CSci-454 Computer & Network Security
Midterm Exam – Mar 10, 2021
Deadline: Mar 11 at 3:30 PM

I understand that violations such as submitting work that isn't my own, sharing questions from this exam, asking for assistance from anyone other then the instructor, copy-pasting answers from outside resources may result in permanent failure of the course and/or any other sanctions in accordance with the William & Mary Honor Code. I understand that the contents of this exam are required to keep secret and after completing of the course such materials are required to be deleted including from any cloud resources such as Google Drive.

Student name: Daniel Quiroga

Question 1 (12 Pt)

Alice wants to communicate securely with Bob across a **non-encrypted** network. For this purpose they developed a simple authentication protocol. First they **securely** (using another secure channel) shared a secret randomly generated 8-character long password **S**. Then, every message **M** that is sent across network will be combined with authenticator **h** resulting in a datagram **h|M**. Authenticator **h** is calculated using a **secure** hash function **H** in the following way: **h = (S)**. Assume a malicious Eve is capable of 1) reading any datagram 2) sending any datagram 3) changing the contents of any datagram sent across the network. However she cannot prevent datagrams from being delivered.

For questions **1-5** assume Eve cannot find out / break **S**:

1. Explain how Eve can violate confidentiality of Alice and Bob's communication.

Since eve is able to read any of these messages being sent via her capability of reading any datagram, she can leak any of the contents of their conversations and hence the data would be disclosed to entities that are not Alice and Bob.

2. Explain how Eve can violate integrity of Alice and Bob's communication.

Via Eve's third capability, she would be able to change what the message is being sent from Alice to Bob and hence creating this integrity of the conversation since Alice may have not wanted to disclose something or a lie was sent to Bob via Eve's capability. This would be an example of modification of protected information in this case being the messages sent.

3. Explain how Eve can violate availability of Alice and Bob's communication.

Since Eve has capabilities of sending any datagram, she has the ability to use the service of the communication without actual permission, i.e. Eve is not a legitimate user and is still using the service without any request. This is 100% not what was intended since the only legitimate users in this system are Alice and Bob.

4. Explain how Eve can brake authentication of this scheme.

Eve broke the authentication of the system since the only people whom should have access to the services are Bob and Alice and with Eve capabilites of sending and receiving any datagram, she has the exact same services as a user without any authentication and also the authorization to preform such

services since this was created solely and legitimately for Alice and Bob. Another example would be the ability for Eve to change any datagram when this should only be possible by the user who created it but since Eve is able to do so this is a brake in the authorization and authentication.

5. Assume now **h** is calculated in the following way: **h = H(S|M)**. What security propert(y/ies) does this provide? Explain why.

This would allow more security property in Confidentiality and integrity since a hacker would need now the exactly message that say Alice wanted to send and would need more information than just an 8-character string. Since hash functions are non revesible the hacker would need an extra set of information to be able to modify any personal information of a user say the message being sent and to actually find out what is being sent and what is stored inside of the datagram.

6. Which attack(s) can be prevented if **h** is calculated in the following way **h = H(S|"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa")**? Why?

This would prevent rainbow table attacks since the attack would need to deal with the salt that is constant in all of the hashed values and hence a main protection against these type of attacks since the rainbow attack algorithm does not work when a salt is present. However since this is constant, the hacker could use a salted rainbow table if they are able to figure the salt out.

Question 2 (8 Pt)

There are two user accounts in a system (user1, user2) and a root account. There are three data files on disk, f1,f2,f3 with following permissions:

```
-rw---x--- 1 user1 user1 14 Oct  9 16:43 f1
-rw-rw---- 1 user2 user2 15 Oct  9 16:43 f2
-rw------x 1 root  root  14 Oct  9 16:44 f3
```

Program listed bellow is compiled as executable read1, read2 and read3

```
#include <stdio.h>
int main(int argc, char **argv){
        char buf[1001];
        if(argc!=2){
                fprintf(stderr, “Usage: ./cat_u <filename>\n”);
                return 1;}
        FILE *f;
        f = fopen(argv[1],”r”);
        if(f == NULL){
                fprintf(stderr, “Could not open file\n");
                return 1;}
        fscanf(f, "%1000s",buf);
        buf[1000] = 0;
        printf("%s\n",buf);}
```

/bin/cat and read1, read2 and read3 have following permissions:

```
-rwxr-xr-x 1 root  root  5208 Jan  2 11:11 /bin/cat
-rwxr-xr-x 1 user1 user1 8536 Oct  9 18:55 read1
-rwsr-xr-x 1 user2 user2 8536 Oct  9 18:55 read2
-rwsr-xr-x 1 root  root  8536 Oct  9 18:55 read3
```

All commands bellow are executed when logged in as user1. For each command indicate if it is **permitted/not-permitted and explain why**:

1	user1$ /bin/cat f1
2	user1$ /bin/cat f2
3	user1$ /bin/cat f3
4	user1$ ./read2 f1
5	user1$ ./read2 f2
6	user1$ ./read2 f3
7	user1$ ./read3 f3
8	user1$ ./read3 f2

1) yes, the user has execute access to the /bin/cat executable and read access to f1

2) no, the user does not have read access to f2 hence would not be able to run this command
3) no, the user does not have read access to f3 hence would not be able to run this command
4) no, since the set-uid bit is present when we execute this executable we have an euid of user2 and since the data is not readable by user2 we are not permitted to run the command
5) yes, since the set-uid bit is present we execute the executable as a user2 permission and hence have read access to f2 allowing us to follow through on this command
6) no, the user does not have read access to f2 hence would not be able to run this command
7) yes, since the set-uid bit is present we have root access when executing this executable since our euid = root, hence any data would be readable and this command would run without an issue
8) yes, since the set-uid bit is present we have root access when executing this executable since our euid = root, hence any data would be readable and this command would run without an issue

Question 3 (9 Pt)

a. Alice had an account at website X.com. As a result of data breach, the **website's database storing hashed passwords were stolen**. All passwords were hashed using the **same but relatively long salt**. However, the investigation discovered the value of the salt was not disclosed. Alice uses the same (short, but randomly generated, e.g. "4$Av") password on multiple websites. (6 pt)

i. Assume X.com's login process requires sending plaintext passwords, then computing hashes on server side. Would the attacker be able to login into X.com using the information they gathered from the described attack?

Yes	**No**

ii. Would the attacker be able to login into other websites where Alice used the same password using the information they gathered from the attack?

Yes	**No**

iii. (2 pt) Assume attacker also had an account at X.com, how can they attempt to recover the salt?

Since the salt is constant, the attacker can see how the hashed function reacts to different types of passwords that the attacker may pass in. For example, begin to create a list of passwords and hashed values returned and eventually they can perform a salted rainbow attack to figure out the salt and then the passwords of the database.

iv. Would having a unique salt for each entry in password database help against an attacker whos the **only** goal is to break Alice's password?

Yes	**No**

v. Would it help against an attacker who wants to break as many passwords as possible?

Yes	No

b. (3 pt) Tim's linux password (P) contains 4 randomly generated alphanumerical (26 lower-case, 26 upper-case letters and 9 digits) characters. Hashed password is stored in /etc/shadow, which stores: the 8 alphanumerical character salt (S) and hashed password. (4 pt)

i. Assuming hash function H() write down the expression showing how hashed password is computed.

Hash_password = H(P|S)

ii. (2 pt) Assume Malory owns a rainbow table that covers all 4 alphanumerical character inputs for the same function H() and manages to get read access to /etc/shadow. What would be the complexity (number of operations) of the attack to crack Tim's password?

Since we have 61 different options for characters and we also have 4 different characters. In order crack Tim's password Malory would at worst case have to go through every single combination of the 4 alphanumerical characters. So we see that Malory would need to do 61^4 = 13845841 different passwords.

Question 4 (6 Pt)

For the program below, please indicate where the following objects are located, assume a 32-bit x86 system:

```
int c = 0x777;
int d;
const char * hello_string = "Hello, World!";

int plus(int a, int b){
   return a + b;
}

int main(){
   int e = 0x777;
   char *str = malloc(1000);
   printf(hello_string);
   return 0;}
```

	Locations										
	stack	rodata	data	heap	text	library	cache	main	mmap	bss	register
variable a	X										
variable b	X										
plus()'s return value											X
plus()'s return address	X										
variable c			X								
variable d										X	
*str (data)				X							
str (pointer)	X										
hello_string (pointer)	X										
hello_string[0] (first character)		X									
main()	X										
printf()	X										

Question 5 (13 Pt)

Consider following function:

```
int hello(){
  char buf[100];
  printf(“Please type in your name:”)
  scanf("%s", buf);
  printf("Hello, %s\n", buf);
  return 0;}
int main(){
 hello();}
```

Assume there are no protections present.

1) Explain why function hello is vulnerable (2pt)

We see that the variable buf is vunerable to a buffer overflow attack since depending on the input given by the user, the memory could be impacted on the stack and hence a directed change of return address could uncover certain information to say an attack trying to steal information that is held in the system.

2) Assume attacker is typing in inputs. Explain how they can inject code and execute it inside the vulnerable program. Draw the layout of the stack after the moment of code injection, showing a) stack frame of function hello, b) the location of saved %ebp register, c) return address, d) injected code. (4pt)

So the attacker here would input more than the alloted space for the buf, hence when the program receives this, must accommodate it due no restraints and since the memory is in stack it will begin to work its way first through the memory of the ebp register and then to the return address. So the attack could redirect the return address to say a different location with their input and hence do some damage depending on their next move.

3) What would happen if attacker types in "%s%s%s%s"? (3pt)

This would just print out “Hello, %s%s%s%s” since the formatting of printf is properly created and the size of the input is less than that to cause any harm so the program would just perform the operation as normal.

4) What protection methods can prevent injected code from being executed? Give minimum **two** protections and briefly describe how they stop the attack. (2pt)

We can use the %ns, where n is a positive integer to prevent this attack. This will make it so that only a certain number of characters are inserted into the buffer and hence we would avoid any injuection of an input that bypasses the memory that was allocated to the buffer currently.

The function fgets would also be useful here since it will limit the string lengths and hence prevent the opportunity to overwrite data that is currently allocated to the buffer and other registers and/or stack information.

5) For code below, can attacker overflow buffer b into buffer a? Why or why not? (2pt)

```
void foo(){
  char* a = malloc(100);
  char b[100];
  scanf("%s", b);
  printf("Hello, %s\n", buf);}
```

No, this would not be possible since the stack and heap grow and read in opposite directions, when the buffer is being overflowed into b it would never be able to intersect where a would be in the heap in memory and hence the attack overflow would not actually be able to get into a.

Question 6 (7 Pt)

Consider the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
const char *secret = "Enigma"; //[1]
const char *not_secret = "This is public data, nothing to see here!"; //[2]
int main(int argc, char ** argv){
  if(argc!=2)
    return -1;
  int offset = strtol(argv[1], NULL, 16);
  int str_len = strlen(not_secret);
  if(offset > str_len){ //[3]
    printf("OUT OF BOUNDS %d\n", offset);
    return -1;
  }
  printf("%s\n", not_secret + offset);  }
```

The program receives a single argument and prints out the not_secret string [2] starting from the character number provided as the argument in hexadecimal format. The program checks the string bounds as shown in [3] to prevent reading the memory passed the string limit if user provides offset above the end of the line. The secret string [1] must never be printed out. Below provided examples of program use:

```
$ ./a.out 0x0
This is public data, nothing to see here!
$ ./a.out 0x5
is public data, nothing to see here!
$ ./a.out 0xf
data, nothing to see here!
$ ./a.out 0x0ffffff8
Enigma
```

Explain why the program outputs the secret sting, what vulnerability is responsible for that (4pt). Write down line of code that makes the code vulnerable along with the fixed version (3pt).

The vulnerability that is responsible for that is the integer overflow vulnerability. Since a value was passed in that is greater than the max signed int value, when doing the initialization, that value was wrapped around and was given a value less than that of the string length.
“Int offset = …. “ -> “unsigned int offset = …”

Question 7 (4 Pt)

```
int main(){
        int a = 2147483640;
        int b = 10;
        unsigned int c = 100;
        int d = 100;
        printf("%d\n", a+b > c);
        printf("%d\n", a+b > d);}
```

Explain why the code above prints out:
$ ./a.out
1
0

in the first printf we have the comparison between signed int and unsigned int so everything is changed to unsigned but the value of a +b does not actually change here.

In the second printf we have a comparison between two signed int and since the a + b yeilds a value larger than what can be represented by signed int, there would be a wrap around which hence would return a negative number and making the d a greater value afterwards.

Question 8 (10 Pt)

Consider following code and a 32-bit x86 system:

```
echo(){
        char user_str[100]
        scanf("%99s", user_str);
        printf(user_str);}
```

Suppose user_str is located at address 0x2210 (on stack), while its address is read by **printf** function located at address 0x2200 (on stack).

a. What is the name of vulnerability/attack here? (1 pt)

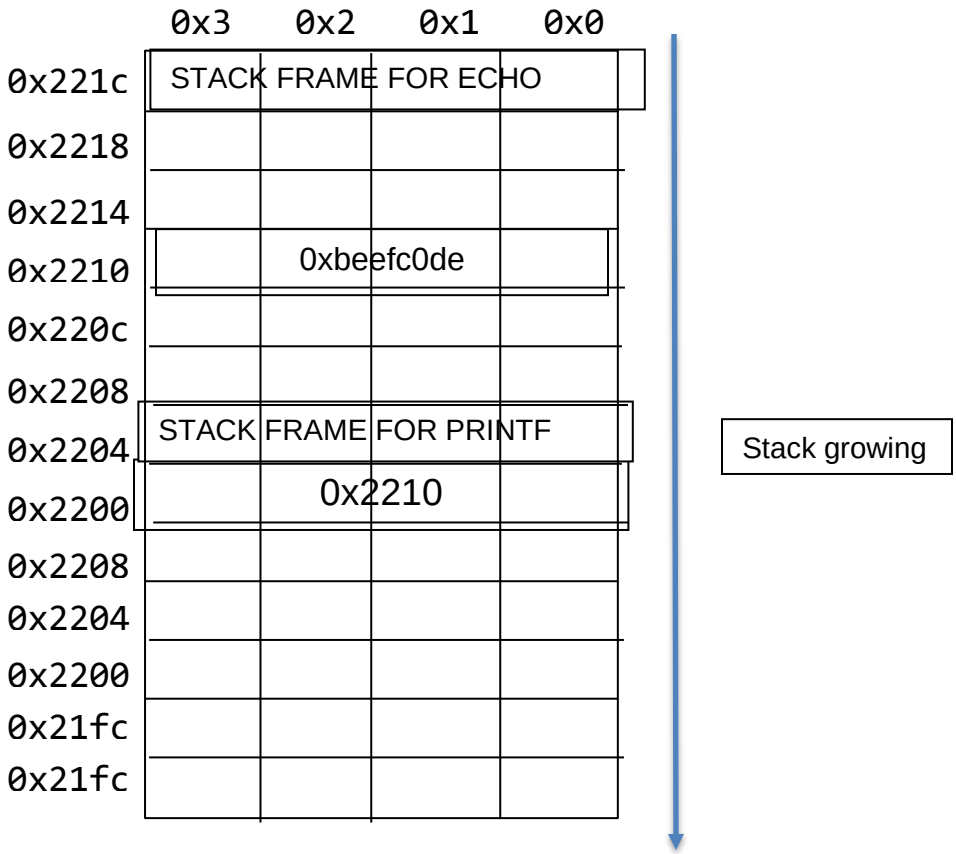
This is a format string attack as shown in quiz3.

b. What does the attacker need to provide as input to read memory (string) at address 0xbeefc0de? Assume 32-bit little-endian x86 platform. (4 pt)

"\xde\xc0\xef\xbe%x%x%x%x%s" This would allow us to read the contents of the stack at address 0xbeefc0de and hence a huge vulnerability

attack_inputs:

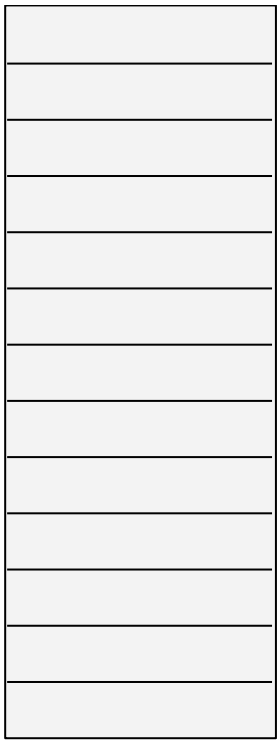
c. Using memory diagram below please show following at the moment execution enters the printf function with attack inputs from above: (1) direction in which stack is growing; (2) stack frames of echo and printf; (3) contents of 0x2210; (4) contents of 0x2200. You can ignore everything else, e.g. stack canary, return addresses, local arguments, etc. (5 pt)



Question 9 (8 Pt)

Code below is used in an ROP-attack. It performs 5 actions. Please write down **description** of these actions in order they are performed, e.g. **"register %aex is loaded with data 12345, etc."**. Make sure to use words instead of assembly instruction/pseudocode. Note, the assembly is written using the AT&T assembly syntax. E.g. add %eax, %ebx adds %eax and %ebx together and stores in **%ebx**. Parentheses (X) represent memory accesses at address X. Assume execution starts at 0xb880040 and stack pointer %esp points as shown.

Stack	Code	Actions

esp 	0xb880000:
	pop %ecx
	ret
	...
	0xb880010:
	pop %ebx
	ret
	...
	0xb880020:
	add %ebx, %eax
	ret
	...
	0xb880030:
	mov %eax, (%ecx)
	ret
	...
	0xb880040:
	pop %ecx
	ret
		1. here we have the top element on the stack (0x002) saved into register ecx and then the program pops the top of stack and goes to the location in memory that was popped off 0xb880010
		
		2. here we will remove the top element of the stack (0x005) and store it inside the register ebx and then the program pops the top of stack and goes to the location in memory that was popped 0xb880020
		
		3. here we add the value of ebx and eax and store the value of that operation in eax and then the program pops the top of stack and goes to the location in memory that was popped off 0xb880040
		
		4. here we have the top element on the stack (0xbeefcace) saved into register ecx and then the program pops the top of stack and goes to the location in memory that was popped off 0xb880030
		
		5. here we see that the value of eax over writes the current value at the memory location of ecx meaning wherever ecx was pointing to on the stack or what variable now has a value of eax. then the program pops the top of stack and goes to the location in memory that was popped off in this case no address so the program finishes execution

Question 10 (4 Pt)

1. (4 pt) Bob managed to hijack control in a vulnerable program exploiting a the buffer overflow vulnerability. To carry out a successful attack he needs to execute an ROP gadget: IMUL AL and RET. Note that the machine code for IMUL AL is f6 e8. Below listed the disassembly of vulnerable executable. What does Bob need to write into the return address to trigger the gadget execution?

<address>	<bytes>	<disassembly>
0000000000000000	4531c9	XOR R9D, R9D
0000000000000003	4531c0	XOR R8D, R8D
0000000000000006	c3	RET
0000000000000007	31f6	XOR ESI, ESI
0000000000000009	4889c6	MOV RSI, RAX
000000000000000c	4989f6	MOV R14, RSI
000000000000000f	4531f6	XOR R14D, R14D
0000000000000012	e8c3580000	CALL 0x58da
0000000000000017	90	NOP
0000000000000018	4531c9	XOR R9D, R9D
000000000000001b	4531c0	XOR R8D, R8D
000000000000001e	31f6	XOR ESI, ESI
0000000000000020	4889c6	MOV RSI, RAX
0000000000000023	4989c7	MOV R15, RAX

All we would need to do in the return address is to write in NOP to move the pointer once and then once the address of 0x00f is hit we get the call f6 e8 and hence properly execute our ROP gadget IMUL AL.