Executive Summary

Our project will demonstrate a ROP attack on ARM by building an application on the BBB and exploiting it.

Table of Contents

The Team

Pankul Garg
I am a Cybersecurity Graduate Student at University of Maryland College Park. Binary analysis and language based attacks are my primary fields of interest.

Shoumit Karnik
I am a Cybersecurity Graduate Student at University of Maryland College Park working to solve Cybersecurity issues. Discovering bugs/exploits, assessing their risk and impact on financial and software systems and security consultation are some of my key interest areas.

Farzin Nabili
I'm a cybersecurity system engineer working for Boeing Defense, Space & Security company. Currently, I'm a part-time graduate student at University of Maryland, College Park.

Khoa Nguyen
I'm a security software engineer working for a tech company in the greater Seattle area and a part-time remote student at UMD.

We communicate via Google Hangouts. Also, we are using Google Docs and Google Drive to manage and work on this project.

Related Works & Background

Knowledge of buffer overflow attacks, ARM architecture, disassembly and assembly language.

Buffer overflow attacks
A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. A buffer overflow occurs when more data is put into a fixed-length buffer than the buffer can handle. By sending suitably crafted user inputs to a vulnerable application, attackers can force the application to execute arbitrary code to take control of the machine or crash the system.

ARM architecture
https://developer.arm.com/architectures/learn-the-architecture/introducing-the-arm-architecture/single-page

Disassembly and assembly language
https://web.sonoma.edu/users/f/farahman/sonoma/courses/es310/310_arm/lectures/Chapter_3_Instructions_ARM.pdf

For reference, we will be using a research paper on gadget chaining. This paper is based on x64 but explains the concepts of chaining.
https://users.suse.com/~krahmer/no-nx.pdf

On x86 gadgets are small groups of instructions ending with a "ret" instruction. Each gadget ends with the "ret" instruction so gadgets can be chained together to perform arbitrary computations. Since the attacker controls the stack they can pop values into registers then execute code to use them. ARM architecture is different compared to x86, but it is still possible to use the ROP technique.

For learning ARM disassembly, please refer to the following link:
https://azeria-labs.com/writing-arm-assembly-part-1/
https://www.instructables.com/id/Beaglebone-Black-Web-Control-Using-WebPy/

For introduction to Return-Oriented Exploitation on ARM architectures we will be referring to the following links:

[1]https://media.blackhat.com/bh-us-11/Le/BH_US_11_Le_ARM_Exploitation_ROPmap_Slides.pdf

[2]https://icyphox.sh/blog/rop-on-arm/

[3]https://blog.3or.de/arm-exploitation-return-orientedprogramming.html

[4]https://www.exploit-db.com/docs/english/14548-exploitation-on-arm---presentation.pdf

[5]http://s3-us-west-2.amazonaws.com/valpont/uploads/20160326012043/Exception_handling.pdf

[6]https://www.sciencedirect.com/topics/computer-science/interrupt-vector-table

Detailed Description

What is your project idea?
We will be using an example binary file on BeagleBone Black (BBB) and try to exploit the vulnerabilities in the binary using Return-Oriented-Programming (ROP).

ROP is an exploitation technique where an attacker avoids security defenses such as executable space protection and code signing by exploiting a buffer overflow and stringing groups of instructions already existing in the target application, called “gadgets”, to do what the attacker wants.

Binaries compiled on ARM using insecure functions which can lead to buffer overflow, can be exploited by an ROP attack if the Address space layout randomization (ASLR) is switched off, we will be taking such an example binary to build our proof of concept (POC) and try to exploit the same using ROP.

How are you going to implement it?
We will be building an example executable which will have insecure functions leading to buffer overflow which will be used to do an ROP exploit. ROP requires a buffer overflow or memory write ability to change the return address on the stack. The idea that we have is to explore ROP in the context of hijacking the Interrupt Vector Table.
An interrupt vector table is a table of interrupt vectors (pointers to routines that handle interrupts). Actually, two copies of it are required to be present in the first 256 locations of the program memory. One is used during normal program execution, and the second (or Alternate IVT) during debugging.

What do you need in order to implement it?
The probable resources that we will be using to debug, disassemble and run the project are:

Software:
Vmware Workstation, Ubuntu VM, ARM disassembler (IDA, Ghidra, or gdb), C programming language, Python (exploit script), Putty/SSH.

Hardware:
BeagleBone Black, Mini USB cable, USB Serial cable, Power AC adapter, and SD card.

What milestones do you need to hit in order to demonstrate the idea?

- Create an example binary with buffer overflow for POC
- Find ROP gadgets (pieces of assembly code) which can be used for probable attacks
- Chain ROP gadgets to run the payload of our choice
- Use the ROP gadgets to break the executable Control Flow Integrity (CFI) and gain control
- Run and demonstrate a ROP attack
- Enumerate preventive measures as to how we could prevent the attack

A detailed timeline plan such that the project, presentation and demo is complete by end of day May 4, 2020.

- Final project proposal: March 31
- Create a POC binary with buffer overflow vulnerabilities: April 15
- Find ROP gadgets: April 18
- Chain ROP gadgets: April 22
- Use the ROP gadgets to break into the application: April 24
- Run and demonstrate a ROP attack: April 28
- Enumerate preventive measures: April 30
- Demos & presentation: May 3