CSE4830 Cyber Grand(er) Challenge Notes

CSE4830 : Binary Reverse Engineering

Overview

In class, we have reviewed automating the analysis of binaries using static, dynamic, and symbolic/concolic execution. Throughout the course, we have built small programs/scripts to perform this analysis. Further, we have examined methods for exploiting binaries using ROP Chains and Format String Vulnerabilities. For the final exam, you will write an automatic exploit generation engine that attempts to *automatically exploit a binary* (e.g., does not require any human analysis.)

Scope

The scope of the exploitation required to win is limited exploits we have covered in class so far, including:

1. Constructing ROP Gadgets to return to PLT
2. Constructing ROP Gadgets to populate parameters & call Execve
3. Constructing ROP Gadgets to populate parameters & call Syscall
4. Constructing OneGadget ROP Chain
5. Constructing ROP Gadget to Pivot the Stack
6. Constructing ROP Gadget to Perform Write Primitive
7. Using Format String Vulnerabilities to leak a variable from the stack
8. Using Format String Vulnerabilities to overwrite a variable
9. Using Format String Vulnerabilities to overwrite the GOT entry for a function
10. Using Format String Vulnerabilities to leak the address of Libc

Constraints

To focus efforts on the automatic exploitation aspect and not semantics, the following constraints will apply to the challenge binaries:

- The name of the binary will always be **vuln**
- The goal of every challenge is display the contents of **flag.txt**
- Vulnerable binaries will request input by printing: **"input >>> "**
- Vulnerabilities binaries will display by printing: "<<< **output: "**
- If present, **win()** will always be the name of a function that displays the contents of **flag.txt**
- If present, **pwnme** will always be the name of a variable that when set to 1337 results in displaying the contents of **flag.txt**
- If present, **flag** will always be the name of a variable that holds the contents of **flag.txt**
- Binaries will always use the provided libc.so.6

Sample Binaries

The instructor will provide sample binaries for testing. These are not representative of the full scope of challenges.

Competition & Grading

The final consists of teams of 3-4 teammates. Each team will submit their program to be run against a set of instructor-provided binaries. Each team will receive points as follows:

- 1.0 points if execution results in display contents of flag.txt
- 0.75 points if execution results in partial display of flag.txt
- 0.25 points if team can automatically spot to address of code that introduces vulnerability, but fails to exploit it.

Each team should turn in their program and a README, detailing the individual efforts of each team member. Team members should contribute equally and understand how the teams' solution works. The instructor reserves the right to ask individual team members about their contribution and understanding to compute the final individual grades.

(+25) points will be awarded to the final exam grades of the team that wins the competition.