Week 04
Reverse Engineering II

Nathan

Announcements

-

sigpwny{work_smart_not_hard}

Table of Contents

- Tools/Techniques
  - Symbolic Execution
  - Instruction Counting Side Channels

- Obfuscation
  - Self modifying code
  - VM obfuscation

Symbolic Execution

- Solve for inputs to program that achieve a desired output
  1. Interpreter steps through program
  2. Generates expressions for symbolic variables
  3. Solves inputs for a given a set of constraints

- x = ?
- y = x ** 2
- z = y + 1
- Solve for x such that z == 5, and x > 0

Input

Constraints

Symbolic Execution Usages

- Reversing without reversing
    - Solve for input on stdin (flag) such that the flag checker prints “That flag is correct!”
- Automated PWN
    - Solve for input such that the instruction pointer is overwritten
- Automating ROP gadget discovery
    - Find a gadget such that register a = register b after execution

Introducing Angr

- Angr can be used for automating CTF chals

```
1. import angr
2. simgr = angr.Project('./brute').factory.simgr()
3. simgr.explore(find=lambda s: b'Correct' in s.posix.dumps(1))
4. print(simgr.found[0].posix.dumps(0))
```

1. Import angr (install w/ pip3 install angr)
2. Create a simulation manager with the “brute” binary
3. Explore all paths such that “Correct” is in stdout
4. Print the first stdin input which yielded “Correct” on stdout

https://gist.github.com/nathanfarlow/7befd30ee4de5bceaa7ca329b21ef43f

Instruction Counting

- Given a flag as input, count how many instructions are executed
  - More instructions executed => flag is closer to being correct
  - Depends that the program terminates early if flag character is incorrect
  - Depends on order that flag is traversed

```
// strequals checks each character in order
// and stops immediately if characters differ
if (!strequals(user_input, true_flag)) {
    puts("Correct!");
} else {
    puts("Wrong flag");
}
```

Instruction Counting

- Can use Intel’s Pin
  - https://github.com/ChrisTheCoolHut/PinCTF

- Can use valgrind’s exp-bbv or callgrind tool
  - valgrind --tool=exp-bbv ./a.out sigpwny{...}

- aaaaaaa => 148862 instructions
- sigpwny => 148962 instructions

Self Modifying Code

- Typically code is only readable and executable
- You can mmap or mprotect a region to make readable, writable, and executable memory
- Code in this region can modify itself as it runs (see signals)
- Often RE’d using a debugger

```
mprotect(addr, true_size, PROT_READ | PROT_WRITE | PROT_EXEC)
```

VM Obfuscation

- Actual program is a virtual machine executing other program instructions
  - Reasoning: lots of good tools exist for reversing x86, but if I invent my own custom architecture and write an emulator for it, then people can't reverse it easily
  - VMProtect, ropfuscated, hell

- Ways to reverse include staring at emulator to understand mode of instruction execution, then writing tools (disassemblers, decompilers), crying

Go try for yourself!

https://ctf.sigpwny.com

- Start with Reverse Engineering I if not completed, then move on to Reverse Engineering II
- See slides for angr auto solver script
- Practice practice practice! Ask for help!

Next Meetings

Next Thursday: Pwn I

- Go over pwn fundamentals
- How to exploit programs with vulnerabilities

Sunday Seminar: Pwn II (maybe?)

- More practice with pwn
- Other pwn techniques