FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Master in Informatics and Computing Engineering

Secure programming

Computer Systems Security

Carlos Miguel Correia da Costa - 200903044 - carlos.costa@fe.up.pt

Vítor Amálio Maia Martins Moreira - 200502069 - vitor.moreira@fe.up.pt

October 31, 2013

Index

1 Introduction

In a globalized world where critical systems are controlled by computer software, it is imperative that their implementation is robust enough to endure possible attacks that can compromise the system confidentiality, integrity or availability and may put people's lives at risk.

As such, given the critical role that such systems have in our society, and the adverse effects that an attack could unleash, every critical systems should adhere to several programming best practices, and be certified by independent authorities in order to achieve a stable and secure implementation.

Although computer security is absolutely fundamental in a critical system, every computer software should also take in consideration some of the security best practices in order to make sure their software isn't misused.

As such, our project's main objective is to raise awareness to programmers of the value of well written software implementations, and why they should invest their time in building secure systems.

To achieve that, we will start by presenting the main vulnerabilities that can be exploited in an insecure implementation and what kind of programming and design flaws made then possible.

Then, we will present the most important software techniques and methodologies that can be used to improve the security of a system and what kind of tools could help in the implementation and testing of these secure systems.

2 Context

This project is being developed for a Computer Security course at FEUP, and as such it will focus in the software engineering methodologies and technics that can be used to build secure systems.

These methodologies take into account that every complex system implementation is bound to have vulnerabilities, but as programmers, we should adhere to best practices and systems designs in order to try to avoid then.

To be clear about what is a software vulnerability, let's quote the definition given by IETF in the RFC 2828:

Computer security vulnerability:

"*A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.*"

3 Motivation

The main motivation of our project is to deepen our knowledge in computer security, and to try to raise awareness amongst our colleagues to the importance of well written implementations when building secure systems and how that could avoid the misuse of software they may build in the future.

4 Project specification

4.1 Main vulnerabilities categories

In the next sections we will present the main categories of vulnerabilities and a list of specific exploits that can be used in each of them.

4.1.1 Memory safety

Memory vulnerabilities can be used to change the content of programs variables, or affect the availability of the system by making it crash or have unpredictable behavior.

1) Buffer overflows
    a. Heap overflows
    b. Stack buffer overflows
2) Integer overflows
3) Heap feng shui
4) Heap spaying
5) JIT spaying
6) Return to libc attack
7) Dynamic memory errors
    a. Dangling pointers
    b. Double frees
    c. Invalid frees
    d. Null pointer accesses
8) Uninitialized variables
    a. Wild pointers
9) Out of memory
    a. Stack overflow
    b. Allocation failures

4.1.2 Race conditions

Race conditions can be used to the bypass some authentication system or to access protected resources.

1) Time of check to time of use
2) Symlink race

4.1.3 Input validation

Improper input validation can lead to unauthorized database access and modification and can also be used to inject code that can lead to remote system access or data sniffing.

1) Format string attacks
2) Code injections
3) Frame injection
4) Email injections
5) SQL injections
6) HTTP header injections
7) HTTP response splitting
8) Remote file inclusion
9) Directory traversal attack
10) Cross-application scripting (CAS)

4.1.4 Privilege escalation

Privilege escalation exploits can be used to gain unauthorized access to protected resources.

1) Shatter attack
2) Cross-zone scripting
3) Cross-site cooking
4) Session hijacking
5) Session fixation

4.1.5 Browser exploits

Browsers exploits can be used to gain unauthorized access to data from other webpages and use it to bypass authentication protocols or steal data from users.

1) Cross-site scripting (XSS)
2) Cross-site tracing (XST)
3) Cross-site request forgery
4) Drive-by download
5) Click jacking

4.1.6 Hardware vulnerabilities

Some encryption systems rely on hardware to speed up the runtime of their protocols, but this can lead to some vulnerabilities at the hardware level, that can lead to breaks into the system's security.

1) Cold boot attack
2) Smudge attack

4.1.7 Network protocols vulnerabilities

Some vulnerabilities in communication protocols can lead to exploits used to steal information from users or impersonate other webpages.

1) DSN spoofing
2) Email spoofing
3) Phishing
4) FTP bounce attack

4.1.8 Denial of service

Other types of attacks to a system aim to disrupt its availability to the users.

Such is the case of a denial of service attack.

1) DoS
2) DDoS

4.2 Most common programming errors used in exploits

4.2.1 Programming errors

This is the genesis of software exploits. This includes failing to check return values, misusage of software libraries or functions, failing to test corner cases in function calls, failing to check and sanitize input, failing to check memory bounds...

Since this is the root of nearly all software vulnerabilities, we will make a list of common programming errors that are used in exploits, and provide ways to avoid then.

4.2.2 Using compromised libraries

Most software uses some sort of external library to interact with the system or use some functionality. These libraries can have bugs themselves and require the user to know about them and update them when they are found.

Also, when building a secure system, all of the components should be taken into consideration, from the hardware to the application itself. And that includes the proper selection of operating system, frameworks and libraries.

As such, we will give some useful guidelines to use when selecting the environment in which a secure application is going to run.

4.2.3 Bad architecture design

Sometimes the software has no bugs, but was flawed in its conception, allowing the user to bypass the security system by using it in a non-predicted ways.

We intend to showcase some of these flaws and present tools and techniques that can be used to prevent them. We will also provide some examples of good architecture designs to use in secure systems.

4.3 Secure programming methodologies

Every software implementation is susceptible to have flaws, but we can adopt some methodologies that try to minimize then. As such, we will present some software methodologies that can be used to build secure systems.

Some of these include coding standards, norms available for critical systems and secure programming, tools that can speed up development, etc.

4.4 Useful tools to develop secure systems

4.4.1 Static analyzers

Static code analyzers are tools that analyse source code and point out possible flaws.

We will briefly show what kind of bugs can be caught using these tools and what advantages they bring to the developers.

We will also make a list of tools, both free and paid, that can perform these code analyses.

4.4.2 Vulnerability testing tools

The two major sources of bugs in software are failing to check the return value of functions and lack of input validation. Vulnerability testing tools focus on the latter, injecting misconstrued input into computer systems to try and find flaws.

We will briefly show what these tools can do, as well as list which tools, free and paid, are available.

4.4.3 Sandboxes

When testing out software, especially unknown or questionable software, the need to prevent the system from being compromised is paramount. Sandboxes and virtual machines play a major role in these situations.

We intend to summarize the benefits of sandboxing and virtualization on software analysis and testing, such as using virtual machines to test corner case usage scenarios in software and attack endurance in a compromised and infested environment.

5 Project planning

For the remaining time of the project we will continue to research about secure programming technics and exploits.

Our work plan for the remaining weeks is:

1) Describe in detail each exploit presented is section 4.1, and if possible, present an implementation example for each one.
2) Compile a list of common errors being used in exploits, showing how they can be avoided.
3) Make a guide showing what should be used as a platform for a secure system.
4) Assemble a list of good architecture designs that can help in the implementation of a secure system.
5) Research about methodologies that can be used when developing a secure system.
6) Analyze tools that can speedup testing and implementation of secure systems.

6 Useful links

1) http://en.wikipedia.org/wiki/Defensive_programming
2) http://www.codeproject.com/Articles/7904/Defensive-programming
3) http://en.wikipedia.org/wiki/Computer_Security
4) http://en.wikipedia.org/wiki/Computer_insecurity
5) http://en.wikipedia.org/wiki/Vulnerability_(computing)
6) http://en.wikipedia.org/wiki/Exploit_(computer_security)
7) http://en.wikipedia.org/wiki/Attack_(computing)
8) http://en.wikipedia.org/wiki/Computer_virus
9) http://en.wikipedia.org/wiki/Cyber_security_standards
10) http://en.wikipedia.org/wiki/Sandbox_(computer_security)
11) http://en.wikipedia.org/wiki/Security-Enhanced_Linux
12) http://en.wikipedia.org/wiki/AppArmor
13) http://en.wikipedia.org/wiki/Buffer_overflow
14) http://en.wikipedia.org/wiki/Return-to-libc_attack
15) http://en.wikipedia.org/wiki/Stack_buffer_overflow
16) http://en.wikipedia.org/wiki/Integer_overflow
17) http://en.wikipedia.org/wiki/Dangling_pointer
18) http://en.wikipedia.org/wiki/Heap_feng_shui
19) http://en.wikipedia.org/wiki/Heap_overflow
20) http://en.wikipedia.org/wiki/Heap_spraying
21) http://en.wikipedia.org/wiki/Shatter_attack
22) http://en.wikipedia.org/wiki/Code_injection
23) http://en.wikipedia.org/wiki/JIT_spraying
24) http://en.wikipedia.org/wiki/SQL_injection
25) http://en.wikipedia.org/wiki/Cross-application_scripting
26) http://en.wikipedia.org/wiki/Cross-site_scripting
27) http://en.wikipedia.org/wiki/Cross-site_request_forgery
28) http://en.wikipedia.org/wiki/Session_hijacking
29) http://en.wikipedia.org/wiki/Clickjacking
30) http://en.wikipedia.org/wiki/Format_string_attack
31) http://en.wikipedia.org/wiki/Smudge_attack
32) http://en.wikipedia.org/wiki/E-mail_injection
33) http://en.wikipedia.org/wiki/Directory_traversal_attack
34) http://en.wikipedia.org/wiki/DNS_spoofing
35) http://en.wikipedia.org/wiki/HTTP_header_injection
36) http://en.wikipedia.org/wiki/HTTP_response_splitting
37) http://en.wikipedia.org/wiki/Frame_injection
38) http://en.wikipedia.org/wiki/FTP_bounce_attack

39) http://en.wikipedia.org/wiki/In-session_phishing
40) http://en.wikipedia.org/wiki/Symlink_race
41) http://en.wikipedia.org/wiki/Race_condition
42) http://en.wikipedia.org/wiki/Time-of-check-to-time-of-use
43) http://en.wikipedia.org/wiki/Privilege_escalation
44) http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
45) http://www.metasploit.com/
46) https://community.rapid7.com/docs/DOC-2248
47) http://secunia.com/
48) http://sectools.org/