Popa & Wagner
Spring 2020

CS 161
Computer Security

Discussion 4

Exam Review

Question 1 ***Cryptography Short Answer***

Give **concise** answers to the following questions. Do not provide multiple answers to a question; you will not receive credit even if your answers contain a correct answer.

(a) Alice and Bob are thinking of using asymmetric encryption with a **single** public & private key that they both share. Explain one advantage of symmetric encryption over this asymmetric encryption model.

> **Solution:** Symmetric encryption is faster. Symmetric encryption has smaller key sizes. Symmetric encryption can be chained more easily.

(b) What is the computationally difficult problem that RSA digital signatures rely on?

> **Solution:** Finding prime factors of large integers.

(c) In all the confidentiality cryptography games we have discussed, the adversary gets to send a pair of messages (m_0, m_1) to the challenger. The challenger only reveals the encryption of one of these messages. Explain why m_0 and m_1 need to be the same length.

> **Solution:**
>
> Encryption methods (to some extent) preserve the length of the original message, and so sending two messages of wildly different lengths would let the adversary distinguish.

(d) Say we have two similar messages M and M'. We encrypt both messages in CBC mode, but accidentally reuse the same IV. Then we encrypt both messages in CTR mode, but accidentally reuse the same IV (but different from the one we used for CBC mode). CBC mode will compromise lesser or equal amounts of information compared to CTR mode.

[✓ True] False

Solution: CBC encryption will reveal information up to the first difference in the messages, whereas CTR will reveal information anywhere that both messages are identical (including up to the first difference).

(e) Alice, Bob, and Charlie decide to make a shared key using a slightly altered DH key exchange: They agree on primes p and q and each choose their secret values a, b, and c. They then send off $p^a \mod q$, $p^b \mod q$, $p^c \mod q$ respectively. With no further communication, can they now agree on secret value which a passive eavesdropper Eve cannot determine?

If so, give such a value and prove why Eve cannot recreate the value. If not, explain why.

Solution: No. There's no way to produce a private combination since that would, for example, require Alice to recreate p^{abc} with no knowledge of b or c and only p^b and p^c and a which cannot be done.

Question 2 ***Food!***

Suppose there is a database where each entry is a name of a person and the person's favorite food, all encrypted. Mallory is an "honest but curious" employee at the company who knows the names of every person in the database but wants to know about their favorite food. However, she does not have the private keys to any encryption scheme the database uses.

Note that in this particular database, the names do not repeat, but the foods may repeat. Also assume that when encrypting, padding is used so the length is the same.

(a) Suppose there is a request made to the database to fetch all the names. Each name is encrypted and sent out, and Mallory can see all of these encrypted names. More concretely, she sees $E_k(\text{name}_1), E_k(\text{name}_2), \ldots$ If the encryption scheme was deterministic could Mallory learn anything new? Why or why not?

> **Solution:**
>
> Mallory will learn nothing new since the names are unique (and their encryptions are of equal length)

(b) Could Mallory learn anything new if the encryption scheme was IND-CPA? Why or why not?

> **Solution:**
>
> Same answer as above

(c) Suppose a new request is made to obtain all the foods. As previous, Mallory can see all of these encrypted values: $E_k(\text{food}_1), E_k(\text{food}_2), \ldots$ If the encryption scheme was deterministic, could Mallory learn anything new? Why or why not?

> **Solution:**
>
> Yes. This time, Mallory can learn the relative distribution of favorite foods.

(d) Could Mallory learn anything new if the encryption scheme was IND-CPA? Why or why not?

> **Solution:**
>
> No, since encryption that is IND-CPA ensures that the same message will output different ciphertexts.

Question 3 *Student Linked List*

Lord Dirks writes the following code below to manage the students of Leland Junior University:

```
 1 struct student_node {
 2     char name[8];
 3     struct student_node *next;
 4 };
 5
 6 typedef struct student_node student_node;
 7
 8 void add_student(student_node *head, char *student_name) {
 9     student_node *new_student = calloc(1, sizeof(student_node))
          ;
10     while (head->next) head = head->next;
11     head->next = new_student;
12     strcpy(head->name, student_name);
13 }
14
15 student_node first;
16
17 int main() {
18     char *name_to_add;
19     first.next = NULL;
20     while (has_input()) {
21         name_to_add = safely_read_input();
22         /* esp = 0xbfff'f09c */
23         add_student(&first, name_to_add);
24     }
25 }
```

(a) Identify the line which causes the vulnerability. What vulnerability is this?

Solution:

The line with `strcpy`. Buffer overflow.

(b) Raluca needs your help to PwN Lord Dirks. To help you, she added some shellcode at the memory address `0xdeadbeef`. What names would you need to enter into the program in order to cause the execution of the shellcode? Note that the value of `esp` at line 21 is `0xbffff09c`. Assume that the compiler does not reorder any local variables or pad stack frames.

Solution:

The intuition for this problem is by overwriting the pointer `head->next`, we can ensure that the **next** name writes onto the stack. Note that solutions which attempted to do a buffer overflow in a single line do not work, because `first` is in static memory, not on the stack. It is also not possible to write all the way from static memory to stack memory.

In order to write `head->next` points to the stack, we need to enter a name which is greater than eight characters.

Since we aim to overwrite the return address at `0xbffff090`, we can try this:

`AAAAAAAA\x90\xf0\xff\xbf`

This successfully makes it so that the pointer `head->next` points onto the stack, and now our program thinks that this is a `student_node` struct. When we insert our next string, the program will "stop" at this memory address. Since the first member of a struct has offset 0, we see that our next `name` will be inputted at `0xbffff090`. Therefore, we can overwrite the return address of `add_student` with our next name:

`\xef\xbe\xad\xde`

However this has a subtle problem: there is no guarantee our program will not keep iterating once it hits `0xbffff090`, because it is likely that `((student_node *) 0xbffff090)->next` is not actually NULL. This would cause our program to keep iterating `head = head->next` and likely crash. (There will be significant credit for solutions which successfully set up a pointer to the stack and attempt to overwrite the return address of any function, even if they ignore this problem.)

The full solution requires us to write significantly below the stack pointer, where all of the memory is untouched zero bytes. Then, we can write our way up to the return address. The following sample solution works:

`AAAAAAAA\x90\xe0\xff\xbf`

`A` (`0x1000` times) `\xef\xbe\xad\xde`

Note that with this solution, we **must** overwrite the return address of `add_student`, since the program will crash before exiting from `main`.