Hochschule Bochum IT-Sicherheit – Übungen und Praktikum

Labor für Informatik und Mathematik im Anwendungsfeld Industrie 4.0

Themenbereich: Betriebssystemsicherheit

Prof. Dr. Christian Scheffer
Stefan Hausotte, M.Sc.
Niklas Schütrumpf, B.Sc.

Diese Übungen behandeln grundlegende Konzepte der Betriebssystemsicherheit mit besonderem Fokus auf Binary Exploitation, Reverse Engineering und Rootkit-Technologien. Die Übungen dienen zum tieferen Verständnis von Angriffsvektoren auf Systemebene und der praktischen Analyse von Sicherheitslücken. Die praktische Arbeit mit Binary-Analyse-Tools ermöglicht das unmittelbare Nachvollziehen von Exploits und Abwehrmaßnahmen.

Das Übungsblatt besteht aus fünf Aufgaben, von denen drei als Pflichtaufgaben (mit * markiert) von allen Studierenden zu bearbeiten sind. Die übrigen Aufgaben dienen der zusätzlichen Übung und zur Prüfungsvorbereitung.

Die Abgabe erfolgt in Gruppen von bis zu 3 Personen über die Moodle-Plattform. Bitte kommentieren und dokumentieren Sie Ihren Code, sodass ein Code-Review ohne große Hürden möglich ist.

Übungsaufgaben

Aufgabe 1* - CrackMe

Die Linux-Binary *Aufgabe1* ist eine CrackMe Datei, welche einen Parameter (*char* argv[]*) akzeptiert. Ist der eingegebene Parameter korrekt, erhalten Sie eine Flag in der Konsole.
Aufgabenstellung:

- Analysieren Sie die Binary mit geeigneten Tools (z.B. `objdump`, `gdb`, `strings`, `strace`, `radare2` oder `Ghidra`).
- Identifizieren Sie den korrekten Parameter und dokumentieren Sie die erhaltene Flag.
- Beschreiben Sie detailliert Ihren Lösungsweg und die verwendeten Tools.
- Analysieren Sie den Programmablauf und erklären Sie, wie die Validierung des Parameters funktioniert.

Dokumentation: Erstellen Sie einen Bericht, der folgende Punkte enthält:

- Die gefundene Flag
- Eine Schritt-für-Schritt-Anleitung Ihres Reverse-Engineering-Prozesses
- Screenshots der verwendeten Tools und deren Ausgaben
- Eine Erklärung der Validierungslogik des Programms
- Diskussion möglicher Schutzmaßnahmen gegen Reverse Engineering

Aufgabe 2* - Buffer Overflow

Die *Aufgabe2* ist anfällig für Buffer-Overflows. Identifizieren Sie die Schwachstelle in dem Programm und führen Sie einen Buffer-Overflow durch.
Aufgabenstellung:

- Analysieren Sie das Programm und identifizieren Sie die Buffer-Overflow-Schwachstelle.
- Ermitteln Sie die kürzestmögliche Eingabe, welche zu einem Buffer-Overflow führt.
- Führen Sie den Buffer-Overflow erfolgreich durch und dokumentieren Sie die verwendeten Eingaben (Benutzername und Kennwort).
- Erhalten Sie die Flag durch erfolgreiche Ausnutzung der Schwachstelle.
- Analysieren Sie den Stack-Aufbau und erklären Sie, wie der Overflow die Programmausführung beeinflusst.

Dokumentation: Erstellen Sie einen Bericht, der folgende Punkte enthält:

- Verwendeter Benutzername und Kennwort für den erfolgreichen Exploit
- Die erhaltene Flag
- Detaillierte Analyse der Schwachstelle im Quellcode
- Erklärung des Buffer-Overflow-Mechanismus mit Stack-Diagrammen
- Diskussion von Schutzmaßnahmen gegen Buffer-Overflows (ASLR, Stack Canaries, DEP/NX)
- Demonstration verschiedener Payload-Längen und deren Auswirkungen

Aufgabe 3 - Userland Rootkit

Implementieren Sie ein Userland Rootkit, das alle Dateien mit *malware* im Namen versteckt.
Aufgabenstellung:

- Erstellen Sie eine Shared Library, die die `readdir`-Funktion überschreibt, um Dateien mit *malware* im Namen zu verstecken.
- Verwenden Sie **LD_PRELOAD** zum Laden Ihrer Library.
- Testen Sie Ihr Rootkit mit dem `ls`-Befehl.
- Nennen Sie Ihre Source-Code-Datei *myreadlib.c*.
- Erstellen Sie ein Makefile zum Kompilieren der Shared Library *libmy_readdir.so*.

Dokumentation: Erstellen Sie einen Bericht, der folgende Punkte enthält:

- Vollständiger Quellcode der Shared Library mit Kommentaren
- Makefile für die Kompilierung
- Anleitung zur Installation und Verwendung des Rootkits
- Demonstration der Funktionalität (Screenshots von vor/nach der Installation)
- Erklärung der LD_PRELOAD-Technik und deren Funktionsweise
- Diskussion von Erkennungsmaßnahmen und Schutzverfahren gegen solche Rootkits

Hinweis: Orientieren Sie sich gerne an diesem Blogpost.

Aufgabe 4* - Integer Overflow

Erstellen Sie eine Demonstration, welche einen Authentifizierungsflow implementiert und durch einen Integer Overflow umgangen werden kann.
Aufgabenstellung:

- Implementieren Sie ein Programm mit einem Authentifizierungsmechanismus, der anfällig für Integer Overflow ist.
- Verwenden Sie eine Programmiersprache Ihrer Wahl (vorzugsweise C/C++).
- Kommentieren Sie alle wichtigen Code-Zeilen ausführlich.
- Demonstrieren Sie die erfolgreiche Umgehung der Authentifizierung durch Ausnutzung des Integer Overflows.
- Testen Sie verschiedene Eingabewerte und dokumentieren Sie deren Auswirkungen.

Dokumentation: Erstellen Sie einen Bericht, der folgende Punkte enthält:

- Vollständiger Quellcode mit ausführlichen Kommentaren
- Verwendete Compiler-Flags und Begründung der Auswahl
- Schritt-für-Schritt-Anleitung zur Umgehung der Authentifizierung
- Mathematische Erklärung des Integer Overflow-Mechanismus
- Beispiele verschiedener Eingabewerte und deren Berechnungen
- Diskussion von Präventionsmaßnahmen gegen Integer Overflows
- Analyse der Auswirkungen in verschiedenen Datentypen (signed/unsigned)

Bonuspunkteaufgabe (2 Bonuspunkte)

Advanced Binary Exploitation mit ROP-Chain
Ziel: Entwickeln Sie einen komplexeren Exploit, der moderne Sicherheitsmaßnahmen umgeht und Return-Oriented Programming (ROP) verwendet.
Anforderungen:

1. Erstellen Sie ein verwundbares C-Programm mit folgenden Eigenschaften:
    - Buffer-Overflow-Schwachstelle in einer Funktion
    - Das Programm soll mit modernen Schutzmaßnahmen kompiliert werden (DEP/NX aktiviert, aber ASLR deaktiviert für Vereinfachung)
    - Implementieren Sie eine “win“-Funktion, die nicht direkt aufrufbar ist
    - Verwenden Sie mehrere Funktionen mit verschiedenen Parametern
2. Entwickeln Sie einen ROP-basierten Exploit:
    - Identifizieren Sie geeignete ROP-Gadgets im Programm oder in verlinkten Libraries
    - Konstruieren Sie eine ROP-Chain, die die "winFunktion aufruft
    - Dokumentieren Sie jeden Schritt der ROP-Chain-Konstruktion
    - Verwenden Sie Tools wie `ropper`, `ROPgadget` oder ähnliche zur Gadget-Suche
3. Implementieren Sie ein Exploit-Skript:
    - Erstellen Sie ein Python-Skript, das den Exploit automatisiert
    - Das Skript soll die ROP-Chain dynamisch generieren
    - Implementieren Sie Fehlerbehandlung und Debugging-Ausgaben
    - Demonstrieren Sie die erfolgreiche Ausführung des Exploits
4. Erweiterte Analyse:
    - Analysieren Sie, wie sich der Exploit bei aktiviertem ASLR verhalten würde
    - Diskutieren Sie mögliche Umgehungsstrategien für ASLR
    - Erklären Sie, wie moderne Mitigationen wie CET (Control Flow Enforcement Technology) den Angriff verhindern würden
    - Entwickeln Sie Gegenmaßnahmen für Ihr verwundbares Programm
5. Dokumentation:
    - Erstellen Sie eine umfassende technische Dokumentation
    - Erklären Sie die Theorie hinter ROP-Angriffen
    - Dokumentieren Sie jeden Schritt der Exploit-Entwicklung
    - Erstellen Sie Diagramme des Stack-Layouts vor und nach dem Overflow

- Diskutieren Sie die Relevanz von ROP in der modernen IT-Sicherheit

6. Präsentation:
    - Erstellen Sie eine kurze Demonstration in Form eines Videos
    - Zeigen Sie die Funktionalität des ursprünglichen Programms
    - Demonstrieren Sie den erfolgreichen Exploit
    - Erklären Sie die wichtigsten Konzepte während der Demonstration