**Homework 2 for CS 165 (Winter 2019)**

***Due: on ilearn by the end of day on Mar 14 2019***

**Instructions:**

**\*** Be brief in your answers. You will be graded for correctness, not on the length of your answers.

\* Remember to submit online through ilearn if you didn’t turn it in in-class. Paper copy will not be accepted.

I. Answer the following multiple choice questions (one or more correct answers). (1 point \* 13)

1. Which of the following types of locations in a process address space that buffer overflow attacks typically target? \_\_\_\_\_\_\_\_\_\_

a) Stack

b) Heap

c) Code

2. Which of the following are common defenses against stack buffer overflow? \_\_\_\_\_\_\_\_\_

a) Address Space Layout Randomization (ASLR)

b) DEP/Non-executable stack

c) Control-Flow Integrity (CFI)

3. Which programming languages are vulnerable to stack buffer overflow? \_\_\_\_\_\_\_\_\_

a) C

b) Java

c) Assembly

4. How can a stack buffer overflow hijack the control flow of the program? \_\_\_\_\_\_\_\_\_

a) Overwriting the return address on the stack

b) Overwriting a function pointer on the stack

c) Overwriting the saved stack frame pointer (saved EBP)

5) With DEP defense enabled, which of the following becomes impossible? \_\_\_\_\_\_\_\_\_

a) Overwriting the return address on the stack

b) Injecting shellcode onto the stack and execute it by jumping to it

c) Finding a useful gadget to jump to in Return-Oriented Programming (ROP)

6) In the ROP attack, which register now plays the role similar to that of instruction pointer (EIP)? \_\_\_\_\_\_\_\_\_

a) EBP

b) EAX

c) ESP

7) Which defenses have been (partially) deployed in modern operating systems? \_\_\_\_\_\_\_\_

a) Access control list

b) ASLR

c) Control-Flow Integrity (CFI)

8) Which are the reasons why blind ROP attack against a web server works despite the fact that all modern defenses are deployed? \_\_\_\_\_\_\_\_\_

a) Web server forks a child process with the same address space layout every time to serve a new connection

b) ASLR is forgotten to be enabled

c) Stack canary value stays the same even if a guess is wrong

9) How does the blind ROP attack determine if a code sequence contains the desired gadget (since it’s blind)? \_\_\_\_\_\_\_\_\_

a) It learns the address of the gadget by obtaining a copy of the binary beforehand

b) It sets up the stack in special ways so that the detected gadget will be uniquely identifiable

c) It leverages the feedback about whether a server has crashed or not

10) Which of the following of x86 architecture makes overlapping instructions possible (i.e., one can jump to the middle of an instruction and the CPU can recognize it as a different yet valid instruction? \_\_\_\_\_\_\_\_\_

a) Instruction lengths are not multiples of 8 bits

b) Instructions have variable length

c) Most byte sequences are legal instructions

11) What are some examples that conceptually map to the BLP or Biba model? \_\_\_\_\_\_\_\_\_

a) In buffer overflow, command line argument is considered a low-integrity object. A root process is considered a high-integrity subject that should not be allowed to read the low-integrity data (thus allowing control flow to be hijacked)

b) In time-of-check, time-of-use attack, the file or directory controlled by an attacker is considered a low-integrity object. A root process is considered a high integrity subject that should not be allowed to read the low-integrity data (thus being tricked to perform unintended operations)

c) In directory traversal attack, the passwd file is considered the high-secrecy object. A root process (web server) is considered a low-secrecy subject since it needs to read the public HTML files and serve them to clients. A low-secrecy subject should not be allowed to read a high-secrecy object (thus leaking the passwd file unintendedly)

12) Which of the following about resource access attacks are correct? \_\_\_\_\_\_\_

a) They are caused by violations of BLP or Biba security policies.

b) They are caused by mismatches of expectations (e.g., high-integrity subjects expect high-integrity objects but mistakenly got low-integrity objects).

c) We need to look at both the code and access control policy to identify resource access attacks.

13) In computer security, there’s a well-known principle called principle of the least privilege. The idea is that every subject (process, user, program) should have access to only the information and resources they absolutely need (no more should be allowed). Which of the following are correct based on your judgement? \_\_\_\_\_\_\_\_\_

a) Not running processes as root when not necessary (e.g., chrome or firefox) is a one example of principle of least privilege

b) The reasoning behind the principle is to prevent an attacker to compromise a subject

c) The reasoning behind the principle is to reduce the damage once a subject is compromised

II. The following C library routines are unsafe and can lead to buffer overflow vulnerabilities. Use the UNIX main page to find out their alternative safe implementations. (2 points)

|  |  |
| --- | --- |
| **gets(char \*str)** | read line from standard input into str |
| **sprintf(char \*str, char \*format, . . .)** | create str according to supplied format and variables |
| **strcat(char \*dest, char \*src)** | append contents of string src to string dest |
| **strcpy(char \*dest, char \*src)** | copy contents of string src to string dest |
| **vsprintf(char \*str, char \*fmt, va\_list ap)** | create str according to supplied format and variables |

III. Here’s a chunk of code that’s intended to “combine” two strings together by taking characters alternately from each. For instance, calling combine on ABC and def should return AdBeCf. Unfortunately you’ll see that this function was not implemented securely at all. (5 points)

char \*zip(char \*a, char \*b) {

char \*result;

int len, i;

len = strlen(a);

result = malloc(2\*len);

for(i = 0; i <= len; i++) {

result[2\*i] = a[i];

result[2\*i+1] = b[i];

}

return result;

}

1. Describe at least three bad things that could happen when running this function in situations that the programmer probably didn’t think of. For each case, identify the programming mistake, the problematic situation, and the bad outcome. (2 points)

2. Provide a safer implementation for this function. Note that your new implementation will have to behave differently than the old implementation in some circumstances (probably including, though not limited to, those situations in which the old one could crash). Think carefully about what behavior would be best, and explain your choices. (2 points)

IV. Consider the following code with a race condition vulnerability. Can you explain how this code can be exploited? (2 points)

void process\_request(int request, int pid) {

if(security\_check(pid) == true) perform(request);

}

bool security\_check(int pid) {

string file = “/proc/” + pid + “/status”;

int fd = open(file); // open the proc file for pid

int uid = read\_uid(fd); // obtain the uid by reading the content of the file.

If(uid == 0) // if it is root process

return true; //passed the security check

return false;

}

V. Briefly answer the following questions (1 point \* 6)

1. Give two reference monitor examples where one is an inline reference monitor work and the other is not.

2. What is the principal difference between the BLP model and the Biba model?

3. In x86 and other modern CPUs, the processors often have different modes of privileges (e.g., kernel and user). If we think of the kernel code as high-integrity subject, kernel data as high-integrity data, user code as low-integrity subject, user data as low-integrity object, then according to the Biba policy, kernel code should not read user data. Give at least one reason why this may not be feasible in practice.

4. In control-flow integrity (CFI), describe why an attacker cannot circumvent the checks that are inserted for every control transfer?

5. Describe the relationship between the return-to-libc attack and ROP attack.

6. In network security, we discussed the following three threat models: 1) eavesdropper, 2) MITM, 3) Off-path. Please list them from the most realistic (how easy it is to achieve the threat model) to the least realistic.

VI. Consider the following new strategy to prevent stack buffer overflows:

Instead of push return address onto the same stack that’s mixed together with data (e.g., local variable: buffers), the CPU will push it to a dedicated stack that stores only the return address. Similarly, when ret instruction is invoked, the CPU will automatically pop from the dedicated stack. This way, a buffer overflow will overflow the stack that no longer stores the return address and therefore all return-oriented buffer overflow attacks are automatically prevented (impossible to occur).

1. Is the above strategy secure? If not, give one counterexample. (1.5 points)

2. Give one other reason (security or non-security reason) why this approach may not be desirable. (1.5 points)

3. Suggest any idea to improve the design to make it more secure or more practical? (3 bonus points)

VII. Information flow is a useful mechanism that can enforce a variety of security policies that are hard to describe in simple access control lists. Describe how you can use information flow as a building block to catch the following attacks or vulnerabilities. Please specify the information sources and sinks that one should track (1 point \* 3):

1. A buffer overflow attack that overwrites a function pointer.

2. Link following attack where a victim process reads a file from an attacker-controlled directory.

3. Skype app on Android accidentally stores the password in plaintext in a file that’s accessible to everyone.

VIII. In off-path UDP or TCP attacks, an attacker needs to guess or predict certain information (sometimes including application-layer information) in order to launch successful attacks Describe below exactly what information is necessary to complete the attack : (1 point \* 3)

1. An attacker using sequence number prediction to establish a TCP connection with a server using a spoofed IP address.

2. The firewall-enabled off-path TCP sequence number inference attack.

3. DNS poisoning attack.

IX. In the firewall-enabled off-path TCP sequence number inference attack, the sequence-number-checking firewall is shown to be the cause of sequence number information leakage. Please answer the following questions about this attack.

1. The bigger the sequence number window (maintained by the firewall), the easier it is for the attack, assuming the window never exceeds half of the sequence number space. True or False? Explain your answer. (1 point)

2. The design of the firewall is that after three-way handshake (ISN of client is X and ISN of server is Y), the firewall maintains the window of (X-WIN, X+WIN) where all future packets with the client IP as source IP needs have sequence numbers within this window. It holds similarly for the window of (Y-WIN, Y+WIN). In the original attack, it is assumed that the window advances only when the firewall sees a packet delivered with the exact expected sequence number, e.g., when a packet from client has sequence number X with a size of 1000, the window will advance to (X+1000-WIN, X+1000+WIN). Otherwise, the window stays at the same place. Consider the following alternative design to advance the window: every time when an in-window sequence number is observed, let’s say X’, the center of the window is shifted immediately to X’, therefore the window becomes (X’-WIN, X’+WIN). In this design, can the original attack still succeed in inferring the exact sequence number X or Y? Explain your answer. (2 points)

3. Can you give an alternative design to the sequence-number-checking firewall to avoid the vulnerability while still maintaining the benefit of filtering blindly injected packets with incorrect sequence numbers? (2 bonus points)