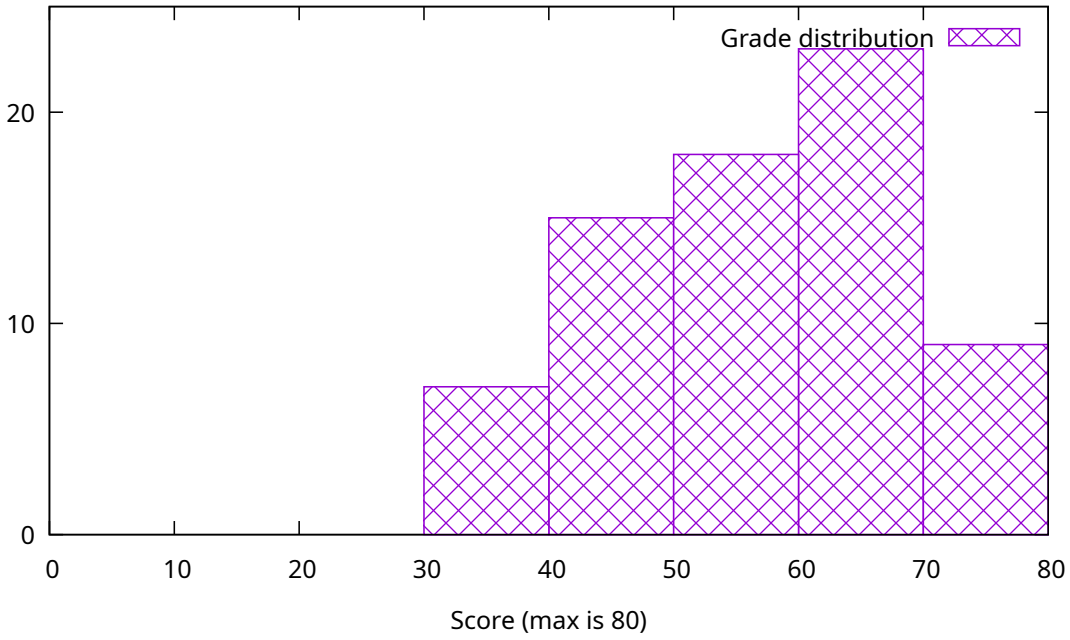
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.566 Spring 2024

Quiz I Solutions

Mean 57.3 Standard deviation 11.3



I OS/VM isolation

An adversary submits a patch to the Linux kernel that introduces a back door in the `ioctl()` system call, where if the system call is invoked with specific arguments, the kernel will change the calling process UID to 0 (root). Suppose this patch gets accepted by Linux and widely deployed. Assume there are no other implementation bugs that an adversary can exploit.

1. [6 points]: Can an adversary use this backdoor to escape from isolation on a system using Firecracker? Explain how (and under what assumptions), or why not.

Answer: No: the guest VM has its own Linux kernel, and the adversary cannot trigger the backdoor ioctl call in the host kernel.

2. [6 points]: Can an adversary use this backdoor to escape from isolation on a system using gVisor? Explain how (and under what assumptions), or why not.

Answer: No: gVisor's sentry process provides its own implementation of ioctl that the sandboxed process gets, and the sandboxed process cannot invoke the host kernel with the backdoor arguments.

3. [6 points]: Can an adversary use this backdoor to escape from isolation on a system using LXC? Explain how (and under what assumptions), or why not.

Answer: Yes: invoke the backdoor ioctl call.

II WebAssembly

4. [12 points]: Suppose that you run a WebAssembly function in a module from an adversary where the function pushes a large number of values onto the stack (more than the compiler/runtime stack has space for). What is the earliest point at which the compiler/runtime can catch this problem, and how would it do so? Assume that the compiler/runtime has no bugs.

Answer: The compiler statically knows the stack depth at every point in the WebAssembly bytecode. If the compiler can tell that the function's stack depth exceeds the maximum possible stack space that will be available at runtime, it could reject the module at compilation time. Additionally, the wasm runtime must generate a runtime check at the entry to each function, to determine if there is sufficient runtime stack space available given the function's stack space requirements. If the compiler does not know the maximum available runtime stack depth at compile time, the error will be caught by this runtime check at function entry time.

III Spectre/Meltdown

Ben Bitdiddle works for a CPU manufacturer and is charged with preventing the original Meltdown attack, "Meltdown-PF," where the CPU speculates that a memory access to kernel memory will succeed, even if it will eventually be rolled back when the CPU discovers the access should not have been allowed. He decides to change the CPU as follows. When a speculatively-executed instruction turns out to have been mis-speculated, and is rolled back, the CPU also evicts any cache lines brought into the cache as part of that speculative operation.

5. [12 points]: Does Ben's design prevent an adversary from using the Meltdown-PF vulnerability to extract data that the adversary should not have access to? Explain why or why not.

Answer: No: adversary can leak information through which other cache line X was evicted when a specific cache line Y was brought into cache by the speculatively-executed code. Even if that cache line Y gets evicted out of the cache during rollback, cache line X will not be brought back into the cache.

IV iOS security

6. [10 points]: A user wants to get data from their broken iPhone, and they have extracted the flash storage chips from it; the rest of the phone doesn't work. The user also manages to get their K_{PIN} (the passcode key from the iOS security paper). Explain how the user can decrypt the data in their flash storage chips, or explain precisely why it's not possible.

Answer: Not possible: need to know K_{fs}, the file system key, which is encrypted with the hardware key (UID), which is on the broken phone. The user would need K_{fs} to decrypt the file system metadata (inodes), since the inode contains the (wrapped) key for decrypting the file contents.

V Baggy

Ben Bitdiddle decides to drop the requirement that out-of-bounds (OOB) pointers must be at most `slot_size/2` past the end of an object from Baggy, and instead allows OOB pointers to be an entire `slot_size` past the end of an object. Consider an application that starts with the following code:

```
char *p = malloc(256);
  // Assume p gets the value 0x1000
char *q = p + 256 + 12;
  // Now q gets the value 0x8000110c
...
```

7. [8 points]: Explain how an application could perform an out-of-bounds write under Ben's design, starting with the above example code. Clearly state any assumptions under which the out-of-bounds write would occur. Assume Ben's variant of Baggy has no implementation bugs (i.e., correctly implements Ben's design), and all of the application's code is compiled with Baggy.

Answer: The value of q is ambiguous: it could be either an overflow or an underflow. The application could now increment q by 4, bringing it in-bounds as a pointer at 0x1110, if there was some other object allocated at that address, and then write to q.

VI Formal verification

Ben Bitdiddle wants to implement a new cryptographic scheme, and outsources the job of writing the code to another developer, who happens to be malicious. Ben asks the developer to write their code in the F* system, and the developer gives Ben an F* file that contains code written in Low*. The file passes the F* type checker, and running Kremlin/Karamel on the file generates a seemingly working C implementation.

8. [8 points]: What should Ben check to make sure there aren't any bugs in the resulting C code? Assume the tools (F*, Kremlin/Karamel, etc) are correct and bug-free.

Answer: Find the Low* definition of the C code that's being generated, and check that there is an F* specification for that Low* code that agrees with Ben's understanding of what the code should be doing.

VII Lab 2

9. [6 points]: An adversary wants to directly modify the database storing their Zoobar balance. What container would they need to break into, and what file (pathname) would they need to modify?

Answer: Break into the bank container and modify `/app/zoobar/db/bank/bank.db`.

VIII 6.566

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

10. [3 points]: Out of the papers that we have covered so far, listed below, circle the one that you think we should definitely 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.
- Transient execution attacks.
- OpenSSH privilege separation.
- Google security architecture, BeyondProd whitepaper.
- iOS security.
- Web security readings.
- Baggy bounds checking.
- EXE symbolic execution.
- HACL* verification.
- Do not remove any papers.

Answer: 23x None. 16x HACL*. 7x Transient execution. 6x OS/VM. 5x Google. 4x Bitlocker. 4x OpenSSH. 3x EXE. 2x iOS. 2x Web security. 1x Baggy.

Suggestions: 3x Bitlocker was superseded by iOS paper. 1x Do OS/VM later on. 1x More recent symbolic execution. 1x More background on transient execution / side channels. 1x Lots of moving components in Google's paper. 1x Wasm was hard to digest so early on. 1x Transient execution attacks already covered in 6.033. 1x Multiple OS/VM papers as the first assignment was rough.

11. [3 points]: What would you like to see improved in this class in the second half of the semester?

Answer: Labs: 7x More guidance (like recitations) on lab assignments. 6x More lab-related guidance in lecture, connection between labs and lectures, review labs in lecture. 1x Better help in setting up lab VM on AWS. 1x Make sure Gradescope works early on. 1x Allow for late submissions on Gradescope by default, without having to ask on Piazza. 1x Modernize Zoobar.

Lectures / topics: 7x More concrete examples / demos / examples in lecture. 5x Answer the assigned reading question. 4x More on modern security applications. 3x Make sure the audio works in lecture recordings. 3x More on attacks / exploits / viruses / malware. 2x Diagrams in lecture notes. 1x Access to demo code. 1x More notes on papers. 1x Lectures on techniques rather than case studies. 1x Focus on confusing aspects of papers in lecture. 1x Compare and contrast systems. 1x Demo attacks on

realistic systems. 1x Find one reading for web security. 1x Brighter lights in 45-230. 1x Connect to specific sections of the paper in lecture notes.

Papers: 4x More exercises / assignments to understand papers (aside from exams). 2x More guidance in reading papers. 1x More motivation / history behind ideas.

Exams: 2x Exam questions should be less detail-oriented. 1x Review session before the final exam. 1x Practice quiz questions for papers that are new this year.

Office hours / Piazza: 5x More office hours, later in the day and when labs are due. 2x Faster responses on Piazza. 1x More clarity on Zoom office hours. 1x More punctual office hours.

End of Quiz