Computer Security

06. Exam 1 Review

Paul Krzyzanowski

Rutgers University

Spring 2019

Question 1 (B: 9, C: 8)

Data integrity in a secure system means that:
(a) The source of the data is properly authenticated.
(b) The data is not modified in an unauthorized manner.
(c) The underlying computer system functions correctly, free from interference.
(d) The data is accessible when needed.

Data integrity: assurance that data remains accurate throughout its lifecycle

(a) This is origin integrity

(c) This is system integrity, and may be essential to ensure data integrity

(d) This is availability

Question 2 (B: 1, C: 9)

Asymmetric force in cyber warfare refers to the fact that:
(a) Well-funded organizations will always have the advantage of having more computing resources.
(b) Vulnerabilities are just as likely to be found in large organizations as well as small ones.
(c) Small organizations can potentially overwhelm huge ones.
(d) Attackers must be prepared for large-scale retaliation from their targets. .

Small players have the ability to infiltrate large orgranizations or create botnets that can invoke large-scale attacks or harness compute power at scale

Question 3 (B: 2, C: 1)

A ***trusted computing base*** (TCB) is:
(a) A system that is accessible only to authorized users.
(b) The hardware, firmware, and software that are needed for an application to have a secure environment.
(c) Specially-built computer hardware that is designed to be secure and tamperproof.
(d) A set of trusted libraries integrated with a trusted operating system that enable secure applications.

Everything needed to insure the integrity of the operation of the application

Question 4 (B: 3, C: 2)

A ***capability list*** is:
(a) A set of access rights associated with an object.
(b) A set of access rights associated with a user.
(c) The set of operations that a program is permitted to invoke.
(d) The set of files that a program is allowed to access.

- Access Control List (ACL)
  - List of access rights associated with an object
- Capability List
  - List of access rights associated with a subject
- Neither are associated with the program

Question 5 (B: 4, C: 3)

The ***Principle of Least Privilege*** states that:
(a) Programs should be permitted to access only the resources they need to perform a task.
(b) Access control rules should be kept as concise as possible to avoid comprehension errors.
(c) All access requests should go to the operating system.
(d) Users should not be able to give away files unless they own them.

Minimum access rights –
A process should have rights to do exactly what it needs to do and no more.

Question 6 (B: 5, C: 4)

A ***Discretionary Access Control*** (DAC) model:
(a) Requires administrators to define access rules.
(b) Enables processes to avoid access permission checks.
(c) Enforces privilege separation by having a separate administrator in charge of access permissions.
(d) Allows users to define access rights for objects they own.

- Mandatory Access Control (MAC)
  - The administrator is in charge of setting access permissions
  - Users cannot override these rights
- Discretionary Access Control (DAC)
  - Subjects are in charge of setting access permissions for the objects they own

Question 7 (B: 6, C: 5)

The focus of the ***Bell-LaPadula model*** is to ensure that a user:
(a) Cannot create content at a higher security level.
(b) Cannot read content from a higher security level.
(c) Cannot read or write content at a different security level.
(d) Can create content at any classification level but can read only from lower levels.

Bell-LaPadula model = MAC with a confidentiality hierarchy
- Subjects can write only to the same or higher confidentiality levels
- Subjects can read only from the same or lower confidentiality levels

(c) Not quite true. The user can read from lower levels and can write to higher levels

(d) The user can only create content at the same or higher levels

Question 8 (B: 7, C: 6)

Which security model was designed specifically to keep users from modifying important data?
(a) Biba.
(b) Bell-LaPadula.
(c) Role-based Access Control.
(d) Type Enforcement.

- Biba
  - Designed with *integrity* in mind. Users can read but not modify data at a higher integrity level
- Bell-LaPadula
  - Designed to keep users from *reading* data at a higher confidentiality level … but they can overwrite it (in theory)
- RBAC
  - General purpose: users can have multiple roles which define what access rights they have
- Type Enforcement
  - General purpose: MAC model that defines which subjects can access which objects

Question 9 (B: 8, C: 7)

A ***lattice model*** for access control:
(a) Provides a MAC model that combines controls for both integrity and confidentiality.
(b) Allows an administrator to define arbitrary access privileges based on categories of users.
(c) Enhances an access control matrix to support objects, subjects, applications, and time of use.
(d) Enhances the Bell-LaPadula model to allow data access only if a level has matching security labels.

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

Question 10 (B: 18, C: 17)

Which access control model requires tracking the state of past data accesses?
(a) Bell-LaPadula.
(b) Biba.
(c) Chinese wall.
(d) Lattice.

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 11 (B: 10, C: 18)

Buffer overflow attacks are unlikely in Java because:
(a) Java is an interpreted language.
(b) The Java Virtual Machine is register-based rather than stack-based.
(c) Java performs bounds checking on all array operations.
(d) Java's stack grows up rather than down in memory.

Buffer overflow can occur because the code that is filling the buffer is not checking for the size of the buffer

Question 12 (B: 11, C: 10)

Heap overflow cannot:
(a) Write outside the current stack frame.
(b) Modify dynamically allocated structures (e.g., those created via *malloc* or *new*).
(c) Occur if address space layout randomization is used.
(d) Overwrite a return address.

- (a) The heap *is* outside the current stack frame!
- The heap contains memory for dynamically allocated structures
- ASLR makes it tricky to compute jump addresses but does nothing to prevent overflowing a buffer
- The return address is on the stack
  - Even if you try to overflow the entire heap to get to the stack area (typically hundreds of megabytes or gigabytes), you will reach unmapped memory and fault

Question 13 (B: 12, C: 11)

Which statement about ***stack canaries*** is FALSE?
(a) They cannot detect data modification before a function returns.
(b) They cannot detect changes within a stack frame.
(c) They are useless for detecting heap overflows.
(d) They cannot prevent *return-to-libc* attacks.

- Stack canaries check to see if data has been modified below the bottom of the current frame (possibly overwriting the return address)
- (a) True – they are checked only when a function returns
- (b) True – they only check for an overflow outside the frame
- (c) True – they only check the start of the frame
- (d) False – this attack means the return address was modified

Question 14 (B: 13, C: 12)

Which statement is *FALSE*?
(a) ASLR is ineffective on libraries compiled without position independent code.
(b) ASLR can sometimes be circumvented with a NOP slide.
(c) ASLR makes heap overflow attacks ineffective.
(d) ASLR makes return oriented programming extremely difficult

(a) True: Libraries without position independent code means they use absolute addresses and cannot use ASLR

(b) True: If you know the range of addresses, you can create a NOP slide and jump to the earliest "safe" address and execute the NOPs until the CPU gets to your code

(c) False: You can overflow a buffer that happens to be in the heap

(d) True: With code in random locations, you don't know the address of any library functions

Question 15 (B: 14, C: 13)

Data Execute Protection (DEP):
(a) Ensures that buffer overflows cannot modify data on the stack.
(b) Guards against return-oriented programming.
(c) Makes code injection ineffective.
(d) Allows the system to detect modifications to the stack.

(a) Buffer overflows can still modify the stack

(b) ROP was created to get around DEP

(c) Injecting code via a buffer overflow will not work because that code lives on the stack, which is now not executable

(d) Stack canaries do that, not DEP

Question 16 (B: 15, C: 14)

Parameterized queries in SQL:
(a) Ensure that the parameters match the required data types.
(b) Keep user input from being part of the query statement.
(c) Avoid buffer overflow attacks.
(d) Minimize security risks by allowing one query to handle multiple parameters.

We want to avoid user input becoming part of a query statement so users cannot enter a user name such as:

```
' OR 1=1 --
```

to change the logic of a query to

```
SELECT * from logininfo WHERE username = 'paul' AND password = '' OR 1=1 – ';
```


Question 17 (B: 16, C: 15)

The ***%n*** format directive of the ***printf*** function:
(a) Outputs the corresponding parameter as a base ten number, allowing an attacker to examine the stack.
(b) Prints a newline, which can make a single log entry look like multiple log entries.
(c) Stores a value in a memory address identified by a parameter.
(d) Reads user input for the corresponding parameter.

- %n stores the number of bytes output so far into a parameter
  - If an attacker can enter a format string, an arbitrary number of format directives (%) allows you to skip parameters until you get to the one that contains the address of interest
  - Specifying the output size in the format directive lets you control what the value written will be

Question 18 (B: 17, C: 16)

A program is at risk of a ***TOCTTOU*** attack if it:
(a) Accepts a user-supplied filename and then opens the file.
(b) Opens a file and then sets its permissions to disallow other users from reading or writing it.
(c) Fails to check the amount of data it reads into an array.
(d) Does not check to make sure that special characters in user-supplied data are properly escaped.

- TOCTTOU is a race condition where an attacker may get a brief window of opportunity to access a resource
- Opening a file and later setting permissions creates such a window
  - We don't explicitly "check" but the OS applies default permissions that we later override

Question 19 (B: 25, C: 24)

Linux ***capabilities***:
(a) Provide custom name spaces to processes.
(b) Can take away some access permissions that a root user has.
(c) Define resources to which a process has access.
(d) Limit the ability for a process to communicate with other processes.

(a) **Namespaces** provide isolated name spaces

(b) Capabilities restrict what a user can do even as root

(c) **Control groups** restrict system resources

(d) **Namespaces** isolate processes so they may not be able to communicate

Question 20 (B: 19, C: 25)

Which of these enables creating a ***communication barrier*** between containers?
(a) Control Groups.
(b) Capabilities.
(c) Chroot jails.
(d) Namespaces.

See the previous question:

Control groups restrict system resources

Capabilities restrict what a user can do even as root

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

Namespaces provide isolated name spaces

Question 21 (B: 20, C: 19)

Containers are said to provide ***operating system level virtualization***.
This means that:
(a) Multiple instances of the same operating system can run on one computer.
(b) Groups of processes are segmented from one another but share the same operating system.
(c) An adaptation layer of software allows an application to run on a different operating system.
(d) Multiple different operating systems can be installed and run concurrently on one computer.

- Containers provide isolation via namespaces, capabilities, and control groups
- Multiple containers can run under the same OS

Question 22 (B: 21, C: 20)

The biggest security risk with containers is that:
(a) They can be deployed on arbitrary systems, some of which may not have sufficient protections.
(b) Applications in different containers share the same operating system.
(c) Containers may conflict with Linux's use of namespaces, control groups, and capabilities.
(d) They do not create a reproducible environment, making it difficult to recreate problems.

Even though processes are isolated, the share the same OS
- Breaking out of a container means you have access to other processes & the shared OS

(a) Not trusting the TCB is a more general problem than containers!

(c) No – they use namespaces, cgroups, & capabilities

(d) They do create a reproducible environment

Question 23 (B: 22, C: 21)

Container orchestration refers to:
(a) Selecting the appropriate container technology for each application.
(b) Being able to mix containers from different vendors and use them together.
(c) Running multiple containers across multiple computers.
(d) Packaging a set of related services into a single container.

Question 24 (B: 23, C: 22)

A ***Type 1 hypervisor***:
(a) Supports the installation of an arbitrary number of operating systems as long as they are of the same type.
(b) Uses a single operating system to provide the illusion of multiple operating systems.
(c) Does not need to send requests to a host operating system to handle interactions with the underlying hardware.
(d) Allows applications to run directly on the hypervisor without a need for an operating system.

- A Type 1 (or *bare metal*) hypervisor virtualizes the underlying hardware directly.
  - Any installed operating systems interact with these virtual devices
- A Type 2 (or *hosted VM*) hypervisor routes device access requests to the "main" OS that owns access to the hardware

Question 25 (B: 24, C: 23)

A ***side-channel attack***:
(a) Compromises an intermediate process, which then attacks the target, hiding the intruder.
(b) Uses information from the behavior of a computer system rather than weaknesses in the algorithm.
(c) Breaks through a container to establish a communication link with a process.
(d) Attacks the underlying operating system, from which it can obtain information about the target process.

The end