Return Oriented Programming

Anonymous Submission

Abstract. In this paper we introduce the concept of Return Oriented Programming, how to apply it, how to protect against it and show a concrete attack.

Keywords: ROP · Return Oriented Programming · Buffer Overflow · Binary Exploitation

1 Introduction

Return Oriented Programming is a type of buffer overflow attack that has been published in 2007 and ever since has become a widely known buffer overflow technique. It has been developed to circumvent the NX-BIT protection that protects the stack from being executed. At the time of writing this paper modern techniques like Stack Carnaries and ASLR prevent these attacks from being practical but there are millions of running systems using old hard-, firm- and software that is possibly vulnerable to these kinds of buffer overflow attacks. Return Oriented Programming is based on chaining return addresses to code just before a return and therefor allowing almost arbitrary code segments to be chained.

2 Gadgets

Gadgets are code segments that sit before a `ret` instruction, that is an instruction that uses the address on the stack to return to a previous stack frame and therefor a previous level in the call hierarchy. This means we can arbitrarily chain these gadgets and achieve arbitary code execution if we find gadgets for our purpose. A gadget can be found by searching for `0xC3` Bytes in the program. The instructions before then represent the code we can use, for that we need the address of the gadget. We could do this manually or use one of the many tools available, to name a few there is ropper, ROPgadget and pwntools. For this paper i will be using ROPgadget since i found it easy to use and fast. Using the following command under Arch Linux we can dump all gadgets to a file and search in it using regular expressions.

Listing 1: Dumping all gadgets into a file

```
ROPgadget --binary vuln > gadgets
```

3 Target Program

Target Program The following program is the target of our attack, it uses an argument for the buffer overflow, using vulnerable input functions also works, though.

Listing 2: The Target Program

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
```

```
  char buffer[8] = {0};
  if (argc != 2) {
    printf("A single argument is required.\n");
    return 1;
  }
  strcpy(buffer, argv[1]);
  return 0;
}
```

Compilation We use the following command to compile the target program

Listing 3: The compliation command

```
clang -o vuln vuln.c -m32 -fno-stack-protector -Wl,-z,relro\
 ,-z,now,-z -static
```

4 Attack

5 Results

6 Protection

7 Discussion