ArchSec
Übungsblatt 5

Hackerpraktikum

FAU Erlangen-Nürnberg, Lehrstuhl für Informatik 1

Binary Exploitation (Aufgabe 1)

Unter dem Link[1] ist mit dem Passwort `hackpra<i1>archsec` ein Archiv `flip.tar.gz` erreichbar, das die für diese Aufgabe benötigten Materialien enthält. Laden Sie das Archiv herunter, entpacken Sie es und machen Sie sich mit den enthaltenen Ordnerstrukturen vertraut. Lesen Sie die Datei `README.md`.

1. Diese Teilaufgabe beschäftigt sich zunächst mit der **AMD64**-Anwendung `flip`, welche sich im Ordner `amd64` befindet. Der Quellcode des Programms steht in der Datei `flip.cpp` im Hauptverzeichnis des Archivs zur Verfügung.
   a) Machen Sie sich mit dem Programm und dessen Funktionsweise vertraut. Beschreiben Sie kurz die Funktion des Programms. Wie groß ist der Puffer, in den geschrieben wird? Welche Anweisung im C-Programm ist fehlerhaft und stellt ein Sicherheitsrisiko dar? (0.75 P.)
   b) Bringen Sie das Programm mit einer von Ihnen kontrollierten Eingabe, welche den *Return Instruction Pointer* überschreibt, aufgrund einer `Segmentation Fault` zum Absturz. Erklären Sie, warum es genau zum Absturz kommt. Visualisieren Sie den Stack vor und nach dieser Anweisung (z.B. als `ASCII`-Art). (1 P.)

      Tipp: Sie dürfen statische und dynamische Analysewerkzeuge nutzen.
   c) Manipulieren Sie mit Ihrer Eingabe den Programmfluss von `flip` so, dass die Funktion `secret` ausgeführt wird:
      i. Bestimmen Sie die Adresse der Funktion `secret` in `flip`. (0.25 P.)
      ii. Bleibt die Funktionsadresse bei mehreren Aufrufen von `flip` identisch? Falls ja, warum ist dies so, obwohl auf Ihrem System unter Umständen ASLR aktiviert ist? (1 P.)

         Tipp: Wenn Sie mit GDB arbeiten, bedenken sie, dass GDB selbst unter Umständen ASLR beeinflusst[2]. Die Aufgabenstellung bezieht sich explit auf ein System auf dem ASLR aktiviert ist und nicht von GDB deaktiviert wird.
      iii. Schreiben Sie ein Skript, das eine Eingabe für `flip` erzeugt, welche `secret` aufruft. (0.25 P.)
      Tipp: Achten Sie dabei auf die Little-Endian Darstellung von Adressen auf AMD64.

[1] `https://faubox.rrze.uni-erlangen.de/getlink/fiNKRQU6GDx5SRfcpQ6BDiMY/`
[2] `https://visualgdb.com/gdbreference/commands/set_disable-randomization`

iv. Injizieren Sie die Ausgabe Ihres Skripts als Eingabe in `flip` so dass schließlich `secret` ausgeführt wird. (0.5 P.)

2. Diese Teilaufgabe beschäftigt sich mit der **ARMv8-A**-Anwendung `flip`. Im Ordner `aarch64` befindet sich eine ausführbare Datei, welche den Emulator QEMU[3] mit den passenden Argumenten startet. Prüfen Sie, ob Ihre Distribution eine vorkompilierte Version von QEMU zur Verfügung stellt. Die Aufgaben wurden auf einem Ubuntu 20.04 (Paket `qemu-system-arm`) System mit QEMU `4.2.1` getestet. Ihr Setup ist erfolgreich, wenn die VM startet und Sie sich mit Nutzer `i1stud` und Passwort `i1stud` erfolgreich einloggen können. Den Ordner `share` können Sie nutzen, um Dateien zwischen dem Host und der VM zu transferrieren. Innerhalb der VM ist dieser an `/mnt` eingebunden. Sobald die VM gestartet ist können sie sich außerdem von ihrem Host aus per SSH mit `ssh -p 5555 i1stud@127.0.0.1` einloggen. Sie können QEMU durch drücken der Tastenkombination `Strg+A`, dann `x` beenden.

   In der VM ist die Applikation `flip` unter `/usr/bin/flip` installiert. Es handelt sich dabei um dieselbe Applikation wie in der vorherigen Teilaufgabe (*identischer Quelltext*). Der einzige Unterschied ergibt sich durch die Kompilierung für die ARMv8-A Architektur.

   a) Bringen Sie das Programm mit einer von Ihnen kontrollierten Eingabe, welche das *Saved Link Register* überschreibt, aufgrund einer `Segmentation Fault` oder einer `Illegal Instruction` zum Absturz. Erklären Sie, warum es genau zum Absturz kommt. Zeichnen Sie den Stack vor und nach dieser Anweisung als `ASCII`-Art. (1 P.)

   b) Vergleichen Sie die textuellen Ausgaben des abstürzenden Programms aus dieser Teilaufgabe mit der Ausgabe der AMD64-Applikation aus der vorherigen Teilaufgabe. Unterscheiden sich die Ausgaben? Wenn ja, erläutern Sie detailliert, warum dies der Fall ist. (1 P.)

   c) Schreiben Sie ein Skript, das eine Eingabe für `flip` erzeugt, welche `secret` aufruft. Injizieren Sie die Ausgabe Ihres Skripts als Eingabe in `flip` so dass schließlich `secret` ausgeführt wird. (0.25 P.)

6 P.

Binary Cracking (Aufgabe 2)

Unter dem Link[4] ist mit dem Passwort `hackpra<i1>archsec` ein Archiv `crackme.tar.gz` erreichbar, das die für diese Aufgabe benötigten Materialien enthält. Laden Sie das Archiv herunter, entpacken Sie es und machen Sie sich mit den enthaltenen Ordnerstrukturen vertraut. Das Archiv enthält eine ausführbare Datei, welche den Emulator QEMU mit einem Abbild einer virtuellen Maschine (ARMv8-A) startet. Das Setup und die

[3] `https://www.qemu.org/`

[4] `https://faubox.rrze.uni-erlangen.de/getlink/fiNKRQU6GDx5SRfcpQ6BDiMY/`

grundlegende Ausstattung der VM ist gleich zur vorherigen Aufgabe (gleiche Software-Abhängigkeiten, SSH Server, `i1stud` Nutzer, ...). Sie dürfen für alle Teilaufgaben sowohl statische als auch dynamische Analysewerkzeuge nutzen, um den Programmfluss nachzuvollziehen.

Die untenstehenden Programme besetehen aus einem in `C` programmierten Einstiegspunkt und rufen dann gelegentlich Funktionen aus der integrierten Bibliothek `hackpra` auf, welche in `Rust` programmiert wurde. Für die in C geschrieben und die von `hackpra` exportierten Funktionen gilt die ARMv8-A Calling Convention.[5] Dies gilt jedoch nicht zwangsweise für die Interna von in `Rust` programmierten Funktionen.

1. In der VM ist ein Programm `crackme1` unter `/usr/bin` installiert. Beantworten Sie folgende Fragen. Achten Sie stets auf eine klare Darstellung des Lösungsweges.
   a) Wie lautet das korrekte Passwort? Beschreiben Sie, wie Sie an das Passwort gelangen sind. (0.5 P.)
   b) Woher stammt die Flagge und wie lautet diese? Erläutern Sie das Vorgehen, das notwendig ist, um an die Flagge zu gelangen. (1 P.)

2. In der VM ist ein Programm `crackme2` unter `/usr/bin` installiert. Beantworten Sie folgende Fragen. Achten Sie stets auf eine klare Darstellung des Lösungsweges.
   a) Welche Schutzmaßnahme wird getroffen, um das Speichern des korrekten Passwortes im Klartext zu verhindern? Erläutern Sie die Funktionsweise der Schutzmaßnahme. (0.5 P.)
   b) Können Sie ein hinreichend schnelles Programm schreiben, dass das ein derart geschütztes Passwort als Eingabe erhält und das Klartextpasswort ausgibt? Falls ja: Schreiben Sie ein solches Programm für AMD64 (empfohlen: in Rust). Falls nein: Warum nicht? (1 P.)
   c) Woher stammt die Flagge und wie lautet diese? Erläutern Sie die Programmschritte, die notwendig sind, um an die Flagge zu gelangen. (1 P.)

3. In der VM ist ein Programm `crackme3` unter `/usr/bin` installiert. Beantworten Sie folgende Fragen. Achten Sie stets auf eine klare Darstellung des Lösungsweges.
   a) Welche Schutzmaßnahme wird getroffen, um das Speichern des korrekten Passwortes im Klartext zu verhindern? (0.5 P.)
   b) Können Sie ein hinreichend schnelles Programm schreiben, dass für den Allgemeinfall ein derart geschütztes Passwort als Eingabe erhält und das Klartextpasswort ausgibt? Falls ja: Geben Sie das Programm an. Falls nein: Warum nicht? (0.5 P.)
   c) Wie lautet im konkreten Fall das korrekte Passwort? Beachten Sie den untenstehenden Hinweis. (1 P.)

[5] `https://developer.arm.com/documentation/ihi0055/latest`

d) Woher stammt die Flagge und wie lautet diese? Erläutern Sie die Programmschritte, die notwendig sind, um an die Flagge zu gelangen. (1 P.)

Hinweis: Für `crackme3` kann als bekannt vorausgesetzt werden, dass das korrekte Passwort aus einem englischen Wort gefolgt von einem Satzzeichen ('!', '?', '.', ',') besteht. Beispiel: `house?`

7 P.

Kernel Exploitation (Aufgabe 3)

Unter dem Link[6] ist mit dem Passwort `hackpra<i1>archsec` ein Archiv `capital.tar.gz` erreichbar, das die für diese Aufgabe benötigten Materialien enthält. Laden Sie das Archiv herunter, entpacken Sie es und machen Sie sich mit den enthaltenen Ordnerstrukturen vertraut. Das Archiv beinhaltet ein Abbild einer virtueller Maschine zum Bearbeiten der untenstehenden Teilaufgaben.

Das Archiv enthält eine ausführbare Datei, welche den Emulator QEMU mit den passenden Argumenten startet. Das Setup und die grundlegende Ausstattung der VM ist gleich zur vorherigen Aufgabe (gleiche Software-Abhängigkeiten, SSH Server, `i1stud` Nutzer, ...).

In der VM ist ein Kernel-Modul `capital` installiert und geladen, welches einen neuen System Call zum System hinzufügt. Der Quellcode des geladenen Kernelmoduls ist im Archiv mit hinterlegt. Machen Sie sich mit der Funktionsweise des Kernelmoduls vertraut.

1. Erklären Sie, welche Funktion der neu eingefügte System Call erfüllt. Schreiben Sie ein C-Programm, das den neuen System Call des Kernelmoduls korrekt verwendet. Beachten Sie, das auf dem Zielsystem selbst kein Compiler installiert ist. Eine für AArch64 kompilierte Binary kann über das `share` Verzeichnis in die VM gereicht weden. (1 P.)

2. Zeigen Sie mit einem weiteren C-Programm, dass der neue System Call eine kritische Schwachstelle beinhaltet. Erklären Sie, wie ihr Programm funktioniert. (2 P.)

3. Eskalieren Sie Ihre Privilegien mit einem weiteren C-Programm und lesen Sie den Inhalt der Datei `/flag` aus. Erklären Sie, wie ihr Programm funktioniert. (4 P.)

Hinweis: Der laufende Linux Kernel wurde zur Vorbereitung dieser Aufgabe gepatcht und entspricht nicht zwangsweise heutigen Sicherheitsstandards.

7 P.

$6 + 7 + 7 =$ **20 Punkte**

[6] `https://faubox.rrze.uni-erlangen.de/getlink/fiNKRQU6GDx5SRfcpQ6BDiMY/`