Computer Security

06. Exam 1 Review

Paul Krzyzanowski

Rutgers University

Spring 2018

Question 1a

Briefly explain how each of these three mechanisms helped to make buffer overflow attacks more difficult:

a. Data execution Protection (DEP)

The OS takes away execute permission from areas of memory that are not loaded with executable code, such as the stack and the heap.

Question 1b

Briefly explain how each of these three mechanisms helped to make buffer overflow attacks more difficult:

b. Address Space Layout Randomization (ASLR)

- The OS's program loader sets a random base address for code (text), heap, stack, and shared library regions.
- If you are injecting code, you will not know what return address to inject to have it refer back to the buffer.
- If you are using Return-Oriented-Programming (ROP), you will not know the address of the library function you want to branch to.

NOT: "randomly assigns data in the stack"

Question 1c

Briefly explain how each of these three mechanisms helped to make buffer overflow attacks more difficult:

c. Stack canaries

- Allows the function to check if the return address on the stack may have been overwritten.
- Notes:
  - The canary is not in place to detect off-by-one errors. That can be a side-effect if the buffer overflow was next to the canary.
  - NOT "detect adjacent buffers". The canary is in place to ensure that the return address and saved frame pointers have not been modified. An overflow of *any* local buffer in the function can cause that to occur.

Question 2

How to POSIX (Linux) capabilities help implement the principle of least privilege better than using a setuid mechanism?

- Setuid is used on a program when the user that runs the program will not have sufficient privileges
- Most often, programs run with the user ID of root (0) to perform certain administrative actions
- Linux capabilities enable an administrator to grant specific privileges to a program
  - It can perform limited administrative operations rather than have broad access to everything that root can do
- Note: Linux capabilities are associated with a file, not a user

Question 3

Usurpation is a great vocabulary word. It primarily refers to:
(a) Taking control of part of a system without permission.
(b) Disrupting the service that a system is providing.
(c) Injecting or modifying data in a network or file system.
(d) Accessing data without authorization.

Definition of **usurp**

transitive verb

1 a : to seize and hold (office, place, functions, powers, etc.) in possession by force or without right usurp a throne

b : to take or make use of without right usurped the rights to her life story

2 : to take the place of by or as if by force : supplant must not let stock responses based on inherited prejudice usurp careful judgment

- Taking control when you have no right
  - Disruption or data access might be side-effects
  - Injection might be the means to do it

Question 4

Snooping is a form of:
(a) Disclosure.
(b) Disruption.
(c) Usurpation.
(d) Repudiation of origin.

(a) Snooping refers to looking at data without permission

(b) It doesn't imply changing the data and usually doesn't disrupt service

(c) Usurpation may be used to enable snooping but not necessarily

(d) You're just observing – not changing anything

Question 5

An important security-relevant aspect of an operating system's use of *hardware timers* is:
(a) Waking the system up from sleep to ensure availability.
(b) Measuring the amount of time various processes use.
(c) Measuring the latency of system calls to look for evidence of tampering.
(d) Ensuring that the operating system can always regain control.

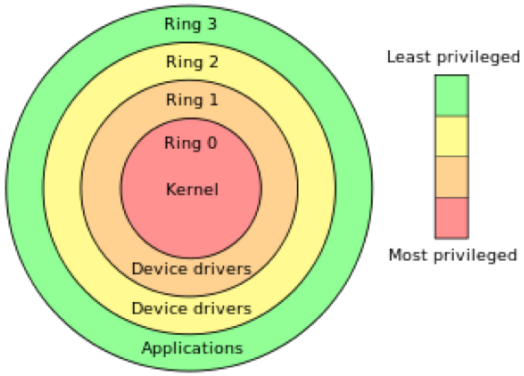
- Hardware timers are used to trigger interrupts in the future
  - This allows the OS to ensure it will get invoked at a set time
  - Generally used for process scheduling

Question 6

Protection Rings in the Intel architecture are designed to:

(a) Allow each process to occupy a unique privilege level.
(b) Protect the processor itself from malicious attacks.
(c) Provide multiple layers of checks when a process requests access to resources.
(d) Provide well-defined mechanisms for a process to switch between privilege levels.

- Protection rings are designed to provide a hierarchical set of protection domains
  - Call gates (like interrupts) transition to specific places in rings
- Envisioned design was to have unprivileged software in Ring 3 and the OS in rings 0-2.
  - Ring 0: kernel: full privileges
  - Rings 1, 2: OS drivers – can access kernel memory but not run privileged instructions
- Rings 1 & 2 almost never used in real life
  - VirtualBox VM runs its guest OS in ring 1



(a) No: all non-kernel processes are in ring 3
(b) No: they protect the OS but not the CPU
(c) No: it's the access control logic in the kernel that decides if you can access a resource

Question 7

Why did Unix systems adopt a limited file access control model?

(a) The Unix OS was designed as a single user system, so detailed per-user controls were not needed.

(b) The model is sufficient since it fully implements an access control matrix.

(c) It made the commands to set permissions a lot simpler.

(d) The limited set of permissions fit within a fixed set of bits in the file's inode.

- Unix's predecessor, Multics, supported full ACLs
- An ACL can take up an arbitrary amount of space
  - It's a list, possibly with lots of users and access rights
  - Implementation would require a variable-size inode or a secondary variable-size storage list in addition to file data … that's painful!
- Unix's ability to assign access permissions for only a single user or a single group is limiting – usually good enough but not = a full access control matrix

Question 8

Why is a compromise of the trusted computing base (TCB) particularly dangerous?

(a) It makes it easy to perform code injection attacks on applications.

(b) You can no longer trust that the system is enforcing security policies correctly.

(c) Malware can easily spread to other less-trusted systems.

(d) One process can freely read another process's memory.

- The TCB includes all lower-level software & hardware
  - E.g., processor, firmware, OS
- If the TCB cannot be trusted, its ability to enforce security policies cannot be trusted

Question 9

Each *access control list* (ACL) allows one to specify which:
(a) Programs can access a specific file.
(b) Files can be accessed by a specific user.
(c) Users can access a specific file.
(d) Users have administrative privileges.

- An ACL is associated with a single file
- It comprises a set of Access Control Entries
  - Each ACE identifies a set of rights for a single subject (e.g., user)
- ACLs manage subject-object mapping
  - They don't allow one to limit what a specific program can do
  - Programs assume the rights granted to a user

Question 10

A program with the *setuid* bit set:
(a) Runs with permissions of the logged-in user instead of the file owner.
(b) Runs with permissions of the file owner instead of the logged-in user.
(c) Runs with administrative privileges.
(d) Runs the process in kernel mode instead of user mode.

- A setuid executable file runs with the permissions of the file owner rather than the user who runs the program
- They are often owned by *root*, in which case they run with admin privileges … but that is not a requirement

Question 11

Privilege separation is the principle where:
(a) Different users are granted different access rights.
(b) The system has different classes of administrators (e.g., network, storage, users).
(c) A process is split into multiple parts, each running at different privilege levels.
(d) A process may request special privileges to do specific operations.

- Break a program into parts, giving each part only the privileges it requires
  - This can be an application of the principle of least privilege
  - Minimizes potential for harm if one of the parts is compromised

Question 12

Mandatory Access Control (MAC):

(a) Is a set of access rights that the kernel enforces while Discretionary Access Control defines only advisory permissions.

(b) Defines access rights that the object owner cannot change.

(c) Is a policy that states all objects must have a set of access rights associated with them.

(d) Is an operating system mode that forces the checking of access permissions to each object.

(a) Kernels enforce both MAC and DAC

The user can define DAC; the admin defines MAC

(b) Yes – MAC is set by the admin, not resource owners

(c) MAC is a mechanism, not a policy

(d) The OS checks access permissions for MAC and DAC

Question 13

The main problem with a program that creates a file and then sets access permissions so that the file is not readable by anyone other than the user is:
(a) The user might be an imposter.
(b) There is a race condition in the logic.
(c) The user account might be deleted, making the file unreadable by anyone.
(d) Either of the operations may fail.

(a) – Possibly, but …

there are other problems then, such as how the user became an imposter

Do we care? The goal of the user is to create a private file

(b) Yes. There's a window of time between creating the file and its setting permissions

(c) Not clear if anyone would need to read the file. Admins can step in.

(d) That's not the main problem

Question 14

A drawback of the Bell-LaPadula model when strictly implemented is:
(a) A user at a low classification level can read files from higher classification levels.
(b) A user at a low classification level may overwrite a file at a higher classification level.
(c) A user at a high classification level can freely read files from lower classification levels.
(d) A user at a high classification level may overwrite files at lower classification levels.

Bell-LaPadula is all about confidentiality

- No read up: you cannot read from a higher secrecy level
- No write down: you cannot create anything less secret than your level

(a) No.

(c) Yes, but that's expected.

(d) No.

Question 15

One way that role-based access control differs from the Bell-LaPadula model in that:

(a) It focuses on preserving integrity rather than confidentiality.

(b) It manages permissions on a functional basis rather than controlling object access.

(c) It is mandatory rather than discretionary.

(d) Users can be assigned to different classification levels.

RBAC is about assigning users to roles and managing roles

- Roles are functional operations (e.g., modify payroll data), not confidential or integrity levels

Question 16

The Biba model differs from Bell-LaPadula in that:

(a) Users are not classified into levels.

(b) It implements a no write up policy instead of no read up.

(c) It allows reading across any levels but restricts writing since it is concerned only with controlling data corruption.

(d) It is a discretionary model rather than a mandatory model.

Biba is an integrity model

- Instead of levels of secrecy, we have levels of integrity
- No read down: you cannot read from lower integrity levels
- No write up: you cannot create objects that are a higher integrity

Question 17

Multilateral security enhances the Bell-LaPadula model in that it:

(a) Restricts what information users can access even at the same classification level.

(b) Applies the same policy uniformly to all users of the system.

(c) Supports discretionary access control within a security level.

(d) Manages integrity as well as confidentiality.

- Multilateral security adds compartment labels within a classification level
- Labels must match for you to have access to the objects

Question 18

The key idea in the Chinese Wall model is:

(a) Groups of users may be partitioned so they cannot communicate.

(b) Access may be disallowed based on what objects were previously accessed by the user.

(c) Groups of users may be defined so that information can flow only in one direction from one group to the other.

(d) It preserves integrity by disallowing a user in one group from modifying data in another.

- The Chinese Wall model is about conflict classes
  - If you access an object A
    and object B is in the same conflict class, you can no longer access B
- It requires tracking state.

Question 19

How come *buffer overflow vulnerabilities* practically don't exist in Java?
(a) Java is not a stack-based language.
(b) The Java runtime environment employs stack canaries.
(c) Java implements bounds checking for all array operations.
(d) Java objects use memory protection to isolate themselves.

- Java is strongly typed and enforces bounds checking for all arrays
- You cannot write to a pointer or fill an array of an unknown size
  - Each array has a `public final length` field that identifies the # of elements

```
int a[];
a.length
```

  - Strings have a `public length` method

```
String s;
s.length()
```

Question 20

A *landing zone* is:

(a) The address of the start of injected code that is written to the stack.

(b) An indirect jump in a buffer overflow exploit to enable the injection of larger chunks of code.

(c) A region of memory that is vulnerable to heap overflow attacks.

(d) A sequence of no-op instructions to account for the fact that we might not know the exact address of a buffer..

In buffer overflow attacks, if you're not sure of the precise starting address of your buffer, you can pad it with NOPs and jump into this list of NOP instructions

Question 21

The *printf* function is a potential attack vector if:
(a) The attacker provides data that exceed the sizes specified in the format string.
(b) The program uses the wrong number of arguments to *printf*.
(c) The attacker can specify the format string.
(d) The size of the output buffer is not specified.

(a) What data is the attacker providing to do this?
A string can cause a lot of output but isn't really an attack

(b) That's a bug but highly unlikely to be an attack vector

(c) This allows an attacker to do things such as examine the stack and modify data

(d) There is no output buffer

Question 22

Fuzzing is a technique that:
(a) Obfuscates compiled code to make it difficult to disassemble.
(b) Provides barriers between buffers to ensure that buffer overflow cannot occur.
(c) Scrambles addresses on the stack to make it difficult for attackers to inject valid addresses.
(d) Overflows a buffer to crash a program and then searches for the location of the data that was input.

- Fuzzing is used to test if a program is resilient against bad input data and – if not – identify where the culprit is

Question 23

Return-oriented programming (ROP) was created to:
(a) Enable code injection without buffer overflows.
(b) Make buffer overflows impossible.
(c) Enable code injection with data execute protection in place.
(d) Bypass the protections of stack canaries.

(a) ROP relies on buffer overflows

(b) It's an attack, not a defense

(d) A stack canary will detect the overflow and refuse to return to the injected address

Question 24

What would be most likely to cause a compiled program to behave differently from the way it was designed?

(a) A change to the shared library path (LD_LIBRARY_PATH).
(b) A change to the shell's search path (PATH).
(c) A change to the environment variable that redefines the shell's field separator characters (IFS).
(d) A change to the current directory before the command is run (cd).

(a) This can lead to the the wrong library function being called

(b) This can invoke a different program if the user doesn't specify the full path

(c) Only in the rare case that the program being run happens to execute a script

(d) Worst answer. Possible if the user was expected to run the file from a specific directory

Question 25

A possible security problem with closing the standard output or standard error stream when running a command is:

(a) The program will not be able to write messages to the user.

(b) The program will crash as soon as it writes to the console, resulting in an availability attack.

(c) If the program opens another file, any attempts to write to the standard output may corrupt that file.

(d) The program will be blocked indefinitely waiting for the output stream to open..

(a) Not likely to be a security problem.

(b) Nope. Failed writes don't result in a crash.

(c) A new open file will get the same file descriptor as the closed stream.

(d) Nope.

Question 26

The Unicode bug that Microsoft had in their IIS (Internet Information Services) server manifested itself because:

(a) Many Unicode characters looks the same.

(b) They processed Unicode characters after validating the pathname.

(c) There was no reliable way of checking whether a URL specifies a path above a base directory.

(d) A buffer overflow attack enabled Unicode characters to be treated as executable code..

- Unicode expansion could have resulted in "/" characters, which altered the search path.

Question 27

Homograph (homoglyph) attacks work because:
(a) Some different characters look the same across multiple international scripts.
(b) Different multi-byte Unicode encodings may ultimately map to the same character.
(c) Systems sometimes validate input before parsing multi-byte characters.
(d) Text may contain a mixture of scripts from different languages.

A homograph attack is a deception attack

– Different characters may look identical in different scripts

(b, c): This was the problem in IIS in question 26

(d) Yes, it may, but the attack is that the characters look as if they belong to another script

Question 28

Which is ***not*** a problem with the *chroot* mechanism?
A process running with administrative privileges can:
(a) Reset the root back to its original value.
(b) Create a device file to access the root file system.
(c) Invoke privileged system calls.
(d) View and kill other processes.

(a) Anything outside of the new root is invisible

(b) Yes – and then mount it and see all contents

(c) Yes – there's no ability to restrict operations

(d) Yes – the process namespace is visible and operations are not restricted

Question 29

FreeBSD Jails improved the *chroot* concept by:
(a) Restricting the operations that a process can perform in a jail.
(b) Managing the amount of resources that a process can consume.
(c) Disallowing processes from resetting the root of the file system.
(d) Logging all activity to an audit file.

- They reduce the power of root in a jail
  - Main goal: disallow the process from creating device files so they can access the full file system

Question 30

Linux *control groups* (*cgroups*):

(a) Monitor and restrict the use of various computing resources for processes.

(b) Allow an administrator to start and stop a collection of processes as one group.

(c) Restrict the administrative functions that processes can perform.

(d) Restrict the part of the file system that is visible to a process.

- Control groups: control resources
- Namespaces: control visibility of files, processes, …
- Capabilities: restrict administrative capabilities

The end