MODEL

Security modeling

Goals of this lab:

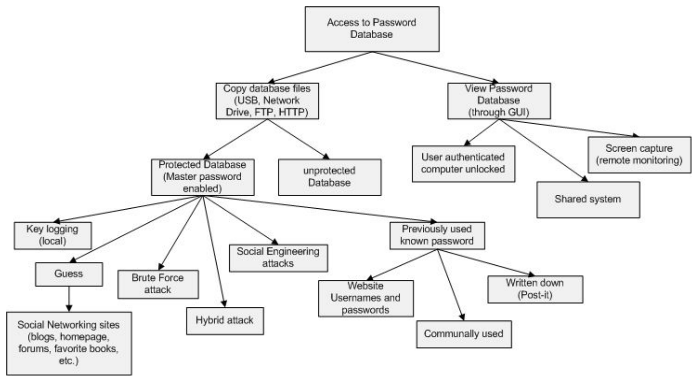
- v Learn how to find information about known vulnerabilities
- v Get experience with performing risk analysis and specifying security requirements.
- v Get hands-on experience analyzing and modeling a vulnerability
- v Learn how to identify possible attacks and misuse cases
- v Get experience with determining how to prevent and mitigate vulnerabilities

Prerequisites: Theory on risk analysis and security modeling

REVISION: 1.0 [2010-10-21]

Table of Contents

MAIN LAB


Access to Password Database
Copy database files (USB, Network Drive, FTP, HTTP)
View Password Database (through GUI)
Protected Database (Master password enabled)
unprotected Database
User authenticated computer unlocked
Screen capture (remote monitoring)
Shared system
Key logging (local)
Guess
Brute Force attack
Social Engineering attacks
Previously used known password
Social Networking sites (blogs, homepage, forums, favorite books, etc.)
Hybrid attack
Website Usernames and passwords
Written down (Post-it)
Communally used

This lab concerns certain aspects of addressing software security during software development. You will work with security modeling techniques that can be used to elicit security requirements, design, determine how to prevent attacks, and determine how to prevent vulnerabilities in the first place. Although such techniques are not in widespread use in software development, all indicators point to techniques like these being used more and more in the future.

Part 1: Vulnerability repositories

Information about known vulnerabilities can be found in several publicly available repositories and databases. Some examples are:

- Security Focus (http://www.securityfocus.com/)
- Open source vulnerability database (http://osvdb.org/)
- National vulnerability database (http://nvd.nist.gov/)
- Common vulnerabilities and exposures list (http://cve.mitre.org/)
- Common weakness enumeration (http://cwe.mitre.org/)

Exercise 1: Vulnerability repositories

1-1 Spend at least ten minutes with each of the databases above to become familiar with their goals, structures, and with the information they provide.

1-2 What are the 20 most recently reported security problems in these websites? Do you see common problems? What type of problems are they?

1-3 Write a one-page report discussing the various repositories, the kind of information they provide, and how they relate to secure software development.

Report: The reports produced in 1-2 and 1-3. You will be assessed on your understanding of the repositories and their relationship to secure software development.

Part 2: Risk analysis and Security Requirements

Assume a ticket purchase and management system in a Cinema. The services provided by the system, among others, are booking, buying and canceling tickets, remote login to personal pages for frequent users to keep track of favorite movies, forums for movie rating and discussions. You will perform risk analysis for this system and will identify the security requirements for the system. The process of risk analysis using RMF is presented in G. McGraw. Risk Management Framework. Cigital.

2005[1]. You will also specify security requirements for the system and for this you need to create misuse cases. Misuse cases and security requirements are described in “Capturing Security Requirements through Misuse Cases” (Sindre, Opdahl, 2001). All papers are available in the labs section of the course homepage.

Exercise 2: Security requirements and misuse/abuse cases

2-1 Using RMF, perform a risk analysis for the system. You are free to make assumptions about the system. Document your assumptions clearly.

2-2 Write 10 security requirements considering the results of the risk analysis.

2-3 Draw misuse case models considering the security requirements you have identified.

Report: The results produced in 2-1 to 2-3.

Part 3: Security Modeling

You will learn about three modeling methods in this part of the lab: vulnerability modeling, attack trees, and abuse/misuse cases. All these methods help developers to build an understanding of vulnerabilities and possible attack scenarios. Such understanding is needed in order to determine how to prevent or mitigate vulnerabilities. Vulnerability modeling is the process of analyzing vulnerabilities and identifying what might have caused them. Attack trees model attacks and threats against a system in a tree structure with the goal (a successful attack) as the root and ways of achieving the goal as the leaf nodes. Abuse/misuse cases are negative scenarios of each leaf node in an attack tree.

The objective of vulnerability modeling is to improve the development process, based on models of vulnerabilities discovered at any point in the software lifecycle. Attack trees and abuse/misuse cases are intended to be used in the requirements and the design phases of the software development process.

Modeling vulnerabilities

You will apply vulnerability modeling, attack trees, and abuse/misuse cases to three buffer overflow vulnerabilities. The process of vulnerability modeling is described in 0[2]. Attack trees are presented in “Attack trees: Modeling security threats” (Schneier, 1999). Abuse/misuse cases are described in “An extended misuse case notation: Including vulnerabilities and the insider threat” (Røstad, 2006). All papers are available in the labs section of the course homepage.

The vulnerabilities you will model are

- CVE-2006-5525, SQL injection vulnerability in PHP-Nuke 7.9 and earlier. The vulnerability allows remote attackers to conduct SQL injection attacks via "/**/UNION " or " UNION/**/" sequences.
- CVE-2008-0227, buffer overflow in yaSSL 1.7.5 and earlier, as used in MySQL allows remote attackers to execute a denial of service attack.
- CVE-2001-0395 is a vulnerability in Lightwave Console Server 3200. This vulnerability results in not disconnecting users after unsuccessful login attempts, which could allow remote attackers to conduct brute force password guessing. (Note: the source code for this vulnerability is not available. You need to model the vulnerability only based on existing information on vulnerability databases.)

[1] On-line: https://buildsecurityin.us-cert.gov/daisy/bsi/250-BSI.html
[2] For further information see “Modeling Software Vulnerabilities With Vulnerability Cause Graphs” (Byers et al., 2006).

Exercise 3: Security modeling

3-1 Examine the technical descriptions for the vulnerability that are available in the on-line vulnerability databases to get an overview of the problem.

3-2 Use manual code inspection techniques, debuggers, and static analysis tools (of your own choice) to analyze the vulnerability. You may also want to explore how the vulnerability was fixed. The result of this analysis should be an in-depth understanding about how the vulnerable code works and precisely what the vulnerability is.

3-3 Construct a VCG for the vulnerability using the method presented in Appendix A.

3-4 Determine how the each of vulnerabilities can be exploited and show your results using attack trees. See "Attack trees: Modeling security threats" for details on attack trees.

3-5 Model misuse cases for the leaf nodes of your attack tree. See "An extended misuse case notation: Including vulnerabilities and the insider threat" for details on misuse cases.

3-6 Based on your misuse case model, can you identify security requirements that could have prevented vulnerabilities from being introduced into the corresponding products in the first place?

3-7 Compare your answer to 3-6, 2-2 and 2-3?

Report: Hand in your analysis results, your VCG(s), attack tree(s) and the misuse cases, and your answer to 3-7. You will be assessed on the quality and completeness of your models.

Based on the models you have created, you should be able to determine how to mitigate (prevent a successful attack) the three vulnerabilities, and how similar vulnerabilities can be prevented in the future, through activities in the software lifecycle.

Exercise 4: Vulnerability mitigation and prevention

4-1 For every cause in the VCGs you created in Exercise 3, identify activities in the software lifecycle that would address the cause.

4-2 Determine how these activities (or a subset of the activities) could be combined to prevent the corresponding vulnerability.

4-3 Use your attack tree to determine how the vulnerability could be mitigated.

4-4 Submit your results from 4-1, 4-2, and 4-3. You will be assessed on the quality and completeness of your answers (have you considered all alternatives for addressing causes, identified a valid subset of activities, and used the attack tree appropriately).

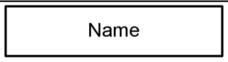
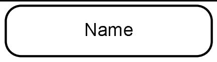
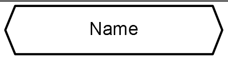
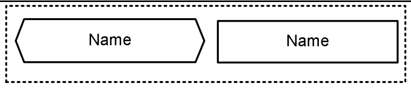
4-5 Based on your results in exercise 2 and 3, write a one page reflection on how you think vulnerabilities similar to those modeled in exercise 3 can be prevented in the ticket purchase and management system.

Appendix A: VCG construction manual

Vulnerability Cause Graphs (VCG): model causes of vulnerabilities and are used as a starting point for detecting and preventing security problems. VCG can be used to both model vulnerability classes and particular vulnerability instances.

Notation:

Name of symbol	Symbol	Explanation
Cause	No symbol	Conditions or events that may lead to vulnerabilities in the software being developed

Simple node		Represents simple causes, not combinations or sequences of causes
Exit node		Represents the vulnerability modelled in VCG
Compound node		Refers to a VCG and facilitates analysis reuse, maintenance of models and improves readability
Conjunction node		Represents the conjunction of two or more nodes
Edge		Represents the relationships between causes, and causes and vulnerabilities

Step-by-step guide:

1. **Initial analysis:** Analyse the vulnerability in order to develop a thorough understanding of its causes. This is done using code review, execution traces, and live debugging and developing a working exploit if possible. Initial analysis is considered complete when you know what types of conditions and/or input would expose the vulnerability.
2. **Create VCG:** Create a base VCG consisting of an exit node, with the name of the vulnerability.
3. **Iteration:** Through a process of iterative refinement analyse any node in the VCG that has not been completely analysed. Analysis of a node consists of the following, which may be performed repeatedly:
    a. **Determine node validity, splitting and conversion:** For each new entered node, determine node validity and analyse if it needs to be split or converted to a compound node: Simple nodes entered into the VCG should always represent simple conditions, not combinations or sequences of conditions. When a complex condition is identified it should be split to several simple nodes and possibly converted to compound or conjunction nodes.
    b. **Find predecessor candidates:** For each of the new entered nodes find their direct predecessors and enter them into the VCG. The predecessors of a node represent conditions that, independently of any other conditions, might cause the condition the node represents to be a concern. The predecessors of the exit node (representing the vulnerability) are its immediate causes. Finding the predecessors of other nodes starts with answering the question “under what circumstances is this cause a concern?”
    c. **Organise predecessor candidates:** Organise the predecessors to present how they are related and how they can lead to the vulnerability. Note that:
        § The predecessor-successor relationship between nodes models how certain conditions cause other conditions to be a concern.
        § Sequences in the graph represent conditions that are ordered by some form of causality.
        § Conjunctions represent conditions that lack any relationship of casual form, but jointly cause some other condition to be a concern.
    d. Repeat steps a, b, and c for all of the nodes in the VCG until no more additions to the VCG can be found.
4. **Optimise the VCG:** When the VCG is complete, it is optimised for reuse and clarity. Optimisation consists of applying graph transformations. For example:

   a. The order of every sequence in the graph is verified to ensure that it is a natural order (e.g. cause-effect or temporal order). Sequences that lack natural order are considered for conversion to conjunction nodes.
   b. If part of the condition represented by one node is the same as part of the condition represented by another node, this indicates that it should be converted to conjunctions and combine identical nodes.
5. **Validate the VCG:** The VCG is validated by a second analyst or team of analysts.
6. Enter the VCG into SVRS and associate it with the vulnerability.
7. **Model compound nodes:** Perform steps 2 and 6 for all compound nodes that have been introduced to the VCG.