Beginner's Guide to Exploitation on ARM

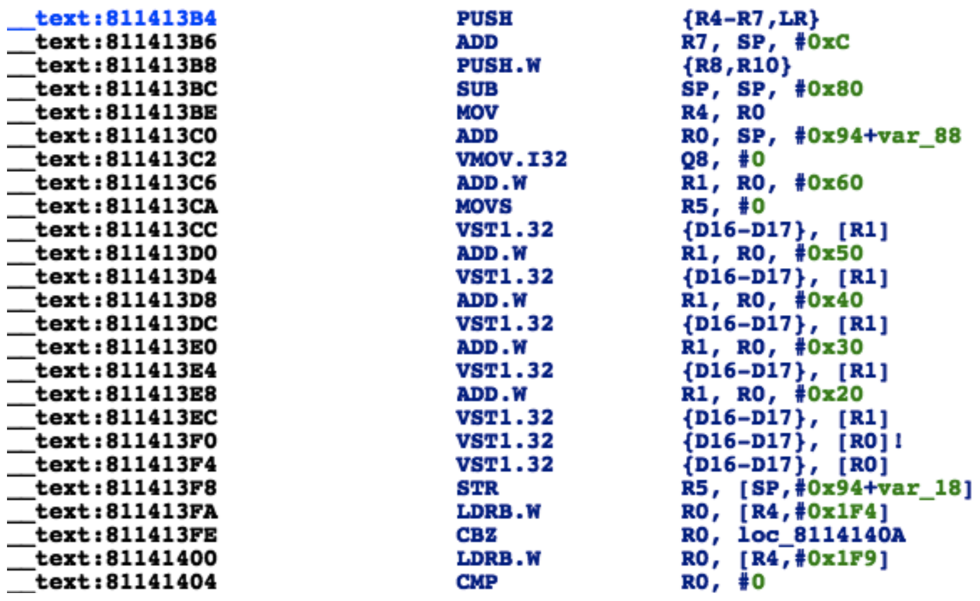
by Billy Ellis

1

Introduction

The term ‘hacking’ covers a huge range of methods used to access personal data or gain control over computer systems without the owners’ consent. Binary exploitation, however, is a specific branch of ‘hacking’ which involves manipulating the way in which an already-compiled program running on a system executes in order to give you, the attacker, an advantage over the system.

Binary exploitation often involves searching for vulnerabilities in a binary through the process of reverse engineering, in which the binary is disassembled (using a disassembler program such as IDA) and the assembly instructions are displayed for you.



```
__text:811413B4                 PUSH            {R4-R7,LR}
__text:811413B6                 ADD             R7, SP, #0xC
__text:811413B8                 PUSH.W          {R8,R10}
__text:811413BC                 SUB             SP, SP, #0x80
__text:811413BE                 MOV             R4, R0
__text:811413C0                 ADD             R0, SP, #0x94+var_88
__text:811413C2                 VMOV.I32        Q8, #0
__text:811413C6                 ADD.W           R1, R0, #0x60
__text:811413CA                 MOVS            R5, #0
__text:811413CC                 VST1.32         {D16-D17}, [R1]
__text:811413D0                 ADD.W           R1, R0, #0x50
__text:811413D4                 VST1.32         {D16-D17}, [R1]
__text:811413D8                 ADD.W           R1, R0, #0x40
__text:811413DC                 VST1.32         {D16-D17}, [R1]
__text:811413E0                 ADD.W           R1, R0, #0x30
__text:811413E4                 VST1.32         {D16-D17}, [R1]
__text:811413E8                 ADD.W           R1, R0, #0x20
__text:811413EC                 VST1.32         {D16-D17}, [R1]
__text:811413F0                 VST1.32         {D16-D17}, [R0]!
__text:811413F4                 VST1.32         {D16-D17}, [R0]
__text:811413F8                 STR             R5, [SP,#0x94+var_18]
__text:811413FA                 LDRB.W          R0, [R4,#0x1F4]
__text:811413FE                 CBZ             R0, loc_8114140A
__text:81141400                 LDRB.W          R0, [R4,#0x1F9]
__text:81141404                 CMP             R0, #0
```

From here, you can go on to carefully analyse each of the binary's functions and methods in hope of discovering a critical security vulnerability that can later be exploited.

Once a vulnerability has been discovered in a binary, depending on the nature of it, it can be exploited in a variety of ways and used to achieve a variety of things. One of the most powerful kinds of vulnerabilities are ones in which the attacker can take control over the Program Counter register (discussed in chapter 2) and thereby redirect the execution flow of a program wherever they want.

This type of vulnerability is especially powerful if found in an important system process or an OS's kernel as it will give the attacker the ability to compromise the entire system. This means that they can disable any OS-level security features, access private information and install their own malicious software/ malware onto the device without it being detected.

Fortunately for developers, many extra security mechanisms have been implemented in order to make exploitation of this vulnerabilities much more difficult and prevent people from using this vulnerabilities when developing malware. In most cases, these protections do their job effectively, but these too can be bypassed as will be discussed in later chapters.

Hardware Requirements

As almost all of my work and experience has been based around Apple iOS devices, to follow along with this book and complete some of the many example tasks that will be covered, it is recommended that you have a jail-broken iOS device at hand to test with.

However, although each example shown throughout the chapters will be demonstrated on iOS, it is still possible to follow along using another ARM-based device of your choice (although certain things may vary).

Software Requirements

To be able to reverse engineer binaries for static analysis, you will need a disassembler tool. There are a variety of options to choose from depending on what platform you are using.

If you are on Windows or OS X, you can use IDA Pro (although this is a very expensive option). For OS X as well as Linux, there is a cheaper alternative known as Hopper Disassembler.

If you want to do all of the work on your iOS device itself, you can use GDB or the popular radare2, both of which are available within the Cydia store. For other ARM platforms, you can cross compile radare2 using the source code on https://github.com/radare/radare2.

Each of the mentioned disassembler programs also function as debuggers and will allow you to attach to a running process.

Once you have downloaded a disassembler tool onto your chosen platform and have an ARM based device ready to test with, you can begin reading Chapter 2 to understand the fundamentals of the ARM platform.

Contact

billy@zygosec.com

https://twitter.com/bellis1000

https://twitter.com/zygosec