Midterm Review - Memory Safety

Question 1 ***True/false*** ()

Q1.1 True or False: Buffer overflows can occur on the stack and heap, but not in the static section of C memory.

○ True ○ False

Q1.2 True or False: The primary danger of format string vulnerabilities is that they let an attacker write more bytes into a buffer than the buffer has space for.

○ True ○ False

Q1.3 True or False: If ASLR is enabled, leaking the address of a stack variable would give an attacker the address of heap variables.

○ True ○ False

Q1.4 True or False: Enabling stack canaries, ASLR, and DEP prevents all buffer overflow attacks.

○ True ○ False

Q1.5 True or False: Coding in a memory-safe language prevents all buffer overflow attacks.

○ True ○ False

Question 2 ***A Dangerous Game*** **(35 min)**

This question has 9 subparts.

Note: This is the hardest question on the exam. We recommend trying the other questions on the exam before this one.

A new online game, *HackMe*, splits 128-512 players into groups of 16 and has all groups compete to hack each other. *HackMe* uses a hash table to create groups and store info about each player.

Recall that a hash table is an array of "buckets" (here each bucket is a linked list). To add a player to the table, a hash function is evaluated to decide which bucket the player goes into, and they are appended to the linked list of that bucket.

```
1  typedef struct Player {
2      int id;
3      int hacking_ability;
4  } Player;
5
6  typedef struct Bucket {
7      int8_t size;    // 8 bit signed integer
8      LinkedList *b; // Pointer to a linked list implementation
9  } Bucket;
10
11 typedef struct HashTable {
12     int players;
13     Bucket buckets[16];
14 } HashTable;
15
16 void add_player(HashTable *t, Player p) {
17     size_t idx = hash(p.id + t->players); // hash range is [0,
           16)
18     append(t->buckets[idx].b, p);          // appends p to
          LinkedList
19     t->buckets[idx].size += 1;
20     t->players += 1;
21 }
```

Q2.1 (3 points) Assume that `hash()` outputs an unsigned integer equal to the last 4 bits of a pseudorandom, cryptographic hash function. If the table contains a number of `Player`s with random `id`s, what do you expect about the size of the buckets?

◯ (A) They will all roughly be the same size

◯ (B) The 0^{th} bucket will be larger than the 1^{st} bucket

◯ (C) The 1^{st} bucket will be larger than the 0^{th} bucket

○ (D) ——

○ (E) ——

○ (F) ——

Q2.2 (3 points) Assume that `hash()` outputs an unsigned integer equal to the last 4 bits of a pseudorandom, cryptographic hash function. If the table contains a number of `Players` with the same `id`, what do you expect about the size of the buckets?

◯ (G) They will all roughly be the same size

◯ (H) The 0^{th} bucket will be larger than the 1^{st} bucket

◯ (I) The 1^{st} bucket will be larger than the 0^{th} bucket

◯ (J) —

◯ (K) —

◯ (L) —

Q2.3 (3 points) Say a user stores a large number (ie. 10000) of `Players` in a `HashTable`.

Which of the following would occur given the code above?

◯ (A) Integer overflow ◯ (C) Off-by-one ◯ (E) —

◯ (B) Buffer overflow ◯ (D) — ◯ (F) —

Q2.4 (3 points) Which line number contains the vulnerability from the previous part?

◯ (G) Line 7 ◯ (I) Line 13 ◯ (K) —

◯ (H) Line 8 ◯ (J) — ◯ (L) —

To register a group for playing *HackMe*, one inputs a list of `Players` to the following function which adds all `Players` to a HashTable, assigns the group to a server based on size of the 0^{th} bucket, and sets a group name.

```
1  void register_group(Player *players, size_t num_players) {
2      char *server_names[128] = { /* Contains 128 server names
           */ };
3      char *a_gift = 0xffffd528; // Pointer to the stack canary
4      char group_name[16];
5      HashTable group;
6      for (int i = 0; i < num_players; i++) {
7        add_player(&group, players[i]);
8      }
9      printf("Use server: %s\n", server_names[group.buckets[0].
           size]);
10     printf("Please provide 16 character group name: \n");
```

```
11      gets(group_name);
12      ...
13  }
```

Q2.5 (5 points) Consider line 9:

```
printf("Use server: %s\n", server_names[group.buckets[0].size]);
```

Which *valid* values of `group.buckets[0].size` would cause this statement to print something outside of `server_names`?

_____ ≤ `group.buckets[0].size` ≤ _____

Please clearly label your final answer on your answer sheet.

Q2.6 (10 points) Mallory challenges you to hack *HackMe*. Assume you can invoke `register_group` with a list of `Player`'s of your choosing, but the list must have length between [128, 512] and `num_players` must always be correct.

HackMe uses a 32-bit x86 system with **stack canaries enabled** (assume that canaries don't contain null bytes) but no W^X bit or ASLR. In order to help you out, Mallory has added a pointer to the stack canary: `a_gift`.

Describe the list of `Player`s you input. Assume that `hash()` is a publicly-known function that you can query before making your list.

Clarification made during the exam: `a_gift` is a pointer to the stack canary of the `register_group` frame.

Clarification made during the exam: Your answer to subpart 6 should give you information to complete the exploit in subpart 7.

◯ (G) —— ◯ (H) —— ◯ (I) —— ◯ (J) —— ◯ (K) —— ◯ (L) ——

If you need more space on your answer sheet, you can write on a blank sheet of paper and attach it with your submission.

Q2.7 (5 points) Write down your exact input to the `gets` call at line 11. Assume that `SHELLCODE` holds 64-byte shellcode, `GARBAGE` is an arbitrary byte, and `OUTPUT` is the output from the print statement at line 9.

You can write constants using hex (e.g., 0xFF or 0xA02200FC). For instance, `4*GARBAGE + OUTPUT[:1] + SHELLCODE` would represent four irrelevant bytes, followed by the first

byte of the print result, followed by the 64-byte shellcode.

◯ (A) —— ◯ (B) —— ◯ (C) —— ◯ (D) —— ◯ (E) —— ◯ (F) ——

Q2.8 (3 points) Which of the following could prevent this attack? Assume `a_gift` always correct points to the stack canary.

☐ (G) ASLR

☐ (H) $W \wedge X$ protection (NX bit)

☐ (I) Increasing the size of `server_names` to 256

☐ (J) None of the above

☐ (K) ——

☐ (L) ——