Madison & Ana
Summer 2023

CS 161 Computer Security

Final

Print your name: ______________________, ______________________
(last) (first)

Print your student ID: ______________________________

You have 180 minutes. There are 10 questions of varying credit (200 points total).

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	3	38	16	25	21	12	15	19	28	23	200

For questions with **circular bubbles**, you may select only one choice.

○ Unselected option (completely unfilled)

● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

☐ You can select

■ multiple squares (completely filled)

Pre-exam activity (for fun, not graded):
Does Eve cheat on Battleship?
Circle your answer.

No! Yes...

Q1 *Honor Code* (3 points)

Read the following honor code and sign your name.

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam.

Sign your name:

____

Q2 *True/False* (38 points)

Each true/false is worth 2 points.

Q2.1 A company spends $100,000 on a fancy security camera system to protect its warehouse with $50,000 in inventory.

True or False: The company violated Security is Economics.

● True ○ False

Solution: True

Q2.2 In order to be able to recover their account in case they lose the password, EvanBot gives three of his friends a copy of the password.

True or False: EvanBot violated Ensure Complete Mediation.

○ True ● False

Solution: False, complete mediation is a different concept than this situation.

Q2.3 True or False: The caller function is responsible for pushing the RIP and SFP of the callee function on the stack.

○ True ● False

Solution: False, callee pushes SFP.

Q2.4 True or False: Non-executable pages is an effective defense because it forces attackers to spend a large amount of time digging through the system code to find ROP gadgets.

○ True ● False

Solution: False, ROP compilers can do this automatically.

Q2.5 True or False: ASLR can be an effective defense against a ret2libc attack.

● True ○ False

Solution: True, since the libc addresses are randomized.

Q2.6 TRUE or FALSE: We expect to compute 2^{64} hashes to find a collision in a hash function with a 128-bit output.

[✓ TRUE] FALSE

Solution: True, this is accomplished by the birthday paradox/attack.

Q2.7 TRUE or FALSE: Unlike deterministic RSA from lecture, RSA with OAEP padding provides forward secrecy.

TRUE [✓ FALSE]

Solution: False, OAEP just makes RSA IND-CPA secure, which is not relevant to forward secrecy.

Q2.8 TRUE or FALSE: URL paths are used when determining if two URLs have the same origin.

TRUE [✓ FALSE]

Solution: The path part of the URL is used when determining origin.

Q2.9 TRUE or FALSE: A webpage with a `<img>` tag will make a GET request to display that image.

[✓ TRUE] FALSE

Solution: True, the img tag needs to fetch the resource at that URL, so it makes a GET request.

Q2.10 TRUE or FALSE: Unlike stored XSS, reflected XSS often requires users to click on attacker-provided links.

[✓ TRUE] FALSE

Solution: True, since stored XSS is served as a response to a normal user request like loading Facebook, whereas reflected XSS usually requires clicking on attacker links.

Q2.11 True or False: Enabling a Content Security Policy completely disables the use of inline JavaScript.

[✓ True] False

Solution: True, by definition of CSP.

Q2.12 True or False: VLANs are useful for defending against ARP attacks.

[✓ True] False

Solution: True, VLANs isolate the local network into different VLANs making it harder to execute ARP attackers.

Q2.13 True or False: We can tell whether a device is in our LAN, based on our network's subnet and the device's IP address.

[✓ True] False

Solution: True, by checking whether the IP address has the subnet as a prefix.

Q2.14 True or False: Switches are an effective defense against DHCP attacks.

True [✓ False]

Solution: False, switches generally only defend against ARP attacks.

Q2.15 True or False: In BGP, a malicious AS could claim to be responsible for a network it isn't responsible for.

[✓ True] False

Solution: True, there is no way to verify that an AS is legitimate.

Q2.16 True or False: UDP is vulnerable to RST injection.

True [✓ False]

Solution: False, UDP does not have RST flags like TCP does.

Q2.17 True or False: In TCP, both the server side and client side ISNs should be different per connection.

[✓ True] False

Solution: True, to help defend against replay attacks.

Q2.18 True or False: In DNSSEC, it is possible to sign groups of DNS records at once.

[✓ True] False

Solution: True, by RRSIG.

Q2.19 True or False: Metamorphic code re-encrypts code with a new key upon propagation.

True [✓ False]

Solution: False, metamorphic code generally changes its own semantic, whereas polymorphic code is the one that recrypts.

Q2.20 (0 points) True or False: `EvanBot is a real bot.`

[✓ True] False

Solution: `True. The proof doesn't fit in this margin tho.`

Q3 *Memory Safety Exploit: Letter from EvanBot* **(16 points)**

```
1 void writeLetter(uint8_t size) {
2     if (size >= 100) {
3         return;
4     }
5     char letter[100];
6     fread(letter, size - 1, 1, stdin);
7     printf("Letter From Evanbot: %s", letter);
8 }
```

Stack at Line 6
(1)
RIP of `writeLetter`
(2)
(3)

- All memory safety defenses are disabled (unless otherwise specified).
- You run GDB once and find that the `letter` buffer is located at the address `0xffffad04`.
- You may use `SHELLCODE` as a 64-byte shellcode.

Q3.1 (2 points) Assume the program is paused at a breakpoint on line 6. What values go in blanks (1), (2), and (3) in the stack diagram above?

[✓ (1) `size` (2) SFP of `writeLetter` (3) `letter`]

○ (1) SFP of `writeLetter` (2) `size` (3) `letter`

○ (1) `letter` (2) SFP of `writeLetter` (3) `size`

Q3.2 (3 points) Which vulnerability is present in the code?

○ Off-by-one [✓ Integer overflow/underflow]

○ Signed/unsigned ○ None of the above

Q3.3 (8 points) Provide inputs for `size` and `fread` that cause the program to execute shellcode.

`size`:

Solution: 0

`fread` on Line 6:

Solution: `SHELLCODE + 'A' * 40 + \x04\xad\xff\xff + EOF`

Solution: This exploit is centered around a integer underflow exploit, since `size` is an unsigned integer. If we pass in 0 for `size`, then subtracting one on Line 7 causes it to wrap-around to $2^8 - 1$.

We can then pass in SHELLCODE (64 bytes) into the buffer, 36 + 4 bytes of garbage to move up to the RIP, then overwrite the RIP with the address of `letter`.

Q3.4 (3 points) **For this subpart only,** assume that non-executable pages (W^X) are enabled and that the program includes a large amount of library code.

Give the name of a type of exploit from class that could be used to execute shellcode. Your answer can be 10 words or fewer.

Solution: Return-oriented programming can bypass W^X .

Q4 *Memory Safety Exploit: The Last Dance* (25 points)

Consider the following code:

```
 1 typedef struct {
 2     char *buf;
 3     int address;
 4 } message;
 5
 6 void f(int addr, char *contents) {
 7     char text[4];
 8     fgets(text, 13, stdin);
 9 }
10
11 void main() {
12     message m;
13     m.buf = (char *) malloc(128);
14     fgets(m.buf, 128, stdin);
15     fread(m.address, 4, 1, stdin);
16     f(m.address, m.buf);
17 }
```

Stack at Line 8

RIP of `main`
SFP of `main`
`m.address`
(1)
(2)
(3)
(4)
(5)
(6)

- You may use `SHELLCODE` as a 120-byte shellcode.
- **ASLR is enabled, not including the code segment**.
- Unless otherwise specified, all other memory safety defenses are disabled.

You use GDB to inspect the code segment of the program and find the following x86 assembly at the corresponding addresses:

```
0x81975ff: nop
0x8197600: ret
```

For the following two subparts, one variable should go in each row of the stack diagram. Assume the program is paused at a breakpoint on line 8.

Q4.1 (2 points) What values go in blanks (1), (2), and (3) in the stack diagram above?

- ○ (1) `m.buf` (2) `addr` (3) `contents`
- [✓ (1) `m.buf` (2) `contents` (3) `addr`]
- ○ (1) `addr` (2) `contents` (3) `m.buf`

Q4.2 (2 points) What values go in blanks (4), (5), and (6) in the stack diagram above?

[✓ (4) RIP of `f` (5) SFP of `f` (6) `text`]

○ (4) `text` (5) RIP of `f` (6) SFP of `f`

○ (4) SFP of `f` (5) RIP of `f` (6) `text`

Solution:

`RIP of main`
`SFP of main`
`m.address`
`m.metadata`
`m.buf`
`data`
`contents`
`addr`
`RIP of f`
`SFP of f`
`text`

The code, reprinted for your convenience:

```c
typedef struct {
    char *buf;
    int address;
} message;

void f(int addr, char *contents) {
    char text[4];
    fgets(text, 13, stdin);
}

void main() {
    message m;
    m.buf = (char *) malloc(128);
    fgets(m.buf, 128, stdin);
    fread(m.address, 4, 1, stdin);
    f(m.address, m.buf);
}
```

Q4.3 (10 points) Provide inputs to the program that will execute shellcode with 100% probability.

Input to `fgets` on Line 8 (for `text`):

Solution: `8 * 'A' + '\x00\x76\x19\x08' + '\n'`

Input to `fgets` on Line 14 (for `m.buf`):

Solution: `'SHELLCODE' + '\n'`

Input to `fread` on Line 15 (for `m.address`):

Solution: `'\x00\x76\x19\x08' + '\n'`

Solution:

`fgets` on Line 8 (for `text`):

This call will be structed as follows: 4 bytes of garbage to fill the `text` buffer, and 4 more bytes of garbage to fill the `SFP of f`, followed by a pointer to a `ret` instruction to overwrite the `RIP of f`. When the function returns, the eip register will point to the `ret` instruction.

Note that `fgets` will append a null terminator, which will overwrite the LSB of `addr`.

When the `ret` instruction is executed, the esp register will be pointing to the `addr` variable and pop this value off the stack into the eip register.

`fgets` on Line 14 (for `m.buf`):

`m.buf` is a buffer that was allocated on the heap. This is the only place that can fit `SHELLCODE`, so we must place it here. Additionally, we have the `contents` pointer (copy of `m.address`) which points to `m.buf`. This will come in handy for us later when we need a pointer to `SHELLCODE` (Recall: ASLR is enabled, so we can't use GDB to find the `SHELLCODE` pointer during our exploit).

`fread` on Line 15 (for `m.address`):

We want to form a `ret` chain such that the `SHELLCODE` pointer ends up in the eip register. To accomplish this, we placed a `ret` instruction in the `RIP of f`, but now we need a second `ret` instruction where `addr` is in order to fill the gap between the `RIP of f` and our `SHELLCODE` pointer, `contents`. Thus, we place a pointer to a `ret` instruction into `m.address`, which will be passed into `f` as the argument `addr`.

Note that we must use this specific address `\x00\x76\x19\x08` because the LSB is a null byte. When the `fgets` call on line 8 is executed, this null terminator will overwrite the LSB of `addr`, so we must choose this instruction to ensure that `addr` still points to a `ret`.

Q4.4 (2 points) **For this subpart only**, assume that the x86 instructions you find are instead located at these addresses:

```
0x8197601: nop
0x8197602: ret
```

True or False: You can construct a similar exploit to the one constructed earlier in this question that would cause the shellcode to be executed.

○ True [✓ ● False]

Solution: If the x86 instructions are moved to these new addresses, the exploit is no longer possible. This is because we need a `ret` instruction (or a `nop` sled leading to a `ret` instruciton) to be at a address that a least significant byte (LSB) of `0x00`. This is because `fgets` call on Line 8 will append a null byte to the LSB of `addr` and we need `addr` to be pointing to a `ret` instruction. If there are no `ret` instructions ending with `0x00`, then `addr` can never point to a `ret` instruction.

Q4.5 (4 points) **For this subpart only**, assume that non-executable pages (W^X) are enabled (in addition to ASLR). Assume that the stack is set as writable and the heap is set as executable.

True or False: The exploit from earlier in this question would still cause the shellcode to be executed.

○ True [✓ ● False]

Justify your selected answer. Your answer can be 15 words or fewer.

Solution: If we enable non-executable pages such that the heap is set as executable, then this means the heap cannot be written to. However, the exploit in this question requires placing `SHELLCODE` on the heap (no input to the stack is large to hold `SHELLCODE`). Thus, the exploit from earlier in the question is no longer possible.

Q4.6 (5 points) **For this subpart only**, assume that the x86 assembly you find is instead these instructions:

```
0x81975ff: pop %eax
0x8197600: ret
```

Also, **for this subpart only**, assume that line 15 is deleted from the program.
True or False: You can construct a similar exploit to the one constructed earlier in this question that would cause the shellcode to be executed.

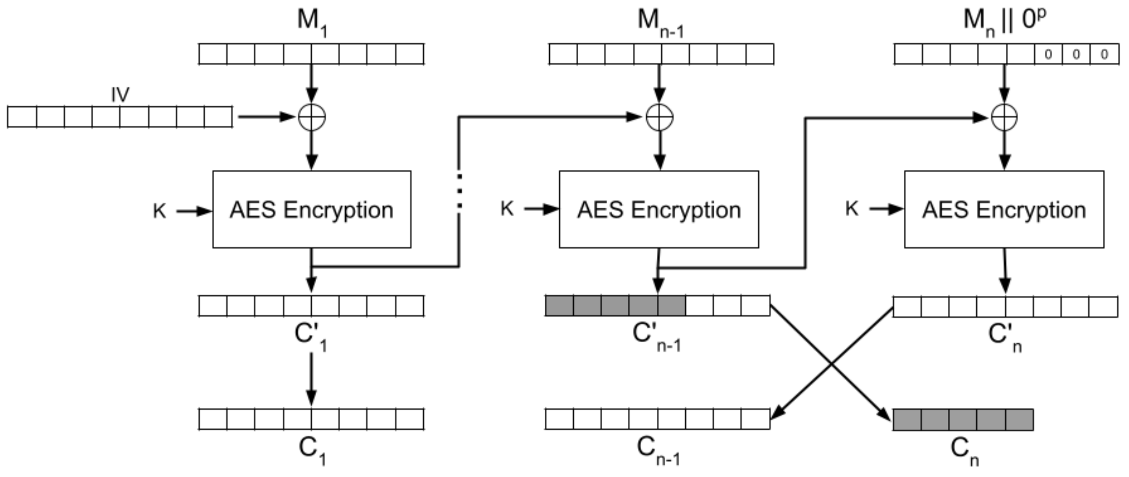
[✓ True] False

Justify your selected answer.

Solution: If we remove line 15, then the attacker has no control over the value of `addr` and is thus not able to complete a full `ret` chain that leads to `contents`. However, since the `nop` instruction has been replaced with a `pop` instruction, we can have `addr` popped off the stack before the `ret` instruction runs (by overwriting the RIP of `f` with `0x8197601`). Now, the bottom of the stack will be a pointer to shellcode when the `ret` instruction runs, meaning shellcode will execute.

Q5 *Cryptography: Ciphertext Thief* **(21 points)**

Alice and Bob invent a new mode of operation named AES Stolen Cipher Text, or AES-SCT.



For this entire question, assume that each AES block is 16 bytes.

C`[:x]` denotes the first x bytes of the block C, and C`[x:]` denotes everything but the first x bytes of C.

0^x denotes a sequence of the byte 0 repeated x times.

To encrypt:

1. Pad the message with p zero bytes such that the message length is a multiple of the block size.
2. Encrypt the message with AES-CBC, getting temporary ciphertext $C' = (IV, C'_1, \dots, C'_{n-1}, C'_n)$.
3. Remove the last p bytes of C'_{n-1} and then swap C'_{n-1} and C'_n.
   The final ciphertext is $C = (IV, C'_1, \dots,\ C'_n,\ C'_{n-1}$`[:16-p]`$)$.

To decrypt:

1. Perform AES decryption on C_{n-1} to recover $T =$________________.
2. Set $C_n =$ ________________.
3. Swap C_{n-1} and C_n to get C'.
4. Decrypt C' using AES-CBC.

Q5.1 (2 points) Fill in the first blank.

[✓ $C'_{n-1} \oplus (M_n \parallel 0^p)$] ○ $C'_{n-1} \oplus (M_n \parallel 0^{16-p})$

○ $C'_n \oplus (M_{n-1} \parallel 0^p)$ ○ $C'_n \oplus (M_{n-1} \parallel 0^{16-p})$

Solution: Decrypting $C_{n-1} = C'_n$ gives us $C'_{n-1} \oplus (M_n \| 0^p)$ by using the AES-CBC formulas.

Q5.2 (2 points) Fill in the second blank.

● $C_n \parallel T$`[16-p:]`   ○ $C_n \parallel T$`[:p]`

○ $C_{n-1} \parallel T$`[16-p:]`   ○ $C_{n-1} \parallel T$`[:p]`

Solution: We want to fill in the rest of C_n, so we take the last p bytes of the decrypted result T. Since the M_n input had zeroes in those bytes, we don't have to worry about removing M_n.

Q5.3 (3 points) Is AES-SCT IND-CPA secure?

● Yes, because AES-CBC is secure and we are simply swapping blocks around.

○ Yes, because padding with zeroes is an acceptable way to pad block ciphers.

○ No, because the attacker can see the exact length of the plaintext instead of the length rounded to the next block multiple.

○ No, because zero-padding is not secure, and we would need to use PKCS#7 padding to prevent padding-oracle attacks.

Solution: AES-SCT is just AES-CBC with a few extra swaps at the end, and since we ignore length in IND-CPA, AES-SCT is secure as AES-CBC.

Q5.4 (2 points) Select all true statements about AES-SCT.

☐ Encryption is parallelizable.   ■ Decryption is parallelizable.

☐ None of the above

Solution: Since AES-SCT is just AES-CBC with a few "setup" steps that are constant time, we can just use the fact that AES-CBC is parallelizable to decrypt but not encrypt.

Q5.5 (3 points) Suppose we change one bit in C_{n-1}. What happens to the decrypted message?

- ○ M_{n-1} becomes unpredictable garbage, but M_n only differs in a single bit.
- [✓ ● Both M_{n-1} and M_n become unpredictable garbage.]
- ○ None of the above

Solution: Both blocks become garbage, since both decryption formulas rely on passing C_{n-1} through an AES block cipher.

Q5.6 (3 points) Suppose we change one bit in C_n. What happens to the decrypted message?

- [✓ ● M_{n-1} becomes unpredictable garbage, but M_n only differs in a single bit.]
- ○ Both M_{n-1} and M_n become unpredictable garbage.
- ○ None of the above

Solution: M_{n-1} becomes unpredictable garbage, but M_n only differs in a single bit.

Q5.7 (3 points) Select the best option in favor of using AES-SCT over AES-CTR.

- ○ AES-SCT encryption is faster to encrypt with than AES-CTR encryption.
- [✓ ● AES-SCT is more resistant to IV reuse than AES-CTR.]
- ○ AES-SCT is IND-CPA secure and AES-CTR is not.
- ○ AES-SCT does not require padding like AES-CTR does.

Solution: AES-SCT is more resistant to IV reuse, since AES-SCT IV reuse (much like AES-CBC) only leaks the fact that certain messages have the same prefix, but nothing beyond that. AES-CTR IV reuse leaks the XOR of the two messages, which is strictly more information.

Q5.8 (3 points) What is the length (in bytes) of the shortest plaintext that can be encrypted with AES-SCT?

○ 1 [✓ 17] ○ 33 ○ 49

Solution: Since we swap the last two blocks, we need to have at least 2 blocks of ciphertext, which 17 bytes is (16 for the first and 1 for the second).

Q6 *Cryptography: EvanBot Signature Scheme* (12 points)

EvanBot decides to make a signature scheme!

To initialize the system, a Diffie-Hellman generator g and prime p are generated and shared to all parties. The private key is some $x \bmod p$ chosen randomly, and the public key is $y = g^x \bmod p$.

To sign a message m such that $2 \leq m \leq p - 2$:

1. Choose a random integer k between 2 and $p - 2$.
2. Set $r = g^k \bmod p$.
3. Set $s = (\mathsf{H}(m) - xr)k^{-1} \bmod (p - 1)$. If $s = 0$, restart from Step 1.
4. Output (r, s) as the signature.

Clarification after exam: k is chosen to be coprime to $p - 1$.

To verify, check that $g^{H(m)} \equiv$ ______ $\bmod p$. We will fill in this blank in the next few subparts.

Q6.1 (3 points) Select the correct expression for $H(m)$ in terms of x, r, k, s and $p - 1$.
HINT: Use Step 3 of the signature algorithm.

○ $k(xr)^{-1} + s \bmod (p - 1)$

○ $k^{-1} + xr \bmod (p - 1)$

○ $ks - xr \bmod (p - 1)$

[✓ $ks + xr \bmod (p - 1)$]

Solution: From step 3:

$$\begin{aligned} s &\equiv (\mathsf{H}(m) - xr)k^{-1} \bmod (p - 1) \\ sk &\equiv \mathsf{H}(m) - xr \bmod (p - 1) \\ sk + xr &\equiv H(m) \bmod (p - 1) \\ H(m) &\equiv ks + xr \bmod (p - 1) \end{aligned}$$

Q6.2 (4 points) Using the previous result, select the correct value for the blank in the verification step. *HINT: Replace the $H(m)$ in $g^{H(m)}$ with your results from the previous subpart.*

○ $y^s r^2 \bmod p$

○ $r^y r^s \bmod p$

[✓ $y^r r^s \bmod p$]

○ $r g^{yr} \bmod p$

Solution:

$$\begin{aligned} &g^{ks+xr} \bmod p \\ &\equiv g^{ks} \cdot g^{xr} \bmod p \\ &\equiv (g^k)^s \cdot (g^x)^r \bmod p \\ &\equiv r^s y^r \equiv y^r r^s \bmod p \end{aligned}$$

Q6.3 (5 points) Show how to recover the private key x if a signature is generated such that $s = 0$ (i.e. the check on Step 3 is ignored).

Solution: If $s = 0$, then $0 \equiv (H(m) - xr)k^{-1} \bmod (p-1)$ per Step 3, which means $xr = H(m) \bmod (p-1)$ and we can solve for x. Note that k is implicitly coprime to $p-1$ by construction in the protocol.

Q7 *Web: Barbenheimer* (15 points)

Alice has recently set up a movie theater, and asks for your help setting up their ticket system.

```
CREATE TABLE tickets (
    customerName TEXT,
    movieName TEXT
    -- Additional fields not shown.
);
```

To find all the movies that a customer has a ticket for, Alice's website runs the following query (directly replacing `$customerName` with the user provided input):

```
SELECT movieName FROM tickets
WHERE customerName = '$customerName';
```

Q7.1 (3 points) Mallory wants to learn the name of all movies that have at least one ticket purchased.

Provide an input for `$customerName` in the above query that would return the name of every movie with at least one ticket for it in the `tickets` table.

`$customerName`:

Solution: `' OR 1 = 1 --` will cause all rows to be printed.

Now consider a different situation. Alice's website contains a page that takes in a movie name, runs the following query, and displays "`N tickets sold`" to the user (where `N` is the number of rows returned).

```
SELECT customerName FROM tickets
WHERE movieName = '$movieName';
```

Q7.2 (5 points) Provide an input to `$movieName` that Mallory could use to determine if Bob has a ticket for the movie "`Barbie`".

`$movieName`:

Solution: `Barbie' AND customerName = 'Bob' --`

Explain how the above query allows you to learn if Bob has a ticket for Barbie.

Fill in the blank: "If ________ then Bob has a ticket for Barbie, otherwise he does not."

Solution: "If the website displays "`1 tickets sold`" then Bob has a ticket for Barbie, otherwise he does not."

Alice releases a quick code patch. Now the page always displays `"Ticket count not available"` rather than `"N tickets sold"`. *However, the original query still runs before the this new message is displayed to the user.*

Q7.3 (7 points) Mallory realizes that when she provides "Oppenheimer" as the `$movieName` input, Alice's website consistently takes less than 2 seconds to display the message `"Ticket count not available"`.

Using this information, provide an input to `$movieName` that Mallory could use to determine if Bob has a ticket for "Oppenheimer".

Assumptions:

- Alice's SQL server has a `SLEEP(X)` command that pauses the program for `X` seconds and then returns the number 0.
- Alice's SQL server correctly implements logical short-circuiting.
  For example, `"func1() OR func2()"` does not execute `func2` if `func1` returns `TRUE`.

`$movieName`:

Solution: `Oppenheimer' AND customerName = 'Bob' AND SLEEP(10) = 0 --`

Explain how the above query allows you to learn if Bob has a ticket for Oppenheimer.

Fill in the blank: "If ________ then Bob has a ticket for Oppenheimer, otherwise he does not."

Solution: If the website sleeps for 10 seconds, then Bob has a ticket for Oppenheimer, otherwise he does not.

Q8 *Passwords and Web: EvanBank* (19 points)

EvanBot is interested in upgrading the EvanBank website to use better password security. Unfortunately, an attacker has leaked the entire passwords database.

H is a cryptographically secure hash function. K is a symmetric key known only to the server, not the attacker. There are N possible passwords and the attacker can compute N hashes.

For each subpart, select all true statements.

Q8.1 (3 points) For each user, EvanBank stores `username` and H(`password`).

■ An attacker can recover all user passwords.

■ An attacker can see whether two users have the same password.

■ An attacker can recover the password of any one user.

■ An attacker can overwrite one user's password with that of another user.

■ An attacker can overwrite passwords arbitrarily.

☐ None of the above

Solution: The attacker can compute all possible hashes per the assumptions, and there is no integrity on the entries. Therefore all results are correct.

Q8.2 (3 points) For each user, EvanBank stores `username` and HMAC(K, `password`).

☐ An attacker can recover all user passwords.

■ An attacker can see whether two users have the same password.

☐ An attacker can recover the password of any one user.

■ An attacker can overwrite one user's password with that of another user.

☐ An attacker can overwrite passwords arbitrarily.

☐ None of the above

Solution: Since the attacker doesn't know K, they cannot bruteforce the HMAC to learn the password, nor can they forge a new HMAC. However, since HMAC is deterministic, they can see whether two entries are the same. They can also copy over one entry into another, since the copied entry is still a valid HMAC.

Q8.3 (3 points) For each user, EvanBank stores `username` and H(`username` || `password`).

☐ An attacker can recover all user passwords.

■ An attacker can see whether two users have the same password.

■ An attacker can recover the password of any one user.

■ An attacker can overwrite one user's password with that of another user.

■ An attacker can overwrite passwords arbitrarily.

☐ None of the above

Solution: The username is effectively a salt here, so the attacker cannot bruteforce all possible username/password combos. However, since the username is public, they can bruteforce a single row. Since there is no integrity, they can also replace passwords with whatever they want.

Q8.4 (2 points) Assume that a cookie has the field `Domain=evanbank.com`, `Path=/account`, and `Secure=True`. Which of the following URLs would this cookie be sent to, according to cookie policy? Select all that apply.

☐ `http://evanbank.com/account`

■ `https://coda.evanbank.com/account`

■ `https://evanbank.com/account/information`

☐ `https://evanbank.org/account`

☐ None of the above

Solution: `http://evanbank.com/account` is not HTTPS, which is required by the Secure flag.

`https://coda.evanbank.com/account` and `https://evanbank.com/account/information` meet all requirements (Domain and Path still match for more specific subdomains / subpaths than the cookie values).

EvanBank's website has an open redirect endpoint at `www.evanbank.com/redirect?url=$URL` that immediately redirects the user to `$URL`. Assume that there are no XSS vulnerabilities on `www.evanbank.com`.

Q8.5 (4 points) Select all attacks that would be possible if a user clicked on an open redirect link sent by the attacker.

■ Make a GET request to `www.evil.com`.

■ Make a POST request to EvanBank.

■ Execute a CSRF attack.

☐ Execute JavaScript with the origin of `www.evanbank.com`.

☐ None of the above

Solution: We can redirect to other pages, which causes a GET request. We can also redirect to an attacker-controlled page that runs a script to make a POST request, but the same-origin policy stops this. We can't execute JS since there are no XSS attacks.

Q8.6 (4 points) Which of the following are benefits of tricking a user into clicking an open redirect, instead of the target URL directly? Select all that apply.

■ Users are less likely to notice that they are clicking on a malicious URL.

☐ Open redirects allow an attacker to exploit browsers that would otherwise block requests to malicious websites.

■ Open redirects can circumvent Referer validation.

☐ Open redirects can circumvent input sanitization.

☐ None of the above

Q9 *Networking: Passwords-R-Us* **(28 points)**

BotLabs wants to let their users authenticate to their website. However, BotLabs does not trust itself to do password management, so it outsource this responsibility to the company Passwords-R-Us.

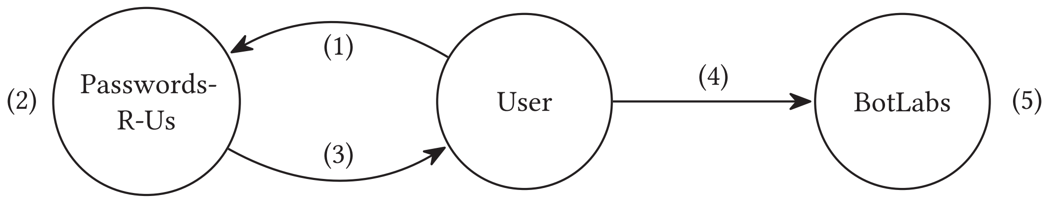
Setting:

- Passwords-R-Us stores a mapping of each user's IP address to their chosen password.
- BotLabs and Passwords-R-Us both know a single, constant value `secret` that nobody else knows. `secret` is a randomly chosen value from a public pool of 10 million options.
- Everyone knows Passwords-R-Us's public key PK. Only Passwords-R-Us knows the private key.

Assumptions:

- Each user has only one IP address and it never changes.
- Each IP address is associated with exactly one user.
- All attackers are not users on BotLabs or Passwords-R-Us, and cannot create new users.
- All attackers cannot send a brute-force number of IP packets.

When a user wants to authenticate to BotLabs's website, they follow these steps:



(1) The user sends an IP packet containing Enc(PK, password) to Passwords-R-Us, where Enc is deterministic RSA encryption without OAEP.

(2) Passwords-R-Us compares the decrypted password to the password in their database associated with the source IP address.

(3) If the password is correct, then Passwords-R-Us generates a random nonce and sends back (nonce, H(`secret` || nonce)) to the user.

(4) The user sends an IP packet containing (nonce, H(`secret` || nonce)) to BotLabs.

(5) BotLabs hashes `secret` (which they already know) with the nonce sent by the user and checks that it matches the hash sent by the user. If so, BotLabs adds the sender's IP address to an authenticated list.

Eve is an on-path an attacker with infinite computational power.

Q9.1 (3 points) Is Eve able to execute an online brute-force attack?

- ○ Yes, because Eve is able to spoof packets.
- ○ Yes, because Eve has infinite computational power.
- ● No, because Eve cannot send a brute-force number of IP packets.
- ○ No, because Eve has no way to verify that their guess is correct.

Solution: Online brute-force attacks would require sending a ton of packets, so even if Eve has offline bruteforce capability the network/end server does not have that capacity.

Q9.2 (3 points) Select all types of attackers that are able to execute an offline brute-force attack.

- ☐ Off-path attacker
- ■ MITM attacker
- ■ On-path attacker
- ☐ None of the above

Solution: Off-path attackers can't see the values in the sent packets, and therefore can't check if their guess is correct.

Q9.3 (4 points) Which of these values, if obtained individually, would allow Eve to execute an offline brute-force attack? Select all that apply.

- ■ Enc(PK, password)
- ☐ nonce
- ■ H(`secret` || nonce)
- ■ (nonce, H(`secret` || nonce))
- ☐ None of the above

Solution: All of the selected values allow us to check whether we guess the correct password or the correct `secret`. Just having the nonce does not let us check if we guess correctly.

Mallory is an on-path attacker without brute-force computational power. She can observe all packets sent and received by EvanBot, and she has previously observed the entirety of EvanBot's authentication process.

Mallory wants to become authenticated on BotLabs's website. Her attack will involve sending one IP packet, receiving one IP packet in response, and then sending a second IP packet.

Q9.4 (3 points) What is the source field of Mallory's first packet?

○ Mallory's IP address ○ Passwords-R-Us's IP address

[✓ EvanBot's IP address] ○ BotLabs's IP address

Solution: Mallory wants to spoof a packet as if it came from EvanBot, so she puts EvanBot's IP in the source field.

Q9.5 (2 points) What is the destination field of Mallory's first packet?

○ Mallory's IP address [✓ Passwords-R-Us's IP address]

○ EvanBot's IP address ○ BotLabs's IP address

Solution: Mallory wants to send this spoofed packet to Passwords-R-Us. In the next subpart, we will fill in the data with a previous correct login packet.

Q9.6 (3 points) Which previous IP packet should the contents of Mallory's first packet be copied from?

[✓ The first IP packet from EvanBot to Passwords-R-Us]

○ The first IP packet from Passwords-R-Us to EvanBot

○ The first IP packet from EvanBot to BotLabs

Solution: The Enc(PK, password) value observed from EvanBot's authentication.

Q9.7 (3 points) What is the source field of Mallory's second packet?

[✓ Mallory's IP address] Passwords-R-Us's IP address

EvanBot's IP address BotLabs's IP address

Solution: Now that Mallory has learned the value of the Passwords-R-Us response, we want to use it to login to BotLabs.

Q9.8 (2 points) What is the destination field of Mallory's second packet?

Mallory's IP address Passwords-R-Us's IP address

EvanBot's IP address [✓ BotLabs's IP address]

Solution: Now that Mallory has learned the value of the Passwords-R-Us response, we want to use it to login to BotLabs.

Q9.9 (3 points) Which previous IP packet should the contents of Mallory's second packet be copied from?

The first IP packet from EvanBot to Passwords-R-Us

The first IP packet from Passwords-R-Us to EvanBot

The first IP packet from EvanBot to BotLabs

[✓ The response to Mallory's first IP packet]

Solution: The (nonce, H(secret || nonce)) returned by BotLabs. These are the contents of the reply to the first IP packet that Mallory sent.

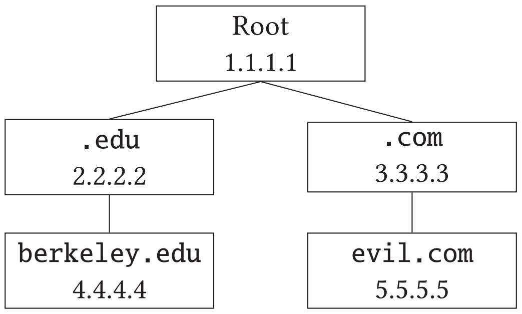
Q9.10 (2 points) If Mallory was an off-path attacker, would the attack be possible?

Yes [✓ No]

Solution: No, because Mallory can no longer observe the reply to Mallory's first packet (which is sent to EvanBot, because it was spoofed).

Q10 *Networking: DNS Hack Finding* (23 points)

Consider the following DNS name server hierarchy:



Setting:

- Mallory controls the `www.evil.com` website. She also controls the `evil.com` name server.
- Mallory has constructed a malicious JavaScript function called `attack` that takes in an argument `url` and sends malware to that URL.
- The IP address of `www.evil.com` is 6.6.6.6 and the IP address of `eecs.berkeley.edu` is 7.7.7.7.
- Alice, the victim user, will load `www.evil.com` in her browser when the attack starts.

Assumptions:

- Unless otherwise specified, assume that Alice has disabled all DNS record caching (Alice's DNS cache is always empty).
- Alice does not use the stub/recursive resolver model. Alice makes all DNS requests on her own computer.
- Unless otherwise specified, assume that all requests are made over HTTP (not HTTPS).

Goal: Mallory wants Alice to successfully send malware to the EECS web server.

First attempt: Mallory tries to include `<script>attack("eecs.berkeley.edu")</script>` on `www.evil.com`. However, when Alice loads `www.evil.com`, nothing happens to the EECS server.

Mallory realizes that her attack isn't working due to the same-origin policy. Mallory's JavaScript is running with the origin of `www.evil.com`, but is trying to make requests to `eecs.berkeley.edu`.

New goal: In this question, we'll design an attack that tricks Alice's browser into making a request that will reach `eecs.berkeley.edu`, without violating the same-origin policy in Alice's browser.

Q10.1 (4 points) Alice loads `www.evil.com` in her browser.

Alice will first make a request to the root name server. What records will the root name server's response contain? Select all that apply.

- ☐ An NS type record with domain of the `.edu` name server
- ■ An NS type record with domain of the `.com` name server
- ■ An A type record with the IP address of the `.com` name server
- ☐ An NS type record with the IP address of `evil.com`
- ☐ None of the above

Solution: The root server will refer Alice to the `.com` NS, which requires the domain of the NS and the IP of the NS.

Q10.2 (3 points) Eventually, Alice will make a request to the `evil.com` name server. Which record should Mallory send in the DNS response?

- ○ An A type record, mapping `eecs.berkeley.edu` to 6.6.6.6
- ○ An A type record, mapping `eecs.berkeley.edu` to 7.7.7.7
- ● An A type record, mapping `www.evil.com` to 6.6.6.6
- ○ An A type record, mapping `www.evil.com` to 7.7.7.7

Solution: We want to send the correct IP for the first response, so we can have Alice download the malicious script from `evil.com`.

Q10.3 (4 points) After the DNS query, Alice can now load `www.evil.com`. What JavaScript should Mallory include on the website?

Solution: `<script>attack("www.evil.com")</script>`

Our goal is to get Alice to reload evil.com under a new DNS IP, so the script needs to make a second evil.com request.

Q10.4 (3 points) When Alice's browser runs the JavaScript from the previous part, Alice will make another query (since she didn't cache any records).

Which record should Mallory send in the DNS response?

- ○ An A type record, mapping `eecs.berkeley.edu` to 6.6.6.6
- ○ An A type record, mapping `eecs.berkeley.edu` to 7.7.7.7
- ○ An A type record, mapping `www.evil.com` to 6.6.6.6
- [✓ An A type record, mapping `www.evil.com` to 7.7.7.7]

Solution: As mentioned in the previous subpart solutions, we now want to make `evil.com` point to the IP of `eecs.berkeley.edu`, so the website loads what it thinks is `evil.com` (but actually goes to `eecs.berkeley.edu`).

At this point, Alice's browser should send malware to the EECS web server.

Next, we'll discuss possible defenses against this attack. For the rest of the question, each subpart is independent.

Q10.5 (3 points) **For this subpart only**, assume that Alice now caches DNS records.

Does the attack still work?

- [✓ Yes, if Mallory sends records with a short TTL (time-to-live).]
- ○ Yes, if Mallory sends records with a long TTL (time-to-live).
- ○ No, because the first DNS response from Mallory will be cached forever, so Alice will never make a second DNS query.
- ○ No, because DNS is vulnerable against on-path attackers.

Solution: The attack relies on the fact that the first DNS response (that is the correct IP to evil.com) isn't cached, or is cached but evicted shortly after. Therefore a long TTL doesn't work, but short TTL would still make the attack feasible.

Q10.6 (3 points) **For this subpart only**, assume that all name servers and Alice's computer use DNSSEC.

Does the attack still work?

○ Yes, because even though Mallory can't sign records, she can still replay old signed records she has previously received.

[✓ Yes, because Mallory can still sign records.]

○ No, because DNSSEC provides end-to-end integrity.

○ No, because DNSSEC provides confidentiality on records.

Solution: Yes, because Mallory still controls the `evil.com` name server and can sign the malicious records being sent to Alice.

Q10.7 (3 points) **For this subpart only**, assume that all websites (such as `eecs.berkeley.edu` and `www.evil.com`) are loaded with TLS (HTTPS), not HTTP.

Does the attack still work?

○ Yes, because TLS only protects against on-path and man-in-the-middle attackers.

○ Yes, because TLS does not stop Alice from making requests to malicious websites.

○ No, because the certificate from `www.evil.com` will not list its domain as `eecs.berkeley.edu`.

[✓ No, because the certificate from `eecs.berkeley.edu` will not list its domain as `www.evil.com`.]

Solution: No, because the certificate sent by the EECS name server contradicts the fact that Alice made a request to `www.evil.com`.

Nothing on this page will affect your grade in any way.

Comment Box

Congratulations for making it to the end of the exam! Feel free to leave any final thoughts, comments, feedback, or doodles here:

Nothing on this page will affect your grade in any way.

(Optional) Post-exam Activity: Battleship - The Rematch

EvanBot had so much fun playing battleship with CS161 students on the midterm and has decided they want to play again.

This time around, it is your turn to decide where to place the ships. You can place five ships on the board, either horizontally or vertically (no diagonal ships). The sizes of the five ships are given below.

1 Destroyer

X	X

2 Submarines

X	X	X

X	X	X

1 Battleship

X	X	X	X

1 Carrier

X	X	X	X	X

Draw the above ships on the grid below, placing an `X` in each square that has a part of a ship.

EvanBot has already decided where they think you are going to place your ships and has decided to preemptively commit to their shots. However, they won't reveal their shots until after the exam. Similar to the midterm, they have published a SHA−3 hash of their shots so you can verify they haven't changed them later:

H(shots) = `0x4983e0cb16d8394934847f32f1dc016278fecceb1079149e412c9ae3f3e9c464`

	1	2	3	4	5	6	7	8	9	10
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										