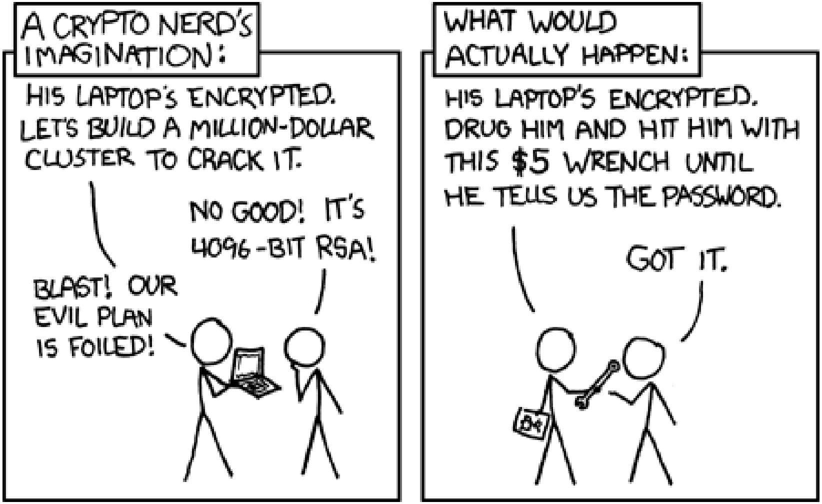
Cybersecurity

Binary Exploitation and Reverse Engineering

Naavin Ravinthran

Monash (Malaysian Campus) SIG


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Who am I?

Who am I?

Who am I?

- Year 2 Student

Who am I?

- Year 2 Student
- I like Security stuff. I've made some challenges before.

Who am I?

- Year 2 Student
- I like Security stuff. I've made some challenges before.
- Email: nrav0005@student.monash.edu

Who am I?

- Year 2 Student
- I like Security stuff. I've made some challenges before.
- Email: nrav0005@student.monash.edu
- Discord: Yes#0952 (join the SOIT Discord server! #cybersecurity channel)

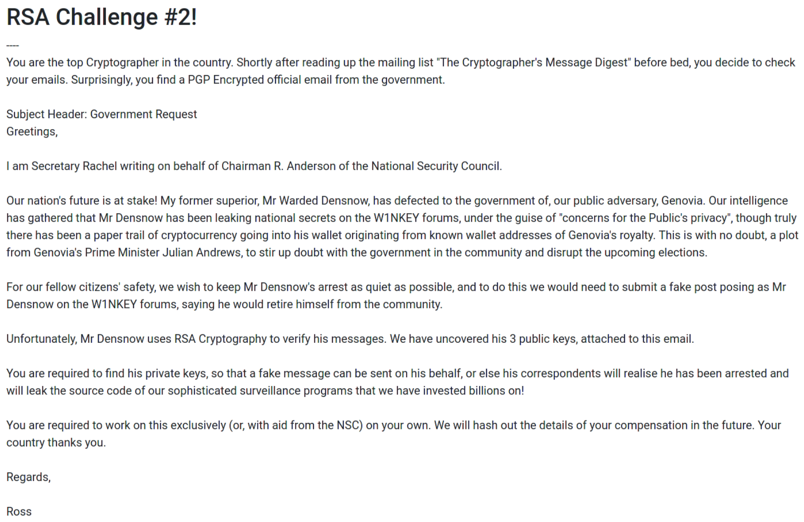
MARIE Christmas

But your work is not done yet! You get up slowly, up from your knees, and start running to your quarters. You feel the wind brushing against your face, you hear the insects of dusk, the smell of the dew from the grass, you can see from your peripherals the orange light emerging as the sun rises. You smile, knowing that you now have a purpose in life. It's all in the palm of your hands.

You unclench your fist to reveal the flash drive from earlier in the palm of your hands. You cautiously plug the flash drive into your laptop, and inside you find this python script. Your job is to figure out the password.

```
#!/usr/bin/env python3
# You could use timeit, but I used time
import time
import sys
import random
import re

class CPU:
    def __init__(self):
        self.ac = 0
        self.pc = 0
        self.halted = False
        self.xor_counter = 0
        self.yes = False
        self.start_time = time.time()

        self.memory = [0 for _ in str(0xFFF)]

        self.func_table = [self.__jns, self.__load, self.__store, self.__add, self.__subt, self.__input, self.__output,
                           self.__halt, self.__skipcond, self.__jump, self.__clear, self.__addi, self.__jumpi,
                           self.__loadi, self.__storei, self.__nop]

    def __get_arg(self):
        return self.memory[self.pc] & 0b0000111111111111

    def __fetch(self):
        return self.memory[self.pc]
```

RSA Challenge



RSA Challenge #2!

You are the top Cryptographer in the country. Shortly after reading up the mailing list "The Cryptographer's Message Digest" before bed, you decide to check your emails. Surprisingly, you find a PGP Encrypted official email from the government.

Subject Header: Government Request
Greetings,

I am Secretary Rachel writing on behalf of Chairman R. Anderson of the National Security Council.

Our nation's future is at stake! My former superior, Mr Warded Densnow, has defected to the government of, our public adversary, Genovia. Our intelligence has gathered that Mr Densnow has been leaking national secrets on the W1NKEY forums, under the guise of "concerns for the Public's privacy", though truly there has been a paper trail of cryptocurrency going into his wallet originating from known wallet addresses of Genovia's royalty. This is with no doubt, a plot from Genovia's Prime Minister Julian Andrews, to stir up doubt with the government in the community and disrupt the upcoming elections.

For our fellow citizens' safety, we wish to keep Mr Densnow's arrest as quiet as possible, and to do this we would need to submit a fake post posing as Mr Densnow on the W1NKEY forums, saying he would retire himself from the community.

Unfortunately, Mr Densnow uses RSA Cryptography to verify his messages. We have uncovered his 3 public keys, attached to this email.

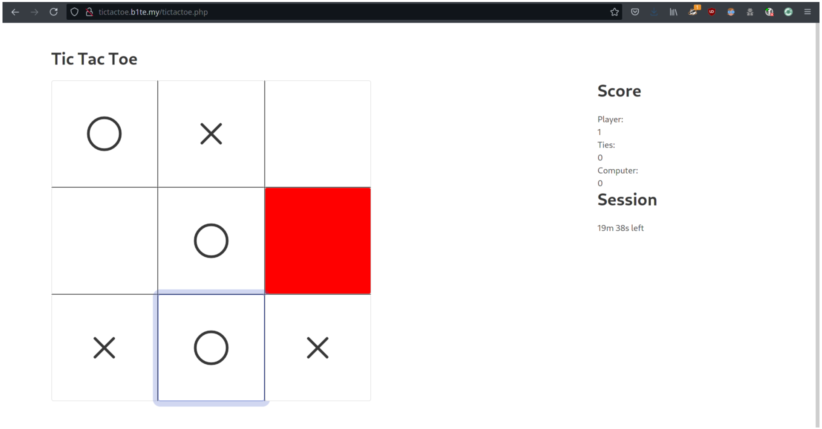
You are required to find his private keys, so that a fake message can be sent on his behalf, or else his correspondents will realise he has been arrested and will leak the source code of our sophisticated surveillance programs that we have invested billions on!

You are required to work on this exclusively (or, with aid from the NSC) on your own. We will hash out the details of your compensation in the future. Your country thanks you.

Regards,

Ross

Tic Tac Toe Challenge



tictactoe.b1te.my/tictactoe.php

Tic Tac Toe

Score

Player:
1
Ties:
0
Computer:
0

Session

19m 38s left

What is a binary?

- Isn't it base-2? 1000101_2?

What is a binary?

- Isn't it base-2? 1000101_2?
- Short for binary executable. A program you run.

What is a binary?

- Isn't it base-2? 1000101_2?
- Short for binary executable. A program you run.
- It is in native machine code, e.g: compiled from C or C++ or Rust, *not* from interpreted languages like Python/Ruby/Java, although the interpreter itself is a binary that can be exploited.

What is a binary?

- Isn't it base-2? 1000101_2?
- Short for binary executable. A program you run.
- It is in native machine code, e.g: compiled from C or C++ or Rust, *not* from interpreted languages like Python/Ruby/Java, although the interpreter itself is a binary that can be exploited.
- The main three are the following:-
    - Executable and Linkable Format (ELF) files, standard binary file format for Unix and Unix-Like systems.
    - Portable Executable (PE), ".exe" extension, used on Windows systems.
    - Mach-O - Used by systems based on the Mach kernel, i.e: iOS, macOS.

What is a binary?

- Isn't it base-2? 1000101_2?
- Short for binary executable. A program you run.
- It is in native machine code, e.g: compiled from C or C++ or Rust, *not* from interpreted languages like Python/Ruby/Java, although the interpreter itself is a binary that can be exploited.
- The main three are the following:-
  - Executable and Linkable Format (ELF) files, standard binary file format for Unix and Unix-Like systems.
  - Portable Executable (PE), ".exe" extension, used on Windows systems.
  - Mach-O - Used by systems based on the Mach kernel, i.e: iOS, macOS.

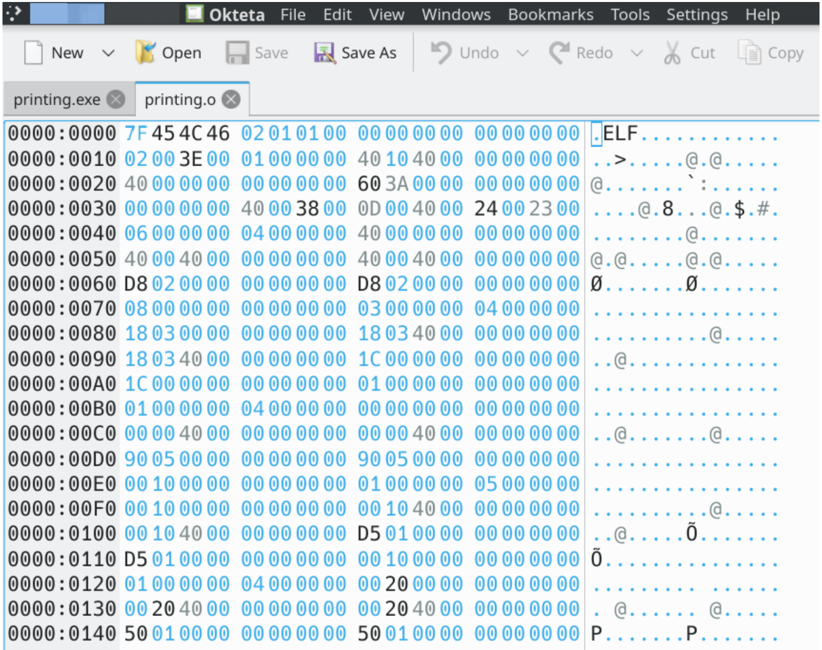
Note: You can also apply the exploitation to shared library files like DLLs or .so files, if a binary uses those.

What is a binary?

- Isn't it base-2? 1000101_2?
- Short for binary executable. A program you run.
- It is in native machine code, e.g: compiled from C or C++ or Rust, *not* from interpreted languages like Python/Ruby/Java, although the interpreter itself is a binary that can be exploited.
- The main three are the following:-
  - Executable and Linkable Format (ELF) files, standard binary file format for Unix and Unix-Like systems.
  - Portable Executable (PE), ".exe" extension, used on Windows systems.
  - Mach-O - Used by systems based on the Mach kernel, i.e: iOS, macOS.

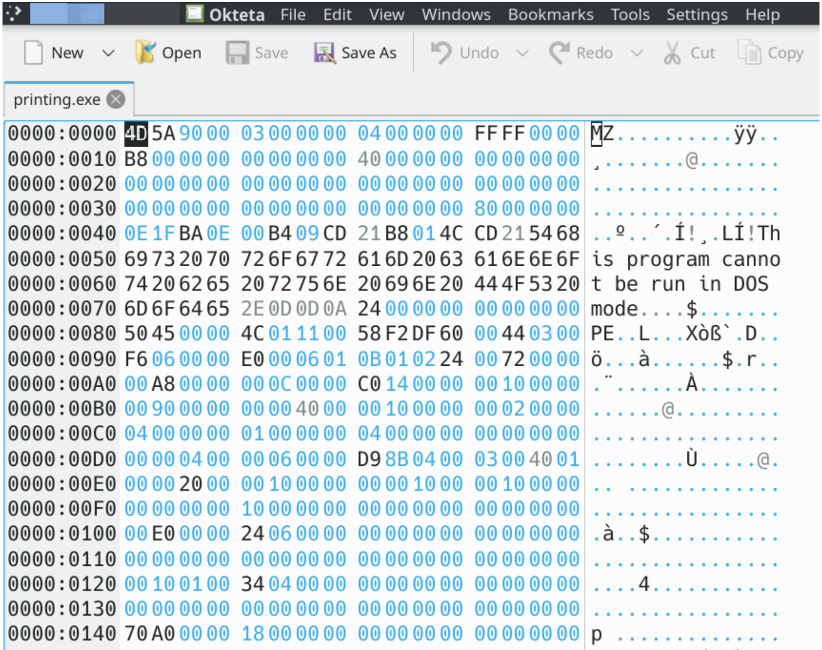
Note: You can also apply the exploitation to shared library files like DLLs or .so files, if a binary uses those.

ELF file



```
0000:0000 7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00 .ELF............
0000:0010 02 00 3E 00 01 00 00 00 40 10 40 00 00 00 00 00 ..>.....@.@.....
0000:0020 40 00 00 00 00 00 00 00 60 3A 00 00 00 00 00 00 @.......`:......
0000:0030 00 00 00 00 40 00 38 00 0D 00 40 00 24 00 23 00 ....@.8...@.$.#.
0000:0040 06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00 ........@.......
0000:0050 40 00 40 00 00 00 00 00 40 00 40 00 00 00 00 00 @.@.....@.@.....
0000:0060 D8 02 00 00 00 00 00 00 D8 02 00 00 00 00 00 00 Ø.......Ø.......
0000:0070 08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00 ................
0000:0080 18 03 00 00 00 00 00 00 18 03 40 00 00 00 00 00 ..........@.....
0000:0090 18 03 40 00 00 00 00 00 1C 00 00 00 00 00 00 00 ..@.............
0000:00A0 1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 ................
0000:00B0 01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00C0 00 00 40 00 00 00 00 00 00 00 40 00 00 00 00 00 ..@.......@.....
0000:00D0 90 05 00 00 00 00 00 00 90 05 00 00 00 00 00 00 ................
0000:00E0 00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00 ................
0000:00F0 00 10 00 00 00 00 00 00 00 10 40 00 00 00 00 00 ..........@.....
0000:0100 00 10 40 00 00 00 00 00 D5 01 00 00 00 00 00 00 ..@.....Õ.......
0000:0110 D5 01 00 00 00 00 00 00 00 10 00 00 00 00 00 00 Õ...............
0000:0120 01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00 ......... ......
0000:0130 00 20 40 00 00 00 00 00 00 20 40 00 00 00 00 00 . @...... @.....
0000:0140 50 01 00 00 00 00 00 00 50 01 00 00 00 00 00 00 P.......P.......
```

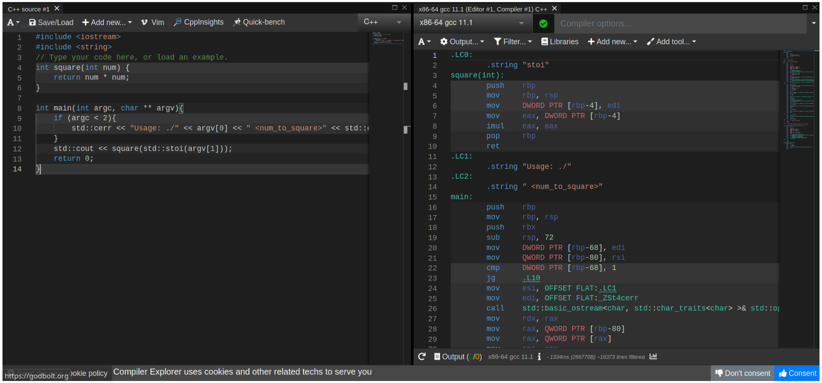
EXE file



```
0000:0000 4D 5A 90 00  03 00 00 00  04 00 00 00  FF FF 00 00  MZ..........ÿÿ..
0000:0010 B8 00 00 00  00 00 00 00  40 00 00 00  00 00 00 00  ¸.......@.......
0000:0020 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0030 00 00 00 00  00 00 00 00  00 00 00 00  80 00 00 00  ................
0000:0040 0E 1F BA 0E  00 B4 09 CD  21 B8 01 4C  CD 21 54 68  ..º..´.Í!¸.LÍ!Th
0000:0050 69 73 20 70  72 6F 67 72  61 6D 20 63  61 6E 6E 6F  is program canno
0000:0060 74 20 62 65  20 72 75 6E  20 69 6E 20  44 4F 53 20  t be run in DOS 
0000:0070 6D 6F 64 65  2E 0D 0D 0A  24 00 00 00  00 00 00 00  mode....$.......
0000:0080 50 45 00 00  4C 01 11 00  58 F2 DF 60  00 44 03 00  PE..L...Xòß`.D..
0000:0090 F6 06 00 00  E0 00 06 01  0B 01 02 24  00 72 00 00  ö...à......$.r..
0000:00A0 00 A8 00 00  00 0C 00 00  C0 14 00 00  00 10 00 00  .¨......À.......
0000:00B0 00 90 00 00  00 00 40 00  00 10 00 00  00 02 00 00  ......@.........
0000:00C0 04 00 00 00  01 00 00 00  04 00 00 00  00 00 00 00  ................
0000:00D0 00 00 04 00  00 06 00 00  D9 8B 04 00  03 00 40 01  ........Ù.....@.
0000:00E0 00 00 20 00  00 10 00 00  00 00 10 00  00 10 00 00  .. .............
0000:00F0 00 00 00 00  10 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0100 00 E0 00 00  24 06 00 00  00 00 00 00  00 00 00 00  .à..$...........
0000:0110 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0120 00 10 01 00  34 04 00 00  00 00 00 00  00 00 00 00  ....4...........
0000:0130 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0140 70 A0 00 00  18 00 00 00  00 00 00 00  00 00 00 00  p ..............
```

Assembly Language

The scary boogeyman that *Real* programmers program in.



```
#include <iostream>
#include <string>
// Type your code here, or load an example.
int square(int num) {
    return num * num;
}

int main(int argc, char ** argv){
    if (argc < 2){
        std::cerr << "Usage: ./" << argv[0] << " <num_to_square>" << std::e
    }
    std::cout << square(std::stoi(argv[1]));
    return 0;
}
```

```
.LC0:
        .string "stoi"
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
.LC1:
        .string "Usage: ./"
.LC2:
        .string " <num_to_square>"
main:
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 72
        mov     DWORD PTR [rbp-68], edi
        mov     QWORD PTR [rbp-80], rsi
        cmp     DWORD PTR [rbp-68], 1
        jg      .L10
        mov     esi, OFFSET FLAT:.LC1
        mov     edi, OFFSET FLAT:_ZSt4cerr
        call    std::basic_ostream<char, std::char_traits<char> >& std::op
        mov     rdx, rax
        mov     rax, QWORD PTR [rbp-80]
        mov     rax, QWORD PTR [rax]
```

Output (0/0) x86-64 gcc 11.1 - 1334ms (266770B) ~16373 lines filtered

Compiler Explorer uses cookies and other related techs to serve you

Assembly Language

- Get used to it...

Assembly Language

- Get used to it...
- Get a reference for the instructions.

Assembly Language

- Get used to it...
- Get a reference for the instructions.
- Godbolt is helpful.

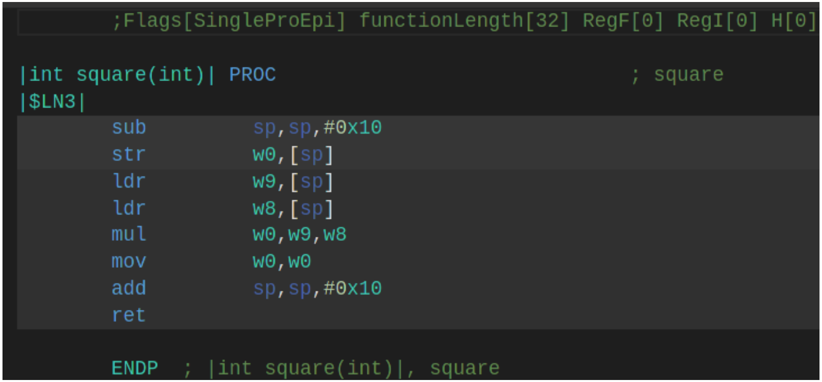
Assembly Language

- Get used to it...
- Get a reference for the instructions.
- Godbolt is helpful.
- disassemble programs.

Assembly Language

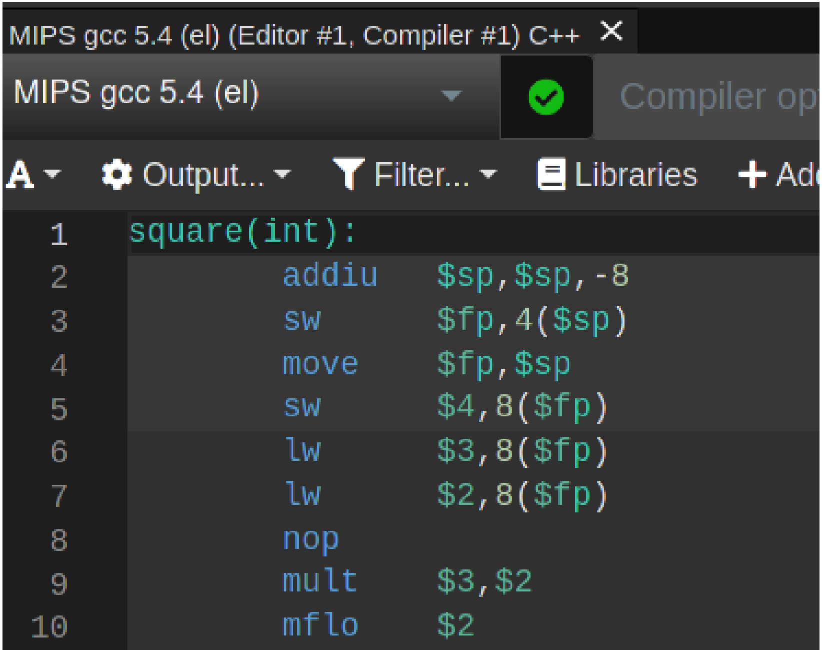
- Get used to it...
- Get a reference for the instructions.
- Godbolt is helpful.
- disassemble programs.
- You can try write your own assembly programs (this will also become helpful if you ever want to write your own shellcode)

There are other assembly languages too...



```
        ;Flags[SingleProEpi] functionLength[32] RegF[0] RegI[0] H[0]

|int square(int)| PROC                              ; square
|$LN3|
        sub         sp,sp,#0x10
        str         w0,[sp]
        ldr         w9,[sp]
        ldr         w8,[sp]
        mul         w0,w9,w8
        mov         w0,w0
        add         sp,sp,#0x10
        ret

        ENDP  ; |int square(int)|, square
```

There are other assembly languages too...



```
square(int):
        addiu   $sp,$sp,-8
        sw      $fp,4($sp)
        move    $fp,$sp
        sw      $4,8($fp)
        lw      $3,8($fp)
        lw      $2,8($fp)
        nop
        mult    $3,$2
        mflo    $2
```

Useful Tools

Main tools I will be talking about.

- radare2/rizin (Open-Source TUI RE debugger) (dynamic)
- gdb/lldb (terminal debuggers) (dynamic)
- Ghidra (Open-Source RE tool by the NSA) (mostly static)

Useful Tools

Main tools I will be talking about.

- radare2/rizin (Open-Source TUI RE debugger) (dynamic)
- gdb/lldb (terminal debuggers) (dynamic)
- Ghidra (Open-Source RE tool by the NSA) (mostly static)

Other tools you may be interested in

- pwntools (python framework for helping in exploitation)
- IDA Pro (Paid RE tool similar to Ghidra)
- Cutter (Open-Source GUI frontend for rizin)
- x64Dbg/WinDbg/OllyDbg (various GUI Debuggers in Windows)
- Binary Ninja
- ... many more! See this list.

A few examples

- I'll go through the very basic use of gdb and rizin, and just show how Ghidra looks like.
- I'll mostly be talking about a buffer overflow to smash the stack.
- gdb manual
- radare2 manual Note: Examples I give were compiled specifically to use as examples (with debugging information, no compiler optimisations, etc.). Real-life situations may vary. Note 2: Although sometimes developers forget. Super Mario 64's North America cartridge was accidentally release without optimisations, making it easier for reverse engineers to understand the code. Apple forgot to strip debug symbols once for iOS.

Try it out yourself

Go ahead, good luck! :) This is just the tip of the iceberg.

Try it out yourself

Go ahead, good luck! :) This is just the tip of the iceberg.

- gdb, radare2 and ghidra all have a lot more features
- For example, gdb and radare2 have watchpoints, to break whenver a certain area of memory is accessed or written to.
- radare2 has built-in binary patching to rewrite some assembly instructions and re-write them to the binary (breaking the digital signature if there is any though.)

Further exploration

- Use of Shellcode (not unreasonable for IoT devices. ROP can be used to mmap/VirtualAlloc executable memory.)

Further exploration

- Use of Shellcode (not unreasonable for IoT devices. ROP can be used to mmap/VirtualAlloc executable memory.)
- Return-Oriented-Programming

Further exploration

- Use of Shellcode (not unreasonable for IoT devices. ROP can be used to mmap/VirtualAlloc executable memory.)
- Return-Oriented-Programming
- String format exploits ("%<specifier>" in printf)

Further exploration

- Use of Shellcode (not unreasonable for IoT devices. ROP can be used to mmap/VirtualAlloc executable memory.)
- Return-Oriented-Programming
- String format exploits ("%<specifier>" in printf)
- Fuzzing (automated tools to try common exploits)
- Heap vulnerabilities
- DLL Injection / LD_PRELOAD Hooking (Liveoverflow has videos on this)
- Liveoverflow Youtube series
- Search for "Crackmes" online, or look for categories on "Binary exploitation" in CTFs. Be careful not to infect your system!
- Exploit Education's "Phoenix" challenges are fun.

Further exploration

- Use of Shellcode (not unreasonable for IoT devices. ROP can be used to mmap/VirtualAlloc executable memory.)
- Return-Oriented-Programming
- String format exploits ("%<specifier>" in printf)
- Fuzzing (automated tools to try common exploits)
- Heap vulnerabilities
- DLL Injection / LD_PRELOAD Hooking (Liveoverflow has videos on this)
- Liveoverflow Youtube series
- Search for "Crackmes" online, or look for categories on "Binary exploitation" in CTFs. Be careful not to infect your system!
- Exploit Education's "Phoenix" challenges are fun.
- I've been told some of you may just be interested in Certificates. Offensive Security is good. Comptia Security+ is more for penetration testers. Security Plus All are expensive, try see if you need it for the job or see if you can get them to sponsor you.

Other Sources

- gdb manual
- rizin manual
- radare2 manual
- intel x86 reference manual
- (PAID) Practical Binary Analysis - Dennis Andriesse
- (PAID) Practical Malware Analysis - Michael Sikorski and Andrew Honig