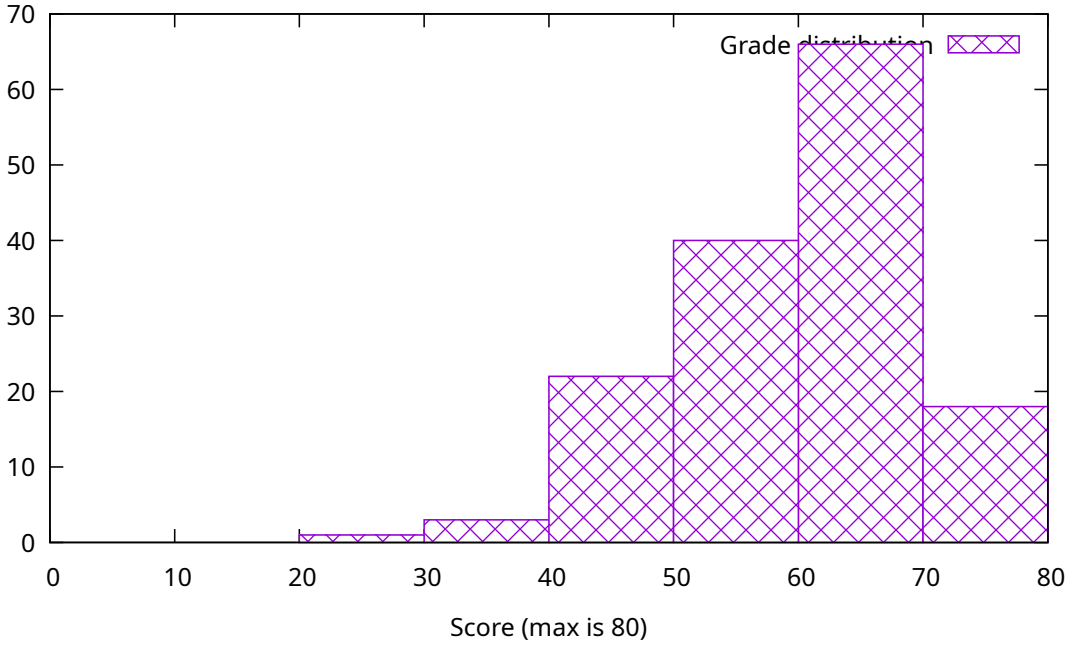
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Spring 2022

Quiz I Solutions

Mean 59.86 Standard deviation 9.55



I Google security architecture

1. [8 points]: Suppose an adversary compromises the storage service responsible for storing the Gmail data for some Google user (call them X), meaning that the adversary's code is already running on that storage service. In which of the following scenarios would the adversary be able to obtain X's Gmail messages?

Circle True or False for each choice. We will give 2 points for a correct answer, 1 point for no answer, and 0 points for the wrong answer.

A. True / False If the inter-service communication configuration allows the adversary to issue RPCs to the compromised storage service.

Answer: False. The storage service requires a user permission ticket (in order to get the decryption key for the user's data), and simply being able to issue an RPC to the service would not provide this ticket.

B. True / False If user X logs into their Google account and accesses another service (such as Google Calendar).

Answer: False. The user's requests would not be sent to the storage server storing the user's Gmail data.

C. True / False If user X logs into their Google account and accesses their Gmail service.

Answer: True. The user's permission ticket would be sent, which gives the compromised storage service access to the decryption key for the user's data.

D. True / False If the adversary compromises the Gmail service in addition to the storage service (but user X never logs in).

Answer: False. No permission ticket.

II U2F

Consider the U2F protocol as described in "Security Keys: Practical Cryptographic Second Factors for the Modern Web." Ben Bitdiddle wants to optimize U2F by not requiring the server to send a randomly-generated challenge during the authentication phase. Instead, in Ben's modified U2F protocol, the client generates its own challenge, uses it in the same way as regular U2F, and sends it to the server along with the signature. The server then uses the challenge sent by the client for verification. Assume that Ben makes no other changes to the server except for using the client-supplied challenge.

2. [9 points]: Describe an attack that an adversary could perform, within the U2F threat model, against Ben's modified U2F which is prevented in the original U2F design. Be specific: describe what the attacker does, what steps they take, and what they achieve.

Answer: The adversary can replay a user's U2F authentication message, since it is not tied to any state held on the server. Thus, an adversary that compromises the user's web browser can record the signed message sent from the U2F key to the server, and replay it from the adversary's machine at a later time to successfully log in as the user.

Strictly speaking, the U2F protocol described in the paper includes a counter value, and a server can require that each login attempt uses a larger counter value than a previous login, to prevent device cloning attacks. However, this is still open to an attack where an adversary with access to the user's browser can get the user's security key to generate multiple signatures (with different counter values), for the same challenge (or, really any challenge known to the adversary). The adversarial browser would then send the signature with the lower counter value to the server, and send the adversary a copy of the signature with a higher counter, allowing the adversary to log in a later point in time, even if the adversary no longer has access to query the user's security key.

III Baggy bounds

Consider the 32-bit version of Baggy Bounds as described in the paper "Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors" by Akritidis et al (plus the errata). Assume a slot size of *256* bytes (instead of 16 as in the paper).

3. [9 points]: An application runs the following code:

```
1 char *q = malloc(130);
2 char *r = q + 300;
3 char *s = r - 100;
4 char a = *s;
5 *r = a;
6 char *n = NULL;
7 *n = a;
```

Assume that the call to `malloc` succeeded. What line will the program crash on, and why?

Answer: The program does not crash on line 2 because the allocated object is 256 bytes (rounding up to the next power of 2), and the resulting pointer is 44 bytes past the end, which is less than slot_size/2=128. The program does not crash on line 3 because that just brings the pointer back in-bounds. Line 4 is an in-bounds read. Line 5 is an out-of-bounds write, and crashes the program.

IV OKWS

Ben Bitdiddle is using OKWS to build his web application, but he is writing all of his services using Rust, a memory-safe programming language that does not allow programmers to make mistakes that lead to buffer overflows. (He avoids any "unsafe" code in his Rust applications, which otherwise could have allowed programmer mistakes to lead to memory corruption.) As a result, he decides he doesn't need to run the services separate from one another: he runs all services in one process, with one connection to the database.

4. [8 points]: Does Ben's choice of running all services together in one process reduce security, given that all of his service code is written in Rust? Explain why the split-service design is no longer needed, or explain what attacks a split-service design might still prevent. Be specific.

Answer: Some kind of logical bug where the adversary can issue arbitrary database queries would be mitigated by a separate-process design that has different database proxies and thus limits the amount of data that an adversary can extract by exploiting such a bug.

V Firecracker

Consider the paper "Firecracker: Lightweight Virtualization for Serverless Applications." For the following scenarios, describe what security goals of the AWS Lambda service running Firecracker an adversary might be able to violate (and how they might be able to do that), or explain why the scenario does not give the adversary the ability to do any additional damage. For each question, assume that this is the only exploitable bug that the adversary knows about.

Circle at most one of "Can violate" or "Cannot violate", and explain.

5. [5 points]: The adversary finds an exploitable buffer overflow in the Linux kernel implementation of the `read()` system call.

Can violate / Cannot violate

Answer: No damage: adversary can compromise the kernel in their AWS Lambda MicroVM, but that's not part of the security boundary anyway. Adversary cannot invoke the read syscall on the host kernel.

6. [5 points]: The adversary finds an exploitable buffer overflow in how the virtio network device implementation in the Firecracker process (say, in the unsafe portions of the Rust code) handles the guest-VM-accessible virtual device memory.

Can violate / Cannot violate

Answer: Adversary can compromise the Firecracker process through this vulnerability. But perhaps the jail still prevents the adversary from doing additional damage.

7. [5 points]: The adversary finds an exploitable buffer overflow in the Linux KVM handler for VM traps.

Can violate / Cannot violate

Answer: Adversary can escape the MicroVM and gain access to the host Linux kernel. The adversary can now access data from other customer MicroVMs.

VI WebAssembly

Consider the paper "Bringing the Web up to Speed with WebAssembly."

8. [10 points]: Alyssa P. Hacker extends WebAssembly to support de-allocating memory, by adding a `shrink_memory` *n* instruction that reduces the allocated memory size by *n* bytes, the opposite of `grow_memory` *n*. How can this change lead to a security problem in a JIT-based WebAssembly runtime that was secure before this change? Assume the runtime executes just one WebAssembly program, once.

Answer: The JIT could emit code that hard-codes the memory size into the generated code. This was safe when memory could not shrink, because checking against a stale size was more strict than necessary. Shrinking memory means such JIT-generated code is no longer safe.

VII iOS security

9. [9 points]: Suppose that Apple suddenly discovers that many iPhones manufactured since 2020 were assigned the same ECID. An adversary physically steals someone's iPhone, with the same ECID as the attacker's phone, and wants to break into it by installing an older version of iOS, from 2015, which has an exploitable vulnerability. Explain how the adversary can perform this attack, or explain why it's not possible.

Answer: The adversary would need to get a signed version of the old iOS with the victim's ECID, which is unlikely to have been known to the adversary when that old version of iOS was still the latest.

VIII EXE

Consider the following code fragment in C in which `stepN` is a symbolic variable, running in the EXE system as described in the paper "EXE: Automatically generating inputs of death".

```
1  char buf[10];
2
3  int f() {
4    int stepN;
5    int sum = 0;
6    int i;
7
8    markSymbolic(&stepN);
9
10   for (i = 0; i < 10/stepN; i++) {
11     sum += buf[i*stepN];
12   }
13   return sum;
14 }
```

10. [10 points]: How many execution paths does EXE explore when executing `f()`?

Answer: One path that divides by zero right away. One path that exits the loop right away (stepN>10 or stepN<0). One path that exits the loop after one iteration (stepN=6,7,8,9,10). One path that exits the loop after two iterations (stepN=4,5). One path that exits the loop after three iterations (stepN=3). One path that exits the loop after five iterations (stepN=2). One path that exits the loop after ten iterations (stepN=1). Total of 7 paths.

IX 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

11. [2 points]: Is there one paper out of the ones we have covered so far in 6.858 that you think we should definitely remove next year? If not, feel free to say that.

Answer: 26x Firecracker; 20x OKWS; 17x Wasm; 12x RLbox; 10x Google; 5x Baggy; 4x MSFT blurb about passwords not mattering; 3x Android; 1x iOS; 43x All papers good.

Recurring comments: Wasm is cool but paper is too PL-focused. Find a modern replacement for OKWS. RLbox, Firecracker, and Wasm didn't connect to labs. iOS paper didn't explain details well enough. Firecracker lacks new ideas, narrow. Would be nice to have a reading guide / background notes / main ideas for each paper (Firecracker in particular). Would be nice to have a lecture on buffer overflows. Google paper was too high-level and too much going on. iOS and Android papers overlap.

End of Quiz