Fakultät für Ingenieurwissenschaften

Exposé zur Master-Thesis

Automating ROP and JOP Chain Generation aka ROP Compiler

eingereicht von:	David Schunke
	Studiengang IT-Sicherheit und Forensik
Matrikelnummer:	417785

Inhaltsverzeichnis

1 Problemstellung und Motivation

Return-oriented-programming (ROP) und *Jump-oriented-programming (JOP)* sind Angriffstechniken, welche genutzt werden, um Schadcode trotz vorhandener Sicherheitsmechanismen ausführen zu können. Dies erfordert im Zielprogramm eine Verwundbarkeit im Bereich Memory Corruption (z.B. Buffer Overflow, Use-After-Free (UAF), o.ä.). Hierzu werden die im Zielprogramm vorhandenen Assembler-Instruktionen, welche sich bereits im Programmspeicher befinden, genutzt, um sogenannte *ROP Gadgets* zu bilden.
Ein Gadget ist eine kleine Code Sequenz, welche eine bestimmte Aufgabe durchführt, z.B. das Kopieren eines Speicherregisters in ein Anderes oder die Addition zweier Register. ROP oder JOP Gadgets enden typischerweise mit einer `return`, `jump` oder `call` Anweisung, welche die Ausführung zu einer vorherig gespeicherten Stelle zurückbringen soll. Die Manipulation dieser Return-Addresse erlaubt es dem Angreifer Gadgets zu verketten, um eigene Operationen durchzuführen und ein gewünschtes Ergebnis zu erzielen (z.B. das Ausführen von eigenem Code). Da lediglich Maschineninstruktionen genutzt werden, welche sich bereits im Speicher des Programms befinden, können viele Sicherheitsmechanismen, wie z.B. ASLR [1], NX / XD [2], W$\oplus$X [3] oder DEP [4], umgangen werden.

Die Entwicklung eines ROP-basierten Exploit erfordert es bisher manuell ROP Gadgets zu verketten - d.h. Chains zu bilden - sowie den passenden Program Input zu finden, um über eine Memory Corruption die ROP Chain zu starten. Diese Prozesse durchzuführen ist nicht nur aus Angreifersicht von Interesse, sondern auch aus Sicht von Sicherheitsforschern wichtig, um mögliche Exploits finden und anschließend beheben bzw. melden zu können.

Die manuelle Suche einer ROP Chain ist zeitintensiv und erfordert tiefgreifende

[1] ASLR (address space layout randomization) randomisiert die Speicherpositionen, sodass Adressbereiche nicht mehr vorhersehbar sind.

[2] NX-Bit (no-execute, AMD) bzw. XD (execute disable, Intel) aktivierte Prozessoren markieren Teile des Speichers als nicht-ausführbar.

[3] W$\oplus$X (write xor execute) sichert Prozess- oder Kernel-Speicherbereiche dadurch ab entweder schreib- oder ausführbar zu sein, niemals beides gleichzeitig.

[4] Data Execution Prevention (DEP) ist das Microsoft Windows Equivalent zu NX.

Kenntnisse der zugrundeliegenden Architektur, des Zielprogramms sowie der Funktionsweise von Maschinenprozessen und Speicherverwaltung. An die durchführende Person werden somit hohe Anforderungen gestellt, welche das eigentliche Auffinden von Schwachstellen erschwert.

Durch die Erstellung eines *ROP Compilers* könnten diese Prozesse schrittweise automatisiert werden. Dies würde den zeitlichen Aufwand verringern sowie die Möglichkeit des Nachweises einer Schwachstelle mittels eines Exploits bzw. PoC an eine breitere Masse von Personen geben, um so Schwachstellen schneller finden, beheben als auch im Bereich der Strafverfolgung sowie Nachrichtendienste nutzen zu können.

2 Ziele und Grenzen

2.1 Zielsetzung

Im Rahmen dieser Arbeit soll ein ROP Compiler entwickelt werden, welcher als Eingaben auf zur Verfügung stehender Gadgets sowie vordefinierten Zielcode (Payload) basierend eine Verkettung der Gadgets automatisiert durchführen soll. Der Compiler übersetzt anschließend den Payload mittels der verfügbaren Gadgets zu einer ROP Chain, welche zum zu übersetzenden Code semantisch gleich ist. Der zur Verkettung benötigte Algorithmus sowie die zur Eingabe benötigten Formatvorgaben sollen ebenfalls innerhalb dieser Arbeit definiert und implementiert werden.

Ziel ist die Erstellung eines Proof-of-concept (PoC), welcher auf Basis einer zuvor ausgewählten spezifischen Prozessorarchitektur (z.B. x64, ARM32 oder ARM64) die grundsätzliche Umsetzbarkeit demonstrieren und Ausgangspunkt für weitere Arbeiten bilden soll.

Außerdem soll der Grad der Berechenbarkeit ermittelt und evaluiert werden, ob mittels der verfügbaren Gadgets eine ROP Chain für den gewünschten Payload gefunden werden kann.

Anhand verschiedener konstruierter sowie realer Beispiele soll der PoC evaluiert und die Umsetzbarkeit eines ROP Compilers nachgewiesen werden.

Zum Schluss werden die Ergebnisse mit anderen ROP Compilern verglichen und die Unterschiede in Performance sowie Ergebnissen evaluiert.

2.2 Abgrenzung

In dieser Arbeit wird die Suche von ROP Gadgets selbst nicht betrachtet. Projekte und Vorarbeiten zu dieser Thematik sind bereits gegeben und dienen als Basis für diese Arbeit.

Das genaue Format des Payload ist in dieser Arbeit noch zu definieren und könnte in Form von Pseudocode oder Konfigurationsvorgaben erfolgen. Eine universelle Übersetzung von Programmcode (z.B. C) kann innerhalb dieser Arbeit voraussichtlich nicht erfolgen, und würde Gegenstand weiterer Arbeiten bilden.

Aufgrund von grundsätzlichen Unterschieden in den verschiedenen CPU-Architekturen, Befehlssätzen sowie Executable File Formats (z.B. ELF, Mach-O, PE, usw.) wird sich in dieser Arbeit auf eine Architektur sowie ein File Format fokussiert. Ein universeller ROP Compiler für beliebige Architekturen kann innerhalb dieser Arbeit nicht entwickelt werden. Nachfolgende Arbeiten und Projekte können aber auf dieser Basis aufbauen.

3 Stand der Forschung

ROP und JOP sind Techniken, welche bereits seit Ende der 90er Jahre genutzt werden. Einer der wohl bekanntesten Exploits auf ROP Basis ist der return-into-libc Overflow Exploit [Des97].

Für das Auffinden von Gadgets gibt es ebenfalls bekannte Projekte, welche in der Lage sind teilweise für verschiedene Prozessorarchitekturen Gadgets aus einer gegebenen Binary zu finden und anzeigen zu lassen. Ein Beispiel für solch ein Projekt wären ROPgadget [Sala] oder Ropper [Salb].

Q [SAB11] war eine Forschungsarbeit der Carnegie Mellon University in Pittsburgh. In diesem Projekt wurden "semantic program verification techniques" genutzt, um einen gegebenen Exploit, welcher durch Sicherheitsmechanismen wie W$\oplus$X verhindert wird, in eine ROP Chain abzubilden, um den "Exploit zu härten" (exploit hardening). Ansätze der Semantik zur Verkettung können eventuell für diese Arbeit interessant werden.

angrop [anga] ist ein Tool zum Auffinden von ROP Gadgets. Außerdem soll es ROP Chains automatisch verketten können. Es nutzt das Binary Analysis Framework angr [angb], welches von verschiedenen Forschern vom Computer Security Lab der UC Santa Barbara sowie dem SEFCOM der Arizona State University entwickelt wird. Gefördert wurde das Projekt von der U.S. Regierung. Die Ergebnisse wurden unter open source gestellt [SWS+16]. angrop verfolgt den gleichen Ansatz von *Q* mit einigen kleineren Abweichungen und Erweiterungen.

GENROP [Bra22] war Thema einer Masterthesis an der Linköping University in Schweden. Das Thema von GENROP war sehr ähnlich zu dieser Arbeit und verfolgte den Ansatz Genetic Programming zum Verketten der ROP Gadgets zu nutzen. Das Ergebnis von GENROP war allerdings, dass die eigentlichen Kernkonzepte von Genetic Programming am Ende nicht mehr richtig genutzt werden konnten, um das Problem zu lösen, da der Algorithmus zu sehr gelenkt werden musste. Genauere Einarbeitung und Analyse, welche Probleme bei GENROP auftraten, müssen noch

erfolgen. Die Implementation des Projekts ist allerdings nicht öffentlich verfügbar.

Pure-Call Oriented Programming (PCOP) [SNR18] fokussiert sich auf Jump-oriented Programming (JOP) und somit Gadgets, welche in einer `call` Anweisung enden. Call Anweisungen werden vom Prozessor umgesetzt, indem die Speicheradresse der auf den Call direkt folgenden Anweisung auf den Memory Stack gelegt und eine `jmp` Anweisung zu der gewünschten Speicheradresse ausgeführt wird. Dadurch, dass der Stack verändert wird, ergeben sich Seiteneffekte, welche die Gadget Chain stören können. PCOP soll einige dieser Seiteneffekte beheben können. Außerdem soll PCOP Touring-Vollständigkeit besitzen, was bedeuten würde, dass beliebiger Code hierdurch emuliert werden könnte. Die genauen Ansätze in diesem Artikel müssen im weiteren Verlauf noch herausgearbeitet werden.

Am Massachusetts Institue of Technology (MIT) wurde ein ROP Compiler [SD15] erarbeitet, welcher die verfügbaren Gadgets zuerst klassifiziert (basierend auf Logik aus *Q* [SAB11]) und anschließend über die klassifizierten Gadgets iteriert. Außerdem soll der vom MIT sogenannte *Gadget Scheduler* auch bestimmte Speicherregister beobachten, sodass keine gegenseitigen Überschreibungen stattfinden sollen. Das Projekt soll auf konstruierten Beispielen sowie auf `rsync` als real-world Binary erfolgreich als PoC umgesetzt werden. Leider ist lediglich der Artikel und nicht die Codebasis öffentlich verfügbar.

Multi-Architecture JOP and ROP Chain Assembler (MAJORCA) [NVLK21] war ein Projekt des Moscow Institue of Physiscs and Technology, welches “spezielle Graphe zur Suche nach `mov` Ketten” zur Bildung der Gadget Chains genutzt hat. Während sich dieses Projekt allerdings auf die x86 Architektur fokussiert, können Ideen und Ansätze hieraus eventuell auch für diese Masterthesis interessant sein.

Es existieren somit einige Projekte, welche sich mit der automatisierten Generierung von ROP Chains befasst haben. Allerdings sind diese entweder nicht verfügbar, funktionieren nicht in realistischen Umgebungen, arbeiten auf anderen Architekturen (meistens x86), oder sind nicht richtig ausgearbeitet.

Im Rahmen der weiteren Vor- und Recherchearbeiten sollen die einzelnen Techniken der vorgestellten sowie weiterer Arbeiten genauer herausgearbeitet, verglichen sowie für die Bildung der Semantik in dieser Arbeit herangezogen werden.

4 Vorarbeiten

Diese Abschlussarbeit wird in Kooperation mit der *Zentralen Stelle für Informationstechnik im Sicherheitsbereich (ZITiS)* erstellt. Hierzu wurde bereits Kontakt mit der ZITiS aufgenommen sowie alle benötigten vertraglichen Grundsätze geklärt. Der offizielle Vertragsbeginn mit der ZITiS wurde auf den 01.03.2023 festgelegt.

Zur Bearbeitung wurde bereits durch die ZITiS ein Betreuer festgelegt, mit welchem bereits ein erster Austausch stattgefunden hat. Des Weiteren soll durch die ZITiS ab offiziellem Vertragsbeginn Hardware sowie Zugang zu gewissen Informationen und Infrastruktur bereitgestellt werden.

Eine Einarbeitung in das Thema, die technischen Grundlagen sowie etwaige Suche von Quellen und verwandten Projekten, fand und findet bereits statt.

5 Vorgehensweisen und Methoden

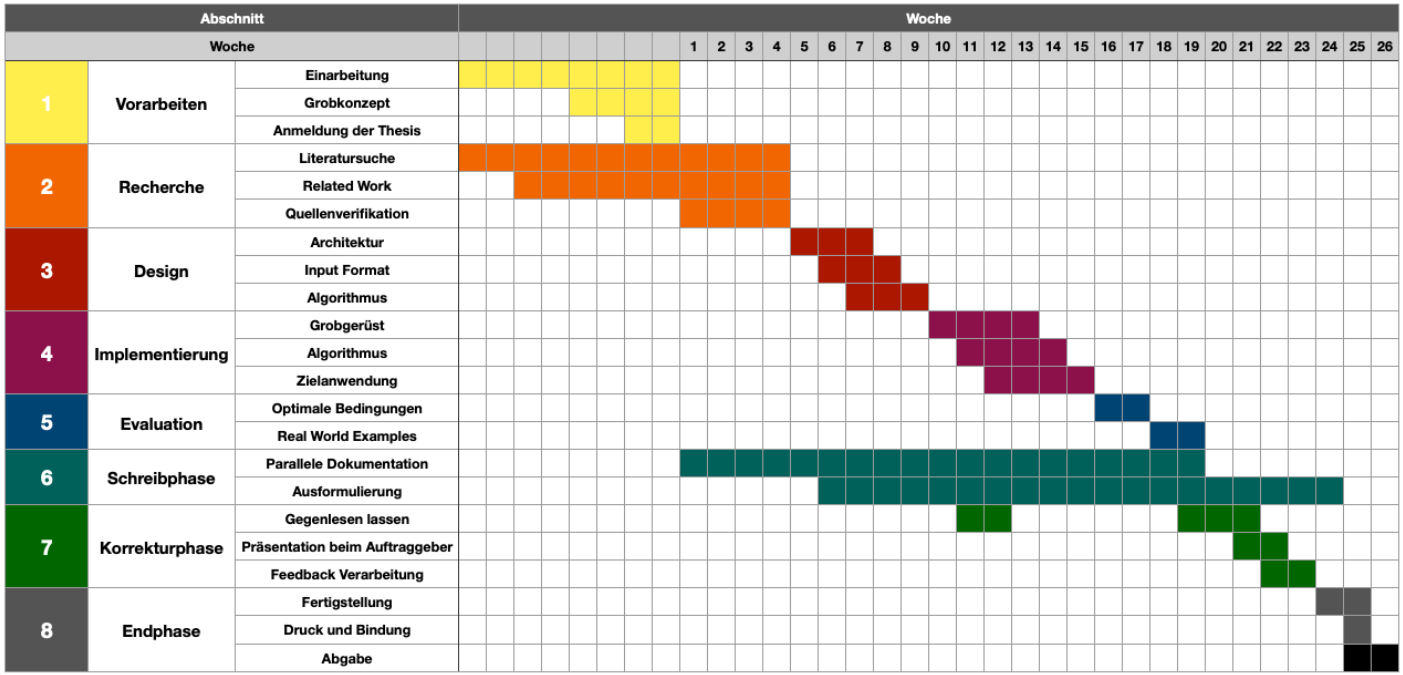
Als konkrete Prozessorarchitektur soll voraussichtlich x64 und als Executable File Format ELF ausgewählt werden.

Als Eingabeformat für die ROP Gadgets soll voraussichtlich JSON oder XML genutzt werden.

Die Ermittlung und Festlegung einer konkreten Logik des Algorithmus zur Verkettung der ROP Gadgets zu einer ROP Chain muss im Rahmen der weiteren Bearbeitung noch erfolgen. Ideen hierzu bilden das Genetic Programming (GP) aus dem Bereich der künstlichen Intelligenz sowie Methoden aus dem Compilerbau, z.B. LALR-, Bottom-Up- oder Top-Down-Parser. Ebenfalls sollen Ideen aus den Bereichen Exhaustive Search, Backtracking sowie weiteren Graph und Tree Search Algorithms gesammelt und betrachtet werden.

Format und Auswahl der Zielcodes (Payload) soll ebenfalls noch stattfinden. Eine Möglichkeit wäre dabei die Eröffnung einer Reverse- oder Bind-Shell, das Nachladen sowie Ausführen einer entfernten Datei oder einfache Manipulation bestimmter Ressourcen. Weitere Möglichkeiten sowie die Festlegung soll im Verlauf der Arbeit noch erfolgen.

6 Zeit und Arbeitsplan



7 Grobe Gliederung

Abstract

Table of Contents

List of Figures

List of Tables

List of Acronyms

1 Introduction

- 1.1 Motivation
- 1.2 Challenges
- 1.3 Goals and Outline

2 Background

3 Related Work

4 Design

5 Implementation

6 Evaluation

7 Conclusion

- 7.1 Future Work

Bibliography

Appendices

- A Misc
- B Stuff

Literaturverzeichnis

[anga] *angr/angrop*. https://github.com/angr/angrop. – Accessed: 09.01.2023

[angb] *angr.io*. https://angr.io. – Accessed: 10.01.2023

[Bra22] BRANTING, Jonatan: *ROP-chain generation using Genetic Programming: GENROP*, Diplomarbeit, 2022

[Des97] DESIGNER, Solar: *Getting around non-executable stack (and fix).* https://seclists.org/bugtraq/1997/Aug/63. 1997. – Accessed: 09.01.2023

[NVLK21] NURMUKHAMETOV, Alexey ; VISHNYAKOV, Alexey ; LOGUNOVA, Vlada ; KURMANGALEEV, Shamil: MAJORCA: Multi-Architecture JOP and ROP Chain Assembler. In: *2021 Ivannikov Ispras Open Conference (ISPRAS)* IEEE, 2021, S. 37–46

[SAB11] SCHWARTZ, Edward J. ; AVGERINOS, Thanassis ; BRUMLEY, David: Q: Exploit hardening made easy. In: *20th USENIX Security Symposium (USENIX Security 11)*, 2011

[Sala] SALWAN, Jonathan: *JonathanSalwan/ROPgadget*. https://github.com/JonathanSalwan/ROPgadget. – Accessed: 09.01.2023

[Salb] SALWAN, Jonathan: *sashs/Ropper*. https://github.com/sashs/Ropper. – Accessed: 09.01.2023

[SD15] STEWART, Jeff ; DEDHIA, Veer: ROP Compiler. In: *URL: https://css.csail.mit.edu/6.858/2015/projects/je25365-ve25411.pdf* (2015)

[SNR18] SADEGHI, AliAkbar ; NIKSEFAT, Salman ; ROSTAMIPOUR, Maryam: Pure-Call Oriented Programming (PCOP): chaining the gadgets using call instructions. In: *Journal of Computer Virology and Hacking Techniques* 14 (2018), Nr. 2, S. 139–156

[SWS+16] SHOSHITAISHVILI, Yan ; WANG, Ruoyu ; SALLS, Christopher ; STEPHENS, Nick ; POLINO, Mario ; DUTCHER, Audrey ; GROSEN, Jessie ; FENG, Siji ; HAUSER, Christophe ; KRUEGEL, Christopher ; VIGNA, Giovanni: SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. (2016)