Fill out the bottom of this page with your computing ID.
Write your computing ID at the top of each page in case pages get separated.

Linux X86-64 calling convention reminder:

- first argument: `%rdi`
- second argument: `%rsi`
- third argument: `%rdx`
- fourth argument: `%rcx`
- fifth argument: `%r8`
- six argument: `%r9`
- return value: `%rax`
- return address: on stack
- seventh argument: on stack after return address
- eight argument: on stack after seventh argument

X86-64 registers reminder:

- `%rax` (64-bit), `%eax` (lower 32 bits), `%ax` (lower 16 bits), `%al` (lower 8 bits)
- (and similar for `%rbx`, `%rcx`, `%rdx`)
- `%rsi` (64-bit), `%esi` (lower 32 bits), `%si` (lower 16 bits), `%sil` (lower 8 bits)
- (and similar for `%rbp`, `%rsp`, `%rdi`)
- `%r8` (64-bit), `%r8d` (lower 32 bits), `%r8w` (lower 16 bits), `%r8b` (lower 8 bits)
- (and similar for `%r9` through `%r15`)

AT&T syntax reminder:

- `0x1234(%r9,%r10,4)` = memory at `0x1234` + `%r9` + `%r10` × 4
- `$0x12345678` = constant
- `0x12345678` = memory at `0x12345678`
- source, destination

On X86-64, popping from the stack reads the value located at (%rsp), then adds 8 to the stack pointer.

1. Consider automatically constructing tests each of the following situations. For each, identify whether coverage-guided fuzz testing or symbolic execution is likely to be better able to handle it.

   (a) (2 points) Triggering an assertion failure that requires input to contain a specific 128-bit number.

   ◯ coverage-guided fuzz testing √ symbolic execution

   (b) (2 points) Testing programs that include assembly code with very new instructions that no compiler or program analysis tools understand.

   √ coverage-guided fuzz testing ◯ symbolic execution

   (c) (2 points) Providing assurance that there is no way to trigger an assertion failure for a particular assertion and input length.

   ◯ coverage-guided fuzz testing √ symbolic execution

2. Suppose a website has a cross-site scripting bug where part of the URL is returned in the website itself. For example, a URL like `https://example.com/vulnerable/?text=<script>foo</script>` results in a webpage containing `<script>foo</script>` that the browser will interpret as JavaScript code to run.

   (a) (9 points) How could an attacker exploit this bug to their advantage? **Select all that apply.**

   ◯ The attacker can create a URL with some JavaScript code that directly accesses parts of the website's database (meant to only be accessible to the website's administrators). Then, they can visit that URL in their web browser.

   √ The attacker can create a URL with some JavaScript code that accesses data from a user's browser meant to only be accessible to `example.com`. Then, they can place a reference to this URL on the web site they control.

   ◯ The attacker can create a URL with a shell command that overwrites `example.com`'s main homepage, then visit that URL with their web browser.

   (b) (8 points) What could the website do that is likely to prevent this attack from being effective? **Select all that apply.**

   ◯ Only allow request to `https://example.com/vulnerable/` (with any parameters) if a valid cookie is sent with the request.

   √ Escape HTML in the value before placing it in the output webpage.

   √ Return an HTTP header telling the browser not to run scripts or similar content in the webpage.

   ◯ Return an HTTP header indicating that `https//example.com/vulnerable/` (with any parameters) is HTML to prevent the browser from guessing based on the returned data.

3. (9 points) Suppose a `example.com`'s website's administrative interface has a command to delete a blog post. When an user presses the button to delete a blog post this sends a request to the url `https://example.com/delete-blog-post` with the following information:

   - the URL of the blog post; and
   - a cookie containing the user's session ID

   On receiving this request, the website looks up the session ID in a database to find out their username, then checks the database to see if that user is authorized to delete the blog post. If the user is not authorized: or the session ID cookie missing, then the request fails.

   Which of the following statements are true about this request? **Select all that apply.**

   √ Since the web server checks for the cookie, an attacker who cannot login to the website as an authorized user cannot make a successful deletion request with **their own** web browser.

   √ If the `example.com` website doesn't take precautions, the attacker could load the administrative interface as part of its own website; then, they could trick a user into clicking on the button when the user thinks they are clicking on an innocent link.

   ◯ Although an attacker can cause a user who visits their website to make a request to `https://example.com/delete-blog-post`, they cannot make the user's cookie be included in the request without giving the user some indication that this is happening.

4. (10 points) Consider the following PHP code snippet which runs a SQL query on a database:

```
$result = $dbh->query("SELECT name, description FROM
        users WHERE username = '$username'")
```

   Suppose `$username` represents an unfiltered input value, so this code is vulnerable to SQL injection. The PHP code later outputs the *first result from the query* as part of a webpage.

   Suppose an attacker wanted to use this vulnerability to get a list of all the usernames on the site. Which value for for `$username` would **help** them do this? (You may assume the database in question supports a function called `SUBSTR` in SQL queries and, if invoked below, the function is called correctly.) **Select all that apply.**

   ◯ `SUBSTR(username, 0, 1) = 'a'`

   ◯ `' OR 1 = 1`

   √ `' OR SUBSTR(username, 0, 1) > 'm`

   ◯ `' AND SUBSTR(username, 0, 1) > 'm`

   √ `' OR 'some string' = 'some string`

5. (12 points) Suppose `example.com` allows user to login on `https://example.com/` and also allows each user to control a webpage `https://example.com/users/`*USERNAME*`.html` where *USERNAME* is the user's login name.

   If `example.com` does not filter the HTML file at all and serves all its webpages with headers indicating they are HTML but no other special HTTP headers, what can a malicious user M do to a logged-in victim user V?

   - √ Transfer the values of V's non-HTTP-only cookies for `example.com` to a server M controls if V visits M's userpage.
   - ◯ Transfer the values of all non-HTTP-only cookies stored by V's browser to a server M controls when V visits M's userpage.
   - √ Transfer the values of V's non-HTTP-only cookies for `example.com` to be transferred a server M controls if V visits a website M controls that is appears unrelated to `example.com`.
   - √ Transfer information from V's "user settings" page on `example.com` to a server M controls even though that page can only be retrieved by sending V's HTTP-only cookie to `example.com`
   - √ Automatically submit a form to `example.com` as V when V visits M's userpage, in spite of protections against CSRF attacks.
   - ◯ Extract arbitrary information from `example.com`'s database when M visits their own specially-crafted userpage.

6. Privilege seperation, where a program is divided into parts and one part is run with less privileges, is a common exploit mitigation technique.

   (a) (9 points) Which of the following are true about this technique?

   - ◯ This technique can be deployed on existing, unmodified programs by monitoring what system calls they normally make.
   - √ This technique can take advantage of operating system support for application sandboxing and/or system call filtering.
   - √ This technique is typically implemented using inter-process communication between multiple processes.

   (b) (10 points) Suppose one wanted to add privilege separation to a word processor. Which of the following would be good choices for functionality to be implemented by the **less privileged** part of the word processor? **Select all that apply.**

   - √ Parsing documents from disk or the network.
   - ◯ Performing automatic upgrades of the word processor using patches available online.
   - √ Rendering the text in a document into an image that can be displayed on the screen.
   - ◯ Rendering file open and save dialog boxes and processing user input to them.
   - √ Running scripts embedded in a word processing document that acts as an interactive form.

7. (16 points) Which of the following are ways to detect or prevent some use-after-free vulnerabilities? **Select all that apply.**

   ◯ Use a version of `malloc`/`new` that places guard pages before and after every heap allocation.

   √ Use a fuzz testing tool on the program and look for crashes.

   √ Use a symbolic execution-based test generation tool along with a tool like AddressSanitizer and a version of `malloc`/`new` that never reuses addresses.

   ◯ Enabling write XOR execute.

   √ Using static analysis, track whether pointers are allocated or freed on each possible code path.

   √ By following a rule that objects only ever have one pointer and clearing that pointer when an object is deallocated.

   √ By following a rule that objects track the number of pointers to them and only deallocating objects when that count is zero.

   ◯ By following a rule that objects are only allocated on the stack.

8. (a) (5 points) Which of the following exploit mitigations can be defeated by information leaks? **Select all that apply.**

   ◯ privilege separation

   √ stack canaries (AKA stack cookies)

   √ address space layout randomization of the stack and heap only

   √ address space layout randomization

   ◯ write XOR execute

   ◯ bounds checking as in baggy bounds checking

   (b) (5 points) Which of the following exploit mitigations can often be deployed on unmodified binaries which were not written or compiled or linked with the mitigation in mind? **Select all that apply.**

   ◯ privilege separation

   ◯ stack canaries (AKA stack cookies)

   √ address space layout randomization of the stack and heap only

   ◯ address space layout randomization

   √ write XOR execute

   ◯ bounds checking as in baggy bounds checking

9. Consider the following code:

```
void vulnerable() {
    /* function is a pointer to a function taking a char * argument */
    void (*functionPointer)(char *) = getCurrentFunction();
    char buffer[1024];
    ...
    gets(buffer);
    ...
    functionPointer(buffer);
}
```

Suppose the stack layout of the function is as follows (from highest to lowest address):

- return address of vulnerable (8 bytes)
- stack canary (8 bytes)
- saved value of rbx (8 bytes)
- functionPointer (8 bytes)
- buffer (1024 bytes)

and the stack pointer is located at the beginning of `buffer` just before `vulnerable` calls `gets` or `functionPointer`. The function's use of `gets` permits an attacker who overflows `buffer` by to overwrite `functionPointer` on the stack.

(a) (2 points) To overwrite `functionPointer`, how many bytes from the beginning of the the input should the attacker place the new pointer address?

1024 bytes or 1025 (1-based indices)

(b) (5 points) Suppose the executable uses write XOR execute, so one need to use something like ROP to exploit it. One can overwrite the function pointer with the address of an appropriate gadget. Which of the following gadgets are most likely to be useful for this? (Instructions are separated by semicolons, and only one answer is correct.)

- ◯ `popq %rsp; ret`
- ◯ `movq %rdx, (%rdi); ret`
- √ `popq %rax; ret`
- ◯ `pushq %rdi; pushq %rsp; ret`
- ◯ `jmpq *%rsp`

(c) (5 points) Assuming we have filled in `functionPointer` with the appropriate gadget from the above, how many bytes from the beginning of the buffer should the address of the next gadget to run be placed?

0 bytes or 1 bytes (1-based indices)

10. (12 points) Suppose some malware places its entry point by looking for the bytes that correspond to the machine code for:

```
retq
nopl (%eax,%eax,1)
```

where the `nopl` is an 8-byte no-operation instruction. Then, it replaces it with a relative jmp to virus code inserted elsewhere in the executable. To compute the relative jmp, the malware uses the difference in bytes between the location of the jmp in the executable file and the location of the virus code in the executable file.

Assuming the malware finds the sequence of bytes, which of the following are reasons that the replaced bytes could be **run but cause the target application to crash instead of running the virus code**?

- ◯ The original executable used stack canaries and replacing a return instruction causes the stack canary check to fail.
- √ The malware may have found this sequence of bytes in the middle of others instruction, and corrupted those instruction instead of replacing a return.
- ◯ The inserted jump was placed in a non-executable region of the program.
- √ The malware's strategy for computing the offset of the relative jump might not work when the virus code and jump are in different segments.

This page intentionally left blank.