CS 370 Introduction to Security **Week 8: Problem Set 8**
Instructor Name: Rakesh Bobba

Introduction/Purpose

The purpose of this assignment is to help you gain a better understanding and insight into software vulnerabilities such as buffer overflows other data and code injection attacks covered in Week 8

Before beginning make sure you have watched the lecture videos on the following and completed the associated practice quizzes.

- Common Software Vulnerabilities
- Input Handling Vulnerabilities
- Buffer Overflow I
- Buffer Overflow II
- OS Interaction Vulnerabilities

Chapter 4.4 from Security Engineering: A Guide to Building Dependable Distributed Systems by Ross Anderson

Instructions/Questions

Please answer the questions below.

Software Vulnerabilities and Injection Attacks

Q1 [4 pts] What is an injection attack? Give 2 examples of injection attacks

- An injection attack is an attack where a user is able to inject text into a program such that the program will interpret that text as code, modifying the expected functioning of the program. A couple of examples of this include:
    - Injecting text into a SQL query that includes a command delimeter character (i.e. ";") so that the text after that is run as a command (i.e. "DUMP TABLE xyz").
    - Providing input "$SHELL ; cat targetfile") to a program that will concatenate your input to "echo " and execute it (exec, eval, system …). This will result in the the target file being printed out as well as the other command completing.

Q 2 [4 pts] Describe what a Cross-Site Scripting attack is. What type of an attack is this?

- XSS is a vulnerability whereupon a user submits text to a site that then gets rendered for other users and unintentionally interpreted and executed as code. This is an injection attack that falls under failure to verify input and output. This often happens in comments sections of websites with that feature as the malicious actor enters their 'text' and when it is rendered for other users, it is actually executed to serve the function the malicious actor intended.

Q3 [4 pts] What is SQL Injection? How can it be prevented?

- SQL Injection is where a user inputs text to a SQL database query, and if the input is not sanitized, then the user can sometimes execute arbitrary SQL commands. This is done by separating the command that it is expected to have been submitted (the query) run with the query and the user’s input (sometimes by terminating it with a “;”) and then writing the malicious actor’s command(s) after that.

Buffer Overflow

Q4 [5 pts] What is a buffer overflow? What are the 3 distinct parts of process memory that buffer overflows typically target?

- A buffer overflow is where a user is allowed to enter more text than can be stored in a given area of memory. This results in the remaining text being written past the buffer.
- Second part of the question is unclear whether referring to memory regions, in which case the answer can be in {stack, globals, env, lib, argv, GOT, ...} but these are less targets, more places that execution will move to after attack (i.e. “target”). Otherwise, the question could refer to what specific parts of the stack are targets of the overflowing of the buffer, to which the answers could be {return address, base pointer of caller, other locals}.

Q5 [3 pts] At a high-level, what are the three steps to exploiting a buffer overflow? What are the possible consequences of a buffer overflow?

- The three steps are:
  1. Inject Code
  2. Hijack Control Flow to Injected Code from [1]
  3. Execute Code Redirected to in [2]
- The consequences of a buffer overflow range from no consequences (i.e. if one buffer is ‘buffered’ away from the return address by another, that remains unused, in which case nothing would change), to a segfault (probably due to invalid return address) to the modification of a local variable, to the modification of EBP/RBP (saved base pointer of caller) to the redirection of code execution.

Q6 [3 pts] What is stack smashing?

- Stack smashing is using a buffer overflow to overwrite things that are present on the stack. Most of the time this involves the overwriting of the return address and seizure of control flow in order to compromise the program.

Q7 [5 pts] What is StackGuard and how does is protect against stack smashing attacks? Can it prevent stack smashing - why or why not? Is StackGuard a compile time or run-time defense?

- StackGuard is a compile time defense that places a canary value right ‘below’ (between the buffer and the base pointer and return address) in order to make it more difficult to overflow the buffer into the base pointer and return address. That said, it cannot completely prevent stack smashing, if there are other vulnerabilities such that the malicious actor can gain access to the value of the canary, then this defense is defeated.

Q8 [2 pts] What is an advantages of runtime defenses against compile-time defenses?

- Run time defenses have an advantage over compile-time defenses when it comes to jumps to shellcode. This is because with some run-time defenses (ASLR, NX), it becomes difficult or impossible to get to or run the shellcode payload that the malicious actor placed there.

Q9 [3 pts] What properties should the *canary value* in StackGuard have?

- It should be unknown to the user, unpredictable (random) and different for each execution and each system. If this value is guessable, then the canary serves no purpose at all, as a malicious actor can get it's value and overwrite it with the correct value as they go about their buffer overflow.

Q10 [5 pts] Name five mechanisms/ways you can think of to protect against buffer overflow attacks.

- Check that buffers are bounded to correct size (manual code / disassembly review)
- Input Fuzzing with a watch on the contents of all buffers and their surrounding address spaces with a focus on long inputs likely to overflow buffers
- Stack canary between buffers and return address and base pointer
- Built in range checking with the use of any buffer (library that includes malloc, write, read, free) that checks for range.
- Use a better language that has built in range checking (python, for example)
- canaries on the end of each buffer no matter its proximity to return address or base pointer.

Q11 [2 pts] What is one advantage of Return Address Defender (RAD) over StackGuard?

- Instead of placing a canary between the buffer, which can be defeated with an arbitrary read (formatstring) vulnerability, the Return Address Defender actually copies a backup of the correct return address to an area unknown to the user, therefore hard to modify. It then can detect stack smashing if the compare fails before it returns to that address. This is better than StackGuard because it is much harder to tamper with given a typical program.

Q12 [3 pts] What is the difference between StackGaurd and Guard Pages?

- Guard pages are zones of memory that will segfault upon being written to that are placed around memory allocations. This is different from the random canary placed by StackGuard, which will fail or pass a compare instruction before returning to the caller.

OS Interaction Vulnerabilities

Q13 [4 pts] What is a TOCTOU error? What is one way to prevent them?

- A TOCTTOU race condition is where the state of the system changes between an availability check against a certain resource, to when that resource is accessed. This can be, for example, a script checks for a lockfile, then tries to open that file. This would encounter a TOCTTOU race condition if between the check for the lockfile and the opening of the file, another program opened that file, then both programs would attempt to access it, creating a race condition.
- To solve this, operations that check then open files depending on the results of the check should be atomic – their execution should complete sequentially, and not allow other commands to run between the check and the open call.

Q14 [4 pts] What is a race condition? Why do they happen?

- A race condition in software development is where two or more resources try to perform conflicting operations on the same data. These operations can include one thread trying to read, one writing to that file, both writing to the file simultaneously, for example. This is not a race condition if both are only reading from the file – in this case neither thread is changing the data.

Q15 [4 pts] How can environmental variables be used for code injection?

- Environment variables can be used for code injection in the example where a program uses the environment variable “CC” to specify the compiler to use without checking it. This could be a vulnerability because where the script expects CC = “gcc” or “clang” (etc.), and it uses that value to run $(“$CC $ARGS”). In this case, an attacker could submit “cat target; gcc” as “CC” to the program, in which case the program would ‘cat target’ with whatever level of permissions it has.

Submission Details

Submit a PDF file with the questions and your corresponding answers

The assignment is worth 55 points. It is due Wednesday of Week 9 at Midnight.