Structured Empirical Evaluation of ROP-Chain Generators

Background

Arbitrary code execution exploits are among the most severe types of cyberattacks, since they allow an attacker to implant and execute malicious code into a running process, sometimes allowing a complete takeover of the affected system. *Return-oriented programming* (ROP) is the de-facto standard technique used by attackers today when exploiting memory errors in native-code programs, allowing the attackers to bypass code-injection protections by chaining together snippets of the vulnerable program's own code. Traditionally, these so-called *ROP-chains* were crafted manually, which is a highly laborious and error prone task. Today, several tools that aim to automate the process exist, lowering the bar both for attackers and white-hat security researchers when crafting ROP-chains. However, due to the high complexity of the problem, all existing tools have various limitations compared to a human attacker.

Aim and purpose

This project is a follow-up on a project from last year[1] where we empirically investigated the speed and effectiveness of a number of open-source ROP-chain generators. Previous years' groups compared the tools' ability to craft chains for a common attack (launching a process using the *execve* system call). However, since different tools have somewhat different notions of how to implement this attack, the comparison wasn't entirely fair. In this project we will take a more rigorous approach, where we define the exact semantics of the chain to be generated ourselves, so that each tool is solving the exact same problem. We will also focus on a subset of the tools considered in the previous project. (Specifically, the tools *angrop*, *ROPium* and *SGC*, which turned out to be the most effective and mature ones, plus the tool *GadgetPlanner*, which wasn't thoroughly investigated due to time constraints.)

Your task will be to set up the tools, and learn how to configure them to generate a chain according to a specification that will be provided. This step will require you to learn about the tools by reading documentation and/or source code. Configuring the chain generation might also entail writing some custom code/scripts. Once this step is completed, you will test the tools on a set of binaries that will also be provided by the supervisor.

Deliverables

In addition to the written report, you are expected to provide any code you have written, including some documentation, so that the experiments can be repeated.

Prerequisites

You are expected to have some experience with building/configuring open-source projects in Linux. You are also expected to be able to automate experiment runs on Linux (Bash scripting, etc.) Prior basic knowledge of binary exploits and ROP (e.g., from TDDC90) is a merit, but not strictly necessary. A basic course on low-level or system-level programming (e.g., TDDB68/TDDE68) is highly recommended, however. Courses on software verification (e.g., TDDE34), and compilers (TDDB44/TDDE66) can be helpful for understanding how ROP-chain generators work, but are not a strict requirement.

---

[1] Reports can be found here:
https://www.ida.liu.se/~TDDE63/oldprojects/2024/TDDE63_rop_report.pdf
https://www.ida.liu.se/~TDDE63/oldprojects/2024/TDDE63_ROP.pdf