Computer Security Exam

Professors F. Maggi & S. Zanero

Milan, 15/01/2018

Last (family) Name ______________________________

First (given) Name ______________________________

Matricola or Codice Persona ______________________________

Have you done any challenges/homework, even partially? [ ] Yes [ ] No

Professor [ ] Maggi [ ] Zanero

Instructions

- The exam is composed of 11 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 note, book, 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.

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 (9 points)

Consider the C program below, which runs on the usual IA-32 architecture (32 bits), with the usual “`cdecl`” calling convention.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

int login() {
    struct {
      char secret[16];
      char buf[16];
      char *error;
      int unused_variable;
    } s;

    s.error = "Access Denied!";

    int fd = open("secret.txt", O_RDONLY);

    read(fd, s.secret, 16); /* load secret into memory */
    s.secret[15] = '\0';

    scanf("%s", s.buf);
    if(!strcmp(s.buf, s.secret)) {
       return 1; /* Access OK */
    } else {
        printf("%s\n", s.error);
        return 0; /* Access KO */
    }
}

int main(int argc, char** argv) {
    login();
}
```

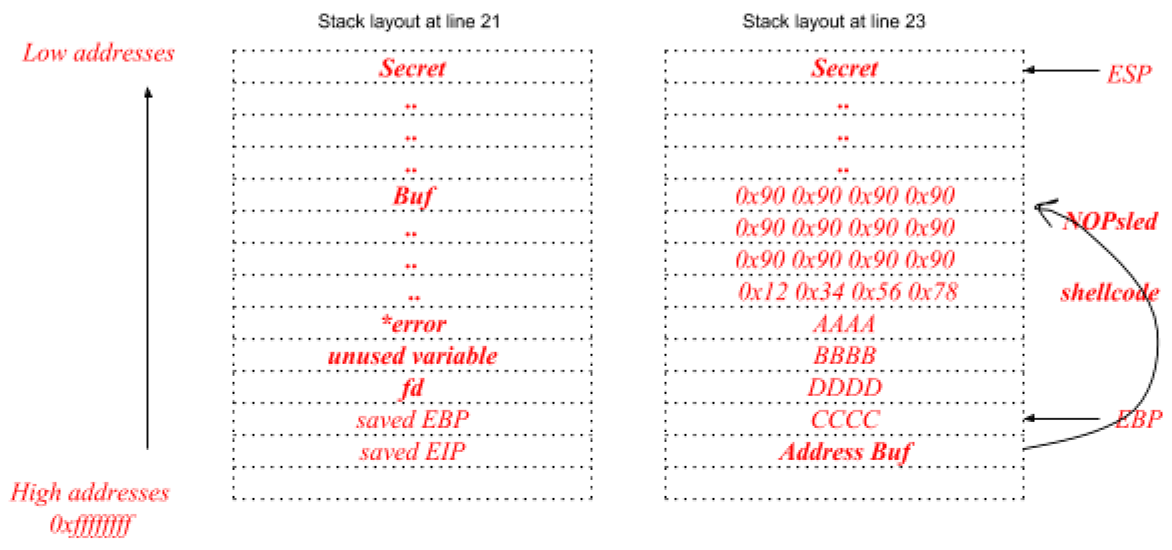
1. [4 points] Assume that the program is compiled and run with no mitigation against exploitation (**no canary, executable stack**, environment **with no ASLR** active).

The program is affected by a typical buffer overflow vulnerability. Write an exploit for this vulnerability to extract the content of the file secret.txt. For this purpose, assume that the following shellcode, composed by 4 bytes of instructions, will output the content of secret.txt: `0x12 0x34 0x56 0x78`.

Write the exploit clearly, detail all the steps and assumptions you need for a successful exploitation, and draw the stack layout right before and after the execution of the `scanf` (line 22) during the program exploitation showing:

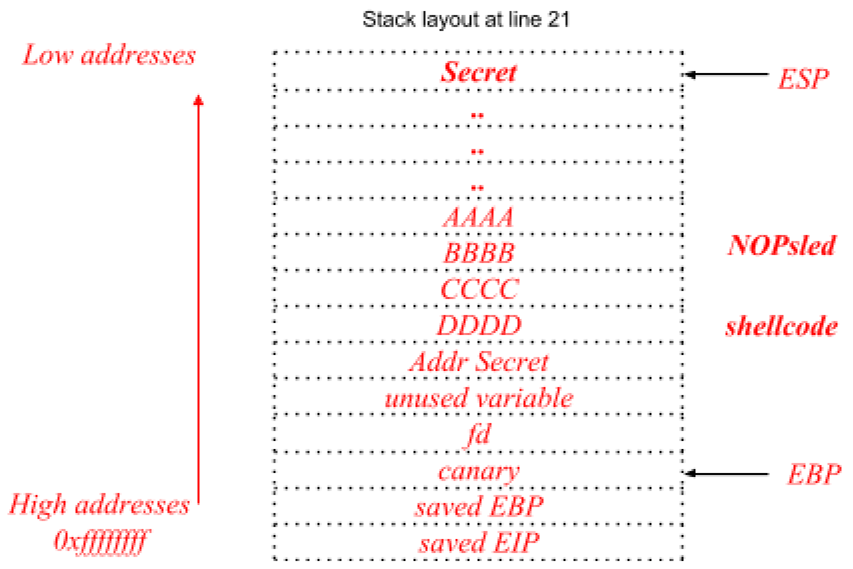
a. Direction of growth and high-low addresses;
b. The name of each allocated variable;
c. The boundaries of frame of the function frames (`main` and `login`).

Show also the content of the caller's frame (you can ignore the environment variables, just focus on what matters for the vulnerability and its exploitation).



2. [2 points] This time the program is compiled with a **canary** and the **stack is made not executable** as security mitigations. Is the exploit you wrote before still working? If not, write an exploit to read the content of secret.txt . Draw the stack layout after your attack is executed.

No, because you cannot overwrite EIP.



3. [1 points] If canary, non executable stack, and *address space layout randomization (ASLR)* are active, is the exploit you just wrote still working *without modifications*? Why?

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

4. [1 points] If **only** address space layout randomization (ASLR) is active as security mitigation, which of the two exploits you wrote is able to exploit the vulnerable program ? Why?

*Neither, because the **address of the stack** would be **randomized** for every execution, and we must have to have a **leak** in order to exploit it successfully.*

5. [1 points] Write the simplest modification to the program source code that will eliminate the vulnerability (preserving the program's intended functionalities).

scanf("%15s", l.buf);

Question 2 (9 points)

"Watson Files" is a cloud computing services to store files similar to Dropbox or Google Drive.

Customers complain that old links often return a "*404 File not found*" error. Sherlock decides to fix this problem modifying how the web server responds to requests for missing files.

The Python-like pseudocode that generates the "missing file" page is the following:

```
def missing_file(request):
    print "HTTP/1.0 200 OK"
    print "Content-Type: text/html"
    print ""
    user = check_cookie(request.cookie)
    if user is None:
        print "Please log in first"
        return

    print "<p>We are sorry, but the server could not locate file" + request.path
    print "<br>Try using the search function.</p>"
```

The code checks if the user is logged in using the function `check_cookie`, which returns the username of the authenticated user checking the session cookie. If the user is authenticated, the page will contain a message detailing the path of the file that could not be located.
Assume that the `request.path` variable is populated by the web server with the path requested by the user (e.g., if the user visits *https://www.watsonfiles.com/dir/file.txt*, the variable `request.path` contains `dir/file.txt`), and that all the functionalities concerning the user authentication (i.e., `check_cookie`) are securely implemented and do not contain vulnerabilities.

2. [2 points] Only considering the code above**,** identify which of the following web application vulnerabilities are present in the above code:

Vulnerability class	*Is there a vulnerability belonging to this class in Sherlock's code? If so, explain why it is present and specify how an adversary could exploit it.*	***If the vulnerability is present in the code above****, explain the simplest procedure to remove this vulnerability.*
Stored cross-site scripting (XSS)	No, the above code does not allow to store information on the server that can be exploited.	
Reflected cross-site scripting (XSS)	Yes, an attacker can supply a filename containing e.g., <script>alert(document.cookie)</script>, and the web server would print that script tag to the browser, and the browser will run the code from the URL. Ma	The simplest procedure to prevent this vulnerability is to apply escaping/filtering to the "request.path" variable.
Cross site request forgery (CSRF)	No, there is no state-changing action in the page that needs to be protected against CSRF.	

As part of its core functionalities, Watson Files allows every user to retrieve a shareable public download link to every file he\she can access.
The way this works is the following: when a user wants to retrieve the download link for, as an example, the file '*secret.txt*', he\she will visit a page ('/link') passing the filename as a GET parameter (i.e., https://watsonfiles.com/link?filename=secret.txt). Then, the backend code will perform access control checks and redirect him\her to https://watsonfiles.com/files/sFPVMasg , where sFPVMasg is the *token* associated with the file (every file is associated with a *pre-generated token*), the download link is accessible without authentication.

The Python-like pseudocode of the backend that manages the link retrieval is the following:

```
def retrieve_download_link(request):
  # .... code to initialize the HTTP response and to retrieve
  # the user's session information into the object "user" (see previous point)

  filename = request.params['filename']
  query = 'SELECT filename, token FROM files, permissions WHERE filename = ' + 
filename + ' AND permissions.u_id = ' + user.id + ' AND permissions.f_id = 
files.f_id;'
  db.execute(query)
  row = db.fetchone()

  # ... code to generate a redirect to https://watsoncode.org/files/<row.token>
```

where `params` is a dictionary containing the GET parameters (e.g., if a user visits */link?filename=holmes.txt*, then `params['filename']` will contain '`holmes.txt`').
The output of the query is limited to only one result.

The database queries are executed against the following tables:

users

u_id	username	password
1	Rick	1234
2	Morty	password
3	PickleRick	bestpass
4	Mr meeseeks	unknown
...	...	...

permissions

u_id	f_id
1	1
2	2
3	2
4	3
...	...

files

f_id	filename	token
1	unk	75ZsvBcU
2	secret.txt	sFPVMasg
3	flag.txt	5vcV6xdU
4	summer.jpg	xmEBFPLU
...	...	...

3. [2 points] Identify the class of the vulnerability, 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**.

4. [2 points] Write an exploit for the vulnerability just identified to get the **username** and the **password** of the **only** user authorized to access the file 'flag.txt'. By assuming that filenames are unique, state all the necessary steps and conditions for the exploit to take place.

```
As an authenticated user I make a GET request with the following 'filename'
parameter:

' UNION SELECT username, password FROM users, permissions, files WHERE
permissions.u_id = users.u_id AND permissions.f_id = files.f_id AND
files.filename = 'flag.txt';--
```

5. [1 points] Explain the simplest procedure to remove this vulnerability.

Use parameterized queries, ...

6. [2 points] What is a **blind SQL injection**? Does it mean it cannot be exploited?

In a blind injection the data retrieved by the modified SQL query is not displayed back to the attacker.

This can still be exploited: changes can be blindly executed in the database, or by using side-effects of queries the attacker can guess the answers.

Question 3 (9 points)

An application server in the network of Politecnico di Milano, with IP address **131.175.14.10/24** and MAC address ending with **7a:ce:29**, is trying to connect to a database server in the same network with IP address **131.175.14.12/24** and MAC address ending with **7a:ce:25**. We suspect that this server is victim of an attack. Indeed, observing the network traffic, we notice the following pattern:

```
7a:ce:29 → ff:ff:ff (broadcast)    [ARP] Who has 131.175.14.12? Tell 131.175.14.10
4b:74:28 → 7a:ce:29                [ARP] 131.175.14.12 is at 4b:74:28
4b:74:28 → 7a:ce:29                [ARP] 131.175.14.12 is at 4b:74:28
7a:ce:25 → 7a:ce:29                [ARP] 131.175.14.12 is at 7a:ce:25
4b:74:28 → 7a:ce:29                [ARP] 131.175.14.12 is at 4b:74:28
```

1. [2 point] Describe what attack you think is going on, and what is the feature (or lack thereof) of the involved protocol(s) that enable this attack.

ARP spoofing, see slides, the protocol lacks authentication.

2. [1 point] Describe what you think is the **concrete** goal(s) of the attack in this scenario

Intercepting or tampering with the communications between the app server and the DB server, or denial of service

3. [2 point] Can you tell the IP and MAC address of the attacker? Why?

No, the IP address is spoofed for sure, and the MAC address can be spoofed as well.

4. [2 point] Can the application server detect the attack? And the database server? How?

The application server can detect that there are multiple ARP responses with different mac addresses for the same ARP request (note that this "technique" may be used by legitimate networks to implement e.g., redundancy of the gateway, thus it's prone to false positives). The database server can't detect the attack in a common switched scenario, as the attacker's response is not broadcast to the entire network.

5. [2 point] An attacker wants to target the above server (131.175.14.10) from the comfort of its home, where he/she is connected to the Internet through a residential ISP connection (the ISP assigned to the attacker the public IP address 79.54.132.120).
Can this attacker launch the above attack from this setting? Why?

A: No, the attacker and the victim must be on the same network (same broadcast domain)

Question 4 (5 points)

1. [3 points] Consider **computer malware**. Please describe the difference between a **virus**, a **worm** and a **trojan**.

virus	*See slides*
worm	*See slides*
trojan	*See slides*

2. [2 points] A new malware just broke out, causing a world-wide infection and a huge amount of damages. Unfortunately, all the anti-malware systems are not able to detect this malware. You were able to retrieve a couple of samples.

Consider the code snippets reported below, extracted from the two malware samples you retrieved:

Sample 1

```
pop  ebx
lea ecx, [ebx + 42h]
push ecx
push eax
push eax
sdt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

Sample 2

```
pop  ebx
lea ecx, [ebx + 42h]
push ecx
push eax
nop
push eax
inc eax
sdt [esp - 02h]
dec eax
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

It is clear that the malware is showing evasive behavior. What technique is implemented? How this technique works?

Metamorphism. See slides.