

North Carolina Agricultural and Technical State University
Aggie Digital Collections and Scholarship

Theses

Electronic Theses and Dissertations

2014

Using Capec Attack Patterns For Developing Abuse Cases

Emmanuel Borkor Nuakoh

North Carolina Agricultural and Technical State University

Follow this and additional works at: <https://digital.library.ncat.edu/theses>

Recommended Citation

Nuakoh, Emmanuel Borkor, "Using Capec Attack Patterns For Developing Abuse Cases" (2014). *Theses*. 245.

<https://digital.library.ncat.edu/theses/245>

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Aggie Digital Collections and Scholarship. It has been accepted for inclusion in Theses by an authorized administrator of Aggie Digital Collections and Scholarship. For more information, please contact iyanna@ncat.edu.

Using CAPEC Attack Patterns for Developing Abuse Cases

Emmanuel Borkor Nuakoh

North Carolina A&T State University

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department: Computer Science

Major: Computer Science

Major Professor: Dr. Dorothy Xiaohong Yuan

Greensboro, North Carolina

2014

The Graduate School
North Carolina Agricultural and Technical State University

This is to certify that the Master's Thesis of

Emmanuel Borkor Nuakoh

has met the thesis requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2014

Approved by:

Dr. Dorothy Xiaohong Yuan
Major Professor

Dr. Dorothy Xiaohong Yuan
Committee Member

Dr. Anna Huiming Yu
Committee Member

Dr. Justin Zhijun Zhan
Committee Member

Dr. Gerry Vernon Dozier
Department Chair

Dr. Sanjiv Sarin
Dean, The Graduate School

© Copyright by

Emmanuel Borkor Nuakoh

2014

Biographical Sketch

Emmanuel Borkor Nuakoh obtained his Bachelor of Science Geological Engineering degree from the University of Mines and Technology, Tarkwa, Ghana. He worked at Noble Gold Resources Ltd., Bibiani, Ghana, from October 2011 to February 2012 as an Exploration Geologist. He then went on to work as a Consulting Geologist at Avocet Mining Sarl, Inata, Burkina Faso and later at Taruga Gold, Dolbel, Niger.

In January 2013, Mr. Nuakoh enrolled at the North Carolina A & T State University, Greensboro, USA, to pursue a Master of Science degree in Computer Science. He has played roles as both Teaching Assistant and Research Assistant. His research in Information Assurance has led to the publication of two conference papers and is focusing on publishing a journal article. Secure software engineering and software security are some of his current research interests.

Dedication

This work is dedicated to my mom Mrs. Comfort Nkrumah for her help and guidance through life. My dad, Mr. Philip Nuakoh, for his support and encouragement. My siblings, Kyei, Ebenezer, Kwabena, Evelyn and Serwaa for their encouragements and moral support. Love you guys. God bless you!

Acknowledgments

I would like to first and foremost acknowledge God for His protection and guidance in my life. I appreciate the support of my family, my siblings, for pushing me this far, my dad, for providing financial assistance, and my mom, for always teaching me never to give up. A special thank you also goes to my professor, Dr. Dorothy Yuan for her guidance, criticisms and financial support to make my academic journey a success. My sincere gratitude goes to my committee members, Dr. Anna Huiming Yu and Dr. Justin Zhan whose efforts and advice have helped towards the completion of this thesis. I appreciate all the faculty of the department and the staff of the university for helping nurture my whole well-being not just my education-wise, but an individual as a whole, especially, Dr. Kevin Bryant, Pr. Shearon Brown, Dr. Torres, Dr. Jinsheng Xu and Dr. Kenneth Williams of the computer science department, for teaching me and all the professors that did not teach me, Ms. Rosemary Williams for her warm smiles and tremendous work in information dissemination, Ms. Loreatha Graves and the staff of ISSO, Ms. Anita Sanders and the staff of the School of Graduate Studies, not forgetting Mr. Zeb Talley and the staff of the Office of Career Services for all the careering counselling. Last but not the least, a very special thank you goes to Dr. Gerry Dozier for making pursuit of my master's degree a successful run. Without your efforts, I wouldn't be where I am.

Table of Contents

List of Figures	ix
List of Tables	x
Abstract	1
CHAPTER 1 Introduction.....	2
CHAPTER 2 Literature Review	4
2.1 Attack Patterns.....	4
2.2 Use and Abuse/Misuse Cases	6
2.3 Microsoft SDL Threat Modelling	7
2.4 A Tool for Retrieving Attack Patterns (TrAP)	10
CHAPTER 3 The Methodology for Using CAPEC Attack Patterns to Develop Abuse Cases....	13
3.1 Methodology of Developing Abuse Cases Using CAPEC Attack Patterns	13
3.2 Illustrating the Methodology with an Example	16
3.2.1 Use Cases for Doctor.....	16
3.2.2 Abuse Cases for Doctor.....	17
3.2.3 Threat Modelling	17
3.2.4 Generate Keywords	20
3.2.5 Searching Relevant Attack Patterns using TrAP (Tool for Retrieving Attack Patterns)	20
3.2.6 Selecting Attack Patterns for extending an Abuse Case	22
3.2.7 Extending Abuse Cases using selected attack patterns	23
CHAPTER 4 Developing Abuse Cases for a Hospital Information System	28
4.1 Using SDL Threat Modelling to Generate Keywords for Selecting Attack Patterns	29
4.2 Searching Attack Patterns using Keywords.....	30
4.3 Selecting Attack Patterns for Extending the Brainstormed Abuse Cases.....	32

4.4 Extending Abuse Cases with selected Attack Patterns	33
4.4.1 Extending “Intercept and Analyze Data” Abuse Case with the selected Attack Patterns	33
4.4.1.1 Abuse Case: “Using Man-in-the-Middle Attack to Intercept and Analyze Data”	34
4.4.1.2 Abuse Case: “Intercept and Analyze HTTP Cookies”	35
4.4.2 Extending “Impersonate Doctor” Abuse Case with the selected Attack Patterns .	37
4.4.2.1 Abuse Case: “Spoof Doctor through Session Exploitation”	38
4.4.2.2 Abuse Case: “Spoof Doctor through Password Brute Forcing”	39
4.4.2.3 Abuse Case: “Spoof Doctor through Phishing”	40
4.4.3 Extending “Change Doctor’s Findings” Abuse Case with the selected Attack Patterns	42
4.4.3.1 Abuse Case: “Change Findings Using SQL Injection”	42
4.4.3.2 Abuse Case: “Change Findings through Path Traversal”	45
4.4.3.3 Abuse Case: “Change Findings through File Inclusion”	47
4.5 Finding New Abuse Cases	50
CHAPTER 5 Conclusion and Future Research	52
References	55
Appendix	58

List of Figures

Figure 1. An Example Data Flow Diagram (DFD).....	8
Figure 2. Interface of TrAP Showing Attack Patterns Retrieved under Spoofing Threat	11
Figure 3. Using Keyword “session” to Retrieve Attack Patterns in TrAP	12
Figure 4. The Methodology for Developing Abuse Cases Using Attack Patterns	14
Figure 5. Doctor use & misuse case model in HIS	18
Figure 6. DFD for HIS	18
Figure 7. MSDL Interface Showing Threats and Questions Generated By Tool for Each Threat	19
Figure 8. Closer Look at Questions Generated by MSDL Threat Modeling Tool for Spoofing Threat of the Findings Process.....	19
Figure 9. Child Abuse Cases revealed after Extending Brainstormed Ones	50
Figure 10. New Abuse Cases Discovered.....	51

List of Tables

Table 1. Threats Affecting Elements (Hernan et al, 2006).....	10
Table 2. List of Selected Attack Patterns for Spoofing Findings Process using Keywords generated by SDL Tool.....	20
Table 3. Generated Keywords from Questions provided by SDL Tool for each Element and Threats.....	29
Table 4. Objective Mapping of Applicable Attack Patterns from List of Retrieved Attack Patterns to Elements and Threats.....	31
Table 5. Mapping Attack Patterns and Abuse Cases to Applicable Respective Elements	32

Abstract

To engineer secure software, it is imperative to understand attackers' perspectives and approaches. This information has been captured by attack patterns. The Common Attack Patterns Enumeration Classification (CAPEC) repository hosts over 450 attack patterns that contain information about how attacks have been launched against software. Researches have indicated that attack patterns can be utilized for developing secure software; however, there exists no systematic methodology to address this concern. This research proposes a methodology for utilizing CAPEC attack patterns for developing abuse cases at the requirements stage of the secure software development lifecycle (SDLC). In previous research, a tool for retrieving attack patterns (TrAP) was developed to retrieve CAPEC attack patterns according to Microsoft STRIDE threat categories. This tool also features a search function using keywords. The proposed methodology starts with a set of initial abuse cases developed through brainstorming. Microsoft SDL threat modelling tool is then used to identify and rank possible security threats in the system. The SDL tool generates a series of questions for each threat and these questions are used to extract keywords that serve as input to the TrAP tool to retrieve attack patterns relevant to the abuse cases. Keywords can also be system prerequisites or any technology being implemented in the system. From the list of retrieved attack patterns, the most relevant attack patterns are selected and used to extend the initial abuse cases. New abuse cases can also be discovered through this process.

CHAPTER 1

Introduction

To improve the security of computer systems, information, and the cyber space, it is critical to engineer more secure software. Research has shown that the majority of the security defects are due to known software defects. To develop secure and reliable software, software developers must anticipate abnormal behavior. Therefore, software developers need to have the mindset of an attacker (McGraw, 2006). Attack patterns are valuable resources to help software developers to think like an attacker. Attack patterns capture attackers' perspectives and approaches used by attackers to exploit software. They are valuable resources to help software developers to think like an attacker and have the potential to be used in each phase of the secure software development life cycle. However, systematic processes or methods for utilizing existing attack pattern resources for secure software development are needed.

The Common Attack Pattern Enumeration and Classification (CAPEC) initiative sponsored by the Department of Homeland Security and maintained by Cigital hosts over 450 attack patterns along with a comprehensive schema and classification taxonomy. CAPEC however, is not easy to use, since users have to go through the whole list to get the attack pattern they are looking for. This does not make using CAPEC for software development attractive to developers. In our quest to improve the usability of the CAPEC library and make it more user friendly, we proposed a methodology to utilize CAPEC to develop abuse cases.

This research describes our methodology of utilizing attack patterns for extending abuse cases. Pauli & Xu (2005) described a use/misuse case model for Hospital Information System (HIS) in their paper. We would extend their misuse cases using our approach. The most relevant

attack patterns retrieved by the tool for retrieving CAPEC attack patterns would be further refined and used for extending the abuse cases.

This methodology is utilized to develop abuse cases for a Hospital Information System (HIS) described in (Pauli & Xu, 2005). Our case study demonstrates that the proposed methodology could be used to select attack patterns to extend a set of initial abuse cases generated through brainstorming and discover new abuse cases.

The rest of the thesis is organized as follows: Chapter 2 provides a literature review on attack patterns and use and abuse cases, and also introduces Microsoft SDL threat modelling tool and the tool for retrieving attack patterns (TrAP). Chapter 3 discusses the proposed methodology for utilizing attack patterns to develop abuse cases. Chapter 4 describes how our proposed methodology is used for developing abuse cases for HIS. Lastly, chapter 5 concludes the thesis and raises concern for future work.

CHAPTER 2

Literature Review

This chapter introduces the concept of attack patterns and how they are applied to secure software development from various researchers point of view. The chapter also briefly introduces the concept of use and abuse cases. Microsoft SDL threat modelling tool and the tool for retrieving attack patterns (TrAP) are also introduced in this chapter.

2.1 Attack Patterns

Attack patterns generalize attacks employed by attackers on software systems. Though a relatively new concept, attack patterns has received significant research attention in recent times (Moore, Ellison, & Linger, 2001). Sethi and Barnum (2006) also illustrated that attack patterns have the potential to be used in each phase of the SDLC, including requirement gathering, architecture and design, implementation and coding, as well as testing. A number of researches that have focused on attack patterns and how they are related to this work are discussed below.

Hoglund and McGraw (2004) described 49 attack patterns in their book “*Exploiting Software: How to Break Code*”. Sponsored by the Department of Homeland Security, the ongoing Common Attack Pattern Enumeration and Classification (CAPEC) initiative collects a set of publicly available core attack patterns along with a comprehensive schema and classification taxonomy. Currently CAPEC includes over 450 attack patterns contributed by the community.

Sethi and Barnum (2006) illustrated using examples how attack patterns can be utilized in each phase of the SDLC, a systematic process or method of utilizing existing attack patterns to develop secure software is lacking. There is little research on how to use attack patterns such as

CAPEC in the secure software development life cycle. Towards utilizing CAPEC information for secure software development.

Pauli and Engebretson (2008a) proposed a prototype tool that retrieves related CAPEC attack patterns based on system prerequisites user inputs to the tool. The attack patterns and mitigation strategies for the attack patterns presented to the user can be used during system design and implementation. The prerequisites include hardware, operating system, server configuration and programming language.

Gegick and Williams (2005) constructed 53 attack patterns that can be used for identifying security vulnerabilities during software design. These attack patterns were developed based on four existing vulnerability databases. These attack patterns were represented using regular expressions to encapsulate the steps that can be used to attack the software application. These attack patterns were used to identify vulnerabilities via matching a sequence of elements in a system design that permits the sequence of events in the attack pattern to occur. They used attack patterns to identify security vulnerabilities, we map CAPEC attack patterns to STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) to develop abuse cases.

Pauli and Engebretson (2008b) developed an approach for teaching attack patterns based on a hierarchy to present information logically. This hierarchy includes the following levels of abstraction from highest to lowest: vulnerability, attack pattern, exploit, bug and flaw, activation zone, injection vector, payload, and reward. Students were asked to map CAPEC Release 1 to the abstraction levels of this hierarchy. The objective of this work is to assist students to learn and retain information on attack patterns through the mapping process. They mapped attack

patterns to abstraction level for teaching, we mapped attack patterns to STRIDE to retrieve relevant attack patterns from CAPEC to develop abuse case.

Barnum and Sethi (2007) give a general overview of the structure and content of attack patterns and explain how they can be applied in each stage of the secure software development lifecycle. They explained in detail the concept of CAPEC attack patterns with examples to show their usage as well.

Wiesauer and Sametinger (2009) developed a taxonomy for security design patterns using attack patterns. In their taxonomy, they described a criteria for selecting attack patterns based on security requirements. The purpose of the taxonomy was to help users see relevant security design patterns when selecting attack patterns. Their work assigned security design patterns to CAPEC attack patterns and employs the STRIDE model to group attacks into different categories to classify security patterns, while our work maps CAPEC attack patterns to STRIDE for developing abuse cases.

McGraw (2006) mentioned in his book that attack patterns can be used for developing abuse cases, however, he did not discuss an approach to select and use relevant attack patterns for developing abuse cases. Our work introduces an approach for selecting and utilizing CAPEC attack patterns for developing abuse cases.

2.2 Use and Abuse/Misuse Cases

Use case is a functional description of how a user might interact with a system. McDermott & Fox (1999) classifies it as an abstract episode of interaction between a system and its environment. Use cases are represented by UML diagrams and their descriptions. It tells what the system is intended to be used for, thereby leveraging functional requirements. They describe the system's behavior under normal expected use conditions. However, when the system is used

in an inappropriate way (abused), we need to have an idea of how the system may behave. This introduces the concept of abuse/misuse cases.

Misuse case is a use case from an attacker perspective with the intent to harm the system (Alexander, 2003). A misuse case might harm an actor of the system, a stakeholder or the system itself (McDermott, & Fox, 1999). Misuse cases threaten use cases, it is considered as the opposite form of a use case. Abuse cases serve as a support for developers and elicits security requirements. Countermeasures are developed to mitigate misuse cases in the form of security use cases (Tndel, *et al.* 2010) Some authors maintain a stand that abuse cases and misuse cases are different. In this research however, abuse cases and misuse cases will be used interchangeably and carry the same meaning.

Hope, McGraw & Anton (2004) stated informed brainstorming as the simplest, most practical method for creating abuse cases.

McGraw (2006) discussed a process for developing abuse cases. It takes into account, a set of requirements and standard use cases, and a list of attack patterns. This research focuses on how to find relevant attack patterns from CAPEC, and how to use these attack patterns to extend abuse cases and discover new abuse cases.

2.3 Microsoft SDL Threat Modelling

Software architectural risk analysis refers to the activity of identifying and ranking risks applied to architecture and design-level artifacts. One risk analysis methodology is Microsoft's threat modeling (Meier *et al.* 2003; Hernan, *et al.* 2006). It is the process of hypothesizing potential security threats, evaluating the threats, ranking the threats and suggesting mitigation strategies. It includes the following steps: (1) Identify assets; (2) Create an architecture overview; (3) Decompose the application; (4) Identify, document, and rate the threats. Threats are classified

into six categories: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege (STRIDE). Threats are ranked based on their Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability (DREAD) (Meier *et al.* 2003; Hernan, *et al.* 2006).

Microsoft Security Development Lifecycle Threat Modeling Tool 3.1.8 is a free tool for software developers and architects to identify possible security threats and mitigate potential security risks. It is based on the Microsoft STRIDE framework. The tool requires architectural information to develop data flow diagrams (DFD) and has the ability to analyze model, describe environment and generate several reports. The steps of using Microsoft SDL threat modeling tool 3.1.8 are described below:

(1) Draw data flow diagram. The software system needs to first be decomposed into relevant elements. Each element is analyzed for susceptibility to the threats. Data flow diagrams (DFDs) are used to represent the decomposition of the system. Figure 1 shows an example DFD diagram that the MSDL tool presents to users when the tool is ran. The DFDs comprises of the following elements in relation to the elements of a system: data flows, data stores, processes, external interactors, and trust boundaries (Hernan, *et al.* 2006).

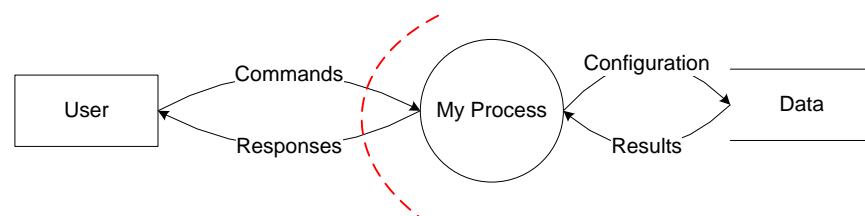


Figure 1. An Example Data Flow Diagram (DFD)

Data flows are represented by arrows and connect interactors, processes and data stores to each other to show how data flows between the elements of the system. The data flows in figure 1 are “Commands”, “Responses”, “Configuration” and “Results”.

Data stores represent storage repositories where data is stored in the system. They are represented by two parallel lines with the name of the data store in-between the lines. In figure 1, the data store is “Data”.

Processes are represented by circles in the DFD diagram. They represent some kind of data processing or system configurations are done. The process in figure 1 is My Process. It takes commands from the User and saves configuration to the Data.

External interactors are human users or non-human actors such as computers. They are outside the system and interact in various ways with the system. They might have to cross certain boundaries to interact with the system. They are represented by rectangular box. User is the external interactor in figure 1. The commands coming from User cross a trust boundary represented by a red dotted arc.

The system may comprise of one or more of the following boundaries: trust, machine, process or other boundaries. These boundaries represent authentication and/or authorization and may restrict access to certain areas in the system without authentication or certain resources without authorization or both.

(2) Analyze model. This step of the Microsoft Threat Modeling process presents all the threat types associated with all the elements of the DFD model to the developer. The elements refer to the data flows, the processes, data store and interactors. For each element and for each threat type to the element, the developer needs to consider a series of questions presented by the tool and describe the threat impact and how to mitigate the threat. Table 1 illustrates a summary of the threats that affect the various elements of the DFD.

Table 1

Threats Affecting Elements (Hernan et al, 2006)

Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Data Flows (DF)		X		X	X	
Data Stores (DS)		X		X	X	
Processes (P)	X	X	X	X	X	X
Interactors (I)	X		X			

(3) Describe environment. During this step, the developers note elements that the system is linked to, especially third-party code, as well as assumptions made by the developers. The developers also describe external security notes and document header information.

(4) Generate reports. This step generates several reports for the developer, such as “Bug Report”, “Recommended Fuzzing”, “Analysis Report”, “Threat Model Report”, etc.

2.4 A Tool for Retrieving Attack Patterns (TrAP)

A tool for categorizing attack patterns based on Microsoft STRIDE framework was developed by mapping attack patterns to STRIDE categories. Various textual values of properties such as Severity, Completeness, Attacker Skills, Likelihood of Exploit and etc. were converted to numerical values to calculate a metric for each attack pattern. The textual values were in the form of high to low or very high to very low rankings. The conversions were as follows: high in a textual context was converted to 3 in the numerical context, medium to 2, and low to 1. Similarly, very high was converted to 5 and very low to 1. The tool calculates a metric from these values and uses it to rank each attack pattern according to each particular STRIDE

category. This ranking puts the attack patterns most relevant to a particular STRIDE category at the top of the list of retrieved patterns.

TrAP is implemented in PHP and MySQL and is web-based for easy access (Figure 2 shows the interface of TrAP displaying retrieved attack patterns under the spoofing category). TrAP is currently running on a localhost server and would be deployed soon to the internet. The retrieved attack patterns are ranked from most relevant to least relevant under each STRIDE category (Yuan *et al.*, 2014).

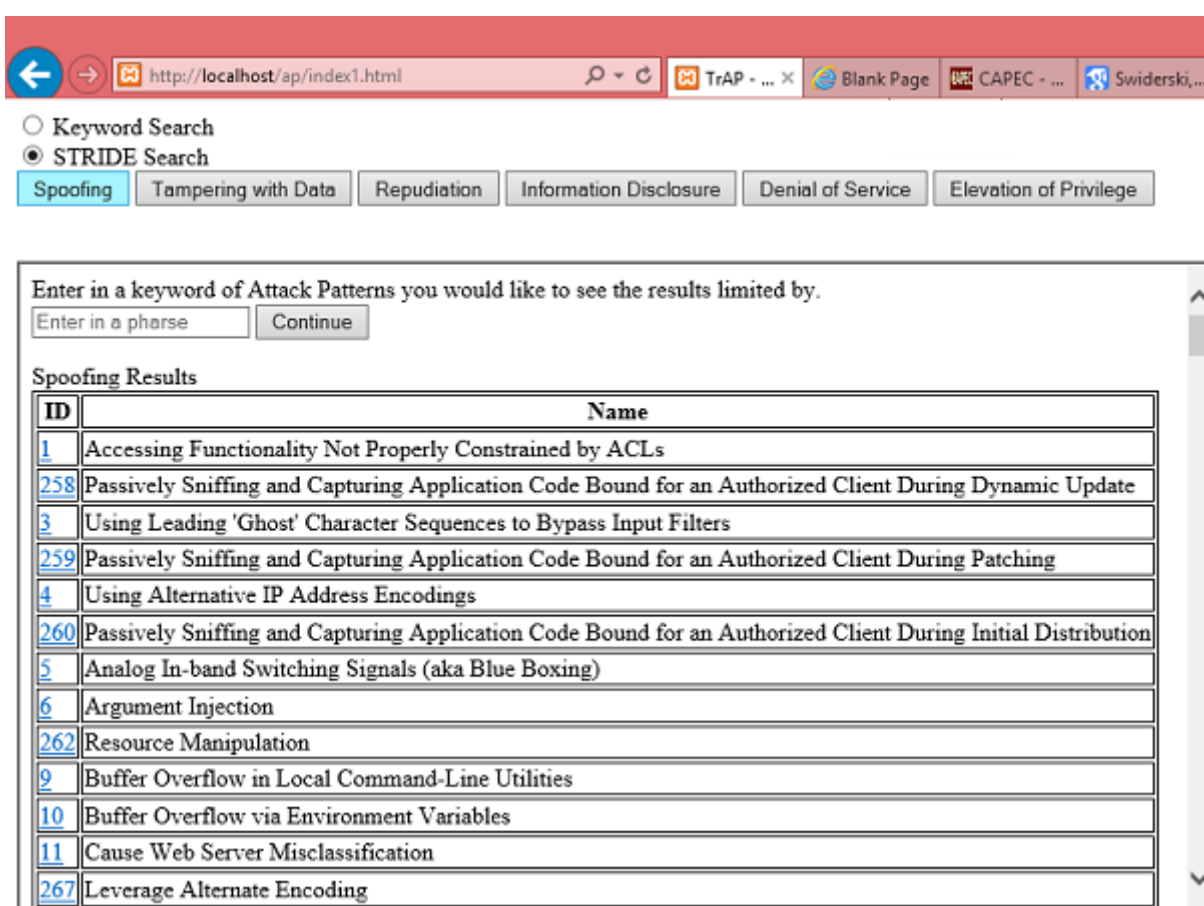


Figure 2. Interface of TrAP Showing Attack Patterns Retrieved under Spoofing Threat

TrAP also features a keyword search that allows easy access to attack patterns containing these type of words. The radio buttons in figure 2 are used to toggle between how a user would like to search for patterns, either from the whole database or from a particular STRIDE type.

Figure 3 shows the result of searching for attack patterns in TrAP using the keyword “session” under the Spoofing threat category.

The screenshot shows a web browser window with the URL `http://localhost/ap/index1.html`. The application interface includes a search section with two radio buttons: Keyword Search and STRIDE Search. Below these are several filter buttons: Spoofing (highlighted in blue), Tampering with Data, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. The main content area displays the title "Spoofing Results with the following keyword: session" above a table of results.

ID	Name
21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
278	Web Services Protocol Manipulation
31	Accessing/Intercepting/Modifying HTTP Cookies
311	Fingerprinting Remote Operating Systems
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery (aka Session Riding)
327	TCP Options Probe
102	Session Sidejacking
103	Clickjacking
107	Cross Site Tracing
114	Authentication Abuse
436	Physical Security Attacks
106	Session Credential Falsification through Prediction

Figure 3. Using Keyword “session” to Retrieve Attack Patterns in TrAP

CHAPTER 3

The Methodology for Using CAPEC Attack Patterns to Develop Abuse Cases

This chapter introduces our methodology for using CAPEC attack patterns to develop abuse cases. The approach uses SDL threat modelling to assist in retrieving CAPEC attack patterns most relevant to the system. These attack patterns are then used to extend a set of initial abuse cases and discover new abuse cases. We use an example to illustrate the steps of this methodology.

3.1 Methodology of Developing Abuse Cases Using CAPEC Attack Patterns

Figure 4 below describes the methodology of developing abuse cases using attack patterns. Developers develop use cases and brainstorm abuse cases from the use cases considering the behaviors of the users. Microsoft SDL threat modeling process is followed to decompose the system architecture into elements and analyze each element for threats. This process generates questions based on threat type per element. Keywords are extracted from the questions generated to search for attack patterns using the tool for retrieving attack patterns (TrAP). Keywords are also generated from the system architecture documentation considering any technology being implemented in the system. Scanning through the list of attack patterns generated by TrAP according to STRIDE should also be considered to select relevant attack patterns that may be missed by keywords search. The retrieved attack patterns are then used to extend the initially brainstormed abuse cases in and can sometimes introduce new abuse cases that should be considered for the system.

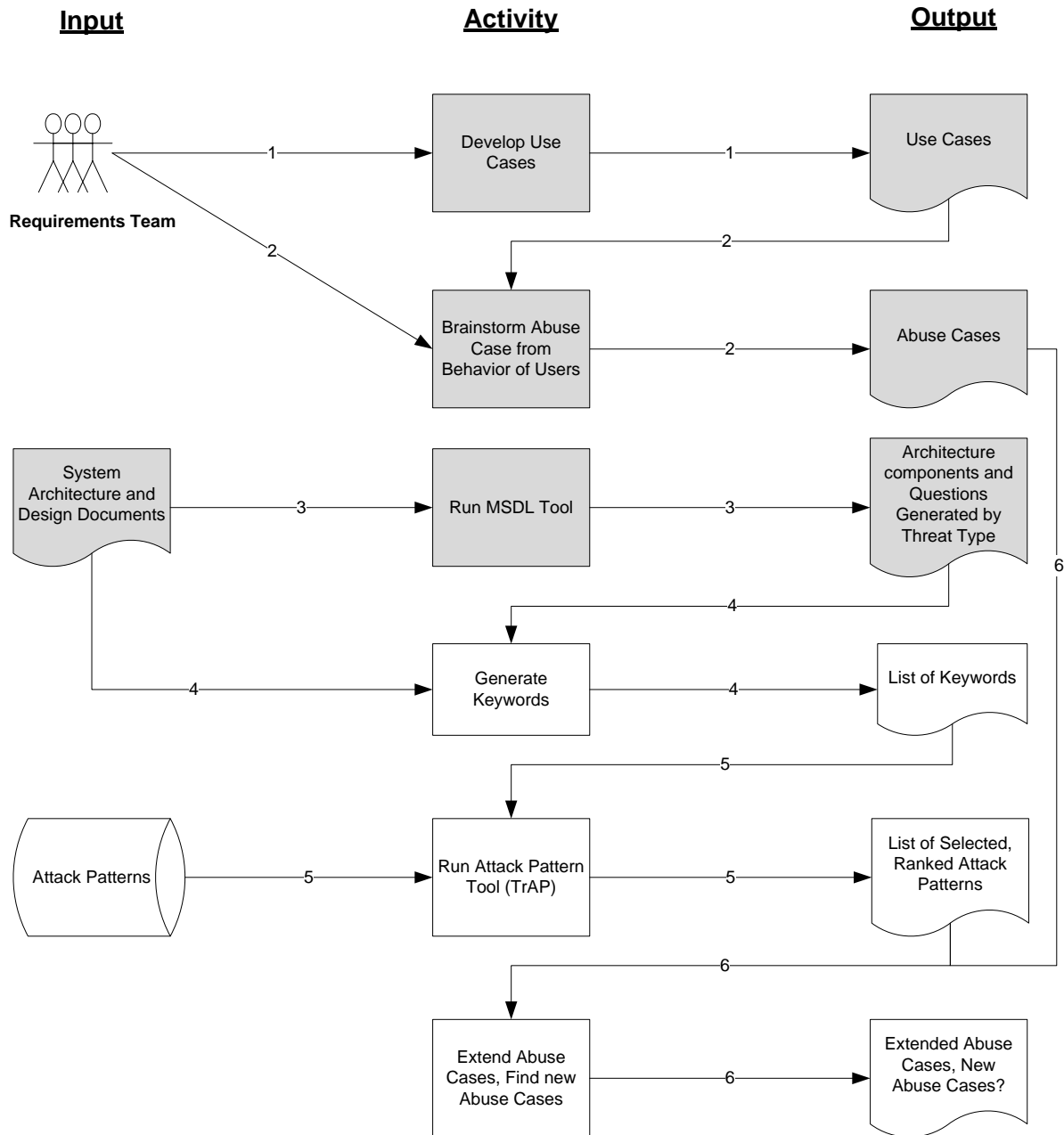


Figure 4. The Methodology for Developing Abuse Cases Using Attack Patterns

The detailed process depicted in Figure 4 is described below: The numbering in Figure 4 is in correspondence with the order of activity flow.

1. System architecture information is collected and used to develop use cases.
2. The system architecture information and the developed use cases are used in a brainstorming process to develop initial abuse cases.

3. The Microsoft SDL threat modelling tool is then ran to analyze threats that pertain to the various elements of the system. This is done by first developing a DFD (data flow diagram) from the system architecture and design documents. The elements of the system are external interactors, processes, data flows and data stores. This process constitutes the architectural risk analysis of the system and the output is a list of threats that pertain to the system elements. The threat types are Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE). The tool generates questions about each threat type for each element of the system. These questions are used to extract keywords for searching attack patterns in TrAP. The elements implement the use cases, which means the elements have a direct relationship with the use cases and a corresponding relationship with the abuse cases that threaten the use cases.
4. Keywords are extracted from the questions generated by the SDL tool and system architecture documentation.
5. These keywords are used to search attack patterns in the Tool for Retrieving Attack Patterns (TrAP) to select most relevant attack patterns by STRIDE category.
6. The selected attack patterns are used to extend the initially brainstormed abuse cases. The retrieved attack patterns are ranked in order of relevance by the TrAP tool. Information used for extending abuse cases is extracted from the “Description” and/or “Example Instance” section of the attack pattern. The result of this exercise is a document of extended abuse cases for the system under development. If there are too many abuse cases, remove the ones with: 1.) Low likelihood of exploit, severity, attacker skills; 2.) Obsolete technology as attack prerequisites and required resources; 3.) High attacker skills and knowledge required. Document any new abuse cases that may be discovered during the process.

The “Solutions and Mitigations” section of the attack patterns in conjunction with the mitigations provided by the threat model analysis can be used to suggest a strong mitigation strategy for the system. However mitigation strategies are out of scope of this research.

3.2 Illustrating the Methodology with an Example

In this section, we use an example Health Information System (HIS) based on work done by Pauli & Xu (2005). In their paper, Pauli & Xu (2005) suggested an approach for design and analysis of secure software systems based on use and misuse cases. They illustrated their approach by performing a case study on a security-intensive hospital information system. We will adopt their hospital information system as an example to illustrate our methodology. For the purpose of clarification, we simplify the use and abuse case model of their example HIS.

The following sections introduce using the steps presented in Figure 4 to illustrate how to develop abuse cases.

3.2.1 Use Cases for Doctor. The HIS has many users which may include secretaries, nurses, doctors, pharmacists, IT personnel, business office personnel and administrative personnel (Pauli & Xu 2005). These user have certain behaviors or interactions closely related to the HIS security: 1) Secretary entering patient information; 2) Nurse entering preliminary appointment information; 3) Doctor entering appointment findings; 4) Doctor transmitting pharmacy orders to the pharmacy; 5) Pharmacist receiving pharmacy order. For the purpose of this work, we consider only the doctor’s role of entering appointment findings which depends on the nurse’s role of entering preliminary appointment information.

The users are given enough permissions to access data they need to execute their job or parts of the system. The doctor is assigned the following tasks in HIS: 1) patient diagnoses; 2) treatment prescription; 3) documenting details of the appointment findings into the HIS after the

completion of an appointment; 4) entering pharmacy orders to be transmitted to the pharmacy; and 5) setting the access levels for the nurses and secretaries. For the purpose of illustrating our methodology, we select one use case based on which abuse cases are developed.

“Enter Appointment Findings” Use Case Description. The doctor logs in to HIS server using a secure browser. The server authenticates the Doctor and opens a session for him. The Doctor enters patient appointment findings and transmits pharmacy orders to the pharmacy, and then logs out.

Abuse/misuse cases could be developed based on the interactions described in the above description.

3.2.2 Abuse Cases for Doctor. An attacker may abuse HIS by changing the doctor’s appointment findings. The attacker might do so by impersonating the doctor or by intercepting data packets and analyzing them for further attacks. Figure 5 below shows the use-misuse case model for the doctor. Use cases are represented by white ovals, black ovals represent abuse cases.

3.2.3 Threat Modelling. Microsoft SDL tool is used for performing threat modelling analysis to find threats pertaining to elements of a system. The system is decomposed into elements to develop a data flow diagram (DFD). The elements are external interactors, data flows, processes, and data stores. Figure 6 represents the DFD diagram of HIS. The diagram depicts interactions between the doctor and the HIS system.

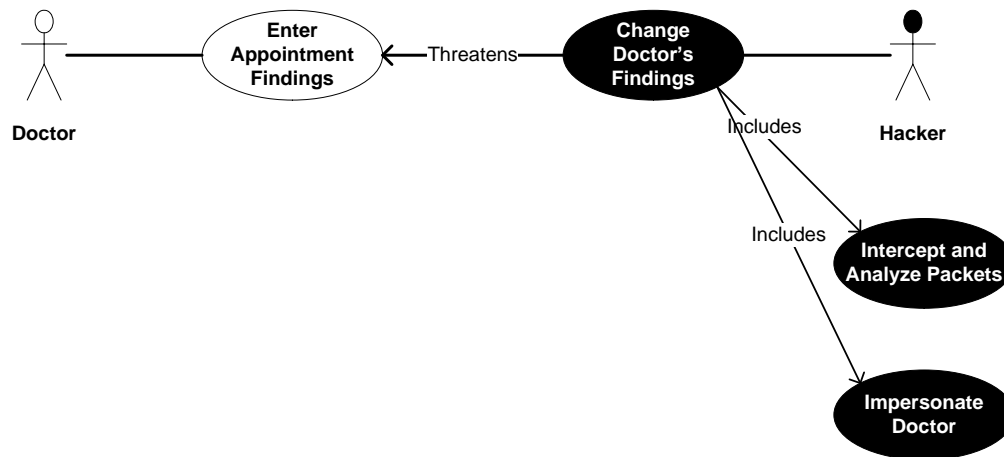


Figure 5. Doctor use & misuse case model in HIS

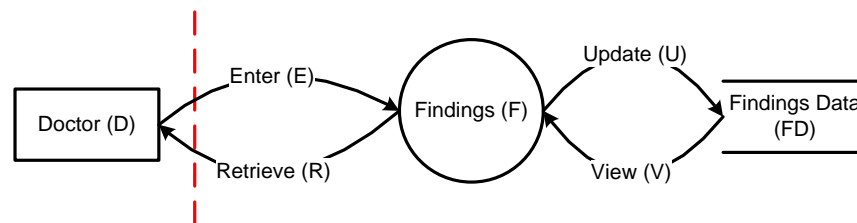


Figure 6. Data Flow Diagram (DFD) for HIS

In Figure 6, the doctor has the ability to enter or retrieve appointment findings after he is successfully authenticated and authorized (red dashed line represents authentication and authorization) by the system. The Findings process takes the data from doctor, processes it and sends to the findings data store. The arrows show the data flow between the doctor, the Findings process and the data store.

Figure 7 shows the interface of the SDL tool where the threats are generated for each element. The “Analyze Model” stage of the SDL generates threats accompanied by questions that address it each threat. The same threats pertains to elements of the same type (as summarized by Table 1 in section 2.3).

Figure 8 is a closer look at the questions posed by the tool for the Spoofing threat under the Findings element.

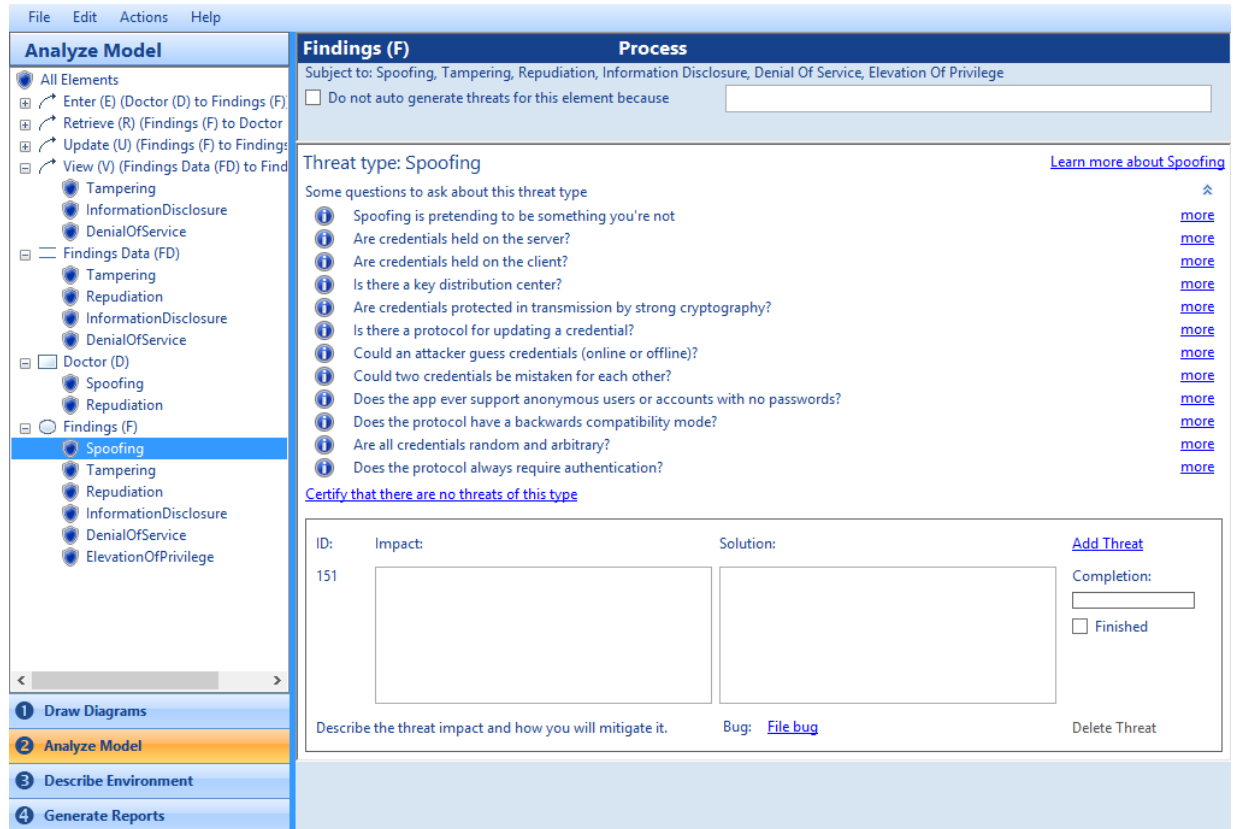


Figure 7. MSDL Interface Showing Threats and Questions Generated By Tool for Each Threat

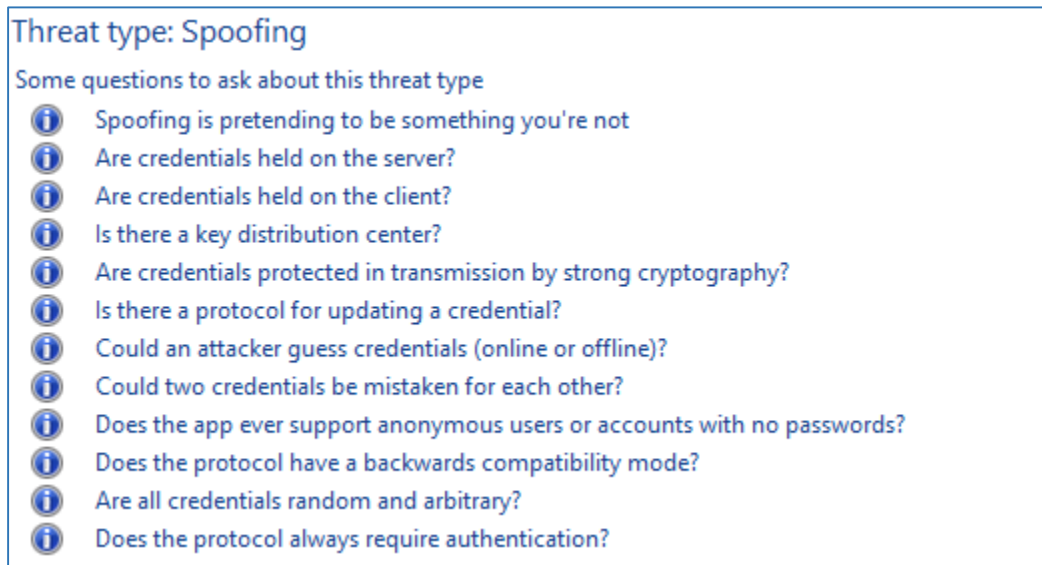


Figure 8. Closer Look at Questions Generated by MSDL Threat Modeling Tool for Spoofing Threat of the Findings Process.

3.2.4 Generate Keywords. The questions presented by the SDL tool serve as a useful resource for extracting keywords used to search for most relevant CAPEC attack patterns. The keywords may range from words to phrases based on the developer's discretion. These extracted keywords are input to the Tool for Retrieving Attack Patterns (TrAP) to search for attack patterns related to the security concern the Microsoft SDL tool tries to address. The most related attack patterns would be used in extending the abuse cases.

For our example, the keywords extracted from figure 8 include: spoofing, credentials, key, cryptography, password, and authentication.

3.2.5 Searching Relevant Attack Patterns using the Tool for Retrieving Attack Patterns (TrAP). TrAP retrieves the most relevant attack patterns from highest to lowest rankings by STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege) category.

Each keyword found from threat modeling was input in the TrAP tool to retrieve a set of CAPEC attack patterns. We combine all the attack patterned retrieved from all the keywords, eliminating duplicates and less relevant ones (those with little or no detailed information). For our example, the resulting attack patterns are listed in Table 2.

Table 2

List of Selected Attack Patterns for Spoofing Findings Process using Keywords generated by SDL Tool

CAPEC ID	Attack Pattern Name
21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
60	Reusing Session IDs (aka Session Replay)

Table 2

Cont.

59	Session Credential Falsification through Prediction
37	Lifting Data Embedded in Client Distributions
196	Session Credential Falsification through Forging
98	Phishing
107	Cross Site Tracing
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)
90	Reflection Attack in Authentication Protocol
70	Try Common(default) Usernames and Passwords
199	Cross-Site Scripting Using Alternate Syntax
69	Target Programs with Elevated Privileges
68	Subvert Code-signing Facilities
97	Cryptanalysis
112	Brute Force
49	Password Brute Forcing
55	Rainbow Table Password Cracking
16	Dictionary-based Password Attack
50	Password Recovery Exploitation
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
114	Authentication Abuse

Table 2

Cont.

11	Cause Web Server Misclassification
136	LDAP Injection
83	XPath Injection
39	Manipulating Opaque Client-based Data Tokens
22	Exploiting Trust in Client (aka Make the Client Invisible)
207	Removing Important Functionality from the Client
5	Analog In-band Switching Signals (aka Blue Boxing)

3.2.6 Selecting Attack Patterns for extending an Abuse Case. In this section, attack patterns that are relevant to an abuse case are selected from the list in Table 2. At this point the table contains attack patterns that are relevant to the specific category of threat, however, the following points needs to be considered in order to select relevant attack patterns for developing a specific abuse case:

1. The motivation of the attack pattern should be similar or match the objective of the abuse case.
2. Any technology implemented by the software should be compared with the prerequisites of the attack pattern to ensure the right attack pattern is selected for developing abuse case for that technology.
3. The resources required by the attack pattern should be compared with the resources required by the abuse case to ensure that the resources required by the attack patterns are not obsolete compared to that required by the abuse case.

4. Attacker skills, likelihood of exploit and severity of the attack pattern should be taken into consideration to determine whether it is suitable for developing abuse case.


For example, for the abuse case “*Impersonate Doctor*”, we select the following attack patterns: “*CAPEC-21 Exploitation of Session Variables, Resource IDs and other Trusted Credentials*”, “*CAPEC-60: Reusing Session IDs (aka Session Replay)*”, “*CAPEC 59: Session Credential Falsification through Prediction*” and “*CAPEC-196: Session Credential Falsification through Forging*”.

3.2.7 Extending Abuse Cases using selected attack patterns. The selected attack pattern could be used to extend the abuse case “*Impersonate Doctor*”. We describe how the attack pattern “*CAPEC-21: Exploitation of Session Variables, Resource IDs and other Trusted Credentials*” is used to extend the abuse case below:

Information displayed in listings 1.1 through 1.3 are excerpts from the attack pattern that are used for developing the abuse case. Detailed information of the attack pattern can be found at: <http://capec.mitre.org/data/definitions/21.html>.

1. **Objective:** To impersonate the doctor and change appointment findings.

Listing 1.1 is an excerpt from the attack pattern and gives a brief idea about the attack pattern and what it is used for. This listing summarizes the attack pattern and contains information for crafting the objective of the abuse case in consideration.

CAPEC-21: Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
Attack Pattern ID: 21	Status: Draft
Abstraction: Standard	Completeness: Complete
▼ Description	
<u>Summary</u>	
<p>Attacks on session IDs and resource IDs take advantage of the fact that some software accepts user input without verifying its authenticity. For example, a message queuing system that allows service requesters to post messages to its queue through an open channel (such as anonymous FTP), authorization is done through checking group or role membership contained in the posted message. However, there is no proof that the message itself, the information in the message (such group or role membership), or indeed the process that wrote the message to the queue are authentic and authorized to do so.</p>	
<p>Many server side processes are vulnerable to these attacks because the server to server communications have not been analyzed from a security perspective or the processes "trust" other systems because they are behind a firewall. In a similar way servers that use easy to guess or spoofable schemes for representing digital identity can also be vulnerable. Such systems frequently use schemes without cryptography and digital signatures (or with broken cryptography). Session IDs may be guessed due to insufficient randomness, poor protection (passed in the clear), lack of integrity (unsigned), or improperly correlation with access control policy enforcement points.</p>	
<p>Exposed configuration and properties files that contain system passwords, database connection strings, and such may also give an attacker an edge to identify these identifiers.</p>	
<p>The net result is that spoofing and impersonation is possible leading to an attacker's ability to break authentication, authorization, and audit controls on the system.</p>	

Listing 1.1. Snippet of the Attack Pattern showing the “Description” field for Crafting Abuse Case Objective.

Listing 1.2 contains information for creating the following sections of the abuse case. They may be directly transferred from the attack pattern.

2. **Prerequisites:** Server software must rely on weak session IDs proof and/or verification schemes
3. **Resource Required:** Ability to deploy software on network. Ability to communicate synchronously or asynchronously with server.
4. **Typical Severity:** High
5. **Likelihood of Exploit:** High
6. **Attacker Skills Level:** Low

7. **Knowledge Required:** To achieve a direct connection with the weak or non-existent server session access control, and pose as an authorized user

<p>▼ Attack Prerequisites</p> <ul style="list-style-type: none"> • Server software must rely on weak session IDs proof and/or verification schemes
<p>▼ Typical Severity</p> <p>High</p>
<p>▼ Typical Likelihood of Exploit</p> <p>Likelihood: High</p>
<p>▼ Attacker Skills or Knowledge Required</p> <p>Skill or Knowledge Level: Low</p> <p>To achieve a direct connection with the weak or non-existent server session access control, and pose as an authorized user</p>
<p>▼ Resources Required</p> <p>Ability to deploy software on network. Ability to communicate synchronously or asynchronously with server</p>

Listing 1.2 Snippet that may be Transferred Directly to the Abuse Case

8. **Crafting Abusive Interaction:** Listing 1.3 the “Attack Execution Flow” part of “Description” is used for crafting the abusive interaction. Abusive interaction is created from part of field of the attack pattern. The “Example Instance” field of an attack patterns with little “Attack Execution Flow” information may be inferred to craft the abusive interaction. Basically, any information the developer deems important should be included in the abusive interaction to build a strong case.

The “Attack Execution Flow” section typically comprises of “Explore”, “Experiment” and “Exploit” sections. The “Explore” section may be used for probing the application. The “Experiment” section might be used for finding out susceptibility to a certain input or query. The “Exploit” section is used for actually carrying an abuse on the system after the exploration and experimentation stages are successful. This is used for developing the abusive interaction section of the abuse case under consideration.

Attack Execution Flow

Explore

1. Survey the application for Indicators of Susceptibility:

Using a variety of methods, until one is found that applies to the target system. The attacker probes for credentials, session tokens, or entry points that bypass credentials altogether.

Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	Spider all available pages	env-Web
2	Attack known bad interfaces	env-Web env-CommProtocol env-ClientServer env-Local

Experiment

1. Fetch samples:

An attacker fetches many samples of a session ID. This may be through legitimate access (logging in, legitimate connections, etc.) or just systematic probing.

Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	An attacker makes many anonymous connections and records the session IDs assigned.	env-Web env-Peer2Peer env-CommProtocol env-ClientServer
2	An attacker makes authorized connections and records the session tokens or credentials issued.	env-Web env-Peer2Peer env-CommProtocol env-ClientServer
3	An attacker gains access to (legitimately or illegitimately) a nearby system (e.g., in the same operations network, DMZ, or local network) and makes a connections from it, attempting to gain the same privileges as a trusted system.	env-Peer2Peer env-CommProtocol env-ClientServer

Exploit

1. Impersonate:

An attacker can use successful experiments to impersonate an authorized user or system

2. Spoofing:

Bad data can be injected into the system by an attacker.

Listing 1.3 Information for Crafting Abusive Interaction section of Abuse Case

Abusive Interaction. The attacker probes HIS (by spidering all available pages) for credentials, session tokens, or entry points that bypass credentials altogether and attacking known bad interfaces. The attacker then fetches many samples of session ids by: 1.) making many anonymous connections and recording the assigned session ids; 2.) making authorized connections and recording session tokens or credentials issued; 3.) An attacker gains access to

(legitimately or illegitimately) a nearby system (e.g., in the same operations network, DMZ, or local network) and makes a connection from it, attempting to gain the same privileges as a trusted HIS system.

An attacker who succeeds in compromising the session keys can impersonate the doctor's session and have the same capabilities as the doctor. There are two main ways for an attacker to exploit session IDs.

1.) A brute force attack: involves an attacker repeatedly attempting to query the system with a spoofed session header in the HTTP request. HIS server might be easily spoofed if it uses a short session ID by trying many possible combinations so the parameters session-ID= 1234 has few possible combinations, and an attacker can retry several hundred or thousand request with little to no issue on their side.

2.) Interception tools such as Wireshark is be used to sniff the wire and pull off any unprotected session identifiers. The attacker then use these variables to access the HIS application.

9. *Post Condition:* Attacker successfully exploits session variables and assumes the identity of a doctor.

CHAPTER 4

Developing Abuse Cases for a Hospital Information System

This chapter illustrates how attack patterns can be used to extend abuse cases for the Hospital Information System (HIS). Attack patterns contain much useful information for developing abuse cases. The extended abuse case includes the following information: objectives, prerequisites, resource required, typical severity, likelihood of exploit, attacker skills or knowledge required, abusive interaction and post condition.

To find out how useful a particular attack pattern is for extending the abuse cases, the following needs to be considered:

1. The objective of the abuse case must correspond to the motivation of the attack pattern.
2. The abuse cases prerequisites in the form of technology being implemented by the system must tally with the prerequisite required to successfully exploit the system using the attack pattern.
3. The resource required might be any form of knowledge or resource an attacker might need or have to be able to exploit the system using the attack pattern. The harder it is for an attacker to find this resource, the harder it is to abuse the system using this attack pattern.
4. The severity, likelihood of exploit, attacker skills are index properties used to check for granularity and trim down (by either combining them or removing duplicates altogether) the list of abuse cases when there seem to be too many attack patterns available for extending a single abuse case.
5. Abusive interaction is the main description and steps required to attack the system. It gives information about how to explore the system for variables needed to exploit it and how to further exploit it when exploration is successful. The “Description” section of an attack

pattern should contain enough information to develop the abuse, if not, information in the “Example Instance” section example could be utilized.

In addition to extending the already brainstormed abuse cases, new abuse cases might be discovered.

The following sections describe the process of developing abuse cases for the HIS. To keep the scope small, we develop abuse cases for only one use case “Enter Appointment Findings”. The process illustrated here starts from Step 3 in Figure 1.

4.1 Using SDL Threat Modelling to Generate Keywords for Selecting Attack Patterns

The SDL tool allows us to develop a data flow diagram (DFD) which is then analyzed for threats. The tool possesses a feature, “Certify that there are no threats of this type”, that allows users to consider if the threats exist in the system or not. To certify that the threat is not present, the user chooses a reason from a list whether the risk is within a trust boundary, mitigated elsewhere in the system or accepted. Table 3 summarizes the threats that pertain to each element and the keywords that were extracted from the questions generated by SDL to address the threat.

Table 3

Generated Keywords from Questions provided by SDL Tool for each Element and Threats

Elements	Threats	Keywords
Doctor	Spoofing	spoofing, credentials, key, cryptography, password, and authentication
	Repudiation	repudiation, digital signature, timestamp, sequence, log

Table 3

Cont.

Data Flows	Tampering	Tamper, bits, dataflow, duplicate, overlap, authenticate, keys, validate, cryptographic, integrity
	Information Disclosure	disclosure, information, authenticate, keys, validate, cryptographic
Findings	Spoofing	spoofing, credentials, key, cryptography, password, and authentication
Process	Elevation of Privilege	Elevation, privilege, alter, execution, code, validate, same origin, LinkDemand, .NET, verification
Findings	Tampering	tamper, alter, data, store, access, resources, datastore, wrap, discard
Data store	Information Disclosure	information, disclosure, access, data, encrypt, channel, recovery, storage

4.2 Searching Attack Patterns using Keywords

TrAP is ran to categorize and rank attack patterns under STRIDE. Under each category, the keywords in Table 3 from section 4.1 above are input one after the other to search for relevant attack patterns. These attack patterns are most related to the threats which pertain to the system as suggested by the SDL tool. The attack patterns are copied in a table and a further selection is done to choose the ones to use for developing the abuse cases. Keywords may also include any technology implemented by the system and can be crafted by the developer to address the security need of the system. In our case, we only limit our keywords to the ones extracted from the questions generated by SDL.

The attack patterns retrieved using keywords in table 3 are listed in the Appendix. The selected attack patterns for developing each abuse case are listed in table 4.

Table 4

Objective Mapping of Applicable Attack Patterns from List of Retrieved Attack Patterns to Elements and Threats.

Elements	Threats	ID	Attack Pattern
Enter & Retrieve Findings Data Flows	Information Disclosure & Tampering	94	Man in the Middle Attack
		31	Accessing/Intercepting/Modifying HTTP Cookies
Doctor	Spoofing	21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
		49	Password Brute Forcing
		98	Phishing
Findings	Elevation of Privilege	180	Exploiting Incorrectly Configured Access Control Security Levels
		1	Accessing Functionality Not Properly Constrained by ACLs
Findings Data store	Information Disclosure & Tampering	66	SQL Injection
		139	Relative Path Traversal

4.3 Selecting Attack Patterns for Extending the Brainstormed Abuse Cases

Table 4 presents a fairly large list of attack patterns. Most of these attack patterns are related to each other with fewer variations. The keywords search retrieve attack patterns that may not be very useful in a particular situation, likewise, the keyword search might miss some attack patterns that are very vital for certain abuse cases. The search may also retrieve attack patterns that show little relevance in terms of information it entails. Manually going through the list of attack patterns retrieved by TrAP to select attack patterns is a best practice after keyword search has been completed. The manual searching process helps capture attack patterns missed by keyword search.

Table 5 maps elements to abuse cases to show elements that stand the highest chance of abuse. The findings process was excluded because it does not have a match with any of the abuse cases. Mapped attack patterns will be used to extend the abuse cases they are mapped to.

Table 5

Mapping Attack Patterns and Abuse Cases to Applicable Respective Elements

Elements	Abuse Case	ID	Attack Pattern
Enter & Retrieve Findings Data Flows (E, R)	Intercept and Analyze Data	94	Man in the Middle Attack
		31	Accessing/Intercepting/Modifying HTTP Cookies
Doctor (D)	Impersonate Doctor	21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
		49	Password Brute Forcing
		98	Phishing

Table 5

Cont.

Findings	–	–	–
Findings Data store (FD)	Change	66	SQL Injection
	Doctor's	139	Relative Path Traversal
	Findings	193	PHP Remote File Inclusion

4.4 Extending Abuse Cases with selected Attack Patterns

After the selection process, we extend our abuse cases using the selected attack patterns. Our abuse case is “*Change Doctor's Findings*” and it includes two intermediary abuse cases: “*Intercept and Analyze Packets*” and “*Impersonate Doctor*”. To change the doctor's findings, one has to be able to assume the role of the doctor. This can be done through either an external attacker spoofing the doctor or causing some level of elevation of privilege if they happen to be users of the HIS system. The attacker might need to intercept data packets on the wire and analyze them as an intermediate step to impersonate the doctor. Our assumption is that a successful impersonation will give the attacker enough privileges to change the doctor's findings.

4.4.1 Extending “Intercept and Analyze Data” Abuse Case with the selected Attack Patterns. The main objective of this abuse varies from disclosing some secured information to tampering the information to perform further attack. For the purpose of this research, we assume that information within the trust boundary is secured from interception. In this case, data coming from the doctor such as credentials, can easily be captured for further analysis. This limits our threats scope to tampering and information disclosure threats. We will select attack patterns

relating to data flow from the doctor into the HIS system. Majority attack patterns inherently exploit the system to disclose sensitive information to assist in further attacks.

Information passing from doctor (client) to HIS server can be disclosed. Before tampering is done, there has to be some form of information disclosure. This information may be relevant for a later attack such as impersonating the doctor from captured credentials.

Information disclosure of this nature is mostly a prerequisite for tampering and is always a sub motivation for tampering.

In reference to table 5, the most relevant attack patterns for tampering are: “CAPEC-94: Man in the Middle Attack” and “CAPEC-31: Accessing/Intercepting/Modifying HTTP Cookies”. These attack patterns would be used to extend the “Intercept and Analyze Packets” abuse case. As a result, two child abuse cases are generated (see figure 9): “Using Man-in-the-Middle to Intercept and Analyze Data” and “Intercept and Analyze HTTP Cookies”.

4.4.1.1 Abuse Case: “Using Man-in-the-Middle Attack to Intercept and Analyze Data”

Objectives. Attacker places himself in the communication channel between server and client to intercept and modify data passing from client to server and vice versa.

Prerequisites: Server software must rely on weak session IDs proof and/or verification schemes

Resource Required: None

Typical Severity: High

Likelihood of Exploit: High

Attacker Skills or Knowledge Required: Level - Low

Abusive Interaction: The attacker probes HIS to determine the nature and mechanism of communication between the client and server looking for opportunities to exploit. He then inserts himself into the communication channel initially acting as a routing proxy between the client and

the server. The attacker may or may not have to use cryptography. He observes, filters or alters passed data of its choosing to gain access to change the appointment findings coming from the doctor to the server.

Post Condition: Attacker places himself between doctor and HIS server and changes appointment findings.

Solutions and Mitigations:

- HIS should use a public key signed by a certificate authority
- HIS communication should be encrypted using cryptography (SSL)
- HIS should Use strong mutual authentication to always fully authenticate both client and server.
- HIS should exchange public keys using a secure channel

4.4.1.2 Abuse Case: “Intercept and Analyze HTTP Cookies”

Objectives: Intercept, modify and forward HTTP cookies to server to gain access to HIS.

Prerequisites: Target server software must be a HTTP daemon that relies on cookies.

Resource Required: Ability to send HTTP request containing cookie to server

Typical Severity: High

Likelihood of Exploit: High

Attacker Skills or Knowledge Required:

Skill Level 1: Low

Knowledge Required 1: To overwrite session cookie data, and submit targeted attacks via HTTP

Skill Level 2: High

Knowledge Required 2: Exploiting a remote buffer overflow generated by attack

Abusive Interaction: The attacker first obtains a copy of the cookie. The attacker may be a legitimate end user wanting to escalate privilege, or could be somebody sniffing on a network to get a copy of HTTP cookies.

Steps:

1. Obtain cookie from local filesystem (e.g. C:\Documents and Settings*\Cookies and C:\Documents and Settings*\Application Data\Mozilla\Firefox\Profiles*\cookies.txt in Windows)
2. Sniff cookie using a network sniffer such as Wireshark
3. Obtain cookie from local memory or filesystem using a utility such as the Firefox Cookie Manager or AnEC Cookie Editor.
4. Steal cookie via a cross-site scripting attack.
5. Guess cookie contents if it contains predictable information.

The attacker may be able to get doctor from the cookie. HIS assumes that cookies are not accessible by end users, and have potentially sensitive information in them.

Steps:

1. If cookie shows any signs of being encoded using a standard scheme such as base64, decode it.
2. Analyze the cookie's contents to determine whether it contains any sensitive information.

The attacker may be able to modify or replace cookies to bypass security controls in the application.

Steps:

1. Modify logical parts of cookie and send it back to server to observe the effects.

2. Modify numeric parts of cookie arithmetically and send it back to server to observe the effects.
3. Modify cookie bitwise and send it back to server to observe the effects.
4. Replace cookie with an older legitimate cookie and send it back to server to observe the effects. This technique would be helpful in cases where the cookie contains a "points balance" for a given user where the points have some value. The user may spend his points and then replace his cookie with an older one to restore his balance.

Post Condition: Attacker successfully subverts security controls on HIS server

Solutions and Mitigations:

- Design: Use input validation for cookies
- Design: Generate and validate MAC (Message Authentication Code) for cookies
- Implementation: Use SSL/TLS to protect cookie in transit
- Implementation: Ensure HIS server implements all relevant security patches, many exploitable buffer overflows are fixed in patches issued for the software.

4.4.2 Extending “Impersonate Doctor” Abuse Case with the selected Attack

Patterns. To impersonate the doctor, an attacker might spoof the login process. The main motivation of this attacker might be limited to assuming identity, however, he can further exploit the system by performing tampering, information disclosure, or denial of service attacks.

In spoofing a doctor, the attacker is trying to assume the identity of a doctor. He does so by launching a number of attacks to first probe the system and then attack the authentication scheme. There are several attack patterns that can be used to build spoofing abuse cases to impersonate a valid user. We select three to extend the “*Impersonate Doctor*” abuse case:

“*CAPEC-21: Exploitation of Session Variables, Resource IDs and other Trusted Credentials*”;

“CAPEC-49: Password Brute Forcing” and “CAPEC-98: Phishing”. As a result, the following child abuse cases were formed (see figure 9): “*Spoof Doctor through Session Exploitation*”, “*Spoof Doctor through Password Brute Forcing*”, and “*Spoof Doctor through Phishing*”.

4.4.2.1 Abuse Case: “*Spoof Doctor through Session Exploitation*”

Objectives: To impersonate the doctor and change appointment findings.

Prerequisites: Server software must rely on weak session IDs proof and/or verification schemes

Resource Required: Ability to deploy software on network. Ability to communicate synchronously or asynchronously with server.

Typical Severity: High

Likelihood of Exploit: High

Attacker Skills Level: Low

Knowledge Required: To achieve a direct connection with the weak or non-existent server session access control, and pose as an authorized user

Abusive Interaction: The attacker probes HIS (by spidering all available pages) for credentials, session tokens, or entry points that bypass credentials altogether and attacking known bad interfaces. The attacker then fetches many samples of session ids by: 1.) making many anonymous connections and recording the assigned session ids; 2.) making authorized connections and recording session tokens or credentials issued; 3.) An attacker gains access (legitimately or illegitimately) to a nearby system (e.g., in the same operations network, DMZ, or local network) and makes a connection from it, attempting to gain the same privileges as a trusted HIS system.

An attacker who succeeds in compromising the session keys can impersonate the doctor's session and have the same capabilities as the doctor. There are two main ways for an attacker to exploit session IDs.

1.) A brute force attack involves an attacker repeatedly attempting to query the system with a spoofed session header in the HTTP request. A web server that uses a short session ID can be easily spoofed by trying many possible combinations so the parameters session-ID= 1234 has few possible combinations, and an attacker can retry several hundred or thousand request with little to no issue on their side.

2.) Interception tools such as Wireshark is be used to sniff the wire and pull off any unprotected session identifiers. The attacker then use these variables to access the HIS application.

Post Condition: Attacker spoofs session ID and assumes doctor's identity to change appointment findings.

4.4.2.2 Abuse Case: "Spoof Doctor through Password Brute Forcing"

Objectives: To impersonate an authorized the doctor

Prerequisites:

- An attacker needs to know the doctor's username.
- The system uses password based authentication as the one factor authentication mechanism.
- An application does not have a password throttling mechanism in place. A good password throttling mechanism will make it almost impossible computationally to brute force a password as it may either lock out the user after a certain number of incorrect attempts or introduce time out periods. Both of these would make a brute force attack impractical.

Resource Required: A powerful enough computer for the job with sufficient CPU, RAM and HD. Exact requirements will depend on the size of the brute force job and the time requirement

for completion. Some brute forcing jobs may require grid or distributed computing (e.g. DES Challenge).

Typical Severity: High

Likelihood of Exploit: Medium

Skills Level: Low

Knowledge Required: A brute force attack is very straightforward. A variety of password cracking tools are widely available.

Abusive Interaction: The attacker tries to determine the password policies of HIS by determining: 1.) the minimum and maximum password lengths allowed; 2.) the formats of allowed passwords (whether they are allowed or required to contain special characters or numbers); 3.) Account lockout policy (a strict account lockout policy will prevent brute force attacks). Given the finite space of possible passwords dictated by the password policy determined in the previous step, the attacker tries all possible passwords for a known doctor's user ID until application/system grants access by: 1.) Manually or automatically entering all possible passwords through HIS's interface. Start with the shortest and simplest possible passwords, because if allowed to do so, most users tend to select such passwords; 2.) Performing an offline dictionary attack or a rainbow table attack against a known password hash.

Post Condition: Attacker determines correct password for a doctor's user ID and obtains access to the HIS system.

4.4.2.3 Abuse Case: "Spoof Doctor through Phishing"

Objectives: Attacker (<https://www.Heath.com>) masquerades as HIS (<https://www.Health.com>) and does business with doctor, gathers credentials and then logs in as the doctor.

Prerequisites:

- An attacker needs to have a way to initiate contact with the victim. Typically that will happen through e-mail.
- An attacker needs to correctly guess the entity (HIS) with which the doctor does business and impersonate it.
- An attacker needs to have a sufficiently compelling call to action to prompt the doctor to take action.
- The replicated website needs to look extremely similar to the original HIS website and the URL used to get to that website needs to look like the real URL of the HIS system.

Resource Required: Some web development tools to put up a fake website.

Typical Severity: Very High

Likelihood of Exploit: High

Attacker Skills: Medium

Abusive Interaction: An attacker creates <https://www.Heatlh.com> which resembles <https://www.Health.com>, the HIS website that he is trying to impersonate. The attacker's website has a login form for the victim to put in his authentication credentials.

Steps:

1. Attacker spiders <http://www.Health.com> to get copies of web pages.
2. He manually saves copies of required web pages from Health.com.
3. Attacker then creates new web pages that have the <https://www.Health.com>'s look and feel, but contain completely new content.

The attacker sends an e-mail to the doctor about a possible login abuse action against HIS website (<https://www.Health.com>) by placing the link https://Heatlh.com/suspicious_activity.php

in the email. Once the doctor clicks on the link included in the e-mail pointing to the attacker's website, he is required to change his password and his credentials are compromised.

Steps:

1. Send the doctor a message from a spoofed legitimate-looking e-mail address that asks him to click on the included link.
2. Place phishing link in post to online forum.

Post Condition: Once the attacker captures these login credentials through phishing, he can leverage this information by logging into HIS to change the doctor's appointment findings.

4.4.3 Extending “Change Doctor’s Findings” Abuse Case with the selected Attack Patterns. “*Impersonate Doctor*” and “*Intercept and Analyze Data*” are sub-abuse cases of “*Change Doctor’s Findings*” abuse case. The main motivation of the attacker to manipulate the findings might first be to gain privileges as doctor or to disclose information, analyze the bits and tamper it.

To tamper with the doctor's findings, an attacker might try to directly get access to the file that the findings data are saved in and then change them. Tampering affects the integrity of the data through forgery. The attack patterns: “*CAPEC-66: SQL Injection*”; “*CAPEC-139: Relative Path Traversal*” and “*CAPEC-193: Remote File Inclusion*” are selected for extending the “*Change Doctor’s Findings*” abuse case. As a result, three child abuse cases are formed (see figure 9): “*Change Findings Using SQL Injection*”, “*Change Findings through Path Traversal*” and “*Change Findings through File Inclusion*”.

4.4.3.1 Abuse Case: “Change Findings Using SQL Injection”

Objectives: To bypass the application completely to talk directly to the database, causing information disclosure and granting ability to modify data in the findings database.

Prerequisites:

- SQL queries used by HIS application to store, retrieve or modify data.
- User-controllable input that is not properly validated by the HIS application as part of SQL queries.

Typical Severity: High

Likelihood of Exploit: Very High

Skill Level: Low

Knowledge Required: It is fairly simple for someone with basic SQL knowledge to perform SQL injection, in general. In certain instances, however, specific knowledge of the database employed may be required.

Abusive Interaction: First take an inventory of the functionality exposed by HIS.

Steps:

1. Spider HIS web sites for all available links
 2. Sniff network communications with HIS application using a utility such as Wireshark.
- Determine the user-controllable input susceptible to injection. For each user-controllable input suspected to be vulnerable to SQL injection, attempt to inject characters that have special meaning in SQL (such as a single quote character, a double quote character, two hyphens, a parenthesis, etc.). The goal is to create a SQL query with an invalid syntax.

Steps:

1. Use web browser to inject input through text fields or through HTTP GET parameters.
2. Use a web application debugging tool such as Tamper Data, TamperIE, WebScarab, etc. to modify HTTP POST parameters, hidden fields, non-freeform fields, etc.
3. Use network-level packet injection tools such as netcat to inject input

4. Use modified client (modified by reverse engineering) to inject input.

After determining that a given input is vulnerable to SQL Injection, hypothesize what the underlying query looks like. Iteratively try to add logic to the query to extract and modify information in the findings database.

Steps:

1. Use public resources such as "SQL Injection Cheat Sheet" at <http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/>, and try different approaches for adding logic to SQL queries.
2. Add logic to query, and use detailed error messages from the server to debug the query. For example, if adding a single quote to a query causes an error message, try : "' OR 1=1; --", or something else that would syntactically complete a hypothesized query. Iteratively refine the query.
3. Use "Blind SQL Injection" techniques to extract information about the database schema.

Post Condition: Attacker achieves goal of unauthorized system access to change doctor's appointment findings.

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.

4.4.3.2 Abuse Case: “Change Findings through Path Traversal”

Objectives: An attacker bypasses input validation on HIS by supplying a specially constructed path utilizing dot and slash characters for the purpose of obtaining access to findings database file and changing doctor’s findings.

Prerequisites: The HIS application must accept a string as user input, fail to sanitize combinations of characters in the input that have a special meaning in the context of path navigation, and insert the user-supplied string into path navigation commands.

Typical Severity: High

Likelihood of Exploit: High

Attacker Skills or Knowledge Required:

Skill Level 1: Low

Knowledge Required 1: To inject the malicious payload in a web page

Skill Level 2: High

Knowledge Required 2: To bypass non trivial filters in the application

Resources Required: None

Abusive Interaction: Using a browser or an automated tool, follow all public links on HIS web site, record all the links found and pick out the URL parameters that may related to access to files.

Steps:

1. Use a spidering tool to follow and record all links. Make special note of any links that include parameters in the URL.
2. Use a proxy tool to record all links visited during a manual traversal of the web application. Make special note of any links that include parameters in the URL. Manual traversal of this type is frequently necessary to identify forms that are GET method forms rather than POST forms.
3. Use a browser to manually explore the website and analyze how it is constructed. Many browsers plug-ins are available to facilitate the analysis or automate the URL discovery.

Possibly using an automated tool, request variations on the identified inputs and send parameters that include variations of payloads.

Steps:

1. Use a list of probe strings as path traversal payload. Different strings may be used for different platforms. Strings contain relative path sequences such as "../".
2. Use a proxy tool to record results of manual input of relative path traversal probes in known URLs.

Inject path traversal syntax into identified vulnerable inputs to cause inappropriate reading, writing or execution of findings data file. A successful attack allows reading HIS directories or files which we would not normally be allowed to read. The attacker could also access data outside the web document root, or include scripts, source code and other kinds of files from external websites. Once there is access to the findings data file, the doctor findings is modified.

Steps:

1. Manipulate findings data file and its path by injecting relative path sequences (e.g. "../").
2. Download findings data file, and modify the file

Post Condition: The attacker accesses the content findings data store and modifies the doctor's appointment findings.

Solutions and Mitigations:

- Design: Input validation. Assume that user inputs are malicious. Utilize strict type, character, and encoding enforcement
- Implementation: Perform input validation for all remote content, including remote and user-generated content.
- Implementation: Validate user input by only accepting known good. Ensure all content that is delivered to client is sanitized against an acceptable content specification -- whitelisting approach.
- Implementation: Prefer working without user input when using file system calls
- Implementation: Use indirect references rather than actual file names.
- Implementation: Use possible permissions on file access when developing and deploying web applications.

4.4.3.3 Abuse Case: "Change Findings through File Inclusion"

Objectives: To control an improperly sanitized "include" or "require" call through an insecurely configured PHP runtime environment to point to findings data store file to load and execute arbitrary code remotely available from HIS to change doctor's findings.

Prerequisites: HIS application server must allow remote files to be included in the "require", "include", etc. PHP directives

Typical Severity: High

Likelihood of Exploit: High

Attacker Skills or Knowledge Required:

Skill Level 1: Low

Knowledge Required 1: To inject the malicious payload in a web page

Skill Level 2: Medium

Knowledge Required 2: To bypass filters in the application

Resources Required: Ability to send HTTP request to a web application. Ability to store PHP scripts on a server

Abusive Interaction: Using a browser or an automated tool, an attacker follows all public links on HIS web site. He records all the links he finds.

Steps:

1. Use a spidering tool to follow and record all links. Make special note of any links that include parameters in the URL.
2. Use a proxy tool to record all links visited during a manual traversal of the web application. Make special note of any links that include parameters in the URL. Manual traversal of this type is frequently necessary to identify forms that are GET method forms rather than POST forms.
3. Use a browser to manually explore the website and analyze how it is constructed. Many browser's plugins are available to facilitate the analysis or automate the URL discovery.

The attack variants make use of a remotely available PHP script that generates a uniquely identifiable output when executed on the target application server. Possibly using an automated tool, request variations on the inputs surveyed before. Send parameters that include variations of

payloads which include a reference to the remote PHP script. Record all the responses from the server that include the output of the execution of remote PHP script.

Steps:

1. Use a list of probe strings to inject in parameters of known URLs. The probe strings are variants of PHP remote file inclusion payloads which include a reference to the attackers' controlled remote PHP script.
2. Use a proxy tool to record results of manual input of remote file inclusion probes in known URLs.

Success in exploiting the vulnerability, enables execution of server-side code within the application. The malicious code has virtual access to the same resources as the HIS application. If required, include shell code in the script to execute commands on the server under the same privileges as the PHP runtime is running with.

Steps:

1. Malicious PHP script that is injected through vectors identified during previous phase and executed by the application server to execute a custom PHP script.

Post Condition: The attacker's script is executed on the HIS server.

Solutions and Mitigations:

- Implementation: Perform input validation for all remote content, including remote and user-generated content
- Implementation: Only allow known files to be included (whitelist)
- Implementation: Make use of indirect references passed in URL parameters instead of file names

- Configuration: Ensure that remote scripts cannot be include in the "include" or "require" PHP directives

Figure 9 provides an overview of all the child abuse cases generated through extending the initial brainstormed abuse cases.

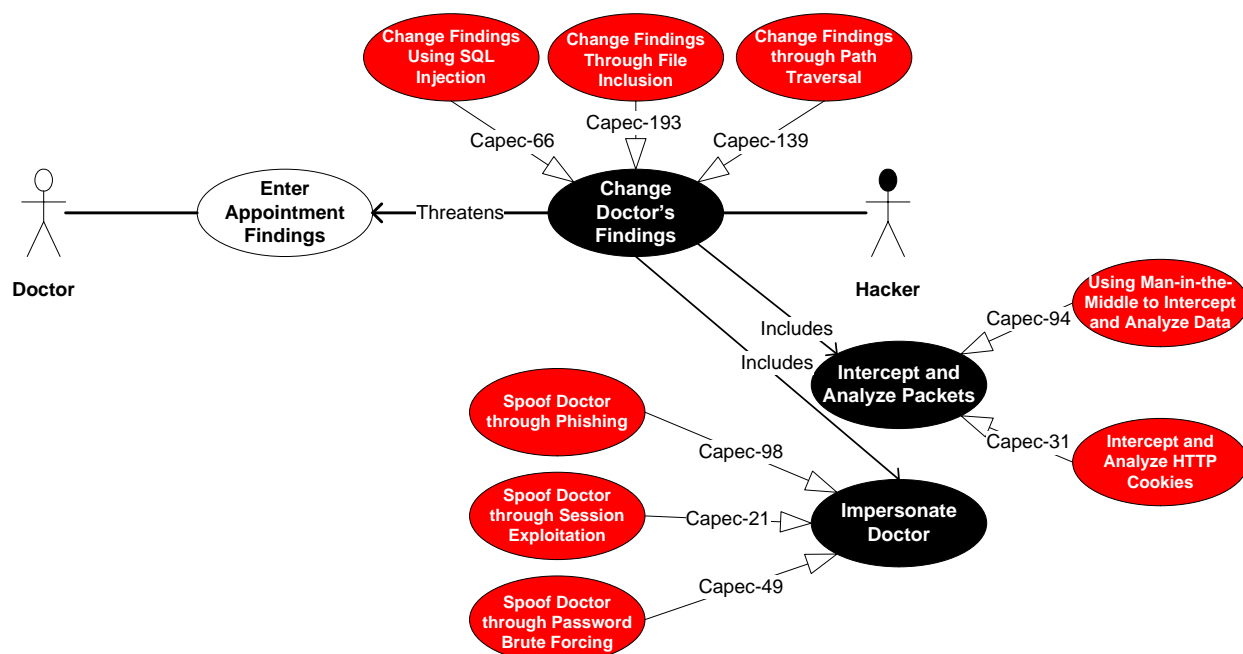


Figure 9. Child Abuse Cases revealed after Extending Brainstormed Ones

4.5 Finding New Abuse Cases

Figure 9 shows abuse cases revealed after extending the brainstormed ones. After extending the abuse cases, we discovered new abuse cases, namely: “*Repudiate Doctor*”, “*Impersonate Findings Process*” and “*Enter Appointment Findings as Nurse*”.

SDL analysis shows us that a doctor can be impersonated and/or repudiated against. The abuse case “*Repudiate Doctor*” is when another doctor is able to enter or change appointment findings s another doctor and totally deny doing so. “*Impersonate Findings Process*” allows an attacker to place himself as a trusted findings process of HIS. He then collects all appointment findings coming from doctor, modifies it and forwards it to the HIS server. A nurse has the role

to enter preliminary appointment information in HIS (Pauli & Xu, 2005). In the abuse case “*Enter Appointment Findings as Nurse*”, the attacker elevates the privileges of the nurse’s process by bypassing the ACL’s confining the findings process to only the doctor users of HIS. If there are ACL’s protecting various elements or the mechanism is weak, the attacker leverages the vulnerability to perform this attack.

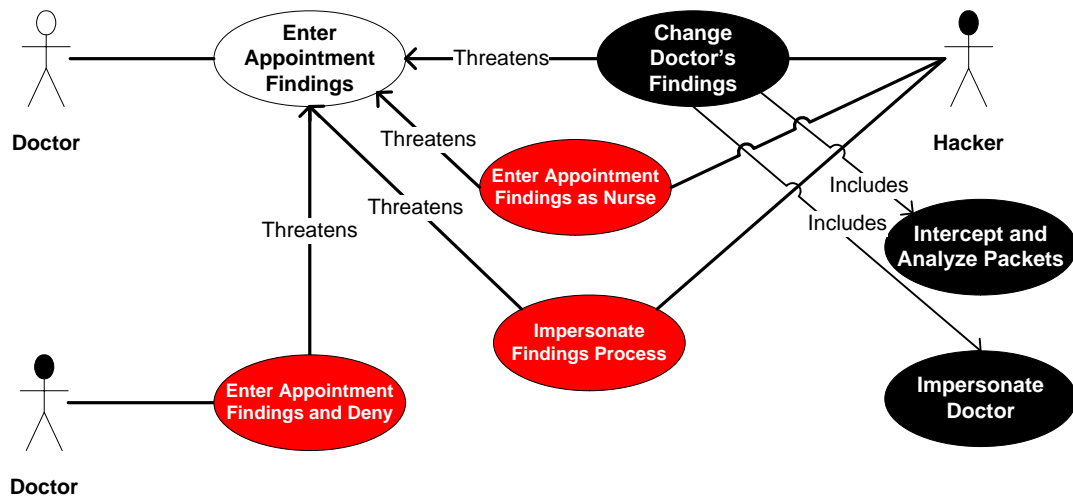


Figure 10. New Abuse Cases Discovered

CHAPTER 5

Conclusion and Future Research

Attack patterns are gaining attention in both research and usage in the secure software development field. This research provides an approach for utilizing CAPEC attack patterns for developing abuse cases. The most useful attack patterns are the most informative and relevant for the system under development. It is not an easy exercise however, to find relevant attack patterns from the CAPEC library which might be useful enough for developing abuse cases. A Tool for Retrieving Attack Patterns (TrAP) was developed to make it easier to retrieve relevant attack patterns from CAPEC library.

An abuse case developer should be able to think about rigorous actions against the software under development. We proposed a systematic method to use relevant CAPEC attack patterns for developing abuse cases.

This method utilizes attack patterns to develop abuse cases using Microsoft SDL threat modelling to aid the selection of most relevant attack patterns. The methodology follows the following process: 1.) develop use and abuse cases from software architecture documentation; 2.) decompose the software into elements and run SDL threat modelling analysis to model threats in the various elements; 3.) extract keywords from questions generated for STRIDE threat types by SDL tool; 4.) use keywords to search for attack patterns in the TrAP tool based on STRIDE threat types; 5.) select relevant attack patterns from the search results and extend abuse cases using information contained in the attack patterns.

After searching for the attack patterns, the list should be checked ensure solid abuse cases are built from the most relevant attack patterns retrieved. Key word search might retrieve related attack patterns or patterns that are related to each other, but might have motivations that vary

slightly from the objectives of the abuse case. This following process should be followed to select relevant attack patterns: 1.) compares the attack motivation of the attack patterns to the objective of the abuse case to make sure they match; 2.) checks for special technology implemented in the software against the attack prerequisites of the attack patterns to make sure attack prerequisites are not obsolete; 3.) checks the resources required by the attack pattern to view its viability to be practical enough for developing abuse cases; 4.) checks the attacker skills needed to exploit the software to see the level of skill and knowledge might be required to exploit software using the pattern; 5.) checks likelihood of exploit and severity of attack to find out how the attack pattern is suited for building a well-grounded abuse case for the software being developed.

The challenges developers might face adopting this methodology is generating keywords for searching attack patterns. Since the same questions might be asked for the same threat pertaining to various elements, this might present a limitation of use of the retrieved/selected attack patterns. To subvert this challenge however, developers should scan through the list of retrieved attack patterns to manually search for relevant attack patterns by STRIDE threat type. The manually searched attack patterns should be combined with the ones searched using the keywords to remove duplicates. Attack patterns should then be selected from the combined list to develop the abuse cases. Also, in the case where many similar attack patterns can be used for developing the same abuse case, deciding between combining information from one or more attack patterns or choosing an attack pattern over the other might pose as a difficult challenge. Users should be familiar with CAPEC attack patterns and Microsoft SDL in order to apply this methodology successfully.

Our future work will focus on using attack patterns for architectural risk analysis, design, and developing test cases.

References

- Alexander, I. (2003). "Misuse cases: Use cases with hostile intent." *Software, IEEE* 20.1: 58-66.
- Barnum, S., & Sethi, Amit. (2007). Attack Patterns as a Knowledge Resource for Building Secure Software. Retrieved January 15, 2013, from http://capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf
- CAPEC: Common Attack Pattern Enumeration and Classification, retrieved from <http://capec.mitre.org/>
- Gegick, M., & Williams, L. (2005, May). Matching attack patterns to security vulnerabilities in software-intensive system designs. In *ACM SIGSOFT Software Engineering Notes* (Vol. 30, No. 4, pp. 1-7). ACM.
- Hernan, S., Lambert S., Ostwald, T., and Shostack, A. (2006). Uncover Security Design Flaws Using The STRIDE Approach, retrieved from <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx#S3>
- Hope, P., McGraw, G., & Anton, A. I. (2004). "Misuse and abuse cases: getting past the positive," *Security & Privacy, IEEE*, vol.2, no.3, pp.90, 92
- Hoglund, G., McGraw, G. (2004). *Exploiting software: how to break code*: Addison-Wesley.
- McDermott, J.; Fox, C. (1999). "Using abuse case models for security requirements analysis," *Computer Security Applications Conference, 1999. (ACSAC '99) Proceedings. 15th Annual* (pp.55, 64). IEEE.
- McGraw, G. (2006). *Software Security: Build Security In*, Addison-Wesley Professionals.
- Meier, J.D., Mackman, A., Dunner, M., Vasireddy, S., Escamilla, R., & Murukan, A. (2003). Chapter 3 Threat Modeling, retrieved from <http://msdn.microsoft.com/en-us/library/ff648644.aspx>
- Microsoft, SDL Threat Modeling Tool 3.1.8, retrieved from <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=2955>

- MITRE. (2012). Common Attack Pattern Enumeration and Classification. Retrieved January 13, 2013, from <http://capec.mitre.org/index.html>
- Moore, A., P., Ellison, R., J., & Linger, R., C. (2001). Attack Modeling for Information Security and Survivability. Retrieved January 15, 2013, from <http://www.sei.cmu.edu/library/abstracts/reports/01tn001.cfm>
- Pauli, J. J., & Engebretson, P. H. (2008a, 7-9 April 2008). *Hierarchy-Driven Approach for Attack Patterns in Software Security Education*. Paper presented at the Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on.
- Pauli, J. J., & Engebretson, P. H. (2008b, 7-9 April 2008). *Towards a Specification Prototype for Hierarchy-Driven Attack Patterns*. Paper presented at the Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on.
- Pauli, J. J., & Xu, D. (2005, April). Misuse case-based design and analysis of secure software architecture. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on* (Vol. 2, pp. 398-403). IEEE.
- Sethi, A., & Barnum, S. (2006). "Attack Pattern Usage", retrieved from <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/588-BSI.html>
- Tndel, I. A., Jensen, J., & Rstad, L. (2010, February). Combining misuse cases with attack trees and security activity models. In *Availability, Reliability, and Security, 2010. ARES'10 International Conference on* (pp. 438-445). IEEE.
- Wiesauer, A., & Sametinger, J. (2009, July). A Security Design Pattern Taxonomy based on Attack Patterns. In *International Joint Conference on e-Business and Telecommunications* (pp. 387-394).

Yuan, X., Nuakoh, E. B., Beal, J. S., & Yu, H. (2014, April). Retrieving relevant CAPEC attack patterns for secure software development. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference* (pp. 33-36). ACM.

Appendix

List of attack patterns retrieved under Spoofing threat using keyword search in TrAP.

CAPEC ID	Attack Pattern Name
21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
60	Reusing Session IDs (aka Session Replay)
59	Session Credential Falsification through Prediction
37	Lifting Data Embedded in Client Distributions
196	Session Credential Falsification through Forging
98	Phishing
107	Cross Site Tracing
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)
90	Reflection Attack in Authentication Protocol
70	Try Common(default) Usernames and Passwords
199	Cross-Site Scripting Using Alternate Syntax
69	Target Programs with Elevated Privileges
68	Subvert Code-signing Facilities
97	Cryptanalysis
112	Brute Force
49	Password Brute Forcing
55	Rainbow Table Password Cracking
16	Dictionary-based Password Attack

List of attack patterns retrieved under Spoofing threat using keyword search in TrAP.

Cont.

50	Password Recovery Exploitation
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
114	Authentication Abuse
11	Cause Web Server Misclassification
136	LDAP Injection
83	XPath Injection
39	Manipulating Opaque Client-based Data Tokens
22	Exploiting Trust in Client (aka Make the Client Invisible)
207	Removing Important Functionality from the Client
5	Analog In-band Switching Signals (aka Blue Boxing)

List of attack patterns retrieved under Elevation of Privilege threat using keyword search in TrAP.

CAPEC ID	Attack Pattern Name
66	SQL Injection
84	XQuery Injection
275	DNS Rebinding
180	Exploiting Incorrectly Configured Access Control Security Levels
77	Manipulating User-Controlled Variables
110	SQL Injection through SOAP Parameter Tampering
1	Accessing Functionality Not Properly Constrained by ACLs

List of attack patterns retrieved under Elevation of Privilege threat using keyword search in

TrAP.

Cont.

10	Buffer Overflow via Environment Variables
104	Cross Zone Scripting
86	Embedding Script (XSS) in HTTP Headers
135	Format String Injection
6	Argument Injection
107	Cross Site Tracing
4	Using Alternative IP Address Encodings
34	HTTP Response Splitting
92	Forced Integer Overflow
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials
163	Spear Phishing
35	Leverage Executable Code in Nonexecutable Files
22	Exploiting Trust in Client (aka Make the Client Invisible)
62	Cross Site Request Forgery (aka Session Riding)
23	File System Function Injection; Content Based
237	Calling Signed Code From Another Language Within A Sandbox Allow This
207	Removing Important Functionality from the Client
65	Passively Sniff and Capture Application Code Bound for Authorized Client

List of attack patterns retrieved under Elevation of Privilege threat using keyword search in TrAP.

Cont.

259	Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Patching
187	Malicious Automated Software Update
177	Create files with the same name as files protected with a higher classification
256	Resource Manipulation
122	Exploitation of Authorization

List of attack patterns retrieved under Information Disclosure and Tampering threats using keyword search in TrAP.

CAPEC ID	Attack Pattern Name
66	SQL Injection
275	DNS Rebinding
51	Poison Web Service Registry
136	LDAP Injection
267	Leverage Alternate Encoding
110	SQL Injection through SOAP Parameter Tampering
87	Forceful Browsing
31	Accessing/Intercepting/Modifying HTTP Cookies
21	Exploitation of Session Variables; Resource IDs and other Trusted Credentials

List of attack patterns retrieved under Information Disclosure and Tampering threats using keyword search in TrAP.

Cont.

7	Blind SQL Injection
37	Lifting Data Embedded in Client Distributions
83	XPath Injection
86	Embedding Script (XSS) in HTTP Headers
6	Argument Injection
101	Server Side Include (SSI) Injection
163	Spear Phishing
196	Session Credential Falsification through Forging
98	Phishing
222	iFrame Overlay
219	XML Routing Detour Attacks
107	Cross Site Tracing
58	Restful Privilege Elevation
91	XSS in IMG Tags
132	Symlink Attack
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)
48	Passing Local Filenames to Functions That Expect a URL
95	WSDL Scanning
12	Choosing a Message/Channel Identifier on a Public/Multicast Channel
11	Cause Web Server Misclassification

List of attack patterns retrieved under Information Disclosure and Tampering threats using keyword search in TrAP.

Cont.

111	JSON Hijacking (aka JavaScript Hijacking)
65	Passively Sniff and Capture Application Code Bound for Authorized Client
259	Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Patching
18	Embedding Scripts in Nonscript Elements
170	Web Server/Application Fingerprinting
215	Fuzzing and observing application log data/errors for application mapping
169	Footprinting
121	Locate and Exploit Test APIs
112	Brute Force