Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2008 Proceedings

Americas Conference on Information Systems (AMCIS)

2008

Threat Modeling the Enterprise

Jeffrey A. Ingalsbe
Ford Motor Company, jingalsb@ford.com

Dan Shoemaker
University of Detroit - Mercy, dshoemaker1@twmi.rr.com

Nancy R. Mead
SEI/CMU, nrm@sei.cmu.edu

Antonio Drommi
University of Detroit - Mercy, drommi@udmercy.edu

Threat Modeling the Enterprise

Jeffrey A. Ingalsbe
Ford Motor Company
jingalsb@ford.com

Dan Shoemaker
University of Detroit Mercy
dshoemaker1@twmi.rr.com

Nancy R. Mead
SEI/CMU
nrm@sei.cmu.edu

Antonio Drommi
University of Detroit Mercy
drommia@udmercy.edu

ABSTRACT

Current threat modeling methodologies and tools are biased toward systems under development. While, organizations whose IT portfolio is made up of a large number of legacy systems, that run on fundamentally different and incongruous platforms and with little or no documentation, are left with few options. Rational, objective analysis of threats to assets and exploitable vulnerabilities requires, the portfolio to be represented in a consistent and understandable way based on a systematic, prescriptive, collaborative process that is usable but not burdensome. This paper describes a way to represent an IT portfolio from a security perspective using UML deployment diagrams and, subsequently, a process for threat modeling within that portfolio. To accomplish this, the UML deployment diagram was extended, a template created, and a process defined.

Keywords

Threat Modeling, DREAD, STRIDE, UML

INTRODUCTION

Current threat modeling methodologies and tools (the few that exist) are biased towards systems under development, in particular, those early in the development lifecycle (although Siderwski [6] does recommend that a second threat model be conducted when a product is "feature complete" which could be when it is in beta release). While this is certainly the ideal time to perform threat modeling, it is a simple fact that the IT portfolio of most fortune 100 companies comprises a large number of legacy systems on fundamentally different and incongruous platforms with little or no documentation.

Organizations that do not perform threat modeling on legacy systems, have several hurdles to overcome. First, they must figure out how to represent complex legacy systems in a consistent and understandable way so that threats and exploitable vulnerabilities can be rationally and objectively analyzed . Second, they must figure out how to perform threat modeling with limited, existing resources. Third, the knowledge of the legacy systems that the organization owns or supports may not be documented. Finally, the organization must figure out how to threat model highly interconnected systems (i.e. systems of systems). This paper describes a way to represent an IT portfolio from a security perspective using UML deployment diagrams in a consistent and understandable way and a prescriptive, collaborative process for threat modeling within that portfolio. To accomplish this, the UML deployment diagram was extended, a template created, and a process defined. This paper discusses these things and future work as well.

PREVIOUS AND RELATED WORK

This work builds on previously published work at Ford Motor Company (Ingalsbe 2004), which represented Ford's IT portfolio from an **infrastructure** perspective using UML deployment diagrams. This work began with that template and expanded it significantly to facilitate threat modeling before progressing to the definition of the process and prototypical tool.

This work was conducted in parallel with other threat modeling work at Ford Motor Company (Ingalsbe, Kunimatsu, Baeten, Mead, 2008) which utilized Microsoft's Threat Analysis and Modeling process and tool to analyze (primarily) systems under development. This work utilized a custom threat modeling process and a prototypical tool to analyze legacy systems possessing little or no documentation.

ON THREAT MODELING

Threat modeling is a generic term that has been given specific meaning by various groups working in the security realm. Federal (NIST 800-53 [7]) and international (ISO27001 [8]) guidelines and standards talk extensively about the identification

and analysis of threats but do not prescribe the "how". Current threat modeling methodologies (which do prescribe the "how") share some common themes. They emphasize systematic, repeatable processes with work products that enhance communication and highlight risk in a clear, easy to understand way for all stakeholders (even those without a security background). They differ on several planes (scope, perspective, formality, and automation). In our opinion, none have adequately addressed legacy systems or systems of systems.

Two threat modeling methodologies were developed at Microsoft. Swiderski and Snyder developed a systemic methodology focusing on data flow and trust boundaries that includes an associated free tool. The Microsoft Application Consulting & Engineering (ACE) team, promotes a methodology (Threat Analysis and Modeling, or TAM) that can be scoped to a portion of the system and focuses on a subset of the application's usage scenarios. The Trike methodology (Saitta, Larcom, and Eddington 2005) emphasizes an application's perspective while encouraging automation. Myagmar (Myagmar et. Al. 2005) presents a methodology that focuses on an attacker's perspective but heavily emphasizes the role threat modeling plays in developing security requirements.

Roadblocks to rolling out Threat Modeling

Personnel

Threat modeling initiatives are just beginning to gain traction in the corporate development community. However that success does not easily transfer over into the legacy world. While systems under development are likely to have dedicated personnel who have the ability and the desire to bring threat modeling competencies into their repertoire, legacy systems (i.e. systems in production) are not. After a system is released to production a different set of personnel with a much different skill set and motivations manage its operation and maintenance. They have existing workloads and (likely) no formal training in threat modeling. Therefore any threat modeling methodology that is to be applied to legacy systems must be conceptually accessible in a short period of time without formal training by the people who would actually be involved and must be minimalistic in terms of its time requirements. Additionally, threat modeling involves both IT and its business customers so any methodology must be optimized for understandability horizontally as well as vertically. The methodology was developed with this in mind.

Knowledge base

To the best of our knowledge, a usable knowledge base related to the production of threat models for legacy systems does not exist. That is, there is no collection of lessons learned about how to perform threat modeling on a large portfolio of (interconnected) legacy systems. However, the MITRE Corporation and the U.S. Department of Homeland Security National Cyber Security Division have teamed up to offer the Common Weakness Enumeration® (a measurable set of software weaknesses) and the Common Vulnerabilities and Exposures ® (a dictionary of publicly known information security vulnerabilities and exposures). Among other things, these serve as an excellent starting point for a knowledge base. Therefore, the team deploying the methodology described in this paper might want to use this prototype knowledge base as a standard point of reference, since it identifies classes of threats, classes of vulnerabilities, classes of systems, and prescribes lessons learned.

Systems of Systems

Current threat modeling methodologies are atomistic in nature. That is, they focus on threat modeling a single application or system as opposed to a set of interconnected systems (or a system of systems). Nevertheless, the truth is that complex global business processes span multiple systems and information flows from system to system to system in an enterprise. This is happening at an ever-increasing rate as consumerization, globalization, and virtualization initiatives take hold. Previously isolated systems are increasingly connected to corporate LANs and the internet in order to facilitate control, reporting, and administration. Handheld and USB devices are routinely connected to employee's laptops and desktops and thus to the corporate LAN. There are numerous examples of security failures centered around the interconnection of such systems. The point is that threat modeling must consider this interconnection.

THE UML

The Unified Modeling Language is a set of notations that provide a standard way to visualize, specify, construct, and communicate the artifacts of software intensive systems (Jacobson, Booch, Rumbaugh 1999). It is considered by most to be the de-facto standard. UML deployment diagrams are used to model the static deployment of a system by showing the configuration of run time nodes and the components that live on them (Jacobson, Booch, Rumbaugh 1999). When

considering legacy systems, the UML and specifically the UML deployment diagram, were the logical choice to represent constituent elements of an IT portfolio.

Extensions

The rich set of modeling concepts and notations that are provided by the UML are sufficient for many software modeling projects. However, in order to meet the requirements of this effort it was necessary to make use of the three built-in extension mechanisms: Constraints, Stereotypes, and Tagged Values. The following sections discuss how each extension mechanism is used in the template.

Stereotypes

Stereotypes allowed us to define a new UML modeling element based on an existing one. Each modeling element in the overall model has a specific purpose. For example, the "Upstream Systems" package is a stereotype of a package. It is a container that will only accept nodes that are upstream systems (i.e. systems outside of the current system that master information for us). Whereas a normal UML package can contain anything, the intent of the stereotyped packages is to restrict containment to a defined purpose. That is, whereas a normal UML package can be placed anywhere and be of any size, the stereotyped packages are intended to enforce a certain real estate apportionment, the attributes of which are protected.

Tagged values

Tagged values allowed us to add new, ad hoc information to model elements (Arlow and Neustadt, 2005). For example, each actor in a deployment diagram has a name that signifies its role. Actors can be customers, administrators, analysts, and etcetera. The tag "Name" tells us what role the actor is playing with respect to this system. For threat modeling, we took tagged values a step further and declared that a tag can be attached to an array of values. For example, presentation nodes, application nodes, and database nodes all have a component stack (the software technologies deployed on that node), an information stack (the information that is created, stored, or processed by that node), a policy stack (the policies that are applicable to that node), and a security stack (the security technologies deployed on that node).

Constraints

Constraints allowed us to extend the semantics of an element by adding new rules. (Arlow and Neustadt, 2005). A *constraint* is a semantic relationship among model elements that specifies conditions and propositions that must be maintained as true; otherwise, the system described by the model is invalid. For example, an actor can only be associated with an endpoint in the diagram. Allowing an actor to have an association with a database node would not make sense because there would be no way for the actor to "interface" directly with a database.

THE TEMPLATE

The Template is a set of rules and guidelines (enforced by UML extensions) for the construction of a deployment diagram and, in turn, a threat model. The template, ultimately, holds both the deployment diagram and the threat model for that deployment diagram. Diagrammatically, the adornments for components were suppressed to help keep the diagrams to one page, although there is no serious impact if they are not. The UML template was developed iteratively and collaboratively with input from a wide range of architects, security personnel, and middle managers.

Previous work (Ingalsbe 2004) showed that a standardized deployment diagram template was useful and well received when used to document the infrastructure portfolio at Ford Motor Company and support its analysis. However, significant modifications to the template had to be made in order for it to facilitate threat modeling. For example, actors are not typically part of a deployment diagram but are required for threat modeling. So, an actor's package and an actor stereotype were added to the template. Figure 1 shows the template. Figure 1 shows the final template.

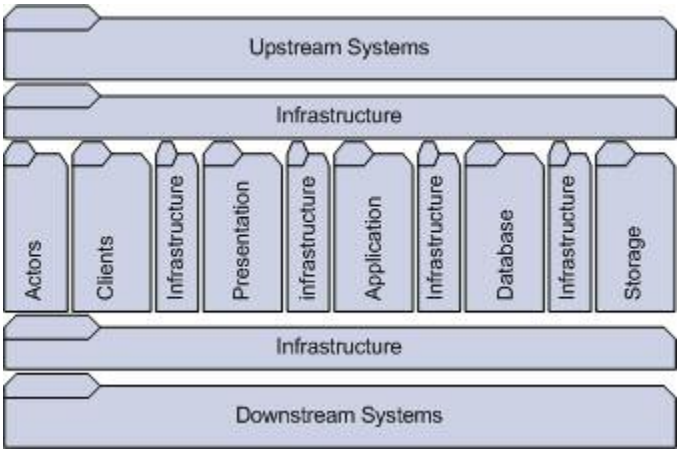

Figure 1. Layout of the Deployment Diagram Template

A good deal of time during the threat modeling process involves understanding the pieces of the system and how they fit together. Only then can the discussion about assets and their value begin. Anything that gets in the way of quickly and accurately understanding a deployed system is wasted time. Therefore, enforcing consistency, not only in real estate apportionment as Figure 1 shows, but also in fonts, adornments, use of color and etcetera facilitates the understanding required for threat modeling. Additionally, if deployment diagrams always look and feel the same then personnel can easily step out of one domain (e.g. product development) and into another domain (e.g. human resources) and perform threat models on very different systems. This eliminates the need for personnel dedicated solely to one business or functional unit.

Enterprise Architecture Framework

Many organizations have an enterprise architecture framework which categorizes the technologies that are used within their enterprise and governs their use. For example, a web browser is one access channel between an application and its users. Whereas, Microsoft Internet Explorer version 7 might be a core technology that is supported by an organization, version 6 might be declining technology that is not supported by the organization. For this work the Federal Enterprise Architecture (FEA) Technical Reference Model was used so that when building the deployment diagrams a standard framework would exist that would allow the modeler to specify the software stack, the security stack, and the information stack for nodes. From a threat modeling perspective, once a threat is discovered that can be shown to capitalize on a given technology a simple query of the deployment diagrams will then show which systems need to be "looked at". Additionally, the systems interconnected with the vulnerable systems will need to be "looked at".

Policy Framework

Many organizations have a policy manual which outlines the policies, procedure, guidelines, and standards that are applicable for a given system. For example, systems which store or transmit personally identifiable information are required to encrypt the data at rest and in transmission. Additionally, the servers on which the data sits must be configured and administered in a prescribed manner. For this work the ISO/IEC 27001 [8] and NIST 800-53 [7] were used so that when building the deployment diagrams a standard model of commonly accepted best practice existed to specify the policy stack for nodes.

THE PROCESS

The process outlined in this paper is prescriptive. While this was our intent, we acknowledge that there is more than a little "art" in the "science" of threat modeling (if indeed you could get away with calling threat modeling a science). Therefore, the initial set of starter questions that we prescribe here should be considered just that. Subsequent discussions should be free and open and "rabbit trails" should not be considered pejorative idioms. The uniqueness of this threat modeling methodology is that it is done while doing something else. That is, in the process of building a deployment diagram you also build a threat model for the system (identify assets, identify threats to those assets, analyze and evaluate the threats, plan the response). At a high level the process for building the deployment diagram and its threat model are as follows: assemble the appropriate application personnel; figure out what the system does and how it is deployed by identifying and specifying actors, endpoints, infrastructure nodes, upstream and downstream systems, presentation nodes, application nodes, database nodes,

and storage nodes and their interconnections. For each of those model elements, ask the set of starter questions documented in subsequent sections. For every threat identified, classify it, analyze it, and determine a response.

Each model element is electronically "attached" to certain electronic forms that allow users to specify the tagged values for that model element. Figure 2 shows the menu of forms available by right-clicking on a presentation node (from the prototype we built). While filling out the forms the team will have discussions about threats by asking starter questions and "diving-in" based on the answers. When threats are identified a form that has been "attached" to the deployment diagram is filled out. The form walks the user through classification and evaluation of the threat using STRIDE (a way of classifying threats or the effects of threats) and DREAD (a way of getting a risk rating by looking at aspects of the threat) (Howard and LeBlanc, 2001). When the deployment diagram is complete, the threat model is complete. The following subsections describe how to build a deployment diagram and hence a threat model by addressing each model element in turn.

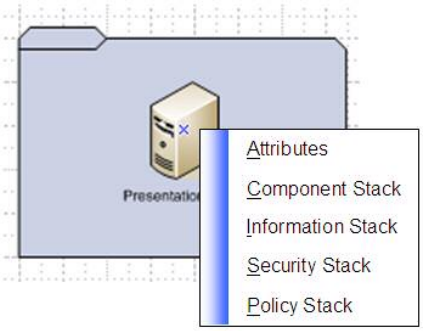

Figure 2. Selecting Forms Attached to a Presentation Node

Actors

Actors are people (or systems) that interact with your system to achieve something of value. The first step in building a deployment diagram for a legacy system and in building the threat model is identifying and specifying actors. Each actor may interact with your system in many ways (use cases). Therefore, the template includes a form for use cases that guides you through asking the following starter questions:

- Why is the actor interacting with your system? (i.e. what is their goal?)
    - How do you know, for sure, who the actor is?
    - What privileges do you give the actor and are they all necessary for this use case?
    - What if the actor was being impersonated? Would you know?
- What trigger motivates the actor to interact with your system?
    - What if they were given a false trigger? Would you know?
- What has to be true about your system the split second before the actor starts?
    - What if those things are not true? Would the actor still be able to proceed? How would you know?
- What has to be true about your system the split second after the actor ends?
    - What if those things were not true? Would you know? How would the system proceed?
- What are the steps the actor takes and what are the system's responses for each step?
    - What if the actor did something incorrectly or differently in one step? Would you know? How would you respond?
- What are some things that must never be allowed?
    - How are you ensuring that these things never happen? If you are making assumptions, can they be validated?
- What are some things that might cause the actor to vary from their prescribed interaction?
    - How do respond?

The outer level bullets tell us how to specify a use case. The inner level bullets are aimed at the identification of threats. For example, an actor (a customer) interacts with the system in order to check their balance because they have received notification of a transaction. Asking the question "What happens if the trigger is false?" drives the discussion into threats that center on loss of confidentiality through activities like phishing. If the actor is supposed to enter personally identifiable information in one step of the use case which in turn is submitted to a database through an sql query then asking the question "What if the actor did something incorrectly or differently in this step?" drives the discussion to threats around confidentiality through things like sql injection.

At the end of this step in the process you will have specified actors, documented use cases, and (possibly) one or more documented threats.

End points or entry points

End points are nodes that allow actors to interact with the system being modeled. End points are "how" actors connect to the system. After asking the question "Who interacts with your system" the next question is "How" does it interact?. Typical end points would include laptops, desktops, browsers, terminals, and etcetera. However, increasingly, the end points are mobile devices. As hardware functionality for mobile devices increases so does the operating system and application functionality. So, what you can "do" with a handheld device approaches what you can "do" with a laptop. Therefore all of the threats that might affect a laptop would apply to a handheld device. Indeed, handheld devices should be discussed at greater length because user awareness of threats to handhelds may not be high. The tagged values for an endpoint include name, type, maximum instances, component stack, security stack, information stack, and policy stack. As these tagged values are being specified, the following starter questions should be asked:

- Who owns the end point, the user or the organization?
- Who is responsible for ensuring that the component stack (i.e. software stack) is at the latest revisions and patch levels? How do you know if it is?
- How do you ensure that only the correct actors are using the correct end points? Would you know if someone was being impersonated? Could you prove it?
- How do you know that the information stack is only seen by authorized actors?
- What would happen if the information stack was not available when needed?
- How do you ensure that the information stack doesn't get corrupted?

At the end of this step in the process you will have the definition of the end points used to interface with a system and (possibly) one or more documented threats.

Infrastructure nodes

Infrastructure nodes allow the representation of things like routers, gateways, firewalls, load balancers, and etcetera. They are often handled by networking groups or vendors. So, discussions can be deferred to those personnel. The important thing to understand is that the infrastructure nodes exist and to document them in the deployment diagram. As you explore the path through which a threat is realized you will need to know about the infrastructure nodes. The following starter questions should be asked:

- Who owns the infrastructure node?
- Who is responsible for ensuring that the component stack (i.e. software stack) is at the latest revisions and patch levels? How do you know if it is?
- Who is responsible for the configuration of the infrastructure nodes?
- Who is responsible for all of the physical security of the infrastructure nodes?

Upstream and downstream system nodes

In the strictest sense, an upstream system is one that is a source of data for the system being modeled and a downstream system is one that is the destination for data mastered by the system being modeled. They are represented as stereotyped nodes. However, in a service oriented architecture an upstream system can be one that provides a service for the current system and, as such, doesn't really master data so much as it does a job. Likewise, a downstream system can be one to which the current system provides a service. The important thing to remember about upstream and downstream systems is that they allow us to understand systems of systems. A simple query can show the interconnectedness of any number of documented

systems. It is important to understand the nature of the connection between two systems. There must be a high water mark which determines how data flowing through multiple systems is treated. The following starter questions should be asked:

- Who is sourcing data or servicing this request?
- How do they do this?
    - What is the mechanism? (e.g. ftp push, web service call, etcetera)
        - What would happen if this mechanism failed? Would you know?
    - What is the frequency? (e.g. daily, ad-hoc, etcetera)
        - What would happen if this frequency was not adhered to? Would you know?
- What is the data that is being sourced (or what is the request that is being serviced)?
    - What is the data's classification? (i.e. how important is the data?)
    - Is the data transmitted in the clear?
- What is the data's classification? (i.e. how important is the data?)
- How do they know that you are a valid requestor of the data or service? What if you were impersonated, would they know?
- How are accounts and passwords handled?

At the end of this step in the process you will have the definition of upstream and downstream systems, an understanding of interconnectedness, and (possibly) one or more documented threats.

Presentation nodes, application nodes, and database nodes

Presentation nodes are those entities that are responsible for presenting the application to clients. Application nodes are responsible for the execution of business logic. Database nodes are responsible for database management. Admittedly, there is some overlap for these three types of nodes. "Application servers" are also "web servers" and may exist on the same node. Mainframes do not have a distinct presentation layer, they present to green screens. However, keeping each node as its own stereotype seemed more conceptually accessible than allowing unstructured grouping. The significant tagged values for presentation, application, and database nodes are component stack, security stack, information stack, and policy stack. As these tagged values are being specified, the following starter questions should be asked:

- Should these nodes be considered assets? Is there a process that is executed that should be considered an asset? Is there any information on the information stack that should be considered an asset? If yes, to any of these, how important are these assets to you?
- What are you presenting to the actors through the endpoints? Is it minimal?
- What business logic are you implementing (i.e. can you talk through it?).
- For each step, what if that step wasn't executed? What if it was executed incorrectly? What if it tried to execute but took a very long time?
- What account / id executes each step? Is it important to protect that account / id?
- What are you storing in the database? Who is responsible for configuration? What account / id is running jobs?

At the end of this step in the process you will have the definition of presentation, application, and database nodes, an understanding of implemented business logic, and (possibly) one or more documented threats.

Storage nodes

Storage nodes are responsible for storage management. For a typical web application, storage nodes might comprise storage components on a SAN. For a typical mainframe application, storage nodes might comprise storage components on a SAN as well. Mirrored storage and backup storage should be specified. Just like infrastructure nodes, there is likely a group or vendor who will help you address threats to the storage.

ON THREATS

It is important to understand that the starter questions that are asked as the model elements are added lay out the threats. The process of building the deployment diagrams and using the starter questions is prescriptive in order to make the process lean and conceptually accessible to those who would implement it. However, it is our experience that the biggest, scariest threats are discovered when someone leans back in their chair, ponders for a moment, and asks the question "What if?" The documentation of the threats is straight forward but deserves some attention. The following three subsections discuss classification, analysis, and response.

Threat Classification

The STRIDE scheme (Howard and LeBlanc, 2001) is used to classify threats. There is some overlap in the scheme (for example, elevation of privilege might result in information disclosure). However it is useful in that it forces discussion about "kinds" of threats. STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

Threat Analysis

After classifying the threat, it must be analyzed. That is, we need to assign a risk rating to the threat. This is done by using DREAD (Howard and LeBlanc, 2001). DREAD stands for Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. By assigning a 0-5 value for each DREAD letter a final risk value is obtained that allows threats to be compared. Despite the fact that a "formula" is used to calculate a risk value, it is important to understand that it is still subjective. A significant effort needs to be made to be consistent.

Threat Response

After classifying the threat, and analyzing it, the business customer needs to decide how they want to respond to the threat. The choices are: avoid (get out of that activity altogether), transfer (let someone else assume the risk, e.g. insurance), and mitigate (reduce the risk by putting controls in place). Implementation of the threat response is not part of the threat model, but it makes perfect sense for the same team to participate in the threat response activities.

CONCLUSIONS AND FURTHER WORK

This paper described a prescriptive threat modeling methodology targeted for use on a portfolio of legacy systems using a customized UML deployment diagram. It is intended to be light weight, conceptually accessible, and able to address systems of systems. This methodology is useful for deployed systems and those systems late in the development lifecycle. It would not be suitable for systems early in the development lifecycle. A prototype has been developed to demonstrate the usefulness of the methodology. Planned next steps include executing the methodology on a set of systems using the prototype.

REFERENCES

1. Arlow, J. and Neustadt, I. (2005) *UML 2 and the Unified Process Second Edition*, Addison-Wesley.
2. Jacobson, I., Booch, G., and Rumbaugh, J. (1999), *"The Unified Modeling Language User Guide"*, Addison-Wesley
3. Saitta, P., Larcom, B., Eddington, M. (2005), "Trike v.1 Methodology Document [Draft],"
4. Swiderski, F. and Snyder, W. (2004), "Threat Modeling," Microsoft Press
5. NIST Special Publication 800-53 (2007), "Recommended Security Controls for Federal Information Systems", National Institute of Standards and Technology U.S. Department of Commerce
6. Office of Management and Budget (2007), "Federal Enterprise Architecture Consolidated Reference Model Document Version 2.3"
7. ISO/IEC 27001:2005 "Information technology -- Security techniques -- Information security management systems – Requirements"
8. Howard, M., and LeBlanc, D., (2001) *"Writing Secure Code (With CD-ROM)",* Microsoft Press