	80	100

Table 60. Results of Usability and Consistency Study for Attack Pattern 6

	CAPEC	TAD
Average Time to Complete Task	29.2	9.2
Average Mitigation Count	2.8	3
High Mitigation Count	3	3
Low Mitigation Count	2	3
% of Users Who Found the Same	80	100

Participants who used our dictionary were able to complete the assigned tasks quicker. Furthermore, participants who utilized our dictionary were more consistent in identifying the number of prescribed mitigations. Throughout each of the six exercises, participants using our approach found the same number of mitigations 100% of the time.

In zero of the six exercises did subjects using the original CAPEC dictionary agree 100% of the time and they matched what was documented.

When asked the question “In the process of attempting to locate specific information about specific attack patterns, did you think there was too much data, not enough data, or just the right amount of data (about each attack pattern)”, CAPEC users averaged a 7.4 out of 10 while participants who used our approach averaged a 4.4 out of 10. These results are introduced in Table 61.

Table 61. Results from the Amount of Data Question.

Participant	CAPEC	TAD
1	9	
2	9	
3	8	
4	3	
5	8	
6		5
7		3
8		5
9		5
10		4
Average	7.4	4.4

As introduced earlier a score of 5 meant that the participant felt there was an appropriate amount of data presented in the given dictionary. Scores above 5 indicated that the user felt there was too much information presented in the dictionary while scores below 5 indicated that there was too little information presented in the dictionary. The farther away from 5 (either higher or lower) means a stronger indicator towards too much / too little information.

The results from Table 62 show that our When asked the question, "Concerning usability (structure, organization, ease of use, format, and ability to locate information quickly), how likely are you to use this dictionary again?", CAPEC users averaged a 4.2 out of 10 while participants who used our dictionary averaged a 9.2 out of 10. These results are introduced in Table 62.

Table 62. Results from the Usability / Format Question.

Participant	CAPEC	TAD
1	6	
2	4	
3	6	
4	2	
5	3	
6		10
7		9
8		9
9		10
10		8
Average	4.2	9.2

As introduced earlier the higher the score the more usable the participant deemed the dictionary to be and the more the likely the participant was to reuse this dictionary again in the future. Scores above 5 indicated that the participant felt the dictionary was useable while scores below 5 indicated that the participants felt the dictionary was not usable.

The results of Process 2 are required to complete the final Process of our approach; the creation of CAPEC-based security metrics. Such metrics will be useful in leveraging the vast quantity of information in the current CAPEC dictionary and provide an applied metric for assisting in security related decisions.

5. CREATING CAPEC-BASED SECURITY METRICS

Our third process creates two security metrics to measure NIST-based mitigation strategies when applied to the CAPEC dictionary. This approach re-organizes the work from chapter 4 into a usable hierarchy that is based on the 11 Parent Threats. Leveraging the hierarchy model introduced in chapter 4, we group the entire attack dictionary by the 17 Parent Mitigations presented in chapter 3. The creation of CAPEC-based security metrics is useful in leveraging the vast quantity of information in the current CAPEC dictionary as well as providing an easy-to-use metric for assisting in security related decisions.

The security metrics are created at two distinct levels. The first metric, Knockout Effect (KOE), is at the individual attack pattern level. The second metric, Parent Mitigation Power (PMP), encompasses all 101 attack patterns viewed as a whole. These metrics assist users in making security decisions when attempting to mitigate a single attack pattern or determining how effective a single mitigation is across multiple attack patterns.

Knock-out Effect (KOE) is a measure of how many Parent Mitigation strategies are needed to fully mitigate a detailed attack pattern. Each of the 101 attack patterns has a KOE calculated and stored as part of the graphical hierarchy and the textual attack description. The addition of KOE aids in the consistency and usability of CAPEC by allowing users to quickly determine the number of Parent Mitigations required to fully mitigate a given attack pattern.

Parent Mitigation Power (PMP) is a measure of the total number of unique attack patterns that were partially mitigated by an individual Parent Mitigation strategy and the total number of Child Mitigation strategies that can be traced to the Parent Mitigation. It is important to note that KOE for each of the 101 attack patterns must be completed before the PMP can be computed. We continue our case study to illustrate our approach to leveraging these metrics by including 1 attack pattern from each of the 11 Parent Threats. Process 3 builds on the work outlined in Process 1 and Process 2 and is introduced in Figure 21 where KOE is calculated before PMP. Once these two metrics are calculated our entire approach is complete. The values calculated for KOE and PMP will not change regardless of the chosen system implementation.

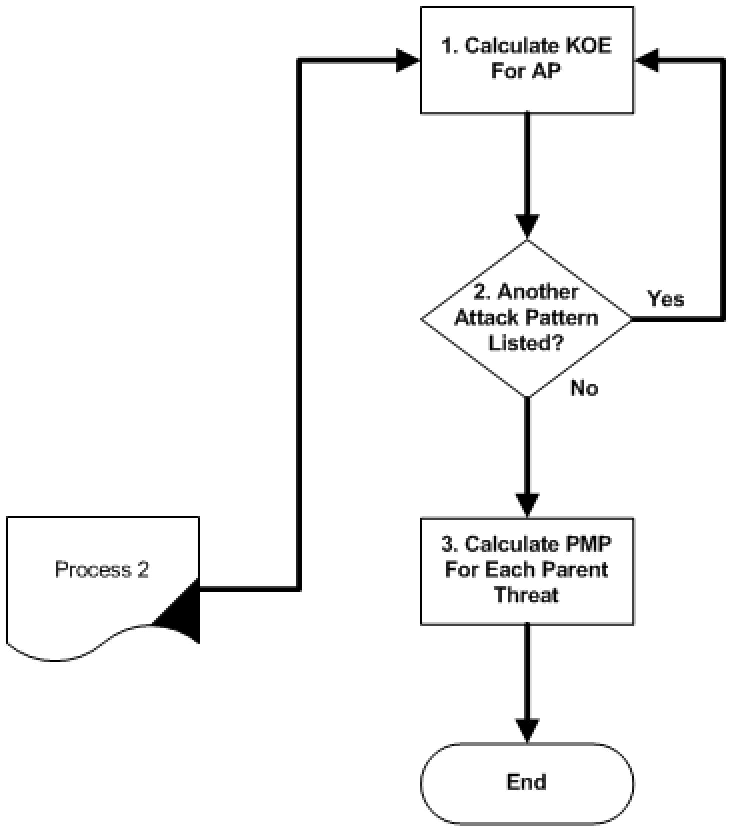

1. Calculate KOE For AP
2. Another Attack Pattern Listed?
Yes
No
3. Calculate PMP For Each Parent Threat
End
Process 2

Figure 21. Individual Steps to Complete Process 3.

5.1. Knock-Out Effect (KOE)

Process 3 begins by calculating the Knock-out Effect for the current attack pattern. This is a numeric value created by adding the total number of Parent Mitigations which were previously abstracted for the attack pattern in Process 2. The KOE value is recorded in both the graphical hierarchy as well as the textual attack description for each attack pattern. Step 1 of Process 3 is repeated for any remaining attack patterns. The detailed actions needed to create the KOE are listed below.

1. Open the completed table for the given attack pattern which was updated in Process 2 (Table 30). Insert a KOE column to the right of the Parent Mitigation column. The new KOE column will become the last column in the table.
2. Count the total number of entries in the Parent Mitigation column for the attack pattern. Enter this value in the newly created KOE column.
3. Add the KOE to the previously created graphical hierarchy by appending the KOE value to the Attack Pattern Name field.
4. Add KOE to the textual attack description by appending the KOE value to the Attack Pattern Name field.
5. Repeat steps 2-4 for any remaining attack patterns.

After a KOE value for all 101 attack patterns has been successfully processed we move to step 3 of Process 3. Step 3 introduces a new security metric (PMP) which is based off of the data resulting from the conglomeration of all 101 attack patterns into a hierarchy.

An example of the new table used in step 1 is introduced in Table 55.

Table 63. Sample Table Used to Complete Step 1 of Process 3.

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**

Step 1 of Process 3 adds the KOE value to the previously used attack pattern tables. KOE counts the total number Parent Mitigations necessary to fully mitigate/prevent the attack pattern from harming the implementation. KOE measures Parent Mitigations because they are easily understood by all users and do not provide too much detail that may intimidate users into non-action. The underlying Child Mitigations that are needed to adequately mitigate the attack pattern are readily available for review as part of the hierarchy. KOE is a count of the total number of Parent Mitigations listed. Table 56 introduces the results of steps 1 and 2 Process 3 for attack pattern 3 “Using Leading ‘Ghost’ Character Sequences to Bypass Input Filters”.

Table 64. Addition of KOE Column for Attack Pattern 3 (“Using Leading ‘Ghost’ Character Sequences to Bypass Input Filters”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation**	**KOE**
Resource Manipulation	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	Perform white list, rather than black list, input validation.	AC-3, AC-4, CM-7, SI-9, SI-10	AC, CM, SI	3
		Canonicalize all data prior to validation	SI-9, SI-10		
		Take an iterative approach to input validation (defense in depth)	SI-10		

Table 57 introduces the results of Step 1 and 2 for Process 3 for attack pattern 75 “Manipulating Writable Configuration Files”.

Table 65. Addition of KOE Column for Attack Pattern 75 (“Manipulating Writable Configuration Files”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Exploitation of Privilege / Trust	Manipulating Writable Configuration Files **75**	Design: Enforce principle of least privilege	AC-6	AC	7
		Design: Backup copies of all configuration files	CP-9, CP-10, CM-2	CP, CM	
		Implementation: Integrity monitoring for configuration files	AU-6, CA-7, CM-4, CM-6, SI-4, SI-7	AU, CA, SI	
		Implementation: Enforce audit logging on code and configuration promotion procedures.	AU1, AU2, CM3, CM4, CM5, CM6		
		Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD	AC-5, SC-2, SC-3	SC	

Step 3 adds the KOE value to the graphical hierarchy trees created for each attack pattern. The KOE for each attack pattern is shown as part of the Attack Pattern Name field.

This allows the total number of Parent Mitigations to be known early in the hierarchy without having to manually count the bottom of the hierarchy. The completed graphical hierarchy tree with KOE for attack pattern 3 is introduced in Figure 22.

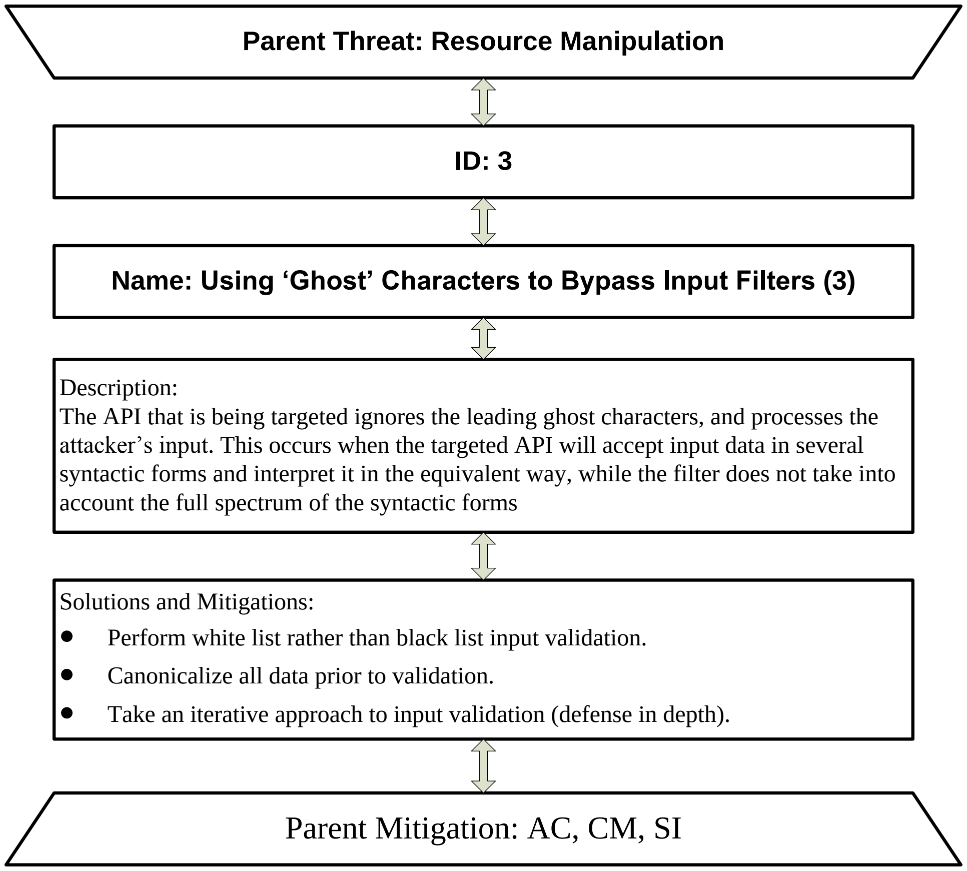

Parent Threat: Resource Manipulation
ID: 3
Name: Using ‘Ghost’ Characters to Bypass Input Filters (3)
Description:
The API that is being targeted ignores the leading ghost characters, and processes the attacker’s input. This occurs when the targeted API will accept input data in several syntactic forms and interpret it in the equivalent way, while the filter does not take into account the full spectrum of the syntactic forms
Solutions and Mitigations:
● Perform white list rather than black list input validation.
● Canonicalize all data prior to validation.
● Take an iterative approach to input validation (defense in depth).
Parent Mitigation: AC, CM, SI

Figure 22. Graphical Hierarchy Tree with KOE for Attack Pattern 3 (“Using ‘Ghost’ Characters to Bypass Input Filters”).

The completed graphical hierarchy with KOE for attack pattern 75 is introduced in Figure 23.

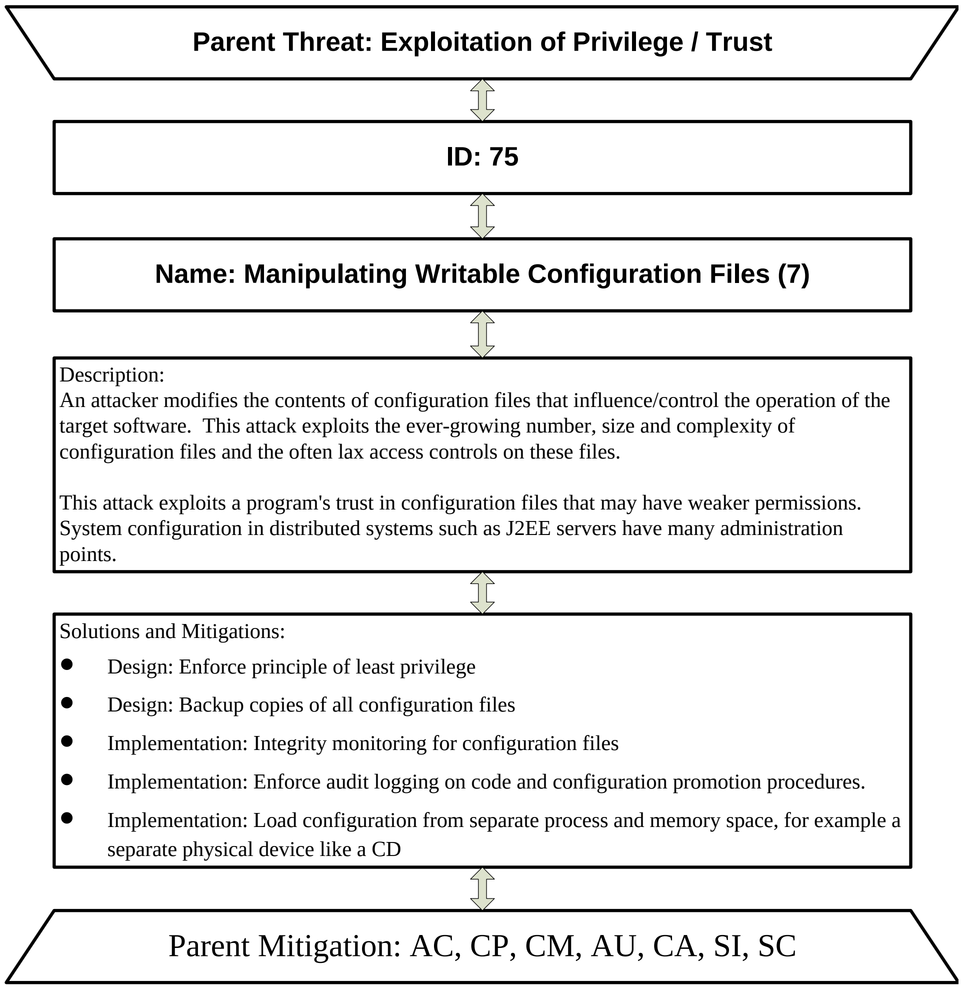

Parent Threat: Exploitation of Privilege / Trust
ID: 75
Name: Manipulating Writable Configuration Files (7)
Description:
An attacker modifies the contents of configuration files that influence/control the operation of the target software. This attack exploits the ever-growing number, size and complexity of configuration files and the often lax access controls on these files.
This attack exploits a program's trust in configuration files that may have weaker permissions. System configuration in distributed systems such as J2EE servers have many administration points.
Solutions and Mitigations:
Design: Enforce principle of least privilege
Design: Backup copies of all configuration files
Implementation: Integrity monitoring for configuration files
Implementation: Enforce audit logging on code and configuration promotion procedures.
Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD
Parent Mitigation: AC, CP, CM, AU, CA, SI, SC

Figure 23. Graphical Hierarchy Tree with KOE for Attack Pattern 75 (“Manipulating Writable Configuration Files”).

Step 4 updates each of the textual attack descriptions with the KOE value. The addition of this step adds significant value by allowing users to quickly ascertain the KOE without having to navigate away from the textual attack descriptions. KOE is appended to the Attack Pattern Name field of each textual attack description. The results of step 4 for attack pattern 3 and 75 are introduced in Tables 58 and 59, respectively.

Table 66. Textual Attack Description with KOE for Attack Pattern 3 ("Using 'Ghost' Characters to Bypass Input Filters").

Attack Pattern ID	3
Attack Pattern Name	Using 'Ghost' Characters to Bypass Input Filters (3)
Description	The API that is being targeted ignores the leading ghost characters, and processes the attacker's input. This occurs when the targeted API will accept input data in several syntactic forms and interpret it in the equivalent way, while the filter does not take into account the full spectrum of the syntactic forms acceptable to the targeted API.
Parent Threat	Resource Manipulation
Solutions and Mitigations	Perform white list rather than black list input validation. Canonicalize all data prior to validation. Take an iterative approach to input validation (defense in depth).
Parent Mitigation	AC, CM, SI

Table 59 introduces the textual attack description for attack pattern 75 "Manipulating Writable Configuration Files".

Table 67. Textual Attack Description with KOE for Attack Pattern 75 ("Manipulating Writable Configuration Files").

Attack Pattern ID	75
Attack Pattern Name	Manipulating Writable Configuration Files (7)
Description	An attacker modifies the contents of configuration files that influence/control the operation of the target software. This attack exploits the ever-growing number, size and complexity of configuration files and the often lax access controls on these files. This attack exploits a program's trust in configuration files that may have weaker permissions. System configuration in distributed systems such as J2EE servers have many administration points. For example, permissions may be set on the administrative GUI, the configuration file for the server as a whole, configuration files for specific domains and applications, special jar and other class files used to load resources at runtime, and even policy specific in .war and .ear files. A mistake in permissions setting in either the file acl or the content is an opening an attacker can use to elevate privilege.
Parent Threat	Exploitation of Privilege / Trust
Solutions and Mitigations	Design: Enforce principle of least privilege Design: Backup copies of all configuration files Implementation: Integrity monitoring for configuration files Implementation: Enforce audit logging on code and configuration promotion procedures. Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD
Parent Mitigation	AC, CP, CM, AU, CA, SI, SC

Step 5 requires the repeating of steps 2-4 for any remaining attack patterns.

5.2. Parent Mitigation Power (PMP)

Parent Mitigation Power (PMP) is a two part metric designated in the "x.y" notation. "x" counts the number of unique Attack Patterns that the Parent Mitigation helped mitigate. "y" counts the total number of Child Mitigations that can be traced to back to the

Parent Mitigation. This provides the ability to measure the impact provided by each of the 17 Parent Mitigations. This can useful for answering “what-if” scenarios such as “Which mitigation provides the most ‘bang for the buck’?” and “If I only have ‘x’ number of security dollars to spend, which mitigations should I invest in?”

The PMP security metric mandates a single graphical hierarchy model of all 101 attack patterns. Because of the exhaustive nature of the CAPEC definition of each attack pattern, there is severe fan-out as Child Mitigations and Parent Mitigations are listed. Once all the graphical hierarchies are compiled into a single view, it is obvious that there is an abundance of mapped Child Mitigations (with their NIST details) and Parent Mitigations.

The hierarchy model introduced in chapter 4 (Figure 9) can also be expanded to incorporate the entire CAPEC dictionary into a single hierarchy structure. This process allows us to leverage the vast mappings created in earlier steps by providing a “forest” view of all 101 CAPEC attack patterns. This forest view is the basis for our PMP metric. When the hierarchy model is used to view individual attack patterns, the model again uses a fan-in-fan-out approach. Our forest view also utilizes this approach by providing abstract details at the top and bottom of the model while providing specific attack pattern information in the middle. The forest view is useful for examining relationship between attack patterns as a whole. Figure 24 introduces the forest view hierarchy along with the maximum number of entries for each level. Section 5.3 will show the completed results of this table for the 11 attack patterns used in our case study.

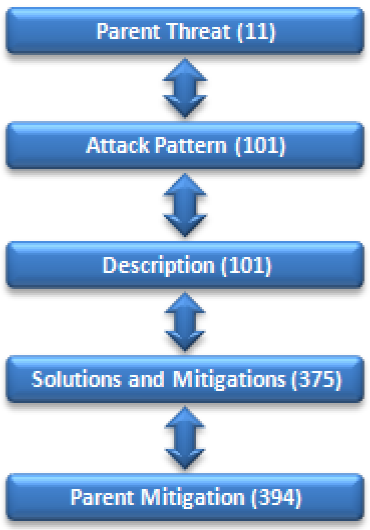


Figure 24. Attack Pattern Forest View with Maximum Number of Entries for Each Level.

For readability purposes we modify Figure 24 to include an additional level. This is accomplished by adding a new row at the bottom to list each of the 17 Parent Mitigations once. This allows us to tie each of the Parent Mitigations into a single Parent Mitigation instance. This value will serve as the “x” value for our PMP metric. This process also provides a location to record the PMP values and fits in with the fan-in-fan-out approach we have made use of throughout our research. Figure 25 introduces the new hierarchy with PMP level included, and the maximum number of entries for each field.

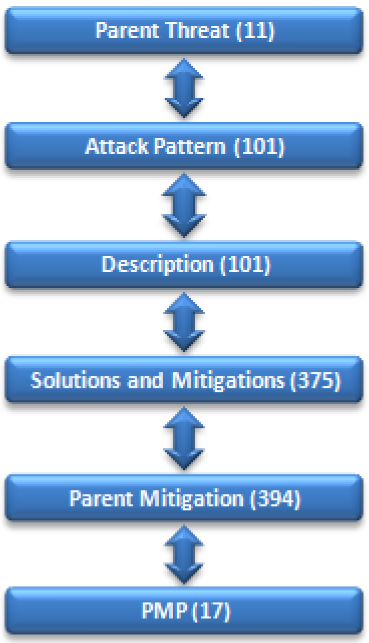


Figure 25. Forest View Hierarchy with Additional PMP and Maximum Number of Entries for Each Level.

Using a model of all 101 attack patterns as outlined in Figure 25 allows for the review of each attack pattern from both a top-down or bottom-up perspective. From the top-down perspective, the Parent Threat (such as “Spoofing”) shows all of the attack patterns derived from it. Similarly, the bottom-up perspective shows all of the attack patterns that a Parent Mitigation (such as “Auditing and Accountability”) helps to prevent. An abbreviated outline of this model is introduced in Figure 26.

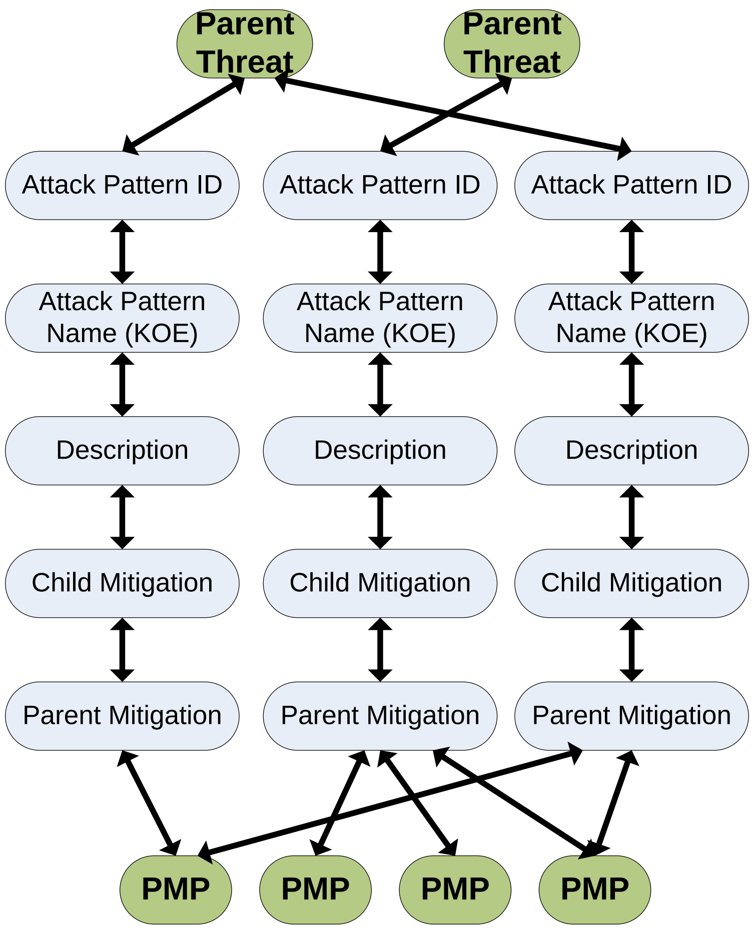


Figure 26. Sample of the Forest View Including Multiple Hierarchies Funneling From Parent Threat to PMP.

Adding PMP to the bottom of our hierarchy groups all of the individual Parent Mitigations into no more than 11 Parent Threats at the top and no more than 17 Parent Mitigations at the bottom. Adding a single instance of Parent Mitigation at the bottom of our figure rids the hierarchies of repeating Parent Mitigations and summarizes what mitigation strategies are needed to protect the implementation.

After all of the attack patterns have been added to the forest view presented in Figure 26, we connect each of the individually listed Parent Mitigations to a single instance of the newly added PMP level. This is completed by connecting every attack pattern Parent Mitigation into a single instance of Parent Mitigation at the bottom of our hierarchy. Upon completion of this work we are ready to calculate PMP. The detailed steps required to calculate PMP are listed below.

1. Calculate the PMP by recording the number of attack patterns each Parent Mitigation traces back to. Record this value in the “x” position of the metric.
2. Utilize a “.” To separate the two-part metric.
3. Calculate the “y” value by adding the total number of entries (across all 101 attack patterns) found in the NIST Child Mitigation(s) column, which relates to the Parent Mitigation. Record this value in the “y” position of the PMP metric.
4. Record the Parent Mitigation and its corresponding PMP in a new table for readability.

Step 1 of PMP creation calculates the initial “x” PMP value by adding the total number of attack patterns each Parent Mitigation helps to mitigate. This is the total number of attack patterns which can be partially or fully mitigated by the given Parent Mitigation. This process can be accomplished by manually counting the total number of times each Parent Mitigation is used in the Parent Mitigation Row or by counting the total number of arrows coming out of the PMP row for each Parent Mitigation. This value is recorded in place of the “x” for PMP.

We calculate the “y” value by adding the total number of times that a related NIST Child Mitigation appears across all 101 attack Patterns. This information is found by reviewing the NIST Child Mitigation column for each attack pattern, as shown in Table 53. Both the “x” and “y” values (along with the Parent Mitigation name) are recorded in the

PMP level of the hierarchy introduced in Figure 26. Complete results of these steps will be shown in section 5.3.

The final step in Process 3 is to review the PMP values and present them in a two column table for ease of use and readability. This table will allow users to quickly review PMP without having to review the graphical hierarchies or the textual attack descriptions. An outline of this table is introduced in table 60.

Table 68. Tabular Format for Presenting PMP Results.

Parent Mitigation	**PMP**

5.3. Case Study Results

Table 61 introduces the results of Steps 1 and 2 for Process 3 for attack pattern 87 where the KOE column and values are added to the right side of to the individual attack pattern tables.

Table 69. Addition of KOE Column for Attack Pattern 87 (“Forceful Browsing”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Abuse of Functionality	Forceful Browsing: **87**	Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.	AC17, IA2, IA3, MA4, SC8, SC23, SI10	AC, IA, MA, SC, SI	9
		Forceful browsing can also be made difficult to a large extent by not hard coding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.	SC18, AT3, CA2, CA4, PL2, SA3, SA8, SA10	AT, CA, PL,SA	

Table 62 introduces the results of Step 1 and 2 for Process 3 for attack pattern 94

Table 70. Addition of KOE Column for Attack Pattern 94 (“Man in the Middle”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Spoofing	Man the Middle: **94**	Get your Public Key signed by a Certificate Authority	CA4, IA5, IA7, SC13, SC17	CA, IA, SC	5
		Encrypt your communication using cryptography (SSL,...)	AC3, AC4, SC7, AC17, IA7, SC8, SC9, SC12, SC13, SI7	AC ,SI	
		Use Strong mutual authentication to always fully authenticate both ends of any communications channel.	AC17, IA1, IA2, IA3, IA4, IA5, SC8, SC11, SC23, SI10		
		Exchange public keys using a secure channel	SC17, SC12, SC13		

Table 63 introduces the results of Step 1 and 2 for Process 3 for attack pattern 55.

Table 71. Addition of KOE Column for Attack Pattern 55 (“Rainbow Table Password Cracking”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Probabilistic Techniques	Rainbow Table Pswd Cracking: **55**	Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.	SI7, SC13, IA5	SI, SC, IA	3

Table 64 introduces the results of Step 1 and 2 for Process 3 for attack pattern 60.

Table 72. Addition of KOE Column for Attack Pattern 60 (“Resuing Session ID’s”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Exploitation of Authorization	Reusing Session ID's: **60**	Always invalidate a session ID after the user logout.	AC3, IA5, SC10, SC23, IA4	AC, IA, SC	4
		Setup a session time out for the session IDs.	AC11, AC12, SC23, IA4		
		Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.	AC4, IA2, IA3, IA7, SC8, SC9, SC11, SC12, SC13, SC16, SC17, SC20, SC21, SC22, SC23	SA	
		Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.	SC9, SC4, SC14, SC16, SA8		
		Encrypt the session data associated with the session ID.	AC3, SC4, SC7, SC23		
		Use multifactor authentication.	IA2		

Table 65 introduces the results of Step 1 and 2 for Process 3 for attack pattern 82.

Table 73. Addition of KOE Column for Attack Pattern 82 (“XML Denial of Service”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Resource Depletion	XMLDoS (XDoS): **82**	Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.	AC4, SI9, SI10, AC3, CM6	AC, SI, CM	6
		Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers	AC4, CA3, SC6, SI4, CP10, SC5, SC22	CA,SC, CP	
		Implementation: Check size of XML message before parsing	SI7, SI9, SI10		

Table 66 introduces the results of Step 1 and 2 for Process 2 for attack pattern 65.

Table 74. Addition of KOE Column for Attack Pattern 65 (“Passive Sniffing”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Data Leakage Attacks	Passive Sniffing : **65**	Do not store secrets in client code	CM6, PE19, RA3, SA8, PL4	CM, PE, RA, SA, PL	10
		Use Well-Known Cryptography Appropriately and Correctly	AC3, AC17, IA7, MA4, SC8, SC9, SC12, SC13	AC, IA, MA, SC	
		Use Authentication Mechanisms, Where Appropriate, Correctly	IA2, IA7, SC23, SI10	SI	

Table 67 introduces the results of Step 1 and 2 for Process 3 for attack pattern 101.

Table 75. Addition of KOE Column for Attack Pattern 101 (“Server Side Includes”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Injection	Server Side Includes (SSI): **101**	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them	CM1, CM6, CM7, SI6, SC3, AC6	CM, SI, SC, AC	4
		All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive	SI7, SI9, SI10		
		Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead	AC6		

Table 68 introduces the results of Step 1 and 2 for Process 3 for attack pattern 10.

Table 76. Addition of KOE Column for Attack Pattern 10 ("Buffer Overflow via Environment Variables").

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Data Structure Attacks	Buffer Overflow via Environment Variables: **10**	Do not expose environment variable to the user.	AC6, CM6, RA3, RA5, SA10, SA11, SC4, SI10	AC, CM, RA, SA, SC, SI	9
		Do not use untrusted data in your environment variables.	AC3, CM6, IA2, SC23, SI17, SI19, SI10	IA,	
		Use a language or compiler that performs automatic bounds checking	SA8, PL2	PL	
		There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.	MA3, PL6 RA5, SA10, SA11, SI2, SI4	MA	

Table 69 introduces the results of Step 1 and 2 for Process 3 for attack pattern 29.

Table 77. Addition of KOE Column for Attack Pattern 29 (“Race Conditions Time of Check and Time of Use”).

Parent Threat	**Attack Pattern**	**Solutions and Mitigations**	**NIST Child Mitigation(s)**	**Parent Mitigation(s)**	**KOE**
Time and State Attacks	Race Conditions (TOCTOU): **29**	Use safe libraries to access resources such as files.	SI7, SC18,	SI, SC	6
		Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.	AT2, AC3, IA2	AT, AC, IA	
		Use synchronization to control the flow of execution.	SC3, AC4		
		Use static analysis tools to find race conditions.	SA11,SI10		
		Pay attention to concurrency problems related to the access of resources.	SA8, SC4	SA	

Step 3 of Process 3 appends KOE to the graphical hierarchy trees. Figure 27 introduces the KOE metric included as part of the “Name” field in the graphical hierarchy trees.

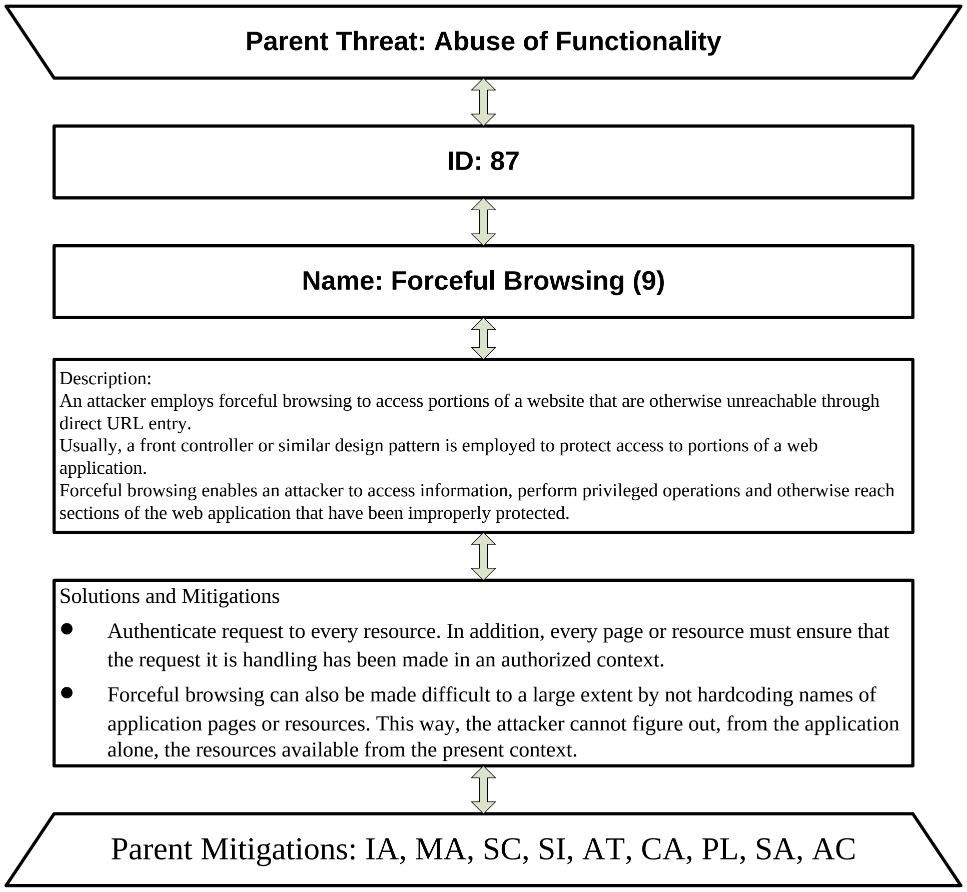

Parent Threat: Abuse of Functionality
ID: 87
Name: Forceful Browsing (9)
Description:
An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry.
Usually, a front controller or similar design pattern is employed to protect access to portions of a web application.
Forceful browsing enables an attacker to access information, perform privileged operations and otherwise reach sections of the web application that have been improperly protected.
Solutions and Mitigations
Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.
Forceful browsing can also be made difficult to a large extent by not hardcoding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.
Parent Mitigations: IA, MA, SC, SI, AT, CA, PL, SA, AC

Figure 27. Graphical Hierarchy Tree with KOE for Attack Pattern 87 (“Forceful Browsing”).

The completed graphical hierarchy for attack patterns 94 is introduced in Figure 28.

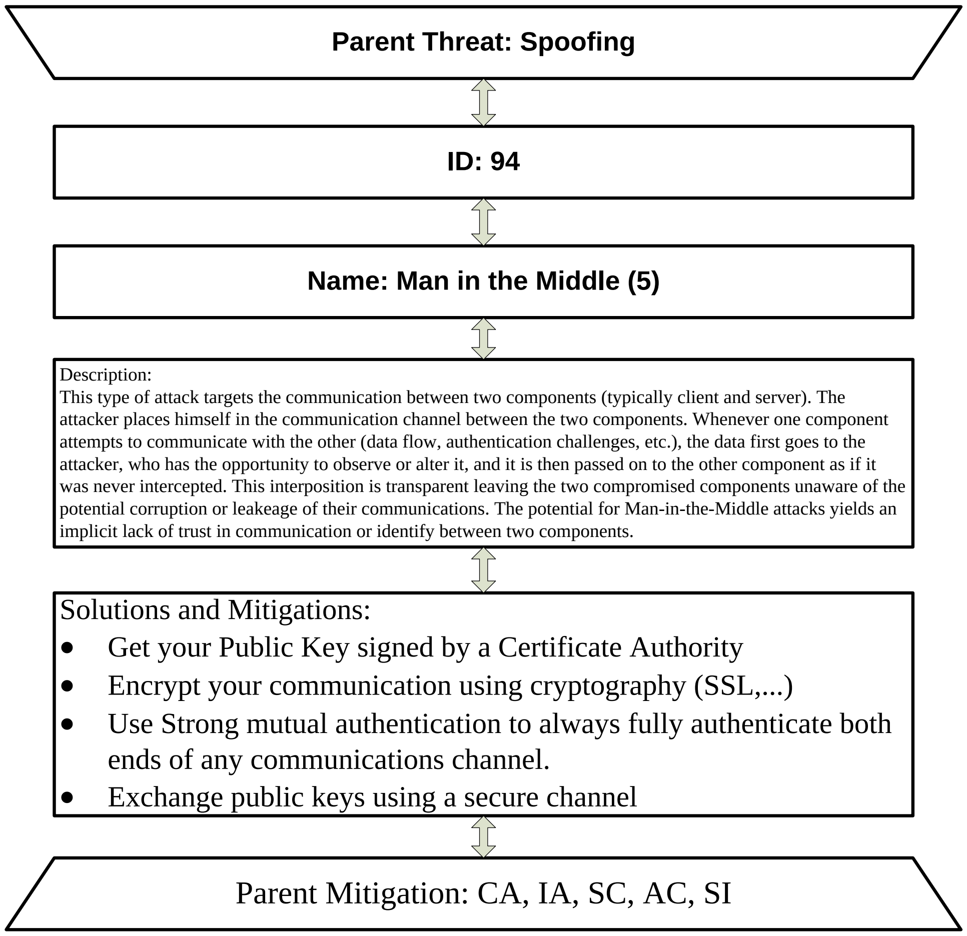

Parent Threat: Spoofing
ID: 94
Name: Man in the Middle (5)
Description:
This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.
Solutions and Mitigations:
• Get your Public Key signed by a Certificate Authority
• Encrypt your communication using cryptography (SSL,...)
• Use Strong mutual authentication to always fully authenticate both ends of any communications channel.
• Exchange public keys using a secure channel
Parent Mitigation: CA, IA, SC, AC, SI

Figure 28. Graphical Hierarchy Tree with KOE for Attack Pattern 94 (“Man in the Middle”).

The completed graphical hierarchy tree for attack patterns 55 is introduced in Figure 29.

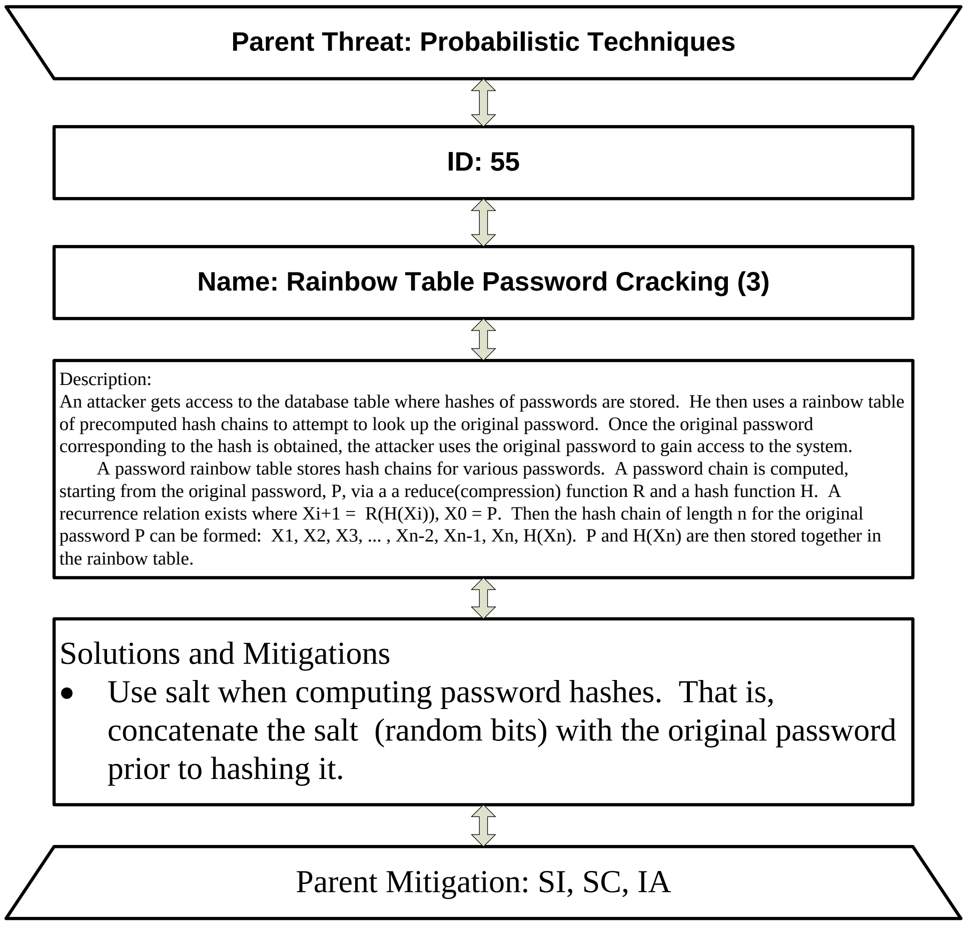
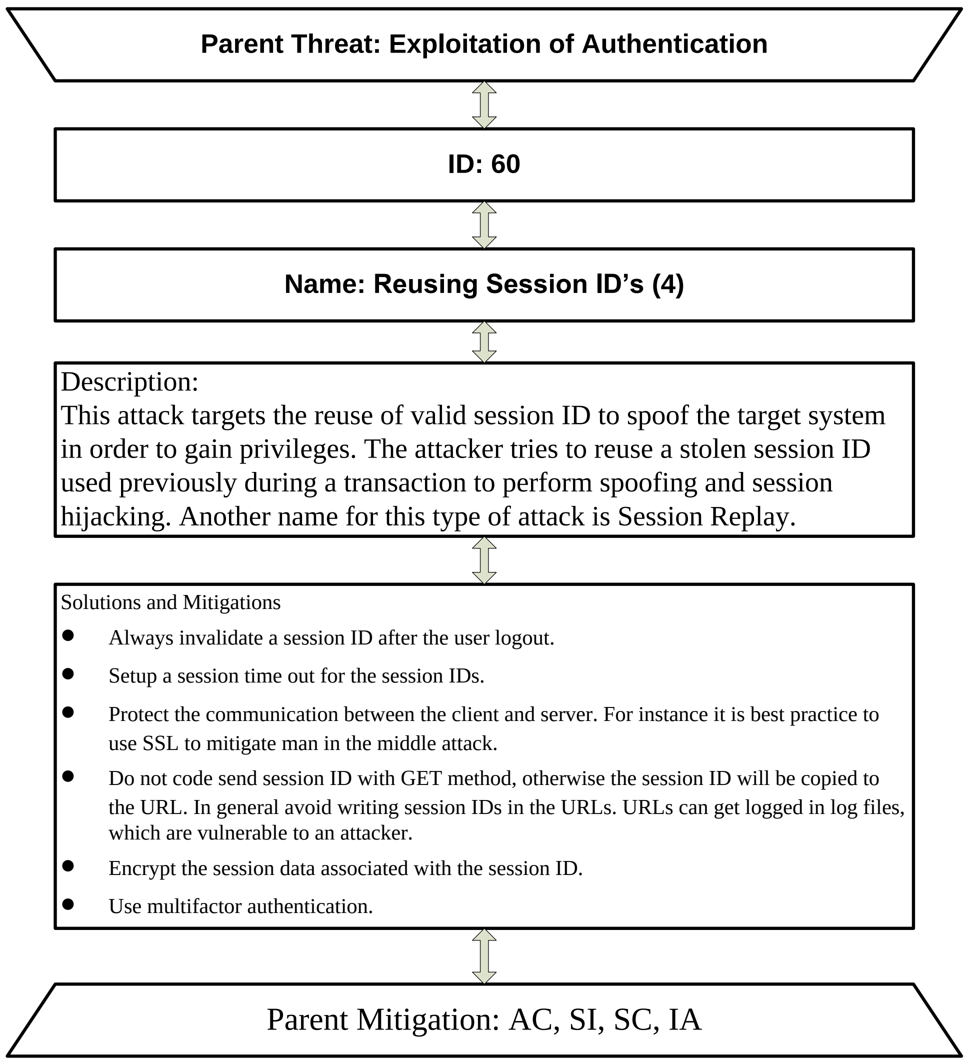

Parent Threat: Probabilistic Techniques
ID: 55
Name: Rainbow Table Password Cracking (3)
Description:
An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system.
A password rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table.
Solutions and Mitigations
• Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.
Parent Mitigation: SI, SC, IA

Figure 29. Graphical Hierarchy Tree with KOE for Attack Pattern 94 (“Rainbow Table Password Cracking”).

The completed graphical hierarchy tree for attack patterns 60 is introduced in Figure 30.


Parent Threat: Exploitation of Authentication

ID: 60

Name: Reusing Session ID’s (4)

Description:
This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.

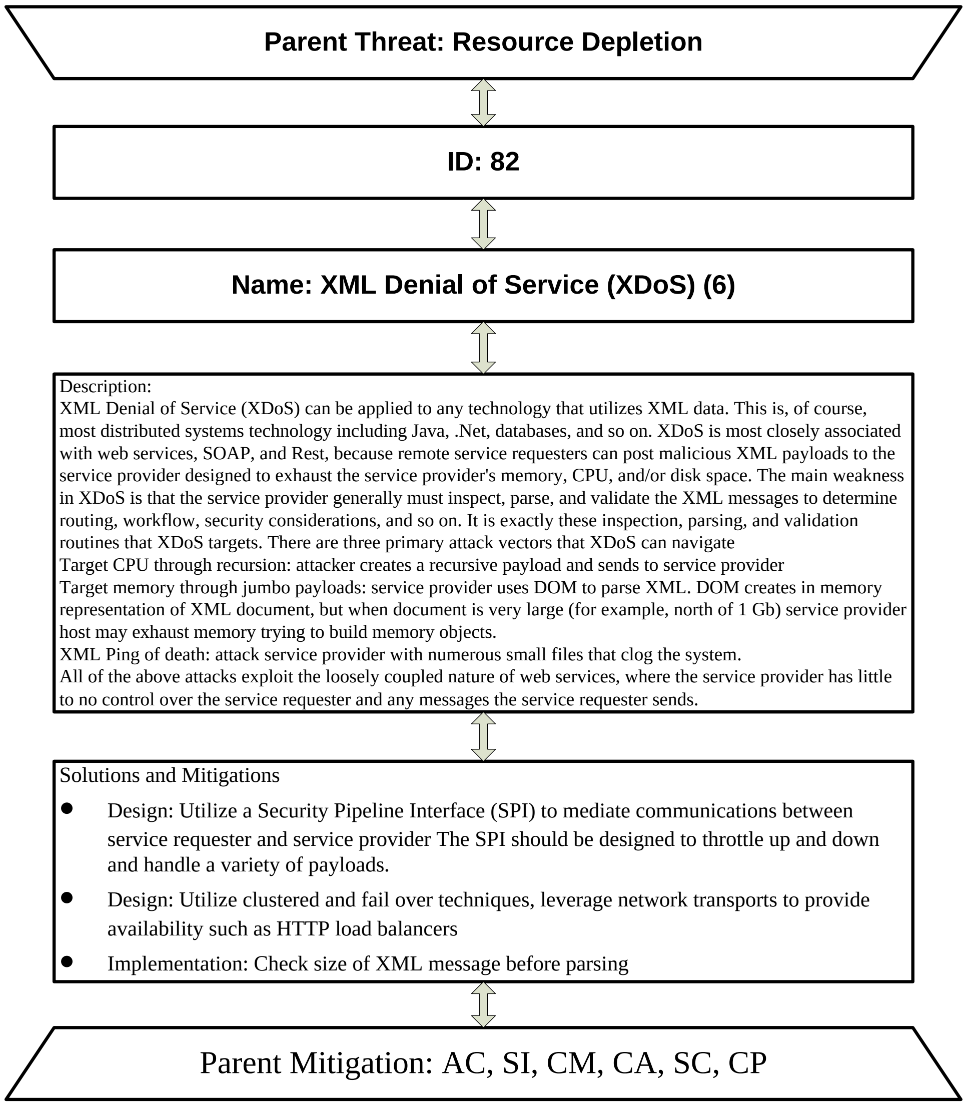
Solutions and Mitigations

- Always invalidate a session ID after the user logout.
- Setup a session time out for the session IDs.
- Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.
- Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.
- Encrypt the session data associated with the session ID.
- Use multifactor authentication.

Parent Mitigation: AC, SI, SC, IA

Figure 30. Graphical Hierarchy Tree with KOE for Attack Pattern 60 (“Reusing Session ID’s”).

The completed graphical hierarchy tree for attack patterns 82 is introduced in Figure 31.


Parent Threat: Resource Depletion
ID: 82
Name: XML Denial of Service (XDoS) (6)
Parent Mitigation: AC, SI, CM, CA, SC, CP

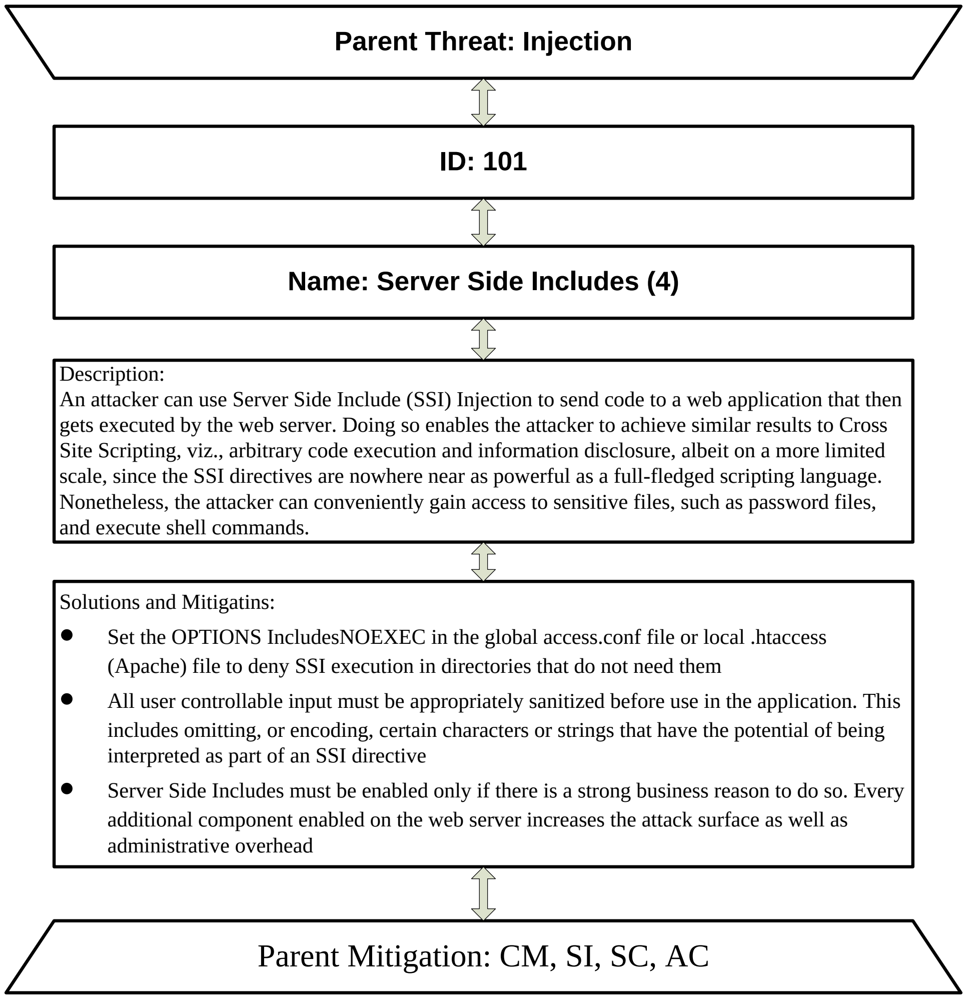
Description:
XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space. The main weakness in XDoS is that the service provider generally must inspect, parse, and validate the XML messages to determine routing, workflow, security considerations, and so on. It is exactly these inspection, parsing, and validation routines that XDoS targets. There are three primary attack vectors that XDoS can navigate
Target CPU through recursion: attacker creates a recursive payload and sends to service provider
Target memory through jumbo payloads: service provider uses DOM to parse XML. DOM creates in memory representation of XML document, but when document is very large (for example, north of 1 Gb) service provider host may exhaust memory trying to build memory objects.
XML Ping of death: attack service provider with numerous small files that clog the system.
All of the above attacks exploit the loosely coupled nature of web services, where the service provider has little to no control over the service requester and any messages the service requester sends.

Solutions and Mitigations

- Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.
- Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers
- Implementation: Check size of XML message before parsing

Figure 31. Graphical Hierarchy Tree with KOE for Attack Pattern 82 (“XML Denial of Service”).

The completed graphical hierarchy tree for attack patterns 101 is introduced in Figure 32.


Parent Threat: Injection
ID: 101
Name: Server Side Includes (4)
Parent Mitigation: CM, SI, SC, AC

Description:
An attacker can use Server Side Include (SSI) Injection to send code to a web application that then gets executed by the web server. Doing so enables the attacker to achieve similar results to Cross Site Scripting, viz., arbitrary code execution and information disclosure, albeit on a more limited scale, since the SSI directives are nowhere near as powerful as a full-fledged scripting language. Nonetheless, the attacker can conveniently gain access to sensitive files, such as password files, and execute shell commands.

Solutions and Mitigatins:

- Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them
- All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive
- Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead

Figure 32. Graphical Hierarchy Tree with KOE for Attack Pattern 101 (“Server Side Includes”).

The completed graphical hierarchy tree for attack patterns 10 is introduced in Figure 33.

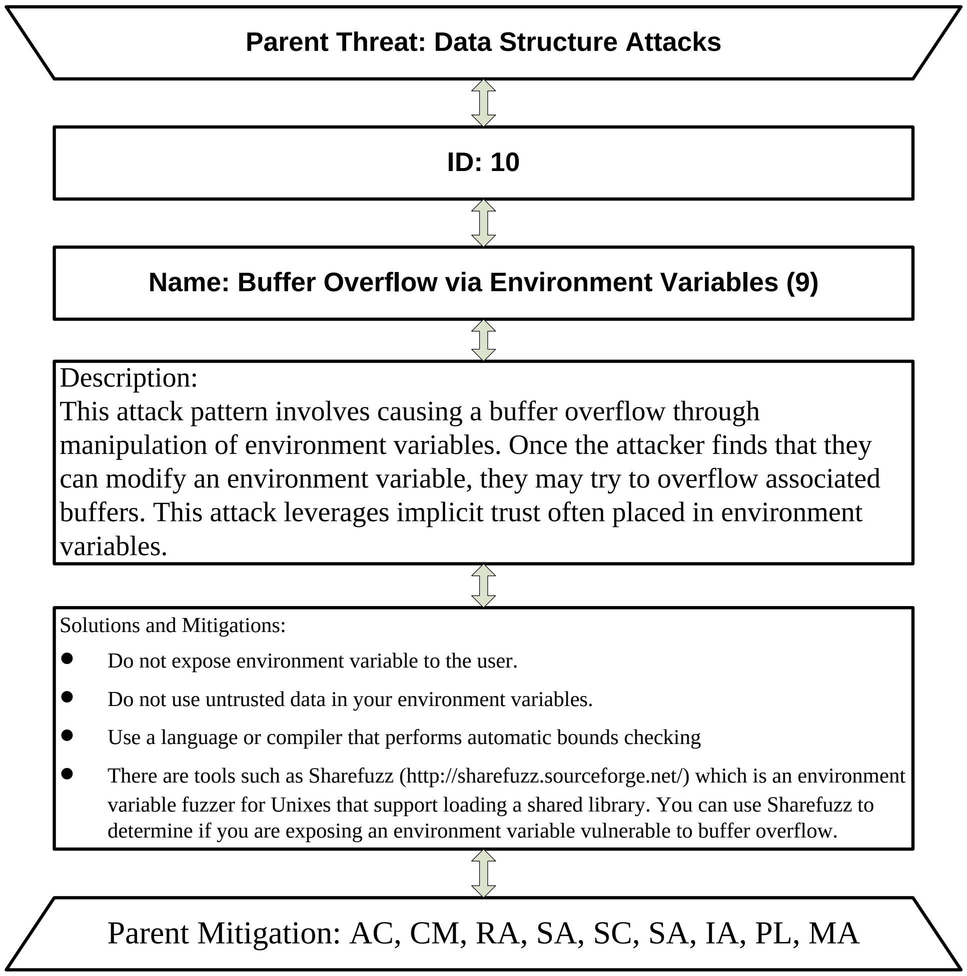

Parent Threat: Data Structure Attacks
ID: 10
Name: Buffer Overflow via Environment Variables (9)
Description:
This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.
Solutions and Mitigations:
Do not expose environment variable to the user.
Do not use untrusted data in your environment variables.
Use a language or compiler that performs automatic bounds checking
There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.
Parent Mitigation: AC, CM, RA, SA, SC, SA, IA, PL, MA

Figure 33. Graphical Hierarchy Tree with KOE for Attack Pattern 10 (“Buffer Overflow via Environment Variables”).

The completed graphical hierarchy tree for attack patterns 65 is introduced in Figure 34.

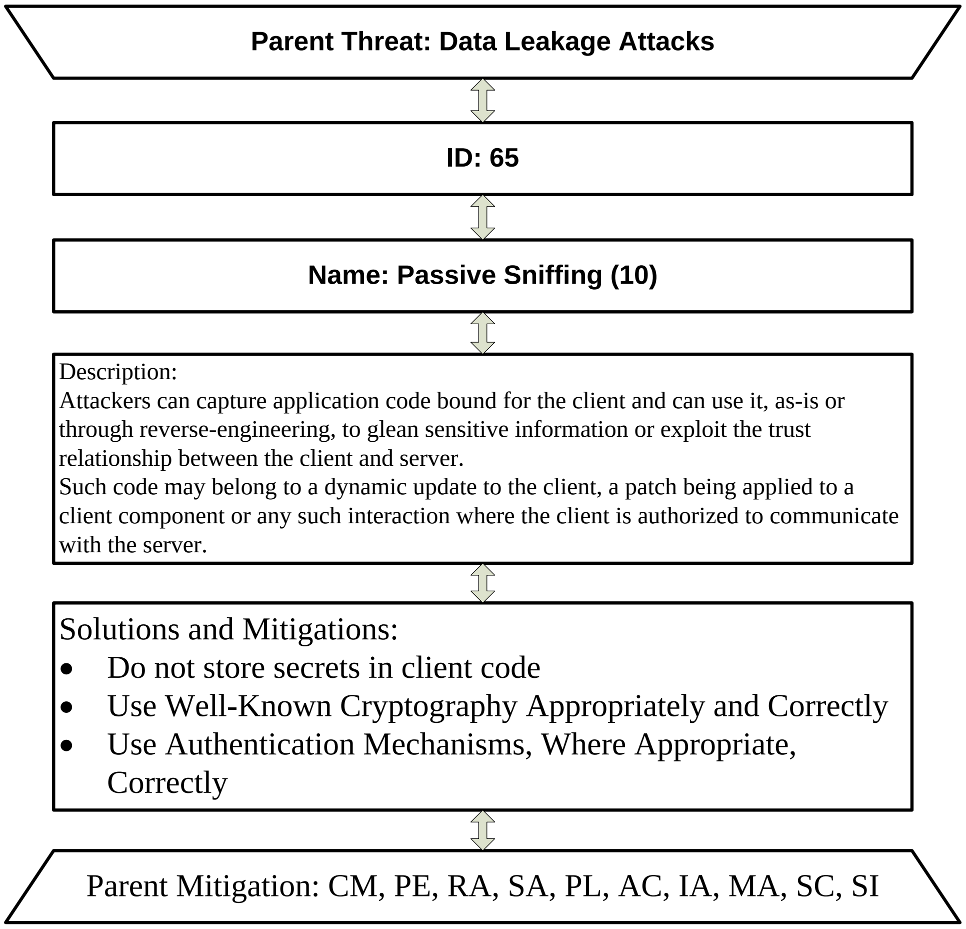

Parent Threat: Data Leakage Attacks
ID: 65
Name: Passive Sniffing (10)
Description:
Attackers can capture application code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server.
Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.
Solutions and Mitigations:
• Do not store secrets in client code
• Use Well-Known Cryptography Appropriately and Correctly
• Use Authentication Mechanisms, Where Appropriate, Correctly
Parent Mitigation: CM, PE, RA, SA, PL, AC, IA, MA, SC, SI

Figure 34. Graphical Hierarchy Tree with KOE for Attack Pattern 65 (“Passive Sniffing”).

The completed graphical hierarchy tree for attack patterns 29 is introduced in Figure 35.

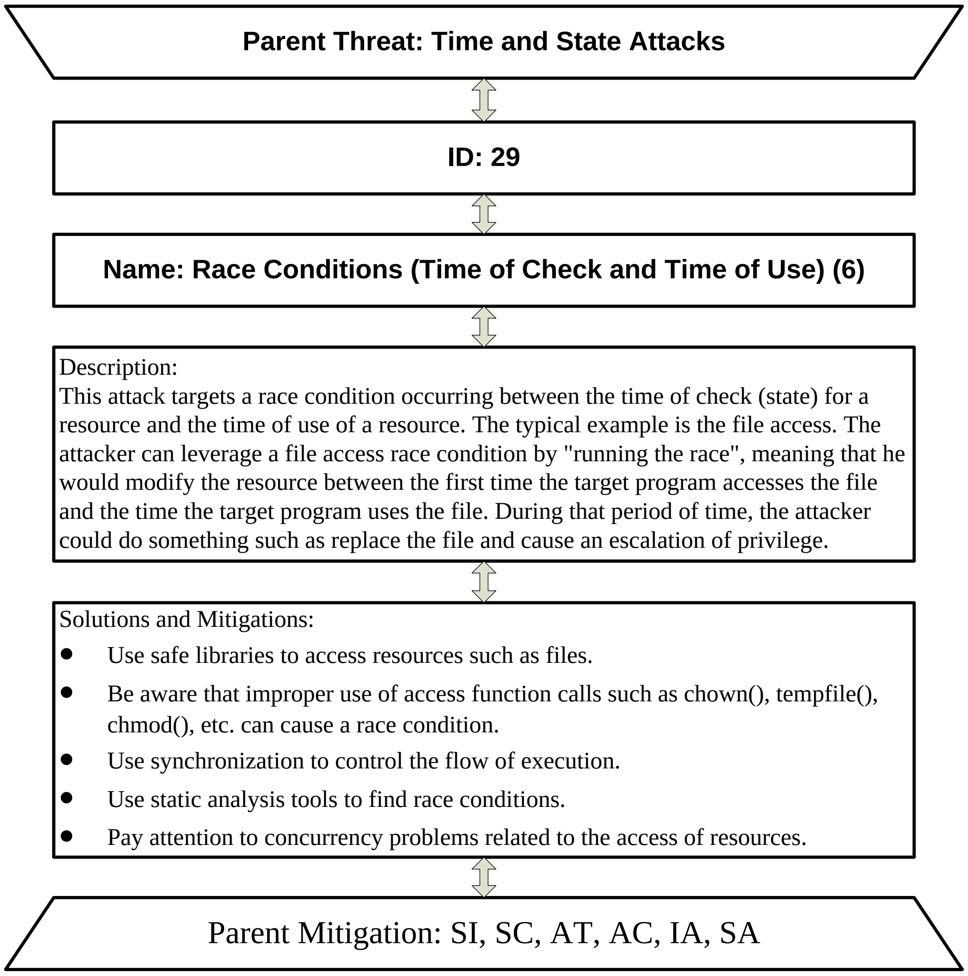

Parent Threat: Time and State Attacks
ID: 29
Name: Race Conditions (Time of Check and Time of Use) (6)
Description:
This attack targets a race condition occurring between the time of check (state) for a resource and the time of use of a resource. The typical example is the file access. The attacker can leverage a file access race condition by "running the race", meaning that he would modify the resource between the first time the target program accesses the file and the time the target program uses the file. During that period of time, the attacker could do something such as replace the file and cause an escalation of privilege.
Solutions and Mitigations:
● Use safe libraries to access resources such as files.
● Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.
● Use synchronization to control the flow of execution.
● Use static analysis tools to find race conditions.
● Pay attention to concurrency problems related to the access of resources.
Parent Mitigation: SI, SC, AT, AC, IA, SA

Figure 35. Graphical Hierarchy Tree with KOE for Attack Pattern 65 (“Race Conditions Time of Check and Time of Use”).

Step 4 of Process 3 appends KOE to the textual attack descriptions for each attack pattern. KOE is show in the textual attack descriptions in order to increase usability. Users who are reviewing textual attack descriptions are not required to look outside of the descriptions for the KOE value. The KOE is included as part of the “Attack Pattern Name” for each of the results below. Table 70 introduces the textual attack description with KOE for attack pattern 87.

Table 78. Textual Attack Description with KOE for Attack Pattern 87 ("Forceful Browsing").

Attack Pattern ID	87
Attack Pattern Name	Forceful Browsing (9)
Description	An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry. Usually, a front controller or similar design pattern is employed to protect access to portions of a web application. Forceful browsing enables an attacker to access information, perform privileged operations and otherwise reach sections of the web application that have been improperly protected.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context. Forceful browsing can also be made difficult to a large extent by not hardcoding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.
Parent Mitigation	IA, MA, SC, SI, AT, CA, PL, SA, AC

Table 71 introduces the textual attack description for attack pattern 94.

Table 79. Textual Attack Description with KOE for Attack Pattern 94 (“Man in the Middle”).

Attack Pattern ID	94
Attack Pattern Name	Man in the Middle (5)
Description	This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.
Parent Threat	Spoofing
Solutions and Mitigations	Get your Public Key signed by a Certificate Authority Encrypt your communication using cryptography (SSL,...) Use Strong mutual authentication to always fully authenticate both ends of any communications channel. Exchange public keys using a secure channel
Parent Mitigation	CA, IA, SC, AC, SI

Table 72 introduces the textual attack description for attack pattern 55.

Table 80. Textual Attack Description with KOE for Attack Pattern 55 ("Rainbow Table Password Cracking").

Attack Pattern ID	55
Attack Pattern Name	Rainbow Table Password Cracking (3)
Description	An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system. A pasword rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.
Parent Mitigation	SI, SC, IA

Table 73 introduces the textual attack description for attack pattern 60.

Table 81. Textual Attack Description with KOE for Attack Pattern 60 (“Reusing Session ID’s”).

Attack Pattern ID	60
Attack Pattern Name	Reusing Session ID’s (4)
Description	This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	Always invalidate a session ID after the user logout. Setup a session time out for the session IDs. Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack. Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker. Encrypt the session data associated with the session ID. Use multifactor authentication.
Parent Mitigation	AC, IA, SC, SA

Table 74 introduces the textual attack description for attack pattern 82.

Table 82. Textual Attack Description for Attack Pattern 82 (“XML Denial of Service”).

Attack Pattern ID	82
Attack Pattern Name	XMLDoS (XDoS) (6)
Description	XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space. The main weakness in XDoS is that the service provider generally must inspect, parse, and validate the XML messages to determine routing, workflow, security considerations, and so on. It is exactly these inspection, parsing, and validation routines that XDoS targets. There are three primary attack vectors that XDoS can navigate Target CPU through recursion: attacker creates a recursive payload and sends to service provider Target memory through jumbo payloads: service provider uses DOM to parse XML. DOM creates in memory representation of XML document, but when document is very large (for example, north of 1 Gb) service provider host may exhaust memory trying to build memory objects. XML Ping of death: attack service provider with numerous small files that clog the system. All of the above attacks exploit the loosely coupled nature of web services, where the service provider has little to no control over the service requester and any messages the service requester sends.
Parent Threat	Resource Depletion
Solutions and Mitigations	Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads. Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers Implementation: Check size of XML message before parsing
Parent Mitigation	AC, SI, CM, CA, SC, CP

Table 75 introduces the textual attack description for attack pattern 101.

Table 83. Textual Attack Description for Attack Pattern 101 ("Server Side Includes").

Attack Pattern ID	101
Attack Pattern Name	Server Side Includes (SSI) (4)
Description	An attacker can use Server Side Include (SSI) Injection to send code to a web application that then gets executed by the web server. Doing so enables the attacker to achieve similar results to Cross Site Scripting, viz., arbitrary code execution and information disclosure, albeit on a more limited scale, since the SSI directives are nowhere near as powerful as a full-fledged scripting language. Nonetheless, the attacker can conveniently gain access to sensitive files, such as password files, and execute shell commands.
Parent Threat	Injection
Solutions and Mitigations	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead
Parent Mitigation	CM, SI, SC, AC

Table 76 introduces the textual attack description for attack pattern 10.

Table 84. Textual Attack Description for Attack Pattern 10 (“Buffer Overflow via Environment Variable”).

Attack Pattern ID	10
Attack Pattern Name	Buffer Overflow via Environment Variables (9)
Description	This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	Do not expose environment variable to the user. Do not use untrusted data in your environment variables. Use a language or compiler that performs automatic bounds checking There are tools such as Sharefuzz (http://sharefuzz.sourceforge.net/) which is an environment variable fuzzer for Unixes that support loading a shared library. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow.
Parent Mitigation	AC, CM, RA, SA, SC, SA, IA, PL, MA

Table 77 introduces the textual attack description for attack pattern 65.

Table 85. Textual Attack Description for Attack Pattern 65 (“Passive Sniffing”).

Attack Pattern ID	65
Attack Pattern Name	Passive Sniffing (10)
Description	Attackers can capture application code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server. Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.
Parent Threat	Data Leakage Attack
Solutions and Mitigations	Do not store secrets in client code Use Well-Known Cryptography Appropriately and Correctly Use Authentication Mechanisms, Where Appropriate, Correctly
Parent Mitigation	CM, PE, RA, SA, PL, AC, IA, MA, SC, SI

Table 78 introduces the textual attack description for attack pattern 29.

Table 86. Textual Attack Description for Attack Pattern 29 (“Race Conditions Time of Check and Time of Use”).

Attack Pattern ID	29
Attack Pattern Name	Race Conditions (Time of Check and Time of Use) (6)
Description	This attack targets a race condition occurring between the time of check (state) for a resource and the time of use of a resource. The typical example is the file access. The attacker can leverage a file access race condition by "running the race", meaning that he would modify the resource between the first time the target program accesses the file and the time the target program uses the file. During that period of time, the attacker could do something such as replace the file and cause an escalation of privilege.
Parent Threat	Time and State Attacks
Solutions and Mitigations	Use safe libraries to access resources such as files. Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition. Use synchronization to control the flow of execution. Use static analysis tools to find race conditions. Pay attention to concurrency problems related to the access of resources.
Parent Mitigation	SI, SC, AT, AC, IA, SA

This concludes the work for calculating and documenting KOE. Before we can calculate the PMP metric we are required to compile each of the individual attack patterns into a forest view as introduced in Figure 36.

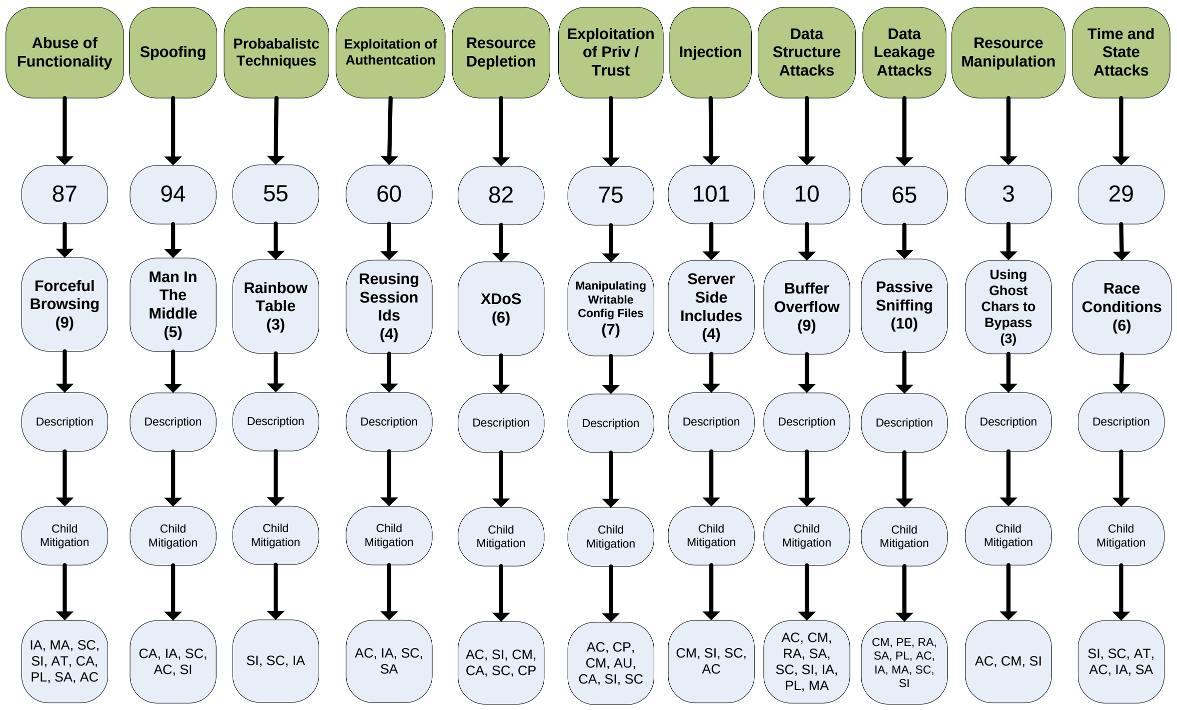


Figure 36. Forest Hierarchy View Including all 11 Attack Patterns from Case Study.

In order to conserve space, the “Description” and “Child Mitigations” are labeled generically. The details for these fields can be found in the textual attack descriptions and the CAPEC Release 1 dictionary.

More work is required before we can calculate PMP. We next add each of the 17 Parent Mitigations at the bottom of Figure 36. This additional level will be used to tie each of the individually listed Parent Mitigations into a single instance. The result of this process is introduced in Figure 37.

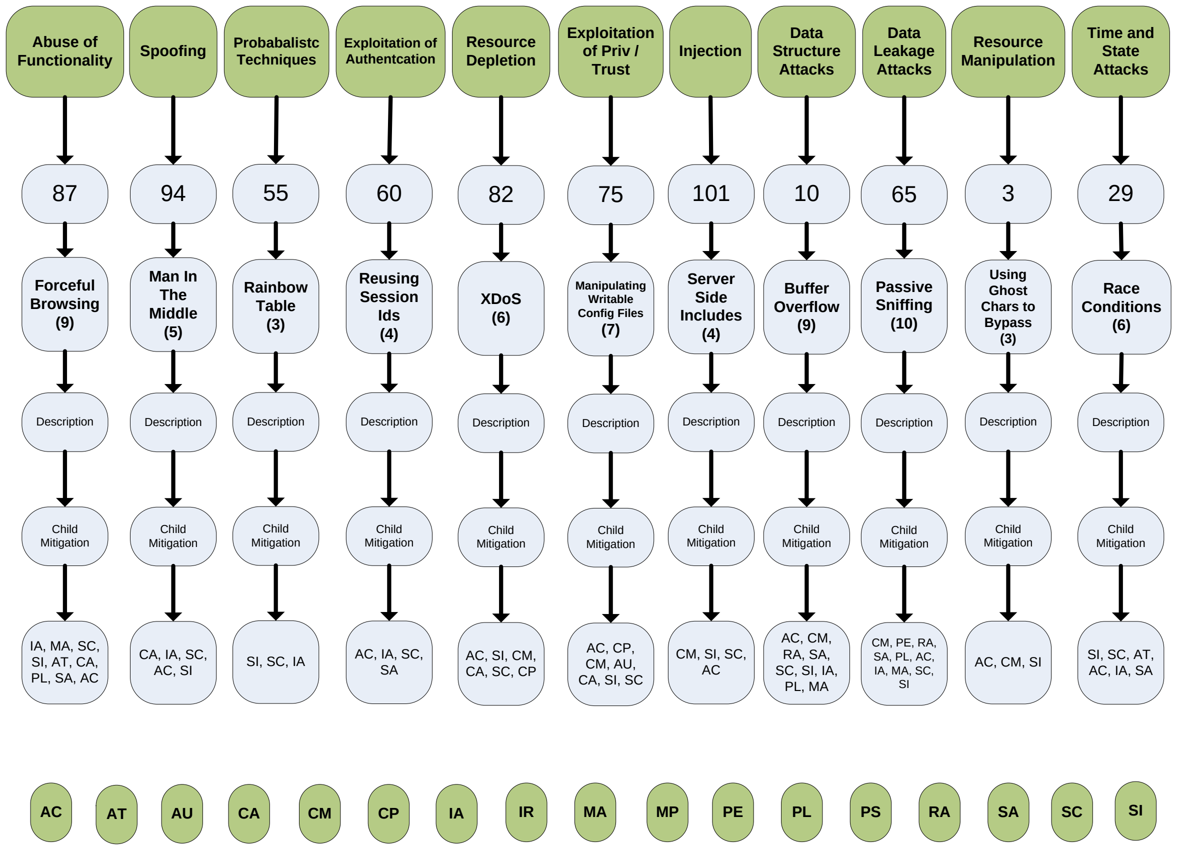


Figure 37. Forest Hierarchy View Including all 11 Attack Patterns from Case Study with 17 Parent Mitigations.

Before we can calculate the PMP metric we must cross reference each of the individually listed Parent Mitigations for each attack pattern back to a single Parent Mitigation (added in Figure 37). Figure 38 introduces the completed process of each Parent Mitigation cross referenced to a single Parent Mitigation.

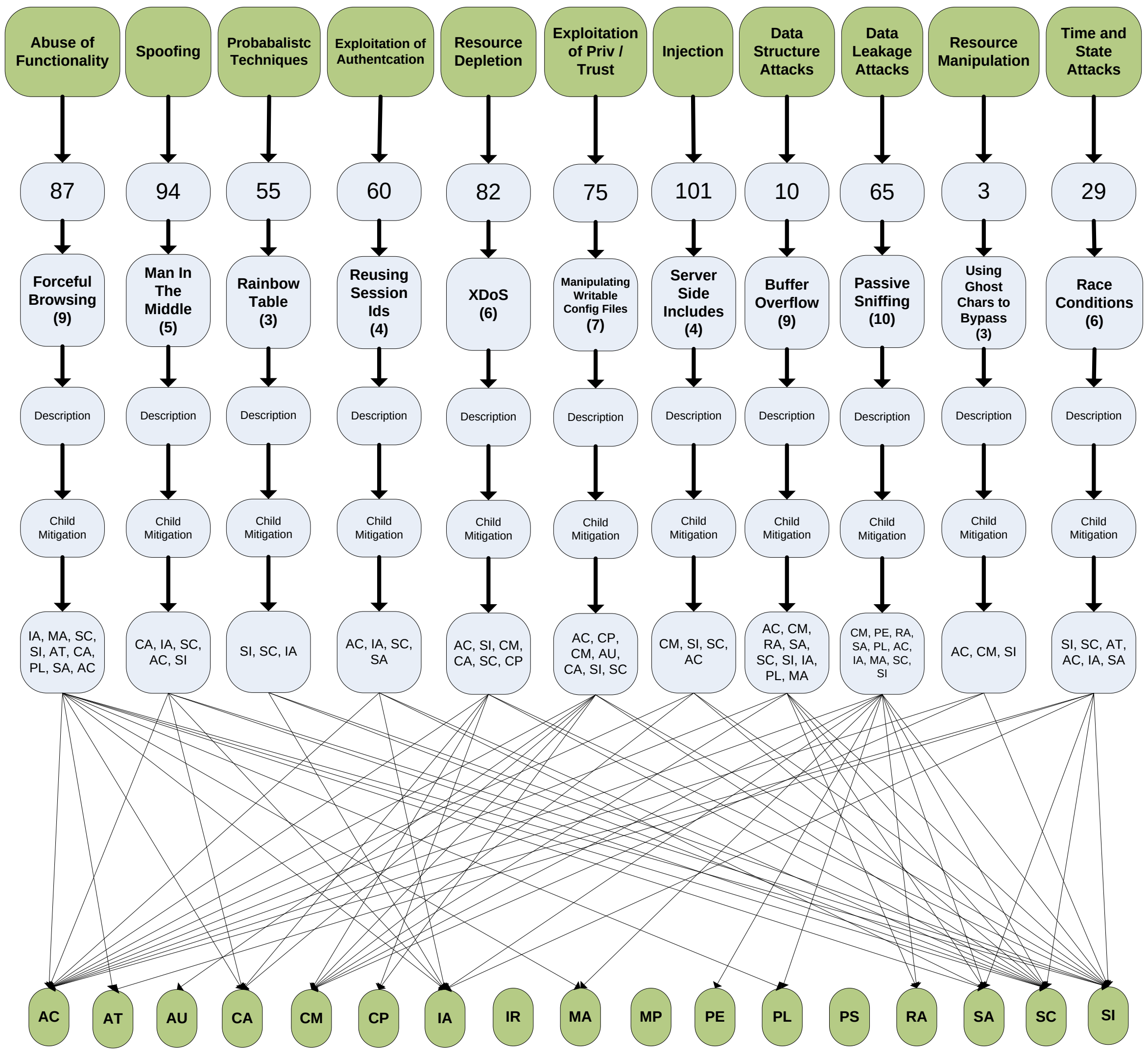


Figure 38. Complete results of Case Study in Forest View.

Figure 38 completes the work necessary to calculate the PMP values. The first step in PMP creation requires that we calculate the PMP “x” value by counting the number of arrows entering each PMP field. Figure 39 zooms in on the first 4 entries of Figure 38 in order to clarify the counting process.

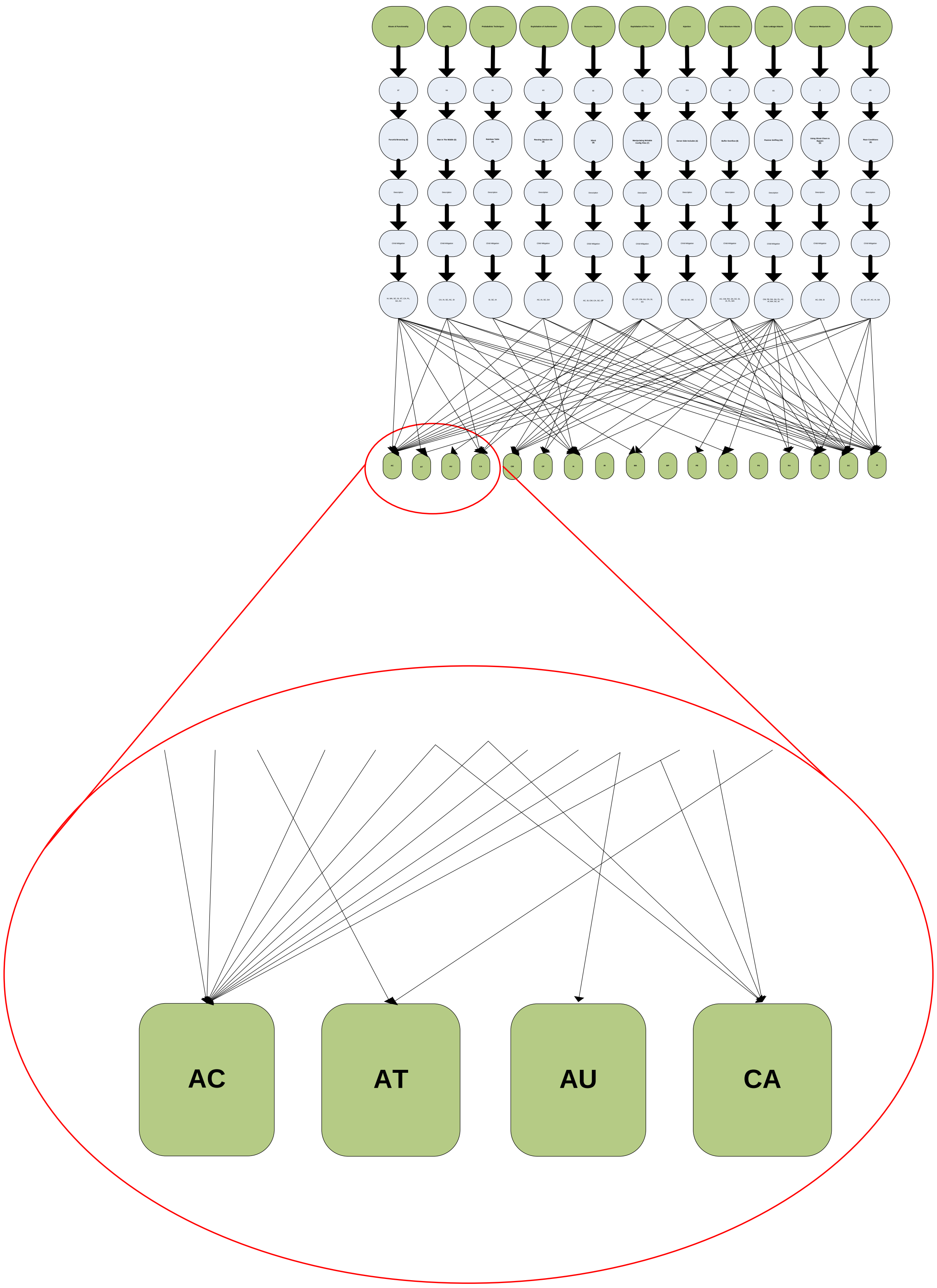


Figure 39. Zoomed in View of Forest View for Purpose of Calculating PMP “x” Value.

Figure 39 shows ten arrows entering the AC Parent Mitigation. This Value is recorded as the “x” value in PMP. Figure 39 shows two arrows entering the AT box, one entering AU, and four entering CA. Each of these values are recorded in the bottom level (PMP) of the hierarchy. The complete results of step 1, are introduced in Figure 40.

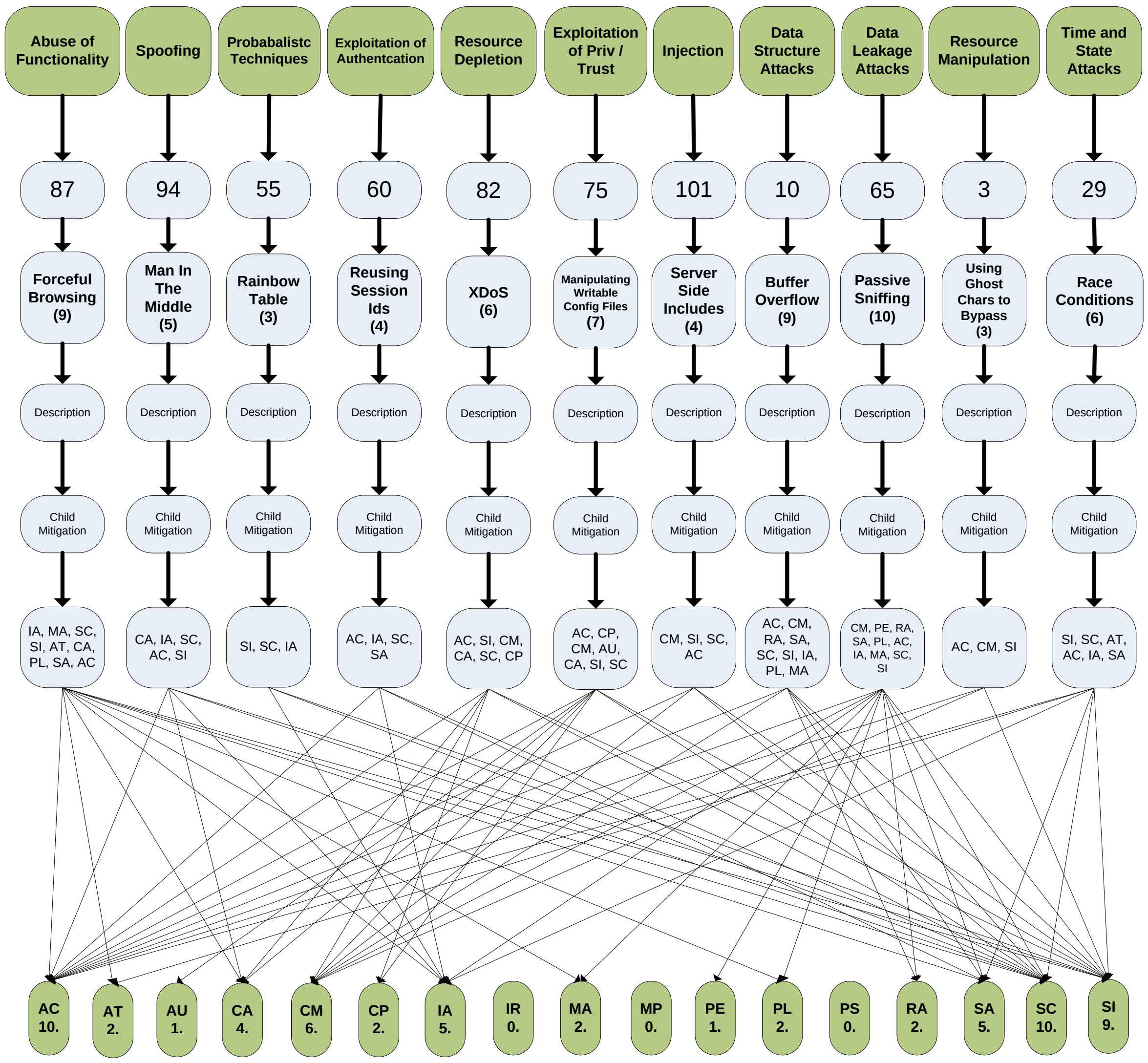


Figure 40. Complete Case Study Results for PMP “x” Value

Step 2 separates the "x" and "y" values with the use of a period. Step 3 calculates the PMP "y" value by examining the number of directly related NIST Children to each Parent Mitigation. This value is calculated by examining the previously created KOE tables (Tables 56-57 and 61-69). Figure 41 introduces the complete case study results for KOE and PMP value.

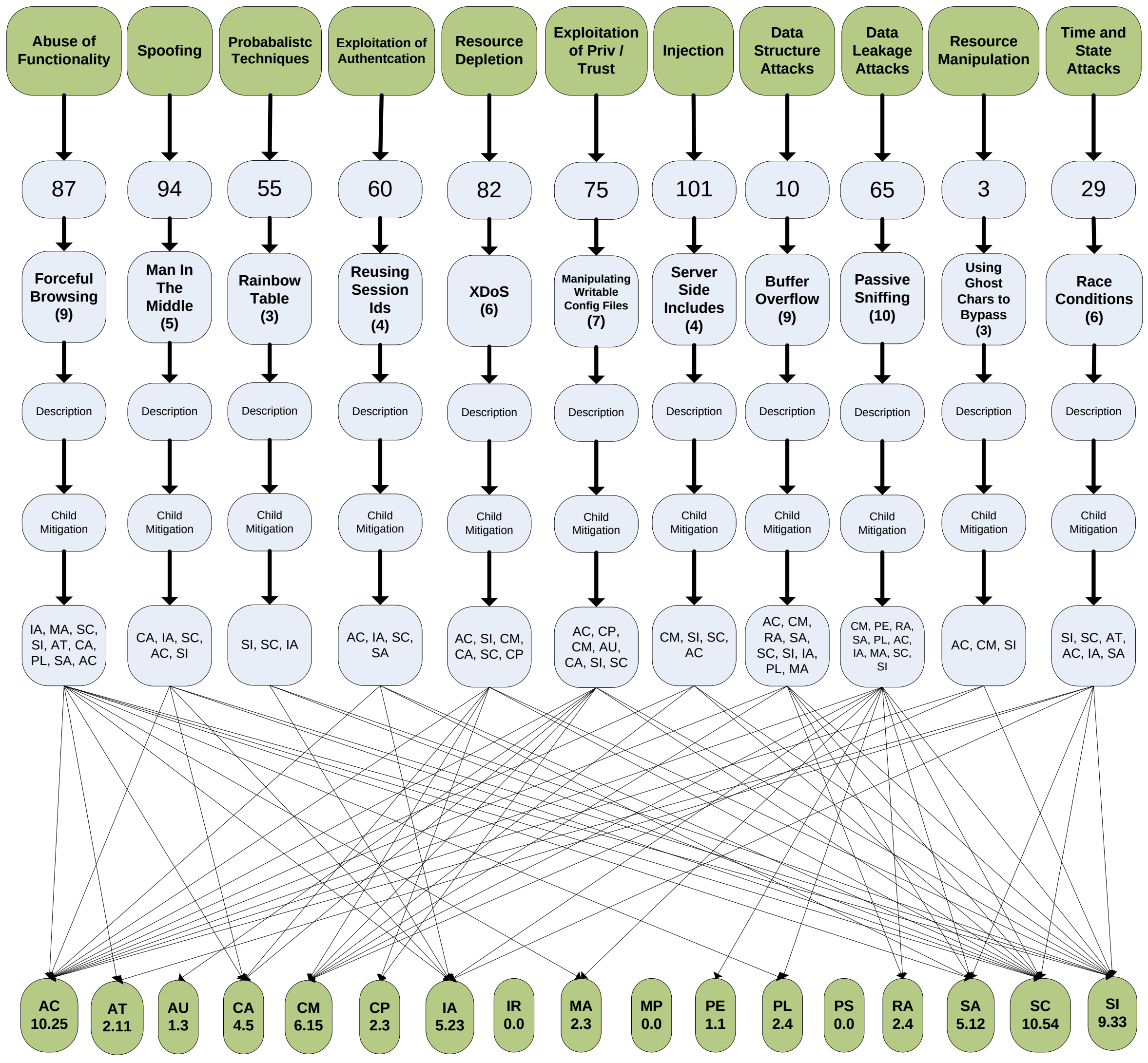


Figure 41. Complete Case Study Results Including KOE and PMP

Figure 41 is different from each of the previous figures because it includes a complete PMP value as show in the bottom level of the hierarchy. Step 4 creates a new table summarizing the complete PMP values for our case study. This allows users to quickly and accurately review PMP values without being overwhelmed by the Forest view presented in Figure 41. Table 79 introduces the PMP summary table.

Table 87. Summarized Parent Mitigation Power.

PM	**PMP**
AC	10.25
AT	2.11
AU	1.3
CA	4.5
CM	6.15
CP	2.3
IA	5.23
MA	2.3
PE	1.1
PL	2.4
RA	2.4
SA	5.12
SC	10.54
SI	9.33

Access Control (AC), System and Communication Protection (SC), System and Information Integrity (SI), and Identification and Authentication (IA) were the most common "x" values. These four Parent Mitigations account for 35 total attack pattern touches. The remaining 13 Parent Mitigations account for only 26 total attack pattern touches. PMP is useful for security managers and decision makers to better leverage where and when to allocate resources.

System and Communication Protection (SC), System and Information Integrity (SI), Access Control (AC), and Identification and Authentication (IA) were the most used “y” values. The “y” values from these four Parent Mitigations make up 135 NIST Child Mitigation touches, while the remaining 13 Parent Mitigations are used a total of 61 times. These findings conclude that System and Communication Protection (SC) is the most commonly recommended NIST mitigation for the CAPEC Release 1 dictionary.

5.4. Discussion and Results

The Knock-Out Effect (KOE) security metric allows for the necessary mitigation strategies for each attack pattern to be calculated and documented. The higher the KOE, the more Child Mitigations it will take to fully prevent and/or recover from a specific attack. KOE is not a listing of necessary mitigation strategies, but rather a numeric count as to how many Child Mitigations are necessary. The exact listing of the Child Mitigations is included as part of the “solutions and mitigation” element for each attack pattern in the full release of the CAPEC dictionary.

The creation of the Parent Mitigation Power (PMP) security metric is a measurable score associated with the chosen mitigation strategies of a specific implementation. Depending on what Parent and Child Threats are to be mitigated, a specific set of Child Mitigations will be employed. Because every Child Mitigation can be traced to a Parent Mitigation, we are able to measure how big of an impact each Parent Mitigation is having on the overall security posture of the system.

Validation for Process 3 can be found through an analytical evaluation of our results with other relevant findings. In January of 2009 SANS released a list of the “Top 25 Most Dangerous Programming Errors”. This list is the result of a collaborative effort between the SANS Institute, MITRE, and prominent software security experts from the United States and Europe (Christey, 2009). The intended purpose of the list is to raise awareness and educate consumers, programmers, and IT managers about the most common programmatic mistakes which lead to serious software vulnerabilities. The vulnerabilities are considered serious because they allow attackers to steal data, compromise systems, or deny access to critical resources (Christey, 2009). Specific details for each of the Top 25 errors are provided which include Common Weakness Enumeration (CWE) ID, Name, Supporting Data Fields, Discussion, Prevention and Mitigations, Related CWEs, Related Attack Patterns, Attack Frequency, Ease of Detection, Remediation Cost, and Attacker Awareness.

KOE in our approach is a count of the number of Parent Mitigations needed to fully mitigate a given attack pattern. Attack patterns with larger KOE scores require more effort to mitigate than attack patterns with smaller KOE scores. One of the descriptive fields provided for each of the Top 25 Programming Errors is “Remediation Cost”. Remediation Cost is defined as “the amount of effort required to fix the weakness” (Christey, 2009). Given the structure of the SANS Top 25 Programming Errors, it is possible to correlate our KOE scores with the SANS Top 25 list. Correlation of data between the KOE and SANS list can be used to provide validation for our metric.

The completion of Process 3 for all 101 attack patterns resulted in KOE scores ranging from 1-10. The SANS Institute ranks Remediation Cost on a 5 point scale with the following values: Low, Low-Medium, Medium, Medium-High, High in the new Top 25. Table 80 introduces the corresponding KOE and Remediation Cost for each of matching attacks between CAPEC and the Top 25.

Table 88. SANS Remediation Cost versus Process 3 KOE.

Error / Attack	**Remediation Cost**	**KOE**
Error Message Info Leak / 54	Low	2
SQL Injection / 66	Low	3
OS Command Injection / 88	Medium	3
Race Condition / 29	Medium-High	8
Cross Site Request Forgery / 62	High	6

Table 80 shows a high degree of correlation between our newly generate KOE and SANS Remediation Cost ranking. It is important to note that only the Errors which had a directly matching name from the CAPEC Dictionary list were considered for comparison.

Our research resulted in the full creation of 101 graphical hierarchies and 101 textual attack descriptions. Our approach also calculated and documented the KOE for each of the 101 attack patterns. We combined each of the graphical hierarchies into a single forest view and calculated the PMP values for all 17 Parent Mitigations across each of the 101 attack patterns. This work is significant as the results will not have to be completed again to be useful. Our approach and subsequent metrics can be used immediately to aid in security related decisions.

6. CONCLUSIONS

6.1. Contributions and Applicability

Our main contribution is an approach which meets the objectives of the problem definition. Our approach makes CAPEC more useable and consistent and is made up of three processes.

1. Abstracting Parent Mitigations
2. Creation of Trimmed Hierarchies for Modeling Attack Patterns
3. Creation of Security Metrics

A breakdown of our approach is introduced in Figure 42.

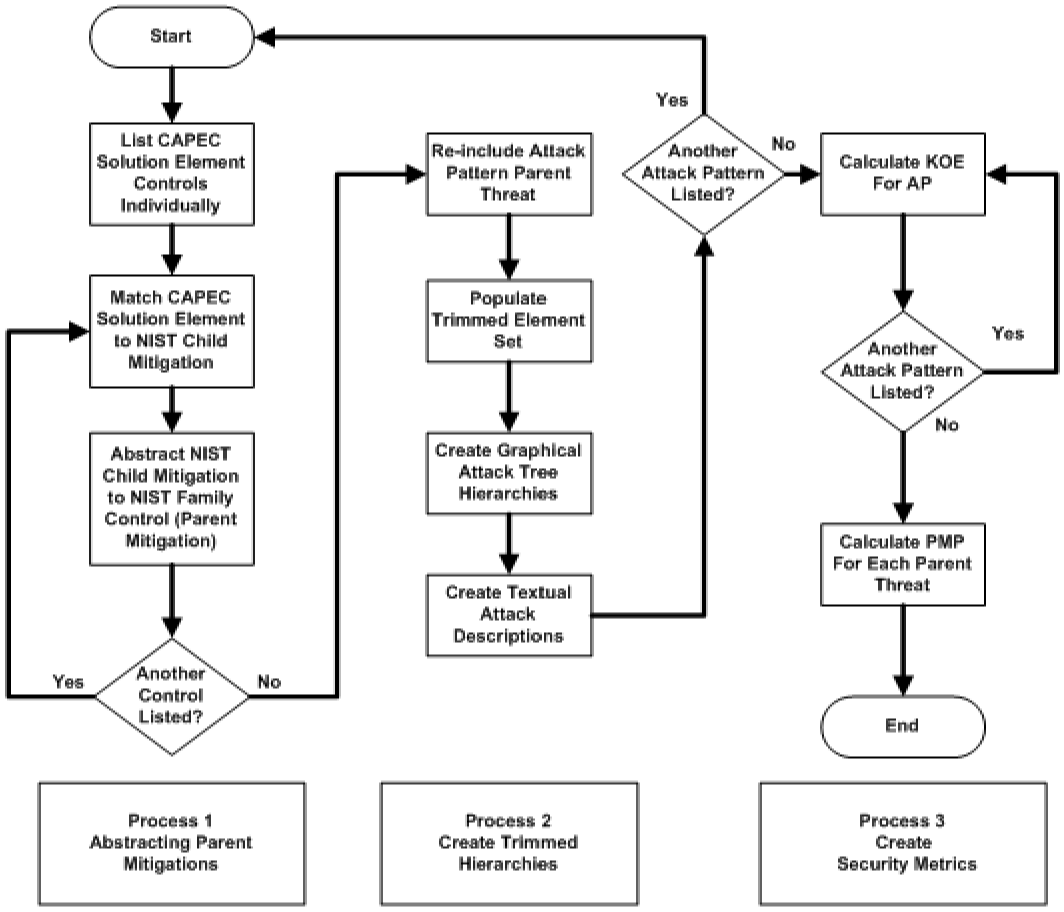

Start
List CAPEC Solution Element Controls Individually
Match CAPEC Solution Element to NIST Child Mitigation
Abstract NIST Child Mitigation to NIST Family Control (Parent Mitigation)
Another Control Listed?
Yes
No
Re-include Attack Pattern Parent Threat
Populate Trimmed Element Set
Create Graphical Attack Tree Hierarchies
Create Textual Attack Descriptions
Another Attack Pattern Listed?
Yes
No
Calculate KOE For AP
Another Attack Pattern Listed?
Yes
No
Calculate PMP For Each Parent Threat
End
Process 1 Abstracting Parent Mitigations
Process 2 Create Trimmed Hierarchies
Process 3 Create Security Metrics

Figure 42. Detailed Diagram of Our Approach

Our approach introduces a Parent Mitigation element to provide consistency and usability to the CAPEC Release 1 dictionary by incorporating the 800:53r2 NIST control repository directly into the CAPEC dictionary. Utilization of the existing NIST control group is important because it provides an accepted level of standardization. There is significant value in completing the abstraction process because CAPEC provides nearly 400 individual controls listed in the current attack pattern dictionary. Each of the mitigation strategies are now standardized into 17 Parent Mitigations at the same level of abstraction thus allowing adopters to make better use of the CAPEC dictionary. By abstracting these mitigations into 17 categories, users are less likely to dismiss a particular attack pattern because the mitigation is too detailed or too vague. This is currently a risk for CAPEC adopters who believe that they are not at risk for a given attack because they do not have the specific technology mentioned in the CAPEC mitigation.

Modeling hierarchy-based attack patterns begins by re-including a Parent Threat element into the dictionary to provide consistency and usability to the CAPEC Release 1 Dictionary. We presented a new model for viewing CAPEC attack patterns which creates a standardized element set that provides context for how the elements are related. Because the current elements provide valuable information, we are not advocating their removal. Rather our intention is to present the data in a manageable and consistent manner and full details of the Release 1 dictionary will be readily available. Because the current dictionary can easily overwhelm users, creating consistent and useable views for each attack pattern is a valuable step to increasing the adoption and wide spread acceptance of the CAPEC standard.

Our models allow CAPEC attack pattern information to be viewed from two distinct points of view. Graphical hierarchies allow for viewing element, attack pattern, Parent Threat and Parent Mitigation relationships. This allows users to trace both up from individual Parent Mitigations or down from individual Parent Threats. The textual attack descriptions present an attack pattern point of view. The creation of this model provides a stand-alone description for understanding the attack pattern.

The ability to accurately implement controls and answer security questions like: “Is my security better this year?”, “What am I getting for my security dollars?” and “How do I compare with my peers?” requires the use of security metrics (Geer, Hoo, & Jaquith, 2003). Metrics are also required to gauge the suitability and effectiveness of controls (Geer et al., 2003). The creation of the Knock-Out Effect (KOE) security metric allows for the necessary mitigation strategies for each attack pattern to be calculated and documented. The higher the KOE, the more Child Mitigations it will take to fully prevent and/or recover from a specific attack. KOE is not a listing of necessary mitigation strategies, but rather a numeric count as to how many Child Mitigations are necessary. The exact listing of the Child Mitigations is included as part of the “solutions and mitigation” element for each attack pattern in the full release of the CAPEC dictionary and the textual attack descriptions.

The creation of the Parent Mitigation Power (PMP) security metric is a score associated with the chosen mitigation strategies of a specific implementation. Depending on what Parent and Child Threats are to be mitigated, a specific set of Child Mitigations will be employed. Because every Child Mitigation can be traced to a Parent Mitigation, we are able to measure the impact each Parent Mitigation is having on the overall security posture of the system.

Our research resulted in the creation of a hierarchy including all 101 attack patterns, 101 textual attack descriptions, calculated the KOE for each of the 101 attack patterns, and calculated the PMP values for all 17 Parent Mitigations. Our approach can be used immediately to aid security related decisions. It is important to note that the results from our approach will always be the same regardless of who is completing the processes as long as they are followed explicitly.

6.2. Limitations

During Process 1 (Abstracting Parent Mitigations) it may be possible for others to reach different conclusions when matching CAPEC Solutions and Mitigations to NIST Parent Mitigations. Repeating this process several times and accumulating the results would help to alleviate this risk. Rigorous research methods need to be employed to ensure the elements and views proposed in Process 2 are appropriate for the target audience. Our approach justified the selection of each element and view, but need to be justified in an applied setting. The use of surveys and qualitative research method tools would ensure that each element and view is appropriate when applied outside of an academic environment.

Another identified limitation of our current approach is the lack of an automated tool for presenting CAPEC attack pattern information. While our approach has made CAPEC more consistent and usable, the implementation of CAPEC in an applied environment still requires manual review of the documentation. This causes the amount of time required to appropriately use CAPEC to be long. Automated tools would help in reducing the time factor of our approach and reduce or eliminate human error.

Due to the severe fan-in-fan-out and abundance of mappings, the completed Forest view, containing all 101 attack patterns, is overwhelming. New views, which include subsets of the tree, are needed if the Forest view is to be useful. The use of an automated tool would be beneficial for showing various components of the Forest view.

6.3. Future Work

Incorporating our work into an automated system would allow for a much quicker way of interacting with the standard. For example, a tool that assisted in managing and displaying the various views would be of great benefit because of the static nature of our work. Such a tool would allow security “what-if” scenarios to be quickly and accurately answered. The tool would present a variety of views and information including: full attack pattern information, graphical hierarchy trees, textual attack descriptions, attacks related by Parent Threat, KOE and PMP values, and a full forest view. The ability to quickly and accurately switch between each of these views would be a positive because time would be saved and human error avoided.

The work completed here needs to be forwarded on to the CAPEC community for review and consideration. Our research shares a common goal with the CAPEC organizers: increasing usability and consistency of the standard. The CAPEC community may be able to make use of both the Parent Threat and Parent Mitigations elements as well as the KOE and PMP metrics.

Other future work includes addressing the issues outlined in 6.2. Specifically, future work calls for ensuring that the views and metrics are both useful and accurate for end users and adopters. The use of surveys and other quantitative research methods needs to be employed to be certain that the appropriate elements and the appropriate number of elements are being used. The use of research methods also need to be applied to measure the appropriateness and effectiveness of our security metrics.

Given the vast repository of information which can be leveraged between the NIST and CAPEC standards, other future work could be conducted to include new security metrics.

REFERENCES CITED

Ali, M., Hilton, E. R., & Peter, S. B. (1985). Bayesian Probabilistic Risk Analysis. *13*(1), 5-12.

Arce, I. (2004). More Bang For the Bug: An Account of 2003's Attack Trends. *Attack Trends*.

Barnum, S. (2007). Attack Patterns as a Knowledge Resource for Building Secure Software. In A. Sethi (Ed.). OMG Software Assurance Workshop: Cigital.

Barnum, S. (2007). *Attack Patterns: Knowing Your Enemy in Order to Defeat Them*. Paper presented at the Black Hat DC 2007.

Barnum, S. (2008). *Common Attack Pattern Enumeration and Classification (CAPEC) Schema Description*.

Barnum, S., & Amit, S. (2006a). Further Information on Attack Patterns. *Build Security In Setting a Higher Standard for Software Assurance*, from https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/attack/589.html

Barnum, S., & Amit, S. (2006b, 2006-11-07). Introduction to Attack Patterns. from https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/attack/585-BSI.html

Barnum, S., & Sethi, A. Attack Pattern Glossary. *Build Security In. https://buildsecurityin. uscert. gov/daisy/bsi/articles/knowledge/attack/590. pdf*.

Bedford, T., & Cooke, R. (2001). *Probabilistic Risk Analysis: Foundations and Methods*: Cambridge University Press.

Bell, D. (1996). The Bell-LaPadula Model. *Journal of Computer Security, 4*(2), 3.

Benioff, M. R., & Lazowska, E. D. (2005). Cyber Security: A Crisis of Prioritization. *President's Information Technology Advisory Committee*

Benoit, A. A., Michel, P., & Suzanne, R. (2005). A Framework for Information Technology Outsourcing Risk Management. *36*(4), 9-28.

Bishop, M. (2003). What Is Computer Security? *University of California,Davis*.

Blakley, B., McDermott, E., & Geer, D. (2001). *Information Security is Information Risk Management*. Paper presented at the Proceedings of the 2001 workshop on New security paradigms.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*: Addison-Wesley Reading Mass.

Borlund, P., & Ingwersen, P. (1997). The Development of a Method for the Evaluation of Interactive Information Retrieval Systems. *Journal of Documentation, 53*, 225-250.

Brenner, J. (2007). ISO 27001: Risk management and compliance. *Risk Management Magazine, 54*(1), 24-29.

Burger, A. K. (2006). US Mobile Security, Part 1: How Great Is the Risk?

Byres, E. J., Franz, M., & Miller, D. (2004). The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. *International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal, December, 4*.

Calder, A. (2006). *Information Security Based on ISO 27001/ISO 17799: A Management Guide*: Van Haren Publishing.

capec.mitre.org. (2007). CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC). from http://capec.mitre.org/

Cavusoglu, H., Mishra, B., & Raghunathan, S. (2004). A Model for Evaluating IT Security Investments. *47*(7), 87-92.

CERT. (2007). Vulnerability Remediation Statistics. *Full Statistics*, from http://www.cert.org/stats/fullstats.html

Chakrabarti, A., & Manimaran, G. (2002). Internet Infrastructure Security: a Taxonomy. *Network, IEEE, 16*(6), 13-21.

Chow, S. S. M., Hui, L. C. K., Yiu, S. M., Chow, K. P., & Lui, R. W. C. (2005). A Generic Anti-Spyware Solution by Access Control List at Kernel Level. *The Journal of Systems & Software, 75*(1-2), 227-234.

Christey, S. (2009). CWE/SANS TOP 25 Most Dangerous Programming Errors. *SANS Institute*

Ciampa, M. (2005). *Security+ Guide to Network Security Fundamentals* (2nd ed.): Thomson Course Technology.

Coleman, T., & Jamieson, M. (1991). Information systems: evaluating intangible benefits at the feasibility stage of project appraisal. *Unpublished MBA thesis, City University Business School, London*.

Daley, K., Larson, R., & Dawkins, J. (2002). A structural framework for modeling multi-stage network attacks. *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, 5-10.

DeLooze, L. L. (2004). Classification of Computer Attacks Using a Self-Organizing Map. *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, 365-369.

Ellison, R. J., Fisher, D., Linger, R. C., Lipson, H. F., Longstaff, T., & Mead, N. R. (1997). Survivable Network Systems: An Emerging Discipline. *Software Engineering Institute at Carnegie Mellon University, CMU/SEI-97-TR-013, November*.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longstaff, T. A., & Mead, N. R. An Approach to Survivable Systems. *NATO 1 stSymposium on Protecting Inform. Systems in the 21 stCentury*, 25-27.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longstaff, T. A., & Mead, N. R. (1999). Survivability: Protecting Your Critical Systems.

Emmanuel, W., & Yu, S. Protecting Networks: Introduction to Network Security. cng.ateneo.net: Ateneo de Manila University.

Engebretson, P., & Pauli, J. (2008, November, 2008). *Realizing Knock-Out Effect and Parent Mitigation Power for Detailed Attack Patterns: A Case Study.* Paper presented at the Software Engineering and Applications (SEA 08), Orlando, FL, USA.

Engebretson, P., Pauli, J., & Streff, K. (2008, July, 2008). *Abstracting Parent Mitigations from the CAPEC Attack Pattern Dictionary.* Paper presented at the Security and Management (SAM 08), Las Vegas, NV, USA.

English, J., Hearst, M., Sinha, R., Swearingen, K., & Yee, K. P. (2002). *Hierarchical Faceted Metadata in Site Search Interfaces*.

Fadia, A. (2002). *Network Security: A Hacker's Perspective*: Course Technology.

Farbey, B., Land, F., & Targett, D. (1992). Evaluating investments in IT. *JIT. Journal of information technology(Print), 7*(2), 109-122.

Fithen, W. (2005). *Ensure that Input Is Properly Canonicalized*. Retrieved. from.

Flowerday, S., & Von Solms, R. (2005). Real-Time Information Integrity = Systems Integrity + Data Integrity + Continuous Assurances. *Computers & Security, 24*(8), 604-613.

Fong, E., Gaucher, R., Okun, V., Black, P. E., & Dalci, E. (2008). Building a Test Suite for Web Application Scanners. *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 478-478.

Fung, C., Chen, Y. L., Wang, X., Lee, J., Tarquini, R., Anderson, M., et al. (2005). Survivability Analysis of Distributed Systems Using Attack Tree Methodology. *Military Communications Conference, 2005. MILCOM 2005. IEEE*, 1-7.

Fung, P., & Longley, D. (2003). Electronic Information Security Documentation. 25-31.

Futoransky, A., Notarfrancesco, L., Richarte, G., & Sarraute, C. *Building Computer Network Attacks*: Technical report, CoreLabs.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Gautam, B., Kenneth, A. D., & Kazuhiko, K. (1989). *Applications of Qualitative Modeling to Knowledge-Based Risk Assessment Studies*. Paper presented at the Proceedings of the 2nd international conference on Industrial and engineering applications of artificial intelligence and expert systems - Volume 1.

Geer, D., Hoo, K. S., & Jaquith, A. (2003). Information Security: Why the Future Belongs to the Quants. *IEEE SECURITY & PRIVACY*, 24-32.

Gegick, M., & Williams, L. (2005). Matching attack patterns to security vulnerabilities in software-intensive system designs. 1-7.

Grance, T., Hash, J., & Stevens, M. (2003). NIST Special Publication 800-64, Security Considerations in the Information System Development Life Cycle. *October, Retrieved on, 26*, 800-864.

Haasl, D. F., Roberts, N. H., Vesely, W. E., & Goldberg, F. F. (1981). *Fault Tree Handbook*: NUREG-0492, Nuclear Regulatory Commission, Washington, DC (USA). Office of Nuclear Regulatory Research.

Hamdi, M., & Boudriga, N. (2003). Algebraic Specification of Network Security Risk Management. 52-60.

Hansche, S., Berti, J., & Hare, C. (2003). *Official (ISC) 2 Guide to the CISSP Exam*: Auerbach Publications.

Hansman, S., & Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security, 24*(1), 31-43.

Hearst, M. A. (2006). Clustering Versus Faceted Categories for Information Exploration. *49*(4), 59-61.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research (Vol. 28, pp. 75-106): MIS RESEARCH CENTER-SCHOOL OF MANAGEMENT.

Hoglund, G., & McGraw, G. (2004). *Exploiting Software: How to Break Code*: Pearson Higher Education.

Holden, G. (2003). *Guide to Network Defense and Countermeasures*: Course Technology Press United States.

Hong, M. *A Phenomenon Approach to Faceted Classification*. Paper presented at the 53rd Conference of the Japan Society of Library and Information Science (JSLIS).

ISACA. (2008). COBIT 4.1. from www.isaca.org/cobit/

ISO. (2005). 27002:2005. from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50297

Jajodia, S. (2007). Topological analysis of network attack vulnerability. *Conference on Computer and Communications Security: Proceedings of the 2 nd ACM symposium on Information, computer and communications security, 20*(22), 2-2.

Jones, K., Shema, M., & Johnson, B. C. (2002). *Anti-Hacker Tool Kit*: McGraw-Hill Osborne Media.

Koziol, J., Litchfield, D., Aitel, D., Anley, C., Eren, S., Mehta, N., et al. (2004). *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*: John Wiley & Sons.

Kwasnik, B. H. (1999). The Role of Classification in Knowledge Representation and Discovery. *Library Trends 48*(1), 22-47.

Lewis, T. G. (2006). *Critical Infrastructure Protection in Homeland Security: Defending a Networked Nation*: Wiley-Interscience.

Lindqvist, U., & Jonsson, E. (1997). How to Systematically Classify Computer Security Intrusions. *IEEE Symposium on Security and Privacy*, 83-99.

Logan, P. Y., & Clarkson, A. (2005). Teaching students to hack: curriculum issues in information security. *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 157-161.

Mauw, S., & Oostdijk, M. (2005). Foundations of attack trees. *Information Security and Cryptography (LNCS 3935), D. Won and S. Kim (Eds.), Springer, Berlin-Heidelberg, Germany*, 186–198.

McClure, S., Scambray, J., & Kurtz, G. (2005). *Hacking Exposed 5th Edition (Hacking Exposed)*: McGraw-Hill Osborne Media.

McCumber, J. (2004). *Assessing and Managing Security Risk in It Systems: A Structured Methodology*: Auerbach Pub.

McGraw, G. (2006). *Software Security: Building Security In*: Addison-Wesley Professional.

Moore, A. P., Ellison, R. J., & Linger, R. C. (2001). *Attack Modeling for Information Security and Survivability*: Carnegie Mellon University, Software Engineering Institute.

Myerson, J. M. (2002). Identifying Enterprise Network Vulnerabilities. *12*(3), 135-144.

Neumann, P. G. (2004). Principled Assuredly Trustworthy Composable Architectures. *Final Report, DARPA, 1*.

NIST. (2002). 800-30. *Risk Management Guide for Information Technology Systems,* 800-830.

NIST. (2006). *800-53*. Retrieved. from.

NIST. (2007). *800-53 Rev. 2*. Retrieved. from.

Pauli, J., & Engebretson, P. (2008a). *Hierarchy-Drive Approach for Attack Patterns in Software Security Education.* Paper presented at the 5th International Conference on Information Technology : New Generations, Las Vegas.

Pauli, J., & Engebretson, P. (2008b). *Towards a Specification Prototype for Hierarchy-Driven Attack Patterns.* Paper presented at the 5th International Conference on Information Technology : New Generations (ITNG 2008), Las Vegas.

Peltier, T. R. (2005). *Information Security Risk Analysis*: Auerbach Pub.

Recipes, S. Gray Hat Hacking: The Ethical Hacker's Handbook.

Rinaldi, S. M., Peerenboom, J. P., & Kelly, T. K. (2001). Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencies. *Control Systems Magazine, IEEE, 21*(6), 11-25.

Rockland, R. H. (2000). Reducing the information overload: a method on helping students research engineering topics using the Internet. *Education, IEEE Transactions on, 43*(4), 420-425.

Rowe, W. D. (1977). An Anatomy of Risk. *New York et al*.

Russell, R. (2002). *Hack Proofing Your Network*: Syngress.

Salter, C., Saydjari, O. S., Schneier, B., & Wallner, J. (1998). Toward a secure system engineering methodolgy. *Proceedings of the 1998 workshop on New security paradigms*, 2-10.

Schechter, S. E. (2005). Toward Econometric Models of the Security Risk from Remote Attacks.

Scheer, A. W., & Habermann, F. (2000). Enterprise resource planning: making ERP a success. *Communications of the ACM, 43*(4), 57-61.

Schneier, B. (1999). Attack Trees. *Dr. Dobb's Journal, 24*(12), 21-29.

Schneier, B. (2000). *Secrets & Lies: Digital Security in a Networked World*: John Wiley & Sons, Inc. New York, NY, USA.

Skoudis, E., & Liston, T. (2006). *Counter Hack Reloaded A Step-by-Step Guide to Computer Attacks and Effective Defenses* (2nd ed.). Upper Saddle River: Prentice Hall.

Steffan, J., & Schumacher, M. (2002). *Collaborative attack modeling*. Paper presented at the Proceedings of the 2002 ACM symposium on Applied computing.

Stoneburner, G., Goguen, A., & Feringa, A. (2002). Risk Management Guide for Information Technology Systems. *NIST Special Publication*, 800-830.

Swiderski, F., & Snyder, W. (2004). Threat Modeling: Microsoft Press.

Templeton, S. J., & Levitt, K. (2001). A requires/provides model for computer attacks. *Proceedings of the 2000 workshop on New security paradigms*, 31-38.

Tidwell, T., Larson, R., Fitch, K., & Hale, J. (2001). Modeling Internet Attacks. *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, 59*.

Viega, J., & McGraw, G. (2002). *Building Secure Software*: Addison-Wesley Boston.

Visintine, V. (2003). An Introduction to Information Risk Assessment. *SANS institute, 8*.

von Solms, B. (2005). Information Security governance: COBIT or ISO 17799 or both? *Computers & Security, 24*(2), 99-104.

Wang, H., & Wang, C. (2003). Taxonomy of Security Considerations and Software Quality. *Communications of the ACM, 46*(6), 75-78.

Weinstein, C. E., & Mayer, R. E. (1986). The Teaching of Learning Strategies. *Handbook of research on teaching, 3*, 315-327.

Whitman, M. E., & Mattord, H. J. (2003). *Principles of information security*: Boston, Mass.; London: Thomson/Course Technology.

Willcocks, L. (1992). Evaluating Information Technology Investments: Research Findings and Reappraisal. *Information Systems Journal, 2*(4), 243-268.

Woerner, R. (2007). Security Friday Fast Fact: Risky Business (without Tom Cruise).

Ye, Y., Barry, B., & Betsy, C. (2006). *Assessing COTS Integration Risk Using Cost Estimation Inputs*. Paper presented at the Proceeding of the 28th international conference on Software engineering.

APPENDIX I: 101 Attack Patterns: Complete

Textual Attack Descriptions

Attack Pattern ID	1
Attack Pattern Name (KOE)	Accessing Functionality Not Properly Constrained by ACLs (2)
Description	In applications, particularly web applications, access to functionality is mitigated by the authorization framework, whose job it is to map ACLs to elements of the application's functionality; particularly URL's for web apps. In the case that the application deployer failed to specify an ACL for a particular element, an attacker may be able to access it with impunity. An attacker with the ability to access functionality not properly constrained by ACLs can obtain sensitive information and possibly compromise the entire application. Such an attacker can access resources that must be available only to users at a higher privilege level, can access management sections of the application or can run queries for data that he is otherwise not supposed to.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. In a J2EE setting, deployers can associate a role that is impossible for the authenticator to grant users, such as "NoAccess", with all Servlets to which access is guarded by a limited number of servlets visible to, and accessible by, the user.. Having done so, any direct access to those protected Servlets will be prohibited by the web container. In a more general setting, the deployer must mark every resource besides the ones supposed to be exposed to the user as accessible by a role impossible for the user to assume. The default security setting must be to deny access and then grant access only to those resources intended by business logic.
Parent Mitigation	AC, IA

Attack Pattern ID	2
Attack Pattern Name (KOE)	Inducing Account Lockout, (2)
Description	An attacker leverages the security functionality of the system aimed at thwarting potential attacks to launch a denial of service attack against a legitimate system user. Many systems, for instance, implement a password throttling mechanism that locks an account after a certain number of incorrect log in attempts. An attacker can leverage this throttling mechanism to lock a legitimate user out of their own account. The weakness that is being leveraged by an attacker is the very security feature that has been put in place to counteract attacks.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. implement intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name. 2. When implementing security features, consider how they can be misused and made to turn on themselves.
Parent Mitigation	IA, PL

Attack Pattern ID	3
Attack Pattern Name (KOE)	Using Leading 'Ghost' Character Sequences to Bypass Input Filters (3)
Description	An attacker intentionally introduces leading characters that enable getting the input past the filters.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Perform white list rather than black list input validation. 2. Canonicalize all data prior to validation. 3. Take an iterative approach to input validation (defense in depth).
Parent Mitigation	AC, CM, SL

Attack Pattern ID	4
Attack Pattern Name (KOE)	Using Alternative IP Address Encodings (3)
Description	This attack relies on the attacker using unexpected formats for representing IP addresses. Networked applications may expect network location information in a specific format, such as fully qualified domains names, URL, IP address, or IP Address ranges. The issue that the attacker can exploit is that these design assumptions may not be validated against a variety of different possible encodings and network address location formats. Applications that use naming for creating policy namespaces for managing access control may be susceptible to queryin directly by IP addresses, which is ultimately a more generally authoritative way of communicating on a network. Alternative IP addresses can be used by the attacker to bypass application access control in order to gain access to data that is only protected by obscuring its location. In addition this type of attack can be used as a reconnaissance mechansim to provide entry point information that the attacker gathers to penetrate deeper into the system.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Design: Default deny access control policies 2. Design: Input validation routines should check and enforce both input data types and content against a positive specification. In regards to IP addresses, this should include the authorized manner for the application to represent IP addresses and not accept user specified IP addresses and IP address formats (such as ranges) 3. Implementation: Perform input validation for all remote content.
Parent Mitigation	AC SC SI

Attack Pattern ID	5
Attack Pattern Name (KOE)	Analog In-band Switching Signals (aka Blue Boxing) (2)
Description	This attack against older telephone switches and trunks has been around for decades. The signal is sent by the attacker to impersonate a supervisor signal. This has the effect of rerouting or usurping command of the line and call. While the US infrastructure proper may not contain widespread vulnerabilities to this type of attack, many companies are connected globally through call centers and business process outsourcing. These international systems may be operated in countries which have not upgraded telco infrastructure and so are vulnerable to Blue boxing. Blue boxing is a result of failure on the part of the system to enforce strong authentication for administrative functions. While the infrastructure is different than standard current applications like web applications, there are hisotrical lessons to be learned to upgrade the access control for administrative functions.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Implementation: Upgrade phone lines. Note this may be prohibitively expensive 2. Use strong access control such as two factor access control for adminsitrative access to the switch
Parent Mitigation	AC, MA

Attack Pattern ID	6
Attack Pattern Name (KOE)	Argument Injection (2)
Description	An attack of this type exploits a programs' vulnerabilities that allows an attacker's commands to be directly or indirectly applied as arguments, for example as shell commands. This may allow an attacker access to files, network resources, media, and in short anything accessible through the shell. The argument injection attack uses the exposed service or method as a launch pad to invoke other programs. If the service does not validate or filter the input data then the client program is granted access to execute commands using the server's privileges. The OS commands can be appended to standard input for shell programs, HTTP Requests, and XML messages.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Design: Do not program input values directly on command shell, instead treat user input as guilty until proven innocent. Build a function that takes user input and converts it to applications specific types and values, stripping or filtering out all unauthorized commands and characters in the process. 2. Design: Limit program privileges, so if metacharcters or other methods circumvent program input validation routines and shell access is attained then it is not running under a privileged account. chroot jails create a sandbox for the application to execute in, making it more difficult for an attacker to elevate privilege even in the case that a compromise has occurred. 3. Implementation: Implement an audit log that is written to a separate host, in the event of a compromise the audit log may be able to provide evidence and details of the compromise.
Parent Mitigation	AT, PL

Attack Pattern ID	7
Attack Pattern Name (KOE)	Blind SQL Injection (2)
Description	Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the attacker constructs input strings that probe the target through simple Boolean SQL expressions. The attacker can determine if the syntax and structure of the injection was successful based on whether the query was executed or not. Applied iteratively, the attacker determines how and where the target is vulnerable to SQL Injection.
Parent Threat	Injection
Solutions and Mitigations	1. Security by Obscurity is not a solution to preventing SQL Injection. Rather than suppress error messages and exceptions, the application must handle them gracefully, returning either a custom error page or redirecting the user to a default page, without revealing any information about the database or the application internals. 2. Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear.
Parent Mitigation	SI, CM

Attack Pattern ID	8
Attack Pattern Name (KOE)	Buffer Overflow in an API Call (2)
Description	This attack targets libraries or shared code modules which are vulnerable to buffer overflow attacks. An attacker who has access to an API may try to embed malicious code in the API function call and exploit a buffer overflow vulnerability in the function's implementation. All clients that make use of the code library thus become vulnerable by association. This has a very broad effect on security across a system, usually affecting more than one software process.
Parent Threat	Buffer Overflow, API Abuse, Injection
Solutions and Mitigations	1. Use a language or compiler that performs automatic bounds checking. 2. Use secure functions not vulnerable to buffer overflow. 3. If you have to use dangerous functions, make sure that you do boundary checking. 4. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 5. Use OS-level preventative functionality. Not a complete solution.
Parent Mitigation	AT, SC

Attack Pattern ID	9
Attack Pattern Name (KOE)	Buffer Overflow in Local Command-Line Utilities (5)
Description	This attack targets command-line utilities available in a number of shells. An attacker can leverage a vulnerability found in a command-line utility to escalate privilege to root.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Carefully review the service's implementation before making it available to user. For instance you can use manual or automated code review to uncover vulnerabilities such as buffer overflow. 2. Use a language or compiler that performs automatic bounds checking. 3. Use an abstraction library to abstract away risky APIs. Not a complete solution. 4. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 5. Operational: Use OS-level preventative functionality. Not a complete solution. 6. Apply the latest patches to your user exposed services. This may not be a complete solution, specially against zero day attack. 7. Do not unnecessarily expose services.
Parent Mitigation	RA, SI, CM, SA, AC

Attack Pattern ID	10
Attack Pattern Name (KOE)	Buffer Overflow Via Environment Variables (4)
Description	This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Do not expose environment variable to the user. 2. Do not use untrusted data in your environment variables. 3. Use a language or compiler that performs automatic bounds checking 4. You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow
Parent Mitigation	SI,AC,CM, RA

Attack Pattern ID	11
Attack Pattern Name (KOE)	Cause Web Server Misclassification (2)
Description	An attack of this type exploits a Web server's decision to take action based on filename or file extension. Because different file types are handled by different server processes, misclassification may force the Web server to take unexpected action, or expected actions in an unexpected sequence. This may cause the server to exhaust resources, supply debug or system data to the attacker, or bind an attacker to a remote process.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Implementation: Server routines should be determined by content not determined by filename or file extension.
Parent Mitigation	CM, IA

Attack Pattern ID	12
Attack Pattern Name (KOE)	Choosing a Message/Channel Identifier on a Public/Multicast Channel (2)
Description	Attackers aware that more data is being fed into a multicast or public information distribution means can 'select' information bound only for another client, even if the distribution means itself forces users to authenticate in order to connect initally. Doing so allows the attacker to gain access to possibly privileged information, possibly perpetrate other attacks through the distribution means by impersonation. If the channel/message being manipulated is an input rather than output mechanism for the system, (such as a command bus), this style of attack could change its identifier from a less privileged to more so privileged channel or command.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. Associate some ACL (in the form of a token) with an authenticated user which they provide middleware. The middleware uses this token as part of its channel/message selection for that client, or part of a discerning authorization decision for privileged channels/messages. The purpose is to architect the system in a way that associates proper authentication/authorization with each channel/message. 2. Rearchitect system input/output channels as appropriate to distribute self-protecting data. That is, encrypt (or otherwise protect) channels/messages so that only authorized readers can see them.
Parent Mitigation	IA, SC

Attack Pattern ID	13
Attack Pattern Name (KOE)	Subverting Environment Variable Values (3)
Description	The attacker directly or indirectly modifies environment variables used by or controlling the target software. The attacker's goal is to cause the target software to deviate from its expected operation in a manner that benefits the attacker.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Protect environment variables against unauthorized read and write access. 2. Protect the configuration files which contain environment variables against illegitimate read and write access. 3. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system. 4. Apply the least privilege principles. If a process has no legitimate reason to read an environment variable do not give that privilege.
Parent Mitigation	AC, SM, SI

Attack Pattern ID	14
Attack Pattern Name (KOE)	Client-side Induction-induced Buffer Overflow (10)
Description	This type of attack exploits a buffer overflow vulnerability in targeted client software through injection of malicious content from a custom-built hostile service.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. The client software should not install untrusted code from a non authenticated server. 2. The client software should have the latest patches and should be audited for vulnerabilities before being used to communicate with potentially hostile servers. 3. Perform input validation for length of buffer inputs. 4. Use a language or compiler that performs automatic bounds checking. 5. Use an abstraction library to abstract away risky APIs. Not a complete solution. 6. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 7. Ensure all buffer uses are consistently bounds-checked. 8. Use OS-level preventative functionality. Not a complete solution.
Parent Mitigation	AC, CM, IA SA, SI, AU, CA, MA, RA, AT

Attack Pattern ID	15
Attack Pattern Name (KOE)	Command Delimiters (4)
Description	An attack of this type exploits a programs' vulnerabilities that allows an attacker's commands to be concatenated onto a legitimate command with the intent of targeting other resources such as the file system or database. The system that uses a filter or a blacklist input validation, as opposed to whitelist validation is vulnerable to an attacker who predicts delimiters (or combinations of delimiters) not present in the filter or blacklist. As with other injection attacks, the attacker uses the command delimiter payload as an entry point to tunnel through the application and activate additional attacks through SQL queries, shell commands, network scanning, and so on.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Perform whitelist validation against a positive specification for command length, type, and parameters. 2. Design: Limit program privileges, so if commands circumvent program input validation or filter routines then commands do not running under a privileged account 3. Implementation: Perform input validation for all remote content. 4. Implementation: Use type conversions such as JDBC prepared statements.
Parent Mitigation	AC, CM, SA, RA

Attack Pattern ID	16
Attack Pattern Name (KOE)	Dictionary-based Password Attack (10)
Description	An attacker tries each of the words in a dictionary as passwords to gain access to the system via some user's account. If the password chosen by the user was a word within the dictionary, this attack will be successful (in the absence of other mitigations). This is a specific instance of the password brute forcing attack pattern.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Create a strong password policy and ensure that your system enforces this policy. 2. Implement an intelligent password throttling mechanism. Care must be taken to assure that these mechanisms do not excessively enable account lockout attacks such as CAPEC-02.
Parent Mitigation	AC, AT, AU, CA, CM, IA, MP, PL, PS, SI

Attack Pattern ID	17
Attack Pattern Name (KOE)	Accessing, Modifying or Executing Executable Files(3)
Description	An attack of this type exploits a system's configuration that allows an attacker to either directly access an executable file, for example through shell access; or in a possible worst case allows an attacker to upload a file and then execute it. Web servers, ftp servers, and message oriented middleware systems which have many integration points are particularly vulnerable, because both the programmers and the administrators must be in synch regarding the interfaces and the correct privileges for each interface.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Design: Enforce principle of least privilege 2. Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands. 3. Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.
Parent Mitigation	AC, AU, IA

Attack Pattern ID	18
Attack Pattern Name (KOE)	Embedding Scripts in Nonscript Elements (5)
Description	This attack is a form of Cross-Site Scripting (XSS) where malicious scripts are embedded in elements that are not expected to host scripts such as image tags (<img>), comments in XML documents (< !-CDATA->), etc. These tags may not be subject to the same input validation, output validation, and other content filtering and checking routines, so this can create an opportunity for an attacker to tunnel through the application's elements and launch a XSS attack through other elements.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting. 2. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 3. Implementation: Perform input validation for all remote content. 4. Implementation: Perform output validation for all remote content. 5. Implementation: Disable scripting languages such as Javascript in browser 6. Implementation: Session tokens for specific host 7. Implementation: Service provider should not use the XMLHttpRequest method to create a local proxy for content from other sites, because the client will not be able to discern what content comes from which host.
Parent Mitigation	AC, SI, SC, IA, MP

Attack Pattern ID	19
Attack Pattern Name (KOE)	Embedding Scripts within Scripts (6)
Description	An attack of this type exploits a programs' vulnerabilities that are brought on by allowing remote hosts to execute scripts. The attacker leverages this capability to execute scripts to execute his/her own script by embedding it within other scripts that the target software is likely to execute. The attacker must have the ability to inject script into script that is likely to be executed. If this is done, then the attacker can potentially launch a variety of probes and attacks against the web server's local environment, in many cases the so-called DMZ, back end resources the web server can communicate with, and other hosts. With the proliferation of intermediaries, such as Web App Firewalls, network devices, and even printers having JVMs and Web servers, there are many locales where an attacker can inject malicious scripts. Since this attack pattern defines scripts within scripts, there are likely privileges to execute said attack on the host.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting. 2. Design: Utilize strict type, character, and encoding enforcement 3. Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from. 4. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 5. Implementation: Perform input validation for all remote content. 6. Implementation: Perform output validation for all remote content. 7. Implementation: Disable scripting languages such as Javascript in browser 8. Implementation: Session tokens for specific host 9. Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this. 10. Implementation: Privileges are constrained, if a script is loaded, ensure system runs in chroot jail or other limited authority mode
Parent Mitigation	PL, SC, AC, RA, AC, SI

Attack Pattern ID	20
Attack Pattern Name (KOE)	Encryption Brute Force(4)
Description	An attacker, armed with the cipher text and the encryption algorithm used, performs an exhaustive (brute force) search on the key space to determine the key that decrypts the cipher text to obtain the plaintext.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. In theory a brute force attack performing an exhaustive keyspace search will always succeed, so the goal is to have computational security. Moore's law needs to be taken into account that suggests that computing resources double every eighteen months.
Parent Mitigation	AC, IA, PS, SC

Attack Pattern ID	21
Attack Pattern Name (KOE)	Exploitation of Session IDs,Resource IDs, Trusted Credentials (3)
Description	Attacks on session IDs and resource IDs take advantage of the fact that some software accepts user input without verifying its authenticity.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	1. Design: utilize strong federated identity such as SAML to encrypt and sign identity tokens in transit. 2. Implementation: Use industry standards session key generation mechanisms that utilize high amount of entropy to generate the session key. Many standard web and application servers will perform this task on your behalf.
Parent Mitigation	AC, IA, SC

Attack Pattern ID	22
Attack Pattern Name (KOE)	Exploitation Trust in Client (aka make client invisible) (3)
Description	An attack of this type exploits a programs' vulnerabilities in client/server communication channel authentication and data integrity.
Parent Threat	Exploitation of Privlege/Trust
Solutions and Mitigations	1. Design: Ensure that client process and/or message is authenticated so that anonymous communications and/or messages are not accepted by the system.
Parent Mitigation	AC, IA,SC

Attack Pattern ID	23
Attack Pattern Name (KOE)	File System Function Injection, Content Based (7)
Description	An attack of this type exploits the host's trust in executing remote content including binary files. The files are poisoned with a malicious payload (targeting the file systems accessible by the target software) by the attacker and may be passed through standard channels such as via email, and standard web content like PDF and multimedia files. The attacker exploits known vulnerabilities or handling routines in the target processes. Vulnerabilities of this type have been found in a wide variety of commercial applications from Microsoft Office to Adobe Acrobat and Apple Safari web browser. When the attacker knows the standard handling routines and can identify vulnerabilities and entry points they can be exploited by otherwise seemingly normal content. Once the attack is executed, the attacker's program can access relative directories such as C:\Program Files or other standard system directories to launch further attacks. In a worst case scenario, these programs are combined with other propagation logic and work as a virus.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Enforce principle of least privilege 2. Validate all input for content including files. Ensure that if files and remote content must be accepted that once accepted, they are placed in a sandbox type location so that lower assurance clients cannot write up to higher assurance processes (like Web server processes for example) 3. Execute programs with constrained privileges, so parent process does not open up further vulnerabilities. Ensure that all directories, temporary directories and files, and memory are executing with limited privileges to protect against remote execution. 4. Proxy communication to host, so that communications are terminated at the proxy, sanitizing the requests before forwarding to server host. 5. Virus scanning on host 6. Host integrity monitoring for critical files, directories, and processes. The goal of host integrity monitoring is to be aware when a security issue has occurred so that incident response and other forensic activities can begin.
Parent Mitigation	AC, CA, CM, CP, SI, SC, IR

Attack Pattern ID	24
Attack Pattern Name (KOE)	Filter Failure Through Buffer Overflow (3)
Description	In this attack, the idea is to cause an active filter to fail by causing an oversized transaction. An attacker may try to feed overly long input strings to the program in an attempt to overwhelm the filter (by causing a buffer overflow) and hoping that the filter does not fail securely (i.e. lets the user input into the system unfiltered).
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Make sure that ANY failure occurring in the filtering or input validation routine is properly handled and that offending input is NOT allowed to go through. Basically make sure that the vault is closed when failure occurs. 2. Pre-design: Use a language or compiler that performs automatic bounds checking. 3. Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 4. Operational: Use OS-level preventative functionality. Not a complete solution. 5. Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
Parent Mitigation	IR, SI, CM

Attack Pattern ID	25
Attack Pattern Name (KOE)	Forced Deadlock (2)
Description	This attack attempts to trigger and exploit a deadlock condition in the target software to cause a denial of service. A deadlock can occur when two or more competing actions are waiting for each other to finish, and thus neither ever does. Deadlock condition are not easy to detect.
Parent Threat	Time and State Attacks
Solutions and Mitigations	1. Use known algorithm to avoid deadlock condition (for instance non-blocking synchronization algorithms). 2. For competing actions use well known libraries which implement synchronization
Parent Mitigation	SC, SI

Attack Pattern ID	26
Attack Pattern Name (KOE)	Leveraging Race Conditions (5)
Description	This attack targets a race condition occurring when multiple processes access and manipulate the same resource concurrently and the outcome of the execution depends on the particular order in which the access takes place. The attacker can leverage a race condition by "running the race", modifying the resource and modifying the normal execution flow.
Parent Threat	Time and State Attacks
Solutions and Mitigations	1. Use safe libraries to access resources such as files. 2. Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition. 3. Use synchronization to control the flow of execution. 4. Use static analysis tools to find race conditions. 5. Pay attention to concurrency problems
Parent Mitigation	AC,MP,SA,SI,CM

Attack Pattern ID	27
Attack Pattern Name (KOE)	Leveraging Race Conditions via Symbolic Links (3)
Description	This attack leverages the use of symbolic links (Symlinks) in order to write to sensitive files. An attacker can create a Symlink link to a target file not otherwise accessible to her. When the privileged program tries to create a temporary file with the same name as the Symlink link, it will actually write to the target file pointed to by the attacker's Symlink link. If the attacker can insert malicious content in the temporary file she will be writing to the sensitive file by using the Symlink. The race occurs because the system checks if the temporary file exists, then creates the file. The attacker would typically create the Symlink during the interval between the check and the creation of the temporary file.
Parent Threat	Time and State Attack
Solutions and Mitigations	1. Use safe libraries when creating temporary files. For instance the standard library function mkstemp can be used to safely create temporary files. For shell scripts, the system utility mktemp does the same thing. 2. Access to the directories should be restricted as to prevent attackers from manipulating the files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. 3. Follow the principle of least privilege when assigning access rights to files. 4. Ensure good compartmentalization in the system to provide protected areas that can be trusted.
Parent Mitigation	SA, MP, SI

Attack Pattern ID	28
Attack Pattern Name (KOE)	Fuzzing (4)
Description	Fuzzing is a software testing method that feeds randomly constructed input to the system and looks for an indication that a failure in response to that input has occured. Fuzzing treats the system as a blackbox and is totally free from any preconceptions or assumptions about the system.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Test to ensure that the software behaves as per specification and that there are no unintended side effects. Ensure that no assumptions about the validity of data are made. 2. Use fuzz testing during the software QA process to uncover any surprises, uncover any assumptions or unexpected behavior.
Parent Mitigation	SI, SA, AC, RA

Attack Pattern ID	29
Attack Pattern Name (KOE)	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions (8)
Description	An attack of this type exploits a system's configuration that allows an attacker to either directly access an executable file, for example through shell access; or in a possible worst case allows an attacker to upload a file and then execute it. Web servers, ftp servers, and message oriented middleware systems which have many integration points are particularly vulnerable, because both the programmers and the administrators must be in synch regarding the interfaces and the correct privileges for each interface.
Parent Threat	Time and State Attacks
Solutions and Mitigations	1. Use safe libraries to access resources such as files 2. Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition 3. Use synchronization to control the flow of execution 4. Use static analysis tools to find race conditions. 5. Pay attention to concurrency problems related to the access of resources.
Parent Mitigation	AC, AU, CM, MP, RA, SA, SC, SI

Attack Pattern ID	30
Attack Pattern Name (KOE)	Hijacking a Privileged Thread of Execution (3)
Description	Attackers can sometimes hijack a privileged thread from the underlying system through synchronous (calling a privileged function that returns incorrectly) or asynchronous (callbacks, signal handlers, and similar) means.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Application Architects must be careful to design callback, signal, and similar asynchronous constructs such that they shed excess privilege prior to handing control to user-written (thus untrusted) code. 2. Application Architects must be careful to design privileged code blocks such that upon return (successful, failed, or unpredicted) that privilege is shed prior to leaving the block/scope.
Parent Mitigation	AC, SA, CM

Attack Pattern ID	31
Attack Pattern Name (KOE)	Accessing / Intercepting / Modifying HTTP Cookies (3)
Description	This attack relies on the use of HTTP Cookies to store credentials, state information and other critical data on client systems. The first form of this attack involves accessing HTTP Cookies to mine for potentially sensitive data contained therein. The second form of this attack involves intercepting this data as it is transmitted from client to server. This intercepted information is then used by the attacker to impersonate the remote user/session. The third form is when the cookie's content is modified by the attacker before it is sent back to the server. Here the attacker seeks to convince the target server to operate on this falsified information.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Use input validation for cookies 2. Generate and validate MAC for cookies 3. Use SSL/TLS to protect cookie in transit 4. Ensure the web server implements all relevant security patches, many exploitable buffer overflows are fixed in patches issued for the software.
Parent Mitigation	SI, SC, CA

Attack Pattern ID	32
Attack Pattern Name (KOE)	Embedding Scripts in HTTP Query Strings (4)
Description	A variant of cross-site scripting called "reflected" cross-site scripting, the HTTP Query Strings attack consists of passing a malicious script inside an otherwise valid HTTP request query string. This is of significant concern for sites that rely on dynamic, user-generated content such as bulletin boards, news sites, blogs, and web enabled administration GUIs. The malicious script may steal session data, browse history, probe files, or otherwise execute attacks on the client side. Once the attacker has prepared the malicious HTTP query it is sent to a victim user (perhaps by email, IM, or posted on an online forum), who clicks on a normal looking link that contains a poison query string. This technique can be made more effective through the use of services like http://tinyurl.com/, which makes very small URLs that will redirect to very large, complex ones. The victim will not know what he is really clicking on.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting. 2. Design: Utilize strict type, character, and encoding enforcement 3. Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from. 4. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 5. Implementation: Perform input validation for all remote content, including remote and user-generated content 6. Implementation: Perform output validation for all remote content. 7. Implementation: Disable scripting languages such as Javascript in browser 8. Implementation: Session tokens for specific host 9. Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this. 10. Implementation: Privileges are constrained, if a script is loaded, ensure system runs in chroot jail or other limited authority mode
Parent Mitigation	SI, AC, CM, AU

Attack Pattern ID	33
Attack Pattern Name (KOE)	HTTP Request Smuggling (3)
Description	HTTP Request Smuggling results from the discrepancies in parsing HTTP requests between HTTP entities such as web caching proxies or application firewalls. Entities such as web servers, web caching proxies, application firewalls or simple proxies often parse HTTP requests in slightly different ways. Under specific situations where there are two or more such entities in the path of the HTTP request, a specially crafted request is seen by two attacked entities as two different sets of requests. This allows certain requests to be smuggled through to a second entity without the first one realizing it.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. HTTP Request Smuggling is usually targeted at web servers. Therefore, in such cases, careful analysis of the entities must occur during system design prior to deployment. If there are known differences in the way the entities parse HTTP requests, the choice of entities needs consideration. 2. Employing an application firewall can help. However, there are instances of the firewalls being susceptible to HTTP Request Smuggling as well.
Parent Mitigation	SA, SI, SC

Attack Pattern ID	34
Attack Pattern Name (KOE)	HTTP Response Splitting (2)
Description	This attack uses a maliciously-crafted HTTP request in order to cause a vulnerable web server to respond with an HTTP response stream that will be interpreted by the client as two separate responses instead of one. This is possible when user-controlled input is used unvalidated as part of the response headers. The target software, the client, will interpret the injected header as being a response to a second request, thereby causing the maliciously-crafted contents be displayed and possibly cached.
Parent Threat	Schema Poisoning
Solutions and Mitigations	1. To avoid HTTP Response Splitting, the application must not rely on user-controllable input to form part of its output response stream. Specifically, response splitting occurs due to injection of CR-LF sequences and additional headers. All data arriving from the user and being used as part of HTTP response headers must be subjected to strict validation that performs simple character-based as well as semantic filtering to strip it of malicious character sequences and headers.
Parent Mitigation	SI, SC

Attack Pattern ID	35
Attack Pattern Name (KOE)	Leverage Executable Code in Nonexecutable Files (4)
Description	An attack of this type exploits a system's trust in configuration and resource files, when the executable loads the resource (such as an image file or configuration file) the attacker has modified the file to either execute malicious code directly or manipulate the target process (e.g. application server) to execute based on the malicious configuration parameters. Since systems are increasingly interrelated mashing up resources from local and remote sources the possibility of this attack occurring is high.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Design: Enforce principle of least privilege 2. Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands. 3. Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables. 4. Implementation: Implement host integrity monitoring to detect any unwanted altering of configuration files. 5. Implementation: Ensure that files that are not required to execute, such as configuration files, are not over-privileged, i.e. not allowed to execute.
Parent Mitigation	AC, CA, CP, CM

Attack Pattern ID	36
Attack Pattern Name (KOE)	Using Unpublished Web Service APIs (5)
Description	An attacker searches for and invokes Web Services APIs that the target system designers did not intend to be publicly available. If these APIs fail to authenticate requests the attacker may be able to invoke services and/or gain privileges they are not authorized for.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. Authenticating both services and their discovery, and protecting that authentication mechanism simply fixes the bulk of this problem. Protecting the authentication involves the standard means, including: 1) protecting the channel over which authentication occurs, 2) preventing the theft, forgery, or prediction of authentication credentials or the resultant tokens, or 3) subversion of password reset and the like.
Parent Mitigation	AC, CA, CM, IA, SC

Attack Pattern ID	37
Attack Pattern Name (KOE)	Lifting Data Embedded in Client Distributions (4)
Description	An attacker can resort to stealing data embedded in client distributions or client code in order to gain certain information. This information can reveal confidential contents, such as account numbers, or can be used as an intermediate step in a larger attack (such as by stealing keys/credentials).
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Never Use Unvalidated Input as Part of a Directive to any Internal Component 2. Treat the Entire Inherited Process Context as Unvalidated Input 3. Use Well-Known Cryptography Appropriately and Correctly
Parent Mitigation	AC, IA, SC, SI

Attack Pattern ID	38
Attack Pattern Name (KOE)	Leveraging/Manipulating Configuration File Search Paths (8)
Description	This attack loads a malicious resource into a program's standard path used to bootstrap and/or provide contextual information for a program like a path variable or classpath. J2EE applications and other component based applications that are built from mutliple binaries can have very long list of dependencies to execute. If one of these libraries and/or references is controllable by the attacker then application controls can be circumvented by the attacker. A standard UNIX path looks similar to this /bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin If the attacker modifies the path variable to point to a locale that includes malicious resources then the user unwittingly can execute commands on the attacker's behalf: /evildir/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin This is a form of usurping control of the program and the attack can be done on the classpath, database resources, or any other resources built from compound parts. At runtime detection and blocking of this attack is nearly impossible, because the configuration allows execution.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Enforce principle of least privilege 2. Ensure that the program's compound parts, including all system dependencies, classpath, path, and so on, are secured to the same or higher level assurance as the program 3. Host integrity monitoring
Parent Mitigation	AC, AU, CA, CM, MP, RA, SC, SI

Attack Pattern ID	39
Attack Pattern Name (KOE)	Manipulating Opaque Client-based Data Tokens (6)
Description	In circumstances where an application holds important data client-side in tokens (cookies, URLs, data files, and so forth) that data can be manipulated. If client or server-side application components reinterpret that data as authentication tokens or data (such as store item pricing or wallet information) then even opaquely manipulating that data may bear fruit for an Attacker. In this pattern an attacker undermines the assumption that client side tokens have been adequately protected from tampering through use of encryption or obfuscation.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. One solution to this problem is to protect encrypted data with a CRC of some sort. If knowing who last manipulated the data is important, then using a cryptographic "message authentication code" (or hMAC) is prescribed. However, this guidance is not a panecea. In particular, any value created by (and therefore encrypted by) the client, which itself is a "malicous" value, all the protective cryptography in the world can't make the value 'correct' again. Put simply, if the client has control over the whole process of generating and encoding the value--then simply protecting its integrity doesn't help. 2. Make sure to protect client side authentication tokens for confidentiality (encryption) and integrity (signed hash) 3. Make sure that all session tokens use a good source of randomness 4. Perform validation on the server side to make sure that client side data tokens are consistent with what is expected.
Parent Mitigation	AU, IA, SI, CM, SA, SC

Attack Pattern ID	40
Attack Pattern Name (KOE)	Manipulating Writeable Terminal Devices (4)
Description	This attack exploits terminal devices that allow themselves to be written to by other users. The attacker sends command strings to the target terminal device hoping that the target user will hit enter and thereby execute the malicious command with their privileges. The attacker can send the results (such as copying /etc/passwd) to a known directory and collect once the attack has succeeded.
Parent Threat	Injection
Solutions and Mitigations	IA-2, IA-4, IA-5, AC-6
Parent Mitigation	IA, AC

Attack Pattern ID	41
Attack Pattern Name (KOE)	Using Meta-characters in E-mail Headers to Inject Malicious Payloads (4)
Description	This type of attack involves an attacker leveraging meta-characters in email headers to inject improper behavior into email programs. Email software has become increasingly sophisticated and feature-rich. In addition, email applications are ubiquitous and connected directly to the Web making them ideal targets to launch and propagate attacks. As the user demand for new functionality in email applications grows, they become more like browsers with complex rendering and plug in routines. As more email functionality is included and abstracted from the user, this creates opportunities for attackers. Virtually all email applications do not list email header information by default, however the email header contains valuable attacker vectors for the attacker to exploit particularly if the behavior of the email client application is known. Meta-characters are hidden from the user, but can containt scripts, enumerations, probes, and other attacks against the user's system.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Perform validation on email header data 2. Implementation: Implement email filtering solutions on mail server or on MTA, relay server. 3. Implementation: Mail servers that perform strict validation may catch these attacks, because metacharacters are not allowed in many header variables such as dns names
Parent Mitigation	AU, IA, SC, SI

Attack Pattern ID	42
Attack Pattern Name (KOE)	MIME Conversion(1)
Description	An attacker exploits a weakness in the MIME conversion routine to cause a buffer overflow and gain control over the mail server machine. The MIME system is designed to allow various different information formats to be interpreted and sent via e-mail. Attack points exist when data are converted to MIME compatible format and back.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Stay up to date with third party vendor patches 2. Disable the 7 to 8 bit conversion. This can be done by removing the F=9 flag from all Mailer specifications in the sendmail.cf file. 3. Use the sendmail restricted shell program (smrsh) 4. Use mail.local
Parent Mitigation	SI

Attack Pattern ID	43
Attack Pattern Name (KOE)	Exploiting Multiple Input Interpretation Layers (4)
Description	An attacker supplies the target software with input data that contains sequences of special characters designed to bypass input validation logic. This exploit relies on the target making multiples passes over the input data and processing a “layer” of special characters with each pass. In this manner, the attacker can disguise input that would otherwise be rejected as invalid by concealing it with layers of special/escape characters that are stripped off by subsequent processing steps. The goal is to first discover cases where the input validation layer executes before one or more parsing layers. That is, user input may go through the following logic in an application: <<parser1>> --> <<input validator>> --> <<parser2>>. In such cases, the attacker will need to provide input that will pass through the input validator, but after passing through parser2, will be converted into something that the input validator was supposed to stop.
Parent Threat	Leverage Alternate Encoding
Solutions and Mitigations	1. An iterative approach to input validation may be required to ensure that no dangerous characters are present. It may be necessary to implement redundant checking across different input validation layers. Ensure that invalid data is rejected as soon as possible and do not continue to work with it. 2. Make sure to perform input validation on canonicalized data (i.e. data that is data in its most standard form). This will help avoid tricky encodings getting past the filters. 3. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.
Parent Mitigation	SI, RA, CM, AT

Attack Pattern ID	44
Attack Pattern Name (KOE)	Overflow Binary Resource File (6)
Description	An attack of this type exploits a buffer overflow vulnerability in the handling of binary resources. Binary resources may includes music files like MP3, image files like JPEG files, and any other binary file. These attacks may pass unnoticed to the client machine through normal usage of files, such as a browser loading a seemingly innocent JPEG file. This can allow the attacker access to the execution stack and execute arbitrary code in the target process. This attack pattern is a variant of standard buffer overflow attacks using an unexpected vector (binary files) to wrap its attack and open up a new attack vector. The attacker is required to either directly serve the binary content to the victim, or place it in a locale like a MP3 sharing application, for the victim to download. The attacker then is notified upon the download or otherwise locates the vulnerability opened up by the buffer overflow.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Perform appropriate bounds checking on all buffers. 2. Design: Enforce principle of least privilege 3. Design: Static code analysis 4. Implementation: Execute program in less trusted process space environment, do not allow lower integrity processes to write to higher integrity processes 5. Implementation: Keep software patched to ensure that known vulnerabilities are not available for attackers to target on host.
Parent Mitigation	CA, MA, AC, RA, SC, SI

Attack Pattern ID	45
Attack Pattern Name (KOE)	Buffer Overflow via Symbolic Links (7)
Description	This type of attack leverages the use of symbolic links to cause buffer overflows. An attacker can try to create or manipulate a symbolic link file such that its contents result in out of bounds data. When the target software processes the symbolic link file, it could potentially overflow internal buffers with insufficient bounds checking.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Enforce principle of least privilege 2. Protect files, secure location (of files), encryption 3. Data sanitization 4. Abstraction, obfuscation, library checking
Parent Mitigation	AC, AU, CA, CM, MP, SI, SC

Attack Pattern ID	46
Attack Pattern Name (KOE)	Overflow Variables and Tags (4)
Description	This type of attack leverages the use of tags or variables from a formatted configuration data to cause buffer overflow. The attacker crafts a malicious HTML page or configuration file that includes oversized strings, thus causing an overflow.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Use a language or compiler that performs automatic bounds checking. 2. Use an abstraction library to abstract away risky APIs. Not a complete solution. 3. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 4. Use OS-level preventative functionality. Not a complete solution. 5. Do not trust input data from user. Validate all user input.
Parent Mitigation	SC,AC,SI,RA

Attack Pattern ID	47
Attack Pattern Name (KOE)	Buffer Overflow via Parameter Expansion (5)
Description	In this attack, the target software is given input that the attacker knows will be modified and expanded in size during processing. This attack relies on the target software failing to anticipate that the expanded data may exceed some internal limit, thereby creating a buffer overflow.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Ensure that when parameter expansion happens in the code that the assumptions used to determine the resulting size of the parameter are accurate and that the new size of the parameter is visible to the whole system
Parent Mitigation	CP, CM, CA, PL, SC

Attack Pattern ID	48
Attack Pattern Name (KOE)	Passing Local Filenames to Functions That Expect a URL (4)
Description	This attack relies on client side code to access local files and resources instead of URLs. When the client browser is expecting a URL string, but instead receives a request for a local file, that execution is likely to occur in the browser process space with the browser's authority to local files. The attacker can send the results of this request to the local files out to a site that they control. This attack may be used to steal sensitive authentication data (either local or remote), or to gain system profile information to launch further attacks.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 2. Implementation: Ensure all configuration files and resource are either removed or protected when promoting code into production. 3. Design: Use browser technologies that do not allow client side scripting. 4. Implementation: Perform input validation for all remote content. 5. Implementation: Perform output validation for all remote content. 6. Implementation: Disable scripting languages such as Javascript in browser
Parent Mitigation	SI, CM, SA, SC

Attack Pattern ID	49
Attack Pattern Name (KOE)	Password Brute Forcing (4)
Description	In this attack, the attacker tries every possible value for a password until they succeed. A brute force attack, if feasible computationally, will always be successful because it will essentially go through all possible passwords given the alphabet used (lower case letters, upper case letters, numbers, symbols, etc.) and the maximum length of the password. A system will be particularly vulnerable to this type of an attack if it does not have a proper enforcement mechanism in place to ensure that passwords selected by users are strong passwords that comply with an adequate password policy. In practice a pure brute force attack on passwords is rarely used, unless the password is suspected to be weak. Other password cracking methods exist that are far more effective (e.g. dictionary attacks, rainbow tables, etc.).
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Implement a password throttling mechanism. This mechanism should take into account both the IP address and the log in name of the user. 2. Put together a strong password policy and make sure that all user created passwords comply with it. Alternatively automatically generate strong passwords for users. 3. Passwords need to be recycled to prevent aging, that is every once in a while a new password must be chosen.
Parent Mitigation	IA, AC, CM, SC

Attack Pattern ID	50
Attack Pattern Name (KOE)	Password Recovery Exploitation (2)
Description	An attacker may take advantage of the application feature to help users recover their forgotten passwords in order to gain access into the system with the same privileges as the original user. Generally password recovery schemes tend to be weak and insecure. Most of them use only one security question . For instance, mother's maiden name tends to be a fairly popular one. Unfortunately in many cases this information is not very hard to find, especially if the attacker knows the legitimate user. These generic security questions are also re-used across many applications, thus making them even more insecure. An attacker could for instance overhear a coworker talking to a bank representative at the work place and supplying their mother's maiden name for verification purposes. An attacker can then try to log in into one of the victim's accounts, click on "forgot password" and there is a good chance that the security question there will be to provide mother's maiden name. A weak password recovery scheme totally undermines the effectiveness of a strong password scheme.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. Use multiple security questions (e.g. have three and make the user answer two of them correctly). Let the user select their own security questions or provide them with choices of questions that are not generic. 2. E-mail the temporary password to the registered e-mail address of the user rather than letting the user reset the password online. 3. Ensure that your password recovery functionality is not vulnerable to an injection style attack.
Parent Mitigation	IA, SA

Attack Pattern ID	51
Attack Pattern Name (KOE)	Poison Web Service Registry (7)
Description	SOA and Web Services often use a registry to perform look up, get schema information, and metadata about services. A poisoned registry can redirect (think phishing for servers) the service requester to a malicious service provider, provide incorrect information in schema or metadata (to effect a denial of service), and delete information about service provider interfaces. WS-Addressing is used to virtualize services, provide return addresses and other routing information, however, unless the WS-Addressing headers are protected they are vulnerable to rewriting. The attacker that can rewrite WS-addressing information gains the ability to route service requesters to any service providers, and the ability to route service provider response to any service. Content in a registry is deployed by the service provider. The registry in an SOA or Web Services system can be accessed by the service requester via UDDI or other protocol. The basic flow for the attacker consists of either altering the data at rest in the registry or uploading malicious content by spoofing a service provider. The service requester is then redirected to send its requests and/or responses to services the attacker controls.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Enforce principle of least privilege 2. Harden registry server and file access permissions 3. Implement communications to and from the registry using secure protocols
Parent Mitigation	AC, AU, CA, CM, MP, SI, SC

Attack Pattern ID	52
Attack Pattern Name (KOE)	Embedding NULL Bytes (1)
Description	An attacker embeds one or more null bytes in input to the target software. This attack relies on the usage of a null-valued byte as a string terminator in many environments. The goal is for certain components of the target software to stop processing the input when it encounters the null byte(s).
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Properly handle the NULL characters supplied as part of user input prior to doing anything with the data.
Parent Mitigation	SI

Attack Pattern ID	53
Attack Pattern Name (KOE)	Postfix, Null Terminate, and Backslash (3)
Description	If a string is passed through a filter of some kind, then a terminal NULL may not be valid. Using alternate representation of NULL allows an attacker to embed the NULL midstring while postfixing the proper data so that the filter is avoided. One example is a filter that looks for a trailing slash character. If a string insertion is possible, but the slash must exist, an alternate encoding of NULL in midstring may be used.
Parent Threat	Input Data Manipulation
Solutions and Mitigations	1. Properly handle Null characters. Make sure canonicalization is properly applied. Do not pass Null characters to the underlying APIs. 2. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.
Parent Mitigation	SI, AC, CM

Attack Pattern ID	54
Attack Pattern Name (KOE)	Probing an Application Through Targeting its Error Reporting (2)
Description	An attacker, aware of an application's location (and possibly authorized to use the application) can probe the application's structure and evaluate its robustness by probing its error conditions (not unlike one would during a 'fuzz' test, but more purposefully here) in order to support attacks such as blind SQL injection, or for the more general task of mapping the application to mount another subsequent attack.
Parent Threat	Data Leakage Attacks
Solutions and Mitigations	1. Application designers can construct a 'code book' for error messages. When using a code book, application error messages aren't generated in string or stack trace form, but are cataloged and replaced with a unique (often integer-based) value 'coding' for the error. Such a technique will require helpdesk and hosting personnel to use a 'code book' or similar mapping to decode application errors/logs in order to respond to them normally. 2. Application designers can wrap application functionality (preferably through the underlying framework) in an output encoding scheme that obscures or cleanses error messages to prevent such attacks. Such a technique is often used in conjunction with the above 'code book' suggestion.
Parent Mitigation	SC, SI

Attack Pattern ID	55
Attack Pattern Name (KOE)	Rainbow table password cracking (3)
Description	An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system. A password rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table. Constructing the rainbow tables takes a very long time and is computationally expensive. A separate table needs to be constrcuted for the various hash algorithms (e.g. SHA1, MD5, etc.). However, once a rainbow table is computed, it can be very effective in cracking the passwords that have been hashed without the use of salt.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.
Parent Mitigation	SI, SC, IA

Attack Pattern ID	56
Attack Pattern Name (KOE)	Removing/short-circuiting 'guard logic' (2)
Description	Attackers can, in some cases, get around logic put in place to 'guard' sensitive functionality or data. The attack may involve gaining access to and calling protected functionality (or accessing protected data) directly, may involve subverting some aspect of the guard's implementation, or outright removal of the guard, if possible.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Use Authentication Mechanisms, Where Appropriate, Correctly 2. Use Authorization Mechanisms Correctly
Parent Mitigation	AC, IA

Attack Pattern ID	57
Attack Pattern Name (KOE)	Utilizing REST's Trust in the System Resource to Register Man in the Middle (3)
Description	This attack utlizes a Rest(REpresentational State Transfer)-style applications' trust in the system resources and environment to place man in the middle once SSL is terminated. Rest applications premise is that they leverage existing infrastructure to deliver web services functionality.
Parent Threat	Spoofing
Solutions and Mitigations	1. Implementation: Implement message level security such as HMAC in the HTTP communication 2. Design: Utilize defense in depth, do not rely on a single security mechanism like SSL 3. Design: Enforce principle of least privilege
Parent Mitigation	SA, SI, AC

Attack Pattern ID	58
Attack Pattern Name (KOE)	Restful Privilege Elevation (2)
Description	Rest uses standard HTTP (Get, Put, Delete) style permissions methods, but these are not necessarily correlated generally with back end programs. Strict interpretation of HTTP get methods means that these HTTP Get services should not be used to delete information on the server, but there is no access control mechanism to back up this logic. This means that unless the services are properly ACL'd and the application's service implementation are following these guidelines then an HTTP request can easily execute a delete or update on the server side. The attacker identifies a HTTP Get URL such as http://victimsite/updateOrder, which calls out to a program to update orders on a database or other resource. The URL is not idempotent so the request can be submitted multiple times by the attacker, additionally, the attacker may be able to exploit the URL published as a Get method that actually performs updates (instead of merely retrieving data). This may result in malicious or inadvertant altering of data on the server.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Design: Enforce principle of least privilege 2. Implementation: Ensure that HTTP Get methods only retrieve state and do not alter state on the server side 3. Implementation: Ensure that HTTP methods have proper ACLs based on what the funcitonality they expose
Parent Mitigation	AC, CM, SI

Attack Pattern ID	59
Attack Pattern Name (KOE)	Session Credential Falsification through Prediction (3)
Description	This attack targets predictable session ID in order to gain privileges. The attacker can predict the session ID used during a transaction to perform spoofing and session hijacking.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	1. Use a strong source of randomness to generate a session ID. Use adequate length session IDs. 2. Do not use information available to the user in order to generate session ID (e.g., time)… 3. Encrypt the session ID if you expose it to the user. For instance session ID can be stored in a cookie in encrypted format.
Parent Mitigation	AC, SI, SC

Attack Pattern ID	60
Attack Pattern Name (KOE)	Reusing Session ID’s (aka Session Replay) (6)
Description	This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	1. Always invalidate a session ID after the user logout. 2. Setup a session time out for the session IDs. 3. Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack 4. Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker. 5. Encrypt the session data associated with the session ID. 6. Use multifactor authentication
Parent Mitigation	AC, SI, PS, SC, IA, SA

Attack Pattern ID	61
Attack Pattern Name (KOE)	Session Fixation (3)
Description	The attacker induces a client to establish a session with the target software using a session identifier provided by the attacker. Once the user successfully authenticates to the target software, the attacker uses the (now privileged) session identifier in their own transactions
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	1. Use a strict session management mechanism that only accepts locally generated session identifiers of their own choice. 2. Regenerate and destroy session identifiers when there is a change in the level of privilege: 3. Use session identifiers that are difficult to guess or brute-force:
Parent Mitigation	AC, IA, SC

Attack Pattern ID	62
Attack Pattern Name (KOE)	Cross Site Request Forgery (aka Session Riding) (6)
Description	An attacker crafts malicious web links and distributes them (via web pages, email, etc.), typically in a targeted manner, hoping to induce users to click on the link and execute the malicious action against some third-party application. If successful, the action embedded in the malicious link will be processed and accepted by the targeted application with the users' privilege level. This type of attack leverages the persistence and implicit trust placed in user session cookies by many web applications today. In such an architecture, once the user authenticates to an application and a session cookie is created on the user's system, all following transactions for that session are authenticated using that cookie including potential actions initiated by an attacker and simply "riding" the existing session cookie.
Parent Threat	Exploitation of Authentication
Solutions and Mitigations	1. Use cryptographic tokens to associate a request with a specific action. The token can be regenerated at every request so that if a request with an invalid token is encountered, it can be reliably discarded. The token is considered invalid if it arrived with a request other than the action it was supposed to be associated with. 2. Although less reliable, the use of the optional HTTP Referer header can also be used to determine whether an incoming request was actually one that the user is authorized for, in the current context. 3. Additionally, the user can also be prompted to confirm an action every time an action concerning potentially sensitive data is invoked. This way, even if the attacker manages to get the user to click on a malicious link and request the desired action, the user has a chance to recover by denying confirmation. This solution is also implicitly tied to using a second factor of authentication before performing such actions. 4. In general, every request must be checked for the appropriate authentication token as well as authorization in the current session context.
Parent Mitigation	AC, CA, CM, IA, SC, SI

Attack Pattern ID	63
Attack Pattern Name (KOE)	Simple Script Injection (5)
Description	An attacker embeds malicious scripts in content that will be served to web browsers. The goal of the attack is for the target software, the client-side browser, to execute the script with the users' privilege level.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting. 2. Design: Utilize strict type, character, and encoding enforcement 3. Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from. 4. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 5. Implementation: Perform input validation for all remote content. 6. Implementation: Perform output validation for all remote content. 7. Implementation: Session tokens for specific host 8. Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.
Parent Mitigation	CM, SI, SC, MP, AC

Attack Pattern ID	64
Attack Pattern Name (KOE)	Using Slashes and URL Encoding Combined to Bypass Validation Logic (3)
Description	This attack targets the encoding of the URL combined with the encoding of the slash characters. An attacker can take advantage of the multiple way of encoding an URL and abuse the interpretation of the URL. An URL may contain special character that need special syntax handling in order to be interpreted. Special characters are represented using a percentage character followed by two digits representing the octet code of the original character (%HEX-CODE).
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. 2. When client input is required from web-based forms, avoid using the "GET" method to submit data 3. Any security checks should occur after the data has been decoded and validated as correct data format
Parent Mitigation	SI, AC, CM

Attack Pattern ID	65
Attack Pattern Name (KOE)	Passively Sniff and Capture Application Code bound for Authorized Clients (7)
Description	Attackers can capture application code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server. Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.
Parent Threat	Data Leakage Attacks
Solutions and Mitigations	1. Do not store secrets in client code 2. All potentially sensitive data, including code, transmitted to the client must be encrypted
Parent Mitigation	AT, SA, SC, SI, CA, IA, PL

Attack Pattern ID	66
Attack Pattern Name (KOE)	SQL Injection (3)
Description	This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended.
Parent Threat	Injection
Solutions and Mitigations	1. Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear. 2. Use of parameterized queries or stored procedures - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails. Note that SQL Injection is possible even in the presence of stored procedures if the eventual query is constructed dynamically. 3. Use of custom error pages - Attackers can glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.
Parent Mitigation	SI, AC, MP

Attack Pattern ID	67
Attack Pattern Name (KOE)	String Format Overflow in syslog() (2)
Description	This attack targets the format string vulnerabilities in the syslog() function. An attacker would typically inject malicious input in the format string parameter of the syslog function. This is a common problem, and many public vulnerabilities and associated exploits have been posted.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. The code should be reviewed for misuse of the Syslog function call. Manual or automated code review can be used. The reviewer needs to ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, do not use the %n operator in format strings. The following code shows a correct usage of Syslog(): ... syslog(LOG_ERR, "%s", cmdBuf); ... The following code shows a vulnerable usage of Syslog(): ... syslog(LOG_ERR, cmdBuf); // the buffer cmdBuff is taking user supplied data. ...
Parent Mitigation	SI, AC

Attack Pattern ID	68
Attack Pattern Name (KOE)	Subvert Code-signing Facilities (1)
Description	Because languages use code signing facilities to vouch for code's identity and to thus tie code to its assigned privileges within an environment, subverting this mechanism can be instrumental in an attacker escalating privilege. Any means of subverting the way that a virtual machine enforces code signing classifies for this style of attack. This pattern does not include circumstances through which a signing key has been stolen.
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. A given code signing scheme may be fallible due to improper use of cryptography 2. avoid reliance on flags or environment variables that are user-controllable
Parent Mitigation	IA

Attack Pattern ID	69
Attack Pattern Name (KOE)	Target Programs with Elevated Privileges (5)
Description	This attack targets programs running with elevated privileges. The attacker would try to leverage a bug in the running program and get arbitrary code to execute with elevated privileges. For instance an attacker would look for programs that write to the system directories or registry keys (such as HKLM, which stores a number of critical Windows environment variables).
Parent Threat	Exploitation of Privilege/Trust
Solutions and Mitigations	1. Apply the principle of least privilege. 2. Validate all untrusted data 3. Apply the latest patches. 4. Scan your services and disable the ones which are not needed and are exposed unnecessarily. 5. Avoid revealing information about your system (e.g., version of the program) to anonymous users. 6. Make sure that your program or service fail safely. 7. If possible use a sandbox model which limits the actions that programs can take. 8. Check your program for buffer overflow and format String vulnerabilities which can lead to execution of malicious code. 9. Monitor traffic and resource usage and pay attention if resource exhaustion occurs. 10. Protect your log file from unauthorized modification and log forging.
Parent Mitigation	AC,SI,RA,PS,SC

Attack Pattern ID	70
Attack Pattern Name (KOE)	Try Common(default) Usernames and Passwords (2)
Description	An attacker may try certain common (default) usernames and passwords to gain access into the system and perform unauthorized actions. An attacker may try an intelligent brute force using known vendor default credentials as well as a dictionary of common usernames and passwords.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Delete all default account credentials that may be put in by the product vendor. 2. Implement a password throttling mechanism. 3. Put together a strong password policy and make sure that all user created passwords comply with it. 4. Passwords need to be recycled to prevent aging, that is every once in a while a new password must be chosen.
Parent Mitigation	AC,IA

Attack Pattern ID	71
Attack Pattern Name (KOE)	Using Unicode encoding to Bypass Validation Logic (3)
Description	An attacker may provide a Unicode string to a system component that is not Unicode aware and use that to circumvent the filter or cause the classifying mechanism to fail to properly understanding the request. That may allow the attacker to slip malicious data past the content filter and/or possibly cause the application to route the request incorrectly.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Ensure that the system is Unicode aware and can properly process Unicode data. Do not make an assumption that data will be in ASCII. 2. Ensure that filtering or input validation is applied to canonical data 3. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against white list should not be permitted to enter the system.
Parent Mitigation	AC, SI, CM

Attack Pattern ID	72
Attack Pattern Name (KOE)	URL encoding (8)
Description	This attack targets the encoding of the URL. An attacker can take advantage of the multiple way of encoding an URL and abuse the interpretation of the URL. An URL may contain special character that need special syntax handling in order to be interpreted. Special characters are represented using a percentage character followed by two digits representing the octet code of the original character (%HEX-CODE). For instance US-ASCII space character would be represented with %20. This is often referred as escaped ending or percent-encoding. Since the server decodes the URL from the requests, it may restrict the access to some URL paths by validating and filtering out the URL requests it received. An attacker will try to craft an URL with a sequence of special characters which once interpreted by the server will be equivalent to a forbidden URL. It can be difficult to protect against this attack since the URL can contain other format of encoding such as UTF-8 encoding, Unicode-encoding, etc. The attacker could also subvert the meaning of the URL string request by encoding the data being sent to the server through a GET request. For instance an attacker may subvert the meaning of parameters used in a SQL request and sent through the URL string (See Example section).
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Refer to the RFCS to safely decode URL 2. Regular expression can be used to match safe URL patterns. May discard valid patterns if too restrictive. 3. Tools available to scan HTTP requests to the server 4. Security checks should occur after data is decoded and validated for format. Bad chars result in validation failure. 5. Assume all input is malicious. Create a white list of acceptable input. Test it yourself. 6. Be aware of alternative encoding such as IP encoding 7. In web-forms, avoid using “Get” and use “Post” when possible
Parent Mitigation	AC, CM, SA, SI, SC, CA, PL

Attack Pattern ID	73
Attack Pattern Name (KOE)	User-controlled Filename (4)
Description	An attack of this type involves an attacker inserting malicious characters (such as a XSS redirection) into a filename, directly or indirectly that is then used by the target software to generate HTML text or other potentially executable content. Many websites rely on user-generated content and dynamically build resources like files, filenames, and URL links directly from user supplied data. In this attack pattern, the attacker uploads code that can execute in the client browser and/or redirect the client browser to a site that the attacker owns. All XSS attack payload variants can be used to pass and exploit these vulnerabilities.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Use browser technologies that do not allow client side script 2. Ensure all content delivered to client is sanitized 3. Validate input for all remote content 4. Validate output for all remote content 5. Disable scripts in browser 6. Scan dynamically generated content 7. Disable scripts in browser
Parent Mitigation	AC, CM, MP, SI

Attack Pattern ID	74
Attack Pattern Name (KOE)	Manipulating User State (6)
Description	An attacker modifies state information maintained by the target software in user-accessible locations. If successful, the target software will use this tainted state information and execute in an unintended manner. State management is an important function within an application. User state maintained by the application can include usernames, payment information, browsing history as well as application-specific contents such as items in a shopping cart. Manipulating user state can be employed by an attacker to elevate privilege, conduct fraudulent transactions or otherwise modify the flow of the application to derive certain benefits.
Parent Threat	Time and State Attacks
Solutions and Mitigations	1. Do not rely solely on user-controllable locations, such as cookies or URL parameters, to maintain user state 2. Do not store sensitive information, such as usernames or authentication and authorization information, in user-controllable locations. 3. At all times sensitive information that is part of the user state must be appropriately protected to ensure confidentiality and integrity at each request
Parent Mitigation	CM, CP, IA, MP, SA, SC

Attack Pattern ID	75
Attack Pattern Name (KOE)	Manipulating Writeable Configuration Files (8)
Description	An attacker modifies the contents of configuration files that influence/control the operation of the target software. This attack exploits the ever-growing number, size and complexity of configuration files and the often lax access controls on these files. This attack exploits a program's trust in configuration files that may have weaker permissions. System configuration in distributed systems such as J2EE servers have many administration points. For example, permissions may be set on the administrative GUI, the configuration file for the server as a whole, configuration files for specific domains and applications, special jar and other class files used to load resources at runtime, and even policy specific in .war and .ear files. A mistake in permissions setting in either the file acl or the content is an opening an attacker can use to elevate privilege.
Parent Threat	Exploitation of Privelage/Trust
Solutions and Mitigations	1. Design: Enforce principle of least privilege 2. Design: Backup copies of all configuration files 3. Implementation: Integrity monitoring for configuration files 4. Implementation: Enforce audit logging on code and configuration promotion procedures. 5. Implementation: Load configuration from separate process and memory space, for example a separate physical device like a CD
Parent Mitigation	AC, AU, CA, CM, CP, IR, SC, SI

Attack Pattern ID	76
Attack Pattern Name (KOE)	Manipulating Input to File System Calls(4)
Description	An attacker manipulates inputs to the target software which the target software passes to file system calls in the OS. The goal is to gain access to, and perhaps modify, areas of the file system that the target software did not intend to be accessible.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Design: Enforce principle of least privilege. 2. Design: Ensure all input is validated, and does not contain file system commands 3. Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands. 4. Design: For interactive user applications, consider if direct file system interface is necessary, instead consider having the application proxy communication. 5. Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.
Parent Mitigation	AC, SI, CM, RA

Attack Pattern ID	77
Attack Pattern Name (KOE)	Manipulating User-Controlled Variables (4)
Description	This attack targets user controlled variables (DEBUG=1, PHP Globals, and So Forth). An attacker can override environment variables leveraging user-supplied, untrusted query variables directly used on the application server without any data sanitization. In extreme cases, the attacker can change variables controlling the business logic of the application. For instance, in languages like PHP, a number of poorly set default configurations may allow the user to override variables.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Do not allow override of global variables and do Not Trust Global Variables. If the register_globals option is enabled, PHP will create global variables for each GET, POST, and cookie variable included in the HTTP request. This means that a malicious user may be able to set variables unexpectedly. For instance make sure that the server setting for PHP does not expose global variables. 2. A software system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary. 3. Separate the presentation layer and the business logic layer. Variables at the business logic layer should not be exposed at the presentation layer. This is to prevent computation of business logic from user controlled input data. 4. Use encapsulation when declaring your variables. This is to lower the exposure of your variables. 5. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should be rejected by the program.
Parent Mitigation	CM, SI, SC, AC

Attack Pattern ID	78
Attack Pattern Name (KOE)	Using Escaped Slashes in Alternate Encoding (5)
Description	This attack targets the use of the backslash in alternate encoding. An attacker can provide a backslash as a leading character and causes a parser to believe that the next character is special. This is called an escape. By using that trick, the attacker tries to exploit alternate ways to encode the same character which leads to filter problems and opens avenues to attack.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Verify that the user-supplied data does not use backslash character to escape malicious characters. 2. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system. 3. Be aware of the threat of alternative method of data encoding. 4. Regular expressions can be used to filter out backslash. Make sure you decode before filtering and validating the untrusted input data. 5. In the case of path traversals, use the principle of least privilege when determining access rights to file systems. Do not allow users to access directories/files that they should not access. 6. Any security checks should occur after the data has been decoded and validated as correct data format. Do not repeat decoding process, if bad character are left after decoding process, treat the data as suspicious, and fail the validation process. 7. Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.
Parent Mitigation	SI, MA, AC, CM, SC

Attack Pattern ID	79
Attack Pattern Name (KOE)	Using Slashes in Alternate Encoding (3)
Description	This attack targets the encoding of the Slash characters. An attacker would try to exploit common filtering problems related to the use of the slashes characters to gain access to resources on the target host. Directory-driven systems, such as file systems and databases, typically use the slash character to indicate traversal between directories or other container components. For murky historical reasons, PCs (and, as a result, Microsoft OSs) choose to use a backslash, whereas the UNIX world typically makes use of the forward slash. The schizophrenic result is that many MS-based systems are required to understand both forms of the slash. This gives the attacker many opportunities to discover and abuse a number of common filtering problems. The goal of this pattern is to discover server software that only applies filters to one version, but not the other.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Any security checks should occur after the data has been decoded and validated as correct data format. Do not repeat decoding process, if bad character are left after decoding process, treat the data as suspicious, and fail the validation process. Refer to the RFCs to safelly decode URL. 2. When client input is required from web-based forms, avoid using the “GET” method to submit data, as the method causes the form data to be appended to the URL and is easily manipulated. Instead, use the “POST method whenever possible. 3. There are tools to scan HTTP requests to the server for valid URL such as URLScan from Microsoft (http://www.microsoft.com/technet/security/tools/urlscan.mspx) 4. Be aware of the threat of alternative method of data encoding and obfuscation technique such as IP address endoding. (See related guideline section) 5. Test your path decoding process against malicious input. 6. In the case of path traversals, use the principle of least privilege when determining access rights to file systems. Do not allow users to access directories/files that they should not access. 7. Assume all input is malicious. Create a white list that defines all valid input to the application based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.
Parent Mitigation	SI, SC, AC

Attack Pattern ID	80
Attack Pattern Name (KOE)	Using UTF-8 Encoding to Bypass Validation Logic (1)
Description	This attack is a specific variation on leveraging alternate encodings to bypass validation logic. This attack leverages the possibility to encode potentially harmful input in UTF-8 and submit it to applications not expecting or effective at validating this encoding standard making input filtering difficult. UTF-8 (8-bit UCS/Unicode Transformation Format) is a variable-length character encoding for Unicode. Legal UTF-8 characters are one to four bytes long. However, early version of the UTF-8 specification got some entries wrong (in some cases it permitted overlong characters). UTF-8 encoders are supposed to use the ``shortest possible" encoding, but naive decoders may accept encodings that are longer than necessary. According to the RFC 3629, a particularly subtle form of this attack can be carried out against a parser which performs security-critical validity checks against the UTF-8 encoded form of its input, but interprets certain illegal octet sequences as characters.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. The Unicode Consortium recognized multiple representations to be a problem and has revised the Unicode Standard to make multiple representations of the same code point with UTF-8 illegal. 2. For security reasons, a UTF-8 decoder must not accept UTF-8 sequences that are longer than necessary to encode a character. If you use a parser to decode the UTF-8 encoding, make sure that parser filter the invalid UTF-8 characters (invalid forms or overlong forms). 3. Look for overlong UTF-8 sequences starting with malicious pattern. You can also use a UTF-8 decoder stress test to test your UTF-8 parser (See Markus Kuhn's UTF-8 and Unicode FAQ in reference section) 4. Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system. Test your decoding process against malicious input.
Parent Mitigation	SI

Attack Pattern ID	81
Attack Pattern Name (KOE)	Web Logs Tampering (3)
Description	Protection services in security are vulnerable so they are backstopped by detection in the so-called protect-detect-respond model. A key element in detection is log files, to identify a threat impact, for audit purposes, or simply responding to a crash. Since most requests to web servers are logged (at least header request response data) the attacker literally has the ability to generate log data in every request Web Logs Tampering attacks involve an attacker injecting, deleting or otherwise tampering with the contents of web logs. Additionally, writing malicious data to log files may target jobs, filters, reports, and other agents that process the logs in an asynchronous attack pattern.
Parent Threat	Resource Location Attacks
Solutions and Mitigations	1. Design: Use input validation before writing to web log 2. Design: Validate all log data before it is output
Parent Mitigation	AC, AU, SI

Attack Pattern ID	82
Attack Pattern Name (KOE)	Violating Implicit Assumptions Regarding XML Content (aka XMl Denial of Service (XDoS)) (5)
Description	XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space.
Parent Threat	Resource Depletion
Solutions and Mitigations	1. Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads. 2. Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers 3. Implementation: Check size of XML message before parsing
Parent Mitigation	SC, IR, PE, RA, SC

Attack Pattern ID	83
Attack Pattern Name (KOE)	XPath Injection (2)
Description	An attacker can craft special user-controllable input consisting of XPath expressions to inject the XML database and bypass authentication or glean information that he normally would not be able to. XPath Injection enables an attacker to talk directly to the XML database, thus bypassing the application completely. XPath Injection results form the failure of an application to properly sanitize input used as part of dynamic XPath expressions used to query an XML database.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as content that can be interpreted in the context of an XPath expression. Characters such as a single-quote(') or operators such as or (|), and (&) and such should be filtered if the application does not expect them in the context in which they appear. If such content cannot be filtered, it must at least be properly escaped to avoid them being interpreted as part of XPath expressions. 2. Use of parameterized XPath queries - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails. 3. Use of custom error pages - Attackers can glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.
Parent Mitigation	SC, SI

Attack Pattern ID	84
Attack Pattern Name (KOE)	XQuery Injection (3)
Description	This attack utilizes XQuery to probe and attack server systems; in a similar manner that SQL Injection allows an attacker to exploit SQL calls to RDBMS, XQuery Injection uses improperly validated data that is passed to XQuery commands to traverse and execute commands that the XQuery routines have access to. XQuery injection can be used to enumerate elements on the victim's environment, inject commands to the local host, or execute queries to remote files and data sources.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Perform input white list validation on all XML input 2. Implementation: Run xml parsing and query infrastructure with minimal privileges so that an attacker is limited in their ability to probe other system resources from xql.
Parent Mitigation	SI, SA, AC

Attack Pattern ID	85
Attack Pattern Name (KOE)	Client Network Footprinting (using AJAX/XSS) (4)
Description	This attack utilizes the frequent client-server roundtrips in Ajax conversation to scan a system. While Ajax does not open up new vulnerabilities per se, it does optimize them from an attacker point of view. In many XSS attacks the attacker must get a "hole in one" and successfully exploit the vulnerability on the victim side the first time, once the client is redirected the attacker has many chances to engage in follow on probes, but their is only one first chance. In a widely used web application this is not a major problem because 1 in a 1,000 is good enough in a widely used application. A common first step for an attacker is to footprint the environment to understand what attacks will work. Since footprinting relies on enumeration, the conversational pattern of rapid, multiple requests and responses that are typical in Ajax applications enable an attacker to look for many vulnerabilities, well known ports, network locations and so on.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting 2. Design: Utilize strict type, character, and encoding enforcement 3. Implementation: Perform input validation for all remote content. 4. Implementation: Perform output validation for all remote content. 5. Implementation: Disable scripting languages such as Javascript in browser 6. Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.
Parent Mitigation	AC, SC, SI, RA

Attack Pattern ID	86
Attack Pattern Name (KOE)	Embedding Script (XSS) in HTTP Headers (4)
Description	An attack of this type exploits web applications that generate web content, such as links in a HTML page, based on unvalidated or improperly validated data submitted by other actors. XSS in HTTP Headers attacks target the HTTP headers which are hidden from most users and may not be validated by web applications. As with all XSS attacks, there are a number of possible targets: 1. Launch attack on web browser clients and client machine 2. Launch attacks on client machines environment, such as LAN or Intranet 3. Launch attack on web server, including remote web servers Web 2.0 technologies rely heavily on mashups and other plug in technologies like multi media players which are effectively composed of content generated by other systems and are vulnerable due to the fact that an attacker may use the HTTP header information that these technologies consume and display as an attack launch pad. Beyond Web 2.0, increasingly system administration software uses web front ends, from firewall administration to application servers, to blogging software, many tools are administered through web browsers. This gives the administrator the ability to administer in a highly distributed environment, but this comes at the cost of exposing the command and control software for the system to web attacks. Additionally, because the rich functionality required these administration applications, many rely on scripting languages. So an attacker can insert HTTP links into logs, audit functionality, error logs, and message queues, then, for example, a Javascript-enabled web browser with administrator rights can be redirected to execute a wide variety of attacks, including those listed here.
Parent Threat	Injection
Solutions and Mitigations	1. Design: Use browser technologies that do not allow client side scripting. 2. Design: Utilize strict type, character, and encoding enforcement 3. Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from. 4. Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification. 5. Implementation: Perform input validation for all remote content. 6. Implementation: Perform output validation for all remote content. 7. Implementation: Disable scripting languages such as Javascript in browser 8. Implementation: Session tokens for specific host 9. Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.
Parent Mitigation	AC, SC, SI, RA

Attack Pattern ID	87
Attack Pattern Name (KOE)	Forceful Browsing (3)
Description	An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context. 2. Forceful browsing can also be made difficult to a large extent by not hard-coding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.
Parent Mitigation	AC, IA, SC

Attack Pattern ID	88
Attack Pattern Name (KOE)	OS Command Injection (3)
Description	An attacker can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Use language APIs rather than relying on passing data to the operating system shell or command line. Doing so ensures that the available protection mechanisms in the language are intact and applicable. 2. Filter all incoming data to escape or remove characters or strings that can be potentially misinterpreted as operating system or shell commands 3. All application processes should be run with the minimal privileges required. Also, processes must shed privileges as soon as they no longer require them.
Parent Mitigation	SI, AC, CM

Attack Pattern ID	89
Attack Pattern Name (KOE)	Pharming (8)
Description	Pharming attacks occur when victims provide sensitive information to websites that do not possess a valid certificate from well-known certificate authorities.
Parent Threat	Spoofing
Solutions and Mitigations	1. All sensitive information must be handled over a secure connection. 2. Known vulnerabilities in DNS or router software or in operating systems must be patched as soon as a fix has been released and tested. 3. End users must ensure that they provide sensitive information only to websites that they trust, over a secure connection with a valid certificate issued by a well-known certificate authority.
Parent Mitigation	AC, CA, CM, CP, IA, RA,SI, SC

Attack Pattern ID	90
Attack Pattern Name (KOE)	Reflection Attack in Authentication Protocol (4)
Description	A single sign-on solution for a network uses a fixed preshared key with its clients to initiate the signon process in order to avoid eavesdropping on the initial exchanges.
Parent Threat	Exploitation of Privilege or Trust
Solutions and Mitigations	1. The server must initiate the handshake by issuing the challenge. This ensures that the client has to respond before the exchange can move any further. 2. The use of HMAC to hash the response from the server can also be used to thwart reflection. The server responds by returning its own challenge as well as hashing the client's challenge, its own challenge and the preshared secret. Requiring the client to respond with the HMAC of the two challenges ensures that only the possessor of a valid preshared secret can successfully hash in the two values. 3. Introducing a random nonce with each new connection ensures that the attacker can not employ two connections to attack the authentication protocol
Parent Mitigation	AC, IA, SC, SI

Attack Pattern ID	91
Attack Pattern Name (KOE)	XSS in IMG Tags (1)
Description	Image tags are an often overlooked, but convenient, means for a Cross Site Scripting attack. The attacker can inject script contents into an image (IMG) tag in order to steal information from a victim's browser and execute malicious scripts.
Parent Threat	Injection
Solutions and Mitigations	1. In addition to the traditional input fields, all other user controllable inputs, such as image tags within messages or the likes, must also be subjected to input validation. Such validation should ensure that content that can be potentially interpreted as script by the browser is appropriately filtered. 2. All output displayed to clients must be properly escaped. Escaping ensures that the browser interprets special scripting characters literally and not as script to be executed
Parent Mitigation	SI

Attack Pattern ID	92
Attack Pattern Name (KOE)	Forced Integer Overflow (4)
Description	This attack forces an integer variable to go out of range. The integer variable is often used as an offset such as size of memory allocation or similarly. The attacker would typically control the value of such variable and try to get it out of range. For instance the integer in question is incremented past the maximum possible value, it may wrap to become a very small, or negative number, therefore providing a very incorrect value which can lead to unexpected behavior. At worst the attacker can execute arbitrary code.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Use a language or compiler that performs automatic bounds checking. 2. Carefully review the service's implementation before making it available to user. For instance you can use manual or automated code review to uncover vulnerabilities such as integer overflow. 3. Use an abstraction library to abstract away risky APIs. Not a complete solution. 4. Always do bound checking before consuming user input data.
Parent Mitigation	CA, RA, SC, SI

Attack Pattern ID	93
Attack Pattern Name (KOE)	Log Injection-Tampering-Forging(5)
Description	This attack targets the log files of the target host. The attacker injects, manipulates or forges malicious log entries in the log file, allowing him to mislead a log audit, cover traces of attack, or perform other malicious actions. The target host is not properly controlling log access. As a result tainted data is resulting in the log files leading to a failure in accoutability, non-repudiation and incident forensics capability.
Parent Threat	Audit Log Manipulation
Solutions and Mitigations	1. Carefully control access to physical log files. 2. Do not allow tainted data to be written in the log file without prior input validation. Whitelisting may be used to properly validate the data. 3. Use synchronization to control the flow of execution. 4. Use static analysis tools to identify log forging vulnerabilities. 5. Avoid viewing logs with tools that may interpret control characters in the file, such as command-line shells.
Parent Mitigation	AC, IA, SC, AU, RA

Attack Pattern ID	94
Attack Pattern Name (KOE)	Man in the Middle (3)
Description	This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.
Parent Threat	Spoofing
Solutions and Mitigations	1. Get your Public Key signed by a Certificate Authority 2. Encrypt your communication using cryptography (SSL,...) 3. Use Strong mutual authentication to always fully authenticate both ends of any communications channel. 4. Exchange public keys using a secure channel
Parent Mitigation	AC, IA, SC

Attack Pattern ID	95
Attack Pattern Name (KOE)	WSDL Scanning (3)
Description	This attack targets the WSDL interface made available by a web service. The attacker may scan the WSDL interface to reveal sensitive information about invocation patterns, underlying technology implementations and associated vulnerabilities. This type of probing is carried out to perform more serious attacks (e.g. parameter tampering, malicious content injection, command injection, etc.). WSDL files provide detailed information about the services ports and bindings available to consumers. For instance, the attacker can submit special characters or malicious content to the Web service and can cause a denial of service condition or illegal access to database records. In addition, the attacker may try to guess other private methods by using the information provided in the WSDL files.
Parent Threat	Abuse of Functionality
Solutions and Mitigations	1. It is important to protect WSDL file or provide limited access to it. 2. Review the functions exposed by the WSDL interface (specially if you have used a tool to generate it). Make sure that none of them is vulnerable to injection. 3. Ensure the WSDL does not expose functions and APIs that were not intended to be exposed. 4. Pay attention to the function naming convention (within the WSDL interface). Easy to guess function name may be an entry point for attack. 5. Validate the received messages against the WSDL Schema. Incomplete solution
Parent Mitigation	AC, SI, AU

Attack Pattern ID	96
Attack Pattern Name (KOE)	Block Access to Libraries (5)
Description	An application typically makes calls to functions that are a part of libraries external to the application. These libraries may be part of the operating system or they may be third party libraries. It is possible that the application does not handle situations properly where access to these libraries has been blocked. Depending on the error handling within the application, blocked access to libraries may leave the system in an insecure state that could be leveraged by an attacker.
Parent Threat	Resource Manipulation
Solutions and Mitigations	1. Ensure that application handles situations where access to APIs in external libraries is not available securely. If the application cannot continue its execution safely it should fail in a consistent and secure fashion.
Parent Mitigation	CM, SA, SC, SI, RA

Attack Pattern ID	97
Attack Pattern Name (KOE)	Cryptanalysis (2)
Description	Cryptanalysis is a process of finding weaknesses in cryptographic algorithms and using these weaknesses to decipher the ciphertext without knowing the secret key (instance deduction). Sometimes the weakness is not in the cryptographic algorithm itself, but rather in how it is applied that makes cryptanalysis successful. An attacker may have other goals as well, such as: 1. Total Break - Finding the secret key 2. Gobal Deduction - Finding a functionally equivalent algorithm for encryption and decryption that does not require knowledge of the secret key. 3. Information Deduction - Gaining some information about plaintexts or ciphertexts that was not previously known 4. Distinguishing Algorithm - The attacker has the ability to distinguish the output of the encryption (ciphertext) from a random permutation of bits The goal of the attacker performing cryptanalysis will depend on the specific needs of the attacker in a given attack context. In most cases, if cryptanalysis is successful at all, an attacker will not be able to go past being able to deduce some information about the plaintext (goal 3). However, that may be sufficient for an attacker, depending on the context.
Parent Threat	Probabilistic Techniques
Solutions and Mitigations	1. Use proven cryptographic algorithms with recommended key sizes. 2. Ensure that the algorithms are used properly. That means: 1. Not rolling out your own crypto; Use proven algorithms and implementations. 2. Choosing initialization vectors with sufficiently random numbers 3. Generating key material using good sources of randomness and avoiding known weak keys 4. Using proven protocols and their implementations. 5. Picking the most appropriate cryptographic algorithm for your usage context and data
Parent Mitigation	IA, SC

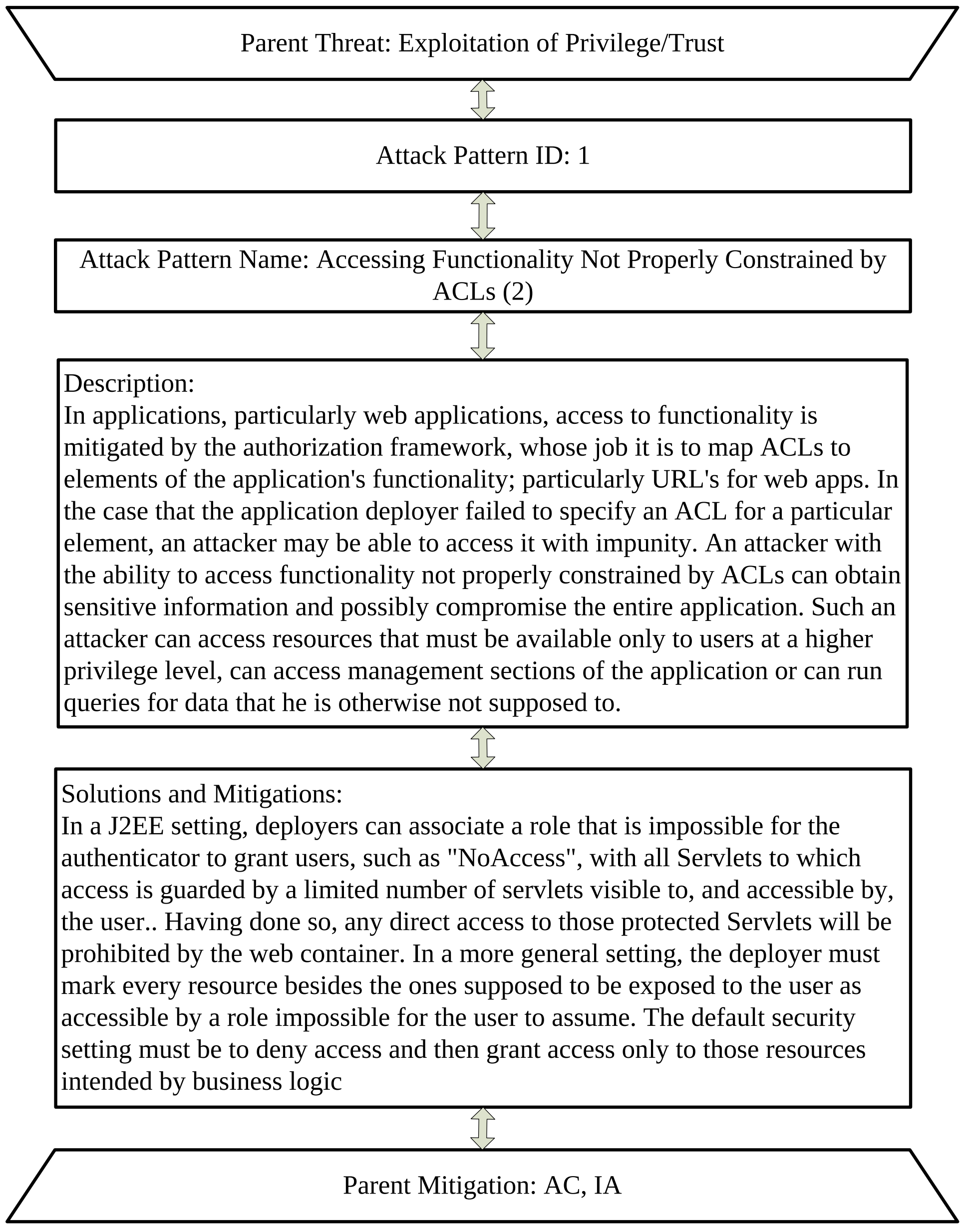
Attack Pattern ID	98
Attack Pattern Name (KOE)	Phishing (4)
Description	Phishing is a social engineering technique where an attacker masquerades as a legitimate entity with which the victim might do business in order to prompt the user to reveal some confidential information (very frequently authentication credentials) that can later be used by an attacker. Phishing is essentially a form of information gathering or "fishing" for information.
Parent Threat	Spoofing
Solutions and Mitigations	1. Do not follow any links that you receive within your e-mails and certainly do not input any login credentials on the page that they take you too. Instead, call your Bank, Paypal, Ebay, etc., and inquire about the problem. A safe practice would also be to type the URL of your bank in the browser directly and only then log in. Also, never reply to any e-mails that ask you to provide sensitive information of any kind.
Parent Mitigation	AT, SA, SI, PL

Attack Pattern ID	99
Attack Pattern Name (KOE)	XML Parser Attack (3)
Description	Applications often need to transform data in and out of the XML format by using an XML parser. It may be possible for an attacker to inject data that may have an adverse effect on the XML parser when it is being processed. These adverse effects may include the parser crashing, consuming too much of a resource, executing too slowly, executing code supplied by an attacker, allowing usage of unintenteded system functionality, etc. An attacker's goal is to leverage parser failure to his or her advantage. In some cases it may be possible to jump from the data plane to the control plane via bad data being passed to an XML parser [1].
Parent Threat	Resource Depletion
Solutions and Mitigations	1. Carefully validate and sanitize all user-controllable data prior to passing it to the XML parser routine. Ensure that the resultant data is safe to pass to the XML parser. 2. Perform validation on canonical data. 3. Pick a robust implementation of an XML parser. 4. Validate XML against a valid schema or DTD prior to parsing.
Parent Mitigation	IR, SA, SI

Attack Pattern ID	100
Attack Pattern Name (KOE)	Overflow Buffers (2)
Description	Buffer Overflow attacks target improper or missing bounds checking on buffer operations, typically triggered by input injected by an attacker. As a consequence, an attacker is able to write past the boundaries of allocated buffer regions in memory, causing a program crash or potentially redirection of execution as per the attacker's choice.
Parent Threat	Data Structure Attacks
Solutions and Mitigations	1. Use a language or compiler that performs automatic bounds checking. 2. Use secure functions not vulnerable to buffer overflow. 3. If you have to use dangerous functions, make sure that you do boundary checking. 4. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. 5. Use OS-level preventative functionality. Not a complete solution. 6. Utilize static source code analysis tools to identify potential buffer overflow weaknesses in the software.
Parent Mitigation	SC,SI

Attack Pattern ID	101
Attack Pattern Name (KOE)	(SSI) Server Side Include Injection (4)
Description	Consider a website hosted on a server that permits Server Side Includes (SSI), such as Apache with the "Options Includes" directive enabled. Whenever an error occurs, the HTTP Headers along with the entire request are logged, which can then be displayed on a page that allows review of such errors. A malicious user can inject SSI directives in the HTTP Headers of a request designed to create an error. When these logs are eventually reviewed, the server parses the SSI directives and executes them.
Parent Threat	Injection (Injecting Control Plane content through the Data Plane)
Solutions and Mitigations	1. Set the OPTIONS IncludesNOEXEX in the global access.conf file or local .htaccess (apache) file to deny SSI execution in directories that do not need them 2. All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive 3. Server Side Includes must be enabled only if there is a strong business reason to do so. Every Additional component enabled on the web server increases the attack surface as well as administrative overhead
Parent Mitigation	SI, CM, RA, SA

Graphical Attack Trees



Parent Threat: Exploitation of Privilege/Trust

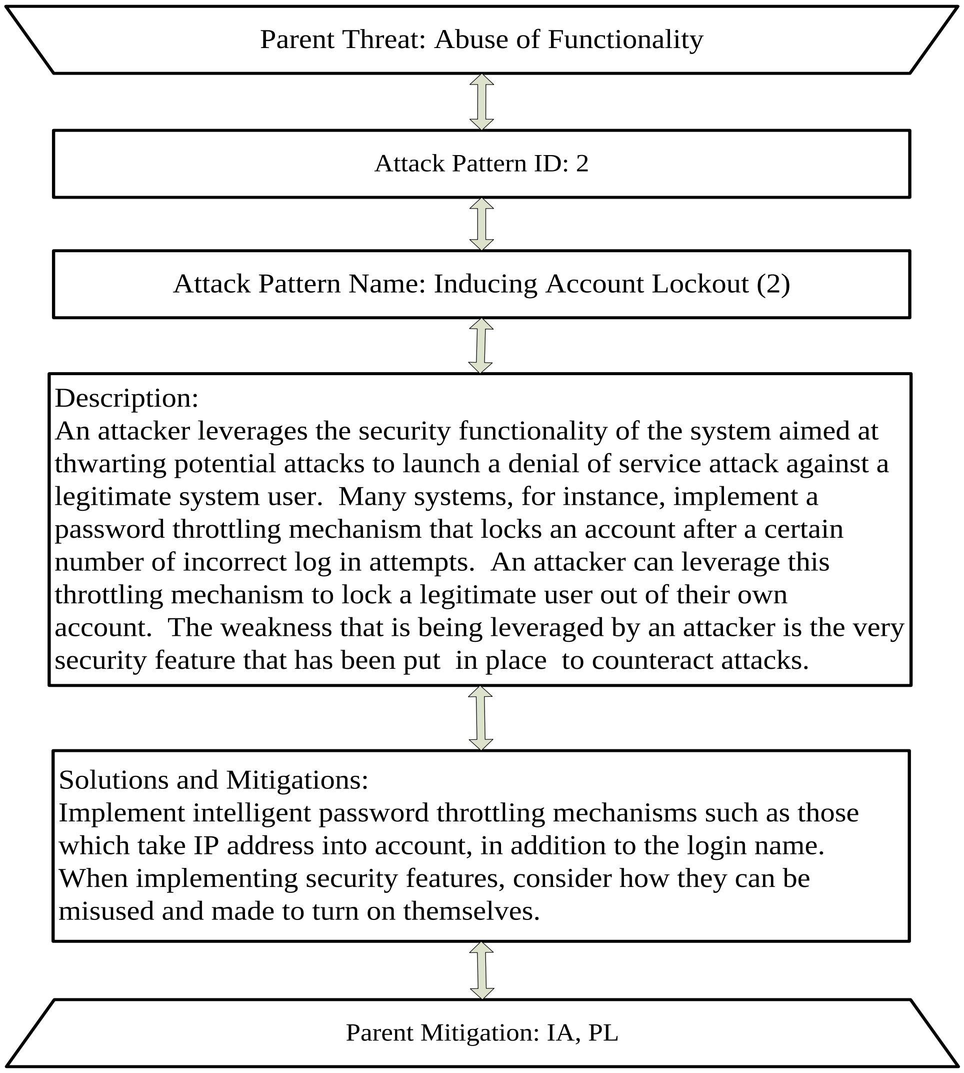
Attack Pattern ID: 1

Attack Pattern Name: Accessing Functionality Not Properly Constrained by ACLs (2)

Description:
In applications, particularly web applications, access to functionality is mitigated by the authorization framework, whose job it is to map ACLs to elements of the application's functionality; particularly URL's for web apps. In the case that the application deployer failed to specify an ACL for a particular element, an attacker may be able to access it with impunity. An attacker with the ability to access functionality not properly constrained by ACLs can obtain sensitive information and possibly compromise the entire application. Such an attacker can access resources that must be available only to users at a higher privilege level, can access management sections of the application or can run queries for data that he is otherwise not supposed to.

Solutions and Mitigations:
In a J2EE setting, deployers can associate a role that is impossible for the authenticator to grant users, such as "NoAccess", with all Servlets to which access is guarded by a limited number of servlets visible to, and accessible by, the user.. Having done so, any direct access to those protected Servlets will be prohibited by the web container. In a more general setting, the deployer must mark every resource besides the ones supposed to be exposed to the user as accessible by a role impossible for the user to assume. The default security setting must be to deny access and then grant access only to those resources intended by business logic

Parent Mitigation: AC, IA



Parent Threat: Abuse of Functionality

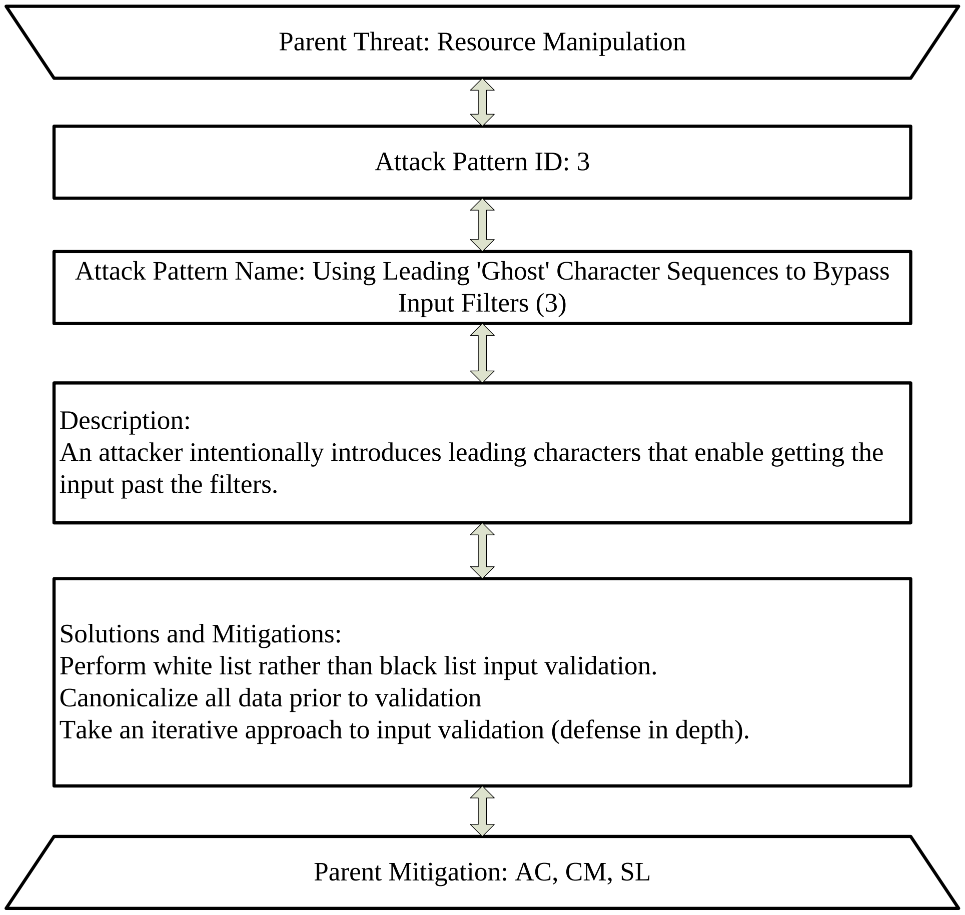
Attack Pattern ID: 2

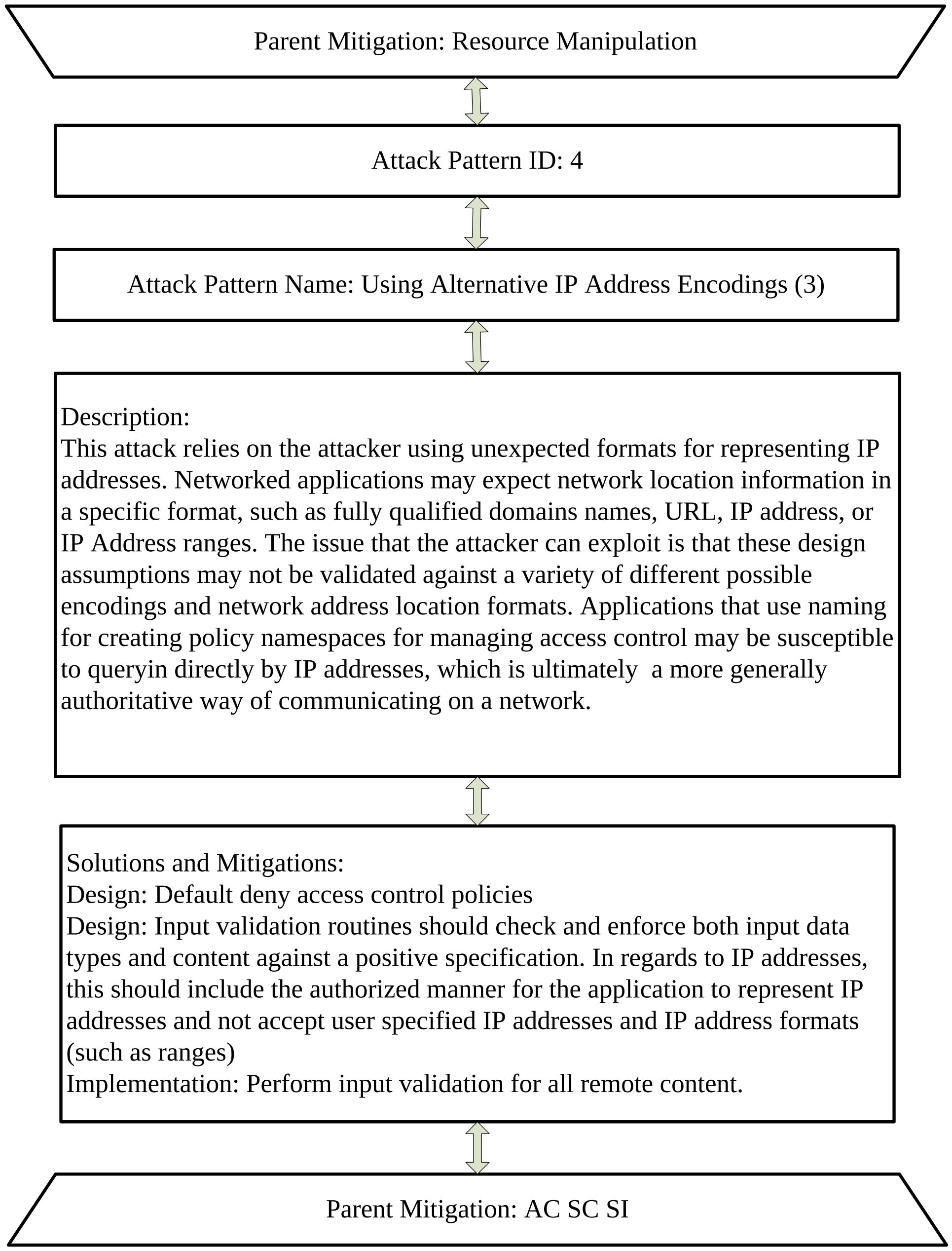
Attack Pattern Name: Inducing Account Lockout (2)

Description:
An attacker leverages the security functionality of the system aimed at thwarting potential attacks to launch a denial of service attack against a legitimate system user. Many systems, for instance, implement a password throttling mechanism that locks an account after a certain number of incorrect log in attempts. An attacker can leverage this throttling mechanism to lock a legitimate user out of their own account. The weakness that is being leveraged by an attacker is the very security feature that has been put in place to counteract attacks.

Solutions and Mitigations:
Implement intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name. When implementing security features, consider how they can be misused and made to turn on themselves.

Parent Mitigation: IA, PL


Parent Threat: Resource Manipulation
Attack Pattern ID: 3
Attack Pattern Name: Using Leading 'Ghost' Character Sequences to Bypass Input Filters (3)
Description:
An attacker intentionally introduces leading characters that enable getting the input past the filters.
Solutions and Mitigations:
Perform white list rather than black list input validation.
Canonicalize all data prior to validation
Take an iterative approach to input validation (defense in depth).
Parent Mitigation: AC, CM, SL



Parent Mitigation: Resource Manipulation

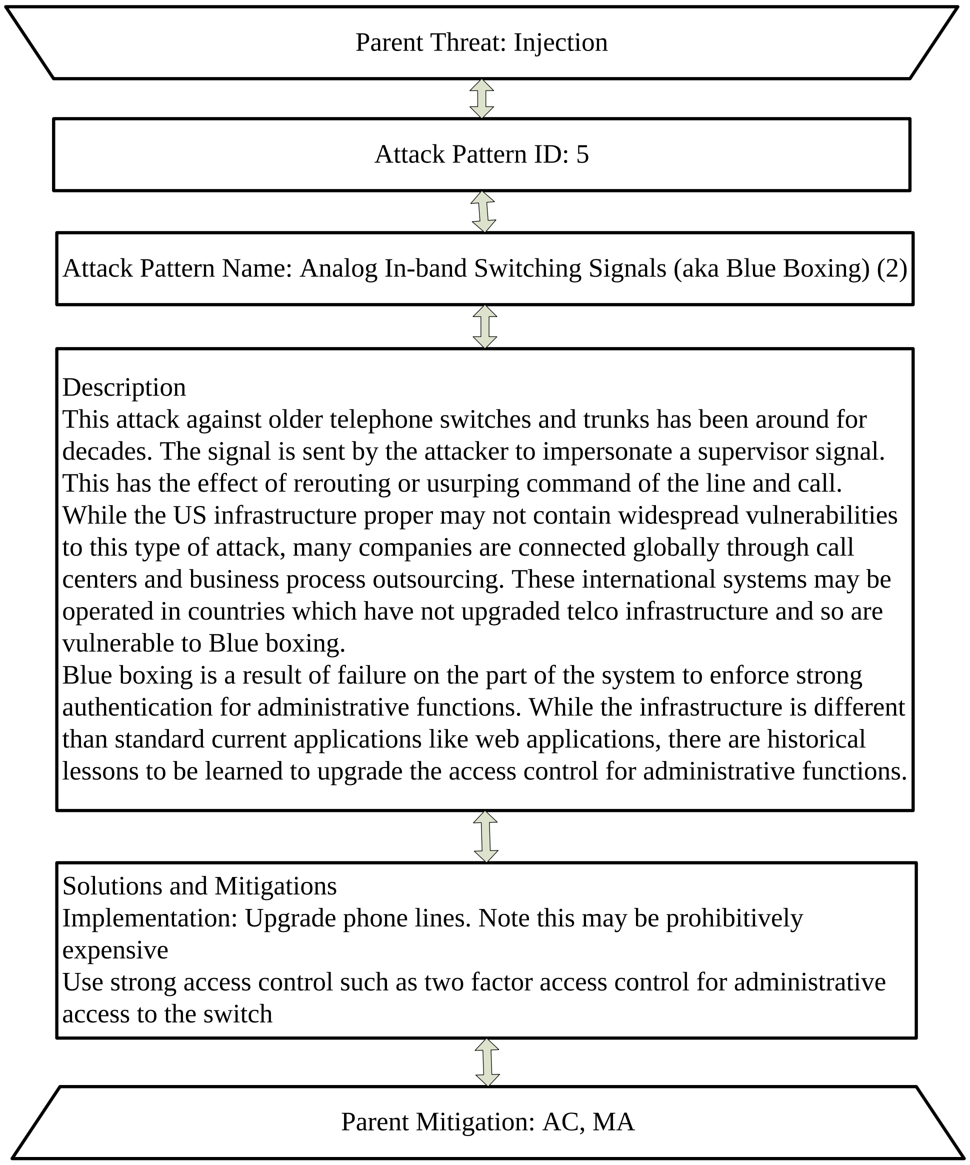
Attack Pattern ID: 4

Attack Pattern Name: Using Alternative IP Address Encodings (3)

Description:
This attack relies on the attacker using unexpected formats for representing IP addresses. Networked applications may expect network location information in a specific format, such as fully qualified domains names, URL, IP address, or IP Address ranges. The issue that the attacker can exploit is that these design assumptions may not be validated against a variety of different possible encodings and network address location formats. Applications that use naming for creating policy namespaces for managing access control may be susceptible to queryin directly by IP addresses, which is ultimately a more generally authoritative way of communicating on a network.

Solutions and Mitigations:
Design: Default deny access control policies
Design: Input validation routines should check and enforce both input data types and content against a positive specification. In regards to IP addresses, this should include the authorized manner for the application to represent IP addresses and not accept user specified IP addresses and IP address formats (such as ranges)
Implementation: Perform input validation for all remote content.

Parent Mitigation: AC SC SI



Parent Threat: Injection

Attack Pattern ID: 5

Attack Pattern Name: Analog In-band Switching Signals (aka Blue Boxing) (2)

Description
This attack against older telephone switches and trunks has been around for decades. The signal is sent by the attacker to impersonate a supervisor signal. This has the effect of rerouting or usurping command of the line and call. While the US infrastructure proper may not contain widespread vulnerabilities to this type of attack, many companies are connected globally through call centers and business process outsourcing. These international systems may be operated in countries which have not upgraded telco infrastructure and so are vulnerable to Blue boxing.
Blue boxing is a result of failure on the part of the system to enforce strong authentication for administrative functions. While the infrastructure is different than standard current applications like web applications, there are historical lessons to be learned to upgrade the access control for administrative functions.

Solutions and Mitigations
Implementation: Upgrade phone lines. Note this may be prohibitively expensive
Use strong access control such as two factor access control for administrative access to the switch

Parent Mitigation: AC, MA

Parent Threat: Injection

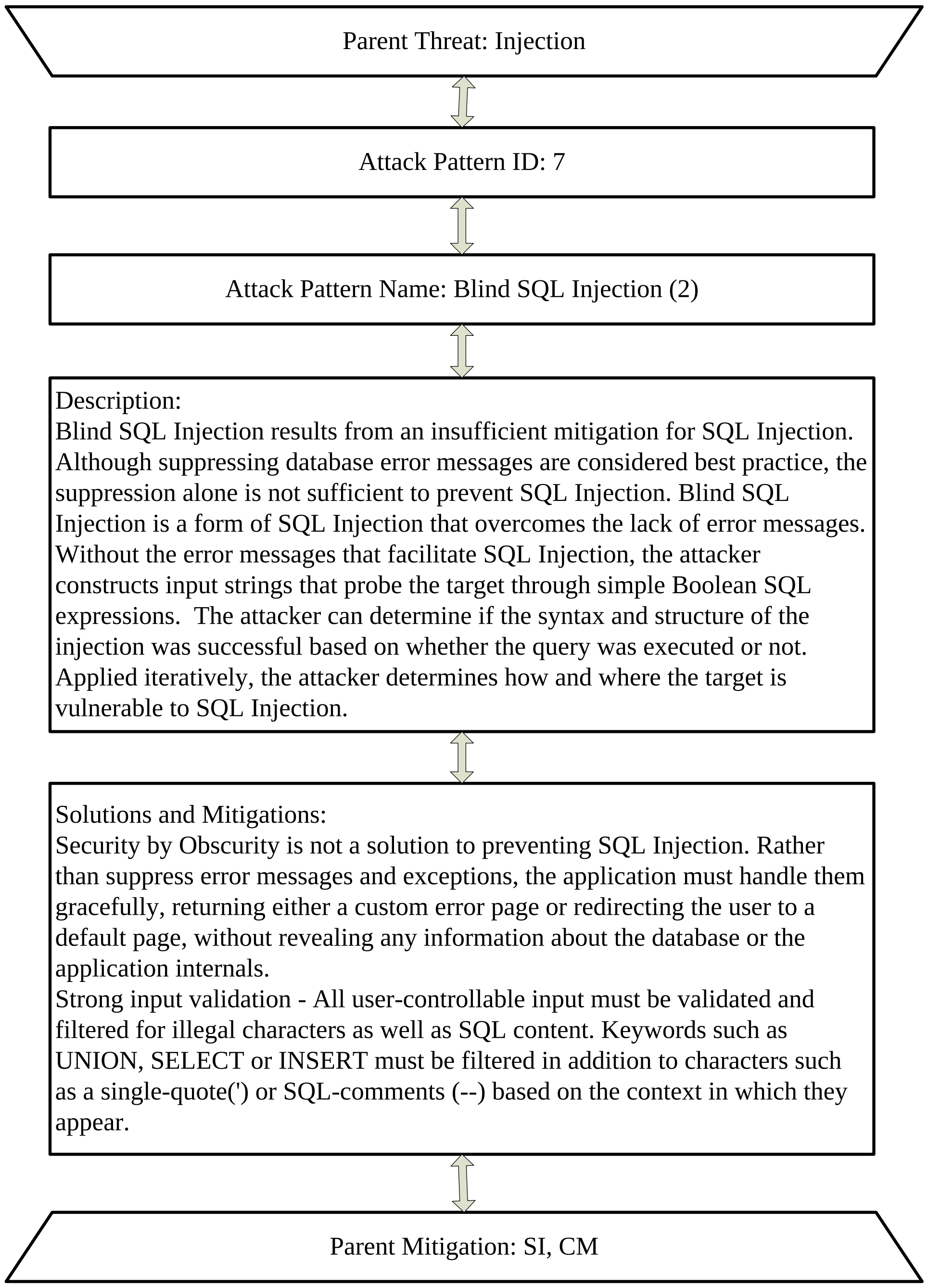
Attack Pattern ID: 6

Attack Pattern Name: Argument Injection (2)

Description
An attack of this type exploits a programs' vulnerabilities that allows an attacker's commands to be directly or indirectly applied as arguments, for example as shell commands. This may allow an attacker access to files, network resources, media, and in short anything accessible through the shell. The argument injection attack uses the exposed service or method as a launch pad to invoke other programs. If the service does not validate or filter the input data then the client program is granted access to execute commands using the server's privileges. The OS commands can be appended to standard input for shell programs, HTTP Requests, and XML messages. The ability to invoke commands is not necessarily sufficient for the attacker to collect the output of the attack. This may or may not be an issue depending on the attacker goal.

Solutions and Mitigations
Design: Do not program input values directly on command shell, instead treat user input as guilty until proven innocent. Build a function that takes user input and converts it to applications specific types and values, stripping or filtering out all unauthorized commands and characters in the process.
Design: Limit program privileges, so if metacharcters or other methods circumvent program input validation routines and shell access is attained then it is not running under a privileged account. chroot jails create a sandbox for the application to execute in, making it more difficult for an attacker to elevate privilege even in the case that a compromise has occurred.
Implementation: Implement an audit log that is written to a separate host, in the event of a compromise the audit log may be able to provide evidence and details of the compromise.

Parent Mitigation: AT, PL



Parent Threat: Injection

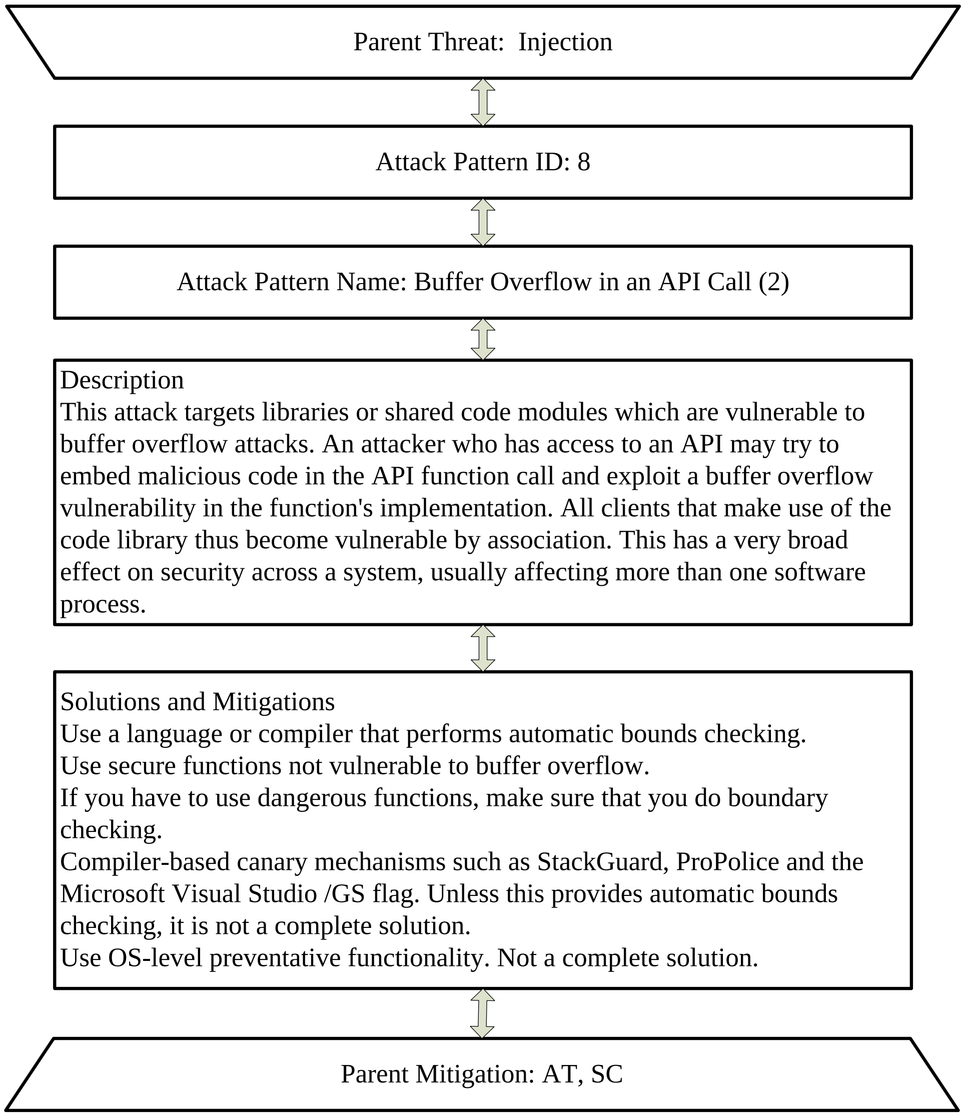
Attack Pattern ID: 7

Attack Pattern Name: Blind SQL Injection (2)

Description:
Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the attacker constructs input strings that probe the target through simple Boolean SQL expressions. The attacker can determine if the syntax and structure of the injection was successful based on whether the query was executed or not. Applied iteratively, the attacker determines how and where the target is vulnerable to SQL Injection.

Solutions and Mitigations:
Security by Obscurity is not a solution to preventing SQL Injection. Rather than suppress error messages and exceptions, the application must handle them gracefully, returning either a custom error page or redirecting the user to a default page, without revealing any information about the database or the application internals.
Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear.

Parent Mitigation: SI, CM



Parent Threat: Injection

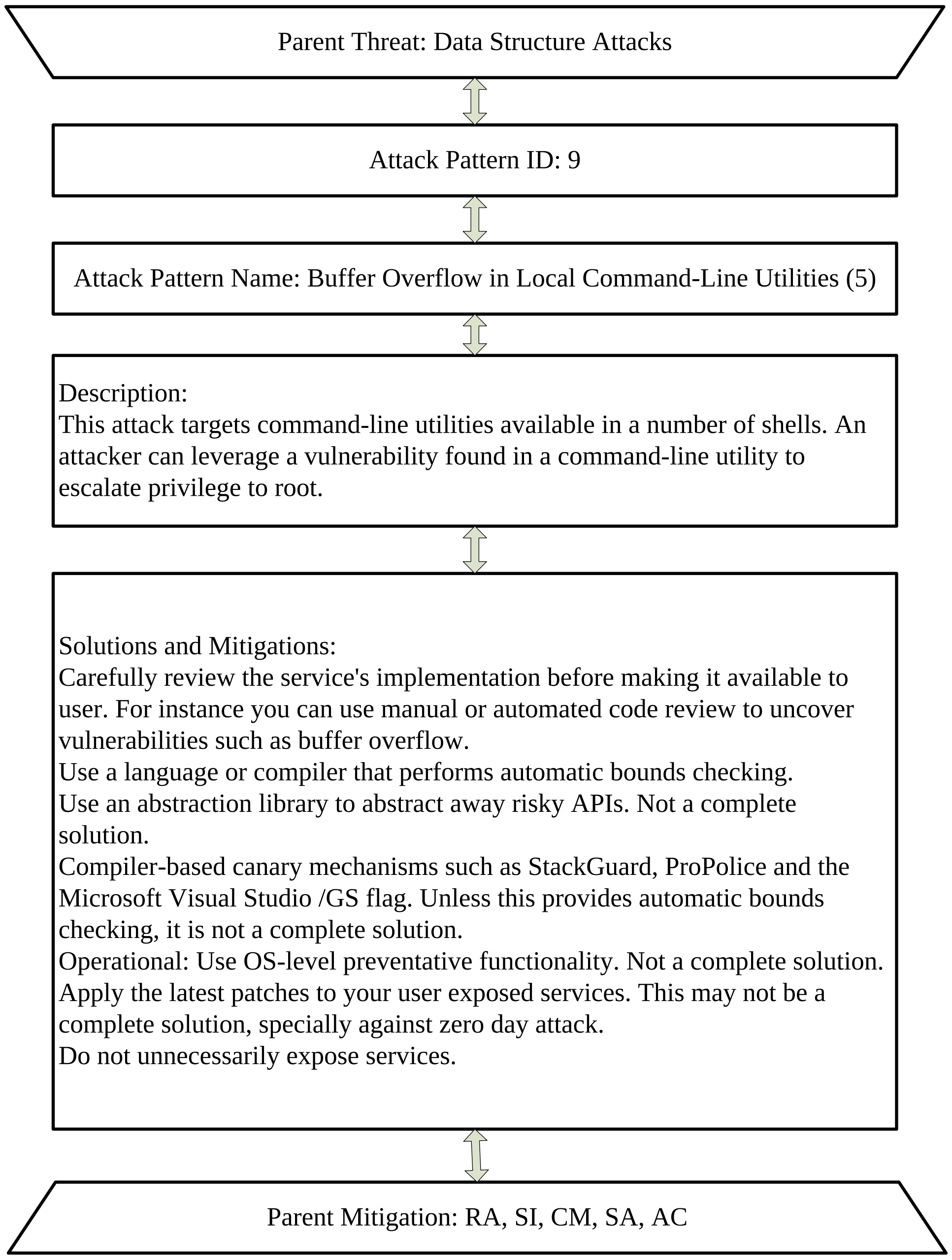
Attack Pattern ID: 8

Attack Pattern Name: Buffer Overflow in an API Call (2)

Description
This attack targets libraries or shared code modules which are vulnerable to buffer overflow attacks. An attacker who has access to an API may try to embed malicious code in the API function call and exploit a buffer overflow vulnerability in the function's implementation. All clients that make use of the code library thus become vulnerable by association. This has a very broad effect on security across a system, usually affecting more than one software process.

Solutions and Mitigations
Use a language or compiler that performs automatic bounds checking.
Use secure functions not vulnerable to buffer overflow.
If you have to use dangerous functions, make sure that you do boundary checking.
Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
Use OS-level preventative functionality. Not a complete solution.

Parent Mitigation: AT, SC



Parent Threat: Data Structure Attacks

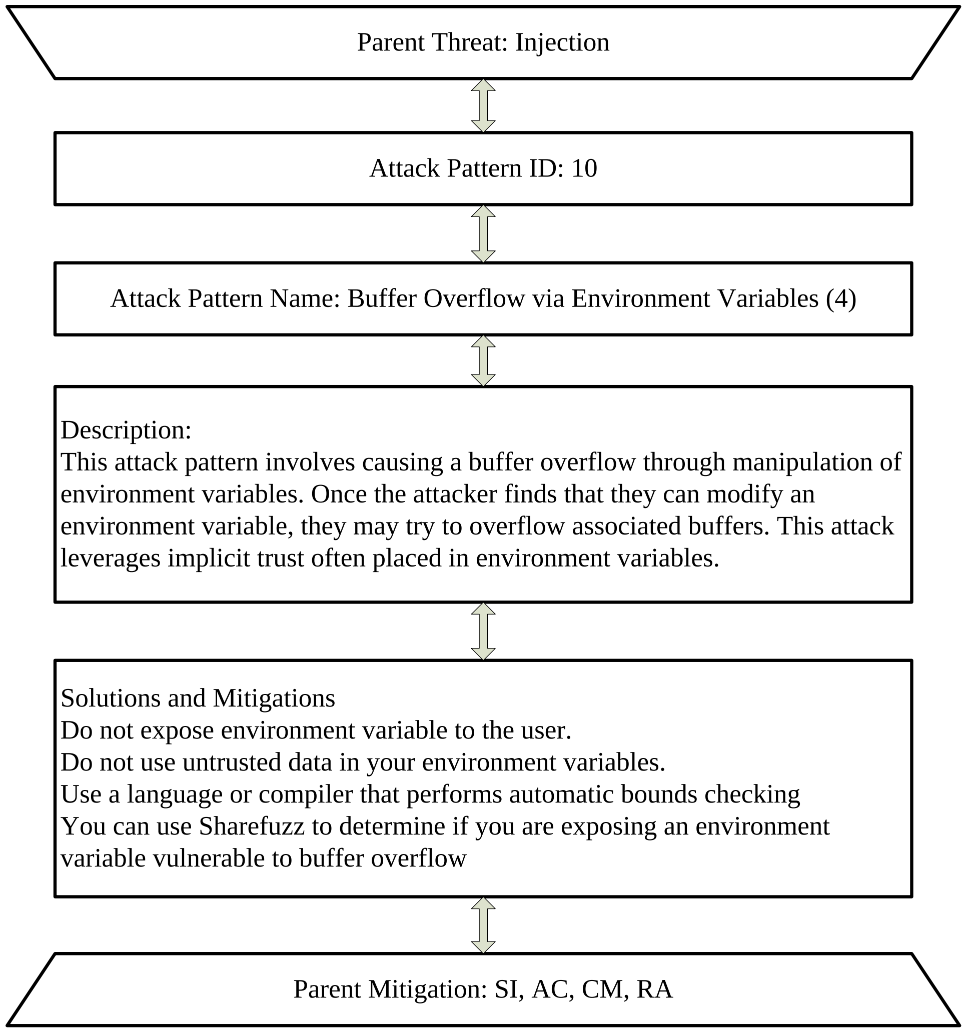
Attack Pattern ID: 9

Attack Pattern Name: Buffer Overflow in Local Command-Line Utilities (5)

Description:
This attack targets command-line utilities available in a number of shells. An attacker can leverage a vulnerability found in a command-line utility to escalate privilege to root.

Solutions and Mitigations:
Carefully review the service's implementation before making it available to user. For instance you can use manual or automated code review to uncover vulnerabilities such as buffer overflow.
Use a language or compiler that performs automatic bounds checking.
Use an abstraction library to abstract away risky APIs. Not a complete solution.
Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
Operational: Use OS-level preventative functionality. Not a complete solution.
Apply the latest patches to your user exposed services. This may not be a complete solution, specially against zero day attack.
Do not unnecessarily expose services.

Parent Mitigation: RA, SI, CM, SA, AC



Parent Threat: Injection

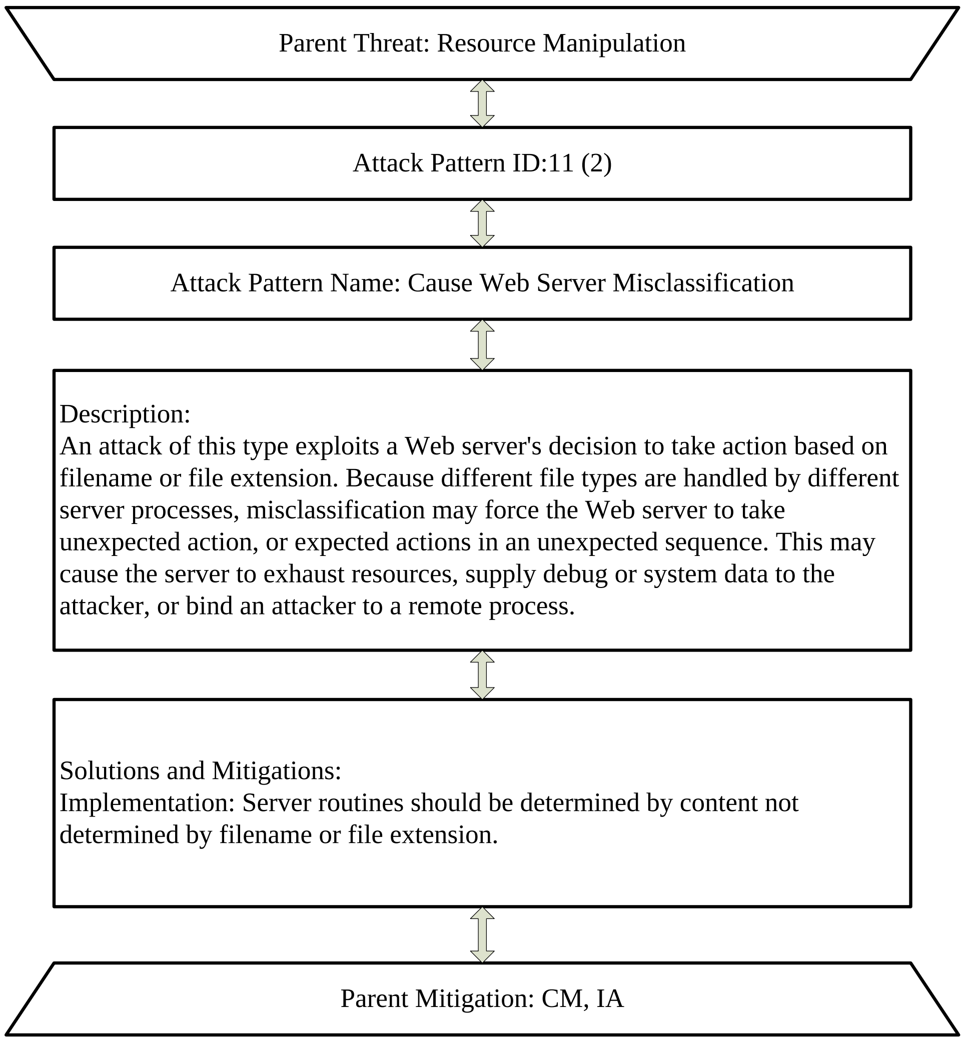
Attack Pattern ID: 10

Attack Pattern Name: Buffer Overflow via Environment Variables (4)

Description:
This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables.

Solutions and Mitigations
Do not expose environment variable to the user.
Do not use untrusted data in your environment variables.
Use a language or compiler that performs automatic bounds checking
You can use Sharefuzz to determine if you are exposing an environment variable vulnerable to buffer overflow

Parent Mitigation: SI, AC, CM, RA



Parent Threat: Resource Manipulation

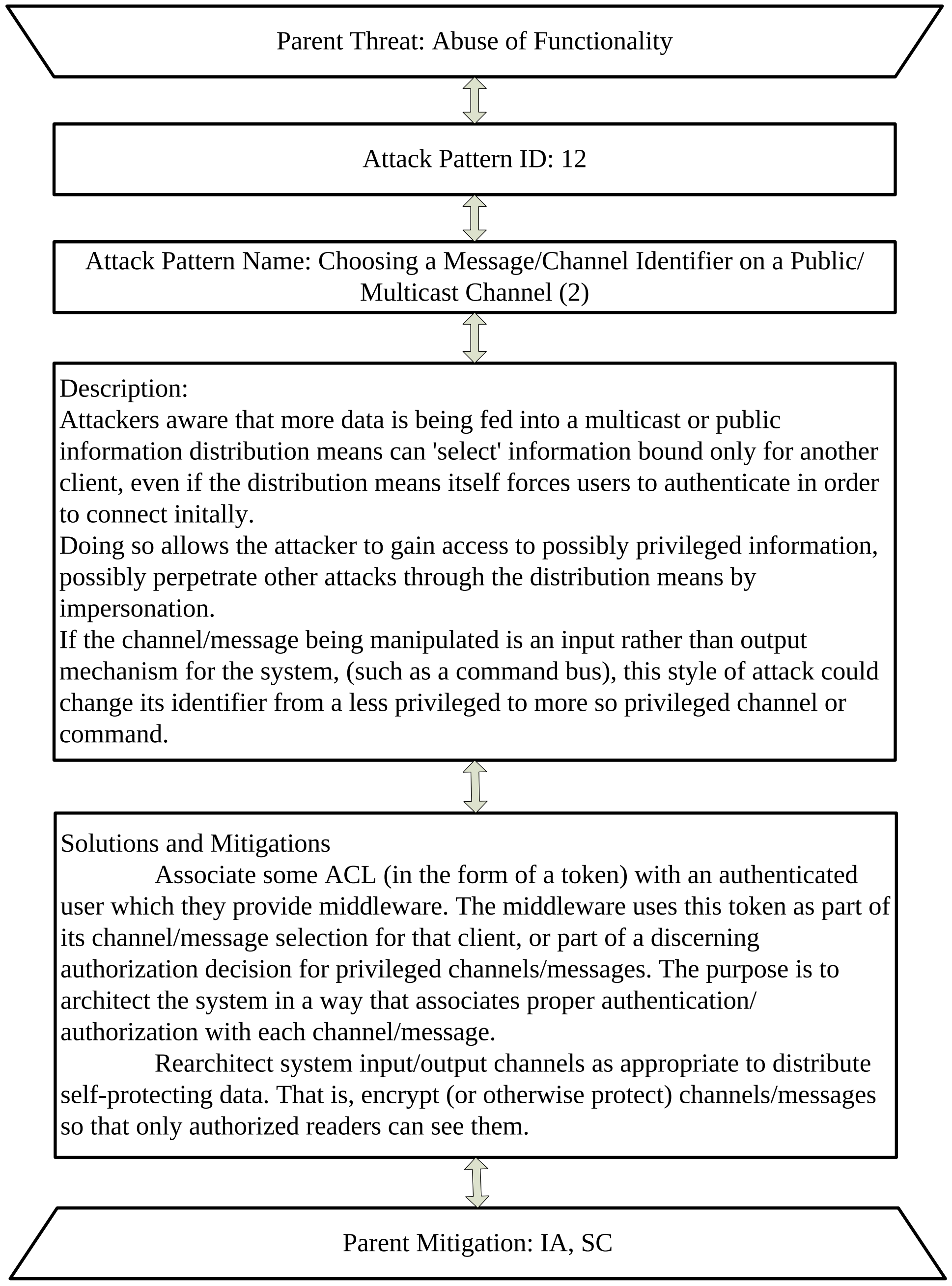
Attack Pattern ID:11 (2)

Attack Pattern Name: Cause Web Server Misclassification

Description:
An attack of this type exploits a Web server's decision to take action based on filename or file extension. Because different file types are handled by different server processes, misclassification may force the Web server to take unexpected action, or expected actions in an unexpected sequence. This may cause the server to exhaust resources, supply debug or system data to the attacker, or bind an attacker to a remote process.

Solutions and Mitigations:
Implementation: Server routines should be determined by content not determined by filename or file extension.

Parent Mitigation: CM, IA



Parent Threat: Abuse of Functionality

Attack Pattern ID: 12

Attack Pattern Name: Choosing a Message/Channel Identifier on a Public/ Multicast Channel (2)

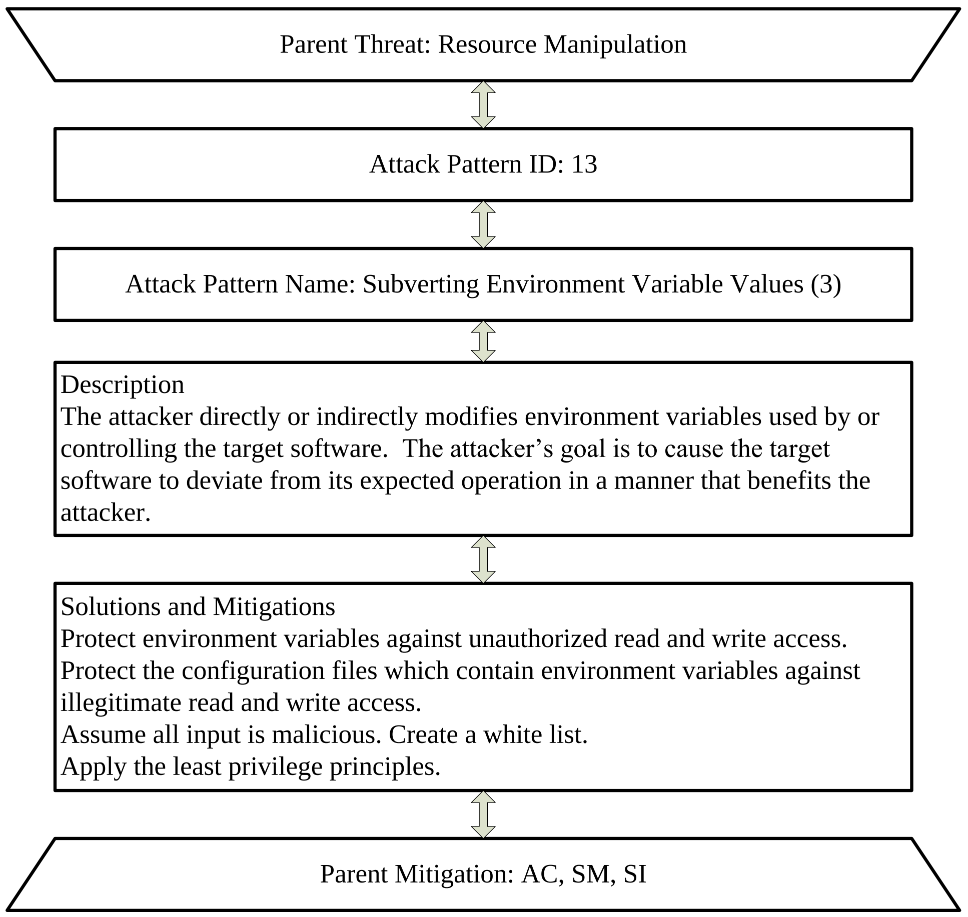
Description:
Attackers aware that more data is being fed into a multicast or public information distribution means can 'select' information bound only for another client, even if the distribution means itself forces users to authenticate in order to connect initally.
Doing so allows the attacker to gain access to possibly privileged information, possibly perpetrate other attacks through the distribution means by impersonation.
If the channel/message being manipulated is an input rather than output mechanism for the system, (such as a command bus), this style of attack could change its identifier from a less privileged to more so privileged channel or command.

Solutions and Mitigations

Associate some ACL (in the form of a token) with an authenticated user which they provide middleware. The middleware uses this token as part of its channel/message selection for that client, or part of a discerning authorization decision for privileged channels/messages. The purpose is to architect the system in a way that associates proper authentication/ authorization with each channel/message.

Rearchitect system input/output channels as appropriate to distribute self-protecting data. That is, encrypt (or otherwise protect) channels/messages so that only authorized readers can see them.

Parent Mitigation: IA, SC



Parent Threat: Resource Manipulation

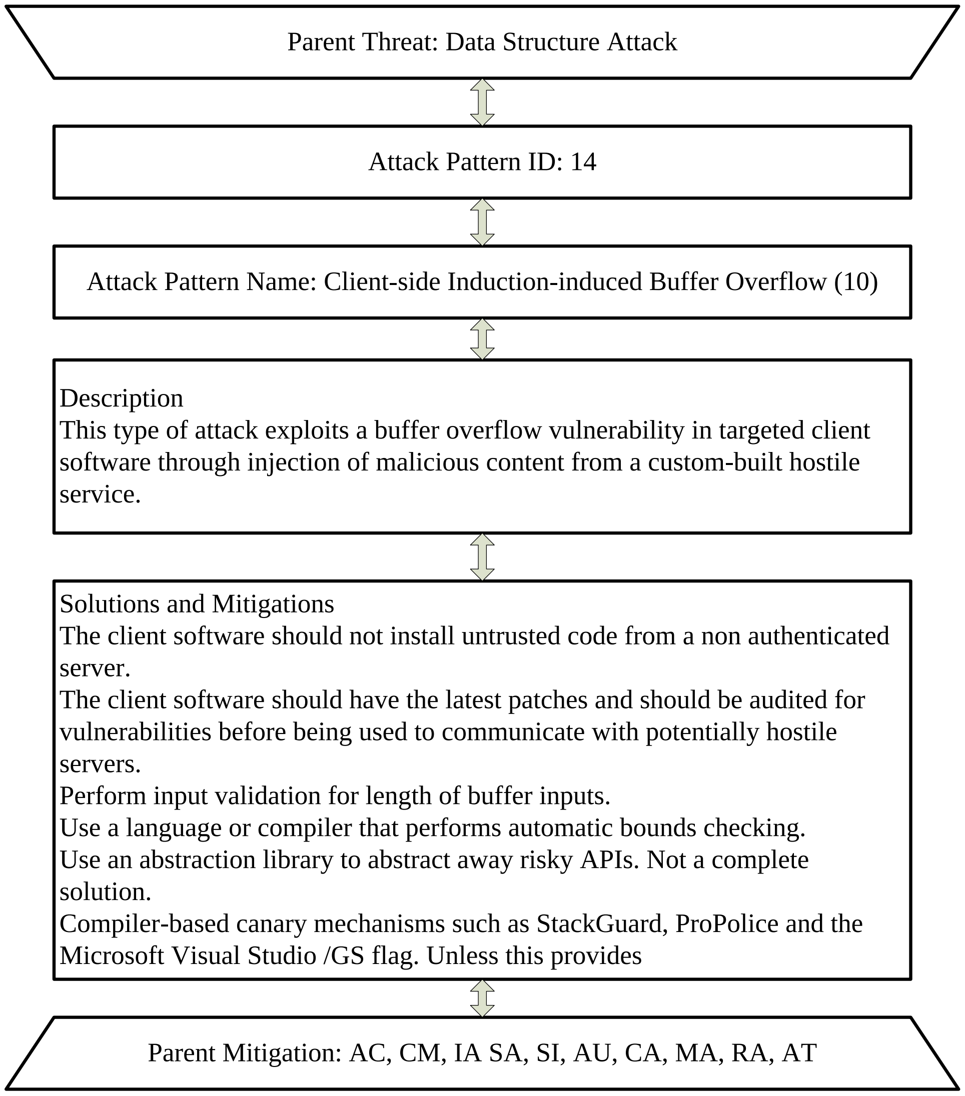
Attack Pattern ID: 13

Attack Pattern Name: Subverting Environment Variable Values (3)

Description
The attacker directly or indirectly modifies environment variables used by or controlling the target software. The attacker's goal is to cause the target software to deviate from its expected operation in a manner that benefits the attacker.

Solutions and Mitigations
Protect environment variables against unauthorized read and write access.
Protect the configuration files which contain environment variables against illegitimate read and write access.
Assume all input is malicious. Create a white list.
Apply the least privilege principles.

Parent Mitigation: AC, SM, SI



Parent Threat: Data Structure Attack

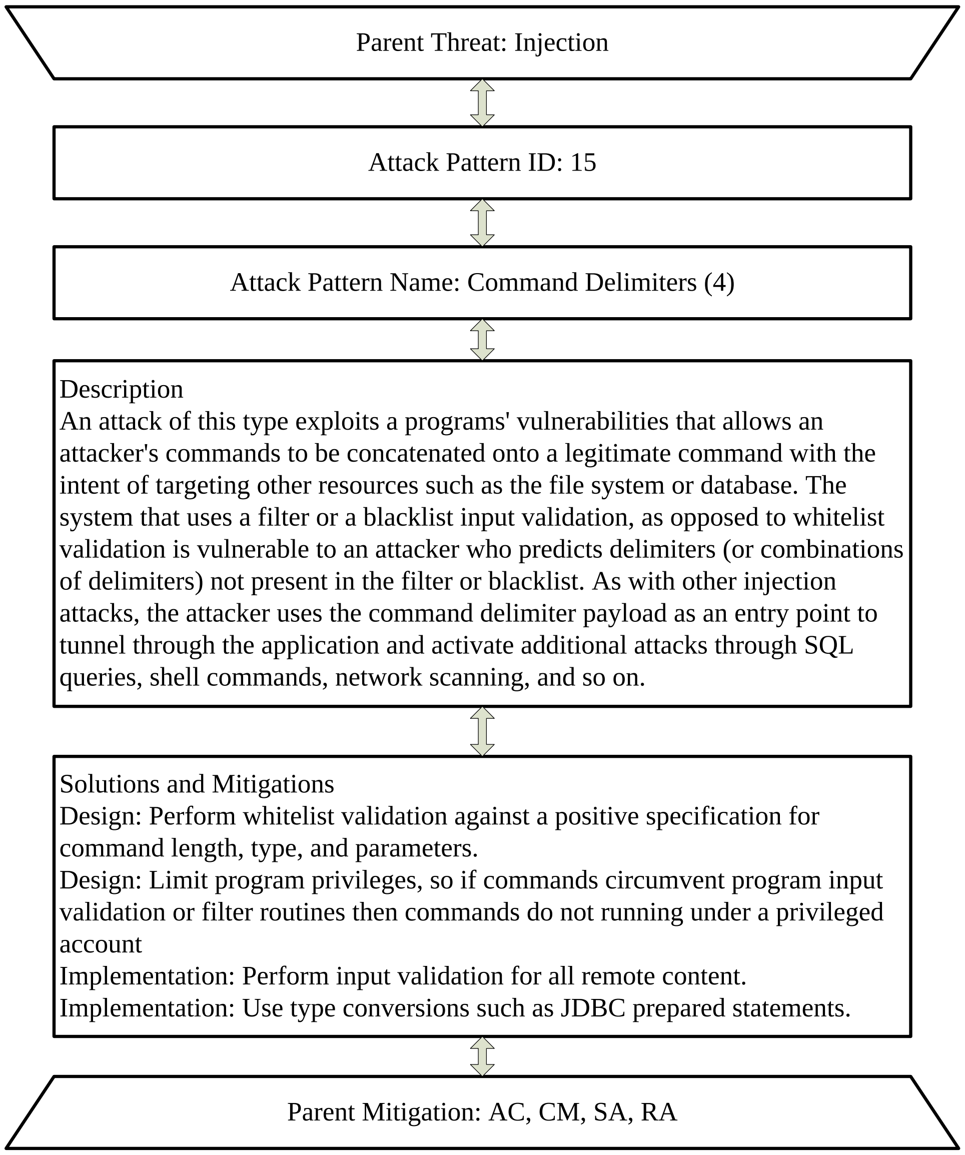
Attack Pattern ID: 14

Attack Pattern Name: Client-side Induction-induced Buffer Overflow (10)

Description
This type of attack exploits a buffer overflow vulnerability in targeted client software through injection of malicious content from a custom-built hostile service.

Solutions and Mitigations
The client software should not install untrusted code from a non authenticated server.
The client software should have the latest patches and should be audited for vulnerabilities before being used to communicate with potentially hostile servers.
Perform input validation for length of buffer inputs.
Use a language or compiler that performs automatic bounds checking.
Use an abstraction library to abstract away risky APIs. Not a complete solution.
Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides

Parent Mitigation: AC, CM, IA SA, SI, AU, CA, MA, RA, AT



Parent Threat: Injection

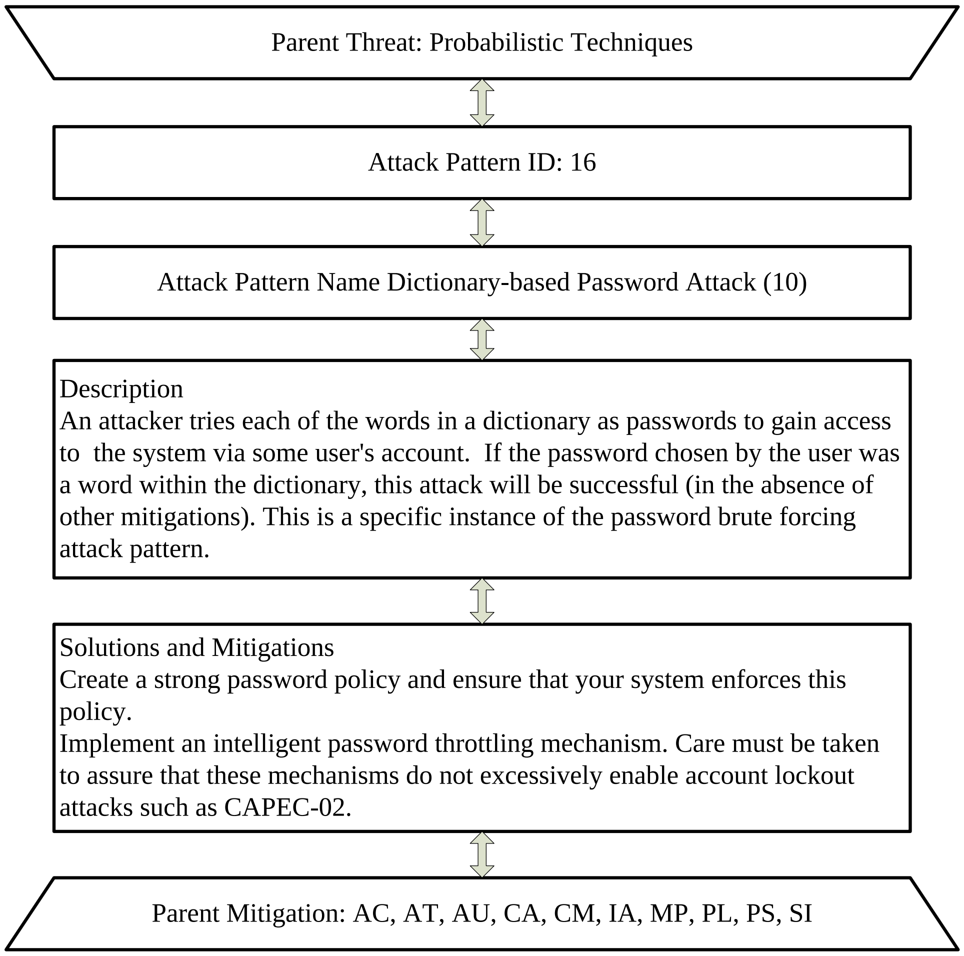
Attack Pattern ID: 15

Attack Pattern Name: Command Delimiters (4)

Description
An attack of this type exploits a programs' vulnerabilities that allows an attacker's commands to be concatenated onto a legitimate command with the intent of targeting other resources such as the file system or database. The system that uses a filter or a blacklist input validation, as opposed to whitelist validation is vulnerable to an attacker who predicts delimiters (or combinations of delimiters) not present in the filter or blacklist. As with other injection attacks, the attacker uses the command delimiter payload as an entry point to tunnel through the application and activate additional attacks through SQL queries, shell commands, network scanning, and so on.

Solutions and Mitigations
Design: Perform whitelist validation against a positive specification for command length, type, and parameters.
Design: Limit program privileges, so if commands circumvent program input validation or filter routines then commands do not running under a privileged account
Implementation: Perform input validation for all remote content.
Implementation: Use type conversions such as JDBC prepared statements.

Parent Mitigation: AC, CM, SA, RA



Parent Threat: Probabilistic Techniques

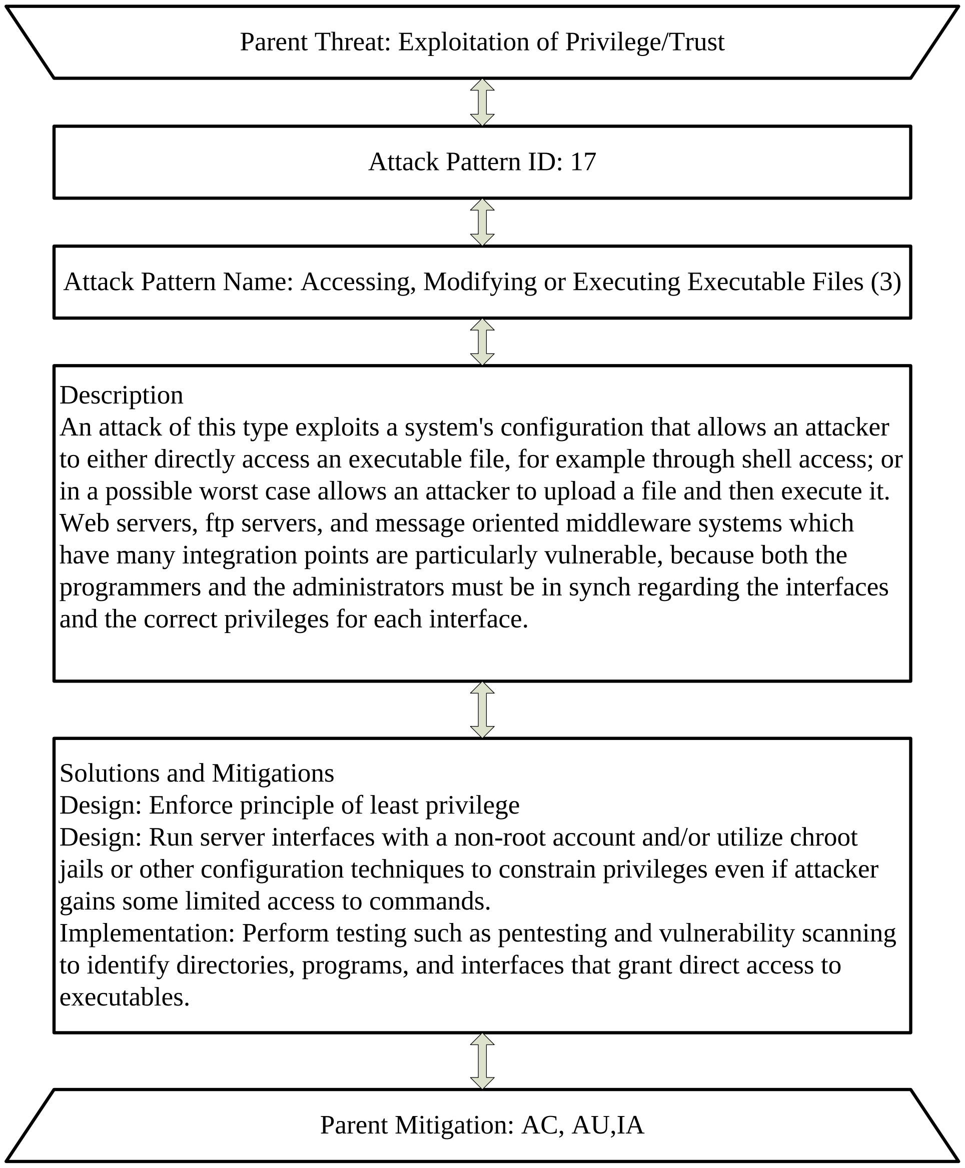
Attack Pattern ID: 16

Attack Pattern Name Dictionary-based Password Attack (10)

Description
An attacker tries each of the words in a dictionary as passwords to gain access to the system via some user's account. If the password chosen by the user was a word within the dictionary, this attack will be successful (in the absence of other mitigations). This is a specific instance of the password brute forcing attack pattern.

Solutions and Mitigations
Create a strong password policy and ensure that your system enforces this policy.
Implement an intelligent password throttling mechanism. Care must be taken to assure that these mechanisms do not excessively enable account lockout attacks such as CAPEC-02.

Parent Mitigation: AC, AT, AU, CA, CM, IA, MP, PL, PS, SI



Parent Threat: Exploitation of Privilege/Trust

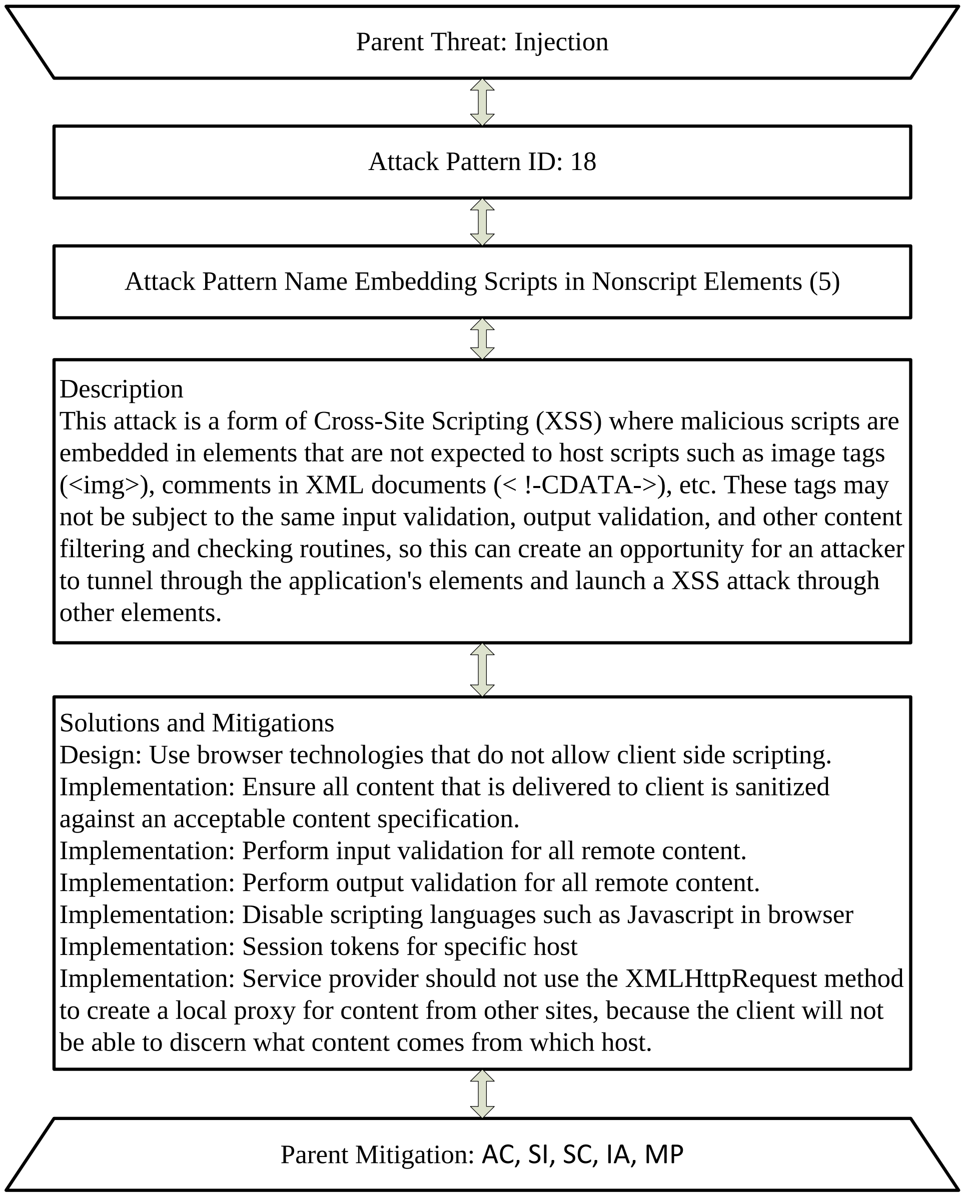
Attack Pattern ID: 17

Attack Pattern Name: Accessing, Modifying or Executing Executable Files (3)

Description
An attack of this type exploits a system's configuration that allows an attacker to either directly access an executable file, for example through shell access; or in a possible worst case allows an attacker to upload a file and then execute it. Web servers, ftp servers, and message oriented middleware systems which have many integration points are particularly vulnerable, because both the programmers and the administrators must be in synch regarding the interfaces and the correct privileges for each interface.

Solutions and Mitigations
Design: Enforce principle of least privilege
Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands.
Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.

Parent Mitigation: AC, AU,IA



Parent Threat: Injection

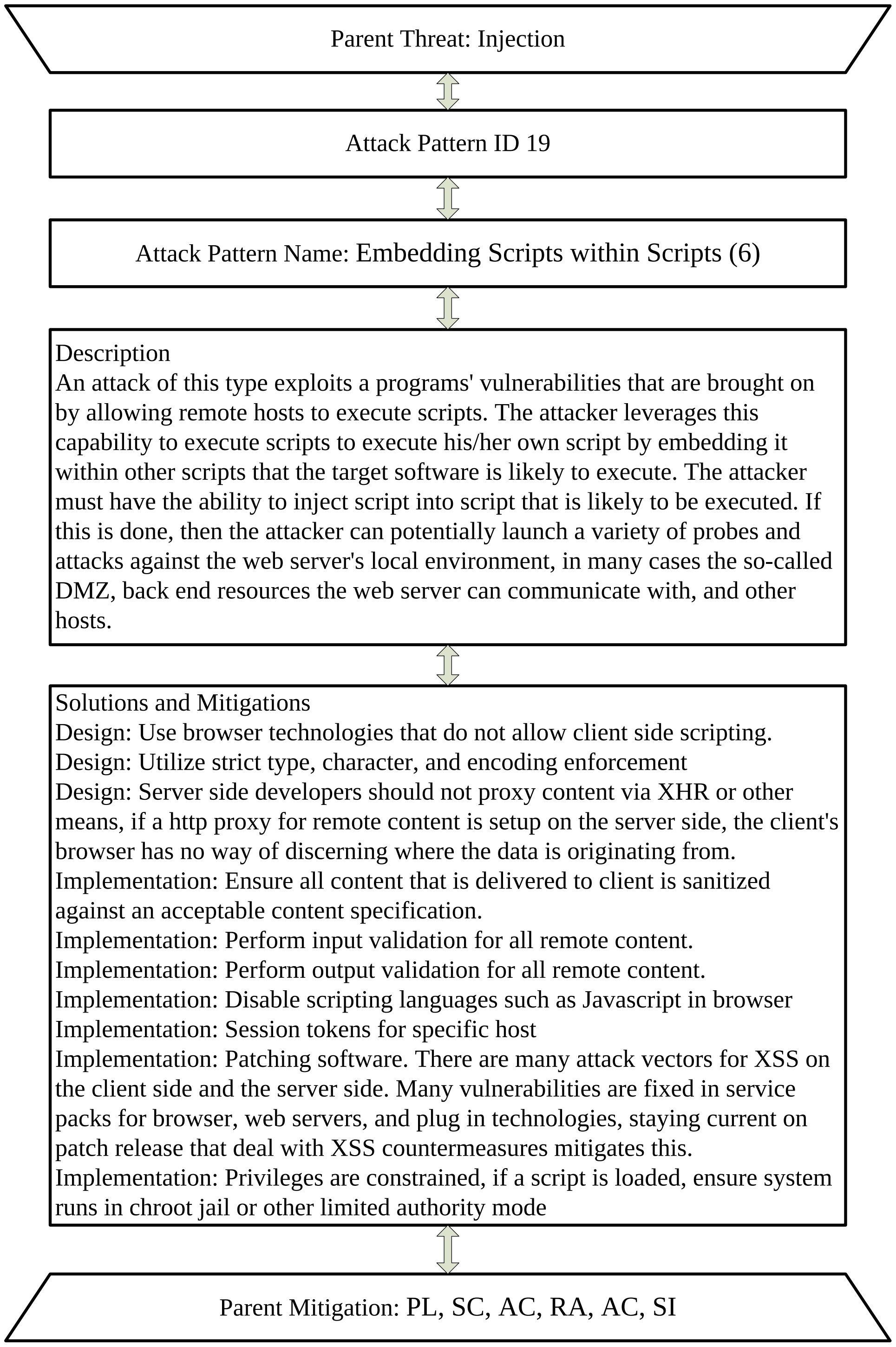
Attack Pattern ID: 18

Attack Pattern Name Embedding Scripts in Nonscript Elements (5)

Description
This attack is a form of Cross-Site Scripting (XSS) where malicious scripts are embedded in elements that are not expected to host scripts such as image tags (<img>), comments in XML documents (< !-CDATA->), etc. These tags may not be subject to the same input validation, output validation, and other content filtering and checking routines, so this can create an opportunity for an attacker to tunnel through the application's elements and launch a XSS attack through other elements.

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting.
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser
Implementation: Session tokens for specific host
Implementation: Service provider should not use the XMLHttpRequest method to create a local proxy for content from other sites, because the client will not be able to discern what content comes from which host.

Parent Mitigation: AC, SI, SC, IA, MP



Parent Threat: Injection

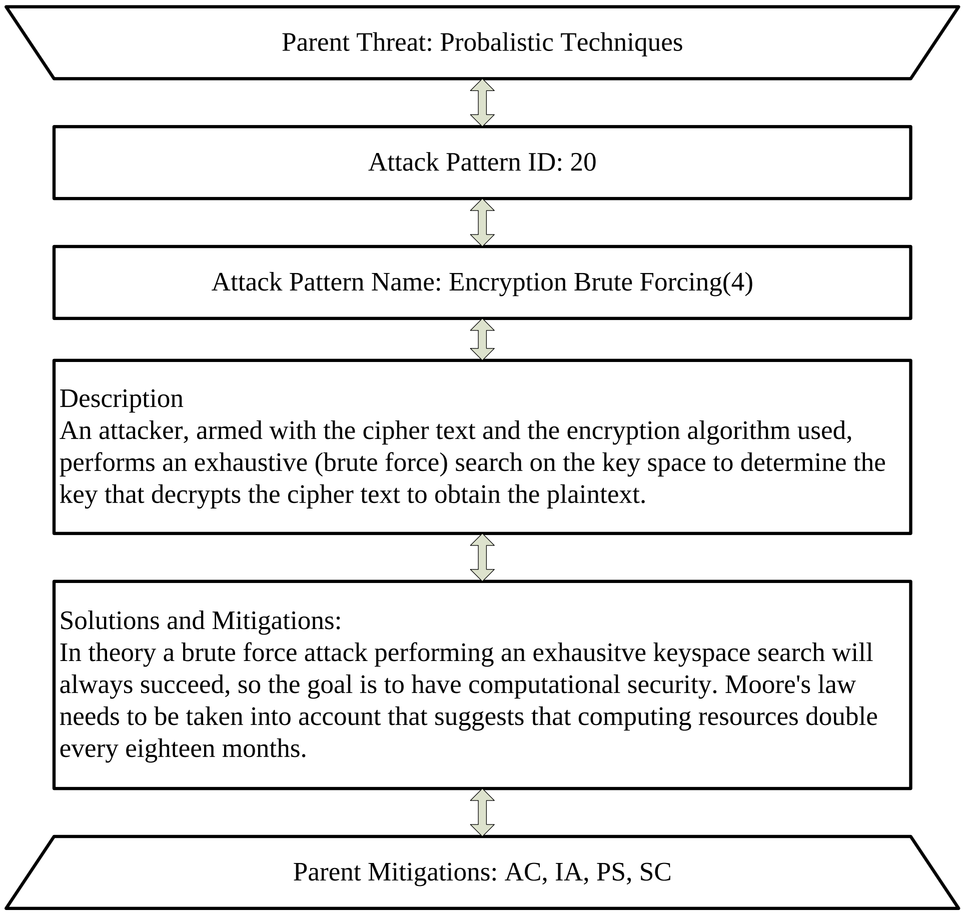
Attack Pattern ID 19

Attack Pattern Name: Embedding Scripts within Scripts (6)

Description
An attack of this type exploits a programs' vulnerabilities that are brought on by allowing remote hosts to execute scripts. The attacker leverages this capability to execute scripts to execute his/her own script by embedding it within other scripts that the target software is likely to execute. The attacker must have the ability to inject script into script that is likely to be executed. If this is done, then the attacker can potentially launch a variety of probes and attacks against the web server's local environment, in many cases the so-called DMZ, back end resources the web server can communicate with, and other hosts.

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting.
Design: Utilize strict type, character, and encoding enforcement
Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from.
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser
Implementation: Session tokens for specific host
Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.
Implementation: Privileges are constrained, if a script is loaded, ensure system runs in chroot jail or other limited authority mode

Parent Mitigation: PL, SC, AC, RA, AC, SI



Parent Threat: Probalistic Techniques

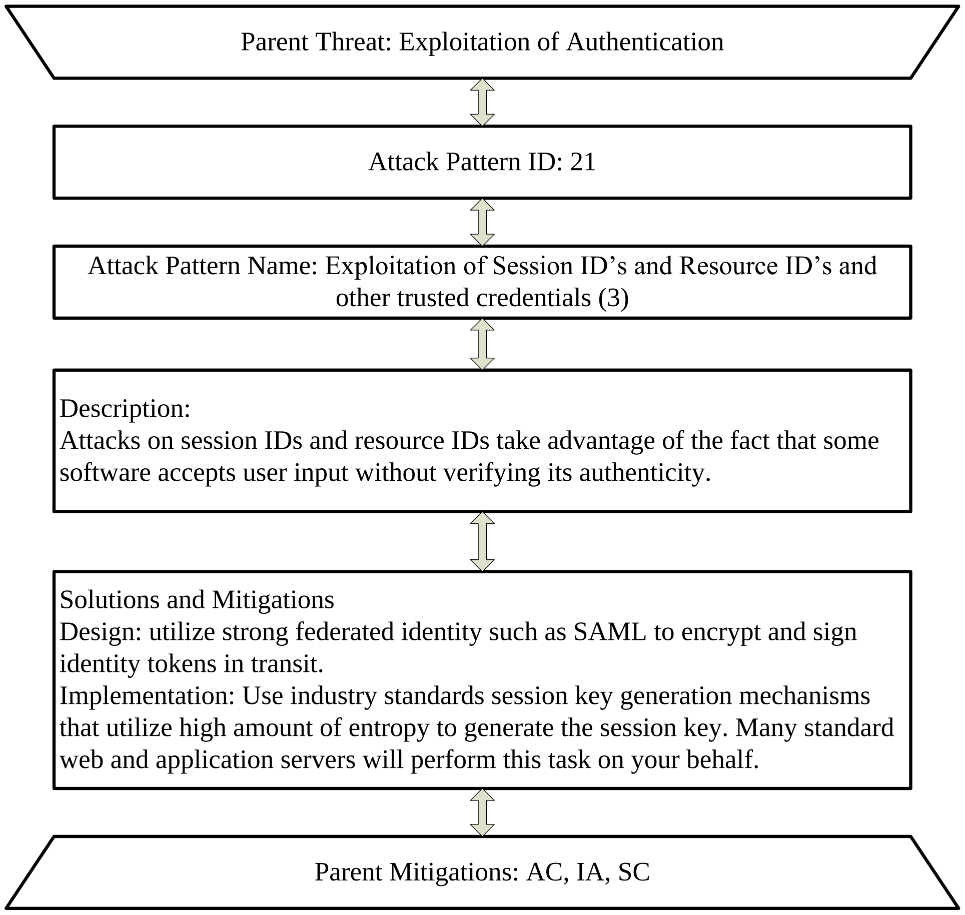
Attack Pattern ID: 20

Attack Pattern Name: Encryption Brute Forcing(4)

Description
An attacker, armed with the cipher text and the encryption algorithm used, performs an exhaustive (brute force) search on the key space to determine the key that decrypts the cipher text to obtain the plaintext.

Solutions and Mitigations:
In theory a brute force attack performing an exhausitve keyspace search will always succeed, so the goal is to have computational security. Moore's law needs to be taken into account that suggests that computing resources double every eighteen months.

Parent Mitigations: AC, IA, PS, SC



Parent Threat: Exploitation of Authentication

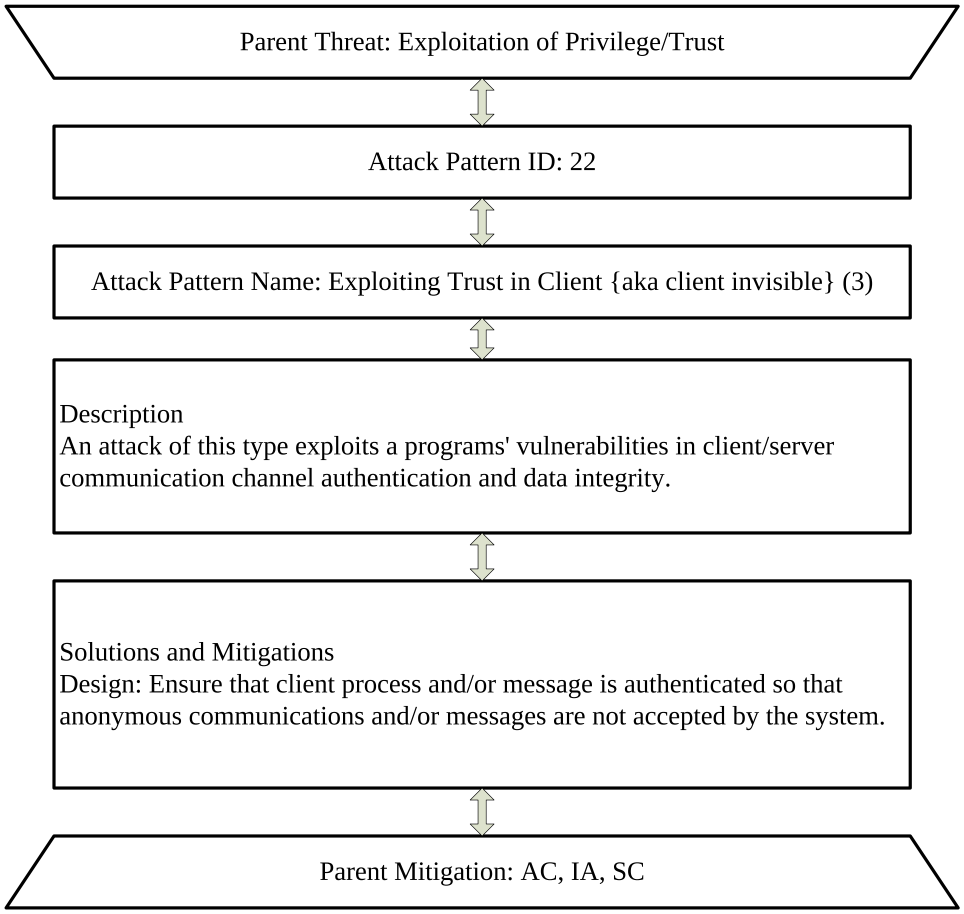
Attack Pattern ID: 21

Attack Pattern Name: Exploitation of Session ID’s and Resource ID’s and other trusted credentials (3)

Description:
Attacks on session IDs and resource IDs take advantage of the fact that some software accepts user input without verifying its authenticity.

Solutions and Mitigations
Design: utilize strong federated identity such as SAML to encrypt and sign identity tokens in transit.
Implementation: Use industry standards session key generation mechanisms that utilize high amount of entropy to generate the session key. Many standard web and application servers will perform this task on your behalf.

Parent Mitigations: AC, IA, SC



Parent Threat: Exploitation of Privilege/Trust

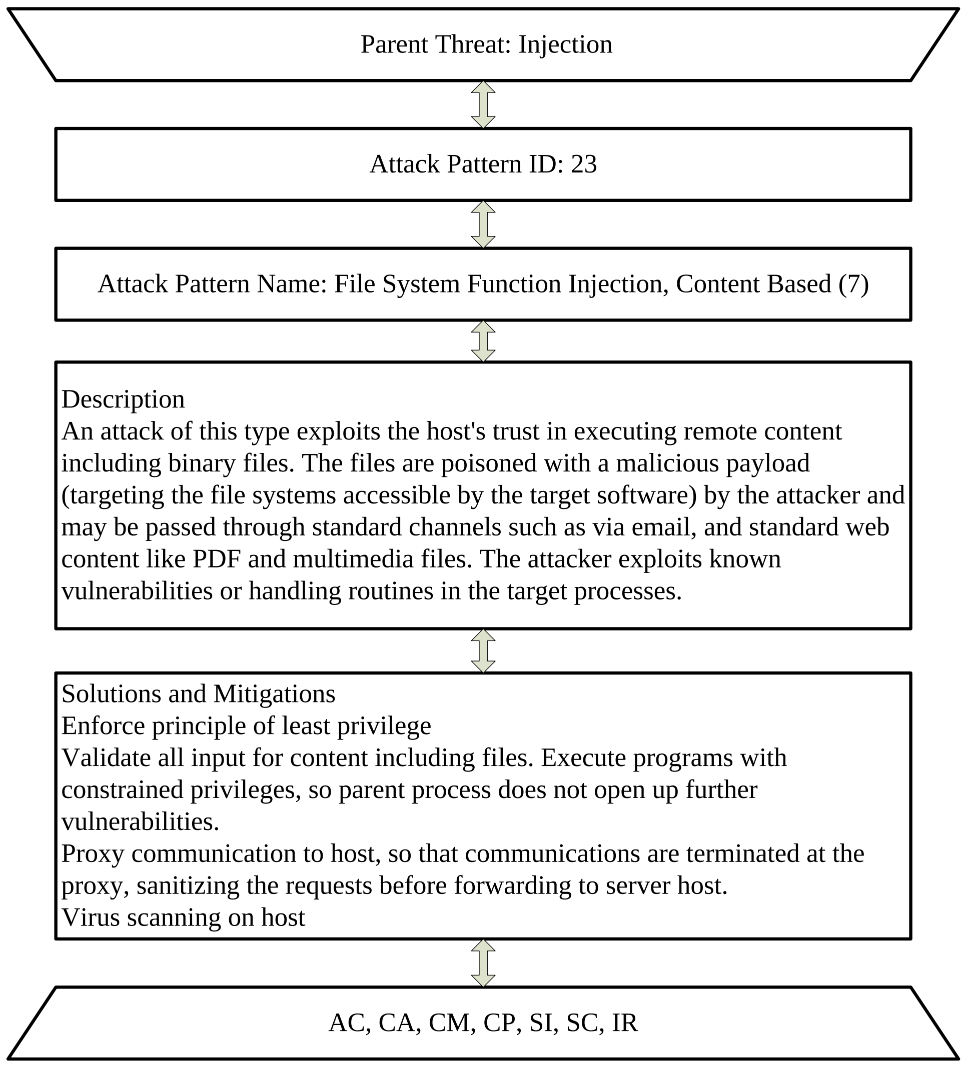
Attack Pattern ID: 22

Attack Pattern Name: Exploiting Trust in Client {aka client invisible} (3)

Description
An attack of this type exploits a programs' vulnerabilities in client/server communication channel authentication and data integrity.

Solutions and Mitigations
Design: Ensure that client process and/or message is authenticated so that anonymous communications and/or messages are not accepted by the system.

Parent Mitigation: AC, IA, SC



Parent Threat: Injection

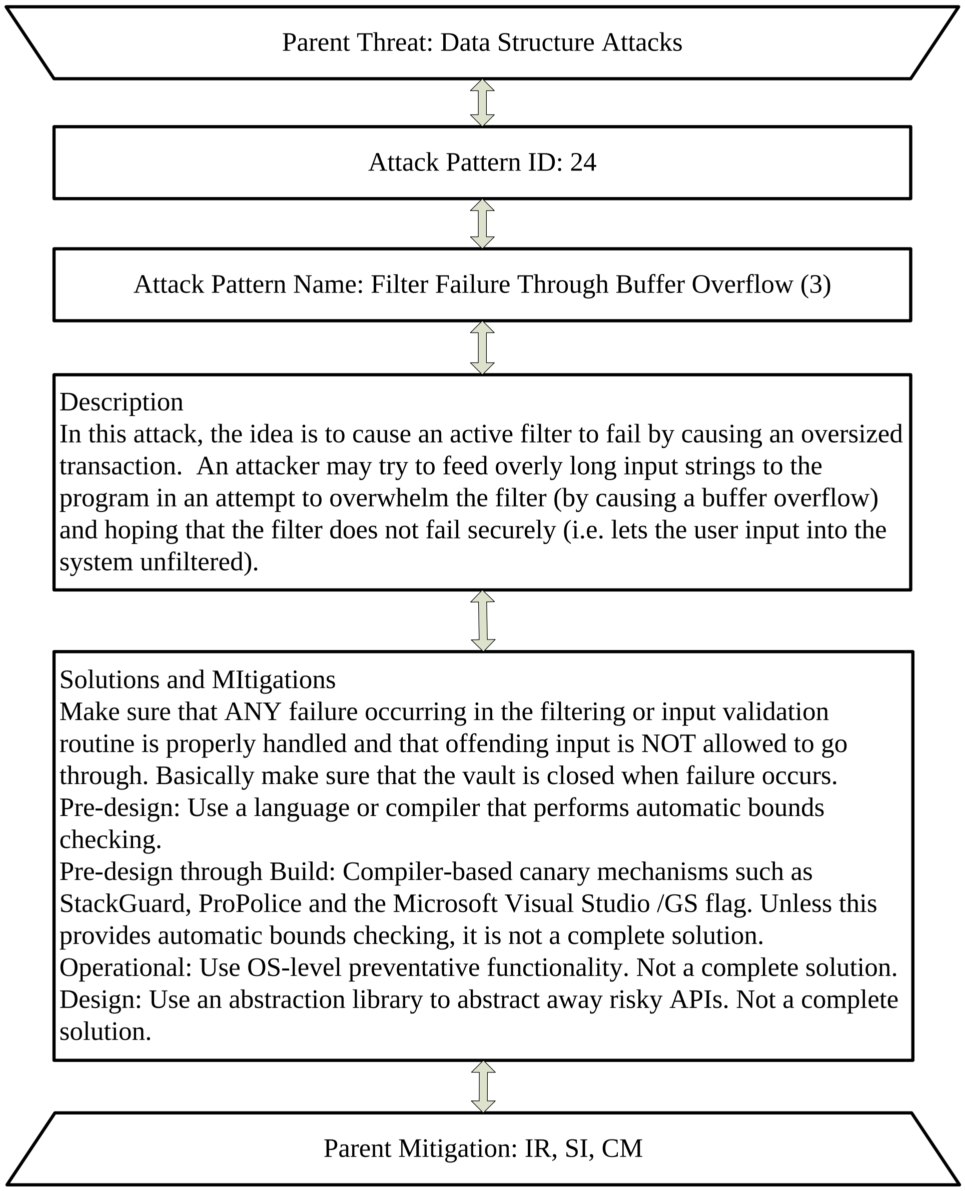
Attack Pattern ID: 23

Attack Pattern Name: File System Function Injection, Content Based (7)

Description
An attack of this type exploits the host's trust in executing remote content including binary files. The files are poisoned with a malicious payload (targeting the file systems accessible by the target software) by the attacker and may be passed through standard channels such as via email, and standard web content like PDF and multimedia files. The attacker exploits known vulnerabilities or handling routines in the target processes.

Solutions and Mitigations
Enforce principle of least privilege
Validate all input for content including files. Execute programs with constrained privileges, so parent process does not open up further vulnerabilities.
Proxy communication to host, so that communications are terminated at the proxy, sanitizing the requests before forwarding to server host.
Virus scanning on host

AC, CA, CM, CP, SI, SC, IR



Parent Threat: Data Structure Attacks

Attack Pattern ID: 24

Attack Pattern Name: Filter Failure Through Buffer Overflow (3)

Description
In this attack, the idea is to cause an active filter to fail by causing an oversized transaction. An attacker may try to feed overly long input strings to the program in an attempt to overwhelm the filter (by causing a buffer overflow) and hoping that the filter does not fail securely (i.e. lets the user input into the system unfiltered).

Solutions and MItigations
Make sure that ANY failure occurring in the filtering or input validation routine is properly handled and that offending input is NOT allowed to go through. Basically make sure that the vault is closed when failure occurs.
Pre-design: Use a language or compiler that performs automatic bounds checking.
Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
Operational: Use OS-level preventative functionality. Not a complete solution.
Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.

Parent Mitigation: IR, SI, CM

Parent Threat: Time and State Attacks

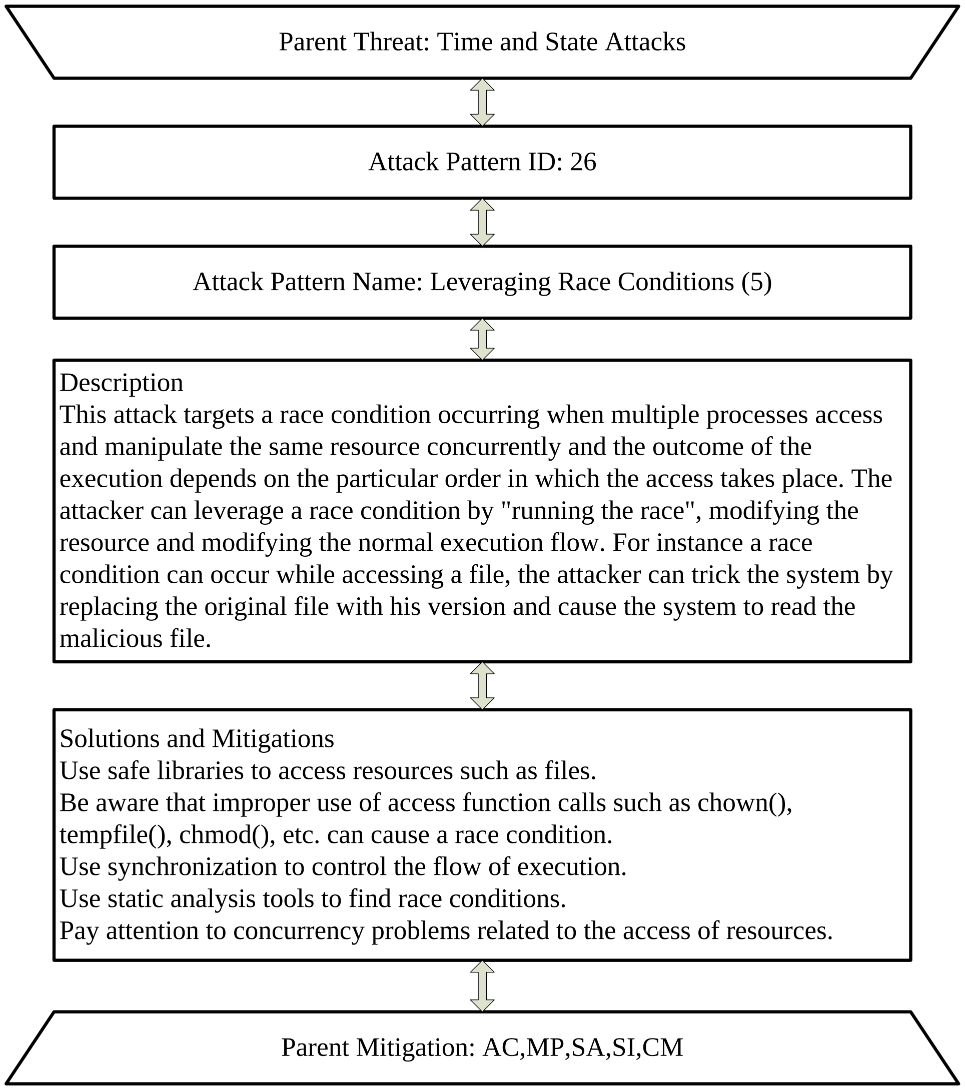
Attack Pattern ID: 25

Attack Pattern Name: Forced Deadlock (2)

Description
Attackers aware that more data is being fed into a multicast or public information distribution means can 'select' information bound only for another client, even if the distribution means itself forces users to authenticate in order to connect initally.
Doing so allows the attacker to gain access to possibly privileged information, possibly perpetrate other attacks through the distribution means by impersonation.
If the channel/message being manipulated is an input rather than output mechanism for the system, (such as a command bus), this style of attack could change its identifier from a less privileged to more so privileged channel or command.

Solutions and Mitigations
Use known algorithm to avoid deadlock condition (for instance non-blocking synchronization algorithms).
For competing actions use well known libraries which implement synchronization

Parent Mitigation: SC, SI



Parent Threat: Time and State Attacks

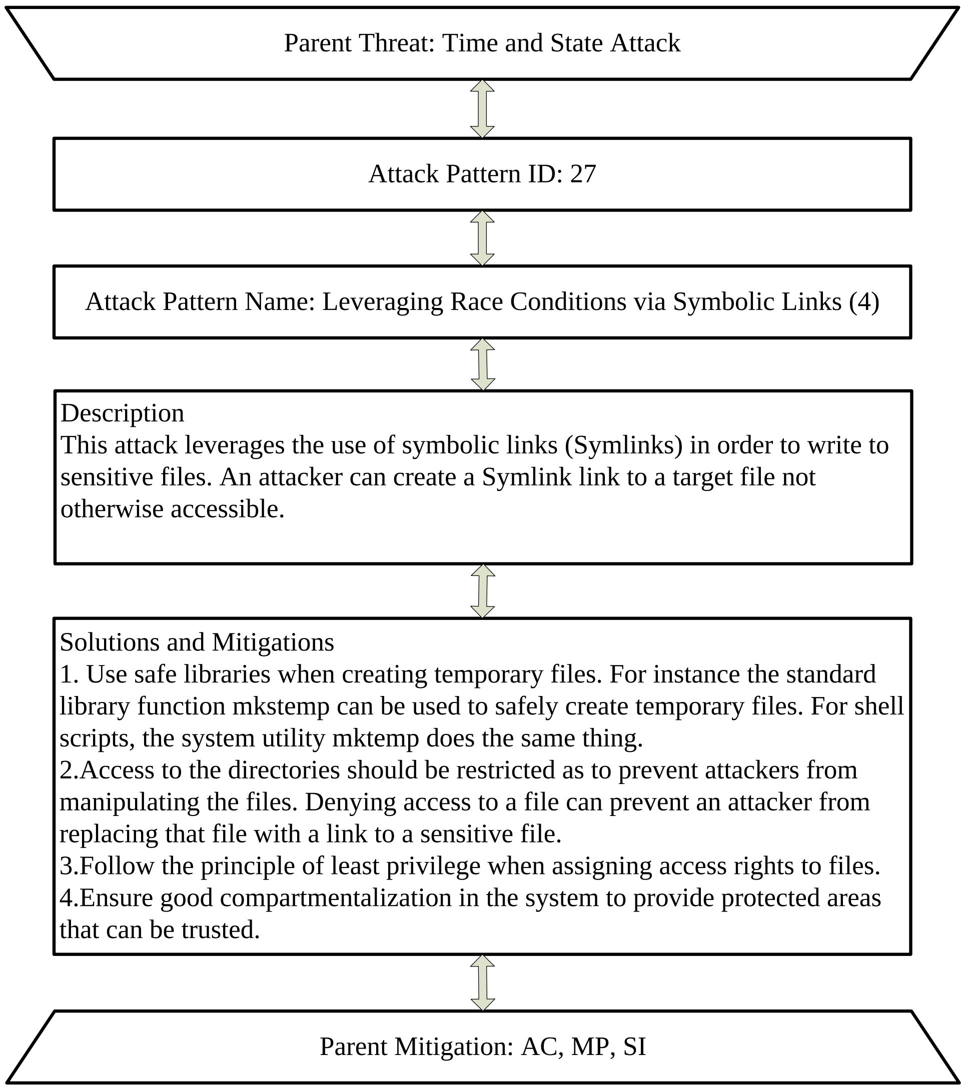
Attack Pattern ID: 26

Attack Pattern Name: Leveraging Race Conditions (5)

Description
This attack targets a race condition occurring when multiple processes access and manipulate the same resource concurrently and the outcome of the execution depends on the particular order in which the access takes place. The attacker can leverage a race condition by "running the race", modifying the resource and modifying the normal execution flow. For instance a race condition can occur while accessing a file, the attacker can trick the system by replacing the original file with his version and cause the system to read the malicious file.

Solutions and Mitigations
Use safe libraries to access resources such as files.
Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.
Use synchronization to control the flow of execution.
Use static analysis tools to find race conditions.
Pay attention to concurrency problems related to the access of resources.

Parent Mitigation: AC,MP,SA,SI,CM



Parent Threat: Time and State Attack

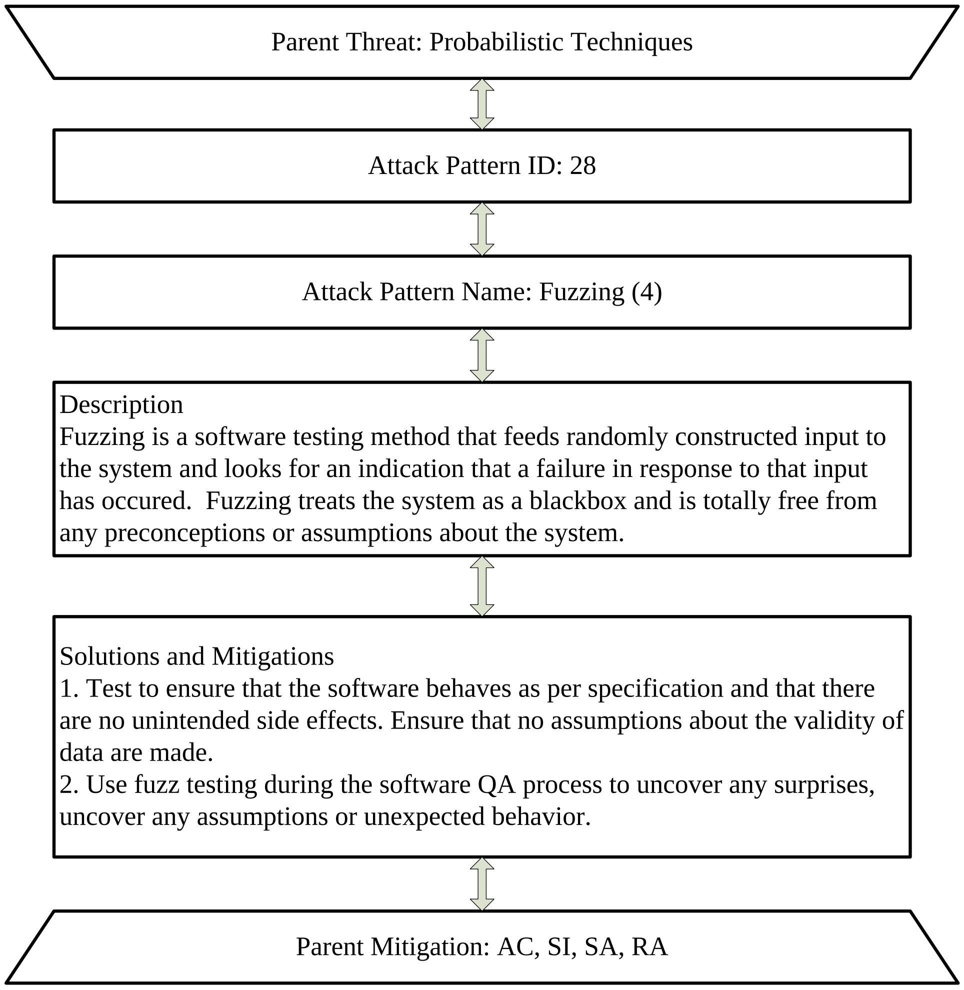
Attack Pattern ID: 27

Attack Pattern Name: Leveraging Race Conditions via Symbolic Links (4)

Description
This attack leverages the use of symbolic links (Symlinks) in order to write to sensitive files. An attacker can create a Symlink link to a target file not otherwise accessible.

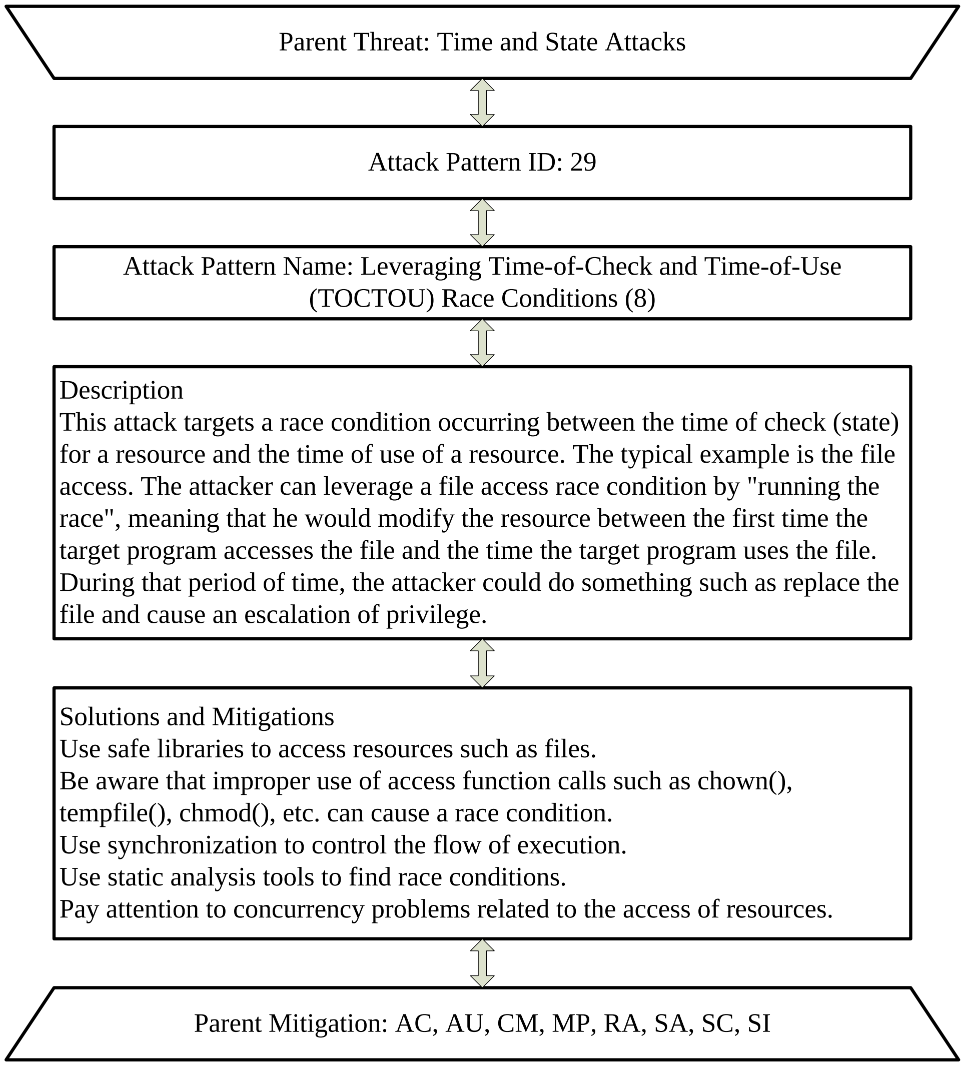
Solutions and Mitigations
1. Use safe libraries when creating temporary files. For instance the standard library function mkstemp can be used to safely create temporary files. For shell scripts, the system utility mktemp does the same thing.
2.Access to the directories should be restricted as to prevent attackers from manipulating the files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file.
3.Follow the principle of least privilege when assigning access rights to files.
4.Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Parent Mitigation: AC, MP, SI


Parent Threat: Probabilistic Techniques
Attack Pattern ID: 28
Attack Pattern Name: Fuzzing (4)

Description
Fuzzing is a software testing method that feeds randomly constructed input to the system and looks for an indication that a failure in response to that input has occured. Fuzzing treats the system as a blackbox and is totally free from any preconceptions or assumptions about the system.

Solutions and Mitigations
1. Test to ensure that the software behaves as per specification and that there are no unintended side effects. Ensure that no assumptions about the validity of data are made.
2. Use fuzz testing during the software QA process to uncover any surprises, uncover any assumptions or unexpected behavior.



Parent Threat: Time and State Attacks

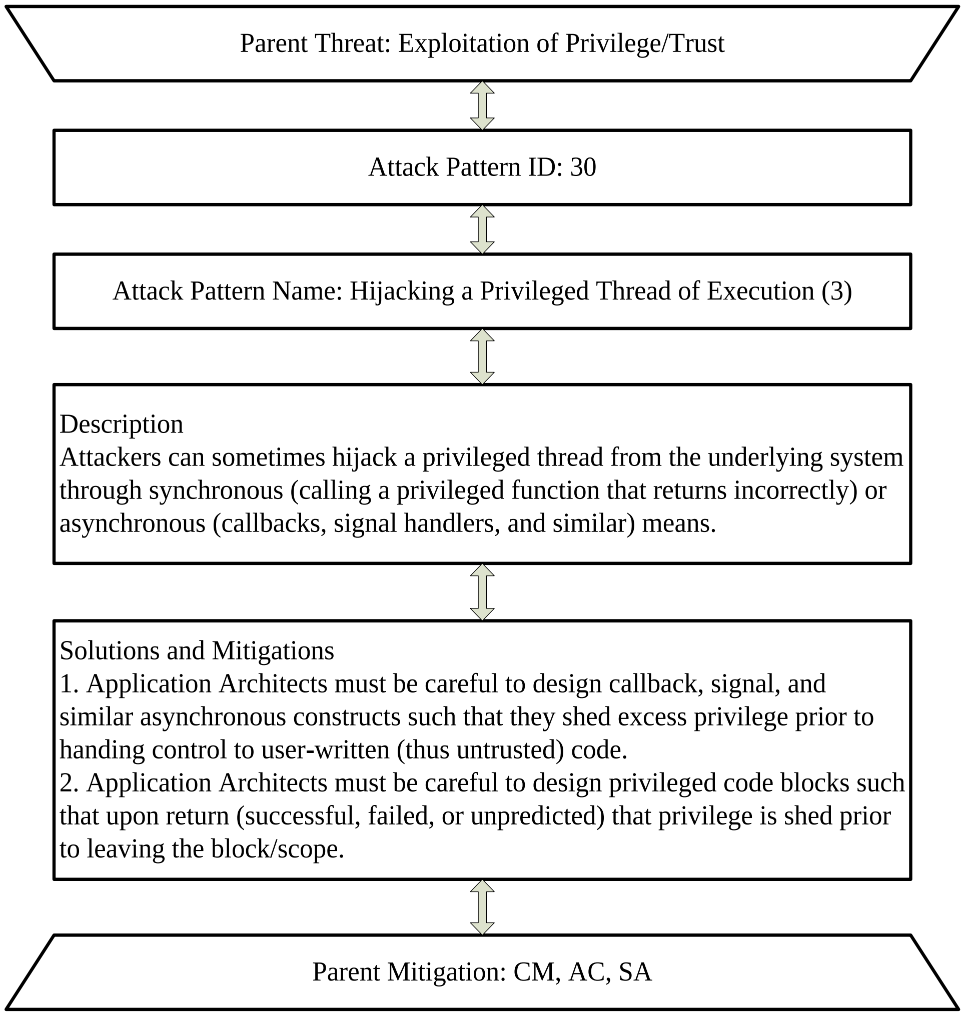
Attack Pattern ID: 29

Attack Pattern Name: Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions (8)

Description
This attack targets a race condition occurring between the time of check (state) for a resource and the time of use of a resource. The typical example is the file access. The attacker can leverage a file access race condition by "running the race", meaning that he would modify the resource between the first time the target program accesses the file and the time the target program uses the file. During that period of time, the attacker could do something such as replace the file and cause an escalation of privilege.

Solutions and Mitigations
Use safe libraries to access resources such as files.
Be aware that improper use of access function calls such as chown(), tempfile(), chmod(), etc. can cause a race condition.
Use synchronization to control the flow of execution.
Use static analysis tools to find race conditions.
Pay attention to concurrency problems related to the access of resources.

Parent Mitigation: AC, AU, CM, MP, RA, SA, SC, SI



Parent Threat: Exploitation of Privilege/Trust

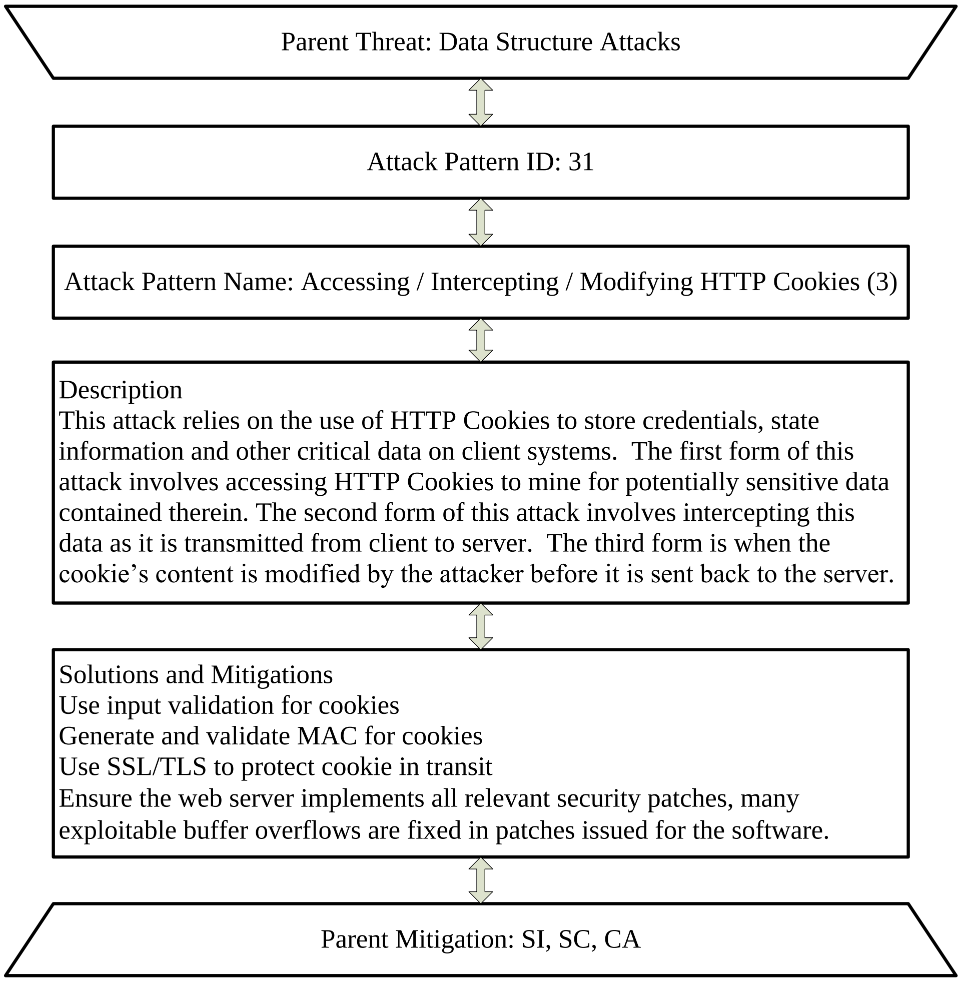
Attack Pattern ID: 30

Attack Pattern Name: Hijacking a Privileged Thread of Execution (3)

Description
Attackers can sometimes hijack a privileged thread from the underlying system through synchronous (calling a privileged function that returns incorrectly) or asynchronous (callbacks, signal handlers, and similar) means.

Solutions and Mitigations
1. Application Architects must be careful to design callback, signal, and similar asynchronous constructs such that they shed excess privilege prior to handing control to user-written (thus untrusted) code.
2. Application Architects must be careful to design privileged code blocks such that upon return (successful, failed, or unpredicted) that privilege is shed prior to leaving the block/scope.

Parent Mitigation: CM, AC, SA



Parent Threat: Data Structure Attacks

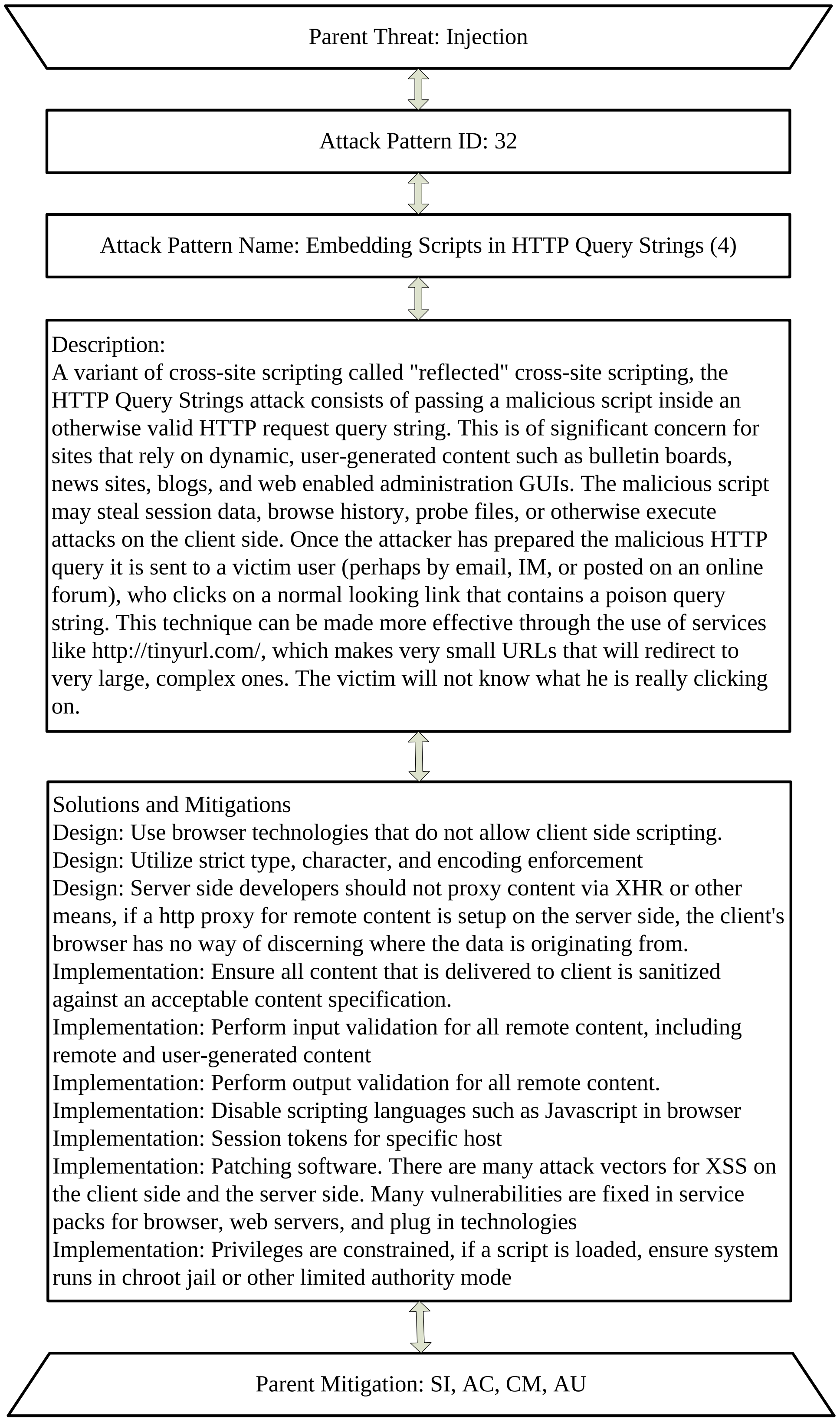
Attack Pattern ID: 31

Attack Pattern Name: Accessing / Intercepting / Modifying HTTP Cookies (3)

Description
This attack relies on the use of HTTP Cookies to store credentials, state information and other critical data on client systems. The first form of this attack involves accessing HTTP Cookies to mine for potentially sensitive data contained therein. The second form of this attack involves intercepting this data as it is transmitted from client to server. The third form is when the cookie's content is modified by the attacker before it is sent back to the server.

Solutions and Mitigations
Use input validation for cookies
Generate and validate MAC for cookies
Use SSL/TLS to protect cookie in transit
Ensure the web server implements all relevant security patches, many exploitable buffer overflows are fixed in patches issued for the software.

Parent Mitigation: SI, SC, CA



Parent Threat: Injection

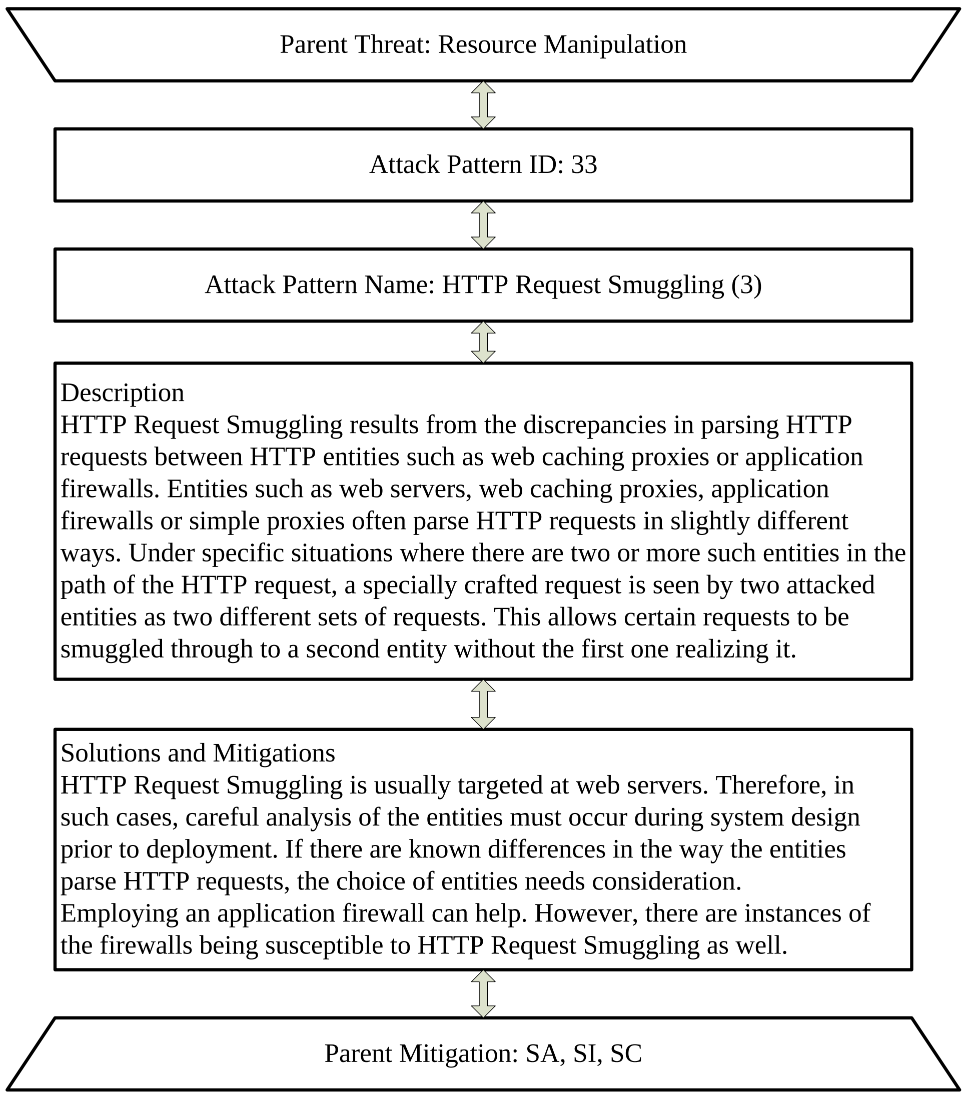
Attack Pattern ID: 32

Attack Pattern Name: Embedding Scripts in HTTP Query Strings (4)

Description:
A variant of cross-site scripting called "reflected" cross-site scripting, the HTTP Query Strings attack consists of passing a malicious script inside an otherwise valid HTTP request query string. This is of significant concern for sites that rely on dynamic, user-generated content such as bulletin boards, news sites, blogs, and web enabled administration GUIs. The malicious script may steal session data, browse history, probe files, or otherwise execute attacks on the client side. Once the attacker has prepared the malicious HTTP query it is sent to a victim user (perhaps by email, IM, or posted on an online forum), who clicks on a normal looking link that contains a poison query string. This technique can be made more effective through the use of services like http://tinyurl.com/, which makes very small URLs that will redirect to very large, complex ones. The victim will not know what he is really clicking on.

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting.
Design: Utilize strict type, character, and encoding enforcement
Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from.
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Perform input validation for all remote content, including remote and user-generated content
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser
Implementation: Session tokens for specific host
Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies
Implementation: Privileges are constrained, if a script is loaded, ensure system runs in chroot jail or other limited authority mode

Parent Mitigation: SI, AC, CM, AU



Parent Threat: Resource Manipulation

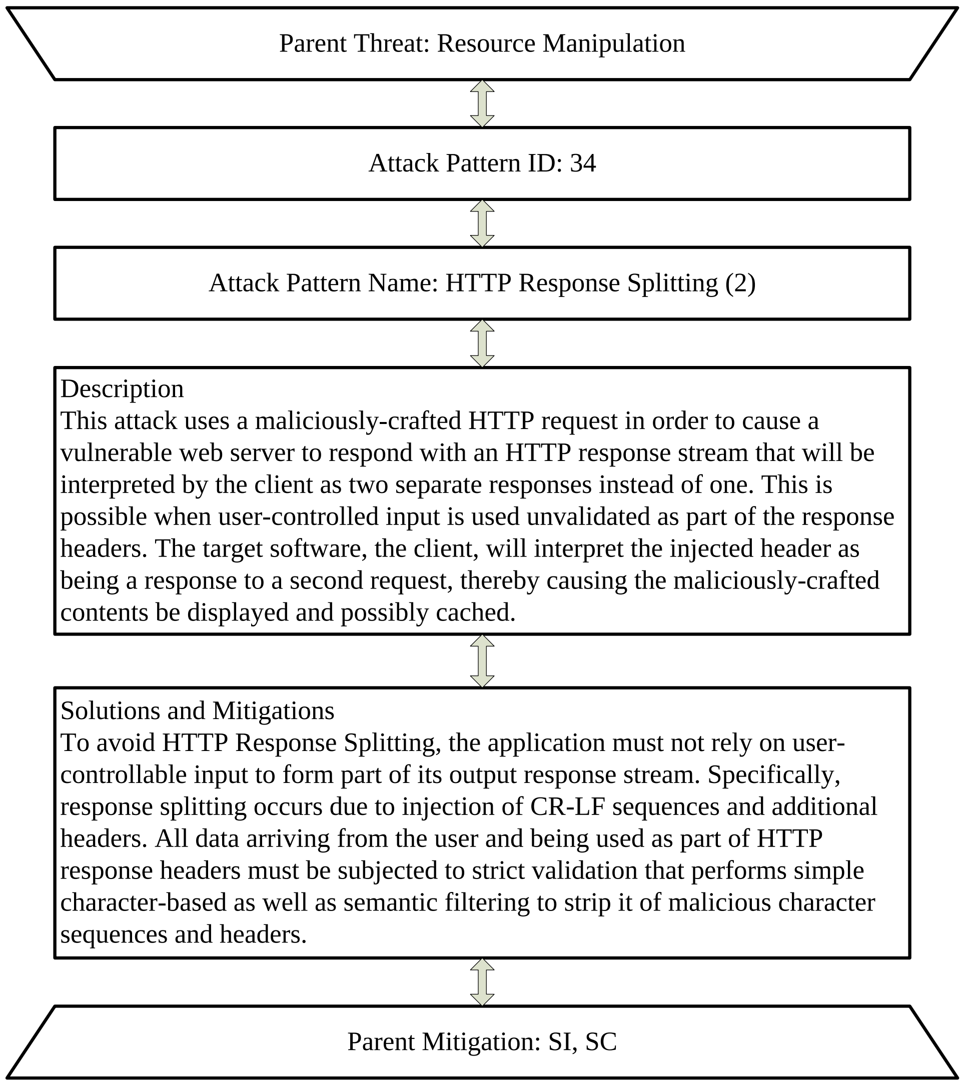
Attack Pattern ID: 33

Attack Pattern Name: HTTP Request Smuggling (3)

Description
HTTP Request Smuggling results from the discrepancies in parsing HTTP requests between HTTP entities such as web caching proxies or application firewalls. Entities such as web servers, web caching proxies, application firewalls or simple proxies often parse HTTP requests in slightly different ways. Under specific situations where there are two or more such entities in the path of the HTTP request, a specially crafted request is seen by two attacked entities as two different sets of requests. This allows certain requests to be smuggled through to a second entity without the first one realizing it.

Solutions and Mitigations
HTTP Request Smuggling is usually targeted at web servers. Therefore, in such cases, careful analysis of the entities must occur during system design prior to deployment. If there are known differences in the way the entities parse HTTP requests, the choice of entities needs consideration.
Employing an application firewall can help. However, there are instances of the firewalls being susceptible to HTTP Request Smuggling as well.

Parent Mitigation: SA, SI, SC



Parent Threat: Resource Manipulation

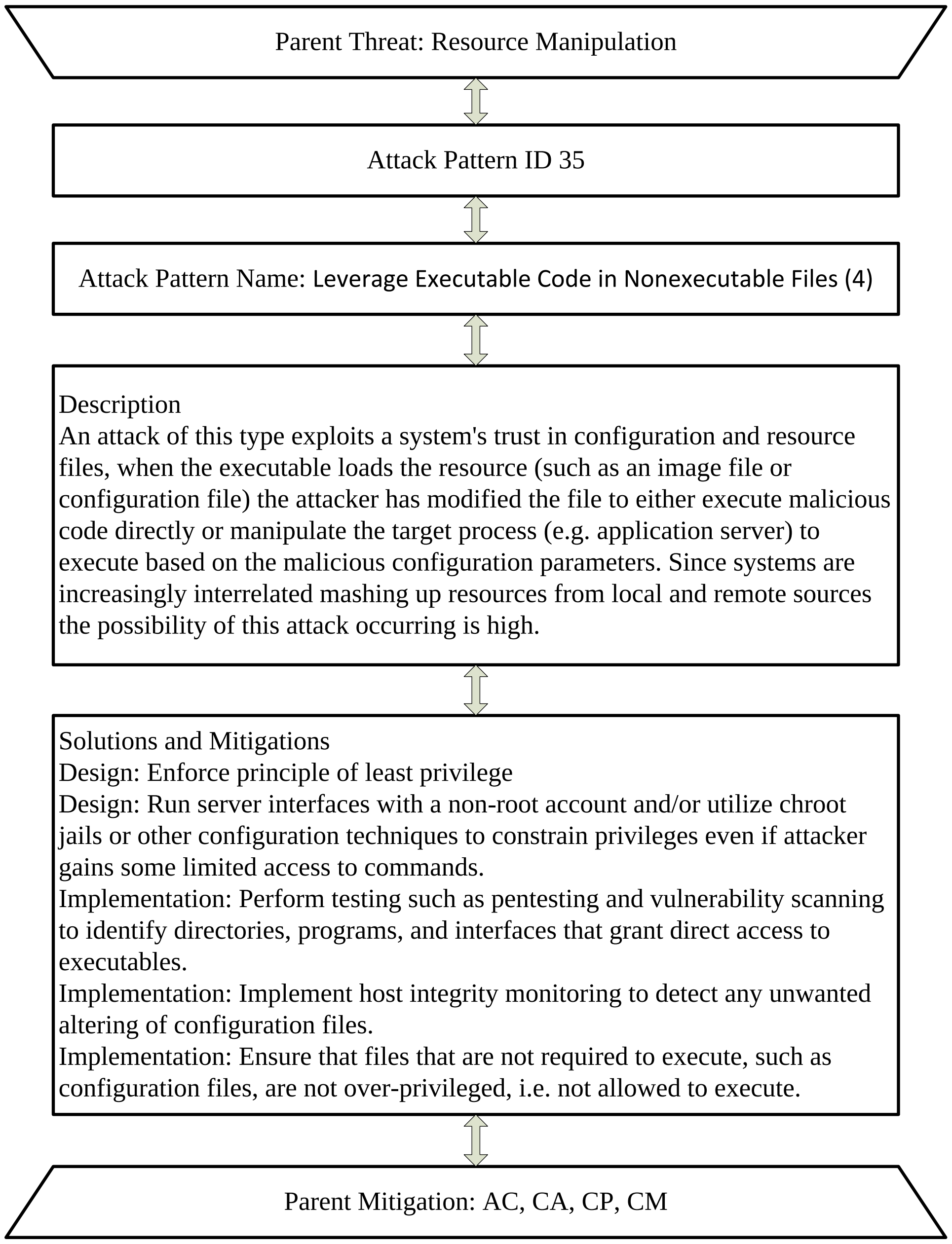
Attack Pattern ID: 34

Attack Pattern Name: HTTP Response Splitting (2)

Description
This attack uses a maliciously-crafted HTTP request in order to cause a vulnerable web server to respond with an HTTP response stream that will be interpreted by the client as two separate responses instead of one. This is possible when user-controlled input is used unvalidated as part of the response headers. The target software, the client, will interpret the injected header as being a response to a second request, thereby causing the maliciously-crafted contents be displayed and possibly cached.

Solutions and Mitigations
To avoid HTTP Response Splitting, the application must not rely on user-controllable input to form part of its output response stream. Specifically, response splitting occurs due to injection of CR-LF sequences and additional headers. All data arriving from the user and being used as part of HTTP response headers must be subjected to strict validation that performs simple character-based as well as semantic filtering to strip it of malicious character sequences and headers.

Parent Mitigation: SI, SC



Parent Threat: Resource Manipulation

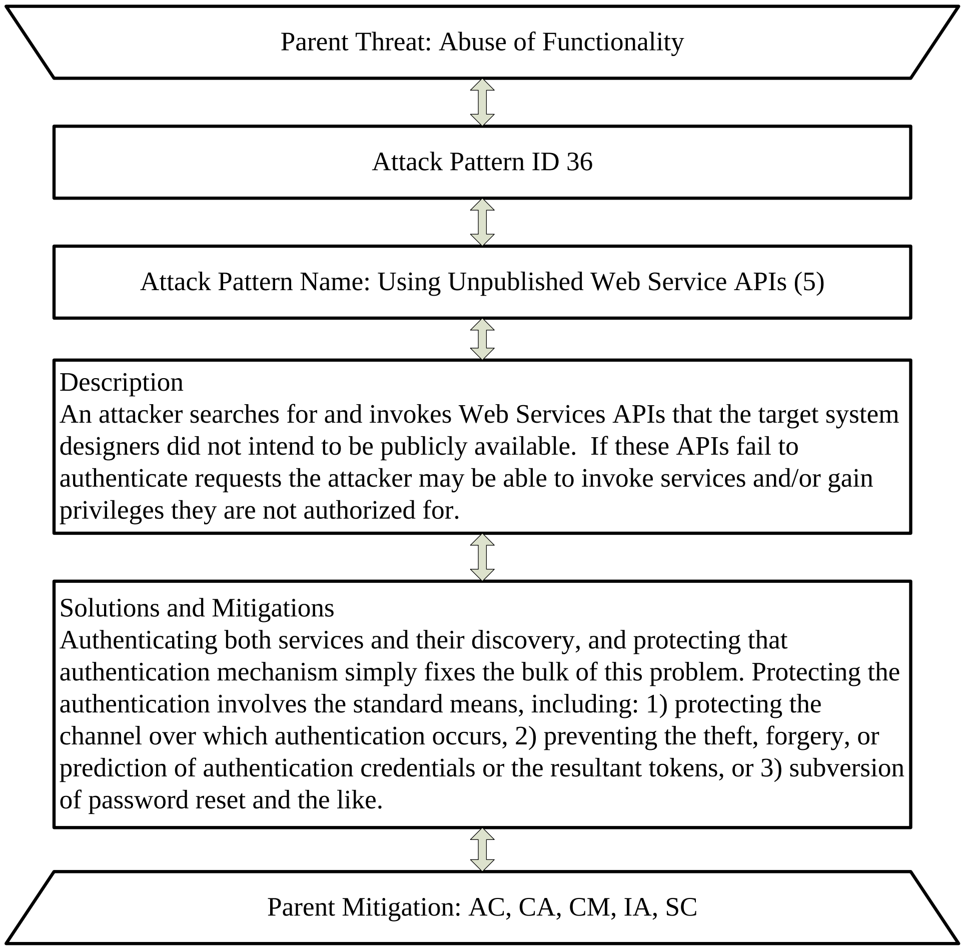
Attack Pattern ID 35

Attack Pattern Name: Leverage Executable Code in Nonexecutable Files (4)

Description
An attack of this type exploits a system's trust in configuration and resource files, when the executable loads the resource (such as an image file or configuration file) the attacker has modified the file to either execute malicious code directly or manipulate the target process (e.g. application server) to execute based on the malicious configuration parameters. Since systems are increasingly interrelated mashing up resources from local and remote sources the possibility of this attack occurring is high.

Solutions and Mitigations
Design: Enforce principle of least privilege
Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands.
Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.
Implementation: Implement host integrity monitoring to detect any unwanted altering of configuration files.
Implementation: Ensure that files that are not required to execute, such as configuration files, are not over-privileged, i.e. not allowed to execute.

Parent Mitigation: AC, CA, CP, CM



Parent Threat: Abuse of Functionality

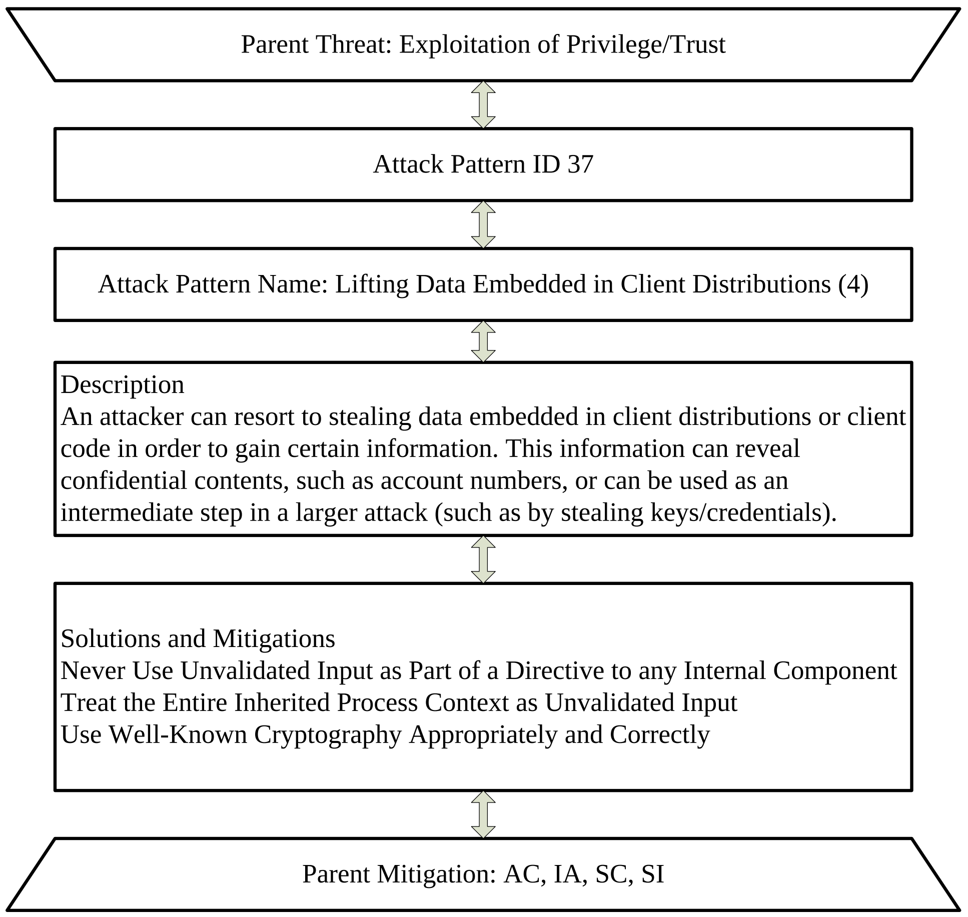
Attack Pattern ID 36

Attack Pattern Name: Using Unpublished Web Service APIs (5)

Description
An attacker searches for and invokes Web Services APIs that the target system designers did not intend to be publicly available. If these APIs fail to authenticate requests the attacker may be able to invoke services and/or gain privileges they are not authorized for.

Solutions and Mitigations
Authenticating both services and their discovery, and protecting that authentication mechanism simply fixes the bulk of this problem. Protecting the authentication involves the standard means, including: 1) protecting the channel over which authentication occurs, 2) preventing the theft, forgery, or prediction of authentication credentials or the resultant tokens, or 3) subversion of password reset and the like.

Parent Mitigation: AC, CA, CM, IA, SC



Parent Threat: Exploitation of Privilege/Trust

Attack Pattern ID 37

Attack Pattern Name: Lifting Data Embedded in Client Distributions (4)

Description
An attacker can resort to stealing data embedded in client distributions or client code in order to gain certain information. This information can reveal confidential contents, such as account numbers, or can be used as an intermediate step in a larger attack (such as by stealing keys/credentials).

Solutions and Mitigations
Never Use Unvalidated Input as Part of a Directive to any Internal Component
Treat the Entire Inherited Process Context as Unvalidated Input
Use Well-Known Cryptography Appropriately and Correctly

Parent Mitigation: AC, IA, SC, SI

Parent Threat: Spoofing

Attack Pattern ID: 38

Attack Pattern Name: Leveraging/Manipulating Configuration File Search Paths (8)

Description
This attack loads a malicious resource into a program's standard path used to bootstrap and/or provide contextual information for a program like a path variable or classpath. J2EE applications and other component based applications that are built from mutliple binaries can have very long list of dependencies to execute. If one of these libraries and/or references is controllable by the attacker then application controls can be circumvented by the attacker.
A standard UNIX path looks similar to this
/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin
If the attacker modifies the path variable to point to a locale that includes malicious resources then the user unwittingly can execute commands on the attacker's behalf:
/evildir/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin
This is a form of usurping control of the program and the attack can be done on the classpath, database resources, or any other resources built from compound parts. At runtime detection and blocking of this attack is nearly impossible, because the configuration allows execution.

Solutions and Mitigations
Enforce principle of least privilege
Ensure that the program's compound parts, including all system dependencies, classpath, path, and so on, are secured to the same or higher level assurance as the program
Host integrity monitoring

Parent Mitigation: AC, AU, CA, CM, MP, RA, SC, SI

Parent Threat: Probabilistic Techniques

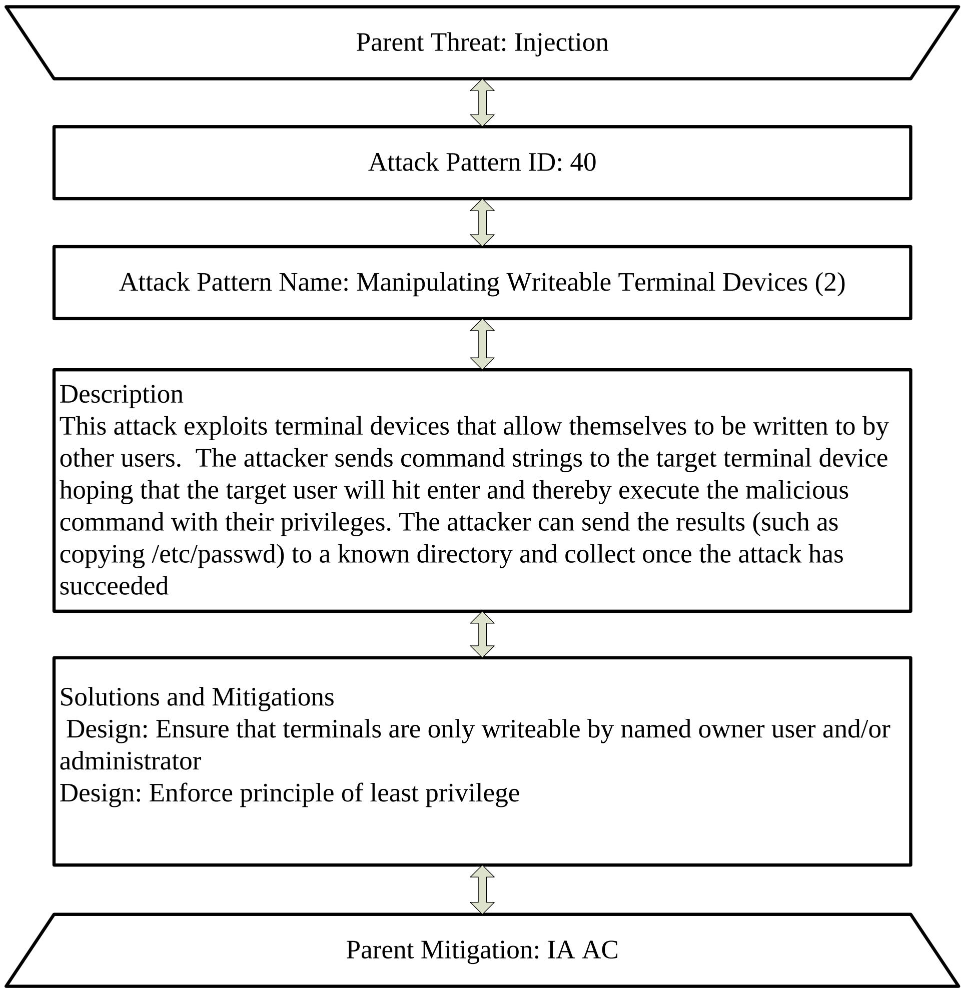
Attack Pattern ID: 39

Attack Pattern Name: Manipulating Opaque Client-based Data Tokens (6)

Description
In circumstances where an application holds important data client-side in tokens (cookies, URLs, data files, and so forth) that data can be manipulated. If client
or server-side application components reinterpret that data as authentication tokens or data (such as store item pricing or wallet information) then even opaquely manipulating
that data may bear fruit for an Attacker. In this pattern an attacker undermines the assumption that client side tokens have been adequately protected from tampering through use of encryption or obfuscation.

Solutions and Mitigations
One solution to this problem is to protect encrypted data with a CRC of some sort. If knowing who last manipulated the data is important, then using a cryptographic "message authentication code" (or hMAC) is prescribed. However, this guidance is not a panecea. In particular, any value created by (and therefore encrypted by) the client, which itself is a "malicous" value, all the protective cryptography in the world can't make the value 'correct' again. Put simply, if the client has control over the whole process of generating and encoding the value--then simply protecting its integrity doesn't help.
Make sure to protect client side authentication tokens for confidentiality (encryption) and integrity (signed hash)
Make sure that all session tokens use a good source of randomness
Perform validation on the server side to make sure that client side data tokens are consistent with what is expected.

Parent Mitigation: AU, IA, SI, CM, SA, SC


Parent Threat: Injection
Attack Pattern ID: 40
Attack Pattern Name: Manipulating Writeable Terminal Devices (2)
Description
This attack exploits terminal devices that allow themselves to be written to by other users. The attacker sends command strings to the target terminal device hoping that the target user will hit enter and thereby execute the malicious command with their privileges. The attacker can send the results (such as copying /etc/passwd) to a known directory and collect once the attack has succeeded
Solutions and Mitigations
Design: Ensure that terminals are only writeable by named owner user and/or administrator
Design: Enforce principle of least privilege
Parent Mitigation: IA AC

Parent Threat: Injection

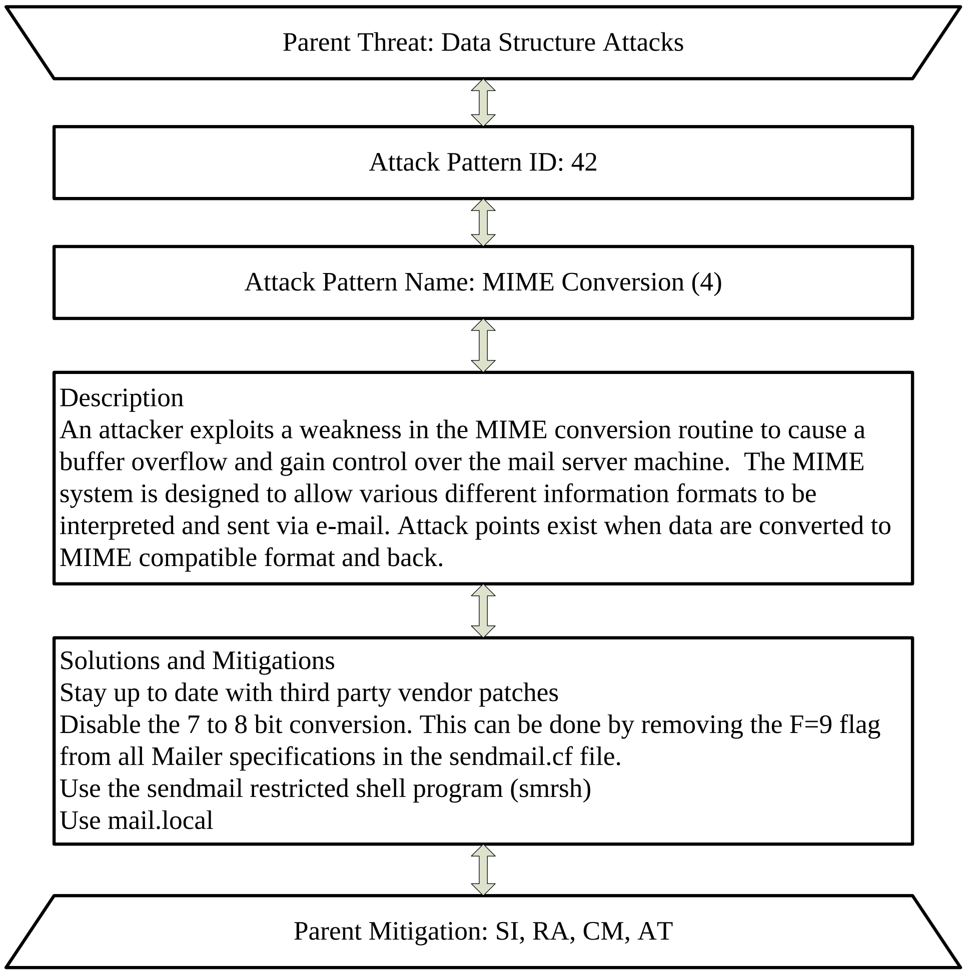
Attack Pattern ID: 41

Attack Pattern Name: Using Meta-characters in E-mail Headers to Inject Malicious Payloads (4)

Description
This type of attack involves an attacker leveraging meta-characters in email headers to inject improper behavior into email programs.
Email software has become increasingly sophisticated and feature-rich. In addition, email applications are ubiquitous and connected directly to the Web making them ideal targets to launch and propagate attacks. As the user demand for new functionality in email applications grows, they become more like browsers with complex rendering and plug in routines. As more email functionality is included and abstracted from the user, this creates opportunities for attackers. Virtually all email applications do not list email header information by default, however the email header contains valuable attacker vectors for the attacker to exploit particularly if the behavior of the email client application is known. Meta-characters are hidden from the user, but can containt scripts, enumerations, probes, and other attacks against the user's system.

Solutions and Mitigations
Design: Perform validation on email header data
Implementation: Implement email filtering solutions on mail server or on MTA, relay server.
Implementation: Mail servers that perform strict validation may catch these attacks, because metacharacters are not allowed in many header variables such as dns names

Parent Mitigation: AU, IA, SC, SI



Parent Threat: Data Structure Attacks

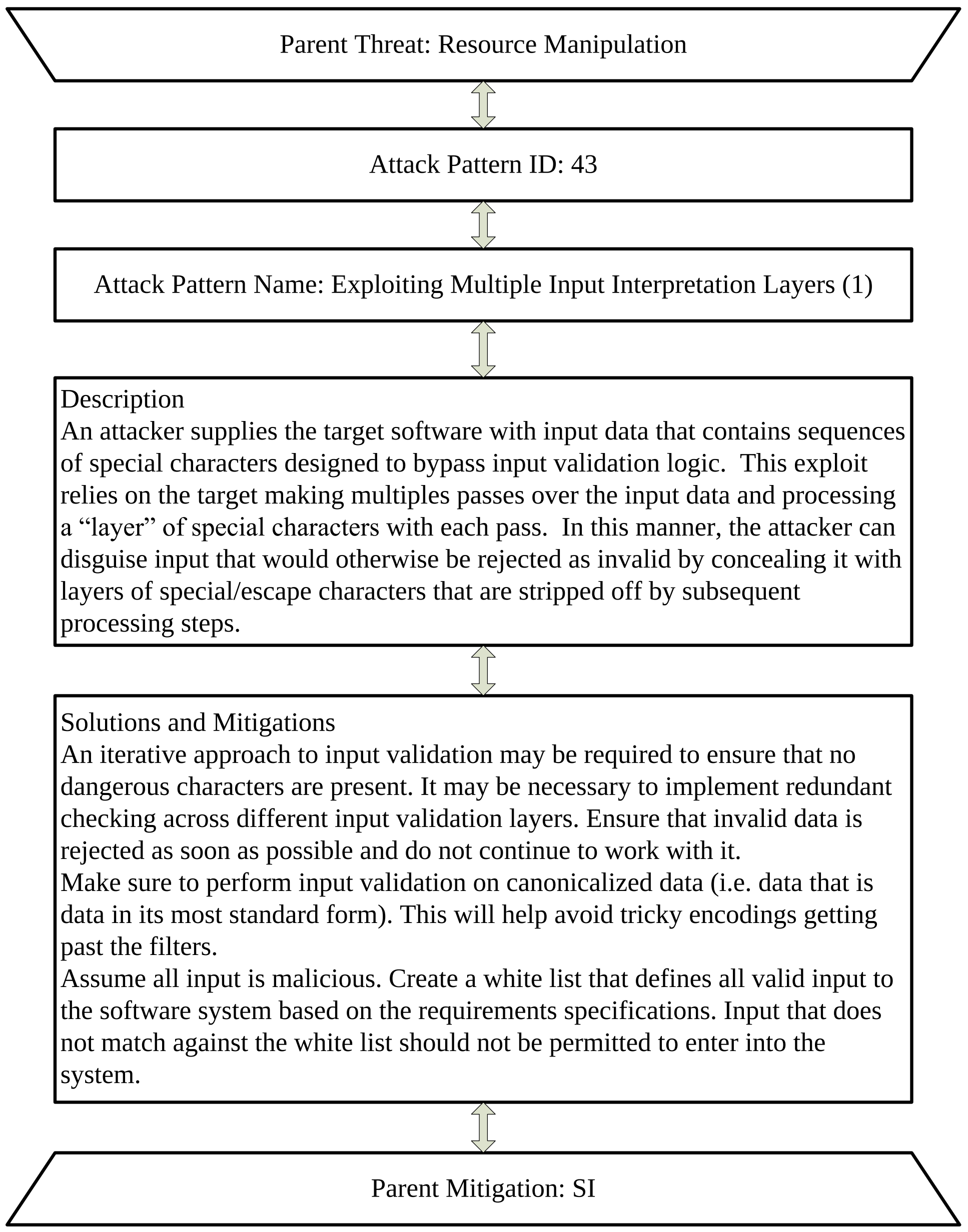
Attack Pattern ID: 42

Attack Pattern Name: MIME Conversion (4)

Description
An attacker exploits a weakness in the MIME conversion routine to cause a buffer overflow and gain control over the mail server machine. The MIME system is designed to allow various different information formats to be interpreted and sent via e-mail. Attack points exist when data are converted to MIME compatible format and back.

Solutions and Mitigations
Stay up to date with third party vendor patches
Disable the 7 to 8 bit conversion. This can be done by removing the F=9 flag from all Mailer specifications in the sendmail.cf file.
Use the sendmail restricted shell program (smrsh)
Use mail.local

Parent Mitigation: SI, RA, CM, AT



Parent Threat: Resource Manipulation

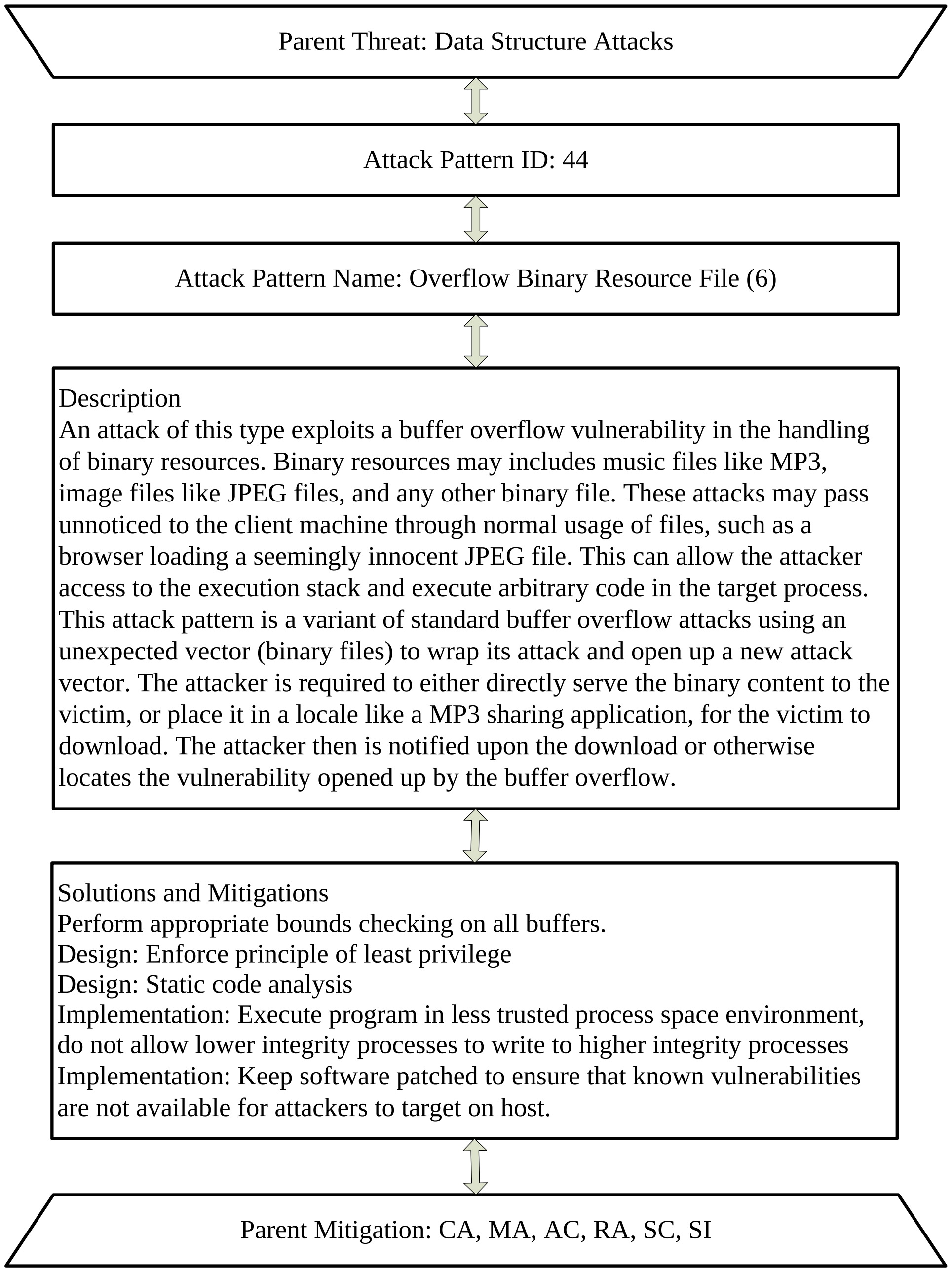
Attack Pattern ID: 43

Attack Pattern Name: Exploiting Multiple Input Interpretation Layers (1)

Description
An attacker supplies the target software with input data that contains sequences of special characters designed to bypass input validation logic. This exploit relies on the target making multiples passes over the input data and processing a “layer” of special characters with each pass. In this manner, the attacker can disguise input that would otherwise be rejected as invalid by concealing it with layers of special/escape characters that are stripped off by subsequent processing steps.

Solutions and Mitigations
An iterative approach to input validation may be required to ensure that no dangerous characters are present. It may be necessary to implement redundant checking across different input validation layers. Ensure that invalid data is rejected as soon as possible and do not continue to work with it.
Make sure to perform input validation on canonicalized data (i.e. data that is data in its most standard form). This will help avoid tricky encodings getting past the filters.
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.

Parent Mitigation: SI



Parent Threat: Data Structure Attacks

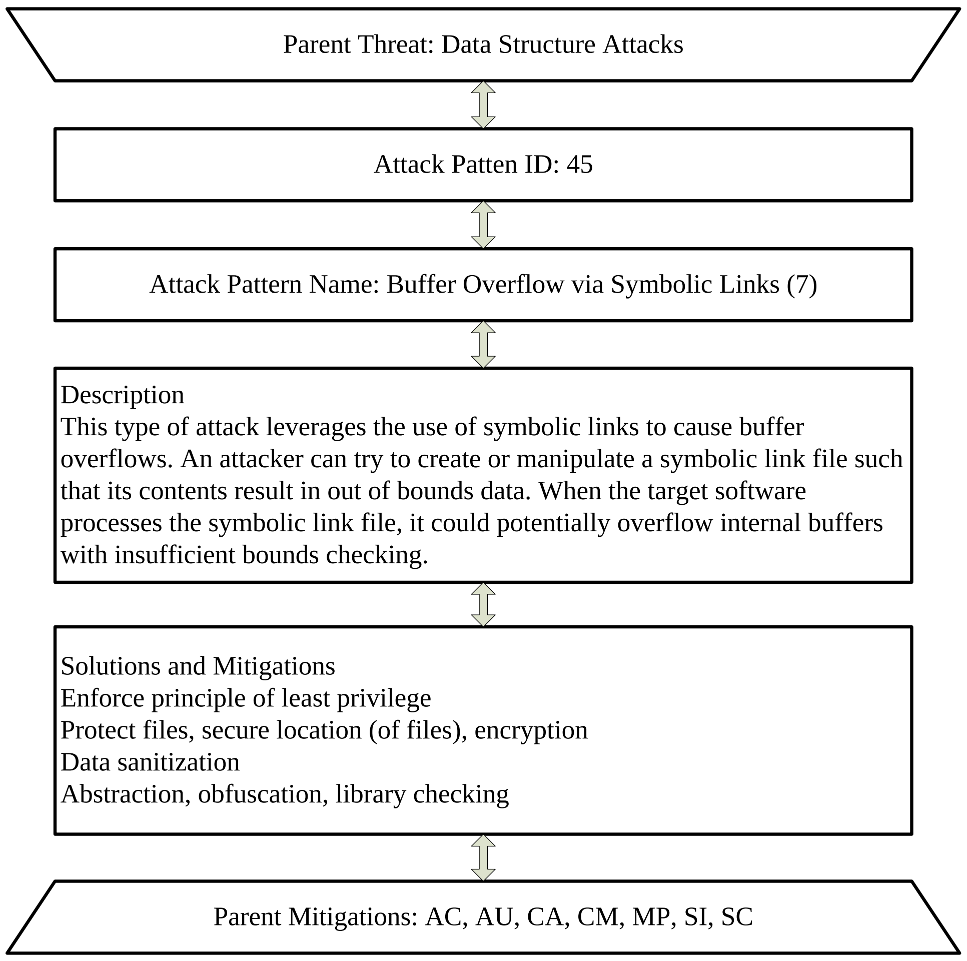
Attack Pattern ID: 44

Attack Pattern Name: Overflow Binary Resource File (6)

Description
An attack of this type exploits a buffer overflow vulnerability in the handling of binary resources. Binary resources may includes music files like MP3, image files like JPEG files, and any other binary file. These attacks may pass unnoticed to the client machine through normal usage of files, such as a browser loading a seemingly innocent JPEG file. This can allow the attacker access to the execution stack and execute arbitrary code in the target process. This attack pattern is a variant of standard buffer overflow attacks using an unexpected vector (binary files) to wrap its attack and open up a new attack vector. The attacker is required to either directly serve the binary content to the victim, or place it in a locale like a MP3 sharing application, for the victim to download. The attacker then is notified upon the download or otherwise locates the vulnerability opened up by the buffer overflow.

Solutions and Mitigations
Perform appropriate bounds checking on all buffers.
Design: Enforce principle of least privilege
Design: Static code analysis
Implementation: Execute program in less trusted process space environment, do not allow lower integrity processes to write to higher integrity processes
Implementation: Keep software patched to ensure that known vulnerabilities are not available for attackers to target on host.

Parent Mitigation: CA, MA, AC, RA, SC, SI



Parent Threat: Data Structure Attacks

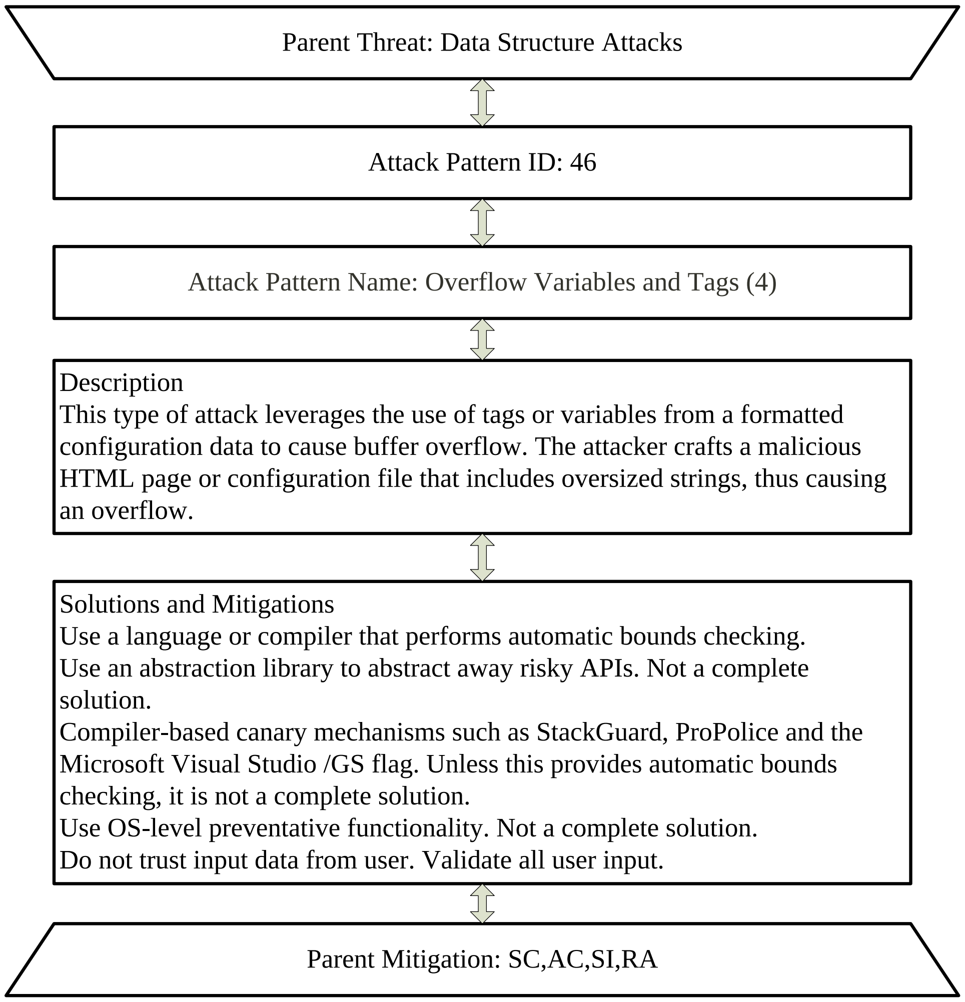
Attack Patten ID: 45

Attack Pattern Name: Buffer Overflow via Symbolic Links (7)

Description
This type of attack leverages the use of symbolic links to cause buffer overflows. An attacker can try to create or manipulate a symbolic link file such that its contents result in out of bounds data. When the target software processes the symbolic link file, it could potentially overflow internal buffers with insufficient bounds checking.

Solutions and Mitigations
Enforce principle of least privilege
Protect files, secure location (of files), encryption
Data sanitization
Abstraction, obfuscation, library checking

Parent Mitigations: AC, AU, CA, CM, MP, SI, SC



Parent Threat: Data Structure Attacks

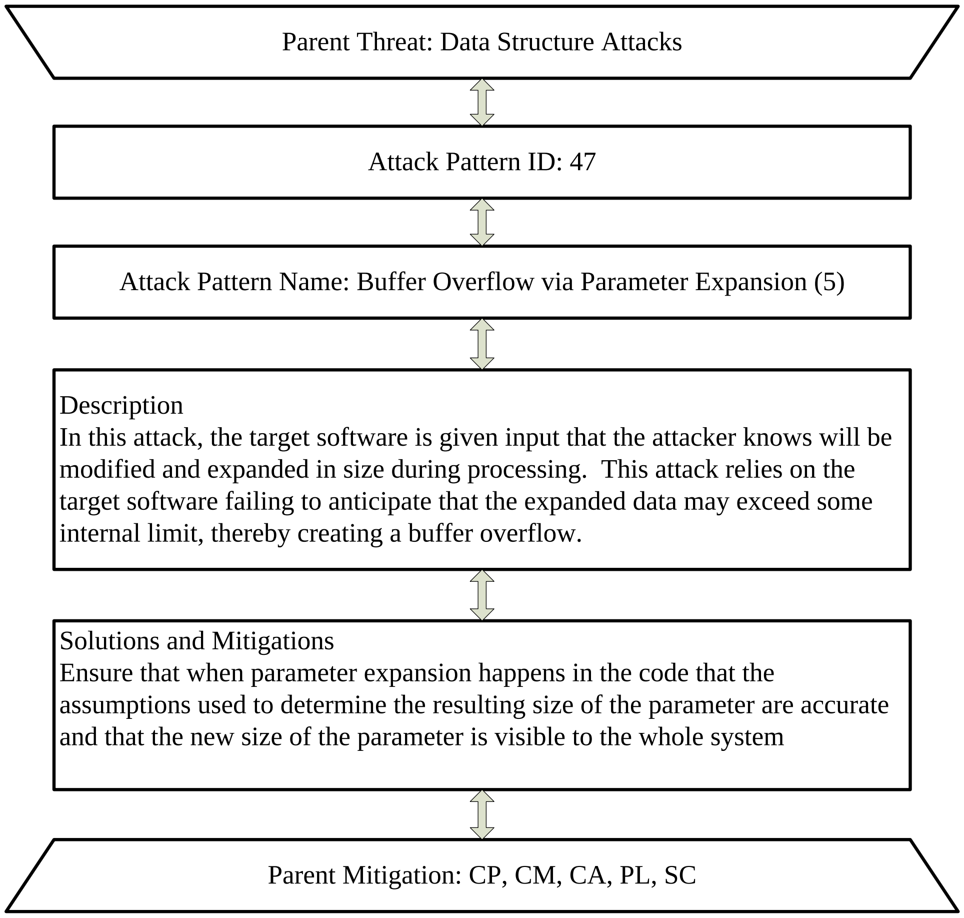
Attack Pattern ID: 46

Attack Pattern Name: Overflow Variables and Tags (4)

Description
This type of attack leverages the use of tags or variables from a formatted configuration data to cause buffer overflow. The attacker crafts a malicious HTML page or configuration file that includes oversized strings, thus causing an overflow.

Solutions and Mitigations
Use a language or compiler that performs automatic bounds checking.
Use an abstraction library to abstract away risky APIs. Not a complete solution.
Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
Use OS-level preventative functionality. Not a complete solution.
Do not trust input data from user. Validate all user input.

Parent Mitigation: SC,AC,SI,RA



Parent Threat: Data Structure Attacks

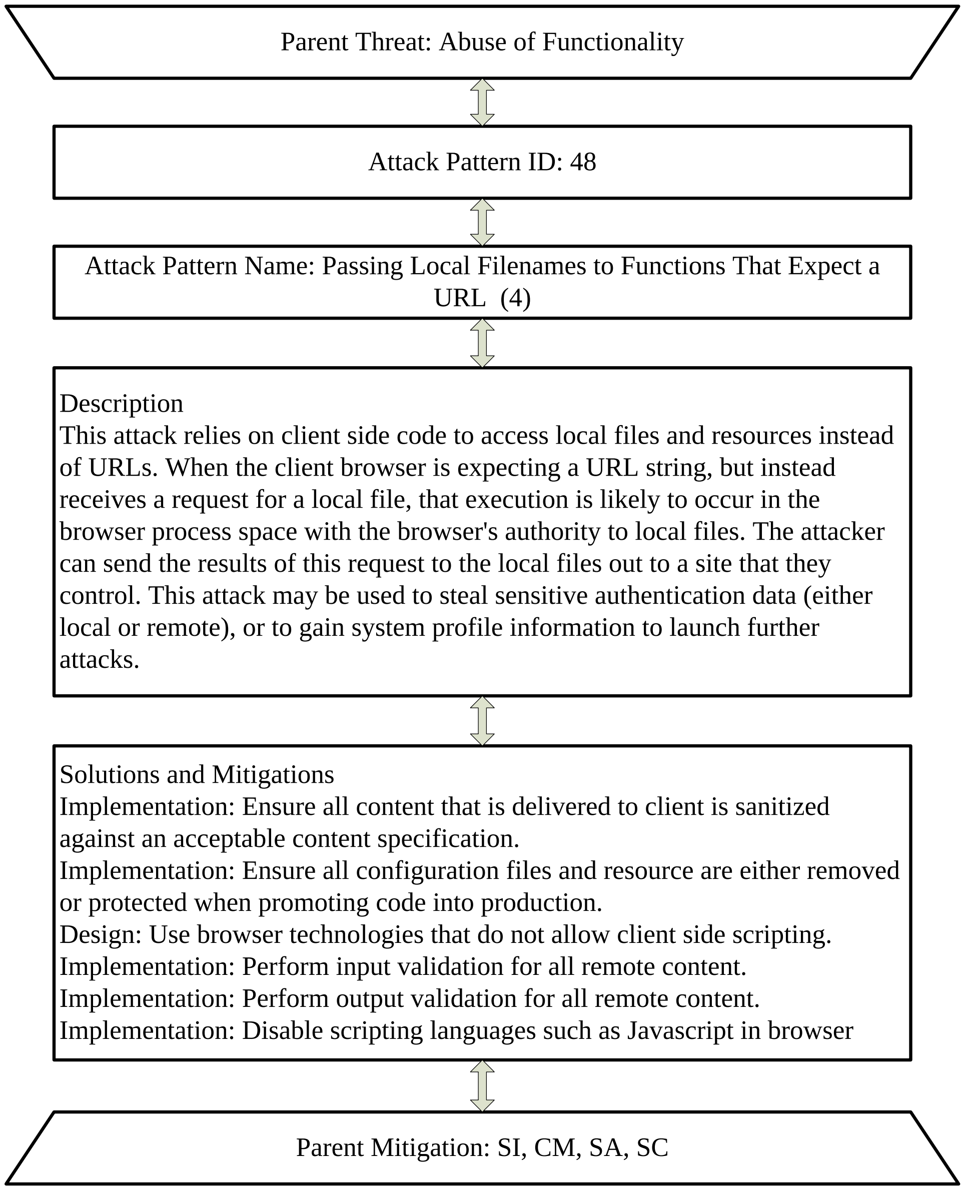
Attack Pattern ID: 47

Attack Pattern Name: Buffer Overflow via Parameter Expansion (5)

Description
In this attack, the target software is given input that the attacker knows will be modified and expanded in size during processing. This attack relies on the target software failing to anticipate that the expanded data may exceed some internal limit, thereby creating a buffer overflow.

Solutions and Mitigations
Ensure that when parameter expansion happens in the code that the assumptions used to determine the resulting size of the parameter are accurate and that the new size of the parameter is visible to the whole system

Parent Mitigation: CP, CM, CA, PL, SC



Parent Threat: Abuse of Functionality

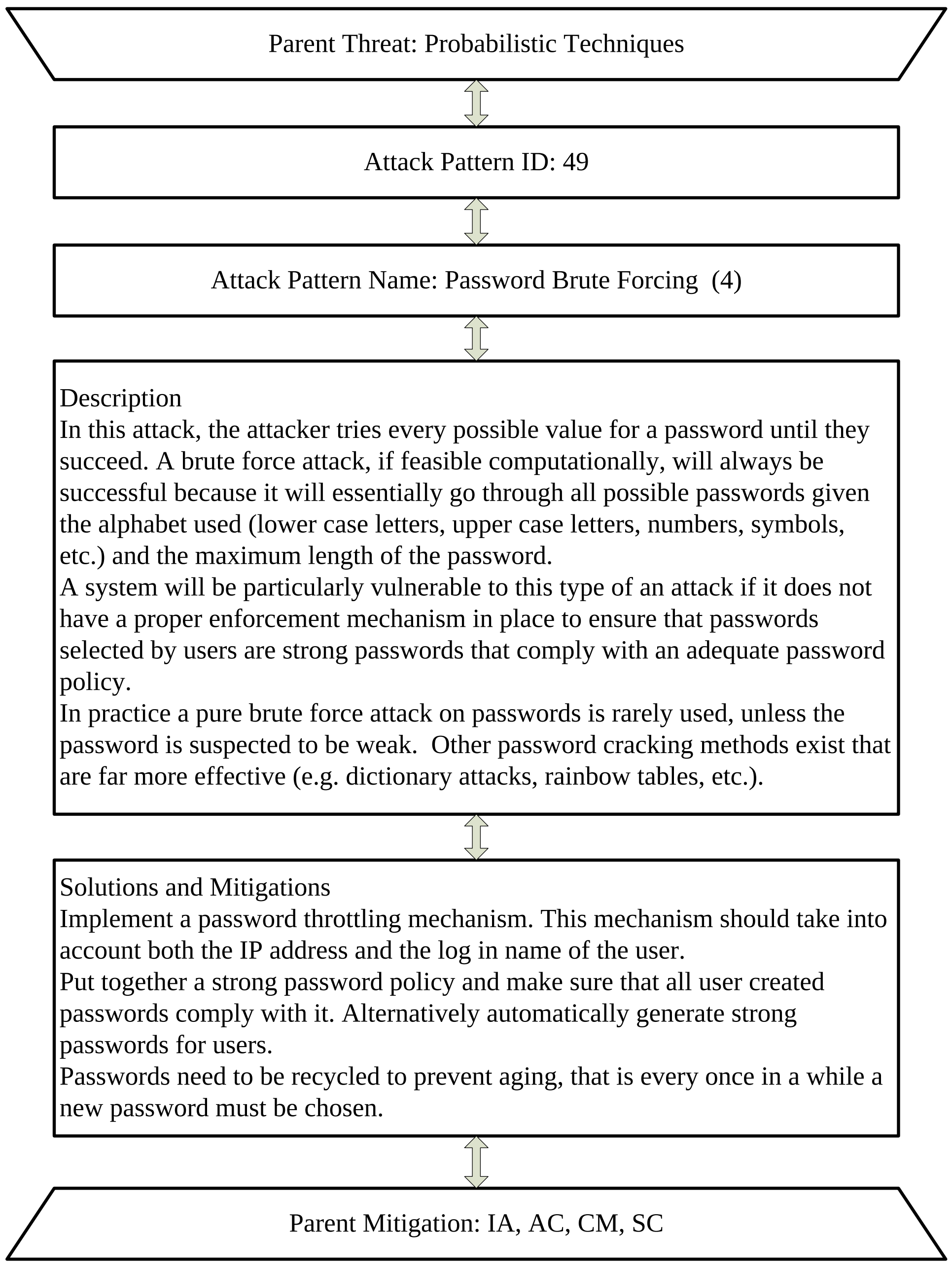
Attack Pattern ID: 48

Attack Pattern Name: Passing Local Filenames to Functions That Expect a URL (4)

Description
This attack relies on client side code to access local files and resources instead of URLs. When the client browser is expecting a URL string, but instead receives a request for a local file, that execution is likely to occur in the browser process space with the browser's authority to local files. The attacker can send the results of this request to the local files out to a site that they control. This attack may be used to steal sensitive authentication data (either local or remote), or to gain system profile information to launch further attacks.

Solutions and Mitigations
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Ensure all configuration files and resource are either removed or protected when promoting code into production.
Design: Use browser technologies that do not allow client side scripting.
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser

Parent Mitigation: SI, CM, SA, SC



Parent Threat: Probabilistic Techniques

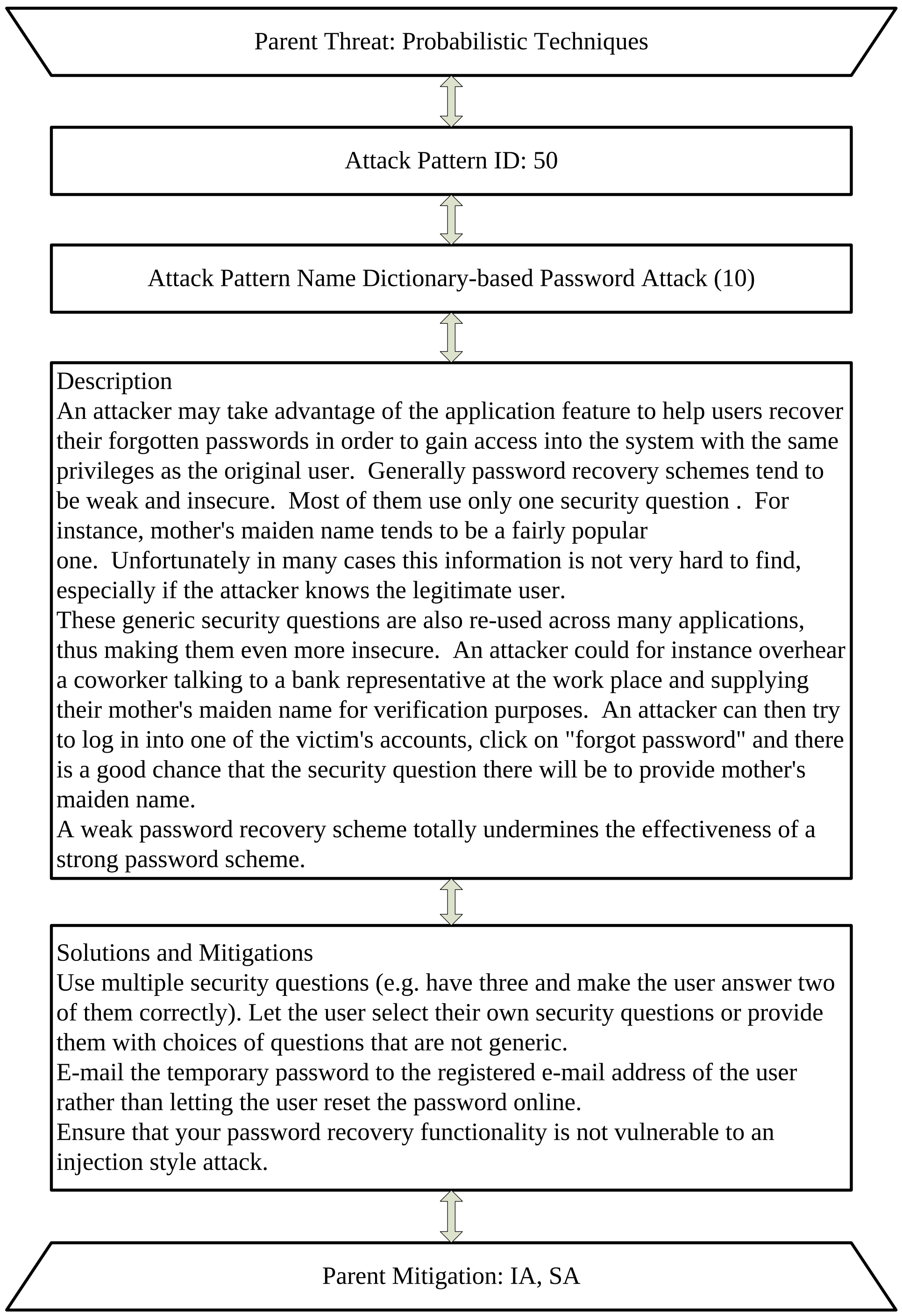
Attack Pattern ID: 49

Attack Pattern Name: Password Brute Forcing (4)

Description
In this attack, the attacker tries every possible value for a password until they succeed. A brute force attack, if feasible computationally, will always be successful because it will essentially go through all possible passwords given the alphabet used (lower case letters, upper case letters, numbers, symbols, etc.) and the maximum length of the password.
A system will be particularly vulnerable to this type of an attack if it does not have a proper enforcement mechanism in place to ensure that passwords selected by users are strong passwords that comply with an adequate password policy.
In practice a pure brute force attack on passwords is rarely used, unless the password is suspected to be weak. Other password cracking methods exist that are far more effective (e.g. dictionary attacks, rainbow tables, etc.).

Solutions and Mitigations
Implement a password throttling mechanism. This mechanism should take into account both the IP address and the log in name of the user.
Put together a strong password policy and make sure that all user created passwords comply with it. Alternatively automatically generate strong passwords for users.
Passwords need to be recycled to prevent aging, that is every once in a while a new password must be chosen.

Parent Mitigation: IA, AC, CM, SC



Parent Threat: Probabilistic Techniques

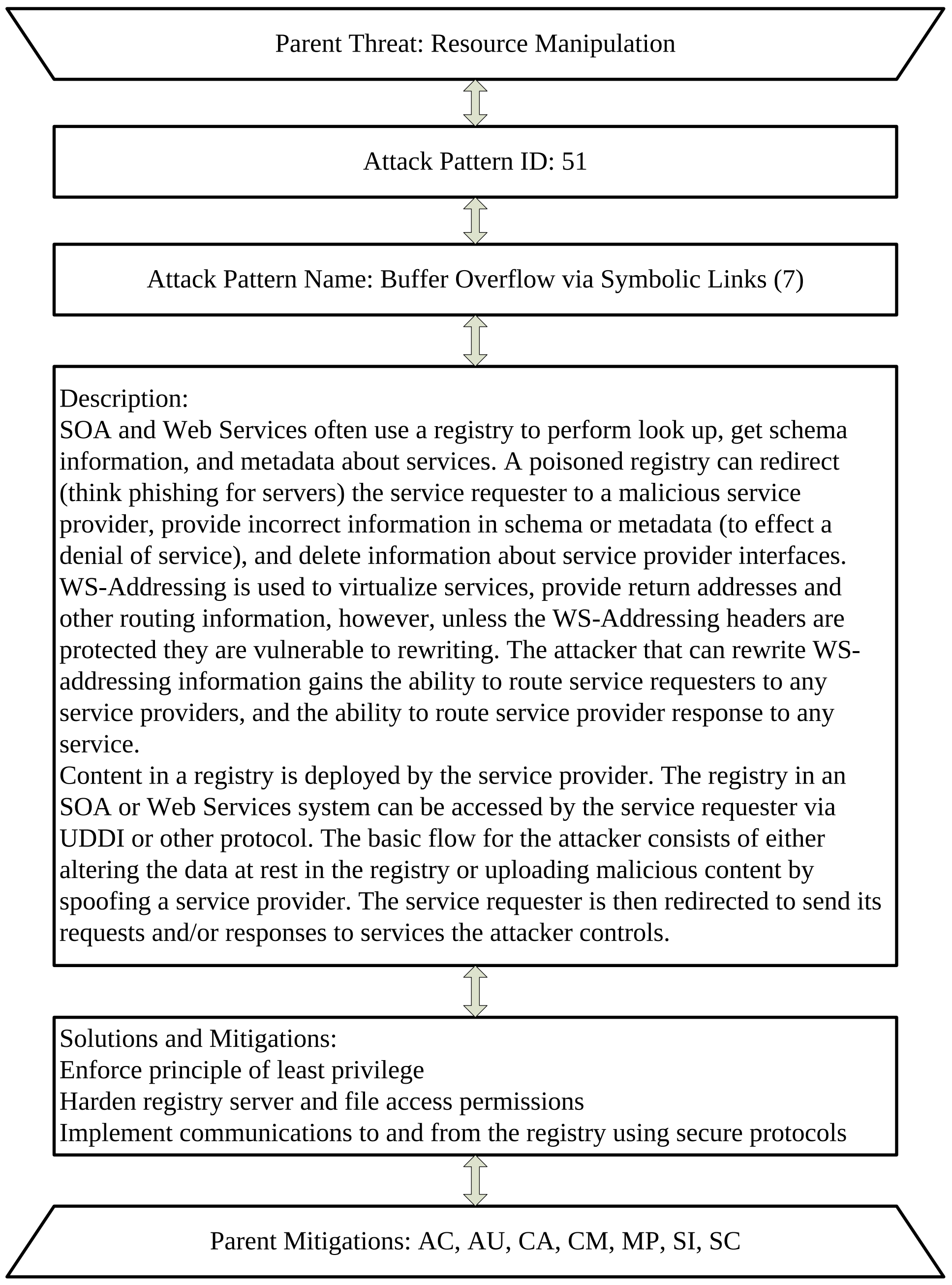
Attack Pattern ID: 50

Attack Pattern Name Dictionary-based Password Attack (10)

Description
An attacker may take advantage of the application feature to help users recover their forgotten passwords in order to gain access into the system with the same privileges as the original user. Generally password recovery schemes tend to be weak and insecure. Most of them use only one security question . For instance, mother's maiden name tends to be a fairly popular one. Unfortunately in many cases this information is not very hard to find, especially if the attacker knows the legitimate user.
These generic security questions are also re-used across many applications, thus making them even more insecure. An attacker could for instance overhear a coworker talking to a bank representative at the work place and supplying their mother's maiden name for verification purposes. An attacker can then try to log in into one of the victim's accounts, click on "forgot password" and there is a good chance that the security question there will be to provide mother's maiden name.
A weak password recovery scheme totally undermines the effectiveness of a strong password scheme.

Solutions and Mitigations
Use multiple security questions (e.g. have three and make the user answer two of them correctly). Let the user select their own security questions or provide them with choices of questions that are not generic.
E-mail the temporary password to the registered e-mail address of the user rather than letting the user reset the password online.
Ensure that your password recovery functionality is not vulnerable to an injection style attack.

Parent Mitigation: IA, SA



Parent Threat: Resource Manipulation

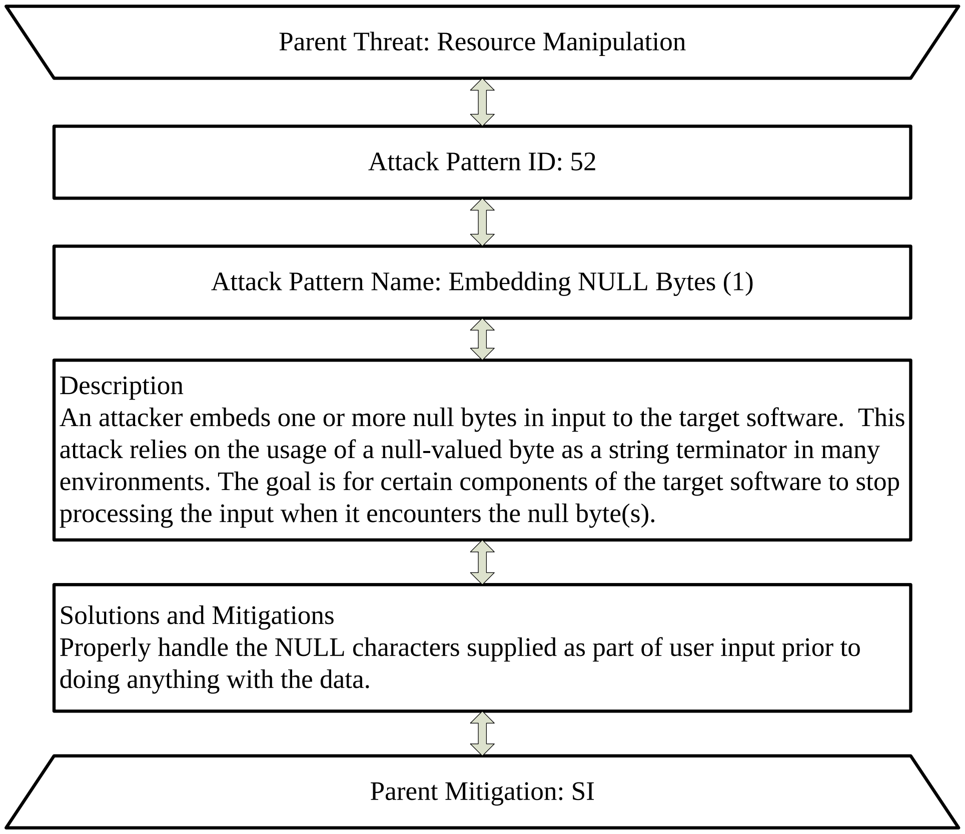
Attack Pattern ID: 51

Attack Pattern Name: Buffer Overflow via Symbolic Links (7)

Description:
SOA and Web Services often use a registry to perform look up, get schema information, and metadata about services. A poisoned registry can redirect (think phishing for servers) the service requester to a malicious service provider, provide incorrect information in schema or metadata (to effect a denial of service), and delete information about service provider interfaces. WS-Addressing is used to virtualize services, provide return addresses and other routing information, however, unless the WS-Addressing headers are protected they are vulnerable to rewriting. The attacker that can rewrite WS-addressing information gains the ability to route service requesters to any service providers, and the ability to route service provider response to any service.
Content in a registry is deployed by the service provider. The registry in an SOA or Web Services system can be accessed by the service requester via UDDI or other protocol. The basic flow for the attacker consists of either altering the data at rest in the registry or uploading malicious content by spoofing a service provider. The service requester is then redirected to send its requests and/or responses to services the attacker controls.

Solutions and Mitigations:
Enforce principle of least privilege
Harden registry server and file access permissions
Implement communications to and from the registry using secure protocols

Parent Mitigations: AC, AU, CA, CM, MP, SI, SC



Parent Threat: Resource Manipulation

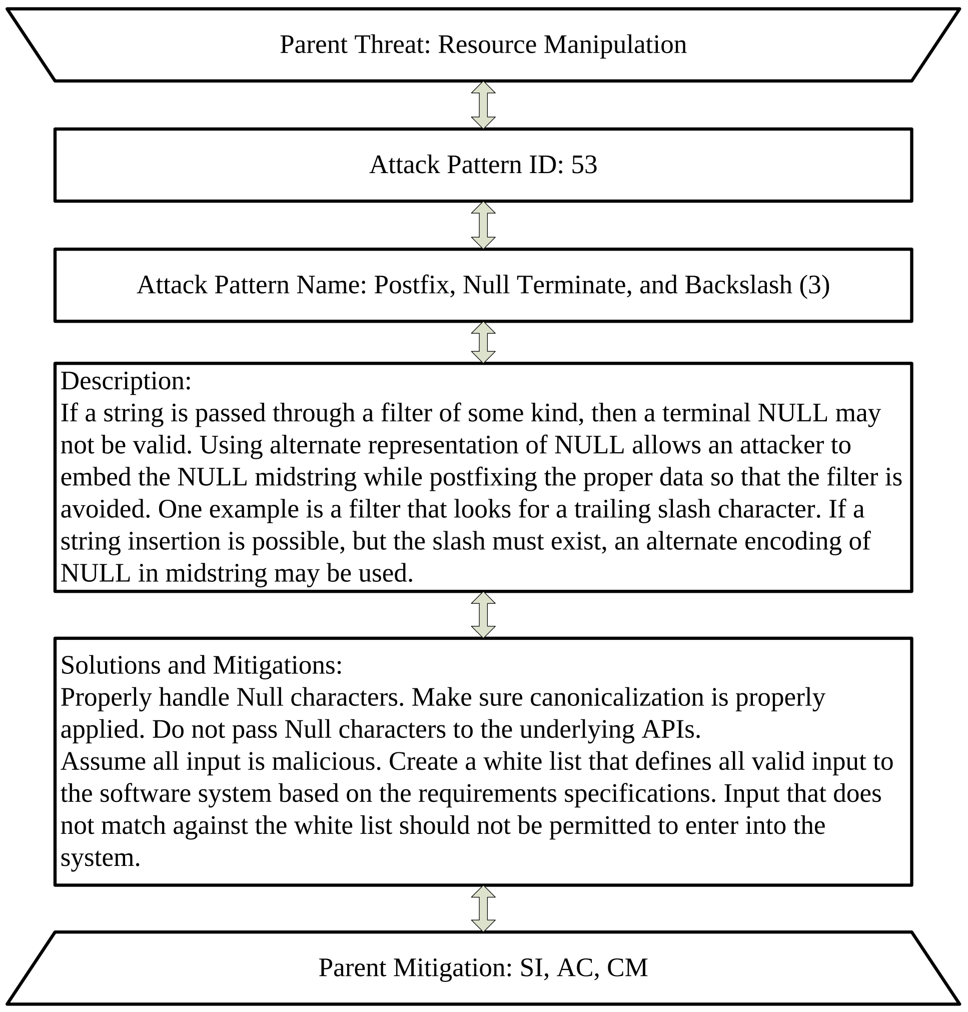
Attack Pattern ID: 52

Attack Pattern Name: Embedding NULL Bytes (1)

Description
An attacker embeds one or more null bytes in input to the target software. This attack relies on the usage of a null-valued byte as a string terminator in many environments. The goal is for certain components of the target software to stop processing the input when it encounters the null byte(s).

Solutions and Mitigations
Properly handle the NULL characters supplied as part of user input prior to doing anything with the data.

Parent Mitigation: SI



Parent Threat: Resource Manipulation

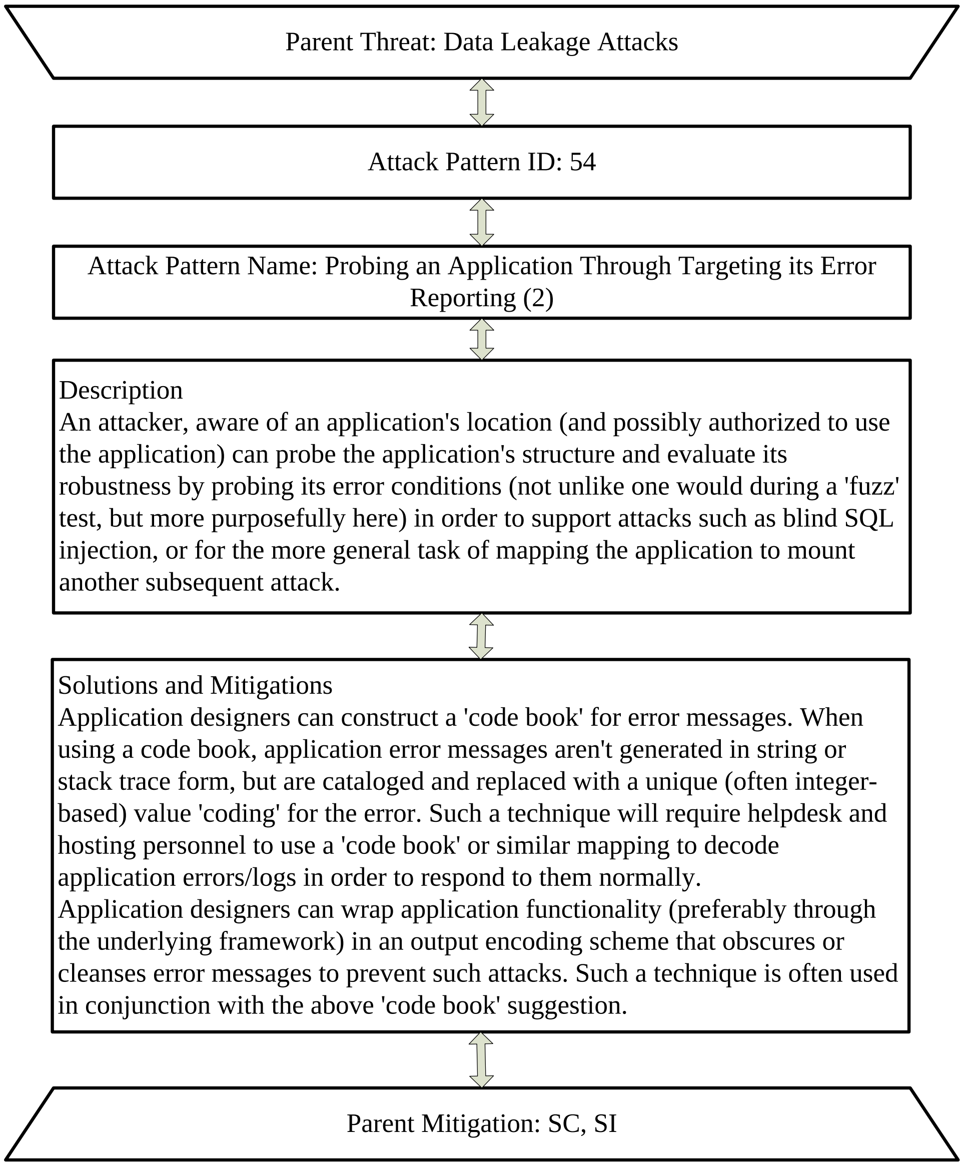
Attack Pattern ID: 53

Attack Pattern Name: Postfix, Null Terminate, and Backslash (3)

Description:
If a string is passed through a filter of some kind, then a terminal NULL may not be valid. Using alternate representation of NULL allows an attacker to embed the NULL midstring while postfixing the proper data so that the filter is avoided. One example is a filter that looks for a trailing slash character. If a string insertion is possible, but the slash must exist, an alternate encoding of NULL in midstring may be used.

Solutions and Mitigations:
Properly handle Null characters. Make sure canonicalization is properly applied. Do not pass Null characters to the underlying APIs.
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.

Parent Mitigation: SI, AC, CM



Parent Threat: Data Leakage Attacks

Attack Pattern ID: 54

Attack Pattern Name: Probing an Application Through Targeting its Error Reporting (2)

Description
An attacker, aware of an application's location (and possibly authorized to use the application) can probe the application's structure and evaluate its robustness by probing its error conditions (not unlike one would during a 'fuzz' test, but more purposefully here) in order to support attacks such as blind SQL injection, or for the more general task of mapping the application to mount another subsequent attack.

Solutions and Mitigations
Application designers can construct a 'code book' for error messages. When using a code book, application error messages aren't generated in string or stack trace form, but are cataloged and replaced with a unique (often integer-based) value 'coding' for the error. Such a technique will require helpdesk and hosting personnel to use a 'code book' or similar mapping to decode application errors/logs in order to respond to them normally.
Application designers can wrap application functionality (preferably through the underlying framework) in an output encoding scheme that obscures or cleanses error messages to prevent such attacks. Such a technique is often used in conjunction with the above 'code book' suggestion.

Parent Mitigation: SC, SI

Parent Threat: Probabilistic Techniques

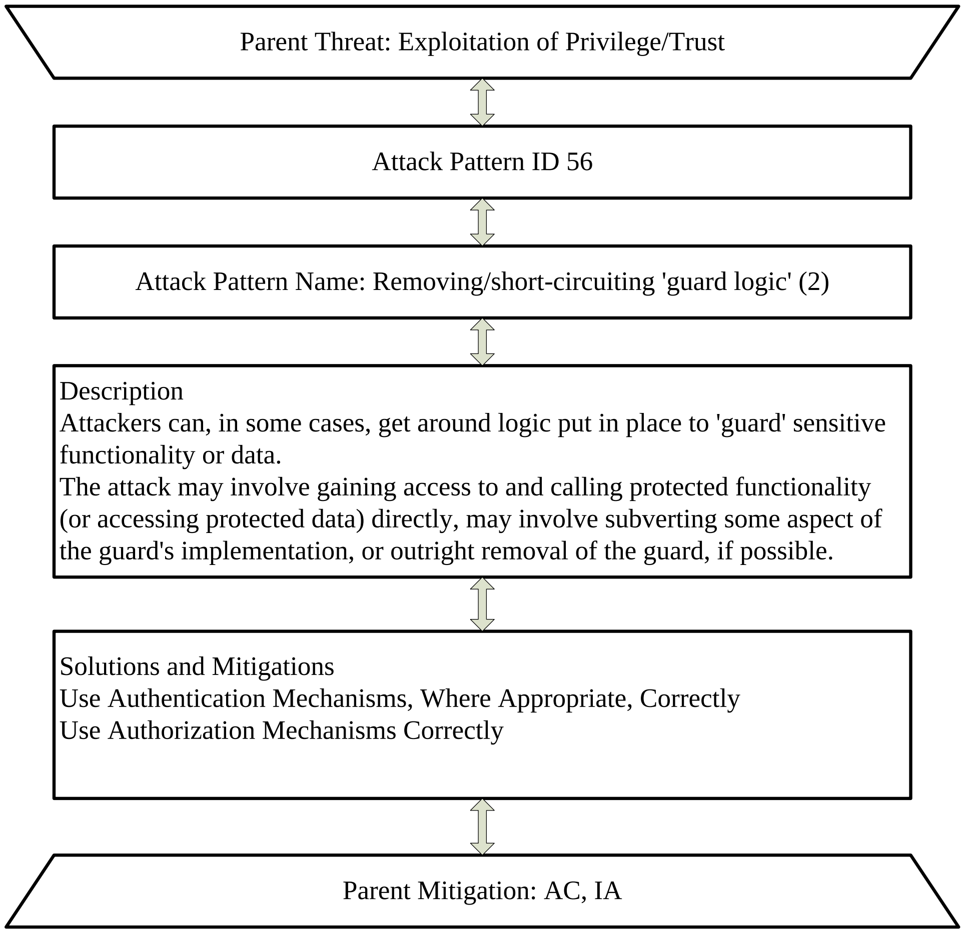
Attack Pattern ID: 55

Attack Pattern Name: Rainbow Table Password Cracking (3)

Description
An attacker gets access to the database table where hashes of passwords are stored. He then uses a rainbow table of precomputed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system.
A password rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table.
Constructing the rainbow tables takes a very long time and is computationally expensive. A separate table needs to be constrcuted for the various hash algorithms (e.g. SHA1, MD5, etc.). However, once a rainbow table is computed, it can be very effective in cracking the passwords that have been hashed without the use of salt.

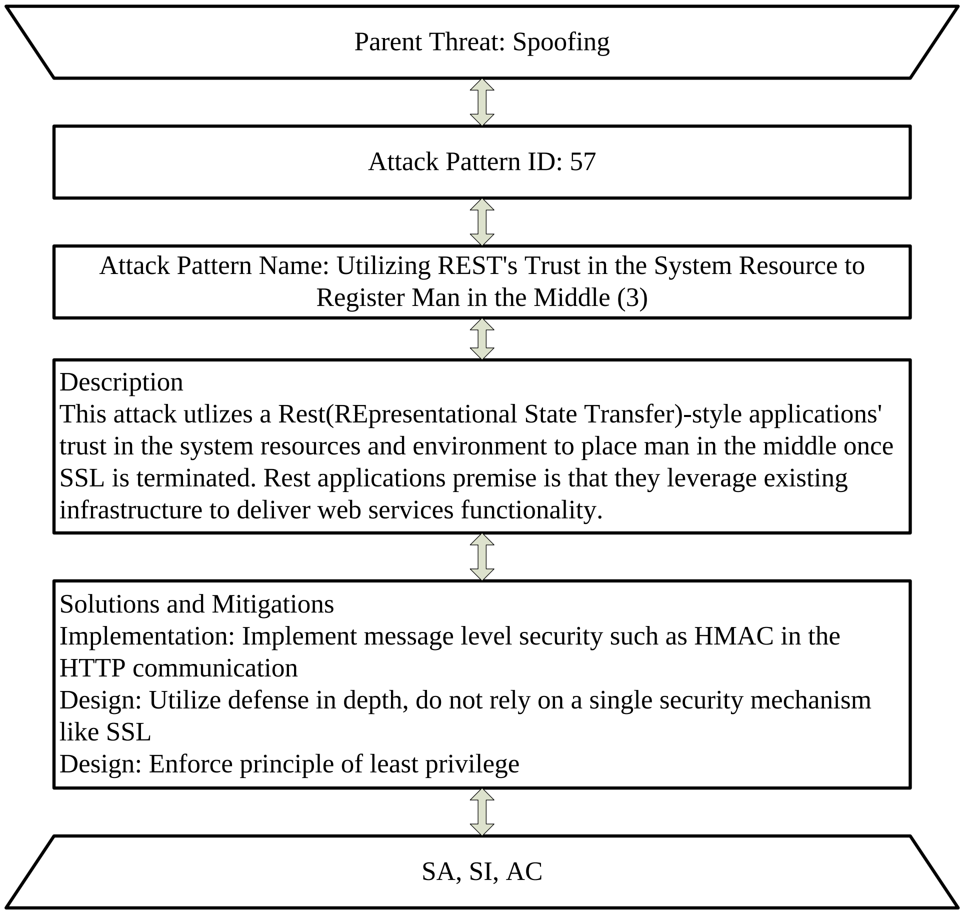
Solutions and Mitigations
Use salt when computing password hashes. That is, concatenate the salt (random bits) with the original password prior to hashing it.

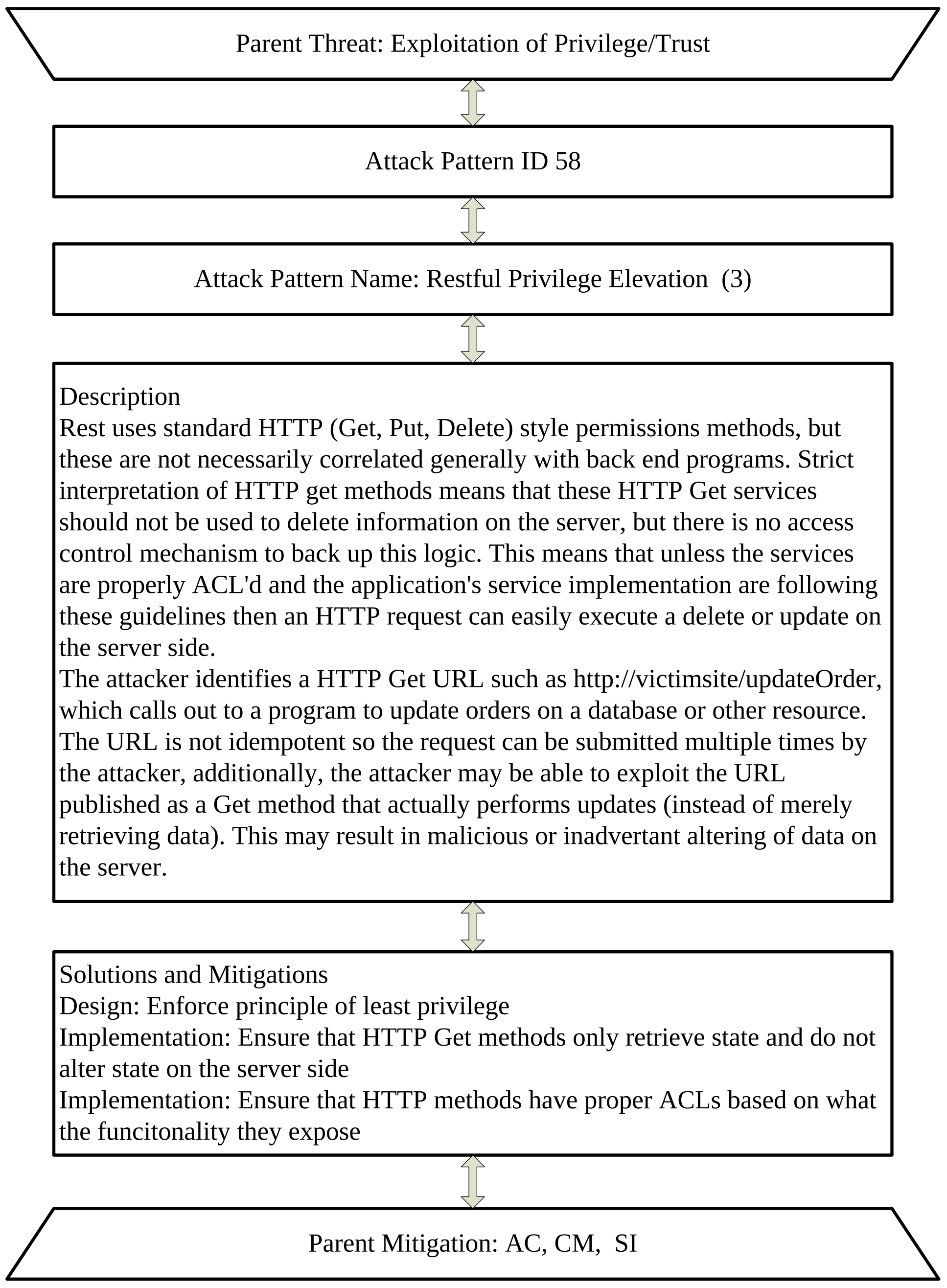
Parent Mitigation: SI, SC, IA


Parent Threat: Exploitation of Privilege/Trust
Attack Pattern ID 56
Attack Pattern Name: Removing/short-circuiting 'guard logic' (2)
Parent Mitigation: AC, IA

Description
Attackers can, in some cases, get around logic put in place to 'guard' sensitive functionality or data.
The attack may involve gaining access to and calling protected functionality (or accessing protected data) directly, may involve subverting some aspect of the guard's implementation, or outright removal of the guard, if possible.

Solutions and Mitigations
Use Authentication Mechanisms, Where Appropriate, Correctly
Use Authorization Mechanisms Correctly


Parent Threat: Spoofing
Attack Pattern ID: 57
Attack Pattern Name: Utilizing REST's Trust in the System Resource to Register Man in the Middle (3)
Description
This attack utlizes a Rest(REpresentational State Transfer)-style applications' trust in the system resources and environment to place man in the middle once SSL is terminated. Rest applications premise is that they leverage existing infrastructure to deliver web services functionality.
Solutions and Mitigations
Implementation: Implement message level security such as HMAC in the HTTP communication
Design: Utilize defense in depth, do not rely on a single security mechanism like SSL
Design: Enforce principle of least privilege
SA, SI, AC



Parent Threat: Exploitation of Privilege/Trust

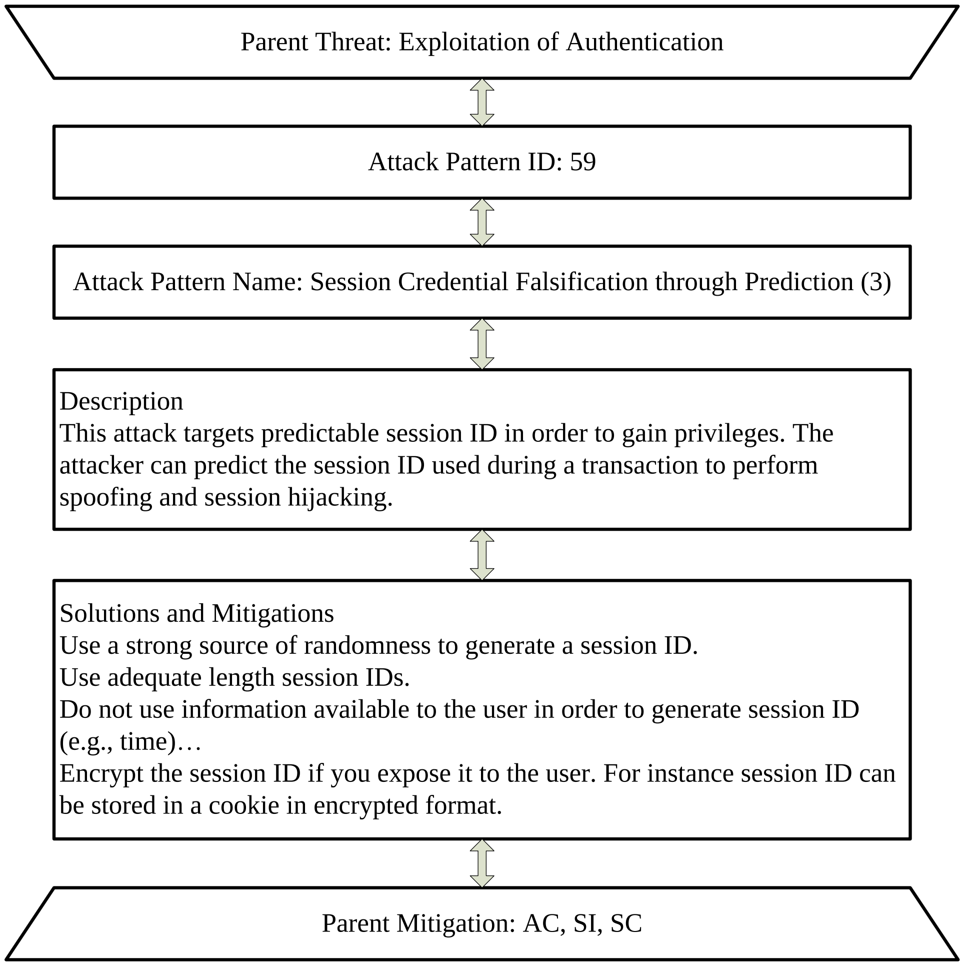
Attack Pattern ID 58

Attack Pattern Name: Restful Privilege Elevation (3)

Description
Rest uses standard HTTP (Get, Put, Delete) style permissions methods, but these are not necessarily correlated generally with back end programs. Strict interpretation of HTTP get methods means that these HTTP Get services should not be used to delete information on the server, but there is no access control mechanism to back up this logic. This means that unless the services are properly ACL'd and the application's service implementation are following these guidelines then an HTTP request can easily execute a delete or update on the server side.
The attacker identifies a HTTP Get URL such as http://victimsite/updateOrder, which calls out to a program to update orders on a database or other resource. The URL is not idempotent so the request can be submitted multiple times by the attacker, additionally, the attacker may be able to exploit the URL published as a Get method that actually performs updates (instead of merely retrieving data). This may result in malicious or inadvertant altering of data on the server.

Solutions and Mitigations
Design: Enforce principle of least privilege
Implementation: Ensure that HTTP Get methods only retrieve state and do not alter state on the server side
Implementation: Ensure that HTTP methods have proper ACLs based on what the funcitonality they expose

Parent Mitigation: AC, CM, SI



Parent Threat: Exploitation of Authentication

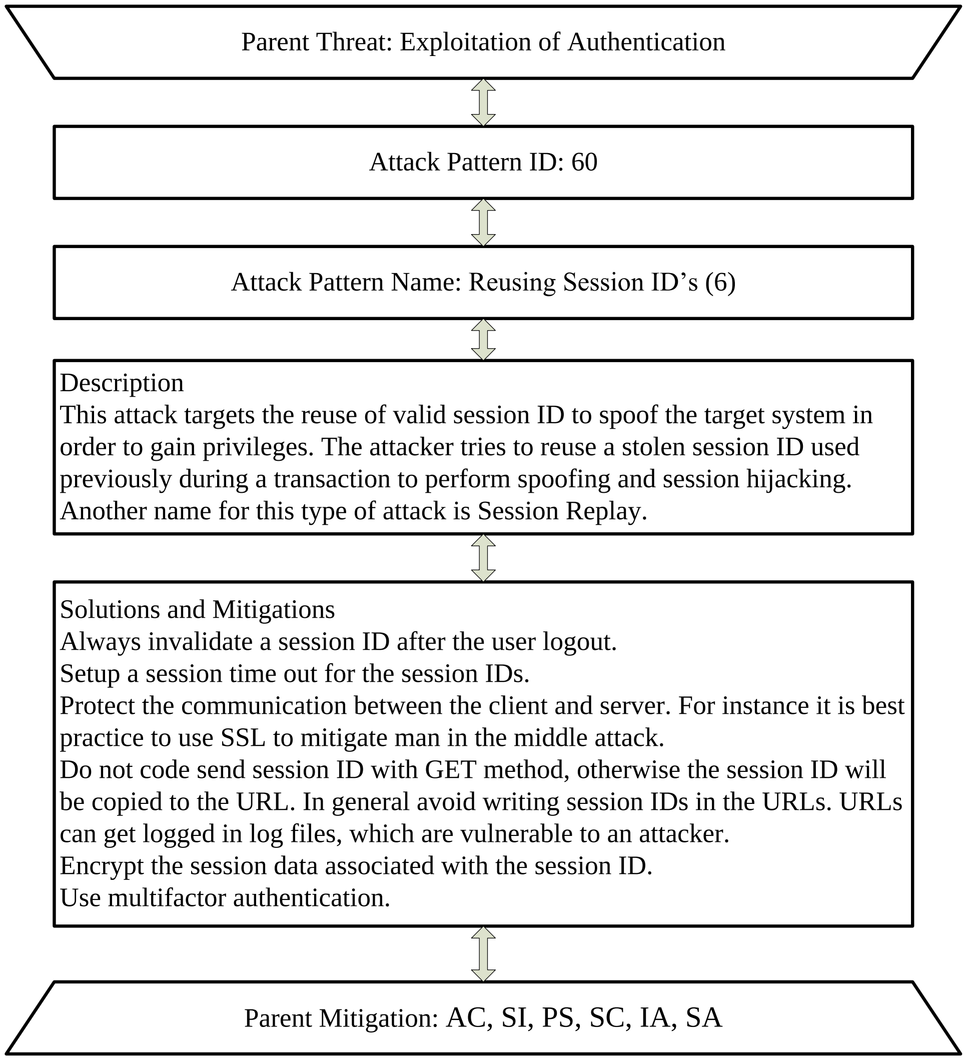
Attack Pattern ID: 59

Attack Pattern Name: Session Credential Falsification through Prediction (3)

Description
This attack targets predictable session ID in order to gain privileges. The attacker can predict the session ID used during a transaction to perform spoofing and session hijacking.

Solutions and Mitigations
Use a strong source of randomness to generate a session ID.
Use adequate length session IDs.
Do not use information available to the user in order to generate session ID (e.g., time)…
Encrypt the session ID if you expose it to the user. For instance session ID can be stored in a cookie in encrypted format.

Parent Mitigation: AC, SI, SC



Parent Threat: Exploitation of Authentication

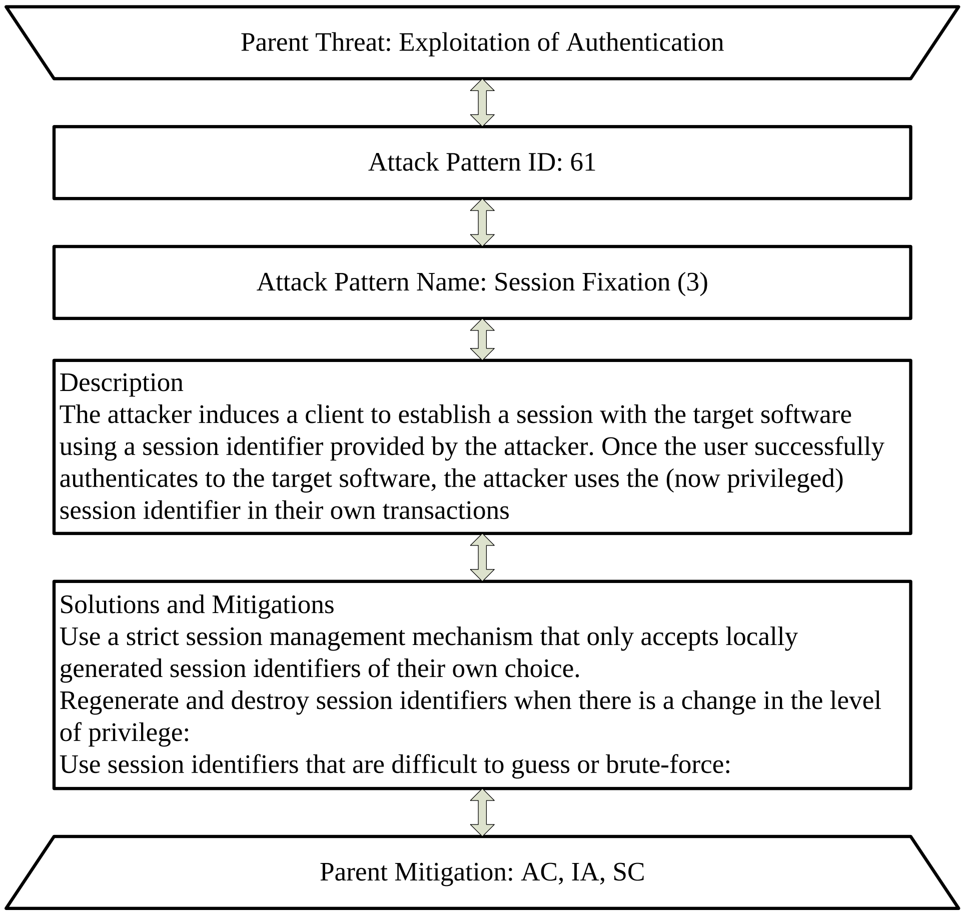
Attack Pattern ID: 60

Attack Pattern Name: Reusing Session ID's (6)

Description
This attack targets the reuse of valid session ID to spoof the target system in order to gain privileges. The attacker tries to reuse a stolen session ID used previously during a transaction to perform spoofing and session hijacking. Another name for this type of attack is Session Replay.

Solutions and Mitigations
Always invalidate a session ID after the user logout.
Setup a session time out for the session IDs.
Protect the communication between the client and server. For instance it is best practice to use SSL to mitigate man in the middle attack.
Do not code send session ID with GET method, otherwise the session ID will be copied to the URL. In general avoid writing session IDs in the URLs. URLs can get logged in log files, which are vulnerable to an attacker.
Encrypt the session data associated with the session ID.
Use multifactor authentication.

Parent Mitigation: AC, SI, PS, SC, IA, SA



Parent Threat: Exploitation of Authentication

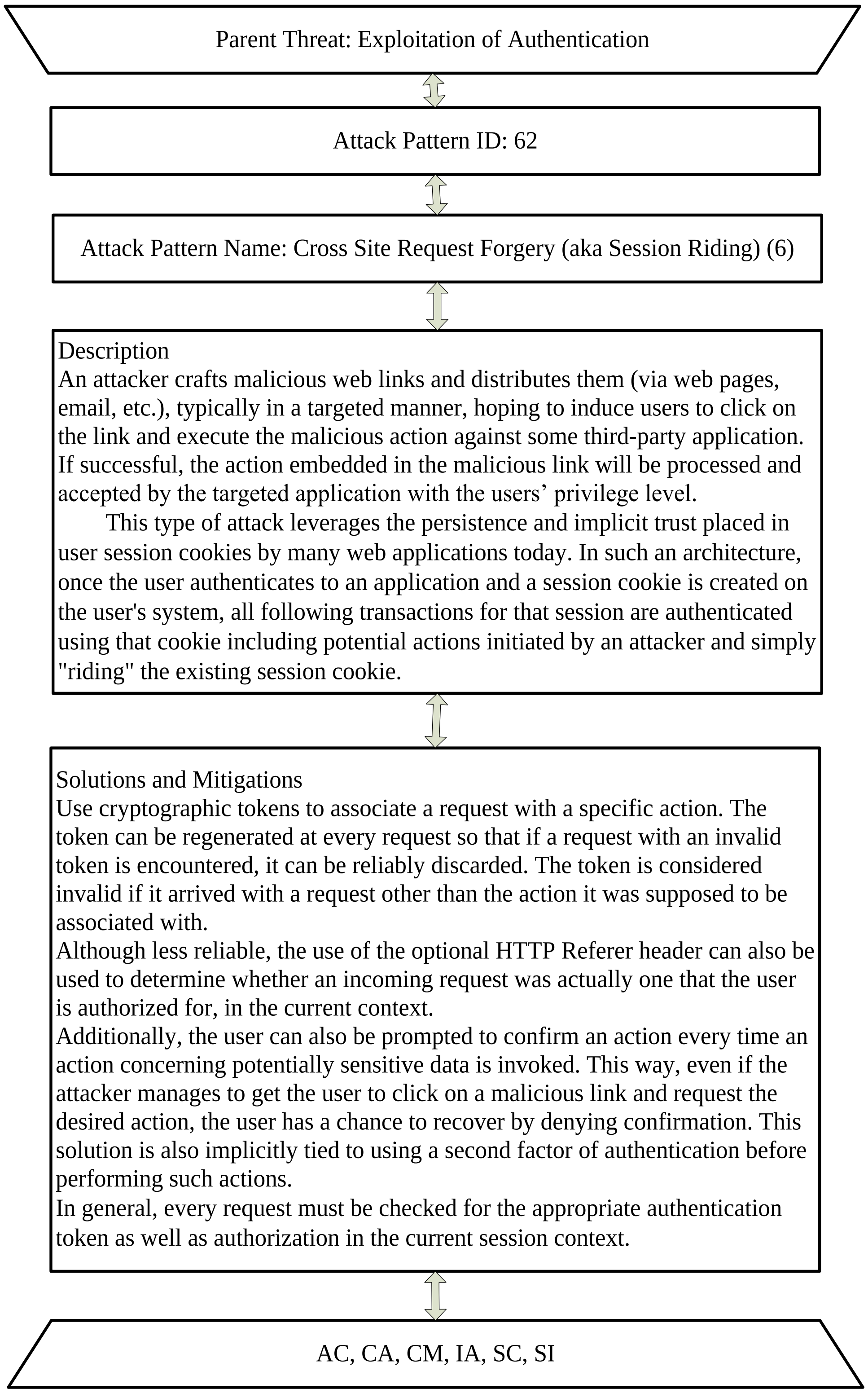
Attack Pattern ID: 61

Attack Pattern Name: Session Fixation (3)

Description
The attacker induces a client to establish a session with the target software using a session identifier provided by the attacker. Once the user successfully authenticates to the target software, the attacker uses the (now privileged) session identifier in their own transactions

Solutions and Mitigations
Use a strict session management mechanism that only accepts locally generated session identifiers of their own choice.
Regenerate and destroy session identifiers when there is a change in the level of privilege:
Use session identifiers that are difficult to guess or brute-force:

Parent Mitigation: AC, IA, SC



Parent Threat: Exploitation of Authentication

Attack Pattern ID: 62

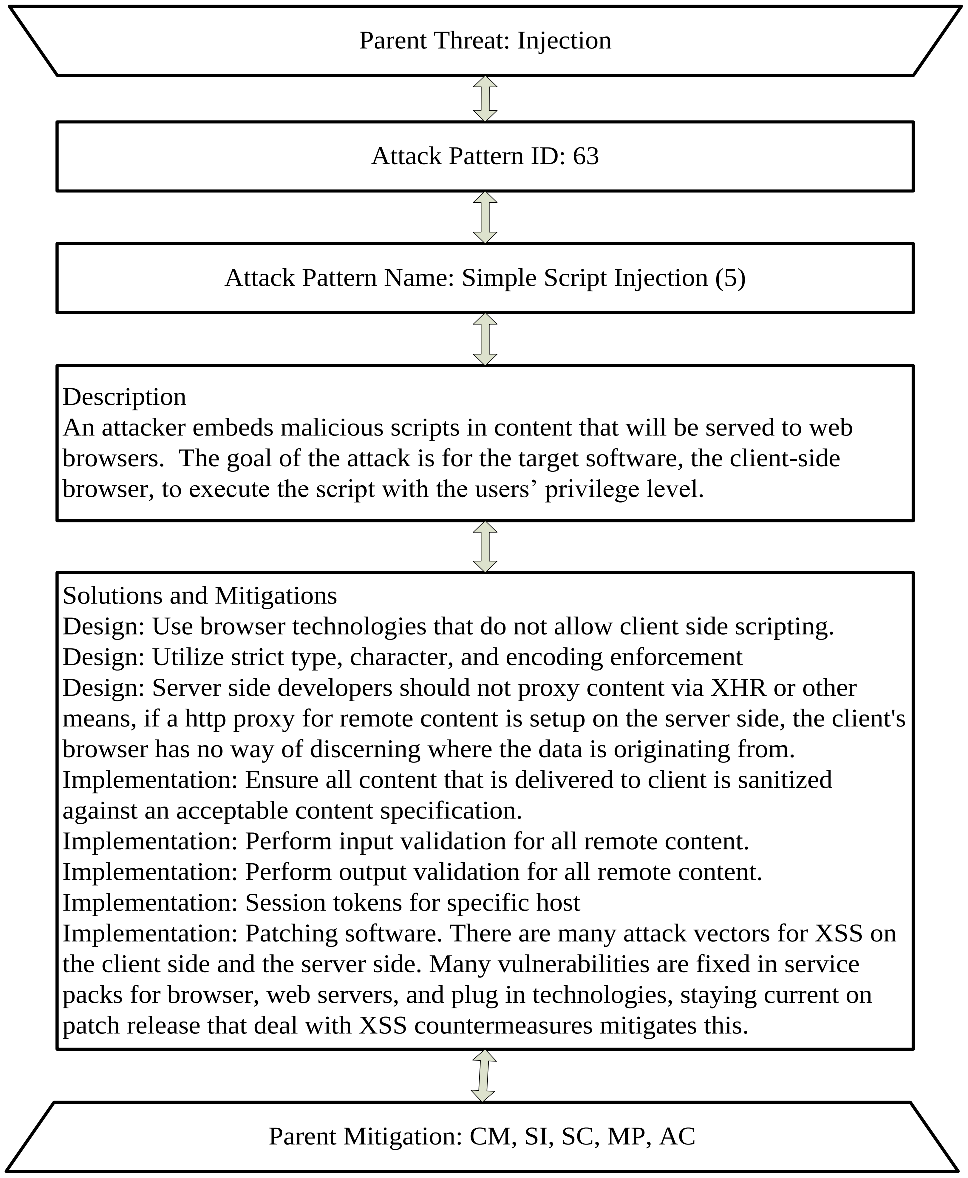
Attack Pattern Name: Cross Site Request Forgery (aka Session Riding) (6)

Description
An attacker crafts malicious web links and distributes them (via web pages, email, etc.), typically in a targeted manner, hoping to induce users to click on the link and execute the malicious action against some third-party application. If successful, the action embedded in the malicious link will be processed and accepted by the targeted application with the users' privilege level.

This type of attack leverages the persistence and implicit trust placed in user session cookies by many web applications today. In such an architecture, once the user authenticates to an application and a session cookie is created on the user's system, all following transactions for that session are authenticated using that cookie including potential actions initiated by an attacker and simply "riding" the existing session cookie.

Solutions and Mitigations
Use cryptographic tokens to associate a request with a specific action. The token can be regenerated at every request so that if a request with an invalid token is encountered, it can be reliably discarded. The token is considered invalid if it arrived with a request other than the action it was supposed to be associated with.
Although less reliable, the use of the optional HTTP Referer header can also be used to determine whether an incoming request was actually one that the user is authorized for, in the current context.
Additionally, the user can also be prompted to confirm an action every time an action concerning potentially sensitive data is invoked. This way, even if the attacker manages to get the user to click on a malicious link and request the desired action, the user has a chance to recover by denying confirmation. This solution is also implicitly tied to using a second factor of authentication before performing such actions.
In general, every request must be checked for the appropriate authentication token as well as authorization in the current session context.

AC, CA, CM, IA, SC, SI



Parent Threat: Injection

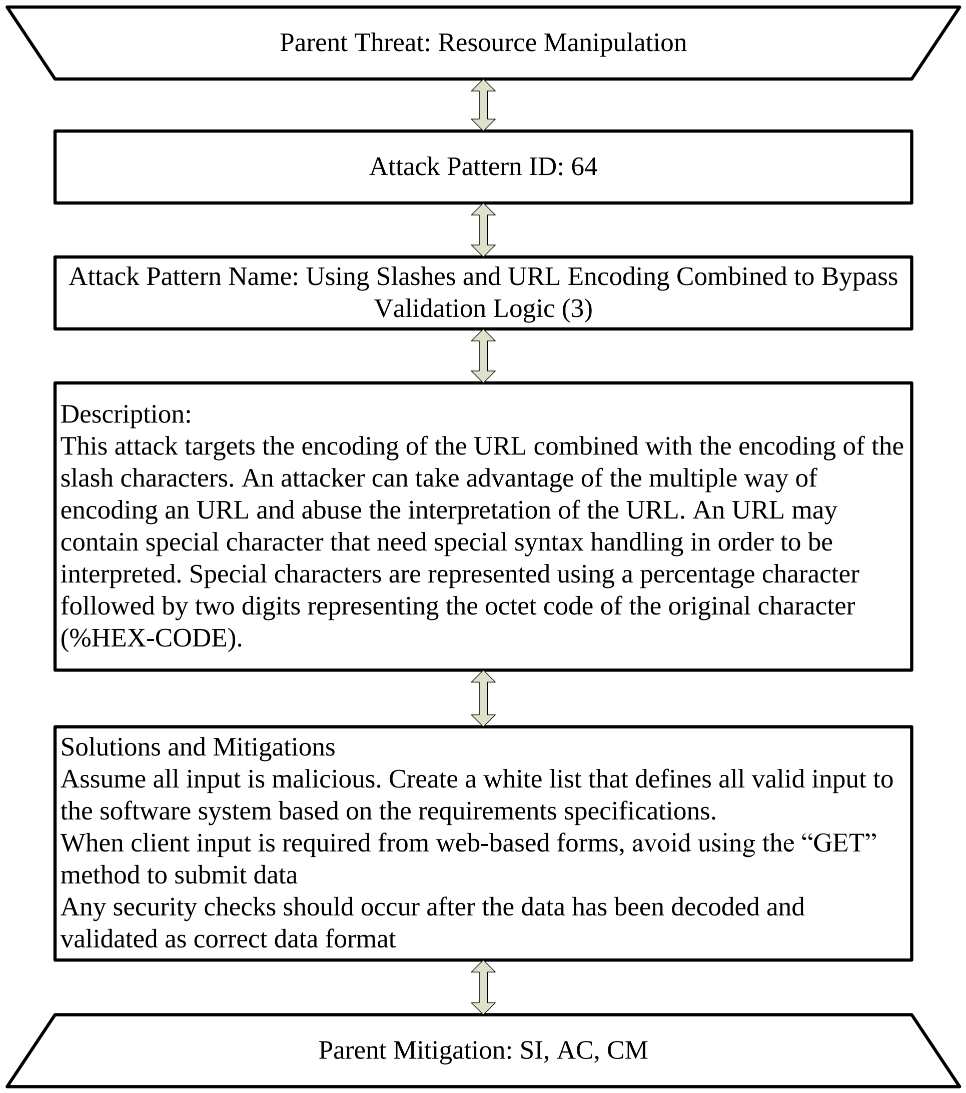
Attack Pattern ID: 63

Attack Pattern Name: Simple Script Injection (5)

Description
An attacker embeds malicious scripts in content that will be served to web browsers. The goal of the attack is for the target software, the client-side browser, to execute the script with the users’ privilege level.

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting.
Design: Utilize strict type, character, and encoding enforcement
Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from.
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Session tokens for specific host
Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.

Parent Mitigation: CM, SI, SC, MP, AC



Parent Threat: Resource Manipulation

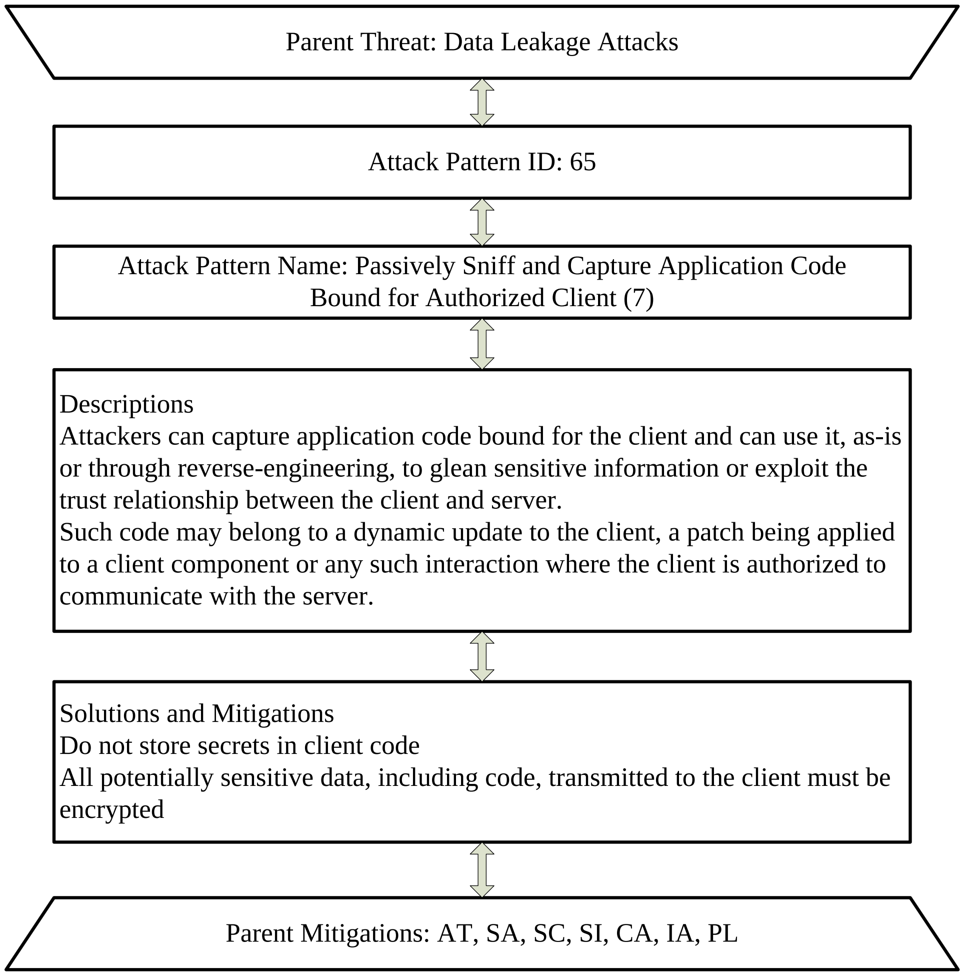
Attack Pattern ID: 64

Attack Pattern Name: Using Slashes and URL Encoding Combined to Bypass Validation Logic (3)

Description:
This attack targets the encoding of the URL combined with the encoding of the slash characters. An attacker can take advantage of the multiple way of encoding an URL and abuse the interpretation of the URL. An URL may contain special character that need special syntax handling in order to be interpreted. Special characters are represented using a percentage character followed by two digits representing the octet code of the original character (%HEX-CODE).

Solutions and Mitigations
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications.
When client input is required from web-based forms, avoid using the “GET” method to submit data
Any security checks should occur after the data has been decoded and validated as correct data format

Parent Mitigation: SI, AC, CM



Parent Threat: Data Leakage Attacks

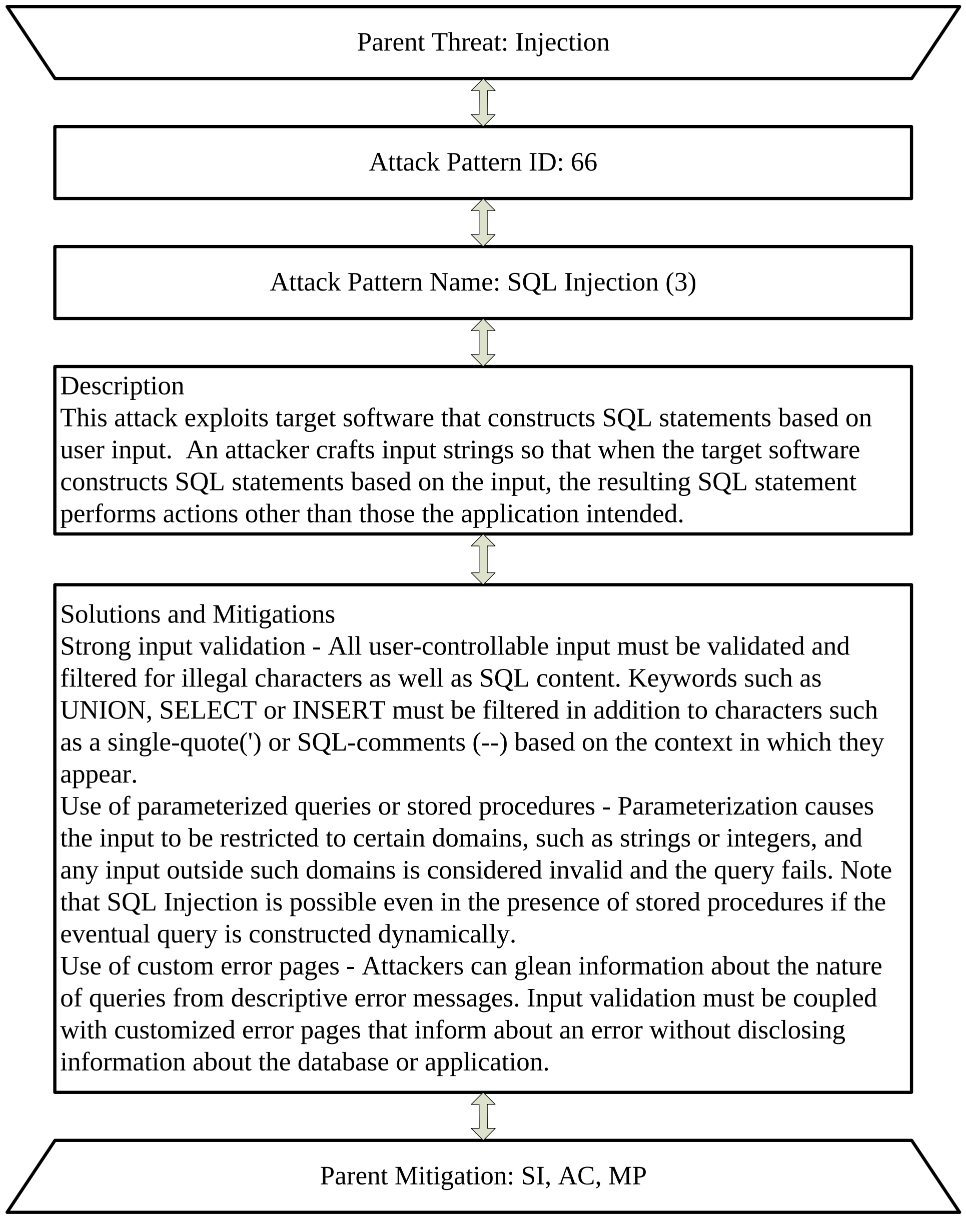
Attack Pattern ID: 65

Attack Pattern Name: Passively Sniff and Capture Application Code Bound for Authorized Client (7)

Descriptions
Attackers can capture application code bound for the client and can use it, as-is or through reverse-engineering, to glean sensitive information or exploit the trust relationship between the client and server.
Such code may belong to a dynamic update to the client, a patch being applied to a client component or any such interaction where the client is authorized to communicate with the server.

Solutions and Mitigations
Do not store secrets in client code
All potentially sensitive data, including code, transmitted to the client must be encrypted

Parent Mitigations: AT, SA, SC, SI, CA, IA, PL



Parent Threat: Injection

Attack Pattern ID: 66

Attack Pattern Name: SQL Injection (3)

Description
This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended.

Solutions and Mitigations
Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear.
Use of parameterized queries or stored procedures - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails. Note that SQL Injection is possible even in the presence of stored procedures if the eventual query is constructed dynamically.
Use of custom error pages - Attackers can glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.

Parent Mitigation: SI, AC, MP

Parent Threat: Data Structure Attacks

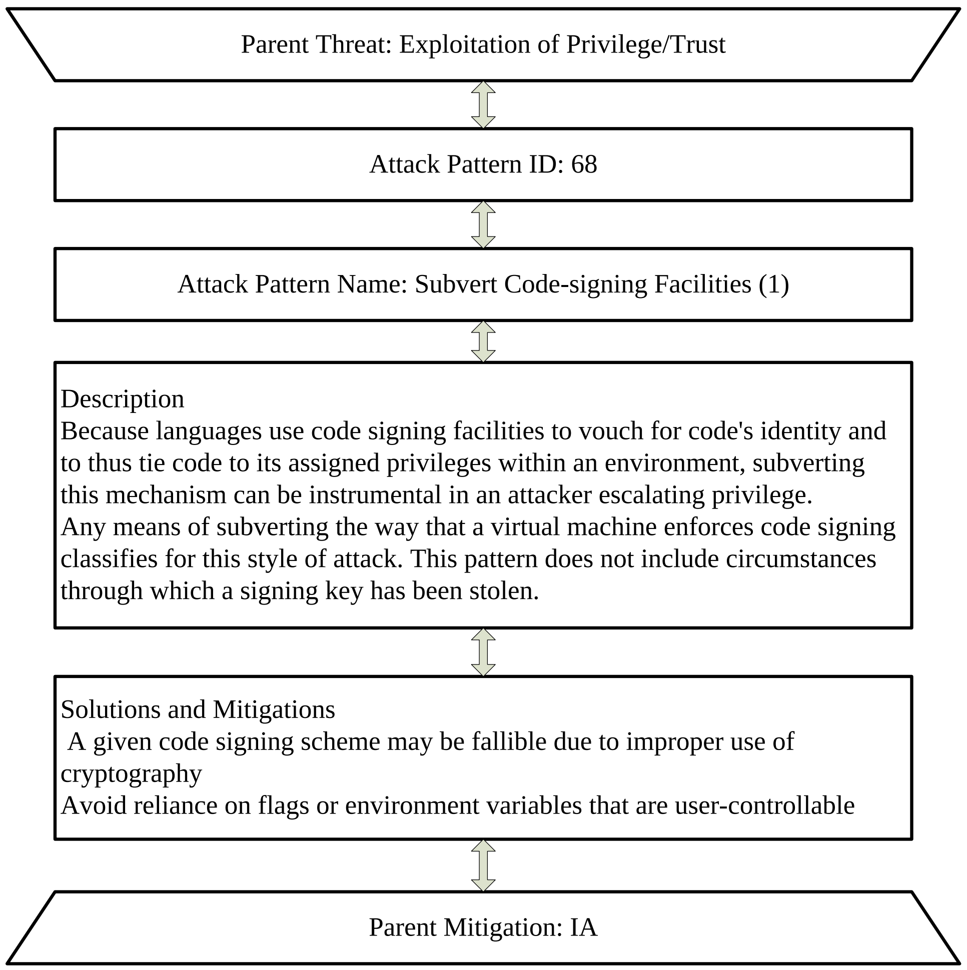
Attack Pattern ID: 67

Attack Pattern Name: String Format Overflow in syslog() (2)

Description
This attack targets the format string vulnerabilities in the syslog() function. An attacker would typically inject malicious input in the format string parameter of the syslog function. This is a common problem, and many public vulnerabilities and associated exploits have been posted.

Solutions and Mitigations
The code should be reviewed for misuse of the Syslog function call. Manual or automated code review can be used. The reviewer needs to ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, do not use the %n operator in format strings. The following code shows a correct usage of Syslog(): ... syslog(LOG_ERR, "%s", cmdBuf); ... The following code shows a vulnerable usage of Syslog(): ... syslog(LOG_ERR, cmdBuf); // the buffer cmdBuff is taking user supplied data. ...

Parent Mitigation: SI, AC



Parent Threat: Exploitation of Privilege/Trust

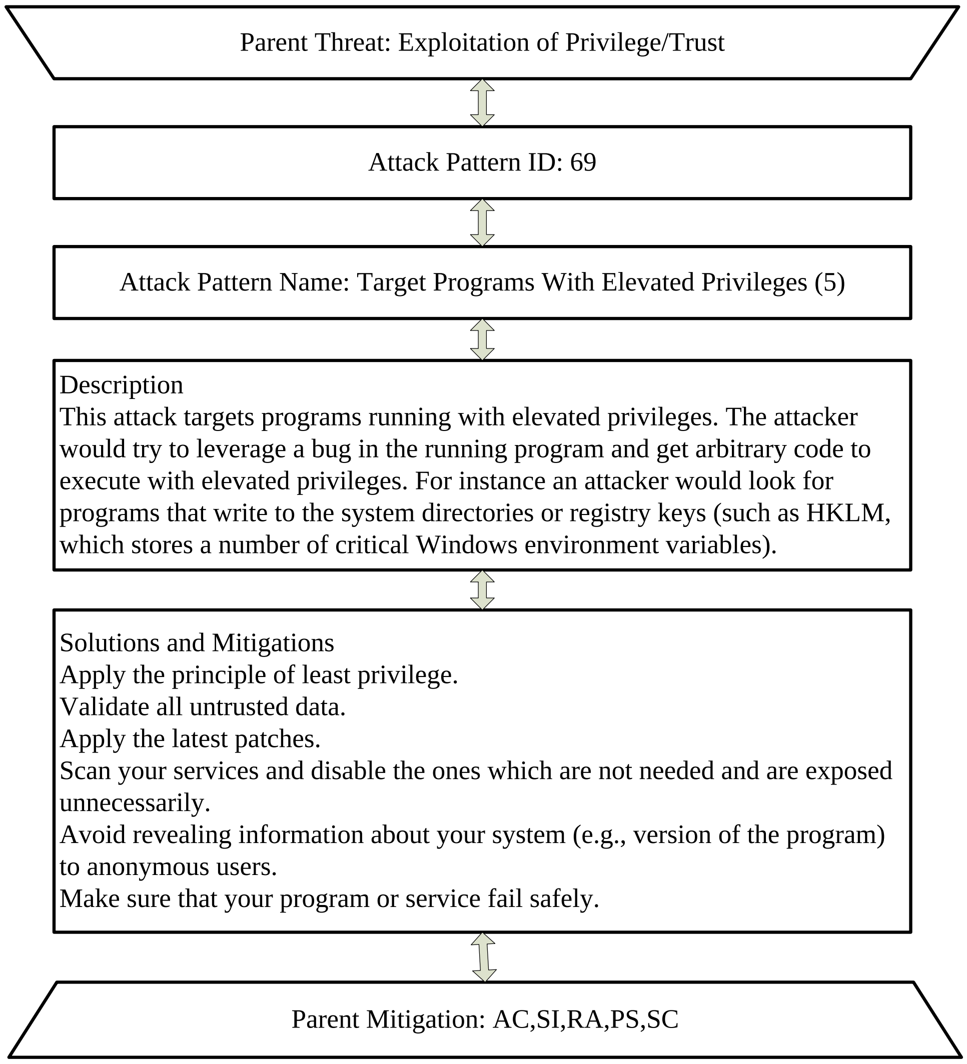
Attack Pattern ID: 68

Attack Pattern Name: Subvert Code-signing Facilities (1)

Description
Because languages use code signing facilities to vouch for code's identity and to thus tie code to its assigned privileges within an environment, subverting this mechanism can be instrumental in an attacker escalating privilege.
Any means of subverting the way that a virtual machine enforces code signing classifies for this style of attack. This pattern does not include circumstances through which a signing key has been stolen.

Solutions and Mitigations
A given code signing scheme may be fallible due to improper use of cryptography
Avoid reliance on flags or environment variables that are user-controllable

Parent Mitigation: IA



Parent Threat: Exploitation of Privilege/Trust

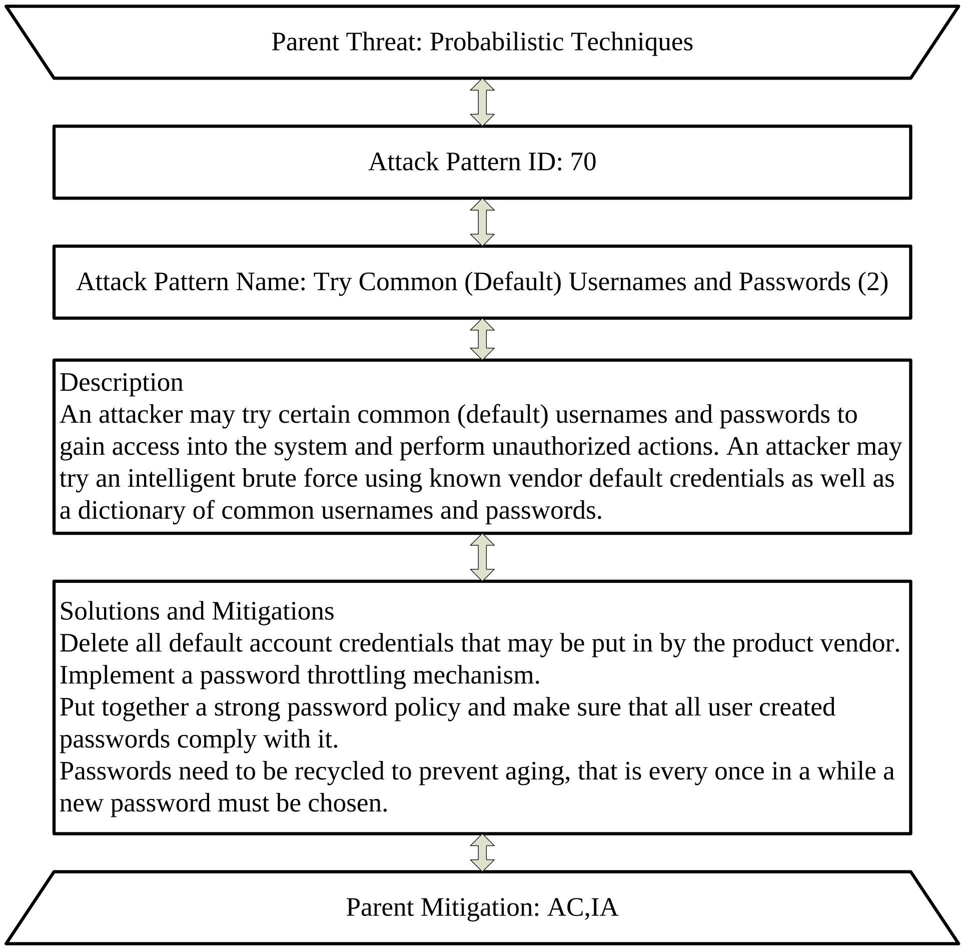
Attack Pattern ID: 69

Attack Pattern Name: Target Programs With Elevated Privileges (5)

Description
This attack targets programs running with elevated privileges. The attacker would try to leverage a bug in the running program and get arbitrary code to execute with elevated privileges. For instance an attacker would look for programs that write to the system directories or registry keys (such as HKLM, which stores a number of critical Windows environment variables).

Solutions and Mitigations
Apply the principle of least privilege.
Validate all untrusted data.
Apply the latest patches.
Scan your services and disable the ones which are not needed and are exposed unnecessarily.
Avoid revealing information about your system (e.g., version of the program) to anonymous users.
Make sure that your program or service fail safely.

Parent Mitigation: AC,SI,RA,PS,SC



Parent Threat: Probabilistic Techniques

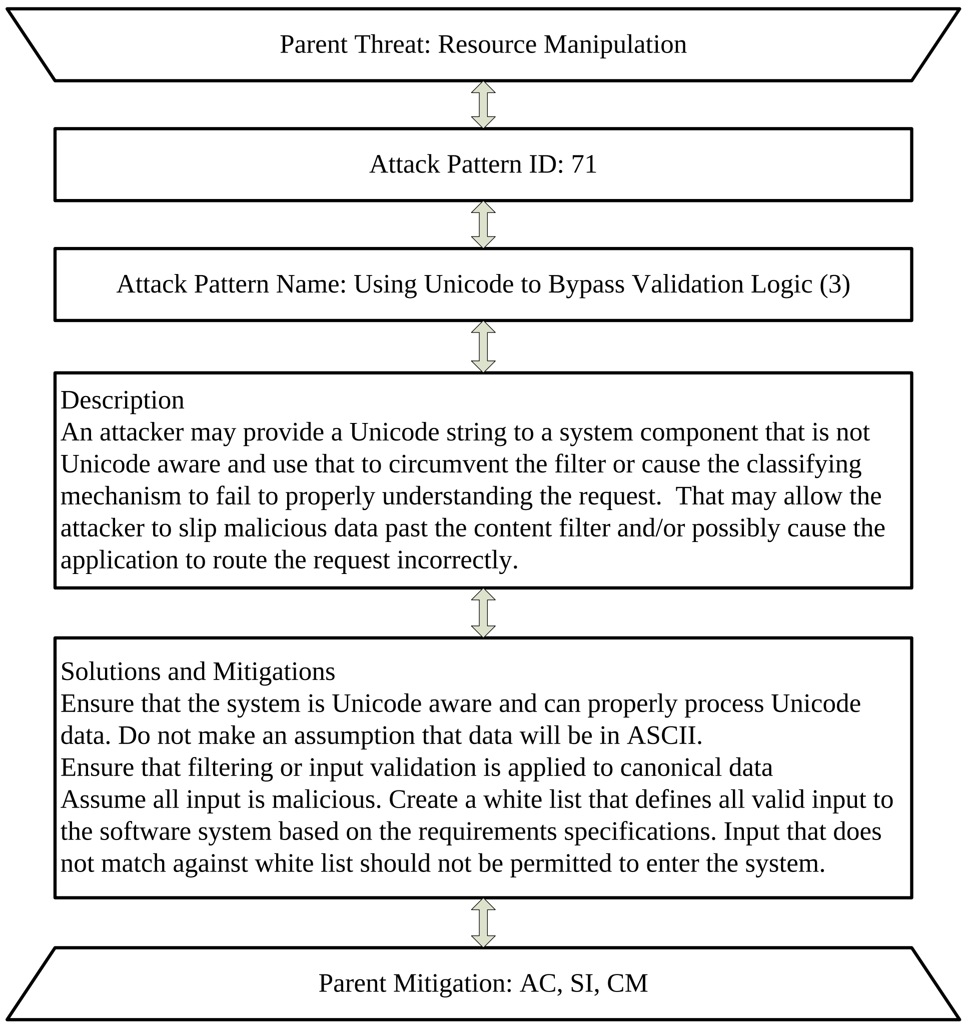
Attack Pattern ID: 70

Attack Pattern Name: Try Common (Default) Usernames and Passwords (2)

Description
An attacker may try certain common (default) usernames and passwords to gain access into the system and perform unauthorized actions. An attacker may try an intelligent brute force using known vendor default credentials as well as a dictionary of common usernames and passwords.

Solutions and Mitigations
Delete all default account credentials that may be put in by the product vendor.
Implement a password throttling mechanism.
Put together a strong password policy and make sure that all user created passwords comply with it.
Passwords need to be recycled to prevent aging, that is every once in a while a new password must be chosen.

Parent Mitigation: AC,IA



Parent Threat: Resource Manipulation

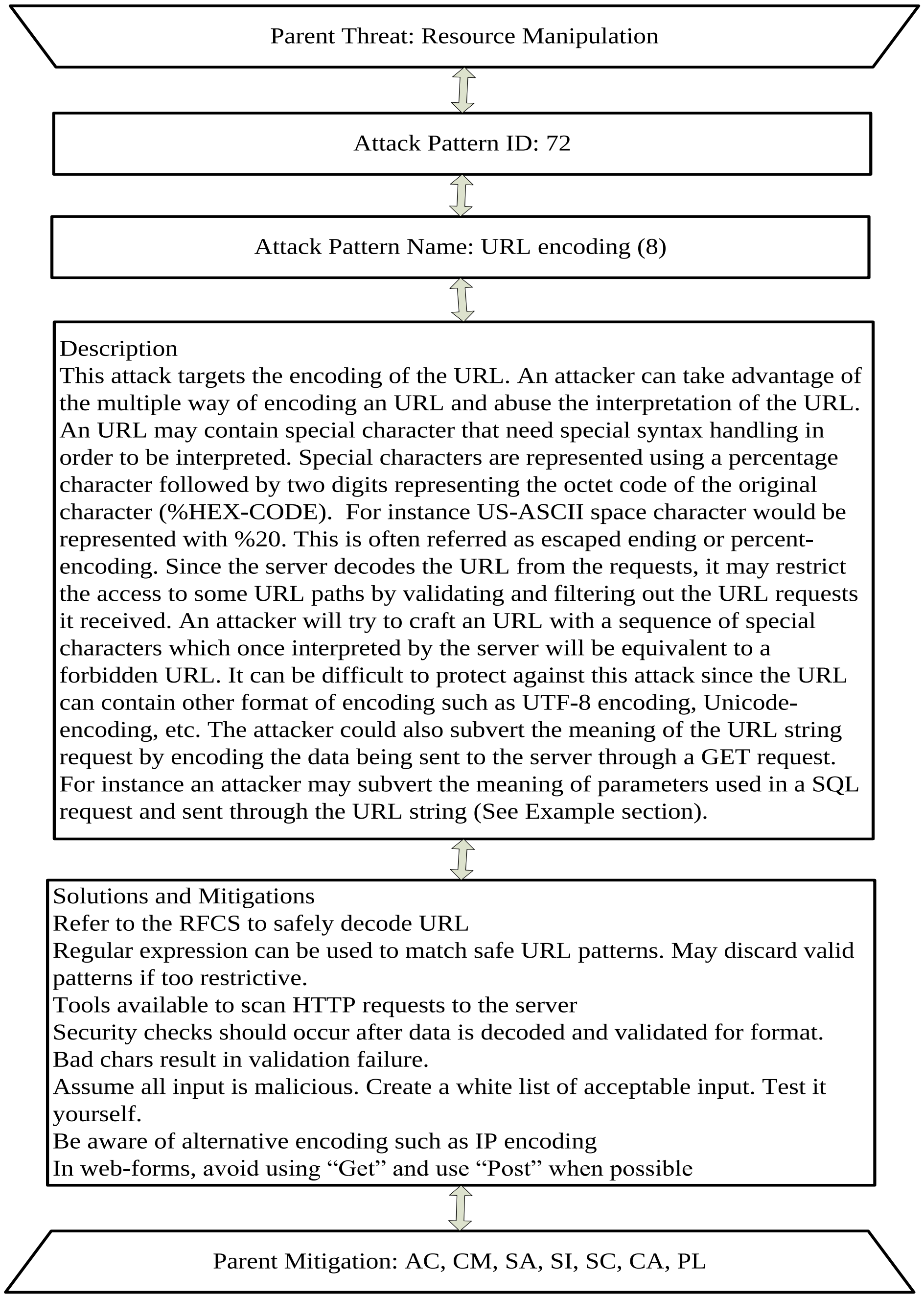
Attack Pattern ID: 71

Attack Pattern Name: Using Unicode to Bypass Validation Logic (3)

Description
An attacker may provide a Unicode string to a system component that is not Unicode aware and use that to circumvent the filter or cause the classifying mechanism to fail to properly understanding the request. That may allow the attacker to slip malicious data past the content filter and/or possibly cause the application to route the request incorrectly.

Solutions and Mitigations
Ensure that the system is Unicode aware and can properly process Unicode data. Do not make an assumption that data will be in ASCII.
Ensure that filtering or input validation is applied to canonical data
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against white list should not be permitted to enter the system.

Parent Mitigation: AC, SI, CM



Parent Threat: Resource Manipulation

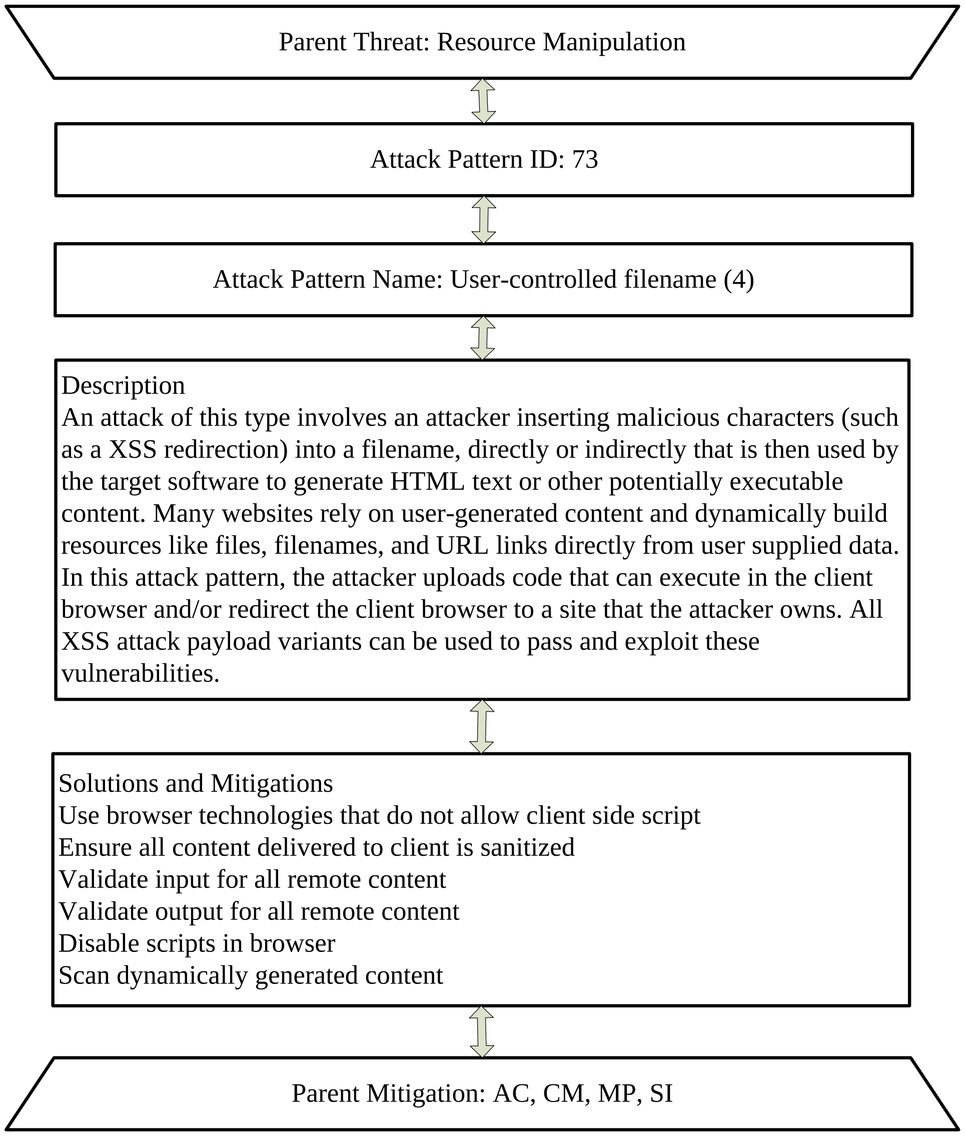
Attack Pattern ID: 72

Attack Pattern Name: URL encoding (8)

Description
This attack targets the encoding of the URL. An attacker can take advantage of the multiple way of encoding an URL and abuse the interpretation of the URL. An URL may contain special character that need special syntax handling in order to be interpreted. Special characters are represented using a percentage character followed by two digits representing the octet code of the original character (%HEX-CODE). For instance US-ASCII space character would be represented with %20. This is often referred as escaped ending or percent-encoding. Since the server decodes the URL from the requests, it may restrict the access to some URL paths by validating and filtering out the URL requests it received. An attacker will try to craft an URL with a sequence of special characters which once interpreted by the server will be equivalent to a forbidden URL. It can be difficult to protect against this attack since the URL can contain other format of encoding such as UTF-8 encoding, Unicode-encoding, etc. The attacker could also subvert the meaning of the URL string request by encoding the data being sent to the server through a GET request. For instance an attacker may subvert the meaning of parameters used in a SQL request and sent through the URL string (See Example section).

Solutions and Mitigations
Refer to the RFCS to safely decode URL
Regular expression can be used to match safe URL patterns. May discard valid patterns if too restrictive.
Tools available to scan HTTP requests to the server
Security checks should occur after data is decoded and validated for format. Bad chars result in validation failure.
Assume all input is malicious. Create a white list of acceptable input. Test it yourself.
Be aware of alternative encoding such as IP encoding
In web-forms, avoid using “Get” and use “Post” when possible

Parent Mitigation: AC, CM, SA, SI, SC, CA, PL



Parent Threat: Resource Manipulation

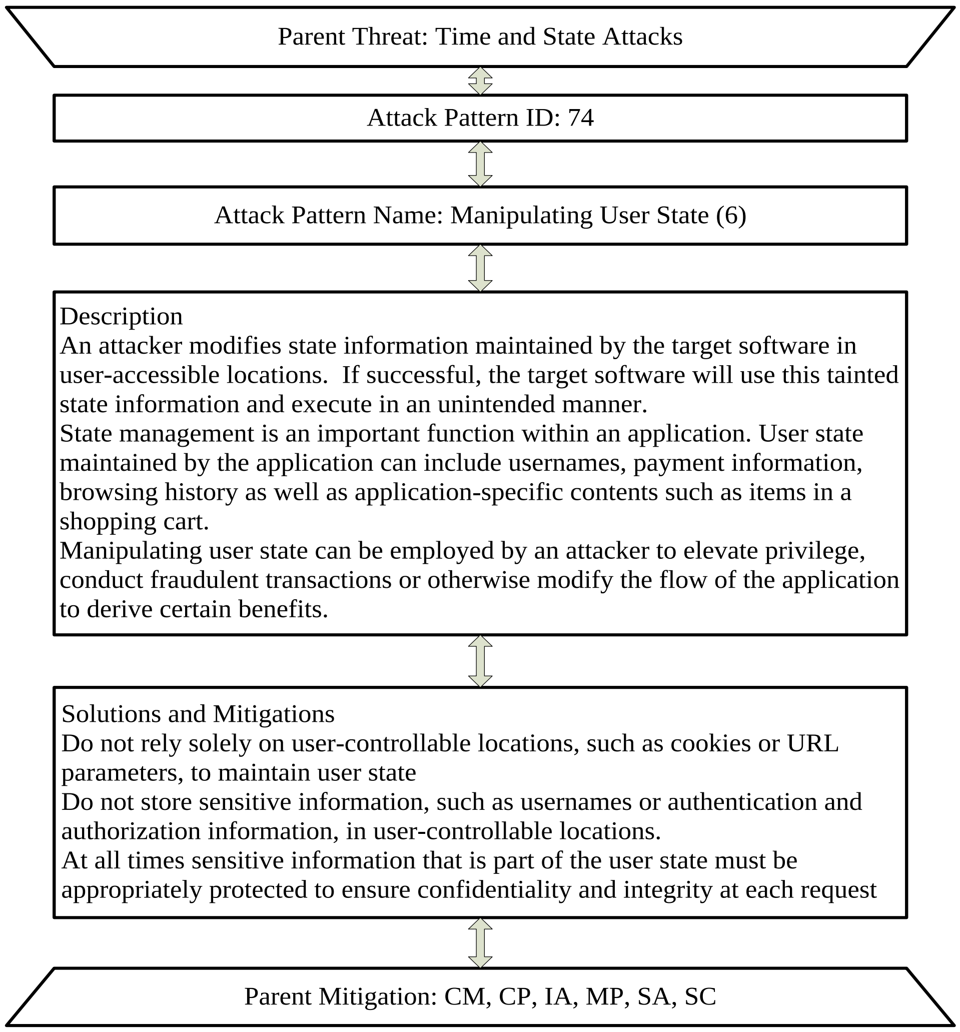
Attack Pattern ID: 73

Attack Pattern Name: User-controlled filename (4)

Description
An attack of this type involves an attacker inserting malicious characters (such as a XSS redirection) into a filename, directly or indirectly that is then used by the target software to generate HTML text or other potentially executable content. Many websites rely on user-generated content and dynamically build resources like files, filenames, and URL links directly from user supplied data. In this attack pattern, the attacker uploads code that can execute in the client browser and/or redirect the client browser to a site that the attacker owns. All XSS attack payload variants can be used to pass and exploit these vulnerabilities.

Solutions and Mitigations
Use browser technologies that do not allow client side script
Ensure all content delivered to client is sanitized
Validate input for all remote content
Validate output for all remote content
Disable scripts in browser
Scan dynamically generated content

Parent Mitigation: AC, CM, MP, SI



Parent Threat: Time and State Attacks

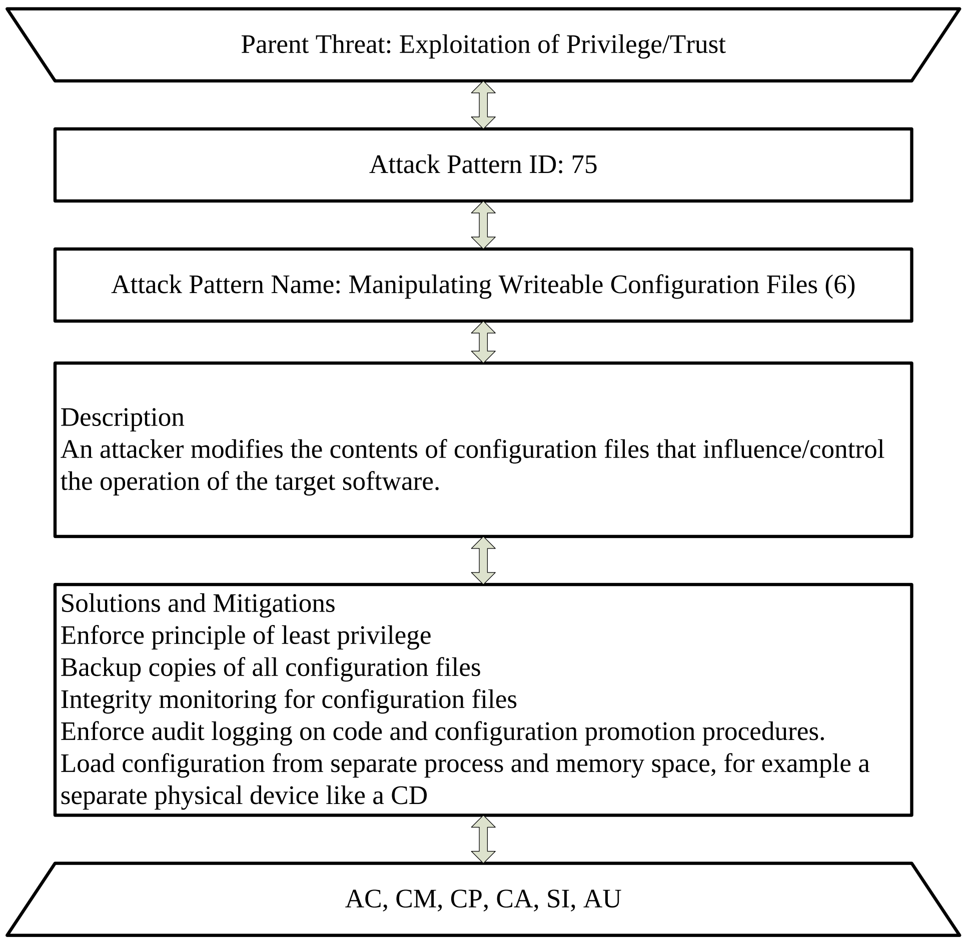
Attack Pattern ID: 74

Attack Pattern Name: Manipulating User State (6)

Description
An attacker modifies state information maintained by the target software in user-accessible locations. If successful, the target software will use this tainted state information and execute in an unintended manner.
State management is an important function within an application. User state maintained by the application can include usernames, payment information, browsing history as well as application-specific contents such as items in a shopping cart.
Manipulating user state can be employed by an attacker to elevate privilege, conduct fraudulent transactions or otherwise modify the flow of the application to derive certain benefits.

Solutions and Mitigations
Do not rely solely on user-controllable locations, such as cookies or URL parameters, to maintain user state
Do not store sensitive information, such as usernames or authentication and authorization information, in user-controllable locations.
At all times sensitive information that is part of the user state must be appropriately protected to ensure confidentiality and integrity at each request

Parent Mitigation: CM, CP, IA, MP, SA, SC



Parent Threat: Exploitation of Privilege/Trust

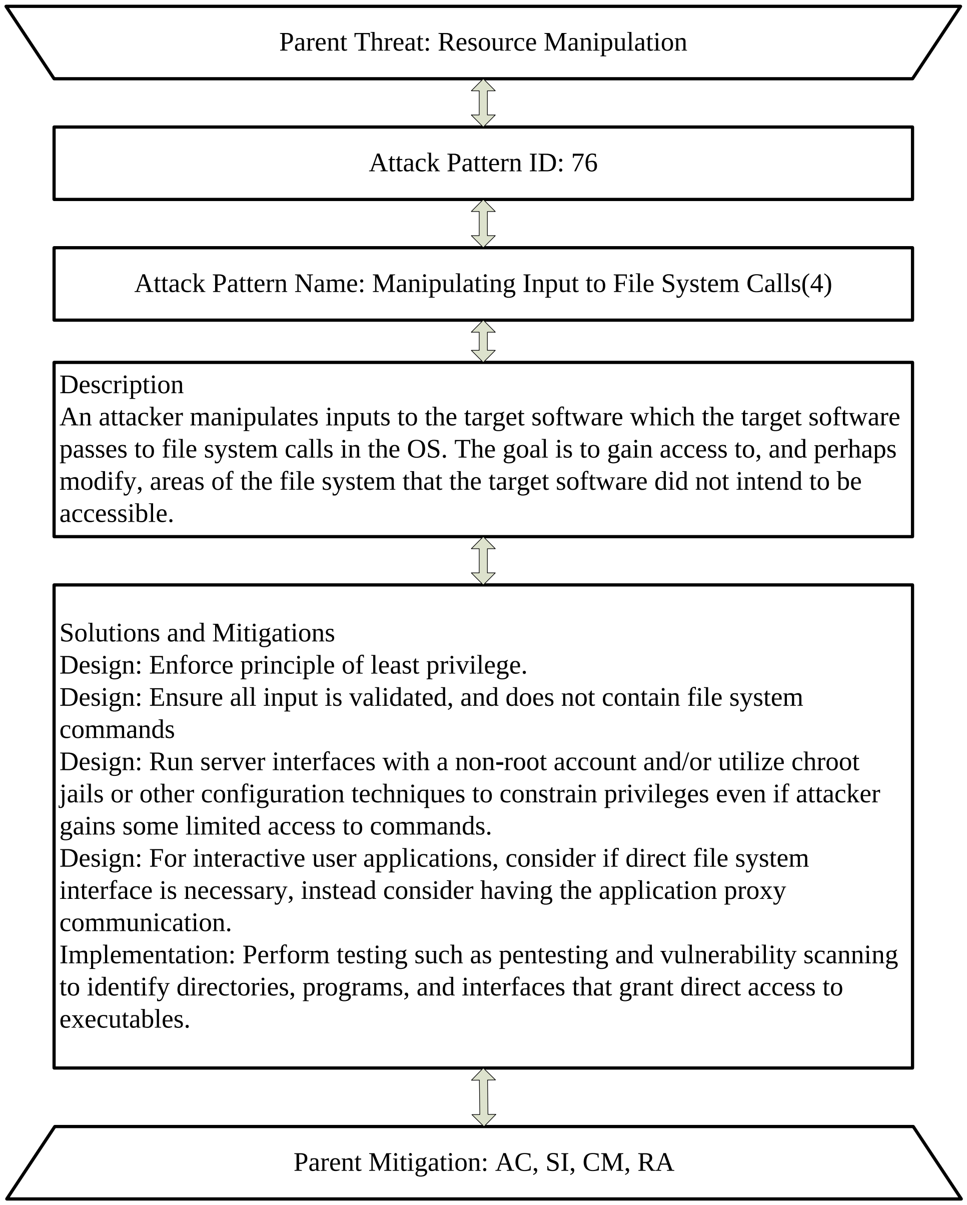
Attack Pattern ID: 75

Attack Pattern Name: Manipulating Writeable Configuration Files (6)

Description
An attacker modifies the contents of configuration files that influence/control the operation of the target software.

Solutions and Mitigations
Enforce principle of least privilege
Backup copies of all configuration files
Integrity monitoring for configuration files
Enforce audit logging on code and configuration promotion procedures.
Load configuration from separate process and memory space, for example a separate physical device like a CD

AC, CM, CP, CA, SI, AU



Parent Threat: Resource Manipulation

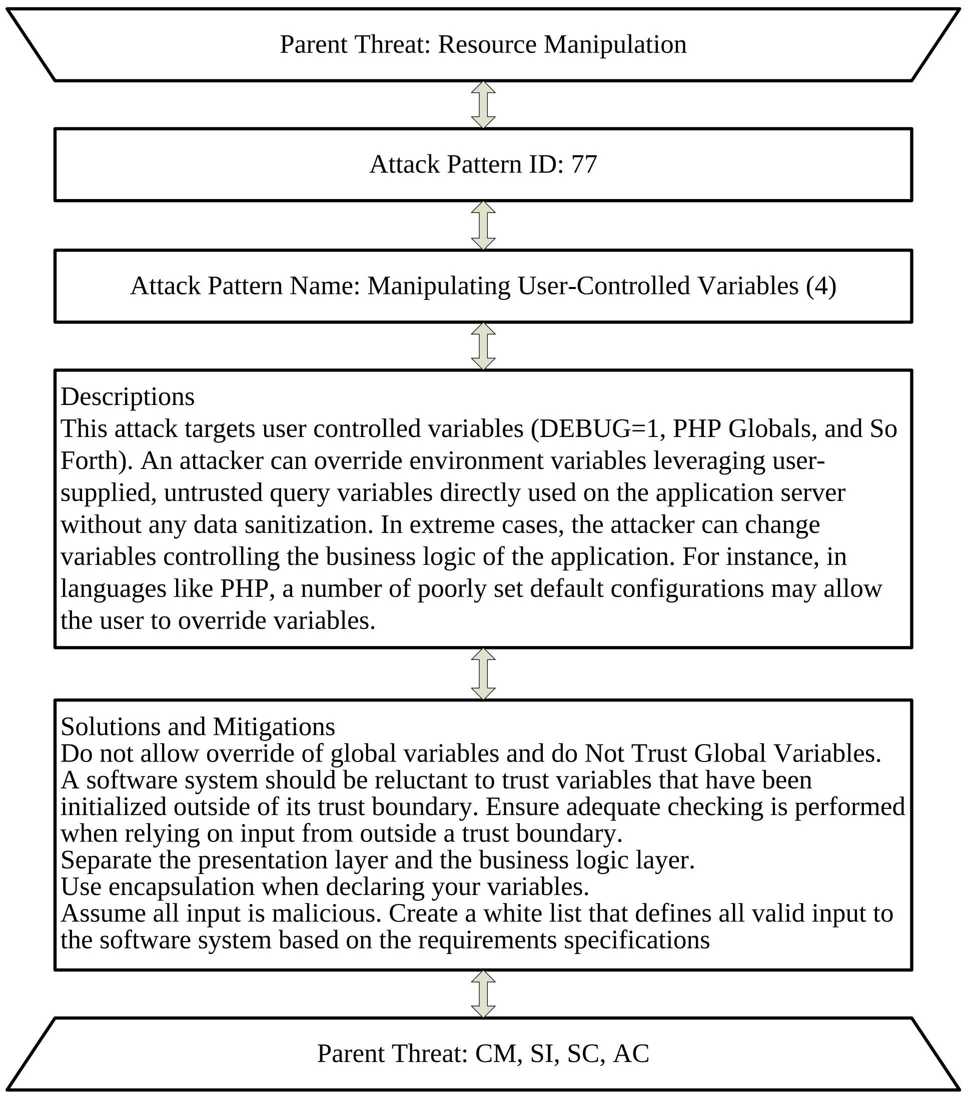
Attack Pattern ID: 76

Attack Pattern Name: Manipulating Input to File System Calls(4)

Description
An attacker manipulates inputs to the target software which the target software passes to file system calls in the OS. The goal is to gain access to, and perhaps modify, areas of the file system that the target software did not intend to be accessible.

Solutions and Mitigations
Design: Enforce principle of least privilege.
Design: Ensure all input is validated, and does not contain file system commands
Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands.
Design: For interactive user applications, consider if direct file system interface is necessary, instead consider having the application proxy communication.
Implementation: Perform testing such as pentesting and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.

Parent Mitigation: AC, SI, CM, RA



Parent Threat: Resource Manipulation

Attack Pattern ID: 77

Attack Pattern Name: Manipulating User-Controlled Variables (4)

Descriptions
This attack targets user controlled variables (DEBUG=1, PHP Globals, and So Forth). An attacker can override environment variables leveraging user-supplied, untrusted query variables directly used on the application server without any data sanitization. In extreme cases, the attacker can change variables controlling the business logic of the application. For instance, in languages like PHP, a number of poorly set default configurations may allow the user to override variables.

Solutions and Mitigations
Do not allow override of global variables and do Not Trust Global Variables.
A software system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary.
Separate the presentation layer and the business logic layer.
Use encapsulation when declaring your variables.
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications

Parent Threat: CM, SI, SC, AC

Parent Threat: Resource Manipulation

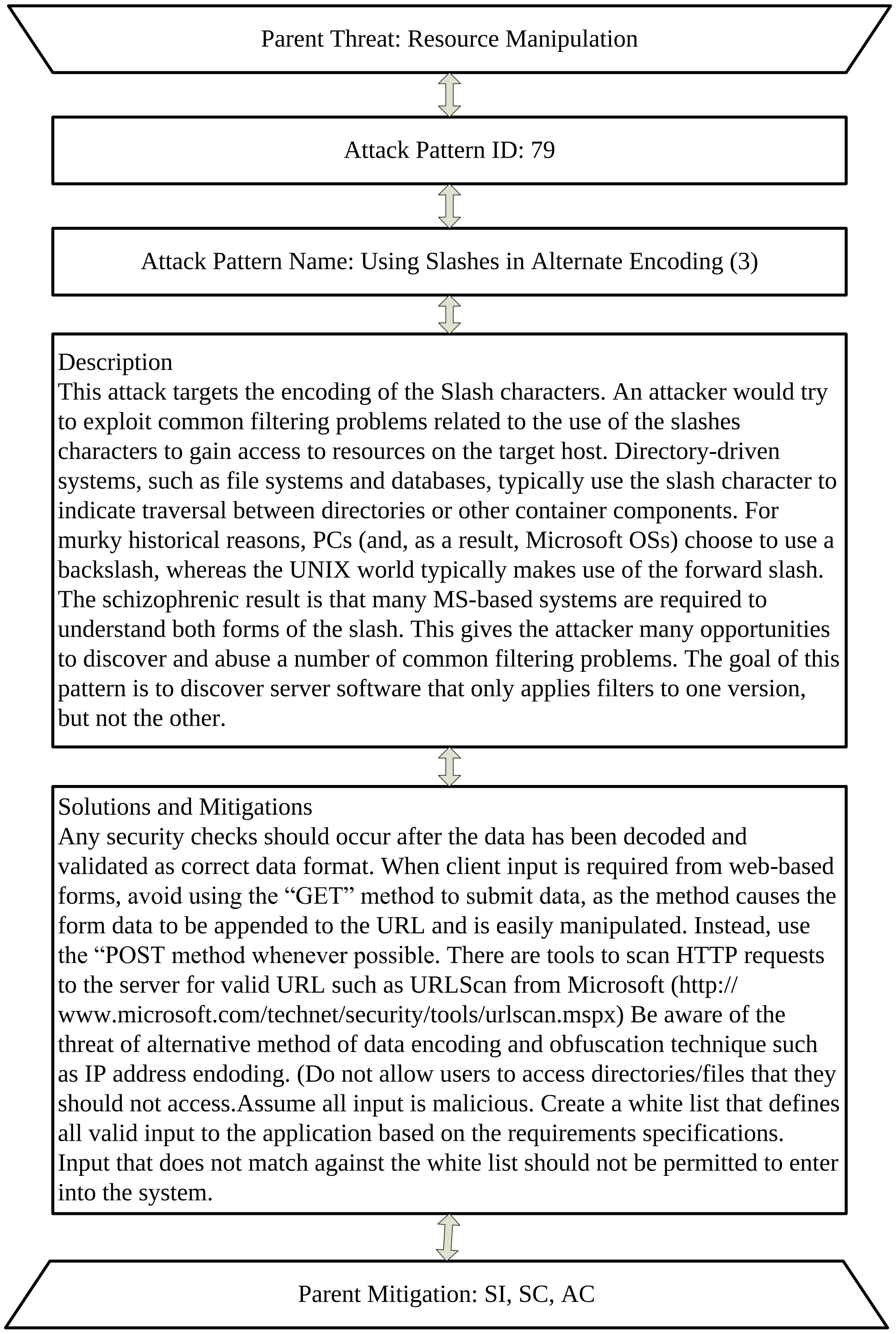
Attack Pattern ID: 78

Attack Pattern Name: Using Escaped Slashes in Alternate Encoding (5)

Description
This attack targets the use of the backslash in alternate encoding. An attacker can provide a backslash as a leading character and causes a parser to believe that the next character is special. This is called an escape. By using that trick, the attacker tries to exploit alternate ways to encode the same character which leads to filter problems and opens avenues to attack.

Solutions and Mitigations
Verify that the user-supplied data does not use backslash character to escape malicious characters.
Assume all input is malicious. Create a white list that defines all valid input to the software system based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.
Be aware of the threat of alternative method of data encoding.
Regular expressions can be used to filter out backslash. Make sure you decode before filtering and validating the untrusted input data.
In the case of path traversals, use the principle of least privilege when determining access rights to file systems. Do not allow users to access directories/files that they should not access.
Any security checks should occur after the data has been decoded and validated as correct data format. Do not repeat decoding process, if bad character are left after decoding process, treat the data as suspicious, and fail the validation process.
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Parent Mitigation: SI, MA, AC, CM, SC



Parent Threat: Resource Manipulation

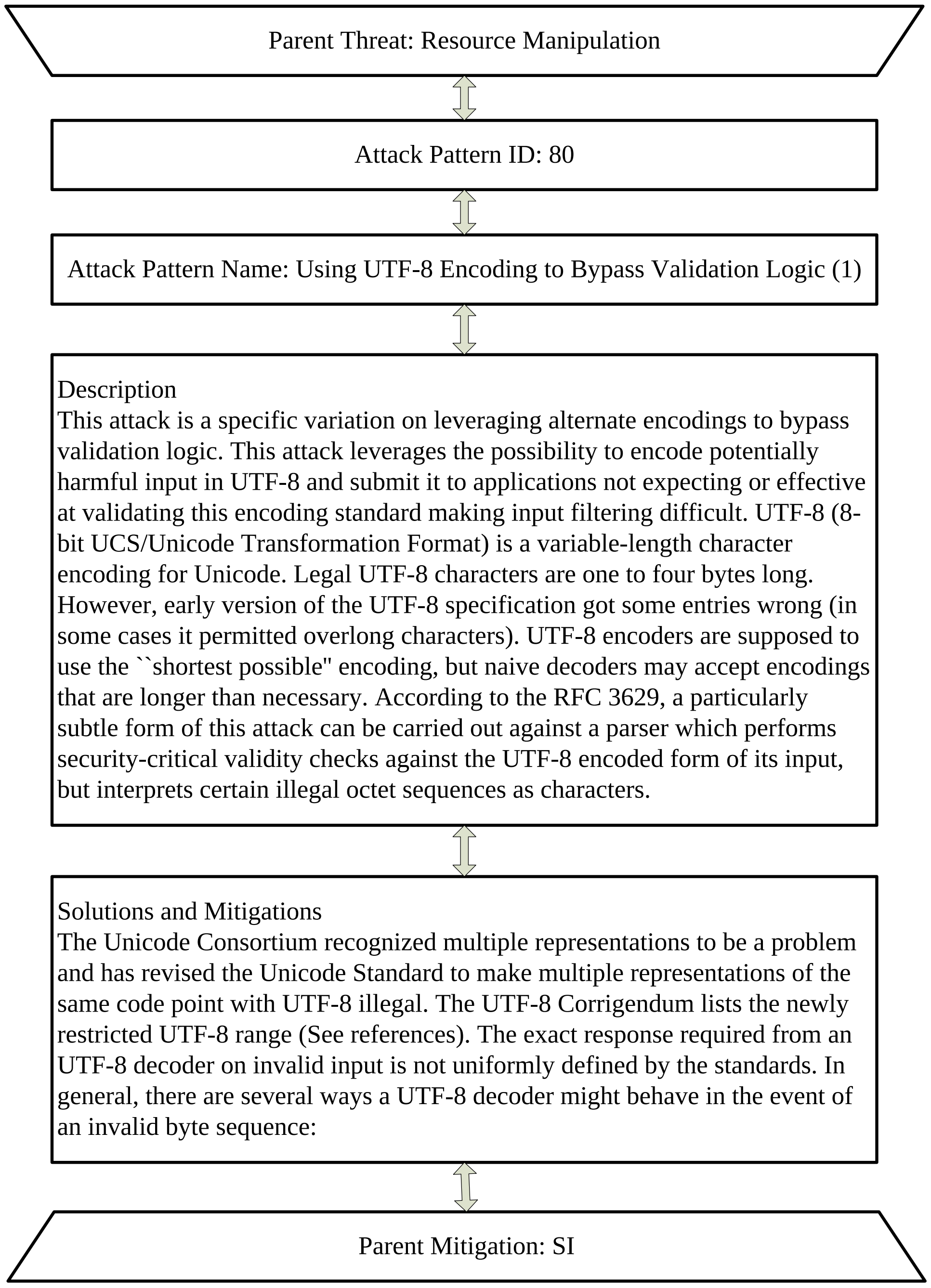
Attack Pattern ID: 79

Attack Pattern Name: Using Slashes in Alternate Encoding (3)

Description
This attack targets the encoding of the Slash characters. An attacker would try to exploit common filtering problems related to the use of the slashes characters to gain access to resources on the target host. Directory-driven systems, such as file systems and databases, typically use the slash character to indicate traversal between directories or other container components. For murky historical reasons, PCs (and, as a result, Microsoft OSs) choose to use a backslash, whereas the UNIX world typically makes use of the forward slash. The schizophrenic result is that many MS-based systems are required to understand both forms of the slash. This gives the attacker many opportunities to discover and abuse a number of common filtering problems. The goal of this pattern is to discover server software that only applies filters to one version, but not the other.

Solutions and Mitigations
Any security checks should occur after the data has been decoded and validated as correct data format. When client input is required from web-based forms, avoid using the “GET” method to submit data, as the method causes the form data to be appended to the URL and is easily manipulated. Instead, use the “POST method whenever possible. There are tools to scan HTTP requests to the server for valid URL such as URLScan from Microsoft (http:// www.microsoft.com/technet/security/tools/urlscan.mspx) Be aware of the threat of alternative method of data encoding and obfuscation technique such as IP address endoding. (Do not allow users to access directories/files that they should not access.Assume all input is malicious. Create a white list that defines all valid input to the application based on the requirements specifications. Input that does not match against the white list should not be permitted to enter into the system.

Parent Mitigation: SI, SC, AC



Parent Threat: Resource Manipulation

Attack Pattern ID: 80

Attack Pattern Name: Using UTF-8 Encoding to Bypass Validation Logic (1)

Description
This attack is a specific variation on leveraging alternate encodings to bypass validation logic. This attack leverages the possibility to encode potentially harmful input in UTF-8 and submit it to applications not expecting or effective at validating this encoding standard making input filtering difficult. UTF-8 (8-bit UCS/Unicode Transformation Format) is a variable-length character encoding for Unicode. Legal UTF-8 characters are one to four bytes long. However, early version of the UTF-8 specification got some entries wrong (in some cases it permitted overlong characters). UTF-8 encoders are supposed to use the ``shortest possible" encoding, but naive decoders may accept encodings that are longer than necessary. According to the RFC 3629, a particularly subtle form of this attack can be carried out against a parser which performs security-critical validity checks against the UTF-8 encoded form of its input, but interprets certain illegal octet sequences as characters.

Solutions and Mitigations
The Unicode Consortium recognized multiple representations to be a problem and has revised the Unicode Standard to make multiple representations of the same code point with UTF-8 illegal. The UTF-8 Corrigendum lists the newly restricted UTF-8 range (See references). The exact response required from an UTF-8 decoder on invalid input is not uniformly defined by the standards. In general, there are several ways a UTF-8 decoder might behave in the event of an invalid byte sequence:

Parent Mitigation: SI

Parent Threat: Resource Manipulation

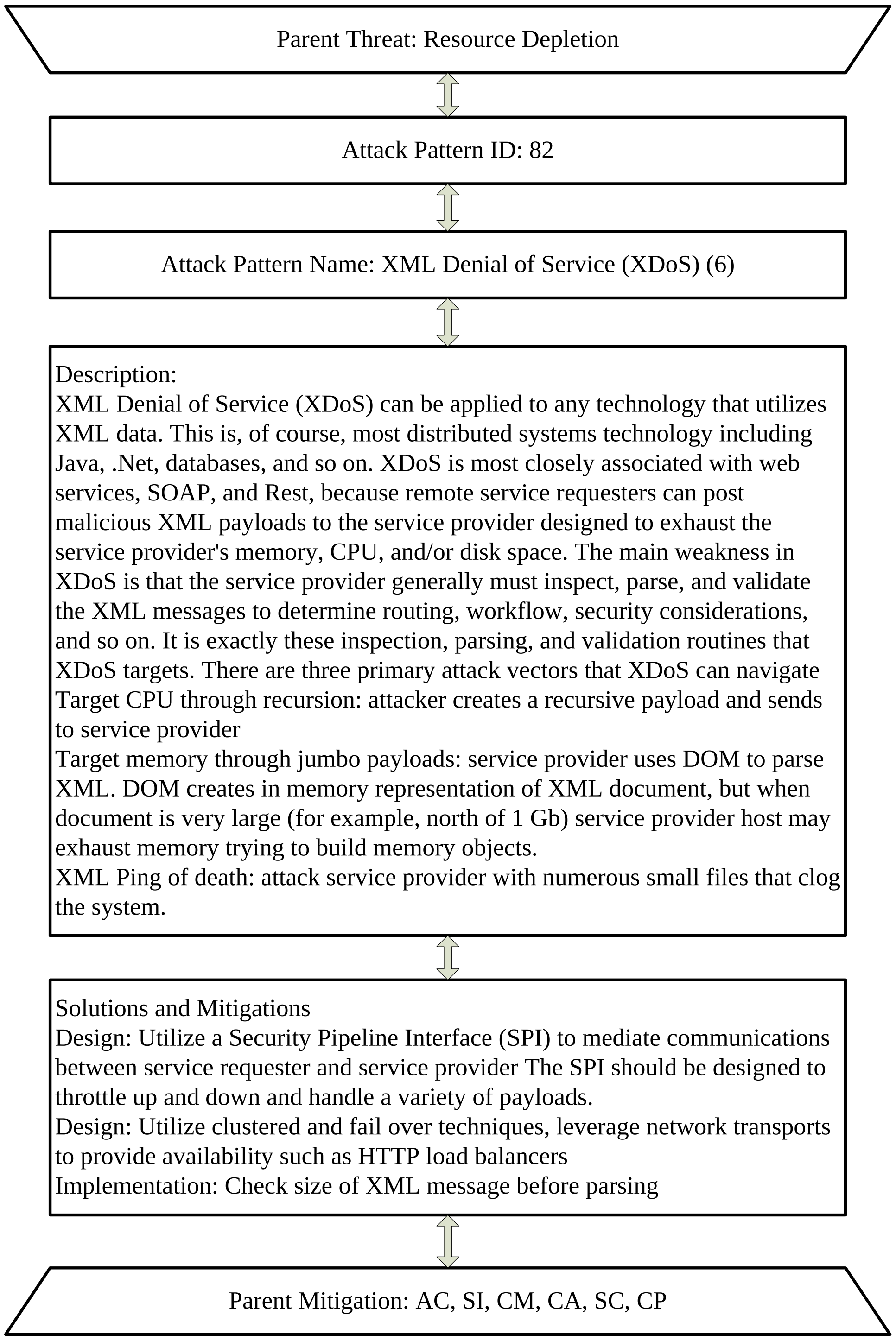
Attack Pattern ID: 81

Attack Pattern Name: Web Logs Tampering (3)

Discription
Protection services in security are vulnerable so they are backstopped by detection in the so-called protect-detect-respond model. A key element in detection is log files, to identify a threat impact, for audit purposes, or simply responding to a crash. While penetrating a system requires a set of skills, more advanced attackers will cover their tracks by manipulating log files to either erase entries or input false entries to throw the system administrators off their trail. Since most requests to web servers are logged (at least header request response data) the attacker literally has the ability to generate log data in every request. Of course this is not the same as always being able to delete otherwise tamper with log data.
Web Logs Tampering attacks involve an attacker injecting, deleting or otherwise tampering with the contents of web logs.
Additionally, writing malicious data to log files may target jobs, filters, reports, and other agents that process the logs in an asynchronous attack pattern.

Solutions and Mitigations
Design: Use input validation before writing to web log
Design: Validate all log data before it is output

Parent Mitigation: AC, AU, SI



Parent Threat: Resource Depletion

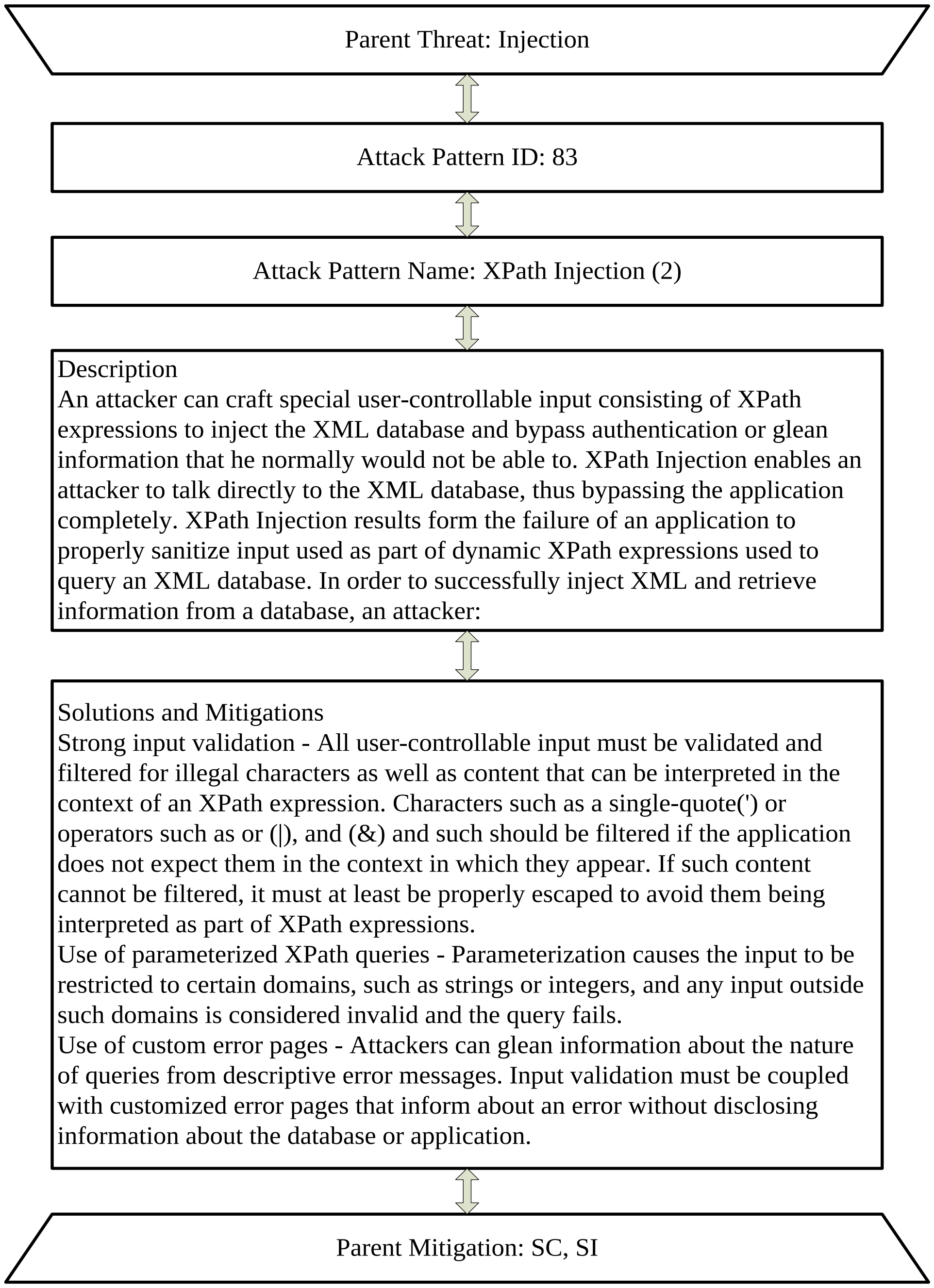
Attack Pattern ID: 82

Attack Pattern Name: XML Denial of Service (XDoS) (6)

Description:
XML Denial of Service (XDoS) can be applied to any technology that utilizes XML data. This is, of course, most distributed systems technology including Java, .Net, databases, and so on. XDoS is most closely associated with web services, SOAP, and Rest, because remote service requesters can post malicious XML payloads to the service provider designed to exhaust the service provider's memory, CPU, and/or disk space. The main weakness in XDoS is that the service provider generally must inspect, parse, and validate the XML messages to determine routing, workflow, security considerations, and so on. It is exactly these inspection, parsing, and validation routines that XDoS targets. There are three primary attack vectors that XDoS can navigate
Target CPU through recursion: attacker creates a recursive payload and sends to service provider
Target memory through jumbo payloads: service provider uses DOM to parse XML. DOM creates in memory representation of XML document, but when document is very large (for example, north of 1 Gb) service provider host may exhaust memory trying to build memory objects.
XML Ping of death: attack service provider with numerous small files that clog the system.

Solutions and Mitigations
Design: Utilize a Security Pipeline Interface (SPI) to mediate communications between service requester and service provider The SPI should be designed to throttle up and down and handle a variety of payloads.
Design: Utilize clustered and fail over techniques, leverage network transports to provide availability such as HTTP load balancers
Implementation: Check size of XML message before parsing

Parent Mitigation: AC, SI, CM, CA, SC, CP



Parent Threat: Injection

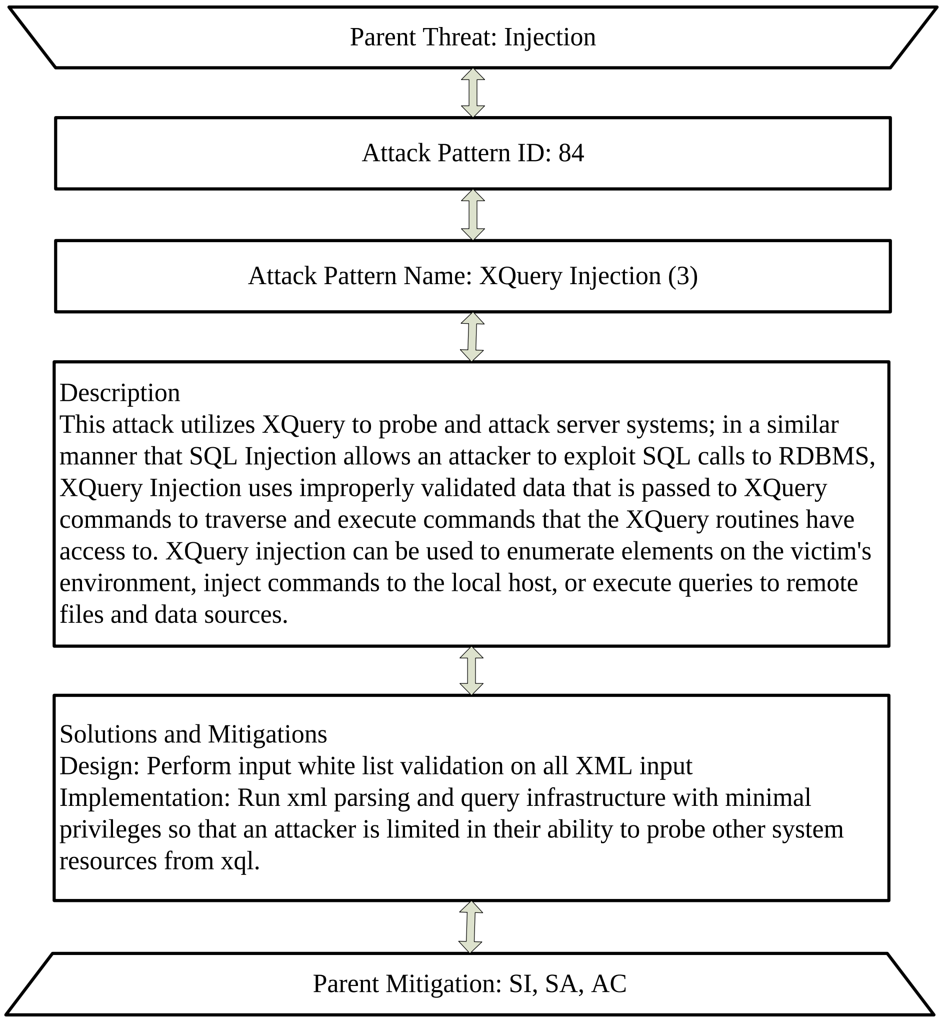
Attack Pattern ID: 83

Attack Pattern Name: XPath Injection (2)

Description
An attacker can craft special user-controllable input consisting of XPath expressions to inject the XML database and bypass authentication or glean information that he normally would not be able to. XPath Injection enables an attacker to talk directly to the XML database, thus bypassing the application completely. XPath Injection results form the failure of an application to properly sanitize input used as part of dynamic XPath expressions used to query an XML database. In order to successfully inject XML and retrieve information from a database, an attacker:

Solutions and Mitigations
Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as content that can be interpreted in the context of an XPath expression. Characters such as a single-quote(') or operators such as or (|), and (&) and such should be filtered if the application does not expect them in the context in which they appear. If such content cannot be filtered, it must at least be properly escaped to avoid them being interpreted as part of XPath expressions.
Use of parameterized XPath queries - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails.
Use of custom error pages - Attackers can glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.

Parent Mitigation: SC, SI



Parent Threat: Injection

Attack Pattern ID: 84

Attack Pattern Name: XQuery Injection (3)

Description
This attack utilizes XQuery to probe and attack server systems; in a similar manner that SQL Injection allows an attacker to exploit SQL calls to RDBMS, XQuery Injection uses improperly validated data that is passed to XQuery commands to traverse and execute commands that the XQuery routines have access to. XQuery injection can be used to enumerate elements on the victim's environment, inject commands to the local host, or execute queries to remote files and data sources.

Solutions and Mitigations
Design: Perform input white list validation on all XML input
Implementation: Run xml parsing and query infrastructure with minimal privileges so that an attacker is limited in their ability to probe other system resources from xql.

Parent Mitigation: SI, SA, AC

Parent Mitigation: Probabilistic Techniques

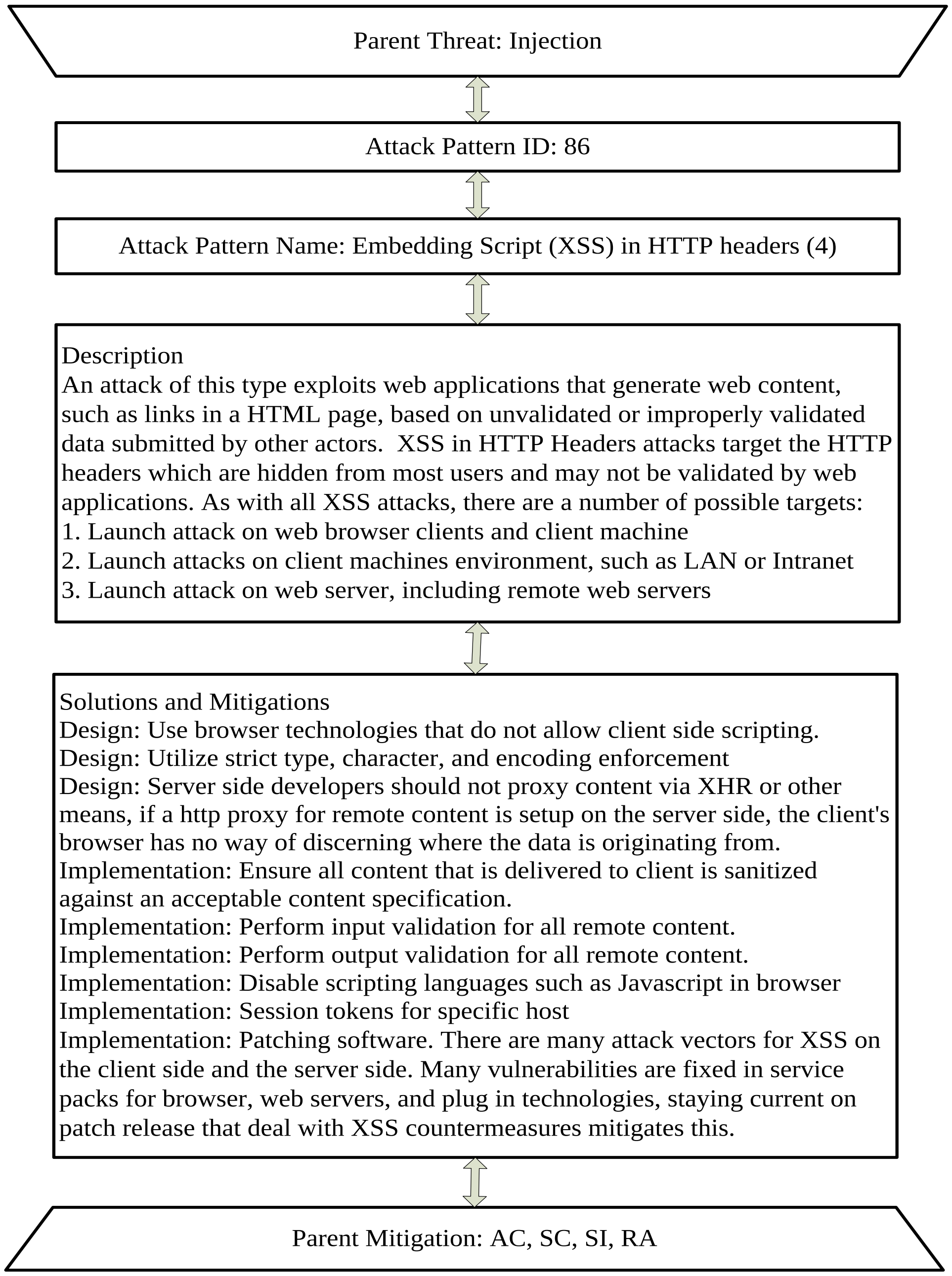
Attack Pattern ID: 85

Attack Pattern Name: Client Network Footprinting (using Ajax/XSS) (4)

Description
This attack utilizes the frequent client-server roundtrips in Ajax conversation to scan a system. While Ajax does not open up new vulnerabilities per se, it does optimize them from an attacker point of view. In many XSS attacks the attacker must get a "hole in one" and successfully exploit the vulnerability on the victim side the first time, once the client is redirected the attacker has many chances to engage in follow on probes, but their is only one first chance. In a widely used web application this is not a major problem because 1 in a 1,000 is good enough in a widely used application.
A common first step for an attacker is to footprint the environment to understand what attacks will work. Since footprinting relies on enumeration, the conversational pattern of rapid, multiple requests and responses that are typical in Ajax applications enable an attacker to look for many vulnerabilities, well known ports, network locations and so on.

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting
Design: Utilize strict type, character, and encoding enforcement
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser
Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.

Parent Mitigation: AC, SC, SI, RA



Parent Threat: Injection

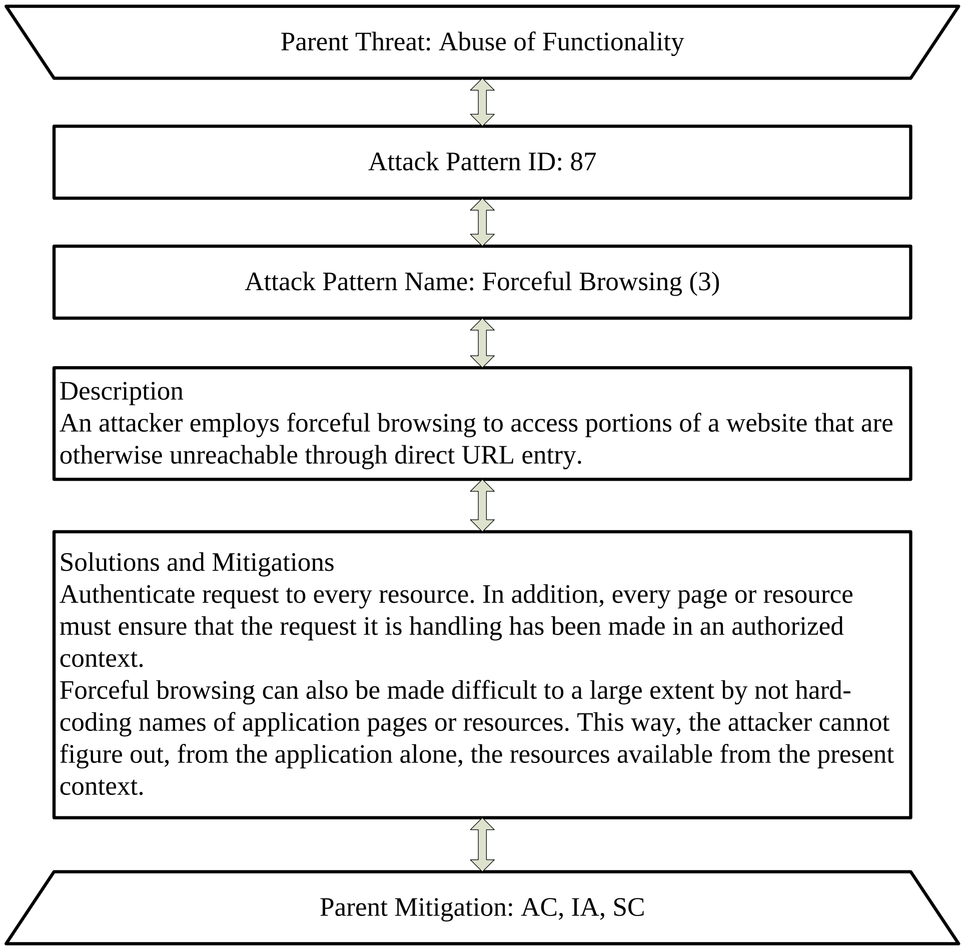
Attack Pattern ID: 86

Attack Pattern Name: Embedding Script (XSS) in HTTP headers (4)

Description
An attack of this type exploits web applications that generate web content, such as links in a HTML page, based on unvalidated or improperly validated data submitted by other actors. XSS in HTTP Headers attacks target the HTTP headers which are hidden from most users and may not be validated by web applications. As with all XSS attacks, there are a number of possible targets:
1. Launch attack on web browser clients and client machine
2. Launch attacks on client machines environment, such as LAN or Intranet
3. Launch attack on web server, including remote web servers

Solutions and Mitigations
Design: Use browser technologies that do not allow client side scripting.
Design: Utilize strict type, character, and encoding enforcement
Design: Server side developers should not proxy content via XHR or other means, if a http proxy for remote content is setup on the server side, the client's browser has no way of discerning where the data is originating from.
Implementation: Ensure all content that is delivered to client is sanitized against an acceptable content specification.
Implementation: Perform input validation for all remote content.
Implementation: Perform output validation for all remote content.
Implementation: Disable scripting languages such as Javascript in browser
Implementation: Session tokens for specific host
Implementation: Patching software. There are many attack vectors for XSS on the client side and the server side. Many vulnerabilities are fixed in service packs for browser, web servers, and plug in technologies, staying current on patch release that deal with XSS countermeasures mitigates this.

Parent Mitigation: AC, SC, SI, RA



Parent Threat: Abuse of Functionality

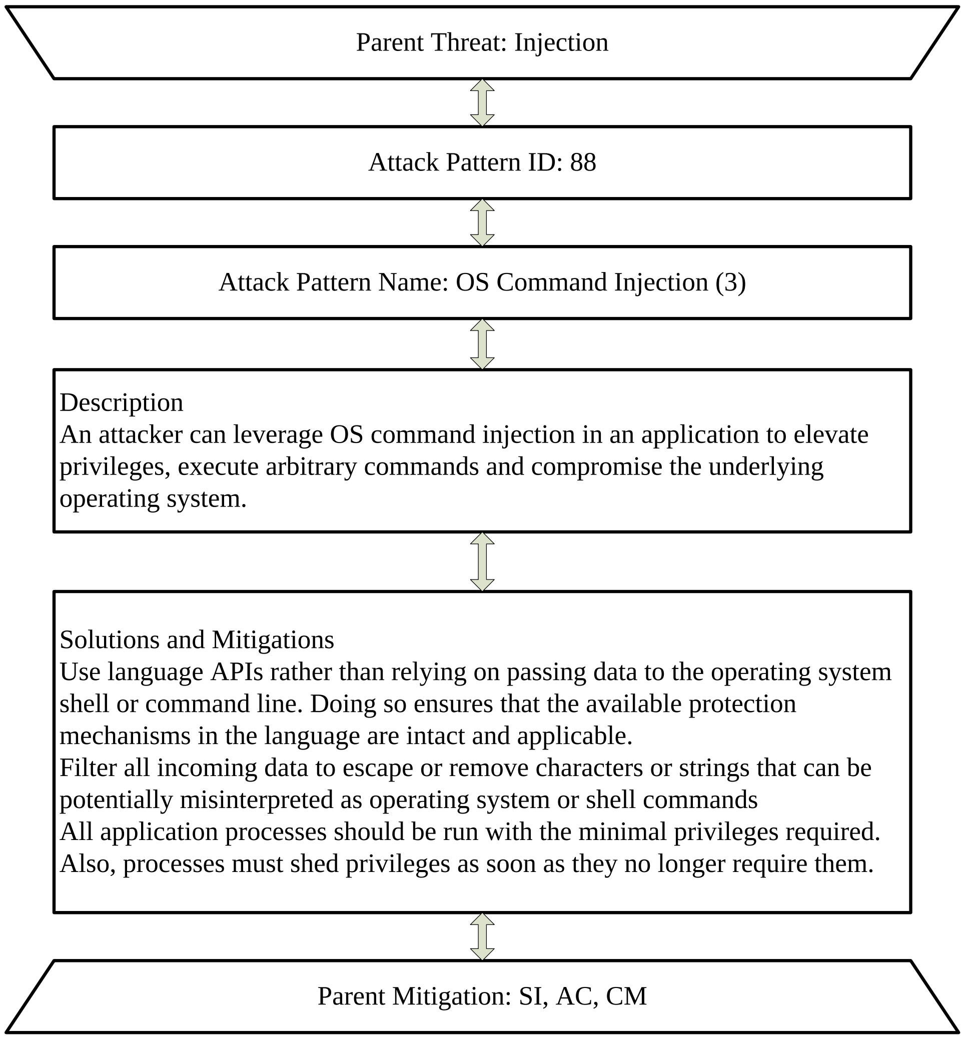
Attack Pattern ID: 87

Attack Pattern Name: Forceful Browsing (3)

Description
An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry.

Solutions and Mitigations
Authenticate request to every resource. In addition, every page or resource must ensure that the request it is handling has been made in an authorized context.
Forceful browsing can also be made difficult to a large extent by not hard-coding names of application pages or resources. This way, the attacker cannot figure out, from the application alone, the resources available from the present context.

Parent Mitigation: AC, IA, SC



Parent Threat: Injection

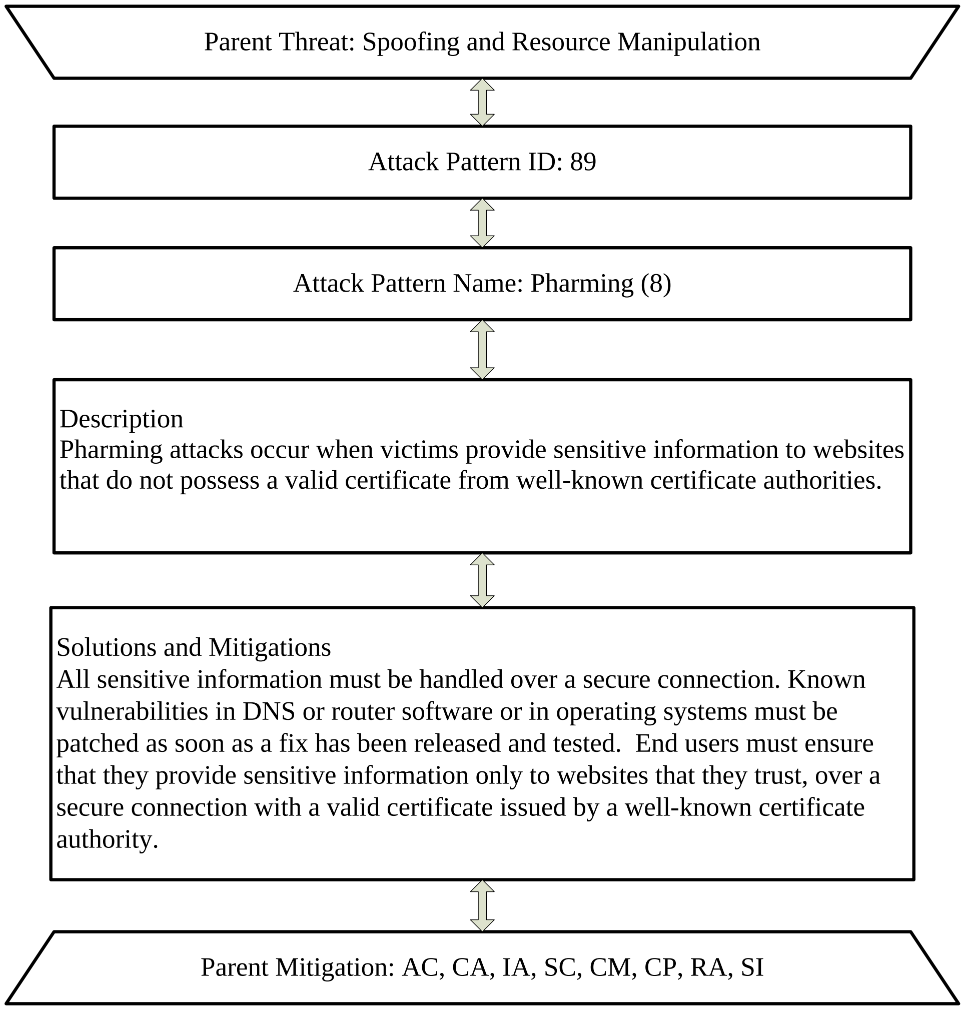
Attack Pattern ID: 88

Attack Pattern Name: OS Command Injection (3)

Description
An attacker can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.

Solutions and Mitigations
Use language APIs rather than relying on passing data to the operating system shell or command line. Doing so ensures that the available protection mechanisms in the language are intact and applicable.
Filter all incoming data to escape or remove characters or strings that can be potentially misinterpreted as operating system or shell commands
All application processes should be run with the minimal privileges required. Also, processes must shed privileges as soon as they no longer require them.

Parent Mitigation: SI, AC, CM



Parent Threat: Spoofing and Resource Manipulation

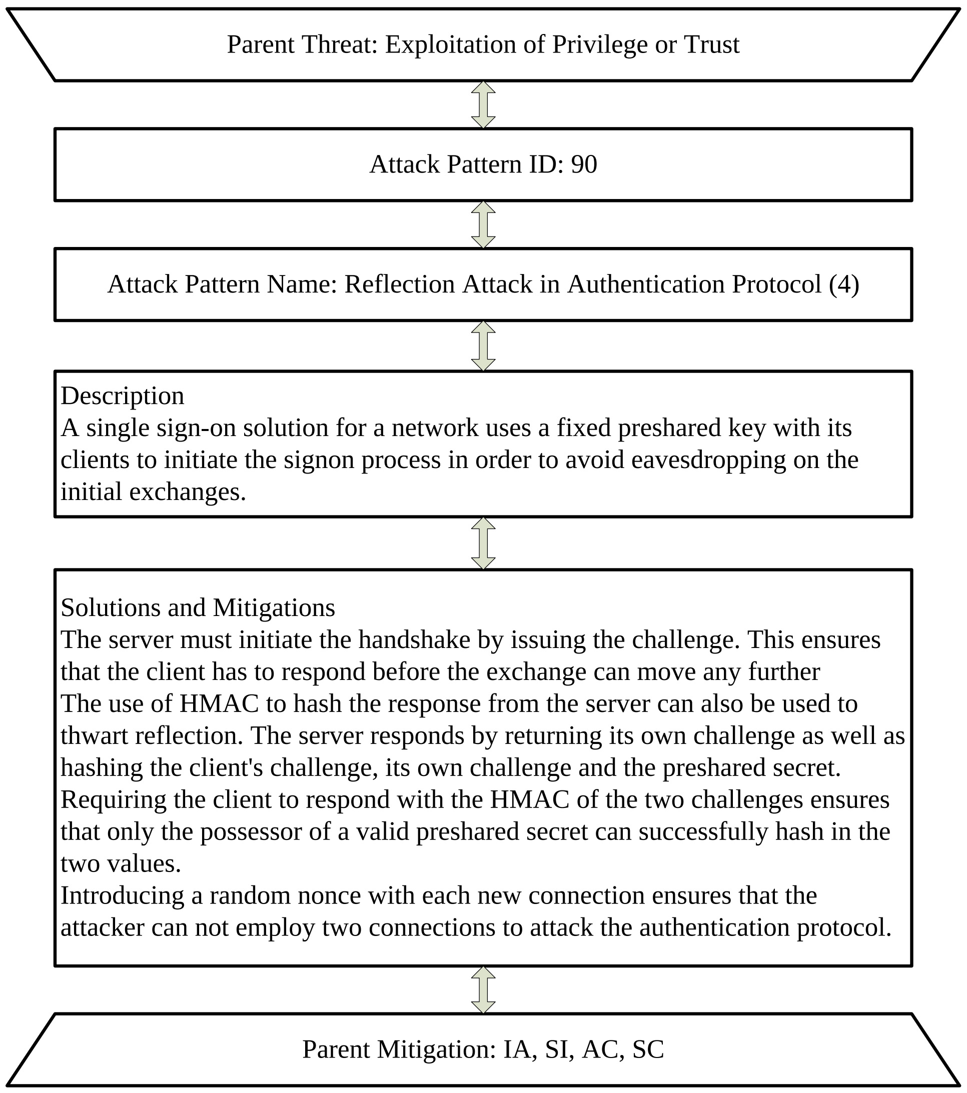
Attack Pattern ID: 89

Attack Pattern Name: Pharming (8)

Description
Pharming attacks occur when victims provide sensitive information to websites that do not possess a valid certificate from well-known certificate authorities.

Solutions and Mitigations
All sensitive information must be handled over a secure connection. Known vulnerabilities in DNS or router software or in operating systems must be patched as soon as a fix has been released and tested. End users must ensure that they provide sensitive information only to websites that they trust, over a secure connection with a valid certificate issued by a well-known certificate authority.

Parent Mitigation: AC, CA, IA, SC, CM, CP, RA, SI



Parent Threat: Exploitation of Privilege or Trust

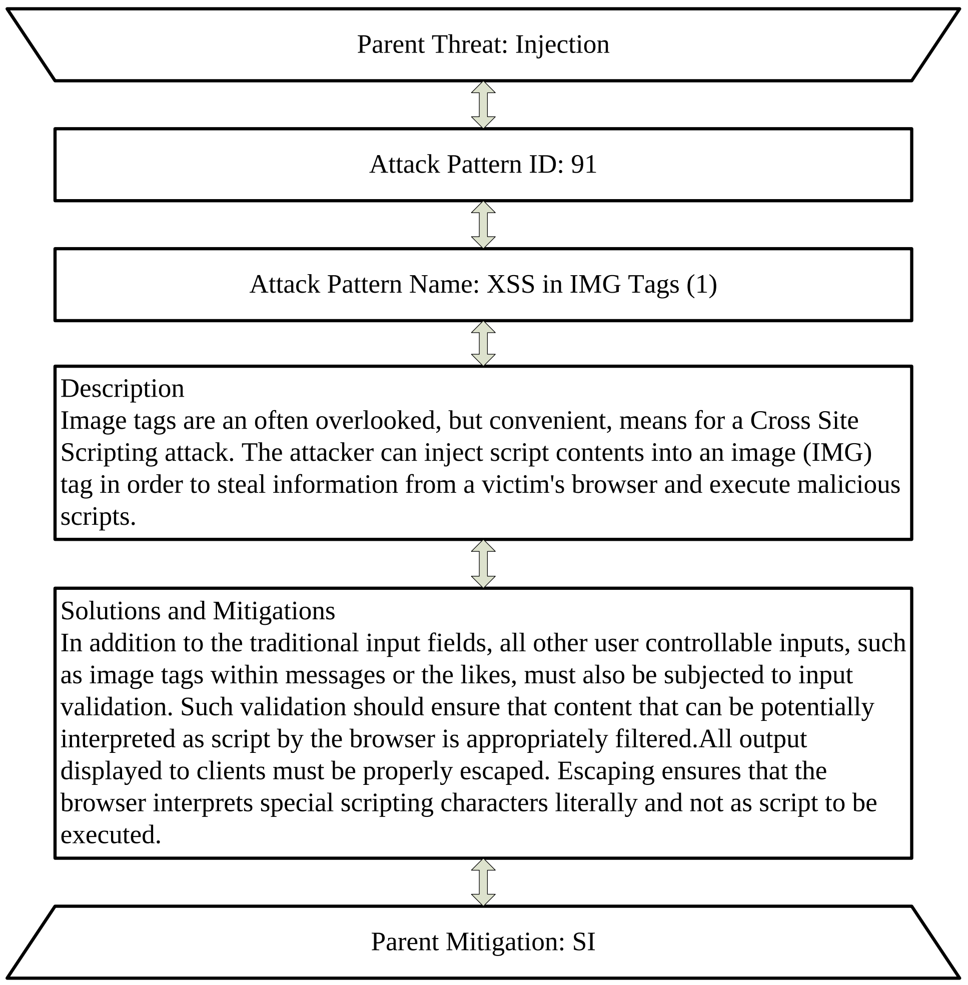
Attack Pattern ID: 90

Attack Pattern Name: Reflection Attack in Authentication Protocol (4)

Description
A single sign-on solution for a network uses a fixed preshared key with its clients to initiate the signon process in order to avoid eavesdropping on the initial exchanges.

Solutions and Mitigations
The server must initiate the handshake by issuing the challenge. This ensures that the client has to respond before the exchange can move any further
The use of HMAC to hash the response from the server can also be used to thwart reflection. The server responds by returning its own challenge as well as hashing the client's challenge, its own challenge and the preshared secret. Requiring the client to respond with the HMAC of the two challenges ensures that only the possessor of a valid preshared secret can successfully hash in the two values.
Introducing a random nonce with each new connection ensures that the attacker can not employ two connections to attack the authentication protocol.

Parent Mitigation: IA, SI, AC, SC



Parent Threat: Injection

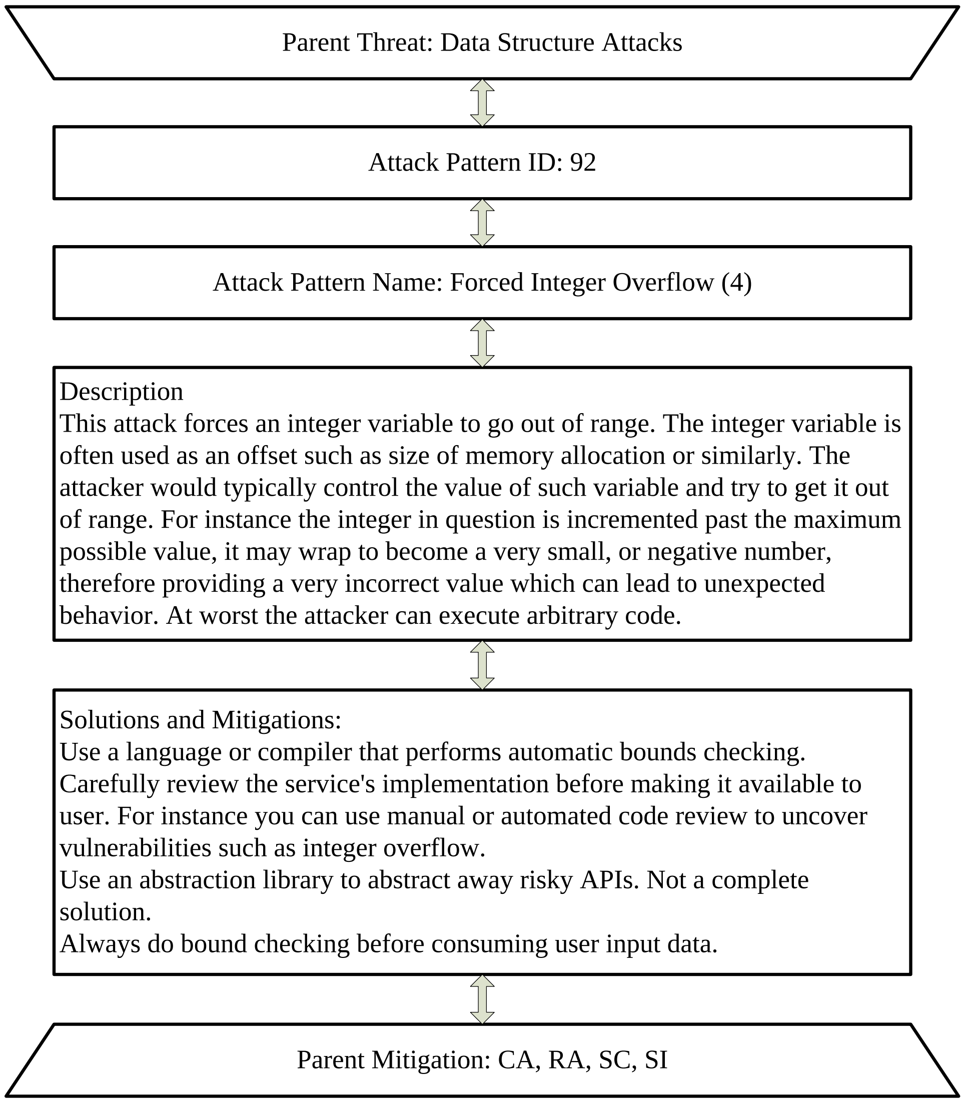
Attack Pattern ID: 91

Attack Pattern Name: XSS in IMG Tags (1)

Description
Image tags are an often overlooked, but convenient, means for a Cross Site Scripting attack. The attacker can inject script contents into an image (IMG) tag in order to steal information from a victim's browser and execute malicious scripts.

Solutions and Mitigations
In addition to the traditional input fields, all other user controllable inputs, such as image tags within messages or the likes, must also be subjected to input validation. Such validation should ensure that content that can be potentially interpreted as script by the browser is appropriately filtered.All output displayed to clients must be properly escaped. Escaping ensures that the browser interprets special scripting characters literally and not as script to be executed.

Parent Mitigation: SI



Parent Threat: Data Structure Attacks

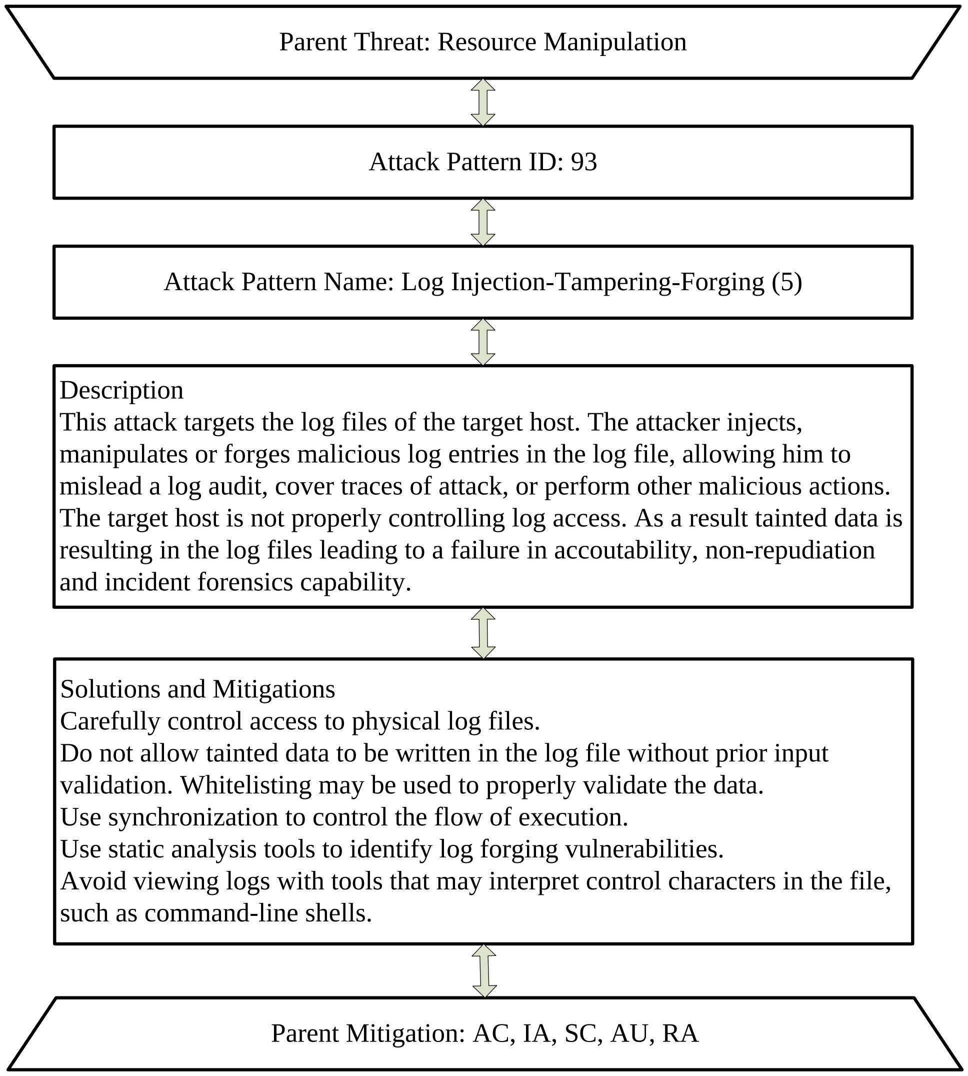
Attack Pattern ID: 92

Attack Pattern Name: Forced Integer Overflow (4)

Description
This attack forces an integer variable to go out of range. The integer variable is often used as an offset such as size of memory allocation or similarly. The attacker would typically control the value of such variable and try to get it out of range. For instance the integer in question is incremented past the maximum possible value, it may wrap to become a very small, or negative number, therefore providing a very incorrect value which can lead to unexpected behavior. At worst the attacker can execute arbitrary code.

Solutions and Mitigations:
Use a language or compiler that performs automatic bounds checking.
Carefully review the service's implementation before making it available to user. For instance you can use manual or automated code review to uncover vulnerabilities such as integer overflow.
Use an abstraction library to abstract away risky APIs. Not a complete solution.
Always do bound checking before consuming user input data.

Parent Mitigation: CA, RA, SC, SI



Parent Threat: Resource Manipulation

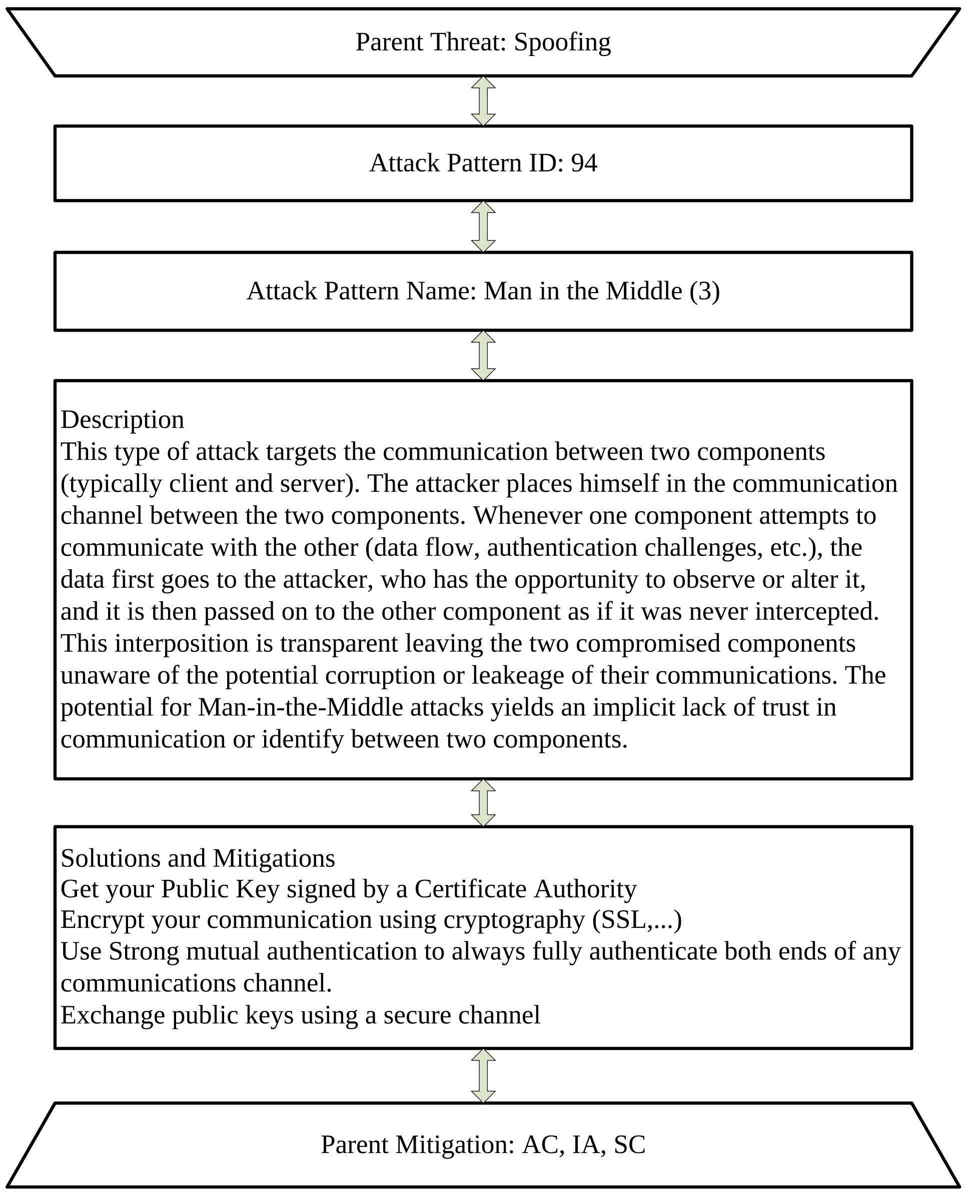
Attack Pattern ID: 93

Attack Pattern Name: Log Injection-Tampering-Forging (5)

Description
This attack targets the log files of the target host. The attacker injects, manipulates or forges malicious log entries in the log file, allowing him to mislead a log audit, cover traces of attack, or perform other malicious actions. The target host is not properly controlling log access. As a result tainted data is resulting in the log files leading to a failure in accoutability, non-repudiation and incident forensics capability.

Solutions and Mitigations
Carefully control access to physical log files.
Do not allow tainted data to be written in the log file without prior input validation. Whitelisting may be used to properly validate the data.
Use synchronization to control the flow of execution.
Use static analysis tools to identify log forging vulnerabilities.
Avoid viewing logs with tools that may interpret control characters in the file, such as command-line shells.

Parent Mitigation: AC, IA, SC, AU, RA



Parent Threat: Spoofing

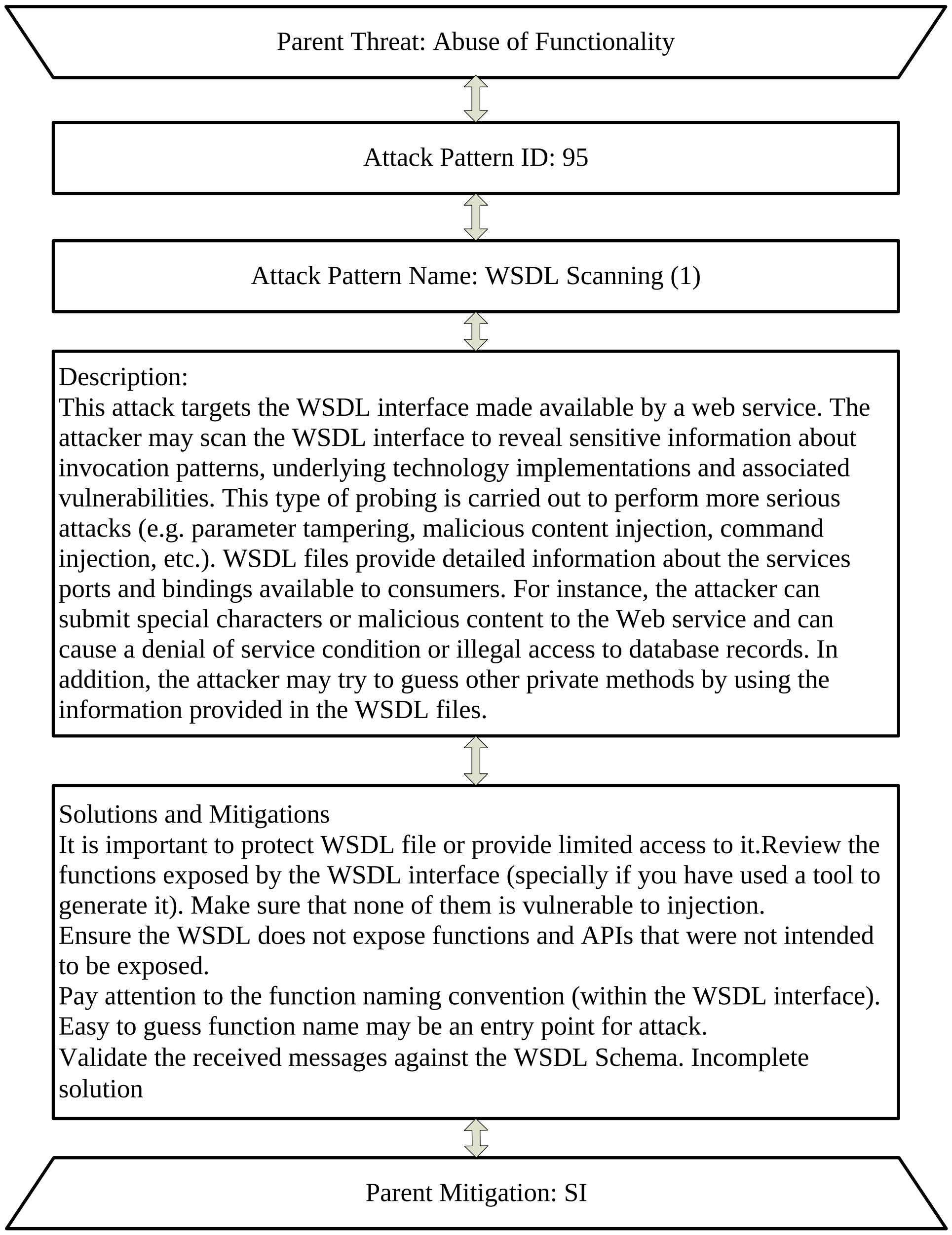
Attack Pattern ID: 94

Attack Pattern Name: Man in the Middle (3)

Description
This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never intercepted. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakeage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components.

Solutions and Mitigations
Get your Public Key signed by a Certificate Authority
Encrypt your communication using cryptography (SSL,...)
Use Strong mutual authentication to always fully authenticate both ends of any communications channel.
Exchange public keys using a secure channel

Parent Mitigation: AC, IA, SC



Parent Threat: Abuse of Functionality

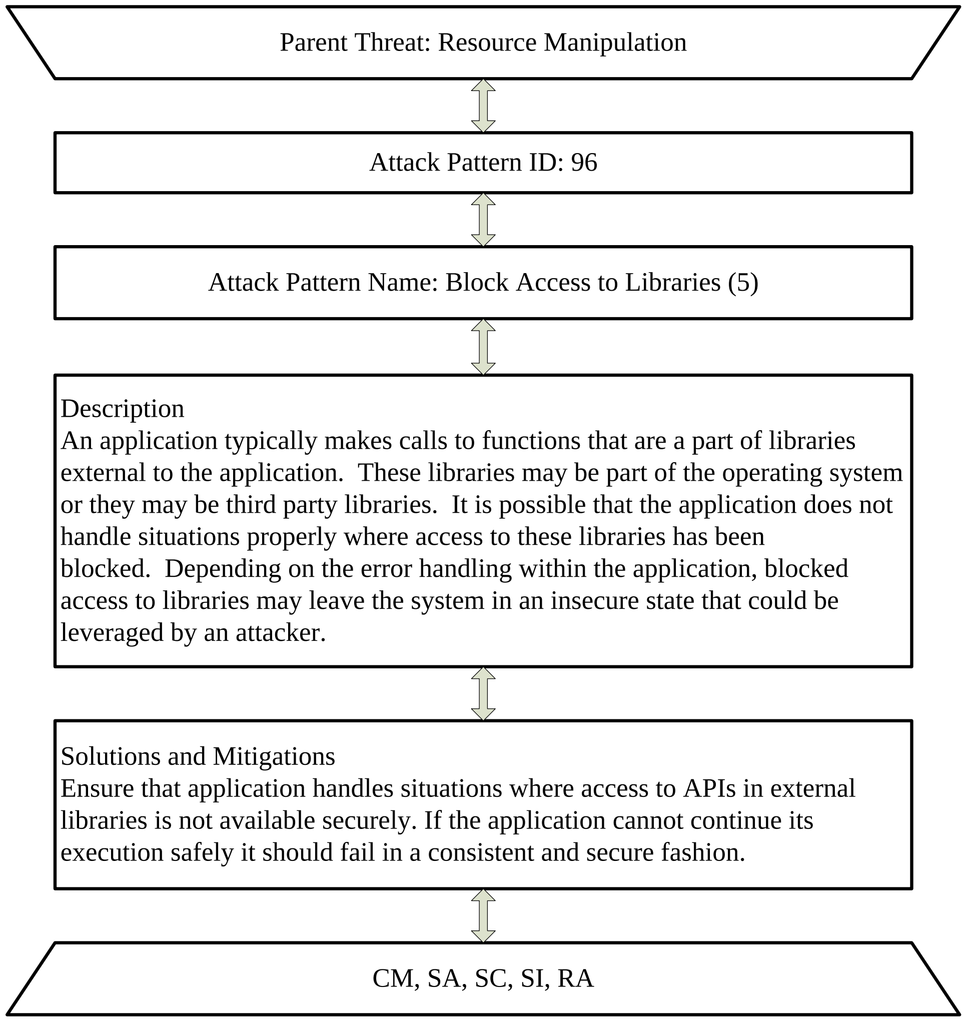
Attack Pattern ID: 95

Attack Pattern Name: WSDL Scanning (1)

Description:
This attack targets the WSDL interface made available by a web service. The attacker may scan the WSDL interface to reveal sensitive information about invocation patterns, underlying technology implementations and associated vulnerabilities. This type of probing is carried out to perform more serious attacks (e.g. parameter tampering, malicious content injection, command injection, etc.). WSDL files provide detailed information about the services ports and bindings available to consumers. For instance, the attacker can submit special characters or malicious content to the Web service and can cause a denial of service condition or illegal access to database records. In addition, the attacker may try to guess other private methods by using the information provided in the WSDL files.

Solutions and Mitigations
It is important to protect WSDL file or provide limited access to it.Review the functions exposed by the WSDL interface (specially if you have used a tool to generate it). Make sure that none of them is vulnerable to injection.
Ensure the WSDL does not expose functions and APIs that were not intended to be exposed.
Pay attention to the function naming convention (within the WSDL interface). Easy to guess function name may be an entry point for attack.
Validate the received messages against the WSDL Schema. Incomplete solution

Parent Mitigation: SI



Parent Threat: Resource Manipulation

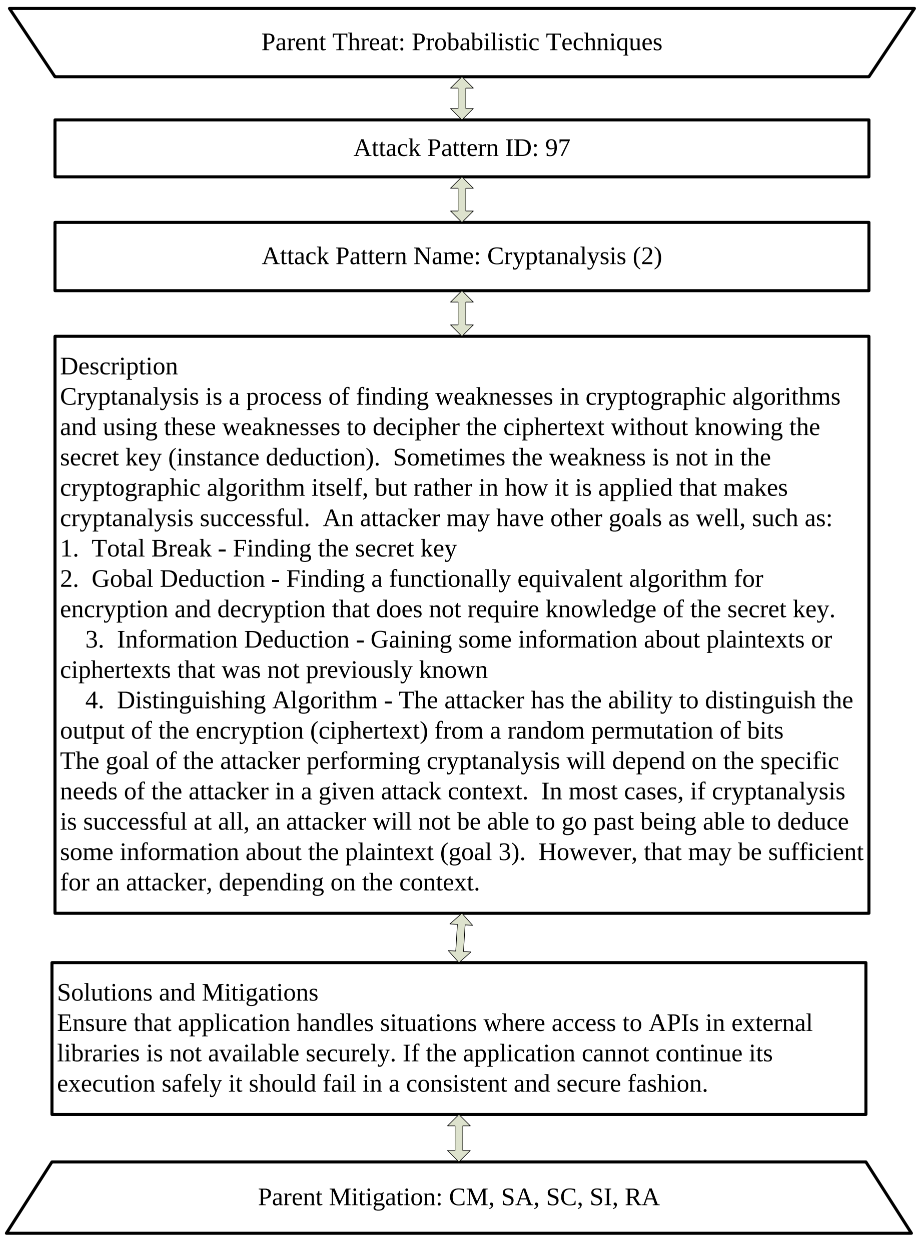
Attack Pattern ID: 96

Attack Pattern Name: Block Access to Libraries (5)

Description
An application typically makes calls to functions that are a part of libraries external to the application. These libraries may be part of the operating system or they may be third party libraries. It is possible that the application does not handle situations properly where access to these libraries has been blocked. Depending on the error handling within the application, blocked access to libraries may leave the system in an insecure state that could be leveraged by an attacker.

Solutions and Mitigations
Ensure that application handles situations where access to APIs in external libraries is not available securely. If the application cannot continue its execution safely it should fail in a consistent and secure fashion.

CM, SA, SC, SI, RA



Parent Threat: Probabilistic Techniques

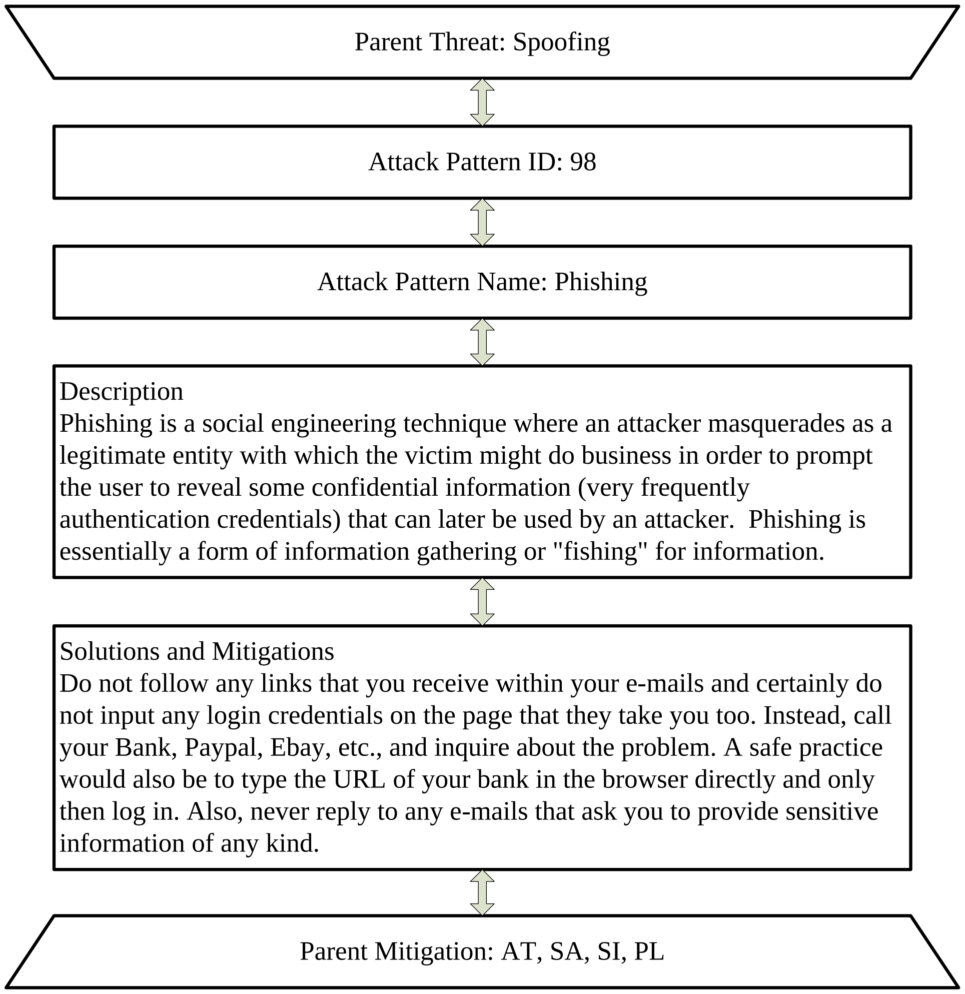
Attack Pattern ID: 97

Attack Pattern Name: Cryptanalysis (2)

Description
Cryptanalysis is a process of finding weaknesses in cryptographic algorithms and using these weaknesses to decipher the ciphertext without knowing the secret key (instance deduction). Sometimes the weakness is not in the cryptographic algorithm itself, but rather in how it is applied that makes cryptanalysis successful. An attacker may have other goals as well, such as:
1. Total Break - Finding the secret key
2. Gobal Deduction - Finding a functionally equivalent algorithm for encryption and decryption that does not require knowledge of the secret key.
3. Information Deduction - Gaining some information about plaintexts or ciphertexts that was not previously known
4. Distinguishing Algorithm - The attacker has the ability to distinguish the output of the encryption (ciphertext) from a random permutation of bits
The goal of the attacker performing cryptanalysis will depend on the specific needs of the attacker in a given attack context. In most cases, if cryptanalysis is successful at all, an attacker will not be able to go past being able to deduce some information about the plaintext (goal 3). However, that may be sufficient for an attacker, depending on the context.

Solutions and Mitigations
Ensure that application handles situations where access to APIs in external libraries is not available securely. If the application cannot continue its execution safely it should fail in a consistent and secure fashion.

Parent Mitigation: CM, SA, SC, SI, RA



Parent Threat: Spoofing

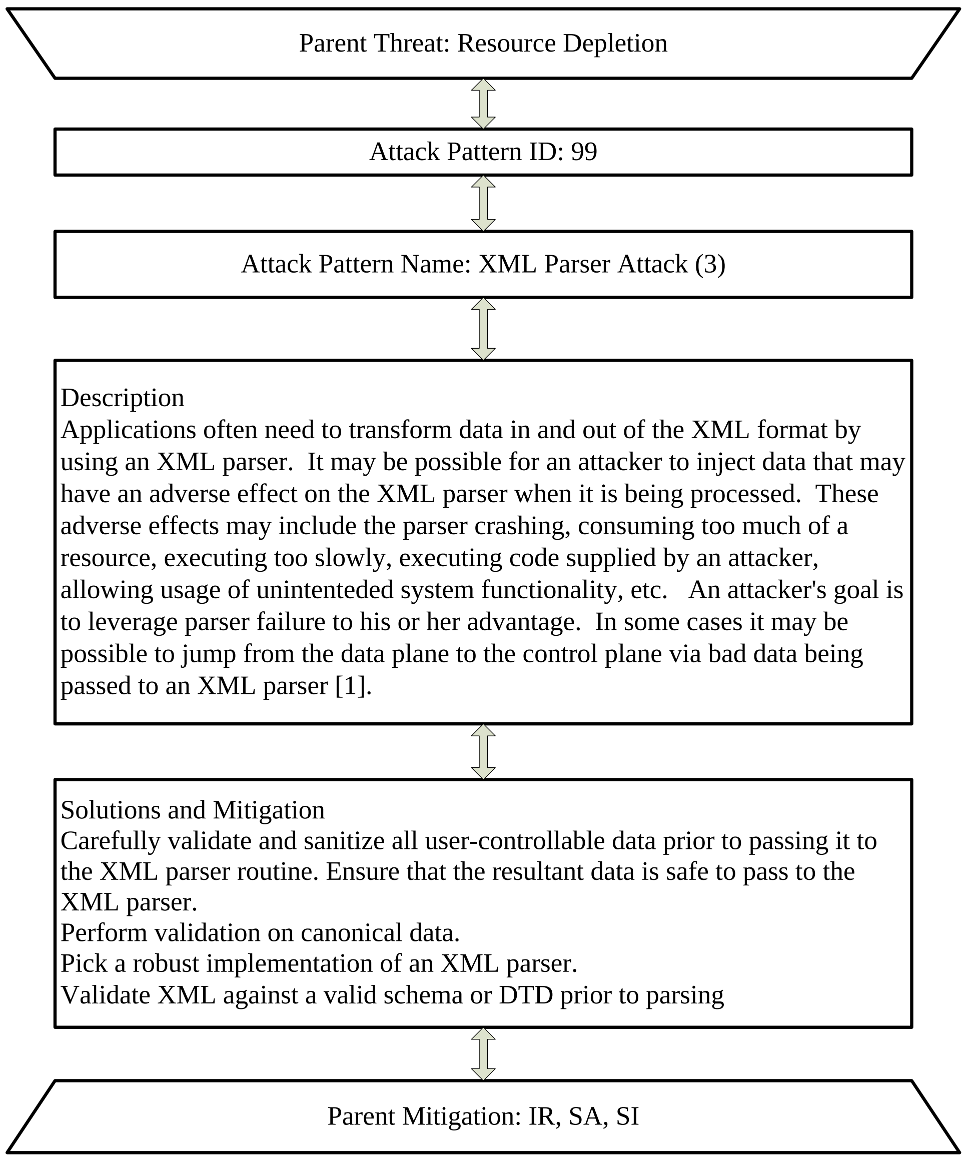
Attack Pattern ID: 98

Attack Pattern Name: Phishing

Description
Phishing is a social engineering technique where an attacker masquerades as a legitimate entity with which the victim might do business in order to prompt the user to reveal some confidential information (very frequently authentication credentials) that can later be used by an attacker. Phishing is essentially a form of information gathering or "fishing" for information.

Solutions and Mitigations
Do not follow any links that you receive within your e-mails and certainly do not input any login credentials on the page that they take you too. Instead, call your Bank, Paypal, Ebay, etc., and inquire about the problem. A safe practice would also be to type the URL of your bank in the browser directly and only then log in. Also, never reply to any e-mails that ask you to provide sensitive information of any kind.

Parent Mitigation: AT, SA, SI, PL



Parent Threat: Resource Depletion

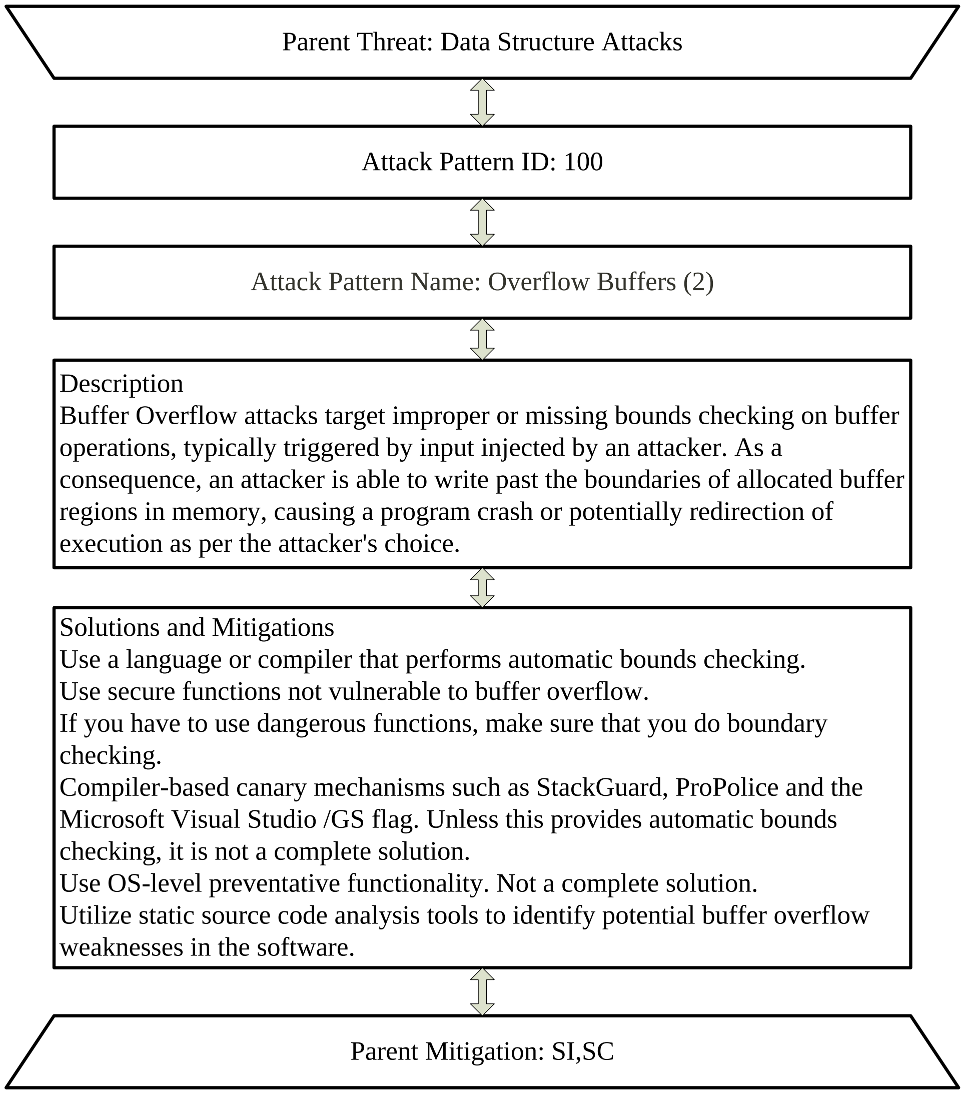
Attack Pattern ID: 99

Attack Pattern Name: XML Parser Attack (3)

Description
Applications often need to transform data in and out of the XML format by using an XML parser. It may be possible for an attacker to inject data that may have an adverse effect on the XML parser when it is being processed. These adverse effects may include the parser crashing, consuming too much of a resource, executing too slowly, executing code supplied by an attacker, allowing usage of unintenteded system functionality, etc. An attacker's goal is to leverage parser failure to his or her advantage. In some cases it may be possible to jump from the data plane to the control plane via bad data being passed to an XML parser [1].

Solutions and Mitigation
Carefully validate and sanitize all user-controllable data prior to passing it to the XML parser routine. Ensure that the resultant data is safe to pass to the XML parser.
Perform validation on canonical data.
Pick a robust implementation of an XML parser.
Validate XML against a valid schema or DTD prior to parsing

Parent Mitigation: IR, SA, SI



Parent Threat: Data Structure Attacks

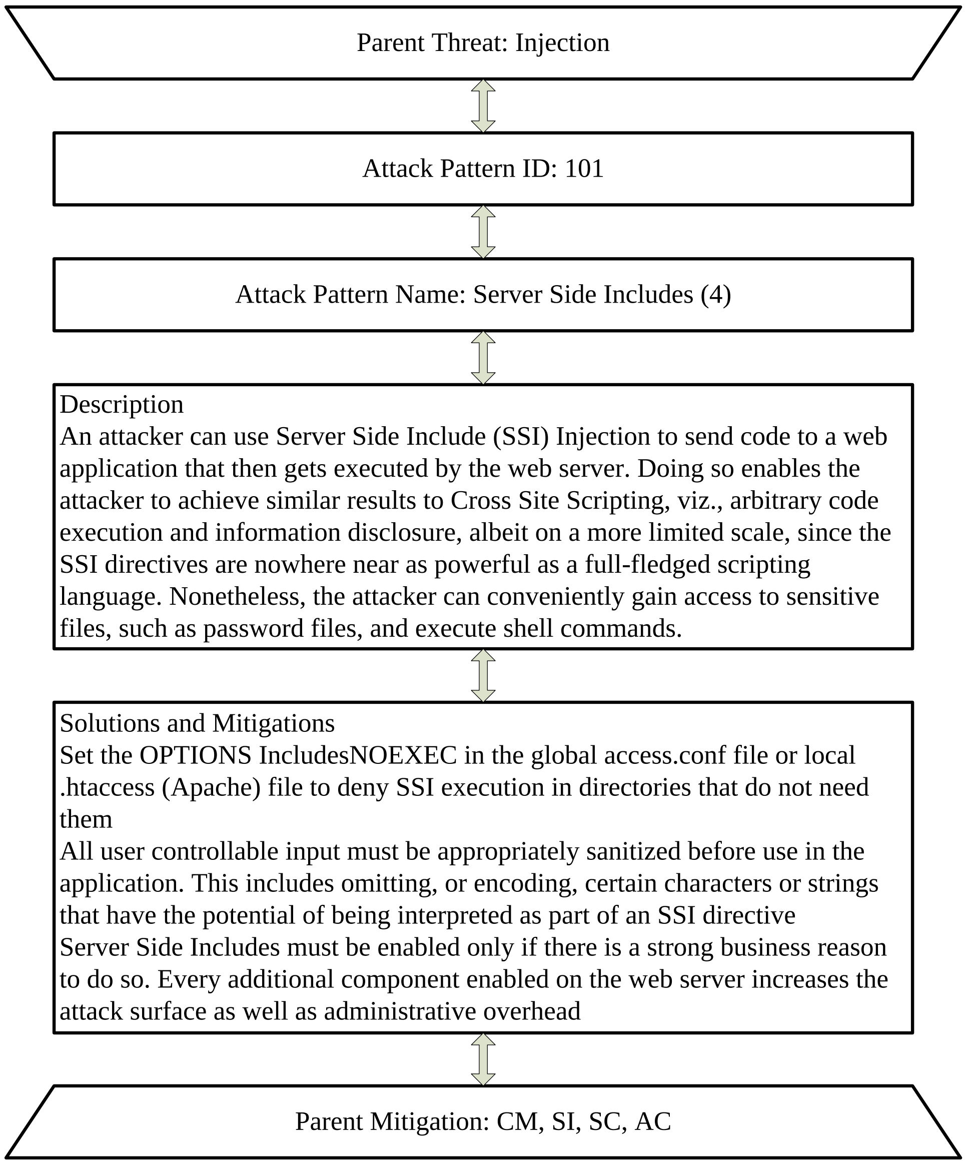
Attack Pattern ID: 100

Attack Pattern Name: Overflow Buffers (2)

Description
Buffer Overflow attacks target improper or missing bounds checking on buffer operations, typically triggered by input injected by an attacker. As a consequence, an attacker is able to write past the boundaries of allocated buffer regions in memory, causing a program crash or potentially redirection of execution as per the attacker's choice.

Solutions and Mitigations
Use a language or compiler that performs automatic bounds checking.
Use secure functions not vulnerable to buffer overflow.
If you have to use dangerous functions, make sure that you do boundary checking.
Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
Use OS-level preventative functionality. Not a complete solution.
Utilize static source code analysis tools to identify potential buffer overflow weaknesses in the software.

Parent Mitigation: SI,SC



Parent Threat: Injection

Attack Pattern ID: 101

Attack Pattern Name: Server Side Includes (4)

Description
An attacker can use Server Side Include (SSI) Injection to send code to a web application that then gets executed by the web server. Doing so enables the attacker to achieve similar results to Cross Site Scripting, viz., arbitrary code execution and information disclosure, albeit on a more limited scale, since the SSI directives are nowhere near as powerful as a full-fledged scripting language. Nonetheless, the attacker can conveniently gain access to sensitive files, such as password files, and execute shell commands.

Solutions and Mitigations
Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them
All user controllable input must be appropriately sanitized before use in the application. This includes omitting, or encoding, certain characters or strings that have the potential of being interpreted as part of an SSI directive
Server Side Includes must be enabled only if there is a strong business reason to do so. Every additional component enabled on the web server increases the attack surface as well as administrative overhead

Parent Mitigation: CM, SI, SC, AC

PMP Summary Table

PM	**PMP**
AC	81.220
AT	9.19
AU	14.28
CA	20.30
CM	43.130
CP	8.17
IA	31.72
MA	9.13
PE	2.2
PL	7.11
RA	21.42
SA	20.67
SC	55.160
SI	74.235