Popa & Weaver
Fall 2016

CS 161
Computer Security

Midterm 1

PRINT your name: _____2.2in, _____2.2in
(last) (first)

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.

SIGN your name: _____4in

PRINT your class account login: `cs161-`_____.5in and SID: _____2in

Name of the person sitting to your left: _____1.6in

Name of the person sitting to your right: _____1.6in

You may consult two sheets of paper (double-sided) of notes. You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted. If you think a question is ambiguous, choose the most reasonable assumption and document your assumption clearly.

You have 90 minutes. There are 8 questions, of varying credit (110 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

Do not turn this page until your instructor tells you to do so.

Problem 1 ***True/False*** **(18 points)**

Circle True or False. Do not justify your answer.

(a) TRUE or FALSE: Having a non-executable stack and heap is sufficient to protect against buffer overflow code execution attacks.

Solution: False.

(b) TRUE or FALSE: Setting the NX bit (i.e. disabling executable permission) on pages spanning the program's stack would prevent buffer overflow attacks

Solution: False. ROP attacks

(c) TRUE or FALSE: Consider a program that is compiled with stack canary protection. The canary has the same value across multiple executions of the program.

Solution: False.

(d) TRUE or FALSE: Alice encrypts messages for Bob using a block cipher in CBC mode. Instead of using fully random IVs every time she encrypts a message, Alice uses R, $R+1$, $R+2$, and so on, where R is a random value she chose once. This scheme is IND-CPA secure.

Solution: False. For example, an eavesdropper can tell if $m_2 = m_1 - 1$.

(e) TRUE or FALSE: The same question as above but for CTR mode.

Solution: True. The IVs will be different.

(f) TRUE or FALSE: Alice encrypts a message, M using a block cipher in CBC mode with a random IV and sends the ciphertext C to Bob along an insecure channel. It is computationally infeasible for an adversary to modify C such that Bob receives and can validly decrypt a different message M'.

Solution: False. Encryption guarantees confidentiality, but not necessarily integrity. An attacker could change the IV to some IV', thus changing the value of the first message block received by Bob.

(g) TRUE or FALSE: If Alice uses the same one-time-pad twice, to encrypt M_1 and M_2, an eavesdropper could discover whether or not $M_1 = M_2$.

Solution: True. Given $M_1 \oplus K$ and $M_2 \oplus K$, an eavesdropper can $\oplus$ the two together to get $M_1 \oplus M_2$. This is 0 if and only if $M_1 = M_2$.

(h) TRUE or FALSE: CBC-mode decryption is parallelizable.

Solution: True. CBC-mode encryption is not parallelizable, while CBC-mode decryption is parallelizable.

(i) TRUE or FALSE: Consider a man-in-the-middle attacker for Diffie-Hellman that cannot modify the network messages or insert new messages - the only thing it can do is eavesdrop messages and sees all the information Alice and Bob send to each other. Diffie-Hellman is insecure for such a man-in-the-middle attacker.

Solution: False

Problem 2 ***Shorts*** **(8 points)**

Provide one short answer (with no explanation). Do not provide more than one answer because you will not receive credit even if the answers include the correct answer.

(a) You try to overflow a vulnerable stack buffer in a target program, and the module is alerted to an overwritten value. What defense mechanism does the module have in place?

Solution : stack canaries

(b) You try to overflow a vulnerable buffer in a target program, overwriting the saved instruction pointer to point to your malicious code on the stack. You know exactly where your malicious code resides, but it does not run. What defense mechanism does the module have in place?

Solution : DEP, W^X

(c) You try to overflow a vulnerable buffer in a target program, but the memory layout has been randomized and you do not know where any code is. What defense mechanism does the module have in place?

Solution : ASLR

(d) Name one method that prevents against Return-Oriented Programming attacks.

Solution : memory safe programming languages

Problem 3 ***Good and bad hashes*** **(18 points)**

The following are some hash function candidates h'. For each, circle whether it is collision resistant or a one-way function (could be either, none, or just one). If you do not circle one property (indicating that h' does not satisfy it), give a concrete example of when h' fails, namely, either show how to invert the function or exhibit two values that collide. (And if you do not circle both properties, you should supply a counterexample for each). Assume that h is a secure cryptographic hash function.

(a) ONE WAY or COLLISION-RESISTANT: h'(x) = x

Not one-way (trivial right inverse), but collision-resistant because there are no collisions at all.

(b) ONE WAY or COLLISION-RESISTANT: h'(x) = h(h(x))

This is fine.

(c) ONE WAY or COLLISION-RESISTANT: h'(x) = h(x) mod 10, where 10 is just the constant number 10

Not one-way, because it is easy to find and x' such that h'(x') = h'(x). And hashing any eleven values will lead to a collision, by the pigeonhole principle.

(d) ONE WAY or COLLISION-RESISTANT: h'(x) = h(first $n-1$ bits of x), where n is the number of bits of x

x and x' of the same length but differing in the last bit will collide.

(e) ONE WAY or COLLISION-RESISTANT: h'(x) = $g^x \mod p$ for p a large prime and g a random generator mod p

One-way from discrete log, but x and $x + p - 1$ collide.

(f) ONE WAY or COLLISION-RESISTANT: h'(x) = h(x) | "hello", where '|' denotes concatenation

This is fine.

(g) ONE WAY or COLLISION-RESISTANT: h'(x) = x^2

Not one-way: can take square roots (note: only a *right inverse* is required, because in general hash functions have infinite domain but finite domain, so there are infinitely many x that hash to the same value).

Not collision-resistant: $h'(x) = h'(-x)$.

(h) ONE WAY or COLLISION-RESISTANT: h'(x) = h(x) | x

Collision-resistant due to the hash in the output but trivially invertible.

Problem 4 ***Vulnerable code*** **(12 points)**

```
1  void greet(char *arg) {
2       char buffer[16];
3       printf("I␣am␣Alice,␣what␣is␣your␣name?\n");
4       scanf("%s", buffer);
5       printf("Whats␣up,␣%s\n", buffer);
6  }
7
8  int main(int argc, char *argv[])
9  {
10  char beg[6] = 'Huh...';
11  char end[9] = 'maybe␣not?';
12  strncat(beg, end, 5);
13  greet(argv[1]);
14  return 0;
15 }
```

(a) What is the line number that has a memory vulnerability and how is this vulnerability called?

Solution : 4. Buffer overflow vulnerability

(b) Just before the program executes line 4, the registers are:

esp: 0xbfffffb20

ebp: 0xbfffffb48

Given this information, describe how an attacker would take advantage of the vulnerability. Also make sure to include the address that the attacker needs to overwrite.

Solution: The ebp frame pointer is pointing at 0xbfffb48, the return address you want to write over is 4 bytes above that at 0xbfffb4c. Write 44 bytes of garbage to get to the return address, then fill it with the address pointing to your malicious code.

(c) What should you change to fix the problem in part (a)?

Solution : length check before filling buffer

(d) Given the code as is, would stack canaries prevent exploitation of this vulnerability? Why or why not?

Solution : Yes. Overflow would be detected before following the poisoned return address.

Problem 5 ***Securing chat*** **(16 points)**

Consider ACME Corporation's secure online messaging protocol, which is as follows. Each user u has a private key SK_u and a public key PK_u. Assume that ACME correctly distributes users' public keys, and attackers did not interfere with this process.

Consider that Alice wants to communicate with Bob. Alice has PK_{Bob}. The protocol is as follows:

1. Alice randomly generates a symmetric key K.
2. Alice encrypts $wrapped\_K = encrypt(PK_{Bob}, K)$.
3. Alice signs $sig = sign(SK_{Alice}, wrapped\_K)$.
4. Alice sends $(wrapped\_K, sig)$ to Bob.
5. Then, when Alice wants to send a message M to Bob, she computes $T = MAC(K, M)$ and sends (M, T).

These are sent through the Internet, and attackers may observe and modify the data.

(a) Bob has PK_{Alice}. When he receives sig, $wrapped\_K$, M, and T, indicate the steps Bob must take to verify that Alice was the one who sent the message M and that it was not modified by an attacker.

Solution:

- Verify wrapped key: $verify(PK_{Alice}, wrapped\_K, sig)$,
- Decrypt the key $decrypt(SK_{Bob}, wrapped\_K) \rightarrow K$,
- Compute $T' = MAC(K, M)$ and check that $T' = T$.

If all checks pass, Bob can be assured that Alice sent this message.

(b) Can Alice initiate a conversation with Bob and send him a message while he is offline? Namely, can he verify the message without interacting with Alice?

Yes

(c) Bob wants to report that Alice sent messages which violates ACME's rules. He decides to disclose the transcript (containing sig, $wrapped\_K$, M, and T) and K to Charlie, who works at ACME. Charlie also has PK_{Alice}. Does this information prove that Alice intentionally sent M? If so, how can Charlie verify that? If not, explain why.

No because Bob can create a new MAC for a separate message. He has the secret key K for that.

(d) This protocol does not protect the confidentiality of the message from an attacker eavesdropping on the network. Indicate which of the steps above 1–5 need to be amended to provide confidentiality, and provide replacements here such that the overall protocol provides confidentiality.

The message M is sent in plaintext. Moreover, the MAC leaks about the data. It provides integrity and not confidentiality.

Alice generates a new key K' and wraps $wrapped\_K' = encrypt(PK_{Bob}, K')$ and signs it $sig' = sign(SK_{Alice}, wrapped_{K'})$. She sends $wrapped\_K'$ and sig' to Bob along with the previous wrapped key. Instead of (M, T), it sends (M', T'), where $M' = encrypt(K', M)$ and $T' = MAC(K, M')$.

Using the same key K instead of K' (and not generating K' above was also acceptable). It is safer to use a different key as with K', but we did not cover this distinction in class. Also, using $encrypt(PK_{Bob}, (M, T))$ was also acceptable.

Problem 6 ***Yet Another Memory Safety Attack*** **(8 points)**

Consider the C program in Figure 1, which is compiled for a 32-bit x86 machine. For your reference, we illustrate the stack frame layout for this procedure. Assume that the compiler has allocated no additional space for alignment or padding, beyond what is needed to allocate the database in stack.

(a) The procedure contains a vulnerability that can be used to exploit control flow and execute shell code, which you can assume is already placed in memory at address 0xdeadbeef. Give an invocation of updateRecord that exploits the program.

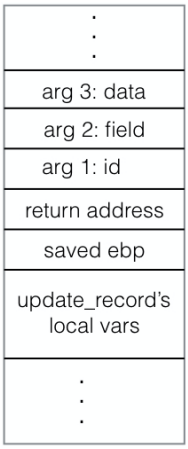
Solution: updateRecord(64, AGE, 0xdeadbeef)

(b) Your colleague tells you to enable stack canaries to avoid an attack of this form. Is the advice sound? Explain why or why not?

Solution: A stack canary would detect when a return address is overwritten because the value of the stack canary would be changed when writing above ebp.

```
1  typedef struct {
2    unsigned int salary;
3    unsigned int age;
4  } record_t;
5
6  #define MAX_DB_SIZE 64
7  enum field_t { SALARY = 0, AGE = 1 };
8
9  bool updateRecord(unsigned int id, field_t field, int data)
10 {
11   record_t database[MAX_DB_SIZE];
12   if (id <= MAX_DB_SIZE) {
13     if (field == SALARY) {
14       database[id].salary = data;
15     }
16     if (field == AGE) {
17       database[id].age = data;
18     }
19     return true;
20   }
21   else { /* invalid argument */
22     return false;
23   }
24 }
```

(a) Vulnerable Procedure



(b) Stack Frame Layout

Figure 1: Vulnerable Procedure

Problem 7 ***El Gamal*** **(18 points)**

Alice is trying to send a message to Bob using El Gamal encryption. They are working modulo some 2048-bit prime p, using some generator g. Bob has private key b, and public key $B = g^b$ as normal. However, Alice wants to send a larger message, m to Bob, that is more than 2048 bits long. She is able to split m into two pieces: m_1 and m_2, where m_1 and m_2 are each integers between 1 and $p-1$. If Bob receives m_1 and m_2, he can easily reconstruct m, so Alice need only focus on sending m_1 and m_2 encrypted to Bob.

Alice and Bob are considering different encryption schemes below. For each encryption algorithm, first determine the corresponding **decryption algorithm** that would allow Bob to recover m given the ciphertext that he receives. Then determine whether or not the scheme is **IND-CPA secure**. If you mark a scheme as secure, explain why. If not, show a potential attack/information leak in the scheme.

(a) Alice randomly selects r from $0, 1, ..., p-2$. Alice then sends ciphertext $(g^r, m_1 \times B^r, m_2 \times B^{r+1})$. Bob receives the ciphertext as a tuple of (R, S_1, S_2).

Decryption algorithm:

Bob can recover m_1 using standard ElGamal decryption ($m_1 = R^{-b} \times S_1$). m_2 can be recovereded by multiplying $B^{-1} \times S_2$ to get $m_2 \times B^r$. From there, Bob can use standard ElGamal decryption, so $m_2 = B^{-1} \times R^{-b} \times S_2$.

Secure?:

This is insecure. Given ciphertext (R, S_1, S_2), Eve can discover if $m_1 = m_2$ by checking if $S_1 \times B = S_2$. Thus information about m is leaked when using this encryption scheme.

(b) Alice uses a block cipher with 128-bit size blocks. She'll be using this block cipher in CBC mode. Her encryption function, $E_k(m; IV)$ takes a message of arbitrary length, splits it up into blocks (padding the last block with random bytes), and encrypts the message using her block cipher in CBC mode, using the supplied IV. Her decryption function, $D_k(c; IV)$ takes ciphertext of arbitrary length and decrypts it, again using her block cipher in CBC mode with the supplied IV.

Alice first randomly selects r from $0, ..., p-2$. Alice then independently randomly selects k, IV from $0, ..., 2^{128} - 1$. Alice does not split up m into m_1 and m_2, and instead sends ciphertext $(g^r, k \times B^r, IV, E_k(m; IV))$. Bob receives the ciphertext as a tuple of (R, K, IV, C).

Decryption algorithm:

$m = D_{R^{-b} \times K}(C; IV)$. Bob just first decrypts the key k using El-Gamal, and then uses the provided AES-CBC decryption algorithm to decrypt the ciphertext.

Secure?:

Solution: This is secure. This encryption scheme can be decomposed into two parts - Alice first encrypts a randomly generated key k using ElGamal encryption

and sends this to Bob. Eve gets no information about k since ElGamal is IND-CPA secure. Then Alice encrypts her message using a block cipher keyed with the previously generated k in CBC mode, which is IND-CPA secure as long as the IV is randomly generated.

Problem 8 ***Reasoning about code*** **(12 points)**

Eve is working on some code to help break a rudimentary cryptosystem. She needs to combine two strings according to a given formula, for example:

```
char out[80];
mix(out, "abcdef", "123456", "LRLLRRLLLRRR");
printf("%s\n", out); // a1bc23def456
```

Here is her code so far:

```
1  void mix(char *mixture, char *left, char *right, char *formula) {
2    size_t n = strlen(formula), li = 0, ri = 0;
3    for (size_t i = 0; i < n; i++) {
4      switch (formula[i]) {
5      case 'L':
6        mixture[i] = left[li++];
7        break;
8      case 'R':
9        mixture[i] = right[ri++];
10       break;
11     }
12   }
13   mixture[n] = '\0';
14 }
```

It works okay on some of her own tests, but when Eve let her friend Mallory try it out, she heard that it crashes and gives strange results on some inputs.

"You're just using it wrong," Eve complains to Mallory, "I'm going to have to spell out the preconditions, postconditions, and invariants, aren't I?"

(a) What preconditions for `mix` ensure that it won't crash and won't output any strange characters?

Solution:

You have some wiggle room for the strictness of conditions. However, we require that Eve's example invocation works (notably, ***`strlen(left)`*** $<$ ***`strlen(formula)`****). We accepted many different wordings as well. Here is an example that would receive full credit.*

All arguments are not `NULL`, `formula` is null-terminated, `mixture` is at least as long as `formula` (including null terminators), `formula` contains only 'L' and 'R', `left` is at least as long as the number of 'L's in `formula`, `right` is at least as long as the number of 'R's in `formula`

(b) What are the loop invariants that hold on line 4 (including for `left` and `right`), provided that the preconditions hold?

Solution:

Notice that in the last few iterations in Eve's example, $li =$ ***`strlen(left)`***

$0 \leq i < n$, `formula[i]` $\in$ {'L', 'R'}, $0 \leq li$, $0 \leq ri$, $li <$ `strlen(left)` if `formula[i]` = 'L', $ri <$ `strlen(right)` if `formula[i]` = 'R'

(c) What postconditions hold, provided that the preconditions hold?

Solution:

`mixture` contains the combined result, `mixture` is null-terminated, `strlen(mixture)` = `strlen(formula)`