Computer Security Exam

Professors M. Carminati & S. Zanero

Milan, 14/09/2018

Last (family) Name	______________________________
First (given) Name	______________________________
Matricola or Codice Persona	______________________________
Have you done any challenges/homework, even partially?	[ ] Yes [ ] No
Professor	[ ] Carminati [ ] Zanero

Instructions

- The exam is composed of 12 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 <u>avoid writing in red</u>.
- No extra paper is allowed.
- The answers should be written exclusively in the space provided below the questions.
- <u>Always motivate your answer.</u>

READ CAREFULLY <u>ALL THE POINTS</u> OF EACH QUESTION <u>BEFORE</u> WRITING YOUR <u>ANSWER</u>

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 now the C program below:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4
5   void vuln() {
6     struct {
7         char flag[4];
8         char buf[16];
9         char *p;
10        int tmp;
11     } s;
12
13    s.p = s.flag;     //  <--------- here
14    scanf("%s", s.buf);
15    if (strncmp(s.buf, "H4CK", 4) == 0) {
16       scanf("%s", s.p);
17       printf("flag: %s", s.p);
18    }
19  }
20
21  int main(int argc, char** argv) {
22    vuln();
23  }
```

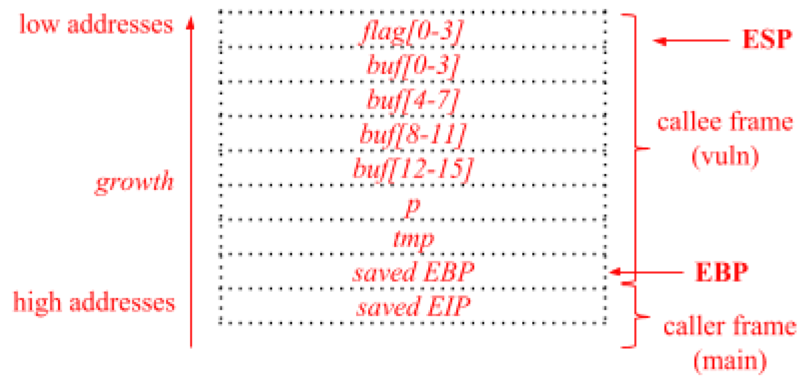
1. [1 points] The program is affected by typical vulnerabilities. Complete the following table, focusing on a vulnerability per row.

Vulnerability	**Line**	**Motivation**
Buffer Overflow	*Line 14,16*	*the scanf reads an user-supplied string of arbitrary length and copies it in a stack buffer*
Format String	*NO VULN*	*No format string, is directly supplied by the (untrusted) user.*

2. [1 points] 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, and the stack is executable).
Draw the stack layout when the program is executing the instruction at line 13, 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 `vuln`).

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



Consider the following compiler-level mitigation: the compiler inserts the following code at the beginning of the function (function prologue):

```
    pushl $0xdeadbeef
    pushl %ebp
    mov %esp, %ebp
```

and, at the end of the function (epilogue), just before the ret, it inserts the following code:

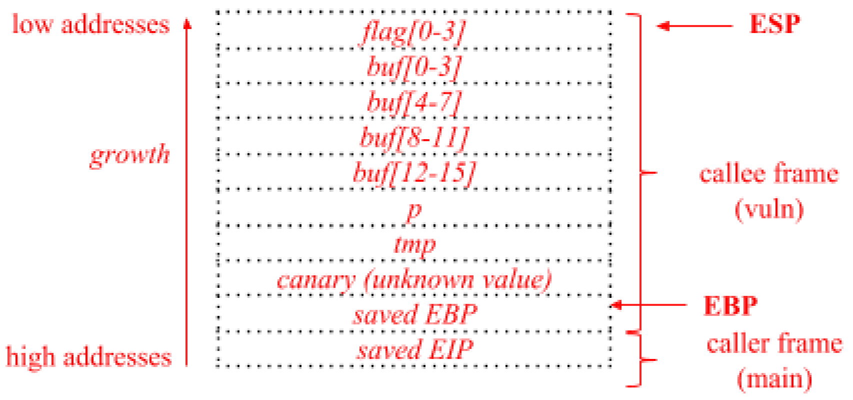
```
    movl %ebp, %esp
    popl %ebp
    cmpl $0xdeadbeef, (%esp)
    jeq function_end
    hlt  // abort the program

  function_end:
    ret
```

3. [1 points] Describe in detail: (a) what the assembly code does, (b) the name of the technique that is being implemented, (c) the weakness in this implementation, (d) a correct implementation of the same mechanism

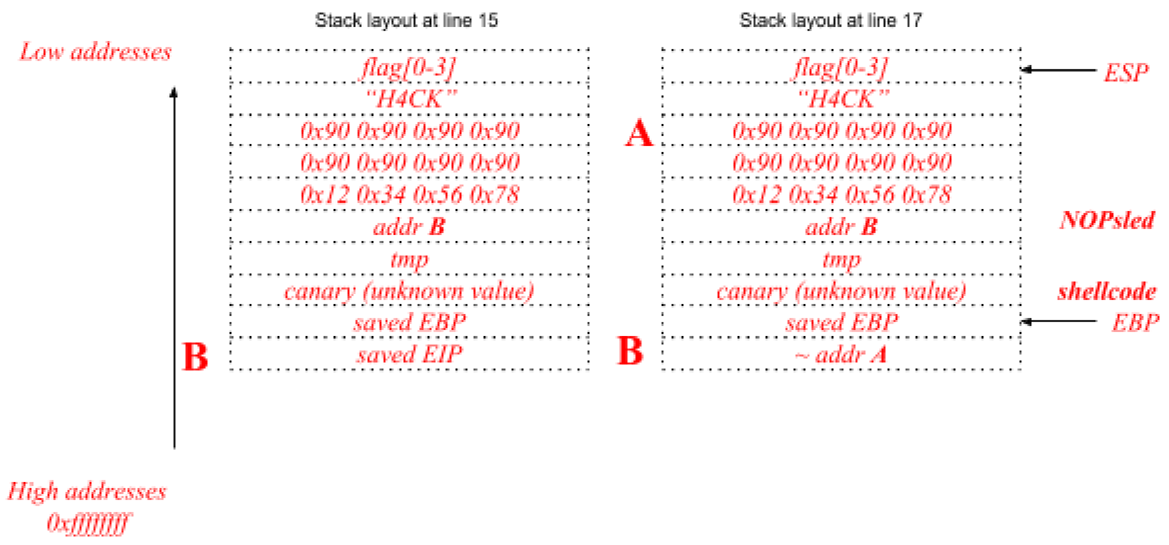
(a) This is a canary implementation. pushl $0xdeadbeef pushes a static value on the stack. At the epilogue, the value is checked by cmpl $0xdeadbeef,(%esp) and if it has been changed execution is diverted. Otherwise the function returns normally.
(b) Static canary
(c) The canary value is statically set in the code. the attacker can learn it by means of a debugger, and then bypassing the check becomes trivial.
(d) The canary should be randomized and stored in a register at program initialization.

4. [1 points] Assume the usual IA-32 architecture (32-bits), with the usual "`cdecl`" calling convention. Also assume that the program is compiled only with the correct implementation of the compiler-level mitigation of question 3 (the address space layout is not randomized, and the stack is executable).
Draw the stack layout when the program is executing the instruction at line 13, showing:
d. Direction of growth and high-low addresses;
e. The name of each allocated variable;
f. The boundaries of frame of the function frames (`main` and `vuln`).
Show also the content of the caller frame (you can ignore the environment variables, just focus on what matters for the vulnerability and its exploitation).



5. [4 points] Write an exploit for the buffer overflow vulnerability in the above program. Your exploit should execute the following simple shellcode, composed only by 4 instructions (4 bytes): `0x12 0x34 0x56 0x78`. Write clearly all the steps and assumptions you need for a successful exploitation, and show the stack layout after the execution of each `scanf()` during the program exploitation. You can assume that addresses in the stack are known precisely.

- *Step 1: Through the scanf() on line 14, write the shellcode on the stack and overwrite buf with the address of a memory location below the canary - namely, the address of the saved EIP. Also, we need to start the buffer with "H4CK!" to actually enter the correct branch of the "if" statement.*
- *Step 2: In the scanf() on line 16, we can directly overwrite the saved EIP bypassing the canary.*



6. [1 points] If *address space layout randomization (ASLR)* is active, is the exploit you just wrote still working *without modifications*? 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.*

7. [1 point] If the stack is made non executable (i.e., NX, DEP, or W^X), is the exploit you just wrote still working *without modifications*? If not, propose an alternative solution to exploit the program.

- *No, the exploit executes shellcode from the stack.*
- *We can use ret to **libc** technique in order to execute the **system** function in the libc and obtain a **shell**. In order to do that, we change the value of **B** and let it point to the **address of system**.*

Question 2 (10 points)

Consider a mobile application that allows users to send each other simple text messages. Under the hood, the application uses a simple HTTP-based API (pseudocode listed below) to handle user login and registration, as well as message sending and retrieval. Users' sessions are handled via HTTP cookies, and users must be logged in to be able to send and receive messages.

Registration: https://api.example.com/register?username=<user>&password=<password>

```
def register(request, response):
  username = db.escape(request.get['username'])
  password = db.escape(request.get['password'])
  q = db.query('INSERT INTO users (username, password)' +
               ' VALUES (' + username + ',' + password + ');')
  response.status_code = 200       # OK
  return response
```

Login: https://api.example.com/login?username=<user>&password=<password>

```
def login(request, response):
  username = db.escape(request.get['username'])
  password = db.escape(request.get['password'])
  q = db.query('SELECT * FROM users WHERE username = ' + username +
                                  ' AND password = ' + password + ' LIMIT 1;')
  if q is NULL:
     response.status_code = 403      # Forbidden
     return response
  response.status_code = 200         # OK
  response.add_header("Set-Cookie: username=" + username);   # Session creation
  return response
```

Send a message: https://api.example.com/send?to=<recipient>&msg=<text_of_message>

```
def send(request, response):
  username = db.escape(request.cookies['username'])
  if username is NULL:
     response.status_code = 403      # Forbidden
     return response
   to = request.get['to']
   msg = request.get['msg']
   db.query('INSERT INTO messages (sender, recipient, msg, timestamp)' +
           ' VALUES (' + username + ', ' + to + ', ' + msg + ', NOW());')
   response.status_code = 200
   return response
```

Retrieve all the messages: https://api.example.com/retrieve

```
30 def retrieve(request, response):
31   username = db.escape(request.cookies['username'])
32   if username is NULL:
33      response.status_code = 403      # Forbidden
34      return response
35   q = db.query('SELECT sender, msg, timestamp FROM messages ' +
36                                    'WHERE recipient = ' + username + ';')
37   for row in q:
38       response.text += '{ From: ' + row.sender + ', Msg: ' + row.msg + '}, '
39   response.text = '[' + response.text + ']'
40   response.status_code = 200
41   return response
```

As it is clear from the code, this application uses a database to store data. The database is structured according to the following schema:

users

id	*username*	*password*
1	s.zanero	76ceaaa34826979e77
2	m.carminati	563c39089151f9df26
3	m.rossi	d1e576b71ccef5978d

messages

id	*sender*	*recipient*	*msg*	*timestamp*
1	m.carminati	s.zanero	"Welcome!"	03:23:21 24/08/2018
2	m.rossi	s.zanero	"I have a question on the course material"	18:31:62 1/09/2018

Assume the following:

- *The dictionaries request.get and request.cookies store the content of the parameters passed through a GET and the content of the Cookies header, respectively;*
- *The methods of the object response can be used to set the various fields of a response (cookies, body, status code, ...);*
- *The function db.escape securely implements database escape functionalities and does not contain vulnerabilities itself.*

1. [1 point]. This API is affected by a typical SQL injection vulnerability. Specify the line(s) of code that introduce this vulnerability, and explain the simplest procedure to remediate to it.

Lines 24 and 25 populate two variables, to and msg, with untrusted input; such variables are then used in a SQL query by simple string concatenation (no escaping and no prepared statements)

Remediation: db.escape(request.get['to']); db.escape(request.get['msg']), or use prepared statements in all the queries

2. [3 points]. Exploiting the vulnerability you just found, write the following exploits (detail all the steps and assumptions, if any, you need to successfully execute your exploit):

a) Write an exploit to send two messages from `f.resta` one to `m.carminati` and one to `s.zanero` with the text `This exam is too difficult! You are fired!`

Logged in as an arbitrary user, e.g., the user 'the.attacker', we want to perform the query: INSERT INTO messages (sender, recipient, msg, timestamp) VALUES ('the.attacker', 'any.user', '', NOW()), ('f.resta', 'm.carminati', 'This exam is too difficult! You are fired!', NOW()), ('f.resta', 's.zanero', 'This exam is too difficult! You are fired!', NOW());

Exploit: https://api.example.com/send?to=any.user&msg='', NOW()), ('f.resta', 'm.carminati', 'This exam is too difficult! You are fired!', NOW()), ('f.resta', 's.zanero', 'This exam is too difficult! You are fired!

b) Write an exploit to retrieve the password of the user `s.zanero`

Once again, logged in as 'the.attacker', let's send us a message with s.zanero's password: INSERT INTO messages (sender, recipient, msg, timestamp) VALUES ('the.attacker', 'to', '', NOW()), ('s.zanero', 'the.attacker', (select password from users where username='s.zanero'), NOW())-- ', NOW());

Exploit: https://api.example.com/send?to&msg=', NOW()), ('s.zanero', 'the.attacker', (select password from users where username='s.zanero'), NOW())-- ;
then just visit https://api.example.com/retrieve

3. [2 points]. Assume that you have no way to modify the source code of the web API, but you want to mitigate the damage that an attacker can do by exploiting the SQL injection vulnerability.

a) Assume that you are the database administrator. How would you mitigate the damage that an attacker can do? If so, how? Which of the exploits 2.a) and 2.b) you would effectively mitigate?

We could restrict, at the database level, the privileges of the user of this application to only perform SELECTs and INSERTs involving tables actually used. However, by naively doing so, we would mitigate neither of the exploits. Noting that the only vulnerable functionality is in the INSERT query (endpoint 'send'), and that queries using the 'users' table do not use the 'message' table and vice-versa, we could use a different DB connection and a different user for the queries related to user management and for the queries related to sending and receiving messages. This way, we can restrict the privileges so that the DBMS-level user involved with managing the message table can't access the user table --- effectively mitigating this specific password-disclosing exploit.
Instead, there is no way to mitigate via privilege restriction the exploit 2.a).

b) Assume that you are the administrator of the frontend proxy, and that you can only add or remove HTTP headers to each request and response. Can you mitigate the damage an attacker can do? If so, how? Which of the exploits 2.a) and 2.b) you would effectively mitigate?

As the vulnerability is in the interaction between the application and the DBMS, changing HTTP headers has no effect on mitigating the vulnerability.

4. [2 points]. Now, assume that the SQL injection vulnerability was completely remediated. Unfortunately, there is a different vulnerability (i.e., NOT a SQL injection) that allows an attacker to write a different exploit with the same effect of 2.a), i.e., that allows to send arbitrary messages with an arbitrary sender. Find this vulnerability, and write an exploit for it, detailing all the steps and assumptions needed for a successful execution. Then, describe a technique to mitigate or remove altogether this vulnerability (assume you are free to modify the source code of the web API).

Authentication bypass as the cookie is just the username.

Exploit: Send a HTTP request with Cookie: username=s.zanero and use the API as intended, i.e., send a GET request to https://api.example.com/send?to=m.carminati&msg=You are fired…

Fix:

- *Encrypt the cookie with a key stored on the server (with a nonce and an expiration to avoid replay attacks)*
- *Store the session data somewhere (e.g., file, database, …) indexed by a random value and set the random value in the cookie instead of the username*

5. [2 points]. Consider how the API implements the storage of users' passwords. Clearly, if there is any vulnerability that allows an attacker to disclose the content of the database, such as the one you just found and exploited, an attacker can leak the passwords of all the registered users.

a) Explain what is the risk for the users of the website exposed by such a password leak, in the scenario where all the website's data is compromised (i.e., impersonating a user is not worth for the attacker)

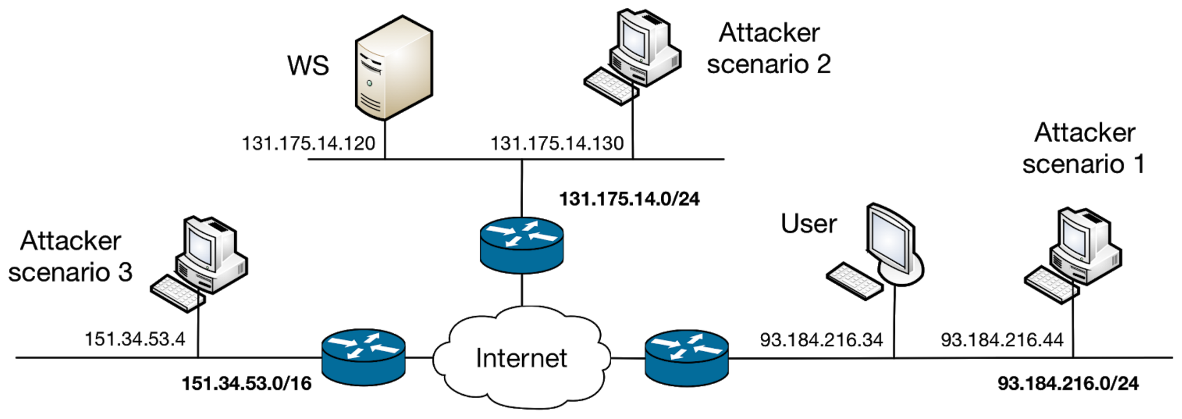
Risk: users reuse password all the time -> the password leak can be used to gain access to more sensitive sites, e.g., Gmail, corporate logins, …
Also the leak of personal information such as password could be used to craft effective phishing campaign ("here's your password! <password>…")

b) As this risk is too much for your users, propose a change to the application that avoids the threat of leaking users' passwords, assuming that the website and the database are completely compromised.

Hash and salt the passwords with a strong algorithm before saving them to the DB, to avoid leaking plain-text password and to avoid easy offline bruteforce in case of DB compromise.

Question 3 (8 points)

Consider the following network diagram:



A user, with IP address 93.184.216.34, is attempting to download a software executable from a webserver in the Politecnico di Milano network, http://downloads.polimi.it, with IP address 131.175.14.120, over the HTTP protocol (no HTTPS, no signatures, nothing). Assume that the user's browser already cached the IP address of downloads.polimi.it (i.e., it does not perform any DNS request), and that there is no firewall involved. An attacker, who knows that the user is about to download this software, wants to target our user by carrying out an attack to replace the downloaded software with a piece of malware.

1. [3 points]. For each of the following attack scenarios, state whether the attacker is able to fulfill his\her goals. If you deem it possible, describe an attack that allows to do so: state the name of the class of attacks, and describe all the steps needed to make it work in this specific scenarios. If multiple classes of attacks are possible, focus on the simplest one that gets the job done. If no attack is possible, please explain why.

Scenario 1 (attacker: 93.184.216.44; same network and broadcast domain of the user):

ARP spoofing ...

Scenario 2 (attacker: 131.175.14.130; same network of web server, but different than user):

ARP spoofing ... (same as before but this time target the server)

Scenario 3 (attacker: 151.34.53.4; attacker, user and webserver on three different networks):

No attack possible, because HTTP uses TCP and, if sequence numbers are correctly implemented, not possible to perform TCP hijacking. Also, DNS poisoning not possible as DNS response already cached.

For the next questions consider ONLY scenario 3. Assume that each involved computer and server implements a custom TCP/IP stack that, for performance reasons, sets the TCP initial sequence number in the SYN and SYN+ACK packets as the most significant bits of the current timestamp.

2. [2 points]. Describe the security issue with the proposed ISN implementation, and propose a way to solve the issue

Can predict ISN -> solve by using random ISN

3. [2 points]. Describe how the attacker can perform the above attack, this time *exploiting the security issues raised by the custom ISN implementation*. Describe all the steps and assumptions that you need to perform this attack.

TCP hijacking. We can guess the ISN of the SYN packet sent by the victim (the user), we can spoof the webserver IP and send a "correct" SYN+ACK to it and, if we can guess the content of the request (we do), subsequent packets. This way we can send a different payload. In parallel we can also use TCP hijacking to send a fake RST to the actual server spoofing the user's IP address.
Problem\assumption: we need to know the ephemeral port used by the client to initiate the connection.

4. [1 points]. Assume you are the network security administrator of the network of the attacker (in the scenario 3 and assuming the custom TCP initial sequence number implementation), and that you control the border router between the network 151.34.53.0/24 and the rest of the Internet. Propose a way to prevent the attack. Can the administrator of the other two border routers in the diagram deploy the same mitigation, and obtain the same result? Why?

We can filter the packets with source IPs not belonging to our network.

Other routers can't do this, they can only filter out packets coming from "outside" with spoofed IPs belonging to their network, but it's of little use in this scenario.

Question 4 (4 points)

1. [1 point] Explain what is a rootkit

See slides

2. [1 point] Explain the difference between a user-land and kernel-land rootkit

See slides.

3. [1 point] You suspect that your machine have been compromised with a kernel rootkit. You tried to use network traffic tools from your machine but you do not see any malicious traffic. Can you conclude that your machine is safe? If is not there are other way to prove you have been compromised?

No you cannot conclude that the machine have not been compromised. Because the malware can hide its own traffic from tools running on the compromised machine. You could inspect network traffic using an external machine as a MitM between your machine and the router.

4. [1 point] A colleague suggests to replace the hard drive of a machine to be sure to get rid of a very sophisticated rootkit. However, after reinstalling the operating system, it seems like that the machine is infected by the same rootkit. Provide an explanation of what happened. Whatever your answer is, explain why.

If it is a BIOS rootkit then No. If it is a kernel rootkin it is ok to just replace the HD or even just reinstall the OS.