**Final Exam for CS 165 (Fall 2021)**

***Dec 7, 2021***

**Instructions (write down your name):**

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

I. (1 point \*10) Answer the following multiple choice questions (one or more correct answers).

1. What can be said about control flow hijacking vulnerabilities? \_\_\_\_\_\_\_\_

a. They are inherently difficult to eradicate as long as we can continue to use unsafe programming languages such as C.

b. C++ will not allow control flow hijacking because it is object-oriented.

c. Format string vulnerability is an example vulnerability type that can lead to control flow hijacking.

2. Which of the following are true about attack surface? \_\_\_\_\_\_\_

b. To a remote attacker, the attack surface a system should include all TCP and UDP ports listening on the system.

a. Entry points of a program can be defined as any external inputs consumed through syscalls, e.g., files.

c. Defining attack surface requires reasoning about an adversary’s capabilities (threat model).

3. What role does the access control play in improving security? \_\_\_\_\_\_\_

a. It eliminates buffer overflow vulnerabilities.

b. It reduces attack surface.

c. It guarantees that an attacker is unable to access certain resources.

4. Why is security hard to achieve in general? \_\_\_\_\_\_\_

a. Because defenses can be bypassed.

b. Because software is complex and hard to avoid loopholes.

c. Because computer systems and networks were designed initially without much consideration of security and it is now difficult to retrofit security (due to backward compatibility).

5. Which of the following are good applications/analogies of security policies? \_\_\_\_\_\_\_

a. “A king does not eat anything prepared by untrusted people” is an application of the Biba policy (the integrity policy).

b. “A web server should not process trusted input from arbitrary users” is an application of the BLP policy (the secrecy policy).

c. “A mail server should not have read from a file that can be modified by a regular user” is an application of the Biba policy (the integrity policy).

6. Which of the following are true about information flow analysis? \_\_\_\_\_\_\_

a. It is possible to use information flow to describe security policies.

b. Information flow can be viewed as a building block for the BLP policy (secrecy policy).

c. Information flow can be viewed as a building block for the Biba policy (integrity policy).

7. What are the pros and cons between static and dynamic analysis for vulnerability discovery?

\_\_\_\_\_\_\_\_

a. Static analysis has the advantage of “covering” all program paths that are not necessarily exercised dynamically.

b. Static analysis is more flexible because it can start the analysis abstractly from any function (e.g, not necessarily the main function).

c. Dynamic analysis is effective only when the program is smaller.

8. Network firewalls and intrusion detection systems can be deployed on the host as well. What are the reasons that the network version be more desirable? \_\_\_\_\_\_\_\_

a. Host-based IDS has significant maintenance and runtime overhead.

b. Network-based IDS has the advantage of covering more hosts.

c. Network-based IDS has a global view of the network and therefore in principle more knowledge on whether certain attacks (e.g., DDoS) are happening.

9. Which of the following are true regarding vulnerability research? \_\_\_\_\_\_\_

a. Fuzzing is highly effective in finding bugs but those bugs are not necessarily exploitable vulnerabilities.

b. One can apply either static analysis or dynamic analysis to find bugs.

c. Exploiting a vulnerability is always easier than finding the vulnerability in the first place.

10. Patching looks like fairly straightforward process to fix bugs and vulnerabilities. Why would it be considered a problem? \_\_\_\_\_\_\_\_

a. IoT devices may not receive patches because devices may outlive the lifespan of manufacturers.

b. The constant stream of new bugs being discovered can be overwhelming for developers to fix.

c. In a complex ecosystem, if a bug is found in the upstream, there can be multiple layers/hops of patch propagation, e.g., from Linux mainline all the way down to Android devices.

II (1.5 points) An important security principle is called defense-in-depth. It advocates multiple layers of defenses as it will still provide some protection even if one defense layer is bypassed. Can you explain how Android applies this principle to defend against malware? Please give at least three major defenses employed by Google.

III. (2 points) One principle of secure programming is called “fail-safe”, where the system or program should maintain security even when it encounters failure or errors. Consider the following general code for allowing access to a resource:

int ret = IsAccessAllowed(...);

if (ret == ERROR\_ACCESS\_DENIED) {

// Security check failed.

// Inform user that access is denied.

} else {

// Security check OK.

}

a. Explain the security flaw in this program from the fail-safe perspective. Hint: no race condition is involved. (1 point)

b. Rewrite the code to avoid the flaw. (1 point)

IV. Below is an interactive program where the main function runs in a while loop and processes commands through command line one after another, i.e., doCmd() is called multiple times with a *cmd* value and *arg* string decoded from the input. Assuming the decoding is done correctly and safely, can you spot any bugs with the program and address them? (4 points)

int \*data\_ptr; // global variable

int size; // global variable

char doCmd (int cmd, char\* arg) {

if (cmd == CREATE) {

size = atoi(arg);

data\_ptr = malloc(size);

}

if(cmd == TEST) {

for (unsigned int i = 0; i < size; ++i) {

printf(“%d\n”, data\_ptr[i]);

}

}

if(cmd == UPDATE\_SIZE) {

size = atoi(arg);

}

}

1. Describe clearly four bad things that could happen. (2 points)

2. Describe how you might use static analysis in terms of control flow and data flow to discover them. (2 points)

V. Thinking like an attacker (1 point \* 6)

1. In project 2, we were able to crack programs easily by modifying its intended behaviors at the binary level. What is the fundamental reason that we can crack a program? Alternatively, would you be able to design a program that prevents a hacker from cracking it (or at least make it substantially more difficult)? Answering either of the question properly would get you the full point.

2. In Intrusion Detection Systems (IDS), one main challenge is to keep up with the new malware samples that appear every day. One specific difficulty is to collect such malware samples. To solve this problem, one solution is to deploy “honeypots” where virtual machines are intentionally configured to be vulnerable and publicized to attract attackers. Once the machine is compromised, all new executables downloaded onto the machine will be considered malware samples. This whole process can be fully automated for malware sample collection. What is the potential risk of this solution?

3. Modern operating systems have already provided the basic isolation mechanism --- each process has its own independent virtual address space. By design, one process is not allowed to read/write the memory of another process. However, there are cases where a single process can have multiple threads which share the same address space. In such cases, what techniques do you think are suitable to solve this problem and why? (please use the knowledge given in the lectures only).

4. For project 3, explain how you might use the vulnerability of the binary to cause trouble of other teams without being easily noticed.

5. Google operates a fuzzer that continuously fuzzes Linux kernels and on average finds 1000+ bugs in a single year. They even publish all the bug reports in real time publicly. Why do you think they do this? Wouldn’t it allow attackers to take an early look at these bugs and try to exploit them? Please construct an argument that this is in some way helpful for security.

6. Linux is known for intentionally obfuscating the nature of a kernel change, i.e., git commit. This means that even if a commit is to fix a serious security vulnerability, the commit message would not indicate as such. This concealing of information is a direct contradiction to what Google practices making everything (e.g., bug reports) public. The intention is to avoid helping attackers identify which patches fix important security vulnerabilities and develop exploits for unpatched systems. Can you come up with an argument this pro-security practice can actually end up hurting security?

VI (1.5 points). In the 2016’s off-path TCP attack, the presence of a connection between two arbitrary hosts can be inferred through side channels by a completely off-path attack. Please answer the following questions.

1. We know side channels always require some shared resources between an attacker and victim. What is the shared resource in this case? (0.5 point)

2. Compared to the off-path attack in 2012 (where malware on the client is used to assist the attack), it generally takes a lot longer to complete the attack proposed in 2016. What makes it more difficult? (1 point)

VII (1.5 points). We introduced a side-channel-enabled DNS cache poisoning which is a serious attack that can cause incorrect mappings between domain names and IP addresses cached in a DNS resolver. The fundamental flaw is that the source port used in a DNS request can be learned by an off-path attacker through a side channel. Since we learned that any side channel would involve some form of shared resource, can we simply eliminate the shared resources such that the side channel can no longer work? Would there be any potential side effects? If so, what would be an alternative to eliminating the sharing?

VIII. (6 points) Design question: Consider the following alternative design to defend against buffer overflows and control flow hijack in general:

a. Buffer overflow seems to be possible because of the fact that the buffer and the stack grow in opposite directions (stack growing from top to bottom while buffer grow from bottom to top). Imagine that if the CPU and OS are redesigned so that stack now grows from bottom to top (same as buffer). Would this prevent any control flow hijacking attacks completely? If your answer is yes, state your reasoning. If not, give one counterexample. (2 points)

b. To prevent return address from being overwritten, we design a specialized hardware register %SRA (Secure Return Address) that always remembers the latest saved return address on the stack. Whenever the call instruction is executed, besides pushing the return address on the stack, the CPU now automatically saves the return address in %SRA as well. When a function returns, the CPU will automatically compare the return address saved on the stack and the value in %SRA. If they differ, then a buffer overflow attack is detected. If they are the same, then the return address will be moved to %EIP. At the same time, the saved return address for the previous stack frame will be located and saved to %SRA. Is this design secure or not? If not, please elaborate your reasoning. (2 points)

c. Consider another design: for every call instruction, the program pushes an encrypted return address (along with a signature) onto the stack using a secret key stored in a specialized register (which is initialized to a random value every time the program runs). When the return instruction is about to be executed, the program decrypts the return address on the stack using the same secret key stored in the register before actually returning. If the attacker tampers with the return address, it will not match the signature. In other words, the program will catch the tampering of saved return address on the stack. Can this really defend against tampering of saved return address? If so, can you explain how this can be applied to protecting the function pointers? If not, please elaborate your reasoning. (2 points)