Research on Hidden Instruction Mining Technology of ARM Processor in Android Devices

Wu Zheng*[1], Chai Song[1], Chen Xi[1], Li Guo[2]

1Southwest Minzu University, Chengdu, Sichuan, 610225, China
2University of Electronic Science and Technology of China, Chengdu, Sichuan, 610054, China

Abstract: With the proliferation of the Arm architecture in Android devices, the instruction-level security of its processors faces severe challenges, particularly as the existence of underlying hidden instructions may introduce unknown backdoors and attack surfaces. Addressing the lack of current research on the instruction-level security of Arm processors in Android devices and the missing detection tools, this paper proposes an efficient mining method for hidden instructions of Arm processors targeting Android devices. The core of this method lies in: firstly, generating a candidate set of undefined instructions not defined in the Armv8-A Architecture Reference Manual based on existing Arm disassemblers and instruction encoding spaces; secondly, designing and implementing a lightweight assembly testing framework by combining page-based execution technology and Android Debug Bridge (ADB), achieving large-scale automated verification of the execution results of underlying instructions on Android devices. To verify the effectiveness of the method, this paper uses the Redmi Note 9 equipped with a Dimensity 800U processor as the experimental platform and conducts mining on the A64 and A32 instruction sets. Experimental results show that the method successfully discovered 1024 hidden instructions in the A64 instruction set (while traditional methods had no findings in this instruction set). In the A32 instruction set, a large number of hidden instructions exist in both Android devices and general Arm processors, reaching proportions of 45.93% and 23.16%, respectively, demonstrating significant differences in instruction implementation between the two. This study not only, for the first time, provides a systematic hidden instruction mining scheme for the A64 and A32 instruction sets of Arm processors in Android devices, but also provides critical data support and a methodological foundation for subsequent instruction-level vulnerability analysis and processor security hardening.

Keywords: Android devices; Arm processors; ADB; Hidden instructions

DOI: 10.62639/sspjinss01.20260301

1. Introduction

(1) Research background and problem statement

In recent years, with the continuous evolution of semiconductor technology and the rapid development of microprocessor architecture, embedded processors have been widely used in all kinds of intelligent devices. According to IC insights, the global chip shipment in 2020 was about 318.1 billion, of which embedded processors accounted for about one third[1]. In the field of Android intelligent devices, arm architecture occupies a dominant position with its advantages of flexibility, low power consumption and scalability, and its market share is as high as 95%. From Xiaomi's Redmi series to Huawei's mate series, from MediaTek's Dimensity series to Qualcomm's Xiaolong series, these mainstream mobile processors are designed based on ARM architecture and bear the core functions of system operation control and auxiliary processing.

(Manuscript NO.: JINSS-26-1-62014)

Corresponding Author

Wu Zheng (1998-), male, of Dong ethnicity, native of Tianzhu, Guizhou Province. Currently a postgraduate student pursuing a Master's degree. Research direction: new generation electronic information technology.

However, while the embedded processor brings powerful functions, it also faces increasingly serious security threats. Research shows that there are often unpublished dark instructions (hidden instructions) in the hardware structure and instruction set of embedded processors. Although these instructions can be recognized, parsed and executed by the processor, they are not disclosed in the official documents of the manufacturer[2]. There are various reasons for the generation of dark instructions: some are due to accidental omissions in the chip design stage, some are left over from the internal test instructions of the manufacturer, and even there is the possibility of malicious implantation. Regardless of the source, the logic vulnerabilities of these instructions may be maliciously exploited by attackers, causing serious hardware security risks[3].

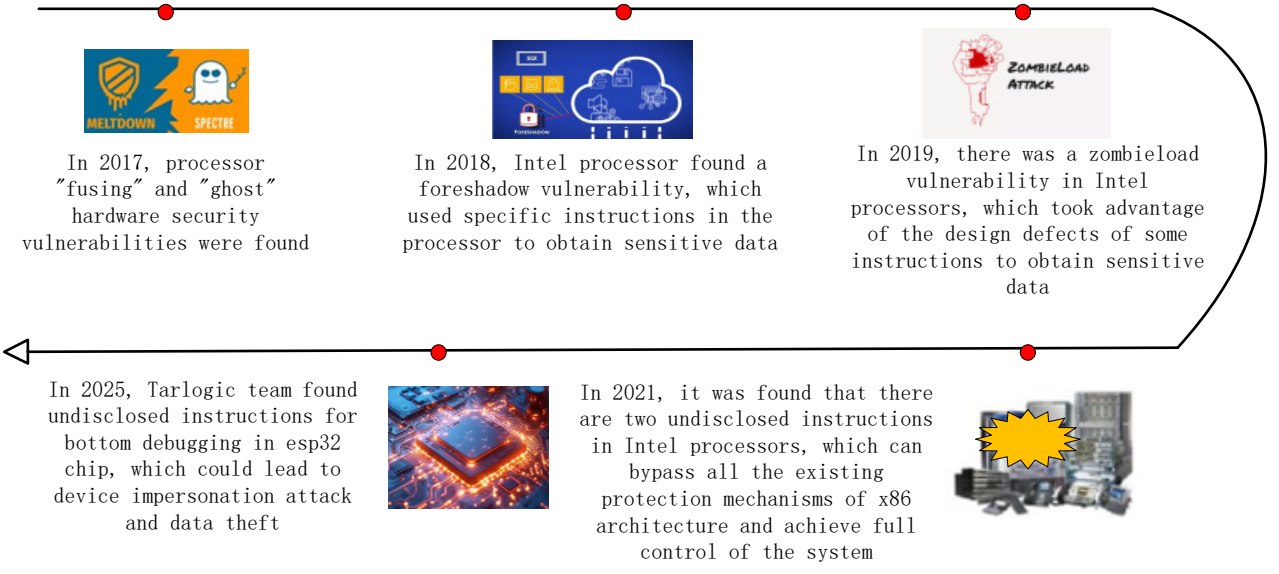

Figure 1: Processor security events in recent year

As shown in Figure 1, in recent years, processor security events have emerged in endlessly. In 2017, the meltdown[4] and spectre[5] vulnerabilities disclosed by Google's security team affected billions of devices worldwide; In 2018, the foreshadow vulnerability was exposed in Intel processors. Attackers can bypass the security isolation mechanism to obtain sensitive data[6]; In 2019, the discovery of zombieload vulnerability further revealed that processor instruction design defects could lead to side channel attacks[7]. It is particularly noteworthy that in 2021, Russian security experts found that there were two undisclosed instructions in Intel processors, which could bypass all existing protection mechanisms of x86 architecture and achieve complete control of the system; In 2025, tarlogic team found undisclosed instructions for bottom debugging in esp32 chip, which could lead to device impersonation attack and data theft[8]. These events fully show that the hardware security of embedded processors is facing unprecedented challenges.

To sum up, it has important theoretical value and practical significance to carry out the research on dark instruction mining for embedded processors of Android devices and reveal the existence and potential risks of hidden instructions for improving the hardware security of devices and ensuring the safe and stable operation of intelligent devices.

(2) Current research status and challenges

The research on instruction set security of processors has not been paid enough attention for a long time. Most of the relevant literatures are published by processor manufacturers. It is very difficult for non manufacturer researchers to conduct instruction level security testing[9]. With the increasing complexity of instruction architecture, the challenges in this research field become increasingly prominent.

With the popularity of RISC architecture in the embedded field, the related research is gradually increasing. In

2020, Rens et al. Developed iSCAN tool for RISC architecture to identify dark instructions by analyzing the execution of instructions one by one and comparing them with the public instruction set[10]. In the same year, Fredrik et al. Launched the armshaker tool for the armv8-a architecture, using an external disassembler to assist in dark instruction detection[11]. In 2021, Wang et al. Proposed a conformance testing method to detect abnormal behavior by executing all undefined instructions on the processor under test. In 2025, Cheng proposed the dark instruction mining method of ARM processor in user mode and kernel mode, realizing precise disassembly and privileged instruction detection[12].

Although the general processor dark instruction mining has made some progress, the systematic research on the specific operating environment of Android devices is still in its infancy, facing the following main challenges:

Challenge 1: research gaps and misplaced concerns. Existing research mostly focuses on X86 and arm based general-purpose processors, and has not yet carried out systematic dark instruction mining for Android devices. The Android platform has been researched on the manufacturer's secret code and the hidden FastBoot command, but these mainly focus on the application layer or system interface layer, and have not yet gone deep into the processor's instruction set level.

Challenge 2: it is difficult to adapt the test environment. The general dark instruction mining tool is mainly designed for bare metal environment or simulator, without considering the particularity of Android system. How to build a test environment that can safely execute undefined instructions while retaining the functions of Android system is the key technical problem of dark instruction mining for Android devices.

Challenge 3: balance mining efficiency and accuracy. The existing methods, such as Armshaker, adopt the linear search method, and the search efficiency of the instruction space is extremely low. How to improve the mining efficiency while ensuring accuracy still needs further exploration.

To sum up, the research on processor dark instruction mining under Android devices is still in the exploratory stage, lacking systematic research methods and mature technical solutions, and in urgent need of new technical paths and practical experience.

(3) Main contributions

In view of the above problems and challenges, this paper proposes a set of dark instruction mining methods for Android device arm processor. The main contributions include the following three aspects:

First, an undefined instruction generation method based on disassembler optimization is proposed. This paper uses the existing arm disassembler to generate the undefined instruction candidate set in the armv8-a architecture reference manual by exhausting the instruction encoding space. At the same time, the instruction type decoding module of the disassembler is optimized to realize multi-level instruction type decoding, which improves the accuracy and efficiency of dark instruction classification and lays the data foundation for the subsequent statistical analysis of dark instructions

Secondly, an efficient large-scale dark instruction mining platform in ADB environment is constructed. The connection between the upper computer and the Android device is established through the Android debugging bridge (ADB), and the assembly test framework and undefined instruction set are sent to the internal terminal environment of the device for execution. The experimental results show that the method successfully finds 1024 hidden instructions in the A64 instruction set on the Redmi Note 9 mobile phone equipped with Dimensity 800U processor (not found in the instruction set by traditional methods); In the A32 instruction set, Android devices and general-purpose ARM processors have a large number of hidden instructions, accounting for 45.93% and 23.16% respectively, showing a significant difference in the implementation of instructions between the two.

2. Construction of Armv8-a Instruction Space and Undefined Instruction Space

(1) Basic information of armv8-a instruction space

Armv8-a architecture is a 64 bit instruction set architecture launched by arm company, which is widely used in embedded processors of Android intelligent devices. Compared with armv7, armv8-a introduces two execution states: aarch64 and aarch32. Aarch64 is a 64 bit execution state and supports A64 instruction set; Aarch32 is a 32-bit execution state and is compatible with A32 and T32 instruction sets. The research scope of this paper focuses on the A64 and A32 instruction sets, excluding the T32 instruction set.

Armv8-a adopts the reduced instruction set computer (RISC) architecture. Compared with the complex instruction set computer (CISC), it has the characteristics of regular instruction format and fixed instruction length. Both A64 and A32 instruction sets use 32-bit fixed instruction length, which means that their instruction space size is 4.3 billion instructions. Each bit in the instruction code has a specific meaning, including fields such as opcode, register address, and immediate value. For example, the subtract instruction sub (immediate) in the A64 instruction set uses 32-bit encoding, which contains fields such as the 5-bit destination register Rd, the 5-bit source register RN, and the 12 bit immediate imm12; However, due to the existence of condition codes and historical compatibility requirements, the encoding formats of similar instructions in A32 instruction set are different, and the number of bits of register address is 4.

(2) Construction of undefined instruction space

In the 4.3 billion instruction space of armv8-a architecture, not all codes are assigned corresponding functions. According to the arm architecture reference manual, there are a large number of coding regions with unallocated functions in the instruction space, which are called undefined instruction space. Undefined instructions can be divided into three categories: undefined (undefined instruction exception is generated during execution), unpredictable (execution result is unpredictable) and constrained unpredictable (execution result is limited by a series of possible results). This paper focuses on the undefined class of instructions as a candidate set for dark instruction mining.

In order to construct undefined instruction space, this paper uses a filter method based on disassembler. The specific steps are as follows: first, use the 32-bit binary code 0x00000000 as the starting point and 0xFFFFFF as the end point to traverse the entire instruction space linearly. In order to facilitate management and observation, 4.3 billion instructions are divided into 256 rounds, and each round contains 16777216 instructions. Secondly, the instruction code generated in each round is input into the armv8-a disassembler for disassembly and analysis. If the disassembly is successful, it indicates that the instruction is defined in the architecture reference manual and is excluded; If disassembly fails, it indicates that the instruction is an undefined instruction, and it is recorded in the undefined instruction set.

Because direct storage of all undefined instructions will occupy a lot of storage space and is not conducive to subsequent processing, this paper uses the interval storage method: record the continuous undefined instruction blocks in the form of left closed right open interval, that is, record the start address and end address of each continuous undefined instruction block. This storage method significantly reduces the data size and provides efficient instruction input for subsequent dark instruction mining on Android devices.

3. Design of Hidden Instruction Mining Scheme for Android Devices Based on Page Based Execution Technology

(1) Test program based on page based execution technology

In the process of mining dark instructions, executing undefined instructions may have unpredictable side

effects, including memory corruption, abnormal register modification, system state disorder and even processor crash. Therefore, effective isolation and protection measures must be taken. This paper uses page based execution technology, using the memory management unit (MMU) and paging mechanism of the operating system to create an isolated memory execution environment for the instructions to be tested.

The test program based on page based includes three core steps: (1) in the initialization phase, register the signal processing function to capture sigill exceptions, apply for independent memory pages through mmap system call, and write the template code including register state saving and recovery, and placeholder for the instruction to be tested; (2) In the execution phase, it traverses the undefined instruction set, writes each instruction to the executable memory page, flushes the instruction cache and jumps to execution; (3) In the judgment stage, judge according to the signal value returned by the signal processing program: if sigill signal is received, it indicates that the instruction triggers an illegal instruction exception, which is an illegal instruction; If no abnormal signal is received, it indicates that the instruction can be executed, which is a hidden instruction.

This technology takes over the exception through the signal processing mechanism, so that the test program can recover from the execution of illegal instructions and continue to test the next instruction, so as to realize the safe and automatic batch test on the real processor.

(2) Assembly test framework of ARM processor

In order to ensure program stability and test efficiency in large-scale instruction testing, this paper designs and implements a complete assembly testing framework, including fork testing mechanism and batch testing framework.

Fork testing mechanism uses the process isolation feature of Linux operating system to create independent sub processes for each instruction to be tested. After the main process calls fork to create a sub process, the sub process is responsible for executing the instruction test program based on page based; The parent process calls waitpid to monitor the status of the child process, and obtains the execution result by analyzing the exit status of the child process. No matter what kind of exception (including crash) the child process produces due to the execution of instructions, it will not affect the stable operation of the parent process, realizing the complete isolation of instruction execution.

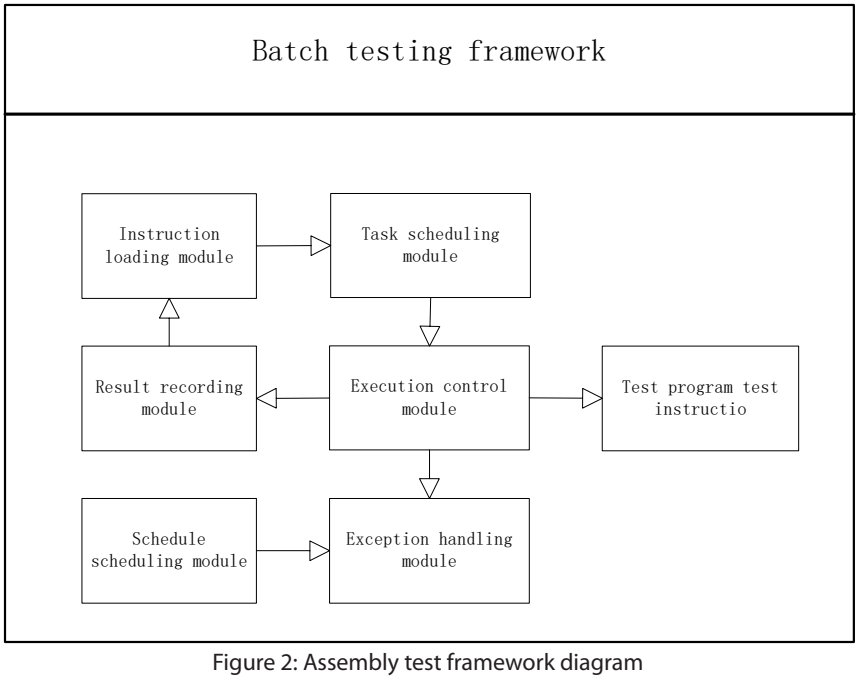

Figure 2: Assembly test framework diagram

As shown in Figure 2, the batch test framework consists of six core modules: (1) instruction loading module, which is responsible for loading undefined instruction sets and initializing test parameters; (2) The task scheduling module supports two modes: single process serial mode and multi process parallel mode to realize the reasonable allocation of test tasks; (3) Execute the control module, call fork test function to execute specific instruction test, and determine whether the instruction is a hidden instruction; (4) The result recording module records the hidden instructions in the form of intervals; (5) The progress monitoring module displays the test progress and statistical information in real time; (6) The exception handling module captures and handles all kinds of exceptions in the test process to ensure the stability of large-scale test for a long time.

In addition, the framework implements undefined instruction interval partition script and automatic scheduling script. The former realizes multi process load balancing, while the latter automatically scans the input file, starts multiple worker processes for parallel testing, collects and combines the results and generates statistical reports, and finally forms a complete hidden instruction mining tool chain.

(3) Design of hidden instruction mining scheme for Android devices

Aiming at the particularity of Android devices, this paper designs a hidden instruction mining scheme based on ADB (Android debug bridge). ADB is a general command line debugging tool provided by the Android SDK. It uses a three-tier architecture of client server daemon to support command interaction and data transmission between the upper computer and Android devices.

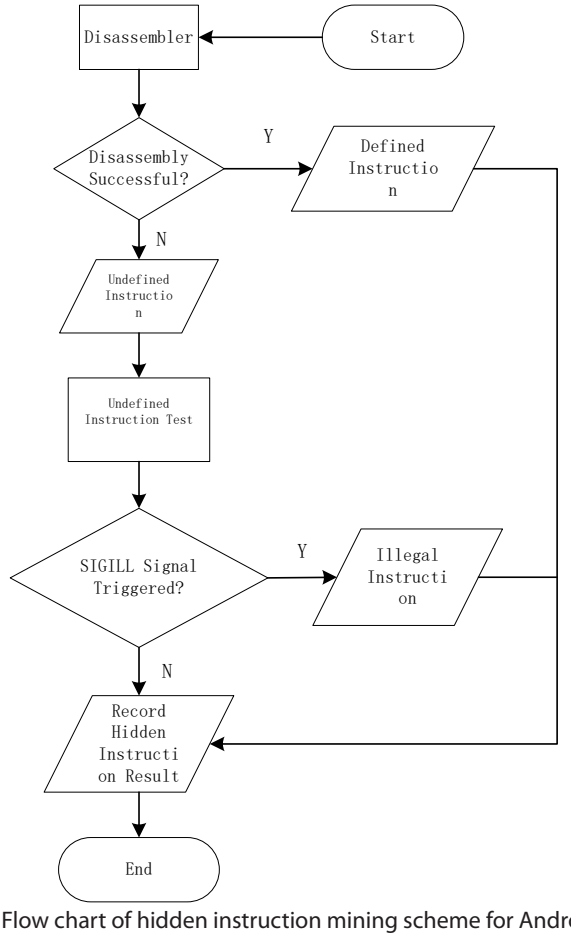

Figure 3: Flow chart of hidden instruction mining scheme for Android device

Figure 3 is the flow chart of hidden instruction mining scheme. The specific process is as follows:: (1) on the upper computer side, the undefined instruction space is constructed by using the armv8-a disassembler, and

the undefined instruction set is divided into multiple sub task files by using the interval segmentation script; (2) Establish the connection between the upper computer and Android device through ADB, and use ADB push command to push the assembly test framework executable file, undefined instruction subtask file and automatic scheduling script to the data directory of Android device; (3) Enter the Linux shell environment inside the Android device through the ADB shell, execute the automatic scheduling script and start the batch test; (4) During the test, the ADB connection maintains the monitoring status and checks the test log and progress in real time; (5) After the test, the ADB pull command is used to pull the hidden instruction result file and statistical report from the device to the upper computer for subsequent analysis.

This scheme has significant technical advantages: firstly, it does not need to obtain the root permission of Android device, and can execute the low-level instruction test only through USB debugging mode, which reduces the test threshold and ensures the security of the device; Secondly, it supports the upper computer to manage multiple test equipment, and realizes large-scale concurrent testing through ADB parallel control, which significantly improves the mining efficiency; In addition, the stable data transmission capability provided by ADB ensures the reliability of test instruction set push and result file pull, avoiding data loss or damage; Finally, through script automation, the whole process closed-loop from task distribution, instruction execution to result collection is realized, which greatly reduces manual intervention, and provides complete technical support for the efficient and large-scale mining of hidden instructions in arm processor of Android device.

4. Experiment of Hidden Instruction Mining in Arm Processor of Android Device

(1) Experimental platform and environment configuration

In order to verify the effectiveness of the hidden instruction mining method for Android devices based on ADB proposed in this paper, an experimental platform was built on real Android devices for verification. The experiment uses Redmi Note 9 mobile phone as the test equipment, which is equipped with MediaTek Dimensity 800U processor and is designed based on armv8-a architecture. It supports two execution states, aarch64 and aarch32, and can fully cover the test requirements of A64 and A32 instruction sets.

Table 1: Experimental environment configuration

Experimental environment	Model or version
Operating System	Windows11 64
Android devices	Redmi Note 9
Master chip	Dimensity 800U
Computer CPU	AMD Ryzen 7 5700

The experimental environment configuration is shown in Table 1. the upper computer uses windows11 operating system processor as AMD Ryzen 75700 computer, and establishes debugging connection with Android device through ADB tool. The test device Redmi Note 9 runs Android 11 system and is connected to the upper computer through USB debugging mode. During the experiment, it is not necessary to obtain the root authority of the device, and all test operations can be completed only by relying on the standard debugging interface of ADB, which reduces the test threshold and ensures the safety of the device.

The undefined instruction set constructed by armv8-a disassembler includes 2873902386 undefined instructions in A64 instruction set and 1204724480 undefined instructions in A32 instruction set. During the test, the assembly test framework and undefined instructions are pushed to the device through ADB, and the results are sent back to the upper computer for analysis after the device executes the test locally.

(2) A64 instruction set hidden instruction mining results

In the A64 instruction set test, the A64 undefined instruction set and the corresponding aarch64 assembly test

framework are pushed to the Redmi Note 9 device based on the ADB mode, and the batch test is performed through the automated scheduling script. The test process completely traversed 2873902386 A64 undefined instructions.

The experimental results show that there are 1024 hidden instructions in the A64 instruction set on the Dimensity 800U processor. This result does not meet the expectation of the armv8-a architecture specification, because according to the architecture reference manual, all undefined instructions in the A64 instruction set should trigger illegal instruction exceptions. However, the actual test found that some undefined instructions were given executable functions at the processor hardware level.

This discovery reveals the differentiation of arm architecture in the implementation of specific processors. Although arm has formulated a unified architecture specification, chip manufacturers may extend the functions of some undefined instruction codes or retain test interfaces during specific implementation. This feature of "unified architecture and diverse implementation methods" is the typical feature of embedded processor ecology. The 1024 hidden instructions of Dimensity 800U processor in the A64 instruction set indicate that MediaTek may have specific hardware debugging, performance optimization or vendor defined functions based on its compatibility with the armv8-a architecture.

(3) A32 instruction set hidden instruction mining results

In the A32 instruction set test, the same ADB test scheme is used to push the A32 undefined instruction set and the corresponding aarch32 assembly test framework to the Redmi Note 9 device for execution. The test process traverses a total of 1204724480 A32 undefined instructions. The mining results are shown in Table 2.

Table 2: A32 instruction set hidden instruction mining results

Instruction set	undefined instructions	hidden instructions	Hidden instruction percentage
A32	1,204,724,480	553,776,384	45.93%

Experimental data show that there are a large number of hidden instructions that can be executed but do not trigger sigill exceptions in the A32 instruction set, and the number reaches 553million, accounting for 45.93% of the undefined instruction space in A32. This result is significantly different from the 23.16% obtained by traditional arm processor, indicating that there are great differences in the implementation of A32 instruction set among different processor models. Dimensity 800U processor is more relaxed in the processing of A32 undefined instructions, and a large number of instruction codes not defined in the architecture manual are implemented as executable functions by hardware.

This discovery has important security significance: first, A32 hidden instructions widely existing in Android devices may become a potential attack surface, and attackers can use these undisclosed instructions to bypass the existing security mechanism; Secondly, the differences in instruction implementation among different processor manufacturers increase the security complexity of the Android ecosystem, and the same set of applications may face different underlying instruction risks on different devices.

5. Conclusion and Future Work

(1) Summary of work

The main work of this paper is as follows: first, the coding characteristics of armv8-a instruction set architecture are deeply analyzed, the instruction space division of A64 and A32 instruction sets is clarified, and a complete undefined instruction set is constructed based on the anti assembler optimization method, which provides a data basis for subsequent mining work. Secondly, an assembly testing framework based on page based execution technology is designed, and fork testing mechanism and batch testing architecture are introduced to solve the problems of isolation protection and execution efficiency in large-scale instruction testing. Thirdly, aiming at the

particularity of Android devices, a hidden instruction mining scheme based on ADB is proposed, which realizes the technical path of batch testing of underlying instructions on real Android devices without root permission. Fourth, the experimental verification was completed on the processor of Redmi Note 9 (Dimensity 800U) armv8-a architecture. The experiment found that there were 1024 hidden instructions (Dimensity 800U) in the A64 instruction set, which broke the expectation of the architecture specification; There are a large number of hidden instructions in the A32 instruction set, and the traditional arm processor models differ significantly (accounting for 45.93% and 23.16% respectively). The work of this paper is the first to realize the systematic hidden instruction mining of the A64 and A32 instruction sets of ARM processors in Android devices, which provides an effective methodological support and real data basis for processor instruction level security assessment.

(2) Future research perspectives

The hidden instruction mining method for Android devices based on ADB proposed in this paper has been effectively verified on A64 and A32 instruction sets, but there is still room for further development. Future research can be carried out in the following two directions: first, the scope of research can be extended to the T32 instruction set. In this experiment, we found that there were a small number of hidden instructions in the T32 instruction set (32-bit T32 accounted for 0.16%), but due to the stability of the test environment, it could not be fully mined. In the future, we can optimize the test framework to complete the comprehensive test of the T32 instruction set; Second, the behavior analysis and function reversal of hidden instructions can be carried out in depth. This paper focuses on the mining of the "existence" of hidden instructions, and has not analyzed the specific functions of hundreds of millions of hidden instructions that have been found. In the future, the actual behavior, trigger conditions and potential security impact of these hidden instructions can be revealed by combining dynamic tracking, hardware simulation and formal verification technologies, so as to provide more accurate support for the underlying security reinforcement of Android devices.

References

[1] W IC Insights. Total Microprocessor Sales to Edge Slightly Higher in 2020[EB/OL]. https//www.icinsights.com/news/bulletins/Total-Microprocessor-Sal es-To-Edge-Slightly-Higher-In-2020/.

[2] Ermolov M, Sklyarov D, Goryachy M. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors[J]. Journal of Computer Virology and Hacking Techniques, 2023, 19(3): 351-365.

[3] Amit N, Tsafrir D, Schuster A, et al. Virtual CPU validation[C]. Proceedings of the 25th Symposium on Operating Systems Principles. 2015: 311-327.

[4] Moritz Lipp et al. "Meltdown". In: arXiv preprint arXiv:1801.01207 (2018).

[5] Paul Kocher et al. "Spectre attacks: Exploiting speculative execution". In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE. 2019, pp. 1–19.

[6] Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution[C]. 27th USENIX Security Symposium (USENIX Security 18). 2018: 991–1008.

[7] Schwarz M, Lipp M, Moghimi D, et al. ZombieLoad: Cross-privilege-boundary data sampling[C]. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 753-768.

[8] M. Tarascó and A. Blanco, "Undocumented commands in ESP32: A security analysis," presented at RootedCON, Madrid, Spain, Mar. 2025.

[9] Wu H, Wei Q, Wu Z. Research on CPU instruction security[C]. 2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC). IEEE, 2020: 233-239.

[10]Dofferhoff R, G ebel M, Rietveld K, et al. iScanU: A Portable Scanner for Undocumented Instructions on RISC Processors[C]. 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2020: 306-317.

[11]Strupe F, Kumar R. Uncovering Hidden Instructions in Armv8-A Implementations[C]. Workshop on Hardware and Architectural Support for Security and Privacy. 2020.

[12]Wang Y, Liu P, Wang W, et al. On a consistency testing model and strategy for revealing RISC processor's dark instructions and vulnerabilities[J]. IEEE Transactions on Computers, 2021, 71(7): 1586-1597.