Report for OVS Assignment: Casper Hacker Wargame

Xueying Deng r0601458

1. Overview

Category	Level	Time
Stack-based buffer overflow Advanced Stack-based	Capser4 LBc1dJGMRUDGGdWCnmo2E3F0QYVMmBmv Capser40 B6OyaozZDg3ysFmEaYeGAzaUP8GbN4An	10 hours
Heap-based buffer overflow Advanced Heap-based	Casper 6 qCeyA3Rb1Xuw4tpDXoaNFftxSU9DwjHJ Casper60 dUPC70LY2jqnpog0cBpZtoSTDSnLD1UG	8 hours
Return-to-libc Advanced Return-to-libc	Casper8 HEpkEkasRhTGxvMxOIHV9L11XPCgw1p6 Casper80 5GOVGN27jm8yoPhAD8IeFO1EBBbQB1OK Casper82 6VeOJIQztAs4lnx9VPvQvlKJmI3hh69U	14 hours
Data-only vulnerability	Casper10 oMX51Zm1PJAJPp25uagP7RUOHRjHizgN	8 hours

2. Level 1 Stack-based buffer overflow & Advanced Stack-based

- **What does the level do?**

I did Casper 4 and Casper 40 in this level, this level is dealing with stack buffer-overflow . We need to know the layout of system in the stack . And the address in the stack is from higher address goes to the lower address.

- **What is the vulnerability?**

strcpy(buf, s) is not safe, which could lead to buffer overflow , because it doesn't check whether the available memory space can meet the space required by s .

• How did you exploit it?

Step1, using an unexpected input causes the system go to the ***Program received signal SIGSEGV, Segmentation fault problem.***

In capser4 , the memory is filled by /x90, but in casper40, it detects the input of /x90, so I changed to /x80(random input) to fill the memory. Shell code is among these memory filler, /x90 can lead the system to run the shell code, so we just need to let the system jump in to the buffer stack for Capser4, but Casper40 need the exact address of the shellcode.

Step2, by running the system inside gdb and using command ***(gdb) x/50x $esp-600*** , I can get the address of the shell code.

Step3, using shell code address replaces ***eip*** , which can leads the system run into the shell, (Casper4 doesn't require the address of shell code by very accurate ,but Capser40 need the exact address of the shellcode.)

• What solution do you propose to remove the vulnerability?

Using safe function like strcpy_s() instead of strcpy().

Or in the inner logic of function ,the system can check the input length , like input validation checking to enhance the security of the system .

• (if applicable) What advanced level did you choose?

Casper40

• (if applicable) Why does your exploit for the base-level not work on the advanced level?

Advanced level requires the filler can't be '\x90' and it also required the attacker find the exact address of shell code.

• (if applicable) How did you modify your exploit to evade the additional checks?

As I mentioned before, I changed filler and used gdb to find the accurate address of shellcode.

3. Level 2 Heap-based buffer overflow & Advanced Heap-based

• What does the level do?

I did Casper 6 and Casper 60 in this level, this level is dealing with heap buffer-overflow .And we also need to know the address layout of heap is from lower address to higher address.

• What is the vulnerability?

It is similar with Casper4 , it uses **strcpy** to copy the inputted string to buffer without checking the length of input, which may lead to buffer overflow. When the buffer overflowed it will overwrite the return address and next executable function pointer.

• How did you exploit it?

Step1, using an unexpected input causes the system go to the ***Program received signal SIGSEGV, Segmentation fault problem.*** The buffer is filled by \x90

Step2, by running the system inside gdb and using command ***(gdb) x/50x $es+500*** , I can get the address of the shell code. ***The only difference with Casper4 and Casper40 is here, the heap is from low address to high address,but stack is from high address to low address.***

Step3, using shell code address replaces ***eip*** , which can leads the system run into the shell, (Casper4 doesn't require the address of shell code by very accurate ,but Capser40 need the exact address of the shellcode.)

• What solution do you propose to remove the vulnerability?

Using safe function like strcpy_s() instead of strcpy().

Or in the inner logic of function ,the system can check the input length , like input validation checking to enhance the security of the system . And random memory layout .

• (if applicable) What advanced level did you choose?

Casper60, Casper60 and Casper 6 are using the same code.

4. Level 3 Return-to-libc & Advanced Return-to-libc

• What does the level do?

I did Casper 8 and Casper 82 , This level is with a non-executable stack ,but it also can be attacked by using buffer-overflow. It is quite tricky to set the runtime environment of the function, and we need to let the system call /bin/xh automatically. And the level is about Return-to-libc attack .

• What is the vulnerability?

Same problem with previous tasks ,Casper 8 takes an argument without checking its strength and directly copies it to an array. Even though, it takes advantage of non-executable stack, but if the attacks inject the address of the function they want to execute properly, the system is still under high risk of being attacked.

• How did you exploit it?

Step1, export \bin\xh to the environment

Step2, use the code[1] to get runtime address for \bin\xh . First we need to set the runtime enviroment of /casper/capser8 to guarantee it is not random address by using ***env -i SHELLMINE=\bin\xh ,*** then we can get a fixed runtime address for SHELLMINE which is ***0xbfffe9***

Step3, using gdb, to get runtime system address ***0xb7e6b0b0***

Step4, using x\90 fills the memory ,putting the address of system to EBP, using fake

_ret (SEXY) fill EIP, then concatenate the address of \bin\xh

- **What solution do you propose to remove the vulnerability?**

Using safe function like strcpy_s() instead of strcpy(), and current days most systems takes the advantage of ASLR (Address space layout randomization) to assign random address to prevent the attack .

- **(if applicable) What advanced level did you choose?**

Casper82

- **(if applicable) Why does your exploit for the base-level not work on the advanced level?**

Advanced level requires the filler can't be **ASCII** , so the filler which is used here is A, and it also check “\0”, so which requires to adjust the length of the input .

- **(if applicable) How did you modify your exploit to evade the additional checks?**

As I mentioned before, I adjust the filler. And Casper80 is using same code with Casper82

5. Level 5 Data-only vulnerability

- **What does the level do?**

I did Casper 10 in this level, it has some problems of data input function . It is about string formatting attack.

- **What is the vulnerability?**

The IO function ***printf(buf)*** here, it doesn't provide the type of the output, which may lead the system being attacked by string formatting attack.

- **How did you exploit it?**

Step1, get the address of isAdmin by using gdb, what we are trying to do here is to rewrite the value of isAmin, the address is 0x8049940

Step2, using /casper/casper10 `python -c 'print "AAAA"+"%x"*"%x"*10'` to get the offset, like how many %x should be ahead the address then we can access to the address. Here is 10

Step3, makeup the string to let isAdmin to be overwritten .

- **What solution do you propose to remove the vulnerability?**

Constraint the type of output ,like using printf(“%s”,buf) instead of printf(buf).

Reference

[1] Jon Erickson *https://security.stackexchange.com/questions/108167/environment-variable-and-scripting-for-return-to-libc-exploit*