jamk.fi

Software Exploitation

Assignment 5 essay

Mikael Romanov

Assignment
Maaliskuu 2018
Tieto- ja viestintätekniikka
Kyberturvallisuus

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

1 Table of Contents

1 Introduction

This assignment was given by Mikko Neijonen. The assignment task was to write an essay about a topic of egg hunter shellcode, return-to-libc, return oriented programming or heap buffer exploitation. This assignment was written about return-to-libc. The essay contains general information of the return-to-libc, how the exploit works and protection methods against the attack.

2 Return to libc

The return to libc was introduced by Solar Designer in 1997. He introduced a way to overwrite the return address of a vulnerable function with libc function. The idea of the exploit was to gain access or control a system.
http://seclists.org/bugtraq/1997/Aug/63

The return to libc is a way to exploit buffer overflow a system that has a non-executable stack via buffer overflow. Return-to-libc is a used method that defeats Linux system stack protection (bypass the NX-bit). The NX prevents program runs and try to execute code in the NX regions, it will cause an error. The NX bit segregates certain areas of the memory to storage data or the processors instructions and is used in processors for security reasons. It is used to prevent malicious programs from taking over another program by inserting code into the programs storage area and running it within the section. The libc idea is to overwrite the return address and point it to a different location that the exploiter can control, so it works almost same way as standard overflow. The problem with non-executable stack is that e.g. shellcode cannot be used by itself, so there is a libc trick that can be used. The libc is just a common library in C programming language (Lib C), which is called system and takes in a command and executes it as a system call. Because the Lib C functions do not reside on the stack, they will allow to bypass the stack protection and execute code. Most modern Linux systems have stack protection to prevent execution from the stack example Fedora allows an administrator to make a stack non-executable, but it doesn't make it fool-proof. The protection method in the processor is called NX-bit which defines all the non-exec regions

The normal buffer overflow is used to overwrite the save frame pointer (EBP) and saved return address (EIP) to redirect to the attackers shellcode that is in the stack in buffer or in an environment variable. The implementation of stack protection allows overflow to happen, but doesn't allow executable instructions form environment variables and the stack isn't executable. The way to bypass the stack protection is to overwrite the return address (EIP) with a libc function address and arguments with a dummy return address which is treated as a valid function call.

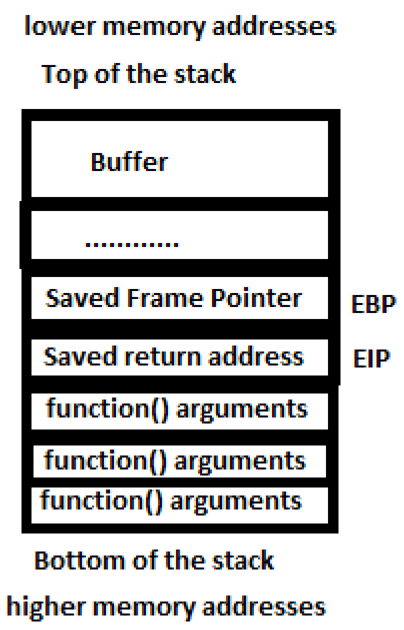

Figure 1 stack

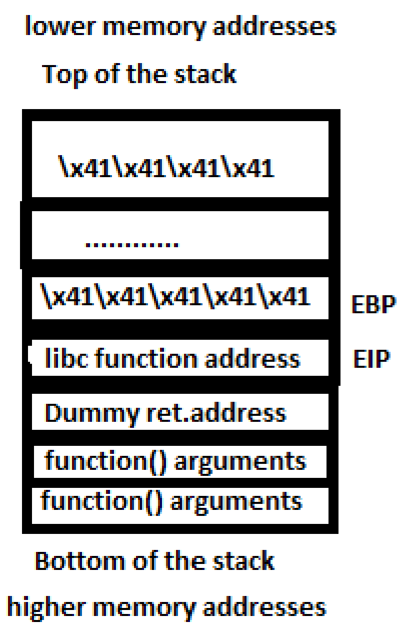

Figure 2 stack return to lib

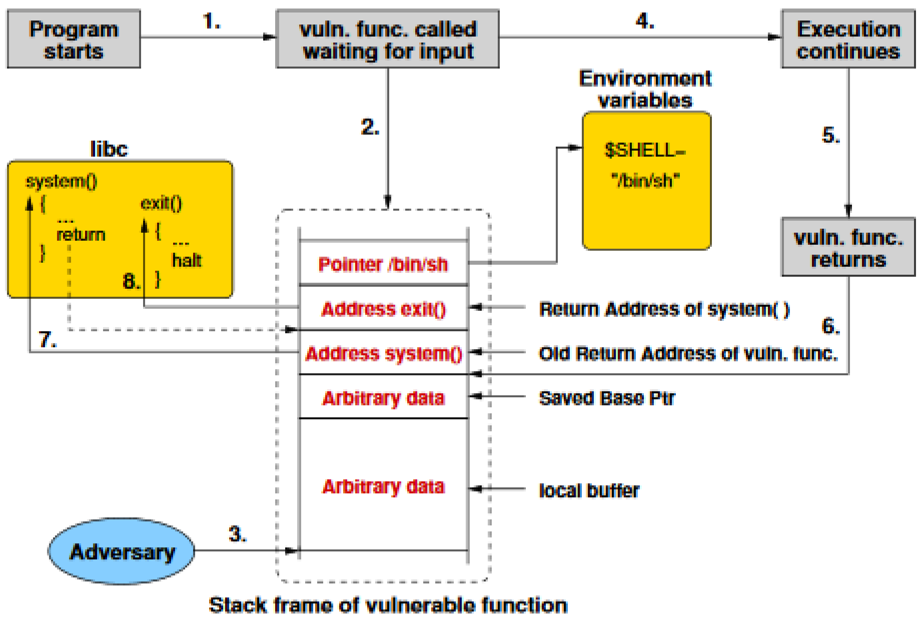

Figure 1 Return to libc https://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/LectureSlides/STC-WS2011/Blatt06_WS11-12.pdf

The idea of the return to libc attack is to find the addresses of the system(), exit() and /bin/bash. Use them with padding to overwrite the return address and e.g. gain access to the system. These addresses can be found e.g. via gdb by the example show below:

- system() & exit():
  - gdb: break main
  - gdb: r
  - gdb: p system
  - $1 = {<text variable, no debug info>} MEM.ADDR <system>
  - gdb: p exit
  - $2 = {<text variable, no debug info>} MEM.ADDR <exit>
- /bin/bash/
  - gdb: x/s *((char **)environ)
  - -using enter until “SHELL=/bin/bash” pops up
  - MEM.ADDR “SHELL=/bin/bash”

- The needed part from shell is only the /bin/bash location, so the "SHELL=/bin/bash" address needs to be modified until /bin/bash is located, lets say the address is 0xbffff59a
- x/s 0xbffff59c
- (gdb) "LL=//bin/bash"
- x/s 0xbffff59e
- (gdb) "=//bin/bash"
- x/s 0xbffff59f
- (gdb) "/bin/bash"

Now that the system, exit and bash addresses are known, the exploit can be used by using padding before the return address(EIP). Overwriting the return address system address + exit address + /bin/bash address.

There are plenty of methods to protect against the attack, but it isn't possible to avoid 100%. The protection methods are made from core Kernel to compiler protection and they make the attack way more difficult to perform. The normal type of protection is stack randomization, removing executable memory regions, stack canary value, library randomization, produce linkage table (PLT) separation and global offset separation (GOT). These methods mentioned bring each a little more protection against heap and stack overflows. There are also applications that are designed to prevent buffer overflows and retun-to-libc. A good example of the program would be PaX and ASLR, which I viewed on my earlier essay. Even with all the possible ways to protect against the attack, the hackers that develop these type of attacks are beyond smarter than the developers that made the program. Since it is impossible to have good defense and features, developers usually need sacrifice something and the hackers find a way to exploit that. Example ways of exploiting and breaking the protection by using memory leaking, return to GOT/PLT and canary replay.

3 Conclusions

Even though this vulnerability was introduced in 1997, it is some way valid even though of the prevention methods. The vulnerability depends highly on the programs construction and used functions. It is highly implausible that any real programmer uses deprecated functions. Even though if the functions aren't used, the prevention methods can be exploited by the ways I mentioned on this essay. This topic was kind of hard to research since there was really no real documentation about the exploit, probably due to the fact it was introduced so many years ago when internet wasn't a huge source of information. I would've wanted to read more about the exploit and research it in a deeper way, but couldn't get hold on good material. All the sources I used in the research were kind of similar and had only examples of the attack, not the theory behind it. This was a niche topic I chose, but I learnt some new things by testing the exploit.

4 Sources

https://css.csail.mit.edu/6.858/2017/readings/return-to-libc.pdf

http://www.cis.syr.edu/~wedu/seed/Labs/Vulnerability/Return_to_libc/Return_to_libc.pdf

https://www.exploit-db.com/docs/english/28553-linux-classic-return-to-libc-&-return-to-libc-chaining-tutorial.pdf

https://everything2.com/title/return-to-libc+attack

http://seclists.org/bugtraq/1997/Aug/63

https://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/LectureSlides/STC-WS2011/Blatt06_WS11-12.pdf