Computer Security

08r. Exam 1 Review

Paul Krzyzanowski

Rutgers University

Fall 2019

Part 1: Introduction

Question 1 (B: 6)

The ***CIA Triad*** is:

(a) A set of security guidelines established by the U.S. Central Intelligence Agency.

(b) A collection of techniques hackers use to break into systems.

(c) Three sets of leaked documents and published on WikiLeaks detailing the CIA's hacking tools.

(d) A model for classifying topics that need to be addressed in computer security.

CIA = Confidentiality, Integrity, Availability

Question 2 (B: 1)

Data confidentiality means that the data:

(a) Is encrypted.
(b) Has personally identifiable information removed.
(c) Cannot be shared without the permission of the owner if it contains personally identifiable information.
(d) Cannot be accessed by unauthorized parties.

(a) Not always necessary

(b, c) These relate to privacy

Question 3 (B: 2)

A system with a large ***attack surface***:

(a) Offers many ways in which an attacker could try to enter the environment.

(b) Has a large number of vulnerabilities.

(c) Is just as likely to be attacked by trusted insiders as well as external attackers.

(d) Uses multiple forms of defenses to detect and prevent attacks.

Attack surface ≠ number of vulnerabilities

It's a measure of the potential for attack

Question 4 (B: 3)

Advanced Persistent Threats (APT) are most likely to be:

(a) Small groups of individuals working alone to avoid detection.
(b) Malicious insiders.
(c) Intelligence agencies.
(d) White hat hackers.

APTs refer to well-funded, highly determined attackers

Question 5 (B: 3)

An ***opportunistic attack*** targets your systems because:

(a) You are a high-value target.
(b) Your organization has a malicious insider.
(c) Attacks from a distance are difficult to trace.
(d) Your systems may have a vulnerability they are prepared to exploit.

- Opportunistic attacks are the opposite of *targeted attacks*
- The attacker picks you because you are convenient
  - Burglarize a house because the front door is open *vs.* because the house contains an original Picasso painting

Question 6 (B: 5)

A **trusted computing base** (TCB) refers to:

(a) All the components of a system that are critical to its security.
(b) A computer system that is only available to trusted users.
(c) Carefully-audited application software that does not interact with non-trusted applications.
(d) Tamper-resistant computing hardware that the software can trust to run correctly..

TCB = collection of all the hardware, firmware, networks, libraries, programs needed to run an application

Part 2: Access control

Question 7 (B: 12)

A ***capability list*** defines:

(a) The operations that various subjects are allowed to perform on an object.
(b) The system calls that a process can call when it is running with root privileges.
(c) The operations that a subject is allowed to perform on various objects.
(d) The full set of system calls that a process is allowed to invoke.

- Access control list (ACL): associated with objects
  - Defines access rights for various subjects
- Capability list: associated with subjects
  - Defines access rights for various objects

Question 8 (B: 7)

Unlike access control lists, POSIX (e.g., UNIX, Linux, FreeBSD) permissions:
(a) Enumerate a list of users who can access an object.
(b) Identify a list of objects along with access permissions for those objects.
(c) Use a fixed amount of space per file to store access permissions.
(d) Allow an administrator to manage group access to objects.

- POSIX file permissions are a restricted form of an access control list
- Each object (file) contains only a set of three subjects: *user, group, other*

Question 9 (B: 8)

Unlike discretionary access control (DAC), ***mandatory access control*** (MAC):
(a) Requires the kernel to check access rights for an object before opening it.
(b) Is configured by administrators, not users.
(c) Organizes users into roles.
(d) Assigns a confidentiality level to each object.

MAC = **mandatory** access control

Users cannot override policies set by administrators

(a) This is done for every form of access control

(c) This is role-based access control

(d) This is the Bell-LaPadula model, one form of MAC

Question 10 (B: 9)

Which access model is most directly implemented with an access matrix to manage read/write access rights?

(a) Bell LaPadula.
(b) Lattice.
(c) Biba.
(d) Type Enforcement.

Type Enforcement is an access control matrix defining access policies of *domains* (groups of users) and *types* (objects)

BLP & Lattice models: read/write access defined based on confidentiality (& compartment) levels

Biba: read/write access defined based on integrity levels

Question 11 (B: 10)

To which access model would the description, "*because you accessed file A, you now cannot access file B*" apply?

(a) Lattice.
(b) Bell LaPadula.
(c) Chinese wall model.
(d) Role-based access control (RBAC).

The Chinese wall model introduces **conflict classes**

If you accessed an object that belongs to Group A, you can no longer access objects that belong to Group B if A and B are in a conflict class

Question 12 (B: 11)

The ***lattice model***:

(a) Creates permissions that may change dynamically based on what objects you previously accessed.
(b) Protects data integrity with a no write up rule.
(c) Is a form of discretionary access control (DAC).
(d) Enhances multilevel security.

MAC model with confidentiality hierarchy like Bell-LaPadula

BUT … compartmentalizes each level to include **labels**

You need to have a matching label in addition to the allowed level to access data

Part 3: Injection/hijacking

Question 13 (B: 19)

A **NOP slide** is useful if you:

(a) Are trying to get stack data to overflow onto the heap.
(b) Are using *return-to-libc* techniques instead of code injection.
(c) Want to pad a region of data to prevent the possibility of off-by-one overflows.
(d) Don't know the precise address of your injected code.

- NOP slide = bunch of NOP instructions before the actual attack code
- CPU can branch anywhere into the slide and skip through these instructions until it reaches the useful code

Question 14 (B: 13)

Fuzzing enables:

(a) Discovery of which input caused a buffer overflow.
(b) Encryption of pointers to protect them from overflows in the heap.
(c) Relocation of the starting addresses of the stack, heap, and text (code).
(d) Run-time buffer overflow checks.

Fuzzing: debugging technique

Supply long chosen data as inputs to cause a buffer overflow

If the program crashes, search for that data in the crash dump

This tells will tell you which input does not check for buffer overflow

Question 15 (B: 14)

Return Oriented Programming (ROP):

(a) Redirects execution to existing code in the program.
(b) Forces functions to return prematurely.
(c) Disables buffer overflow checks.
(d) Injects executable code onto the stack.

ROP does not inject code but stack frames.

The stack frames contain return addresses to code that is already in the executable (usually a library)

Question 16 (B: 15)

Stack canaries:

(a) Detect if a program tries to execute code on the stack.
(b) Ensure that stack data cannot be written outside the stack frame.
(c) Prevent a function from returning if data on the stack has been corrupted.
(d) Prevent buffer overflow in stack-allocated variables.

The compiler generates code to:

1. Push a random number (a canary) onto the stack
2. Check that the canary has not been altered before returning

(a, b, d) They cannot check what happens to data on the stack until the function is ready to return

Question 17 (B: 16)

What technique is ***ineffective*** in preventing Return Oriented Programming (ROP) attacks?

(a) Stack canaries.
(b) Data execution prevention.
(c) Address space layout randomization.
(d) All of the above.

- ROP was designed to bypass no-execute stacks
  - It does not matter if the stack is non-executable because no code is injected into the stack
- Stack canaries – detect corruption of stack data
- ASLR – make it difficult to inject a valid return address

Question 18 (B: 17)

Format string vulnerabilities arise primarily because:

(a) User input is used as the format specifier.

(b) Invalid parameter values are specified.

(c) More parameters are supplied than the format expects.

(d) Assumptions are made on the size of the output buffer.

- When users are given the ability to specify the format string, they can insert directives to dump an arbitrary amount of data from the stack or change a value (%n)
- (b, c): parameters are under control of the programmer, not the user
- Possible with *sprintf*, but output is usually to an output stream

Question 19 (B: 18)

SQL injection attacks cannot be avoided by:

(a) Not using user input as part of a query or command.
(b) Escaping all special characters in the input.
(c) Ensuring the input buffer is sufficiently large to hold the entire query.
(d) Validating the syntax of the input.

The problem is user-supplied query data rather than whether it fits into the buffer

Part 4: Containment

Question 20 (B: 25)

FreeBSD Jails enhance chroot by::

(a) Allowing multiple applications to share the same jail.
(b) Limiting a jailed application's visible file system to a subtree.
(c) Restricting the operations allowable to root (admin) within the jail.
(d) Controlling the system resources (memory, disk) that a jailed process can use.

(a, b) apply to *chroot* as well.

(d) They do not support resource limits

Question 21 (B: 20)

Linux ***namespaces*** do not provide the ability to:

(a) Isolate user IDs.
(b) Restrict access to system calls.
(c) Create per-process network stacks.
(d) Use per-process file system mount points..

Control groups restrict system resources

Capabilities restrict what a user can do as root

Namespaces provide isolated name spaces

Chroot jails are a limited form of namespace that limit what part of the file system you see

Question 22 (B: 21)

Linux **Seccomp-BPF** relies on:

(a) Using containers to isolate processes.
(b) Restricting a process' access to only a subtree of the file system space.
(c) Restrictions on what a process can do when it runs as root.
(d) Kernel-based restrictions on system calls and file access.

Seccomp-BPF (Berkeley Packet Filter) is a sandboxing kernel extension that allows filtering of system calls and their parameters

- Allow/disallow calls
- Allow/disallow access to specific files, network connections, signals

- (a) They are not containers (which generally provide full isolation)
- (b) This is possible via filtering but is a specific example
- (c) They filter calls regardless of whether a process runs as root or not

Question 23 (B: 22)

The **Janus sandbox**:

(a) Uses Linux namespaces for process isolation.

(b) Provides two levels of sandboxing for greater security: user-level and kernel-level.

(c) Uses a user-level process to determine if specific system calls should be allowed.

(d) Validates the operations of a program before it is run to eliminate run-time checks.

Kernel hooks at the system call interface provide callbacks to a user-level process that approves/rejects the call

Question 24 (B: 23)

Linux ***capabilities***:

(a) Restrict what a program can do even if it runs with root privileges.
(b) Restrict the system calls a program can call even if it is not running with root privileges.
(c) Allow parts of the file system to be hidden from an application.
(d) Place limits on the amount of system resources (disk space, network) that a process can consume..

Control groups restrict system resources

Capabilities restrict what a user can do as root

Namespaces provide isolated name spaces

Chroot jails are a limited form of namespace that limit what part of the file system you see

Question 25 (B: 24)

Unlike containers, ***virtual machines*** (VMs) can offer applications:

(a) Separate operating systems.
(b) Isolated namespaces.
(c) Simplified packaging of an application and all its dependencies.
(d) Shared network interfaces.

A VM provides the abstraction of the system hardware (*virtual machine*) and requires an operating system per VM.

The end