1. Record Nr. UNINA9910829852603321

Autore	Markstedter Maria
Titolo	Blue fox : arm assembly internals and binary analysis of mobile and IOT devices / / Maria Markstedter
Pubbl/distr/stampa	Hoboken, New Jersey : , : John Wiley & Sons, Ltd, , 2023
ISBN	1-394-18920-6
Descrizione fisica	1 online resource (xi, 219 pages) : illustrations
Disciplina	005.265
Soggetti	Assembly languages (Electronic computers) Internet of Things Embedded computer systems - Programming
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Nota di contenuto	Introduction -- Part I Arm Assembly Internals -- Chapter 1 Introduction to Reverse Engineering -- Introduction to Assembly -- Bits and Bytes -- Character Encoding -- Machine Code and Assembly -- Assembling -- Cross- Assemblers -- High- Level Languages -- Disassembling -- Decompilation -- Chapter 2 ELF File Format Internals -- Program Structure -- High- Level vs. Low- Level Languages -- The Compilation Process -- Cross- Compiling for Other Architectures -- Assembling and Linking -- The ELF File Overview -- The ELF File Header -- The ELF File Header Information Fields -- The Target Platform Fields -- The Entry Point Field -- The Table Location Fields -- ELF Program Headers -- The PHDR Program Header -- The INTERP Program Header -- The LOAD Program Headers -- The DYNAMIC Program Header -- The NOTE Program Header -- The TLS Program Header -- The GNU_EH_FRAME Program Header -- The GNU_STACK Program Header -- The GNU_RELRO Program Header -- ELF Section Headers -- The ELF Meta-Sections -- The String Table Section -- The Symbol Table Section -- The Main ELF Sections -- The .text Section -- The .data Section -- The .bss Section -- The .rodata Section -- The .tdata and .tbss Sections -- Symbols -- Global vs. Local Symbols -- Weak Symbols -- Symbol Versions -- Mapping Symbols -- The Dynamic Section and Dynamic Loading -- Dependency Loading (NEEDED) -- Program Relocations -- Static Relocations -- Dynamic Relocations -- The Global Offset Table

(GOT) -- The Procedure Linkage Table (PLT) -- The ELF Program Initialization and Termination Sections -- Initialization and Termination Order -- Thread- Local Storage -- The Local- Exec TLS Access Model -- The Initial- Exec TLS Access Model -- The General- Dynamic TLS Access Model -- The Local- Dynamic TLS Access Model -- Chapter 3 OS Fundamentals -- OS Architecture Overview -- User Mode vs. Kernel Mode -- Processes -- System Calls -- Objects and Handles -- Threads -- Process Memory Management -- Memory Pages -- Memory Protections -- Anonymous and Memory- Mapped Memory -- Memory-Mapped Files and Modules -- Address Space Layout Randomization -- Stack Implementations -- Shared Memory -- Chapter 4 The Arm Architecture -- Architectures and Profiles -- The Armv8- A Architecture -- Exception Levels -- Armv8- A TrustZone Extension -- Exception Level Changes -- Armv8- A Execution States -- The AArch64 Execution State -- The A64 Instruction Set -- AArch64 Registers -- The Program Counter -- The Stack Pointer -- The Zero Register -- The Link Register -- The Frame Pointer -- The Platform Register (x18) -- The Intraprocedural Call Registers -- SIMD and Floating- Point Registers -- System Registers -- PSTATE -- The AArch32 Execution State -- A32 and T32 Instruction Sets -- The A32 Instruction Set -- The T32 Instruction Set -- Switching Between Instruction Sets -- AArch32 Registers -- The Program Counter -- The Stack Pointer -- The Frame Pointer -- The Link Register -- The Intraprocedural Call Register (IP, r12) -- The Current Program Status Register -- The Application Program Status Register -- The Execution State Registers -- The Instruction Set State Register -- The IT Block State Register (ITSTATE) -- Endianness state -- Mode and Exception Mask Bits -- Chapter 5 Data Processing Instructions -- Shift and Rotate Operations -- Logical Shift Left -- Logical Shift Right -- Arithmetic Shift Right -- Rotate Right -- Rotate Right with Extend -- Instruction Forms -- Shift by a Constant Immediate Form -- Shift by Register Form -- Bitfield Manipulation Operations -- Bitfield Move -- Sign- and Zero- Extend Operations -- Bitfield Extract and Insert -- Logical Operations -- Bitwise AND The TST Instruction -- Bitwise Bit Clear -- Bitwise OR Bitwise OR NOT Bitwise Exclusive OR The TEQ instruction Exclusive OR NOT Arithmetic Operations Addition and Subtraction -- Reverse Subtract -- Compare CMP Instruction Operation Behavior -- Multiplication Operations -- Multiplications on A64 -- Multiplications on A32/T32 -- Least Significant Word Multiplications -- Most Significant Word Multiplications -- Halfword Multiplications -- Vector (Dual) Multiplications -- Long (64- Bit) Multiplications -- Division Operations -- Move Operations -- Move Constant Immediate -- Move Immediate and MOVT on A32/T32 -- Move Immediate, MOVZ, and MOVK on A64 -- Move Register -- Move with NOT -- Chapter 6 Memory Access Instructions -- Instructions Overview -- Addressing Modes and Offset Forms -- Offset Addressing -- Constant Immediate Offset -- Register Offsets -- Pre- Indexed Mode -- Pre- Indexed Mode Example -- Post-Indexed Addressing -- Post- Indexed Addressing Example -- Literal (PC- Relative) Addressing -- Loading Constants -- Loading an Address into a Register -- Load and Store Instructions -- Load and Store Word or Doubleword -- Load and Store Halfword or Byte -- Example Using Load and Store -- Load and Store Multiple (A32) -- Example for STM and LDM -- A More Complicated Example Using STM and LDM -- Load and Store Pair (A64) -- Chapter 7 Conditional Execution -- Conditional Execution Overview -- Conditional Codes -- The NZCV Condition Flags -- Signed vs. Unsigned Integer Overflows -- Condition Codes -- Conditional Instructions -- The If- Then (IT) Instruction in Thumb -- Flag- Setting Instructions -- The Instruction "S" Suffix -- The S Suffix

on Add and Subtract Instructions -- The S Suffix on Logical Shift Instructions -- The S Suffix on Multiply Instructions -- The S Suffix on Other Instructions -- Test and Comparison Instructions -- Compare (CMP) -- Compare Negative (CMN) -- Test Bits (TST) -- Test Equality (TEQ) -- Conditional Select Instructions -- Conditional Comparison Instructions -- Boolean AND Conditionals Using CCMP -- Boolean OR Conditionals Using CCMP -- Chapter 8 Control Flow -- Branch Instructions -- Conditional Branches and Loops -- Test and Compare Branches -- Table Branches (T32) -- Branch and Exchange -- Subroutine Branches -- Functions and Subroutines -- The Procedure Call Standard -- Volatile vs. Nonvolatile Registers -- Arguments and Return Values -- Passing Larger Values -- Leaf and Nonleaf Functions -- Leaf Functions -- Nonleaf Functions -- Prologue and Epilogue -- Part II Reverse Engineering -- Chapter 9 Arm Environments -- Arm Boards -- Emulation with QEMU -- QEMU User- Mode Emulation -- QEMU Full- System Emulation -- Firmware Emulation -- Chapter 10 Static Analysis -- Static Analysis Tools -- Command- Line Tools 322 Disassemblers and Decompilers -- Binary Ninja Cloud -- Call- By-Reference Example -- Control Flow Analysis -- Main Function -- Subroutine -- Converting to char if Statement -- Quotient Division for Loop -- Analyzing an Algorithm -- Chapter 11 Dynamic Analysis -- Command- Line Debugging -- GDB Commands -- GDB Multiuser -- GDB Extension: GEF -- Installation -- Interface -- Useful GEF Commands -- Examine Memory -- Watch Memory Regions -- Vulnerability Analyzers -- checksec -- Radare2 -- Debugging -- Remote Debugging -- Radare2 -- IDA Pro -- Debugging a Memory Corruption -- Debugging a Process with GDB -- Chapter 12 Reversing arm64 macOS Malware -- Background -- macOS arm64 Binaries macOS Hello World (arm64) -- Hunting for Malicious arm64 Binaries -- Analyzing arm64 Malware -- Anti- Analysis Techniques -- Anti-Debugging Logic (via ptrace) -- Anti- Debugging Logic (via sysctl) -- Anti- VM Logic (via SIP Status and the Detection of VM Artifacts) -- Conclusion -- Index.

Sommario/riassunto

Finding and mitigating security vulnerabilities in Arm devices is the next critical internet security frontier-Arm processors are already in use by more than 90% of all mobile devices, billions of Internet of Things (IoT) devices, and a growing number of current laptops from companies including Microsoft, Lenovo, and Apple. Written by a leading expert on Arm security, Blue Fox: Arm Assembly Internals and Reverse Engineering introduces readers to modern Armv8-A instruction sets and the process of reverse-engineering Arm binaries for security research and defensive purposes. Divided into two sections, the book first provides an overview of the ELF file format and OS internals, followed by Arm architecture fundamentals, and a deep-dive into the A32 and A64 instruction sets. Section Two delves into the process of reverse-engineering itself: setting up an Arm environment, an introduction to static and dynamic analysis tools, and the process of extracting and emulating firmware for analysis. The last chapter provides the reader a glimpse into macOS malware analysis of binaries compiled for the Arm-based M1 SoC. Throughout the book, the reader is given an extensive understanding of Arm instructions and control-flow patterns essential for reverse engineering software compiled for the Arm architecture. Providing an in-depth introduction into reverse-engineering for engineers and security researchers alike, this book: *Offers an introduction to the Arm architecture, covering both AArch32 and AArch64 instruction set states, as well as ELF file format internals *Presents in-depth information on Arm assembly internals for reverse engineers analyzing malware and auditing software for security

vulnerabilities, as well as for developers seeking detailed knowledge of the Arm assembly language *Covers the A32/T32 and A64 instruction sets supported by the Armv8-A architecture with a detailed overview of the most common instructions and control flow patterns *Introduces known reverse engineering tools used for static and dynamic binary analysis *Describes the process of disassembling and debugging Arm binaries on Linux, and using common disassembly and debugging tools. Blue Fox: Arm Assembly Internals and Reverse Engineering is a vital resource for security researchers and reverse engineers who analyze software applications for Arm-based devices at the assembly level.