Proposal: Automated ROP Exploit (Project 4)

Liam Dalgarno (ld17285), Jack Bond-Preston (jb17662), Chris Gora (kg17815)
`Team 0xCA7CAFE`

1 Problem

The method of ROP exploitation we used in the lab was highly manual: you had to manually find the input length to overwrite the RET and manually create the payload to get a chosen `execve` command line target to execute. `execve`-based shellcodes were the only ones used, however, there exists many other potential shellcodes.

2 Solution

We aim to automate steps that we manually conducted during the lab in 32-bit executables. This will be achieved using Python scripts that extend the ROPGadget code. We will aim to keep the ROPGadget source code largely unmodified if possible - mostly relying on importing it and using existing code.

Step 1: In order to find the correct length of the padding we plan to first make the program crash so that we can generate a core dump. We may have to generate inputs of increasing lengths to try to force the program to crash. We will use the `pwntools` Python library to do this and to extract the offset of the `eip` from this core dump [1].

Step 2: We will use `ROPgadget` to generate a ROP chain with this offset. We will expand our tool to be able to handle arbitrary arguments to `execve`.

Step 3: To make sure that the exploit works for any given `.data` address, we will append `0x00` to the end of each argument to `execve` (using gadgets, e.g. to clear and write registers to the `.data` segment), since we cannot assume that the segment is empty.

Step 4: We will implement the ability to generate an exploit by converting arbitrary shellcode to a valid ROP input.

3 Relevant Literature

To inject arbitrary shellcode in step 4, we will likely have to use a technique similar to the one detailed in Section 3.2.1 of [2]. This involves breaking down complex instructions in the shellcode into simpler, more atomic instructions. It is then easier to find ROP gadgets which implement these instructions. This is due to simpler instructions being more common, and smaller atomic instructions using fewer registers and thus being more resistant to register side-effects in gadgets.

4 Collaboration

The tasks for this project are not easily parallelisable. Therefore, to distribute the work, we plan to work together in a Discord call and use the Visual Studio Live Share collaborative development tool. This way, each member is fully engaged with the whole project throughout the next three weeks.

5 Feedback from Lecturers

1. Note that point 3 is not about appending a null byte, but rather taking care of not having an address that has `\00` (e.g. `0x8044ba00`). This will fail as strcpy will stop coping data beyond that null byte.

2. For your final report, please, do not forget to read the relevant literature. Most projects point to at least one paper, but you are expected to have read more for your final report. If you have any concerns about how to identify relevant papers, please, do get in touch (e.g. on Discord).

3. For collaboration, try to think about ways to have tasks. for example, one can finding relevant papers, arbitrary shellcodes, etc. This will help you to parallelize tasks. Step 4 is not that easy!

4. Very important: think about your validation/evaluation part– what arbitrary command to execve you want to take to showcase your tool and what arbitrary shellcode you want to take to show its conversion capability?

5. In any case, contact us if you want to discuss your project.

References

[1] "pwnlib.elf.corefile - Core Files - pwntools 4.3.0 documentation." `http://docs.pwntools.com/en/latest/elf/corefile.html`.

[2] C. Ntantogian, G. Poulios, G. Karopoulos, and C. Xenakis, "Transforming malicious code to rop gadgets for antivirus evasion," *IET Information Security*, vol. 13, no. 6, pp. 570–578, 2019.