Midterm Review - Part 1

Security Concepts

Vulnerabilities, Threats, Attacks, and Controls

Principle of Easiest Penetration

Vulnerability

Threat

Within threat, we also consider *Method*, *Opportunity*, and *Motive.*

Attack

Control

Types of Attacks

Interception

Interruption

Modification

Fabrication

How do these four types of attack apply to hardware, software, and data?

Security Goals — CIA

Confidentiality

Integrity

Availability

Some add *Non-repudiation* and *Assurance*

Apply the CIA concepts to specific examples.

Case Study: Heartbleed

Describe the vulnerability, threat, attack, and control.

What security goals were violated?

Case Study: The Debian Fiasco

Describe the vulnerability, threat, attack, and control.

What security goals were violated?

Vulnerabilities

Types of Vulnerabilities

Relate classification of attacks (interception, interruption, modification, fabrication) to system assets (hardware, software, and data)

Hardware

Can a *hardware* vulnerability expose data or software to attack?

Data

Can a *data* vulnerability *expose* software or hardware to attack?

Software

Can a *software* vulnerability *expose* hardware or data to attack?

Software Modification

This means more than just changing the bits of an executable file — also includes modification of program execution or causing a program to run in a way that was not intended. Many important vulnerabilities are of this type.

Buffer Overflow Attack (basics)

What is it?

Give an example.

How do you prevent it?

Incomplete Mediation

What is it?
Give an example.

How do you prevent it?

What did you learn from lab 1 about incomplete mediation?

Time-of-check to Time-of-use Errors (or *Race Errors*)

What is it?

Give an example.

How do you prevent it?

Side Channel Attacks

Side channels include: timing, power consumption, RF emanations

What is a side channel attack?

Give an example.

How do you prevent it?

Standards

Why standards? Interoperability, assurance of market share.

More important for security *because it is hard!*

The Internet Organization

Internet Architecture Board

Internet Engineering Steering Group

Internet Engineering Task Force (IETF)

Internet Drafts are proposed standards; Requests for Comment (RFCs) are published standards.

RFCs — not just security, but many Internet standards

Examples:

1883 — IPv6 Specification
2065 — DNS Security Extensions
3711 — Secure Real-time Transport Protocol (SRTP)
4250 - 4254 — Secure Shell (SSH)

National Institutes of Standards and Technology (NIST)

Publish Federal Information Processing Standards (FIPS) and Special Publications (SP)

Examples:

FIPS 186-4 — Digital Signature Standard
FIPS 197 — Advanced Encryption Standard
SP 800-90 — Random Number Generation
SP 800-82 — Industrial Control System Security

International Telecommunications Union (ITU)

"is responsible for studying technical, operating, and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis."

Examples:

X.509 Public Key Certificates
X.800 Security Architecture for Open Systems

Stack-Buffer Overflow

Attacking the in/out package

Why is the in/out package "interesting" from a security perspective?

Owner and permissions

User Input

Basic fuzzing with Python

Find the vulnerability

Which function in the in/out package causes the buffer overflow?
Which C library function should the programmer *not* have used and why? What is an alternative that would have been better?

What does the stack look like (roughly) in the vulnerable function?

Exploitation Challenges

Knowledge of stack frame location

Where *exactly* is the function's return address?

Where *exactly* can we locate malicious code?

String processing

Malicious code must survive string processing

Exploit Components

There are three components of a basic stack-buffer overflow attack. All three components are part of a single string that will be passed to the vulnerable program as user input.

Shellcode (see sample)

What is the purpose of shellcode?

What are the two major constraints facing a shellcode writer?

Return Address

Why might an attacker include multiple copies of the return address?

NOP Sled

What is a NOP?

What is the purpose of the NOP sled (block of NOPs before the shellcode)?

Stack-Buffer Overflow Protection

Buffer overflow recap

Attacker must:

1. Overwrite values on stack
2. Execute the code on the stack.
3. Predict location of code in memory.

Modern systems protect against these things.

Stack Protection

Goal is to prevent attacker from overwriting anything important (esp. return address)

Terminator Canaries

Most stack buffer overflows involve string operations

Canary is one word consisting of four bytes (CR, LF, Null, and -1)

String-based buffer overflow cannot write the Null byte. Why?

Additional code checks for changes in “canary” value when the function returns

Example: GCC stack protector; controlled through command line argument

`-fstack-protector`	Protect some vulnerable functions
`-fstack-protector-all`	Protect all functions
`-fstack-no-stack-protector`	Disable stack protection

Function Arguments	–
Return Address	`0xbffff498`
Saved Frame Pointer	–
Saved Registers	–

Terminator Canary	`0x15ff1200`
⋮	
	`0x00000000`
	`0x00000000`
	`0x00000000`
Start of `char check[64]`	`0x00000000`

Location of a StackGuard Terminator Canary (Architecture Dependent)

Some weaknesses with terminator canaries

May not protect against non-string based overflow attacks

May be defeated if multiple overwrites are possible — one overwrite to modify control information on the stack, a second to fix-up the canary

Random Canaries

Use a pseudo-random value rather than a fixed value

Must store pseudo-random value securely so that the canary can be checked when the function returns

Example: Propolice and StackGuard

Some weaknesses with random canaries

Local variables *may* not be protected

Function arguments *may* not be protected, at least within the vulnerable function

Attacker may be able to retrieve or guess random canary value (low entropy)

Some bits of canary may be derived from “guessable” sources

Stack Execution Protection

Prevent execution of code on the stack

Supported by CPU (some alternatives when there is no CPU support)

On Linux, stack is non-executable by default; controlled by one bit in the ELF executable file header

`-zexecstack` linker option to enable execution of stack

Some weaknesses with stack execution protection

CPU may not support it (older CPUs) or it may be disabled in BIOS
May not be enabled in virtualized environment

Attackers have developed workarounds

Return-to-libc / return-to-system

Attacker knows address of library function, e.g. execve()

Attacker overwrites return address with library function address

Attacker "fixes up" stack to that it (a) has the structure the library function expects, and (2) provides "useful" arguments to the library function

Does all this with a buffer overflow!

Address Space Layout Randomization

Goal is to defeat "classic" buffer overflow as well as return-to-libc

Randomly change the location of library/system functions every time the system is booted.

Randomly change stack location each time a program is executed

Randomly change location of buffers `malloc()`-ed on the heap each time a program is executed

Example: Ubuntu implements ASLR on multiple components

Stack, Libs/MMAP, Exec, Brk, Virtual Dynamically Liked Shared Object (VDSO)

How does ASLR from Project 1 compare to what is provided by Ubuntu? What did you learn about ASLR from the project?

Classification of Malicious Software - Propagation

Viruses

Replicate by modifying other files or programs

Require *user assistance* to replicate

Give an example

Trojan Horse

Masquerades as useful or desirable software, enticing users to install

Includes malicious functionality

Give an example

Worms

Spread *without* injecting code into other applications

Typically spread *without user assistance*

Give an example

Trapdoors (or Backdoors)

Method to obtain access that bypasses usual authentication measures

A type of *Insider Attack*; may be malicious or benign

Example: Ken Thompson's *Reflections on Trusting Trust*. (linked from the Lecture 6 web page)

Logic Bombs

Code created to take destructive action given a specific *trigger*

Another form of *Insider Attack*

Give an example

Viruses in Depth

Virus Lifecycle

- Dormant phase
- Propagation phase
- Triggering phase
- Action phase

Virus Types

- File viruses
  - What are they?
  - Give an example
- Macro viruses
  - What are they?
  - Give an example
- Boot sector viruses
  - What are they?
  - Give an example

Virus Signatures

- What is a virus signature?
- How are they used to defend against viruses?
- Given a ClamAV signature, describe how it is used

Encrypted, Polymorphic, and Metamorphic Viruses

Why would a virus writer encrypt the virus code?

What limitations are there on encryption of the code?

What is a *Polymorphic Virus*?

A *Metamorphic Virus* attempt to defeat signature recognition by re-writing its own code.

Garbage code insertion

Register Use Exchange

Code Block Permutation / Jump Insertion

Code Integration

Worms

Much of what we've said about viruses applies to worms as well; the main difference is that worms can spread without user action.

Example: The Morris Worm, 1988 (http://en.wikipedia.org/wiki/Morris_worm)

Exploited three vulnerabilities to gain access to systems (fingerd, SMTP, password); did not require user action to spread

What design flaw turned this "experiment" into a denial-of-service?

Example: Stuxnet, 2010 (http://en.wikipedia.org/wiki/Stuxnet)

A *zero-day vulnerability* is a vulnerability that was previously unknown to the general public. What was special about Stuxnet with regard to zero-days?

Stuxnet targeted a Siemens Programmable Logic Control (PLC). Why is this a concern? Recall the TED video.

How was Stuxnet able to install device drivers that were trusted by Windows?

Was Stuxnet harmful?