
mindthesec
Edição
2017
Rio de Janeiro
18 de maio
Blind Remote Exploitation
Ighor Augusto

Abstract

whoami

Ighor Augusto Barreto Candido

Founder and CTO at Intruder Security

intruder

SEGURANÇA DA INFORMAÇÃO

Objective

- How to write a generic exploit
- How to build a ROP chain in time of exploitation
- How to hack proprietary closed-binary services
- How to bypass anti-exploitation techniques

Disclaimer

- Security really exists?
- How do you expect to get security without understanding the hacker's mind?
- For whom is this talk?
- Penetration testers nowadays

Requirements

- What must we know?
  - Linux syscalls
  - x86 architecture
  - C programming
  - Sockets and network programming
  - Basics of software exploitation
  - Return Oriented Programming
- The same idea can be ported to another Operational Systems!

Introduction

Scenarios

1. Hacking an open-source server for which the binary is unknown

2. Hacking a known vulnerability in a library thought to be used in a proprietary closed-binary software

3. Hacking a proprietary closed-binary services

Scenario example

1. Hacking an open-source server for which the binary is unknown
   - Proftpd Telnet IAC buffer overflow (CVE-2010-4221) vulnerability as example
2. Hacking a known vulnerability in a library thought to be used in a proprietary closed-binary software
3. Hacking a proprietary closed-binary services

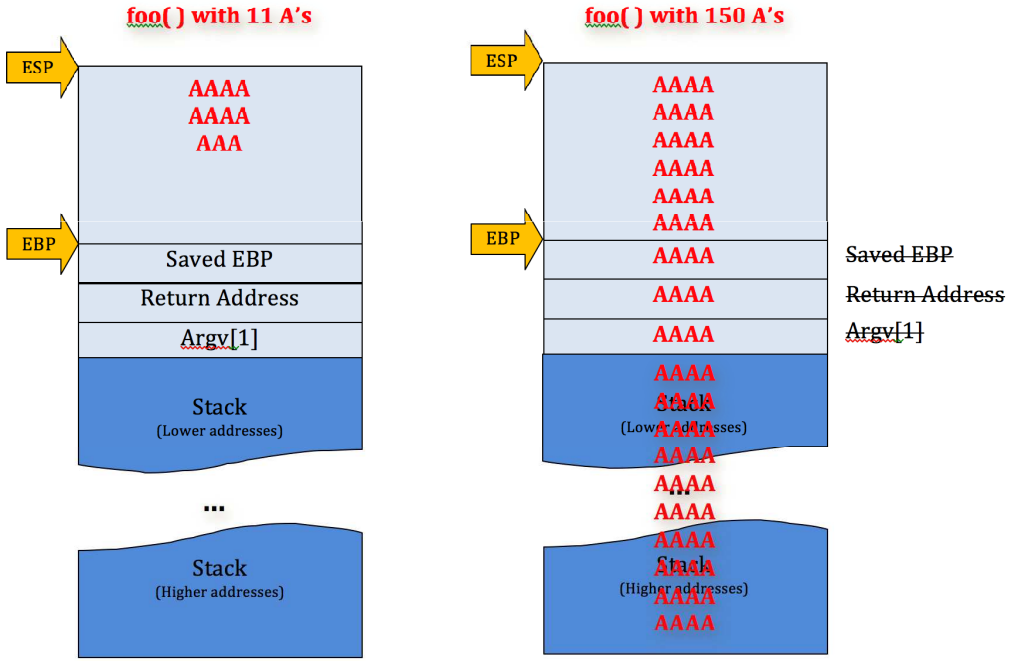
Stack Overflow overview

- Linux uses cdecl as function call convention

```
void foo(int argc, char *argv[]) {
    int variable;
    char buffer[24];

    ...
    strcpy(buffer, argv[1]);
    ...

}
```



Anti-exploitation techniques

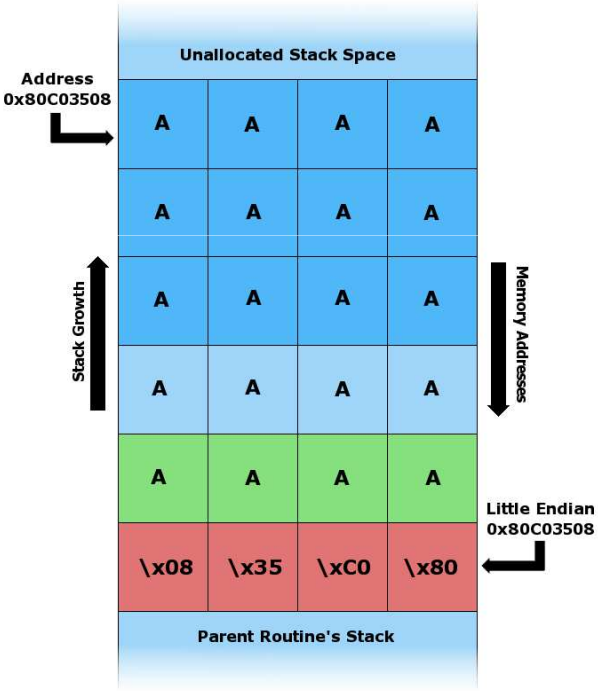
- NX – non-executable stack
- AAAS – ASCII Armored Address Space
- ASLR – Address Space Layout Randomization
- Stack Canary (or stack cookie)

Bypass Anti-Exploitation

Bypassing stack canary

- Process that fork() each connection
- Bruteforce byte-by-byte
- 1020 possible combinations to discover the canary on 32-bits system

- fork() followed by execve() will not work!



Return Oriented Programming

- Chain gadgets to execute malicious code
- A gadget is a suite of instructions which end by the branch instruction ret (Intel) or the equivalent on ARM.

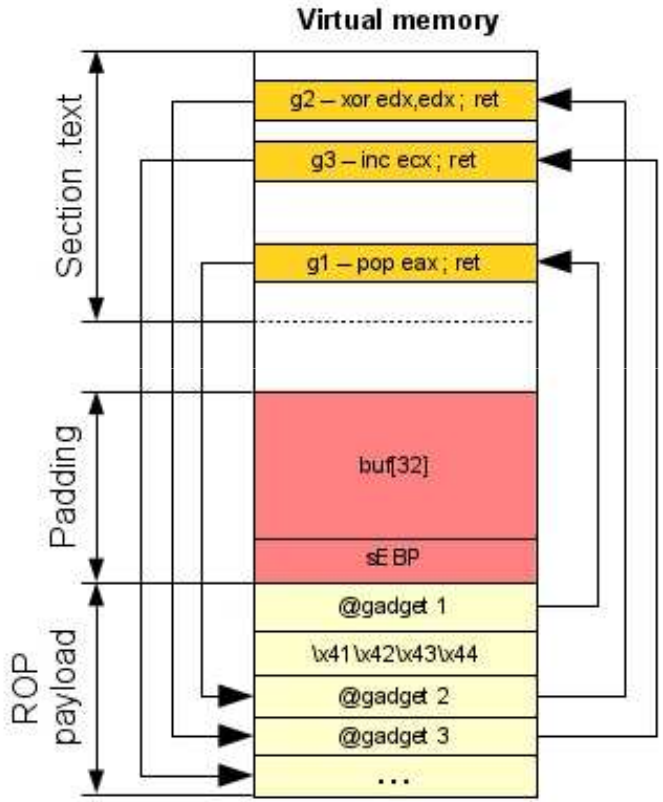
  - Intel examples:
    - pop eax ; ret
    - xor ebx, ebx ; ret
  - ARM examples:
    - pop {r4, pc}
    - str r1, [r0] ; bx lr

- Objective: Use gadgets instead of classical shellcode

Attack using ROP

- Gadget1 is executed and returns
- Gadget2 is executed and returns
- Gadget3 is executed and returns
- And so on until all instructions that you want are executed
- So, the real execution is:

```
pop eax
xor  edx, edx
inc  ecx
```



Blind Return Oriented Programming

- Perform ROP in time of exploitation
- The target server must restart after crash
- Bruteforce byte-by-byte to find write() PLT address
- Perform ret2plt with write() function to leak the server's binary
- Find more gadgets to build our ROP chain to exploit

Blind Remote Exploitation

CVE-2010-4221

- Stack overflow on function pr_netio_telnet_gets() from the file "src/netio.c”
- The server fork() each connection
- The server restore after crash

BROP on CVE-2010-4221

- Bruteforce byte-by-byte write() PLT address
- ret2plt with write(1, 0x080532d8, 0xffffff);
- Leak server's binary
- Search mmap64() and memcpy() PLT entry
- Search gadgets:
  - pop; pop; pop; ret;
  - add esp, 20h; pop; pop; ret;
- Search byte offsets from our shellcode copy routine

Exploiting CVE-2010-4221

- Execute mmap64() to map a read, write and executable memory
- Use memcpy() to copy our shellcode copy routine to the memory mapped
- The shellcode copy routine will copy our shellcode from stack to a executable memory and will pass the flow to the shellcode.

Exploiting CVE-2010-4221

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)
```

ROP chain:

```
mmap() plt address
add esp 20h; pop; pop; ret;
\x00\x00\x00\x10  ----> start address
\x00\x10\x00\x00  ----> length 1000 bytes
\x07\x00\x00\x00  ----> 07 = set read, write and executable permission
\x32\x00\x00\x00  ----> 32 bit flag
\xff\xff\xff\xff  ----> file descriptor (0xffffffff probably will not be in use)
\x00\x00\x00\x00  --|
\x00\x00\x00\x00    |
\x00\x00\x00\x00    |-> offset
\x00\x00\x00\x00    |
\x00\x00\x00\x00    |
\x00\x00\x00\x00  --|
```

Exploiting CVE-2010-4221

```
void *memcpy(void *dest, const void *src, size_t n)
```

First ROP chain:

```
memcpy() plt address
pop; pop; pop; ret;
\x00\x00\x00\x10  ----> destiny
     xxxx         ----> found byte addr (from shellcode copy routine)
\x02\x00\x00\x00  ----> number of bytes to copy
```

Second ROP chain:

```
memcpy() plt address
pop; pop; pop; ret;
\x02\x00\x00\x10  ----> destiny
     yyyy         ----> found byte addr (from shellcode copy routine)
\x01\x00\x00\x00  ----> number of bytes to copy
```

DEMO

Talk is cheap, show me the code!!!

Pros

- We will be able to inject any shellcode that we want with the same technique
- We will be able to exploit any server that runs the vulnerable application, even if it runs on a different distro with different compiling parameters
- We will be able to bypass all principals anti-exploitation methods
- The same technique will works on our another scenarios

Cons

- The server can restart after crash
- In other words: the server must be fork() for each connection
- The server cannot use fork() followed by execve() on creating a child connection
- This type of exploitation may make some noise!
- This attack may take a while to get success

Conclusion

Hacker's vision

- We can build generic ways to exploit applications
- We can attack even with old vulnerabilities
- We can attack a library vulnerability, even if we can't replicate the server's scenario
- We can defeat anti-exploitation protections
- We can attack a application that we don't have the binary

Security really exists?

- Computer vulnerabilities are conceptual
- Always will exist vulnerabilities
- Always will have a way to exploit them

Security is an illusion!!!

What we can do now?

- Security is a continuous process
- Security is: hardware + software + people
- The attacker's vision shows our weakness

We should always be steps ahead of the attackers!!!

References

- Scraps of notes on remote stack overflow exploitation – by pi3
  - http://phrack.org/issues/67/13.html#article
- Uncovering Zero-Days and advanced fuzzing – by Kingcope
  - http://www.exploit-db.com/docs/18924.pdf
- nginx Exploit Documentation About a Generic Way to Exploit Linux Targets – by Kingcope
  - http://www.exploit-db.com/docs/27074.pdf
- An introduction to the Return Oriented Programming and ROP chain generation – by Jonathan Salwan
  - http://shell-storm.org/talks/ROP_course_lecture_jonathan_salwan_2014.pdf
- Hacking Blind – by Andrea Bittau
  - http://www.scs.stanford.edu/brop/bittau-brop.pdf

mindthesec

OBRIGADO!

Ighor Augusto Barreto Candido
ighor@intruder-security.com
https://www.facebook.com/ighorabcandido
www.intruder-security.com

MAIS UM EVENTO: Flipside SECURITY BEYOND TECHNOLOGY