Exploitation

IIIT Hyderabad

India

December '19

Abstract

A thorough foray into exploitation and explanation of why it works the way it does. It is a compilation of sorts, and hence no correlation between chapters (No flow b/w chapters exist) and the book should exist as a case study and used to solve problems which look similar and also has some basics which I go learning on the way forward.

Sources

The websites and competitions referred for writing this piece :

- CTF 101
- Shellter Labs
- Linux Man Pages
- picoCTF
- trailofbits
- Florida University Computer Security
- Megabeets Radare Tutorial
- Android Developers Website

Contents

Declaration

This is just to explain how certain exploits work in a great detail.

"*If it moves, compile it*".

Basics

Data

There are many places to store data on a typical computer like a hard drive (or any other secondary storage), RAM, Caches and Registers.

A hierarchy is decided on their access speed. If a piece of data is required more frequently, it is stored on a faster storage device. (Faster implies lower latency i.e. time taken from being asked for data and providing it). The figure 1 shows this heirarchy.

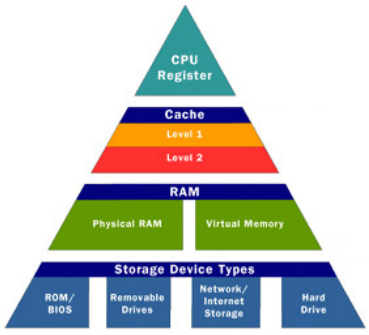

Figure 1: Latency Hierarchy

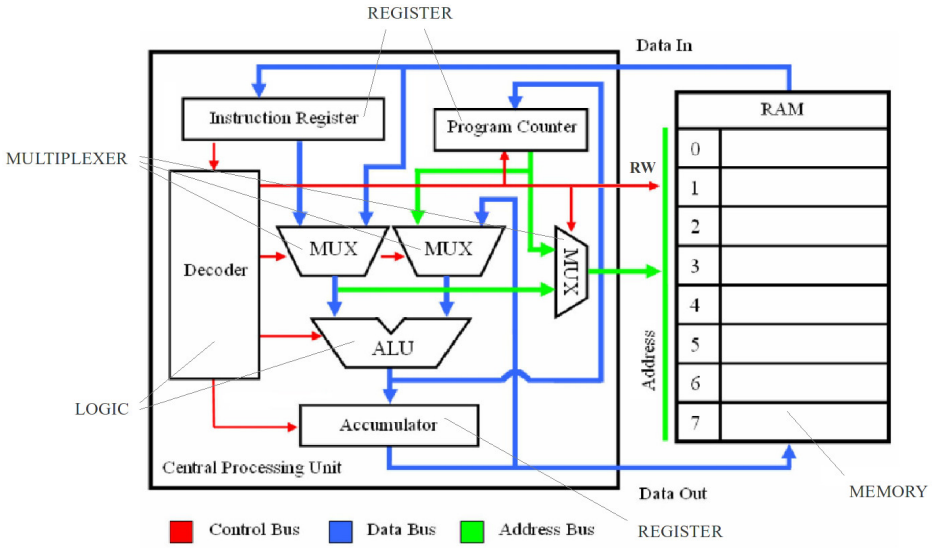

Figure 2: Circuit of CPU

Location

Register

A register is located in the CPU itself and as it is physically the closest, it also the fastest. It is the one that is accessed while running a program

A sample data flow can be seen in Figure 2, which shows data tranfer from RAM to Register and then to the CPU.

Cache

A cache is small piece of memory located near the CPU and works as a faster RAM (sort of). It caches the data which the OS thinks is required the most.

L1 & L2 cache is built into the proccessor (recent ones anyway) and is individual for every physical core whereas L3 cache is located outside of the actual silicon but is in the chip and so is common for all cores.

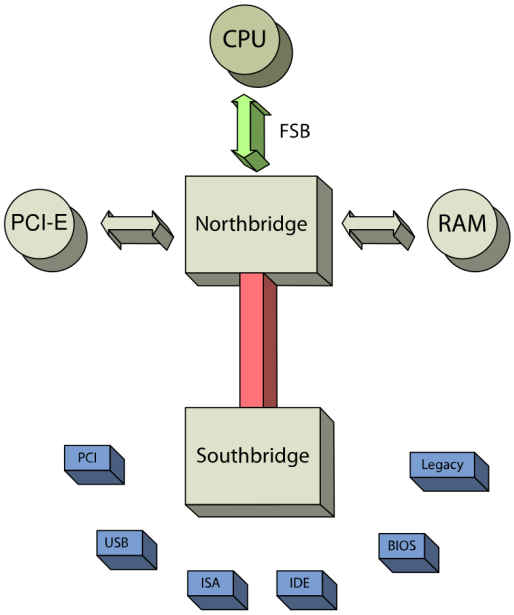

Figure 3: Use of bridges

RAM

RAM is a volatile memory where all the memory that is deemed by the OS as required is stored for instant access with the CPU. The data tranfer happens from RAM → Cache → Register.

Bridges

Bridges are small microcontrollers with built in logic which help in communication with external devices. There are two of there, and are descibed by their position on the board and also the threshold speed they can manage.

North Bridge

North bridge is a controller chip which connect high speed external devices to the chipset. It was originaly outside ot the main chip and an external silicon but now is mostly included in the SoC (System on a chip). The devices which can be connected to it are given in Figure 3.

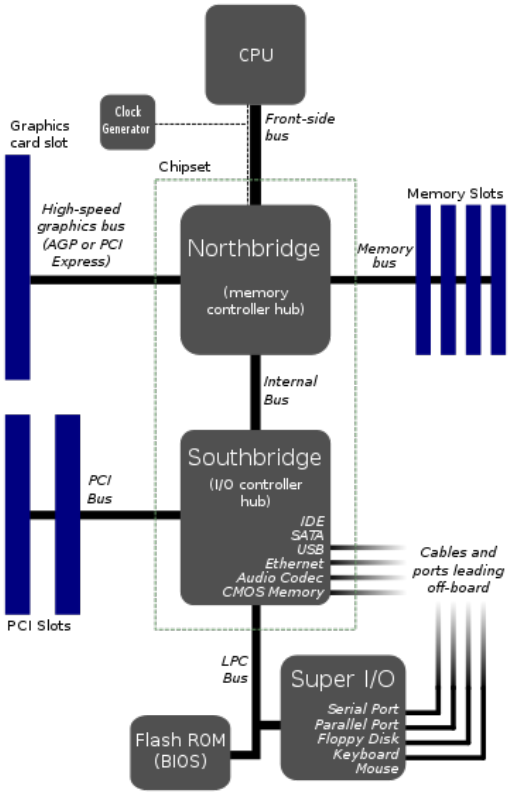

Figure 4: Schematic Design of Bridges

South Bridge

South bridge is the chipset which connects slower and legacy devices to the main chip. It is still seperate on Intel boards but is now starting to get integrated in AMD motherboards.

Data Flow

So the data flow happens as follows :

$$Solid\ State \rightarrow South\ Bridge \rightarrow RAM \rightarrow Register$$

So when a computer is started, it first POSTS and then goes to the BIOS after which the BIOS piont the program to the Magic Number (For legacy MBR systems) which has the bootloader in it (Like GRUB), after which the bootloader takes

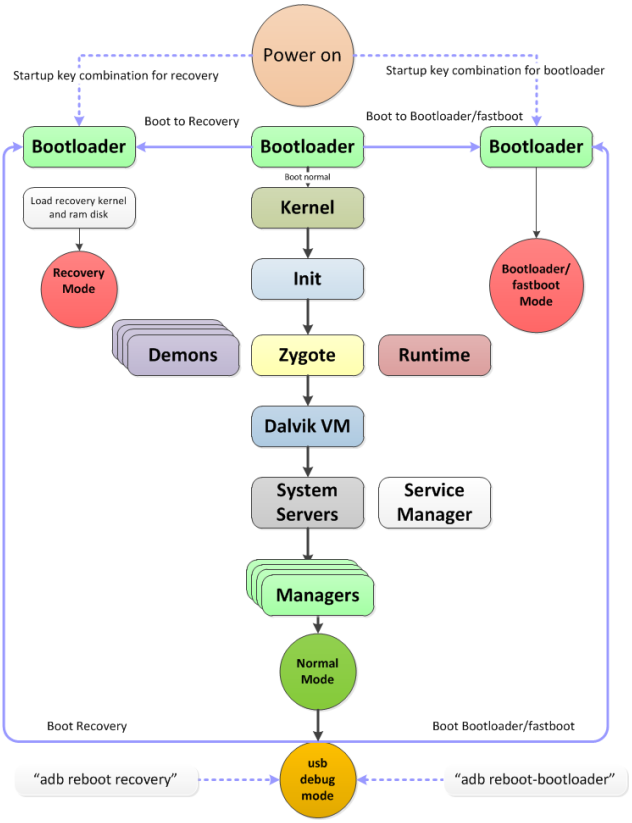

Figure 5: Android booting process

control of the System, then loades the kernel and with that the OS, which loads itself in the RAM, for fast access (The most recently executed programs still in the registers and cache) after which the OS loads a GUI (If it has one) or just greets to a TTY terminal. For the non GUI version, it gets you to a log in screen, whicle for GUI an application is loaded which takes care of logging in (Known as a Display Manager (DM)).

Tools

There are many tools which make analysing and understaing flaws of programs very easy. Most of them are not required and work can be done without them, but they are a good creature comfort and sholud be used as such.

These tools are used to analyse a piece of code and understand its vulnerabilites and also to generate payloads to make exploiting them easier.

GDB

GDB is one of the most importatnt tools. It is one of, if not the most importatnt tool for exploiting binaries. It is essentially a debugger which allows for dissasmbly of binaries, is also usefull for checking the flow of the the binary, and before Ghidra was one of the most popular tools to understand the working of a program.

It is still used for basic analysis, to check if the exploit works, and for initial routing checking. If someone is starting out with binary exploitation, then that someone should exclusively use GDB till the fundamentals of exploitation done are understood.

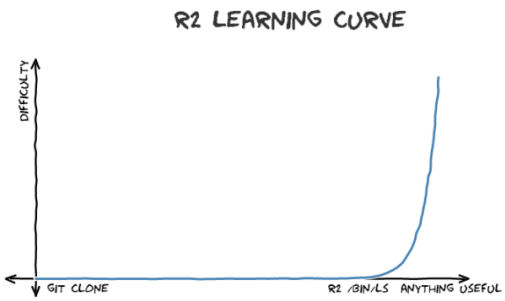

Figure 6: Radare Learning Curve

Basic Setup

GDB is usually pre installed on any linux machine. But if it isn't, it can be installed using the default package manager (Like apt or pacman).

There are some extensions for GDB that make it a much easier tool to operate with. You can use any number of them to make it to your liking, but in the following section, only some extensions are used and explained.

radare2

Radare is a tool which helps in understanding the binary, decompiling it and allowing to modify commands on the fly. It is considered one of the most difficult tools to master, so much so, that they themselves put Figure 6 on their website. I think this can be the Maya of exploiation tools.

Chapter 1

Binary Exploits

Introduction

Binary exploitation is the process of subverting a compiled application t it violates some trust boundary in a way that is advantageous to you, the attacker.The exploits are explained breifly and then given case studies for it to be understood better.

1.1 Buffer Overflow

A buffer overflow occurs when a piece of data overflows the storage space given to it. In these types of exploits, usually stack smashing occurs which changes value of non intended variables and helps in changing the flow of the program in some way.

1.1.1 Stack

One of the most important component to understand for binary exploitation and by extention buffer overflow is the stack. It is the location where all the variables and

Chapter 2

Systems and Exploitation

Introduction

The following chapter is an introduction to case studies of well knowns exploits, intoducing mobile systems and gaming consoles (The easiest to hack, for their exploitation is mainstream). This is an introduction to the basics of actual life exploiting and will go into details on how the exploits were used, how to install them, their working and also how to replicate it.

Specific systems will be covered in a one to one case based scenario.

Terminology

There are some very common terms encountered while browsing through this piece and also while refering to external resources specific to console exploitation (As I told you, it *was* (is) very mainstream)

Exploitation of systems with only a software is known as a softmod, whereas one done by physically modifying your hardware is known as hardmod. There are obvious advantages to both, a softmod is usually protected from in the next firmware update whereas hardmods are usually based on hardware vulnerabilities (But not

always) and thus are harder to protect against after product is in the hands of the public.

2.1 PSP

One of the most exploited (*Hacked*) system out there. It is the one that even I have reaped the benifits of, by doing *legal* things, of course. It is also the one which I remember following to check out if the next version was exploitable, was it safe, did the risks outweigh the positives, etc. So here, I will explain how the PSP was exploited, why was it *easy* and also replication.

2.2 Android

For android devices, it is not technically exploitaion as Android does not dissalow it, it is the OEMs which refuse access to root for customers. So rooting *technically*, if the OEM (like Xiaomi) allows it, is not exploitaiton, but it does allow access to the complete system, so I am going to proceed with calling it exploitaiton of Android.

Rooting on android differs from phone to phone (Or more like OEMs to OEMs), and can be trivially easy or a fairly complicated process. But the general flow of work goes like this :

Normal Device → Unlocking Bootloader → Installing a custom recovery → Installing a root manager (Like magisk or SuperSU) → Reboot → Rooted Android

2.2.1 Bootloader

The bootloader in an android device is hidden and is not accessible to the normal user. It is locked by the OEM, so in order to access it, it has to be unlocked first.

The method differs for every phone, and the details can be found on XDA Developers website*.

Bootloader is explained in detail in the Operating System section, but a brief understanding is that it is piece of code that points which OS (More specifically the kernel†) to load.

So the bootloader, after being unlocked will allow us to load a custom OS (Known as recovery). This is a standalone OS which allows us to *flash* a zip to the main partition.

2.2.1.1 Unlocking the Bootloader

TO DO

2.2.2 Recovery

Recovery is an OS, which has a single purpose of allowing recovery. What it means is that it is a small OS which helps in fixing your main OS by providing tools to fix it (Kind of like the live linux systems). It is a minimal shell which allows for some fastboot and posix commands to run.

It is most usefull in flashing zips which are either ROMs, custom kernels, or root binaries.The vanilla recovery that comes with the device does not allow for any such modifications, and this is where a custom recovery comes into picture. It allows for any modification and installation from within the recovery itself.

*xda-developers.com
†Learn More

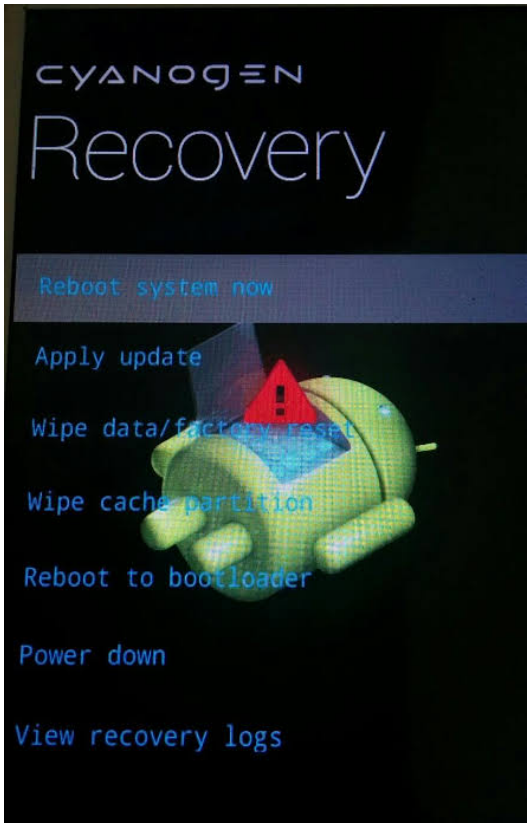

Figure 2.1: Recovery

2.2.2.1 Custom Recovery

Custom recovery is the software which performs function of a recovery but allows for remote application i.e no need for another device to give commands to the system for it to work. One of the earlier custom recovery software was ClockworkMod which was one of the primary custom recovery for Android versions till 4.0, after which TWRP started to take over as the preferred recovery software.

2.2.3 ADB and Fastboot

ADB and Fastboot are utilities

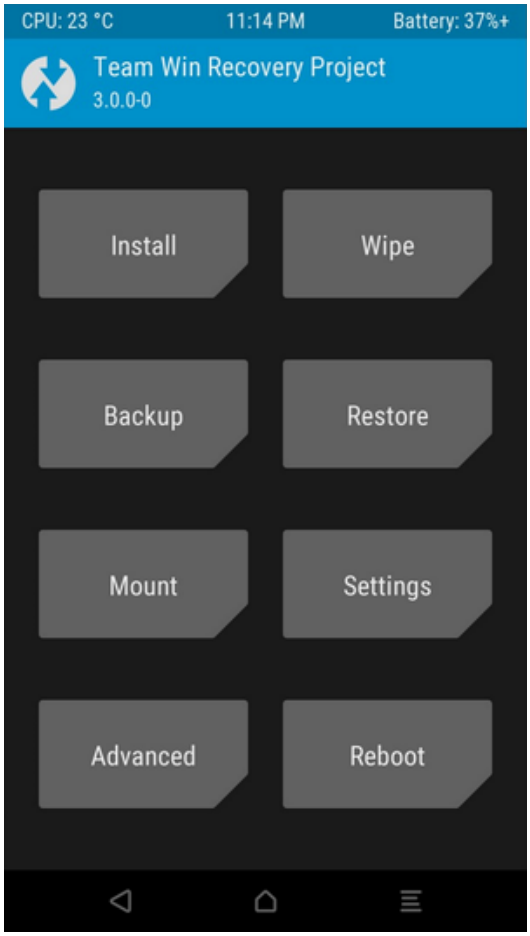

Figure 2.2: TWRP

2.2.3.1 Andoid Debugging Bridge

Android Debugging Bridge (known as ADB) is a tool which allow for communication with an android device via USB. It is a shell with basic commands that allow a device to execute *debug* commands. It is a basic version of a UNIX shell.

Bibliography