Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.566 Spring 2026

Quiz I Solutions

Mean 50.0 Standard deviation 13.8

Double-sided quiz: check the back of this page!

I OS/VM isolation [12 points]

Suppose that Ben Bitdiddle discovers a bug in how the Linux file system deals with file deletion and creation: if a file is created with some file name, deleted, and then another file is created with the same file name, the file turns out to contain 4KB of data from a random unallocated disk block, rather than being empty. For example:

```
int fd = open("newfile", O_CREAT | any other flags, 0666);
close(fd);
unlink("newfile");

fd = open("newfile", O_CREAT | any other flags, 0666);
char buf[4096];
int n = read(fd, buf, 4096);
// Surprise: n=4096 and buf contains data from some
// unallocated disk block.
```

1. [6 points]: What isolation/security guarantees could Ben potentially bypass using this bug on a standard Linux system (e.g., the Athena dialup machines)? Be specific about what could go wrong.

Answer: If there was some sensitive data written to a file, but then that file was deleted, its disk blocks would now be unallocated, and Ben could potentially read that data using his bug.

2. [6 points]: For each of the following isolation schemes discussed in lecture, can Ben bypass the isolation guarantees of that scheme using his bug?

(Circle True or False for each choice.)

A. True / False Ben can bypass LXC isolation.

B. True / False Ben can bypass gVisor isolation.

C. True / False Ben can bypass Firecracker isolation.

Answer: Ben can bypass LXC and gVisor, because he can either directly issue those operations to the Linux kernel, or cause gVisor's gofer process to do that. Ben cannot bypass Firecracker because he cannot create and delete files in the host Linux file system.

II BitLocker [7 points]

Ben Bitdiddle considers the following potential attack against his friend's laptop that uses BitLocker: he will steal his friend's laptop and replace the CPU with a modified chip. The modified CPU works just like the original CPU with one difference: when it detects the instruction sequence used by Windows to check the user's password, it executes and produces results as if the password is correct, regardless of whether the password is actually correct. Ben plans to power up the stolen laptop, boot Windows normally, enter "123" as the password, and log in to access the data in the user's account stored on the laptop.

3. [7 points]: Will Ben's attack work? Explain why or why not.

Answer: Yes, the attack will work. BitLocker relies on the TPM and the BIOS to measure the code that runs on the CPU, and Ben's modified system does run the expected code (it just doesn't run it correctly). Thus, the TPM will decrypt the BitLocker key when the laptop boots up, but when Ben types in "123" as the password, Windows will accept it because the CPU executes the password-checking code incorrectly. Another way to view this: the CPU is part of the trusted computing base for BitLocker.

III OpenSSH [7 points]

Consider the paper "Preventing Privilege Escalation" about privilege-separating OpenSSH, and the corresponding lecture.

Ben Bitdiddle wants to modify OpenSSH to allow users to log in without a password when connecting from a specific IP address. (This is probably a bad idea in general, but let's assume that Ben uses a network where it's not possible to spoof connections from another IP address.) Ben's plan is to modify the monitor to add a new type of request, `MONITOR_REQ_AUTHIP`, which works much like `MONITOR_REQ_AUTHPASSWORD`, except that instead of sending the user's password, the worker/slave sends the client's IP address. Ben also modifies the pre-auth worker/slave process to send this request to the monitor.

4. [7 points]: Why is this a bad design given OpenSSH's privilege separation architecture? Propose a better way to achieve Ben's goal.

Answer: This is a bad idea because the untrusted worker/slave process can tell the monitor any IP address it wants, which might not be the actual IP address of the connection. The fix would be for the monitor to remember the connection's IP address when it first accepts the connection, and use that saved IP address when the worker sends `MONITOR_REQ_AUTHIP`.

IV Buffer overflow defenses [14 points]

5. [6 points]: Consider the paper "Baggy Bounds Checking" by Akritidis et al, with the `slot_size` value set to 16 (as in the paper). For the following code, which line will trigger a panic?
(Circle the one best choice.)

```
0 char *p = malloc(40);
1 char *q = p + 46;
2 char *r = q + 24;
3 char *s = r - 20;
4 char  t = *s;
5 char *u = s - 56;
```

A. Line 1.

B. Line 2.

C. Line 3.

D. Line 4.

E. Line 5.

F. No panic.

Answer: No panic: the allocation is of size 64 bytes, rounding up from 40 to a power of 2. The pointer arithmetic goes up to 46+24=70 bytes from the start of the allocation, which is within `slot_size/2`. The dereference happens at 46+24-20=50 bytes from the start of the allocation, which is within the allocation bounds. The arithmetic then goes down to 46+24-20-56=-6 bytes before the start of allocation, which is again within `slot_size/2`.

6. [8 points]: Which of the following attacks are still possible with a fat-pointer scheme as described in lecture?

(Circle True or False for each choice.)

A. True / False Overflowing a buffer on the stack to corrupt the return address.

B. True / False Overflowing a buffer within an allocated array of structs to corrupt other parts of the struct.

C. True / False Overwriting a buffer after it has been freed.

D. True / False Overflowing a heap-allocated buffer to corrupt other heap-allocated memory.

Answer: A: false, B: true, C: true, D: false.

V Web security [18 points]

Ben Bitdiddle includes the following code on his web page at `https://bitdiddle.com`:

```
var req = new XMLHttpRequest();
req.addEventListener("load", function() { console.log("loaded"); });
req.addEventListener("error", function() { console.log("error"); });
req.open("GET", "https://www.bankofamerica.com/account/balance");
req.send();
```

7. [8 points]: Alice installs a fresh web browser on her computer, and visits Ben's site. Which of the following will happen?

(Circle True or False for each choice.)

A. True / False Alice's web browser will send an HTTP request to `bankofamerica.com`.

Answer: True, the browser will send a request to check if `bankofamerica.com`'s Cross-Origin Resource Sharing policy allows such requests.

B. True / False Alice's web browser Javascript console will print `loaded`.

Answer: False, the browser will not allow Ben's page to get the response because the bank would not configure CORS to allow cross-origin requests of an account's balance.

Ben Bitdiddle sets up a blog web site at `https://bitdiddle.com/` where visitors can post and view comments, and Ben uses cookies to keep track of logged-in users (such as himself, which allows him to post new articles). Alyssa figures out that Ben's blog has a cross-site scripting vulnerability, and posts a comment containing a Javascript snippet, as in `Hi Ben, great blog! <SCRIPT>...</SCRIPT>`. Ben visits his blog, from his web browser, by typing in `https://bitdiddle.com/` in his browser's URL bar, after Alyssa posts her comment.

8. [10 points]: Which of the following security mechanisms will prevent Alyssa's attack from being able to post a new article on Ben's behalf?

(Circle True or False for each choice.)

A. True / False Marking the cookie on Ben's site as `HttpOnly`.

Answer: False. Alyssa can still issue HTTP requests to Ben's blog server from her injected script code.

B. True / False Marking the cookie on Ben's site as `Secure`.

Answer: False. Alyssa can still issue HTTP requests to Ben's blog server from her injected script code.

C. True / False Marking the cookie on Ben's site as `SameSite`.

Answer: False. Alyssa can still issue HTTP requests to Ben's blog server from her injected script code.

D. True / False Requiring a CSRF token when posting a new article.

Answer: False. Alyssa can read any CSRF tokens using her injected script code, since it's running in the `bitdiddle.com` origin.

E. True / False Setting content security policy that disables inline scripts.

Answer: True.

VI Symbolic execution [4 points]

9. [4 points]: Ben Bitdiddle changes the STP constraint solver used by EXE such that, if STP was about to time out, it returns a random result (either "satisfiable" or "unsatisfiable") instead of returning a timeout. What effect might this have on bugs that EXE finds, if Ben lets EXE run to completion, and EXE does not require any concretization?

(Circle True or False for each choice.)

A. True / False Ben's modified EXE could find a bug that the original EXE system does not find.

Answer: True: the modified EXE system might randomly encounter a buggy path, which was unreachable with the original EXE due to timeouts.

B. True / False Ben's original EXE could find a bug that the modified EXE system does not find.

Answer: False: any path reachable by the original EXE is also reachable by the modified EXE.

VII Lab 1 [8 points]

Ben Bitdiddle wants to write shellcode that will invoke the `SYS_migrate_pages` system call on Linux, which happens to be syscall number 256 (`0x100` in hexadecimal). He writes the following code in `shellcode.S`:

```
        mov     $SYS_migrate_pages,%ax /* set up the syscall number */
```

but when he compiles it, he discovers that the resulting instruction contains a zero byte in it:

```
$ objdump -d shellcode.o
...
  11:        66 b8 00 01                 mov    $0x100,%ax
...
```

This is a problem because the buffer overflow he was planning to use with this shellcode uses a string-copy function that stops copying data after seeing a zero byte.

10. [8 points]: How can Ben fix his shellcode to get `SYS_migrate_pages` (256) into the `%ax` register without having a zero in his compiled instructions?

Answer: Move 257 into `%ax` and then add a `dec %ax` instruction to decrement it down to 256, or alternatively load 255 and then add a `inc %ax` instruction.

VIII Lab 2 [8 points]

Suppose that Ben Bitdiddle discovers a vulnerability in the final privilege-separated lab 2 implementation, which allows an adversary to see internal network traffic between all of zoobar's containers except for `main`, and does not allow the adversary to modify the network traffic or directly send their own packets to specific containers.

11. [8 points]: Can the adversary leverage this vulnerability to log in as another user into the Zoobar web application? Explain how, or explain why not.

Answer: Wait for the victim user to log in, watch for the user's cookie or password to be sent from the `dynamic` to the `auth` container, use that cookie or password to access the Zoobar website as the victim user.

Double-sided quiz: check the back of this page!

IX 6.566 [2 points]

We'd like to hear your opinions about 6.566. Any answer, except no answer, will receive full credit.

12. [2 points]: Out of the papers that we have covered so far, listed below, mark the one that you think we should remove next year (or mark none if you think all papers should stay).

- ☐ OS/VM isolation: Firecracker, gVisor, comparison study.
- ☐ WebAssembly: wasm design paper, rWasm/vWasm paper.
- ☐ BitLocker.
- ☐ OpenSSH privilege separation.
- ☐ Google security architecture, BeyondProd whitepaper.
- ☐ iOS security.
- ☐ Web security readings.
- ☐ Baggy bounds checking.
- ☐ EXE symbolic execution.
- ☐ Do not remove any papers.

Answer: 24x No paper. 11x Google. 9x Web security. 8x BitLocker. 7x OpenSSH. 4x Baggy. 4x WebAssembly. 2x OS/VM. 1x iOS. 0x EXE.

Also 1x explicit votes to keep BitLocker, OpenSSH, iOS, Baggy, EXE.

End of Quiz