Discover Internet of Things

Review

A survey on IoT & embedded device firmware security: architecture, extraction techniques, and vulnerability analysis frameworks

Shahid Ul Haq[1] · Yashwant Singh[1] · Amit Sharma[2] · Rahul Gupta[2] · Dipak Gupta[2]

Received: 5 May 2023 / Accepted: 25 September 2023
Published online: 31 October 2023

Abstract
IoT and Embedded devices grow at an exponential rate, however, without adequate security mechanisms in place. One of the key challenges in the cyber world is the security of these devices. One of the main reasons that these devices are active targets for large-scale cyber-attacks is a lack of security standards and thorough testing by manufacturers. Manufacturer-specific operating systems or firmware of various architectures and characteristics are typically included with these devices. However, due to a lack of security testing and/or late patching, the underlying firmware or operating systems are vulnerable to numerous types of vulnerabilities. Reverse engineering and in-depth research of the firmware is required to detect the vulnerabilities. In this paper, we've delved into various aspects of IoT and embedded devices. This includes a comprehensive survey on the architecture of firmware, techniques for firmware extraction, and state-of-the-art vulnerability analysis frameworks for the detection of vulnerabilities using various approaches like static, dynamic, and hybrid approaches. Furthermore, we've scrutinized the challenges of existing vulnerability analysis frameworks and proposed a novel framework to address these issues.

Keywords IoT · Firmware · Embedded · Architecture · Extraction · Vulnerability · Analysis · Frameworks

1 Introduction

The Internet of Things (IoT) has experienced a swift surge in adoption, encompassing a diverse array of applications from personal health care and environmental monitoring to home automation, smart mobility, and Industry 4.0. Consequently, there has been a notable increase in the deployment of IoT devices in both public and private settings, becoming increasingly prevalent in households. With this widespread integration comes an escalated vulnerability to cybersecurity threats, necessitating measures to avert risks such as data breaches, denial-of-service attacks, and unauthorized network access. Addressing these challenges is crucial to ensure the secure and reliable operation of IoT systems across various applications.

The amount of recent assaults on embedded & IoT systems demonstrates their security risk. The Mirai botnet, for example, hijacked millions of IoT devices and coordinated them to conduct a distributed denial of service (DDoS) attack against several domain name system (DNS) servers, taking hundreds of thousands of websites offline throughout the world [1]. The Reaper malware, a more complex version of the Mirai, was originally disclosed in 2016 and specifically aimed at IoT

✉ Yashwant Singh, yashwant.csit@cujammu.ac.in; Shahid Ul Haq, shahidulhaq.lone@gmail.com; Amit Sharma, amitsharma@gov.in; Rahul Gupta, rahul.gupta@hqr.drdo.in; Dipak Gupta, dipak.gupta@hqr.drdo.in | [1]Department of Computer Science & Information Technology, Central University of Jammu, Jammu, J&K 181143, India. [2]Office of the Advisor (Cyber), Department of Defence (R&D), Ministry of Defence, DRDO, Govt. of India, Delhi, India.

devices having certain vulnerabilities rather than just credentials [2]. An Advanced Persistent Threat (APT) known as Black Energy caused a blackout by gaining supervisory control over various operating stations in 2014 [3]. Various other countries have also witnessed similar threats. Intruders obtaining control of more than 50 power plants, for example, could potentially compromise the electrical supply to 93 million Americans [4]. These real-world attacks show how IoT and embedded systems in key infrastructures can be severely harmed. Unfortunately, many commercial IoT goods do not often include sufficient security procedures, and as a result, they can be the target of or even the source of a variety of security threats. IoT and Embedded devices share various technical characteristics which include system architecture based on ARM or MIPS CPUs, Ethernet, Wi-Fi, or Bluetooth-based connectivity, and On-chip debugging interfaces such as UART, JTAG, I2C, or SPI. Most of these devices are controlled by vendor-specific software which is rarely updated to fix security problems. For a thorough security examination of these devices, proper identification of key technological aspects is critical. Furthermore, due to the diverse and non-standardized nature of the hardware and software features of embedded and IoT devices, security evaluation provides a number of issues. Security evaluation of IoT devices has two main aspects: network-based evaluation and firmware-based evaluation. In this work, we have mainly focused on the firmware part. In order to perform firmware security evaluation researchers have to get hold of the firmware and perform reverse engineering to reveal the vulnerabilities in it.

One of the most common causes of attacks on embedded systems has been identified as software vulnerabilities, and new vulnerabilities are discovered on a regular basis. Most of the popular binaries that are reused in software projects and firmware images are usually found vulnerable due to lack of security updates and due to this most of the embedded systems become implicitly vulnerable at an early stage. Several recent papers have also emphasised the importance of the analysis of firmware images [5]. Furthermore, in [6] Cui et al. claim that the third-party libraries used in firmware updates have been found to contain some of the famous vulnerabilities for years. They further reveal that about 80.4% of manufacturers distribute firmware with known flaws. As embedded systems manage critical components, compromising them could result in massive public system failures as well as serious security and safety implications, on a national or perhaps at a global scale. For example, 18 zero-day vulnerabilities were discovered in a Foscam IP camera, which includes insecure credentials, heap or stack buffer overflow, and command injection vulnerabilities [7].

Obsolete system architectures are also one of the main reasons for embedded systems being frequently vulnerable to attacks. Second, embedded systems' internet connectivity, integration, and platform compatibility requirements make them more vulnerable to cyberattacks and exploitation. Finally, standard security techniques and traditional solutions, such as Intrusion detection or prevention systems cannot be used because these devices have limited computing power and memory. As a result, attackers take advantage of these flaws and create tailored malware for embedded systems and IoT devices.

Vulnerabilities in software can be found in both source and binary code. The latter techniques [8] use the source code to identify vulnerabilities. However, because most commercial software products are not open source, these techniques are not always viable. As a result, binary code analysis has become a necessity. Manual binary analysis, on the other hand, is a demanding, error-prone, and difficult process, particularly when dealing with a high number of embedded device firmware images. As a result, automated and scalable vulnerability identification is becoming increasingly important, in particular, it is highly desirable to scan a large number of firmware binaries for known and undiscovered vulnerabilities and produce a vulnerability analysis report in a timely manner. In this work, we have identified the architectural characteristics of IoT and embedded device firmware which include processor architecture, operating systems, bootloaders, protocols, and communication interfaces. Further, we discussed various firmware extraction techniques that are crucial in getting hold of IoT firmware. Furthermore, a detailed review of various vulnerability analysis frameworks is presented. A comparative analysis of these frameworks based on some common parameters is also provided. In the end, a new vulnerability analysis framework is proposed addressing some of the issues in already existing frameworks.

1.1 Research contributions

The following are the main contributions of this paper.

- We have presented the architectural characteristics of IoT and embedded devices.
- We have discussed the techniques for the extraction of firmware from IoT and embedded devices.
- A comprehensive review of the state-of-art vulnerability analysis frameworks is presented with comparative analysis.
- Finally, various challenges and gaps, facing in performing firmware analysis are given.
- A new vulnerability analysis framework is proposed to address some of the challenges in the existing frameworks.

1.2 Methods and materials

We have used an advanced search approach to identify the related papers for our survey. We have mostly included papers from the most reputed journals of the IEEE, ACM, Wiley, Elsevier, and Springer publishers. Proper search strings with appropriate Boolean operators have been used in the advanced search such as "all in title: ("Firmware") AND ("Vulnerability" OR "Analysis" OR "Security" OR "Blockchain" OR "Extraction") source:" Springer" OR source:" ACM" OR source:" IEEE" OR source:" Wiley" OR source:" Elsevier". We then filtered the results using various filters such as year of publication range and name of journals. The results were then properly filtered and various irrelevant papers were also removed.

1.3 Organisation

Table 1 presents the acronyms that are used in this paper. The rest of the paper is organized as follows. Section 2 is divided into three subparts—(A) Architecture of firmware and its various technical characteristics are discussed. (B) Various extraction methods are presented. (C) Various Firmware analysis frameworks are reviewed. Section 3 presents various challenges in firmware analysis. The proposed model is presented in Section 4. Conclusion and future work are given in Section 5.

2 Background

This section focuses on the background of IoT-embedded firmware. This section is divided into four subsections. Firstly, we discuss IoT & Embedded Device Firmware Architecture. Secondly, we discuss the tools and techniques for the extraction of the firmware. In the third subsection, we focused on different types of vulnerability analysis methods. The fourth subsection presents various secure update mechanisms for the IoT device firmware.

2.1 Firmware architecture of IoT & embedded devices

The term "firmware" refers to binary software stored in an EEPROM or FLASH chip. The two available forms of firmware are low-level and high-level firmware. EEPROM usually stores the low-level firmware making it difficult to modify or update, while high-level firmware is stored in Flash memory. Firmware resides between the hardware and the application layer software, it works as an interface program for the software layer by realising the hardware commands. Firmware is the combination of various parts of binary files which include bootloader, OS kernel, file system, and various headers and because IoT devices have limited computational capabilities and storage space, firmware is frequently burnt in the compressed form [9]. IoT devices are more than just wireless sensors integrated into a gadget. The Internet of Things (IoT) is the connectivity of Wireless Sensor Network (WSN) devices with the Internet. The energy and memory resources available to IoT devices are typically limited. They're usually tiny and battery-powered,

Table 1 Acronyms used in the paper

Acronym	Explanation
APT	Advanced Persistent Threat
ISA	Instruction Set Architecture
MIPS	Million instructions per second
JTAG	Joint Test Action Group
UART	Universal Asynchronous Receiver Transmitter
GPIO	General Purpose Input/Output
LE	Little Endian
BE	Big Endian
IED	Intelligent Electronic Device
IoT	Internet of Things
DNN	Deep Neural Network
CNN	Convolutional Neural Network
COTS	Commercially of The Shelf

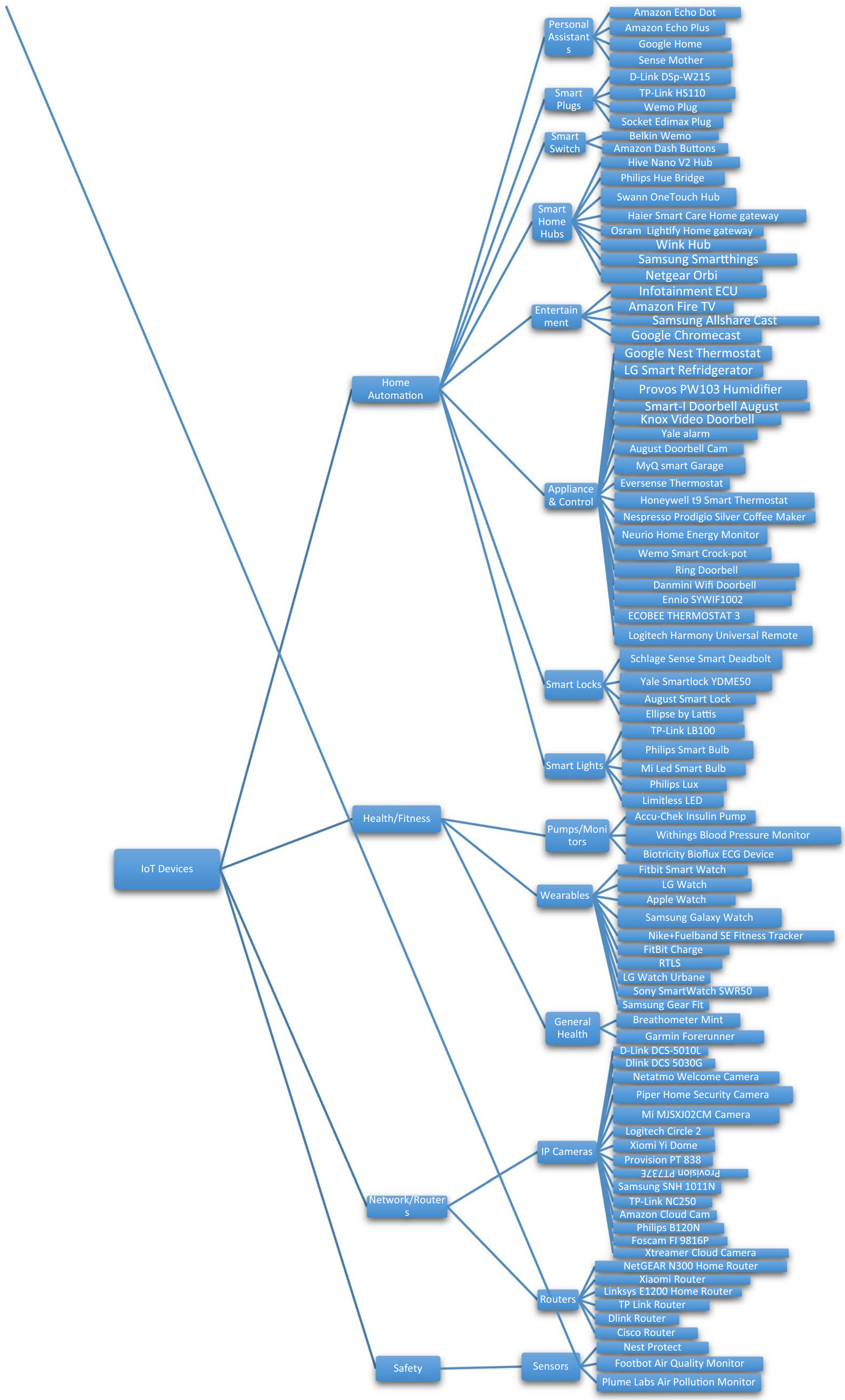


Fig. 1 Taxonomy of commonly used & most popular IoT devices [10–12, 28–30]

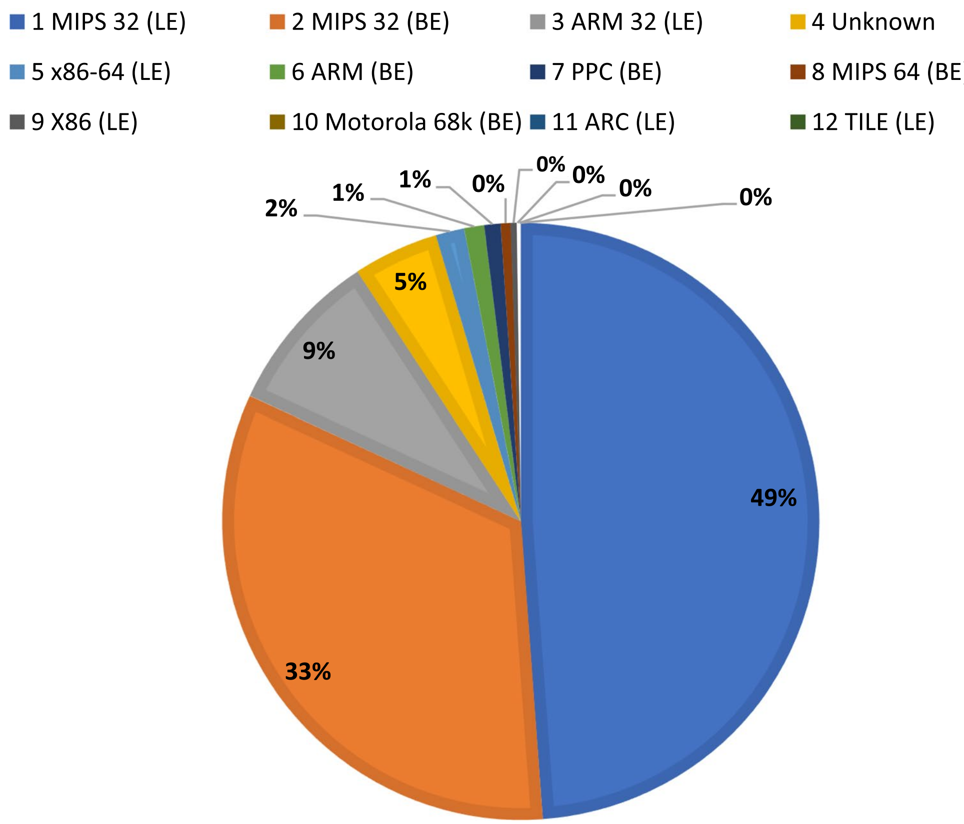


Fig. 2 Types of firmware Instruction Set Architectures (ISA)

having a memory capacity of around 100 kilobytes. Typical 8-bit microcontrollers are used in these machines, which are considerably behind the current generation of Windows/Unix/Mac-based PCs and laptops. Figure 1 presents the taxonomy of the commonly used IoT & embedded devices [10–12]. The devices have been classified under 4 broad categories viz Home Automation, Health/Fitness, Network/Routers, and Safety. These are further divided into 14 subcategories. The architectural characteristics of IoT and Embedded Device firmware have been presented in the subsequent section in terms of processor architecture, operating systems, bootloaders, kernel modules, and protocols.

2.1.1 Processor ISA

The architectures of embedded devices are quite varied. ARM and MIPS processors are widely used in the midrange to upper-class market sectors of processors that offer capabilities such as memory virtualization and high clock rates [13], and Intel is trying to catch up with its ATOM line. Processor designs with tiny memory and lower clock speeds, such as Atmel AVR or Intel 8051, are available in the lower-class market. The authors in [14] have analyzed approximately 9486 firmware images. The analysis resulted in identifying the various technical characteristics of the embedded device firmware including the identification of processor architectures, device operating systems, and protocols through the machine learning approach. The authors have reported that the majority of the firmware images which constitute around 79.4% of the analyzed firmware are based on MIPS 32 bit (Big Endian & Little Endian) architecture. The next most popular processor architecture is ARM 32bit (LE) which constitutes approximately 8.9% of the analyzed firmware. Another report given by Costin et al. [13] shows that after an automatic analysis of about 172,751 possible firmware images out of which 63% of them had ARM architecture and 7% were MIPS based. Together these constitute around 90% of the popular processor architectures used in IoT devices. The remaining portion consists of other different types of architectures. Figure 2 illustrates the architecture share among the analyzed firmware images presented by the authors in their research [14].

Table 2 Processor and ISA specification of various identified IoT devices

Device name	Vendor	Processor/microcontroller	ISA	Source
Nest Thermostat	Nest Labs	TI Sitara AM3703 system-on-chip (SoC) • ST Microelectronics ARM Cortex-M3 based microcontroller • ARM Cortex-A8 core, with ISA Version 7	ARM32 ISA Version 7	[15]
Fit Bit Smart Watch	Fit Bit	ARM cortex processor; the STMicroelectronics 32L151C6	ARM32	[16]
Nike + Fuelband	Nike	STM32L151QCH6 Microcontroller ARM Cortex M3 core	ARM32	[15]
Haier SmartCare	Haier	TI AM3352BZCZ60 ARM Cortex A8	ARM32	[17]
ItronCentron smart meter	Itron	ATMega microcontroller	Atmel AVR	[17]
Amazon Echo Plus	Amazon	MT8163 processor	ARMv8 (A32, A64)	[11, 18]
Yale Easy Fit Smartphone Alarm	Yale	Freescale MK60 CPU	ARM32	[11]
Philips Hue Bridge V2	Philips	Qualcomm QCA4531 SoC	MIPS	[19–21]
Accu-Chek Insulin Pump	Accu-Chek	iMX233 (Soc) ARM926EJ-S	ARM32	[22, 23]
Hive nano v2	Hive Home	Texas Instruments AM3352BZCE30 (MPU ARM Cortex A8 32-Bit RISC Processor)	ARM32	[24]
Samsung Smart Lock SHS-5230	Samsung	Atmega64A(Main Board), Atmega88PA(Rf Board)	AVR	[24]
Swann OneTouch hub	Swann	NA	ARM32	[24]
Amazon Echo Dot v2	Amazon	TI DM3725CUS100 (ARM Cortex A8)	ARM32	[25, 26]
Tp Link Router TL-WR841N	TP-Link	Mediatek MT7628NN SOC	MIPS32	[27]

Table 2 presents the study of various popular IoT devices. This also points to the fact that these devices are based on the popular ARM and MIPS-based architectures. The processors mainly used in these devices are based on ARM, TI, and AVR microcontrollers. These companies are the leading producer of semiconductors for IoT & Embedded devices.

2.1.2 Operating systems

Firmwares of various levels of complexity power embedded systems. A full-fledged operating system, such as Linux or Windows NT, is generally used for more complicated ones. Operating systems like as VxWorks or Windows CE are used by less sophisticated devices, and a variety of special-purpose operating systems are also available. According to the findings given by authors in [14], around 40% of devices had Linux OS, 9% had Unix, and 3.5% had firmware based on VxWorks OS following signature analysis of device firmware. Other firmware had monolithic designs that didn't have a

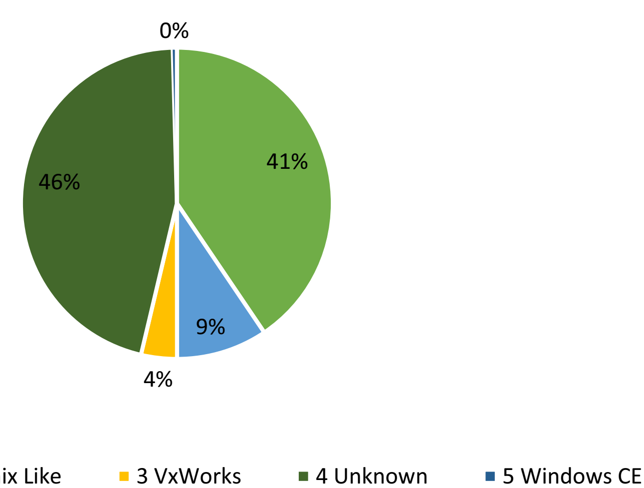


Fig. 3 Popular operating systems used in IoT devices

Table 3 Major operating systems of various IoT devices

Device name	Embedded OS	Source
Nest Thermostat	Embedded Linux Version: 2.6.37	[15]
Haier SmartCare	Embedded Linux	[17]
Amazon Echo Plus	Embedded Linux	[11, 18]
UEI Smart-I Doorbell	Embedded Linux	[11]
Philips Hue Bridge V2	OpenWrt (Linux Based)	[19–21]
Belkin Wemo Switch	OpenWrt	[32, 33]
Accu-Chek Insulin Pump	Windows CE 6.0	[22, 23]
Swann OneTouch hub	Embedded Linux	[12]
Amazon Echo Dot v2	Fire OS v 5.5	[25, 26]
Tp Link Router TLWR84N1	Embedded Linux	[27]

Table 4 Various Bootloaders used in IoT devices

Device name	BootLoader	Source
Nest Thermostat	x-loader (Firststage), U-boot(second stage)	[15]
Haier SmartCare	U-Boot	[17]
Amazon Echo Plus	x-loader (first stage), U-boot (second stage)	[11, 18]
UEI Smart-I Doorbell	U-boot	[11]
Philips Hue Bridge V2	U-Boot	[19–21]
Belkin Wemo Switch	U-boot	[32, 33]
Accu-Chek Insulin Pump	Windows CE Bootloader	[22, 23]
Swann OneTouch hub	Uboot	[12]
Amazon Echo Dot v2	NA	[25, 26]
Tp Link Router TLWR84N1	Uboot 1.1.4	[27]

particular kernel module. Figure 3 presents the most popular firmware OS used in IoT devices. Linux OS dominates the IoT landscape with a wide variety of library implementations/versions along with the same ABI-compatible Linux kernel with versions ranging ($2.4 < x < 4.3$) [31]. Table 3 presents the operating system specification of various identified IoT devices.

2.1.3 Bootloaders

The bootloader is the first programme that a system runs. It puts the kernel into memory for execution and initializes various hardware components which include Flash storage, I/O, RAM. The boot process in embedded systems can be divided into one, two, or three phases, with each step performing a particular function during startup. In a three-step procedure, the first bootloader conducts necessary hardware startup and loads the second-stage bootloader, which is usually located in ROM and is specific to microcontroller. The second stage bootloader initialises all the board-specific components and it usually resides on flash memory. After initialization, it loads the third stage bootloader, which loads the kernel into primary memory, initialises device drivers for the identified system components, and runs the kernel. U-Boot among the most popular second stage bootloaders for embedded devices [34]. A command-line interface is also available in it. Bootloader specification of some of the various widely used IoT devices is given in Table 4. It is evident from this table that the widely used bootloader is U-Boot.

2.1.4 Kernel modules

Kernel modules are small bits of code that may be loaded and unloaded from the kernel as needed. They improve the kernel's functionality without requiring a system reset. Networking modules, cryptography modules, filesystem modules, and peripheral modules are some of the several types of kernel modules that may be found in embedded operating

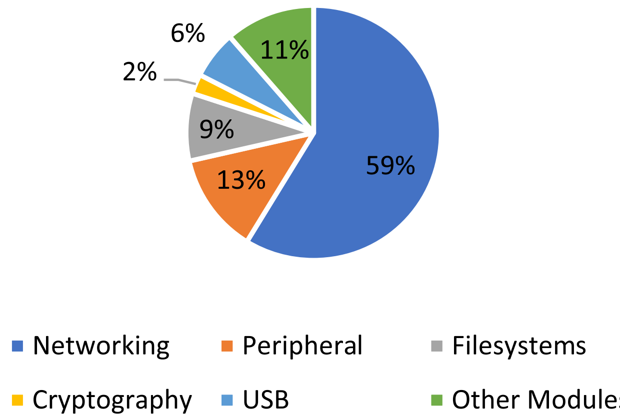


Fig. 4 Breakdown of types of kernel modules present in IoT devices

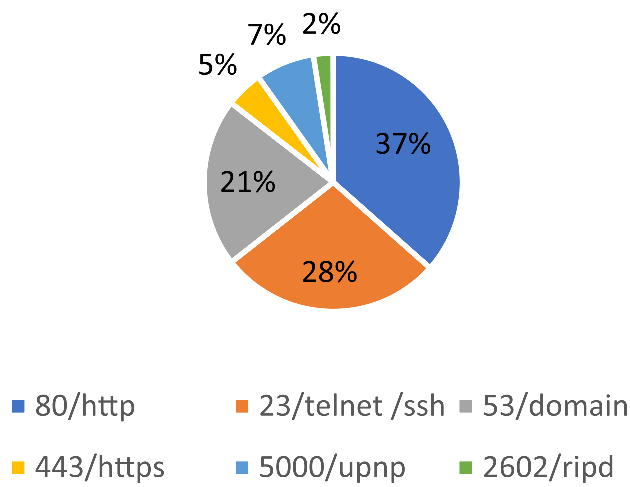


Fig. 5 Commonly used TCP/IP stack protocols in IoT devices

systems. The authors in [14] have analyzed that the networking modules consist of the largest share of the modules which approximates to 58.8% of the 504,815 identified firmware modules. The next largest part consists of peripheral related modules (12.6%) that include support for wireless adapters, chipsets, and I/O functionalities. Figure 4 shows the share of the kernel modules present in the analyzed IoT firmware.

2.1.5 Protocols

IoT devices employ different application layer protocols from the TCP/IP protocol stack, such as HTTP/HTTPS, FTP, Telnet, and ssh for authentication and data transfer. The authors in [14] have performed a network analysis of around 1971 firmware images and the analysis revealed that around 42% of the firmware supported the HTTP/HTTPS protocol. Remote shell access is supported by approximately 37% of the devices using the ssh or telnet protocol, however, 1.9% of the ssh supported devices also support telnet. Figure 5 presents a brief statistic of the commonly used TCP/IP stack protocols in the IoT devices.

2.2 Extraction techniques

When evaluating the safety of IoT devices, extracting the firmware is a critical first step. Preventing the firmware from any adversary is always desired from the standpoint of a designer: for example, to protect cryptographic keys used to recognize a device and prevent device cloning or intellectual property theft. Because of the wide range of IoT devices, multiple techniques for firmware extraction are required depending on the device.

Having access to the firmware of an embedded device may provide a lot of information about how it works and what vulnerabilities it has. Firmware frequently contains sensitive data like passwords and static keys, indicating unsafe design and poor security overall. Furthermore, the methods used for extraction of firmware provide the device write access, thus enabling the firmware to be modified. The extraction of firmware isn't a precise science. IoT devices are very diverse due to their manufacturer specific configurations and software stacks. The firmware extraction of IoT devices is complicated by these specifications.

A number of methods for extracting firmware from a variety of IoT devices were given at DEFCON 25 [35], while authors in [36] emphasised on using the eMMC interface. The Exploitee.rs project was the result of this work. According to the research, The UART debugging interface has been found as the most exploitable programming & debugging interface in IoT devices. UART being vulnerable to firmware extraction in over 45 percent of the devices evaluated. Flash memory access is becoming increasingly crucial for contemporary gadgets. Notably, in virtually all situations when a hardware mechanism for firmware extraction is available, the approach also allows for firmware change and therefore device "rooting". Extraction of firmware raises a number of difficulties for IoT device makers. For starters, there's a chance of losing intellectual property. More significantly, extraction of firmware can often lead to the discovery of security flaws in these devices. Due to severe vulnerabilities found, in some cases, this might have an impact not only on the examined device, but on all of the manufacturer's products. The methods for extracting firmware are classified into three groups:

1. Utilising debug interfaces to get access to a local shell or read the contents of a memory.
2. Implementing a hardware memory dump on a flash chip.
3. Obtaining firmware access using software methods such as firmware upgrades and network eavesdropping.

2.2.1 Hardware methods

Firmware extraction through hardware methods uses the on-chip debug interfaces which include UART, JTAG, SPI, and I2C. The hardware method is often complicated in the process as it requires device-specific tools for carrying out the extraction. The three commonly used hardware-based methods for firmware extraction are discussed as follows.

2.2.1.1 Using UART Direct access to an embedded device's firmware through UART is usually a straightforward [37] method. Simply connecting to UART can lead directly to an unrestricted root shell. The Android debug interface ADB can sometimes provide access to a root shell on Android-based devices. Sometimes a root shell of a device is unavailable or is password secured then in that case a bootloader shell is used to get access to the firmware image. A internal dump of the complete filesystem is one of the main way to dump the firmware of a device with a live root access, all the archived files can then be unpacked using various open source tools. However, because embedded systems employ various types of flash storage with various filesystems, dumping block devices might cause issues. In general, the following steps are carried out when performing extraction of firmware through UART interface:

1. Visually inspect, oscilloscope probe, and trial-and-error to determine the UART interface;
2. An insecure shell can also be used to download the firmware image of a device. Netcat or related programmes and a computer on the same network can be used to download files.
3. If a shell is secured through password, guess all the default password pairing such as admin/admin. If shell is not accessible or no password is not accepted, try interrupting the boot process and entering the bootloader shell.
4. If you can't get into the bootloader shell, try momentarily disrupting the flash interface by grounding a data or clock pin or any other method while the bootloader loads the kernel.

2.2.1.2 Using JTAG The JTAG connection that's used to load firmware during production may well be used to read the chip's entire memory. An appropriate programmer must be able to accept the memory dump and transfer it to a computer in order to read a device's memory through a JTAG connection. After the gadget is manufactured, some manufacturers prevent it from being read or reprogrammed. The device is vulnerable to firmware extraction and injection attacks if the JTAG port is attached and unlocked. The typical procedure for extracting firmware through JTAG is as follows:

1. Manually identify JTAG or SWD debug port pins. JTAG offers variety of pin configuration ranging from 8 to 20 pins and SWD requires just two pins.
2. In using UART ground pin is identified first, when all the pins are identified a suitable UART debugger module is used to dump the contents of the internal memory.
3. Use datasheet to identify the pinout of specific microcontroller, JTAGulator [38] can also be used to identify the pins if datasheet is not available.
4. If no readout protection is activated, use an appropriate JTAG/SWD programmer to dump the internal memory.

Firmware extraction using JTAG becomes more complicated process due to the variety of pinouts and wide range of JTAG debuggers for different types of architectures. Thus, it is relatively easy to extract firmware through UART than JTAG.

2.2.1.3 Dumping flash Directly accessing the flash storage is another method for hardware-based extraction. Older flash memory chips requires a lot of connections to the device and the use of specific programmer devices for transferring data efficiently. However, technologies like eMMC requires few connectors and can also be accessed with an SD card reader. Also, specific tools like easy RiffBox or JTAG Plus can also be used. A detailed process of extraction using eMMC can also be found in [39]. Some of the basic steps for performing flash dumps are as follows:

1. Determine the flash chip's identity (based on a label, packaging type, and number of processor connections) and, if feasible, get a data sheet;
2. Use a datasheet or an oscilloscope to determine the pins. eMMC uses various pins which include CMD, CLK, and DAT0. CLK is a signal that repeats itself, whereas the CMD line includes brief data bursts that occurring before read or write of data on DAT0 pin.
3. Disable the processor's access to eMMC first and link pins to an SD card, which may interface with using an SD card reader.
4. To access the contents of different flash chips, use an appropriate programmer, such as the MiniPro TL866;
5. If an in-circuit dump isn't feasible, disassemble the flash chip and dump it using an appropriate reader.

It is necessary to restrict access from the board's CPU when accessing the memory for in-circuit dumps. This can be accomplished, for example, by momentarily disconnecting the clock line and reconnecting it once the dump is finished. Simply attaching an eMMC interface (such as easy JTAG Plus) might sometimes block the CPU from starting. Alternatively, you may use the appropriate pin to maintain the CPU in reset mode.

2.2.2 Software methods

Extraction of the firmware through software techniques does not require any access to the physical device. Examples include:

1. Browse for publicly accessible firmware on the device manufacturer's website.
2. Analyze the device's network activity while following direct download URLs for firmware upgrades.
3. Use network traffic to intercept firmware upgrades. If TLS is in use, try to decrypt communications using self-signed certificates in a man-in-the-middle attack.

Vendors often provide firmware updates that only include revised files in that case complete firmware retrieval becomes difficult and alternate methods need to be explored. However, in certain situations, firmware upgrades contain entire firmware images, making this approach a quick and easy way to extract the firmware. Sometimes it is difficult to unpack some firmware images due to the implementation of firmware encryption or use of proprietary formats in compressing of firmware.

Table 5 Various onboard programming & debugging tools

Tool	Supported interfaces	Supported device type	References
The Shikra	JTAG, SPI, I2C, UART, GPIO	Not Specified	[40]
Attify Badge	UART, I²C, SPI, JTAG	Not Specified	[41]
Adafruit FT232H	SPI, I2C, serial UART, JTAG	Not Specified	[42]
HydraBus v1.0	SWD & JTAG, SMARTCARD, NAND flash, SPI, I2C, UART	Not Specified	[43]
Keil ULINK2	JTAG, SWD, SWV	ARM	[44]
Flyswatter2	JTAG	ARM, MIPS	[45]
Bus Pirate—v3.6a	UART, I²C, SPI, JTAG	NA	[46]
Black Magic Probe Mini V2.1	JTAG,SWD	ARM	[47]
JTAGulator	JTAG, UART	ARM,MIPS	[48]
AVR Dragon	SPI, JTAG, PDI	Atmel AVR	[49]

Table 6 Supported software tools for firmware extraction

S.No	Tool	Supported CPU Types	Type	References
1	OpenOCD	ARM, MIPS	Opensource	[50]
2	UrJTAG	Not Specified	Opensource	[51]
3	AVR Dude	Atmel AVR	Opensource	[52]
4	Easy JTAG plus	NA	Free	[53]

2.2.3 Firmware extraction tools

Firmware Extraction tools are broadly categorized into two classes: (1) Hardware tools, and (2) Software tools. The software tools are used in combination with supported hardware tools. The software tools can be freely downloaded online from their respective websites. Table 1 lists the various hardware tools that have been identified for the extraction of firmware using hardware-based methods. The various interfaces that are supported by these tools are UART, JTAG, SPI, I2C, and SWD. The cost of these tools typically ranges from $40 to $200 in the global market. These tools are typically used with their software counterpart which is usually free and open source. The tools include OpenOCD, Urtag, easy JTAG, and AVRdude, the details are listed in Tables 5 and 6.

2.2.4 Firmware unpacking & analysis tools

Firmware analysis is not quite straightforward and easy, and it necessitates a number of procedures prior to the analysis phase. Extraction, unpacking, and determining the file system, among other things, are all essential stages. After the firmware has been unpacked/extracted, it may be evaluated and analyzed for security. Using Binwalk [54], It is feasible to reverse engineer and do a rudimentary analysis on IoT device firmware images. Firmwalker [30] may be used to look for essential files such as private keys, certificates, and password files. IDA and Ghidra [55, 56] can be used to disassemble and debug even obfuscated code. For the emulation of the firmware, we can use QEMU [57]. Most of these tools are open-source which can be downloaded online. Table 7 presents all the tools which are used to do firmware-based evaluation & security profiling of IoT Devices.

2.3 Vulnerability analysis frameworks

The security vulnerabilities can be found in various parts of an IoT system which include hardware components, application software [64, 65], underlying firmware, and cloud system [66–70]. Some of the various techniques used to find security flaws in IoT system are Static analysis, Dynamic analysis, Penetration testing, Fuzzing, and various other techniques.

Different sorts of vulnerabilities can be discovered with each approach. Identification of vulnerabilities in the embedded devices and in their underlying firmware serves a crucial role in securing embedded systems. To this end, there are a variety of methods for detecting and triggering possible vulnerabilities in deployed embedded system firmware. In

Table 7 Various tools for firmware unpacking and analysis

Tool	Description	Type	References
Binwalk	Tool for analysing, extracting, and reversing firmware images and other binary blobs	Opensource	[54]
firmware-mod-kit	This package makes it simple to deconstruct and rebuild firmware images for a variety of embedded devices	Opensource	[58]
Radare2	Analyzing binaries, disassembling code, debugging programmes, and connecting to remote gdb servers are all supported	Open Source	[59]
Firmadyne	Emulation and dynamic analysis of Linux-based embedded firmware using an automated and scalable approach	Opensource	[60]
Qemu	A programme that allows you to virtualize your hardware	Opensource	[57]
binvis	It's a programme that allows you to visually inspect binary files	Opensource	[61]
Firmwalker	A simple bash script to search the firmware file system that has been extracted or mounted	Opensource	[62]
FwAnalyzer	FwAnalyzer is a programme that analyses filesystem images such as (ext2/3/4), FAT/VFat, SquashFS, and UBIFS	Opensource	[63]
Ghidra	Reverse engineering tool developed by the National Security Agency	Opensource	[55]
IDA Pro	A disassembler for binaries that converts machine-executable code into assembly language source code	Proprietary	[56]

this work, we give a detailed review of the some of the recent ideas, which utilize different analytic techniques, such as static, dynamic, and hybrid analysis approaches, to discover known and unknown firmware vulnerabilities.

2.3.1 Static analysis frameworks

Static analysis is used to find security flaws in firmware by analyzing the programme, which includes control, data flow, lexical, grammatical, and semantic analysis, among other things. Static analysis is a notion that has been around for a long time. It involves lexically examining a program's source code without running it [70]. Static analysis is used find various security flaws which include buffer overflows, type-checking errors, kernel deadlocks, susceptible function calls, and various other flaws. There are various analysis tools for such purposes which include Visual Code Grepper [71], CP-PCheck [72], PMD [73], Ghidra, IDA pro, and various other tools. Based on its implementation and targeted programming languages, each tool has its own technique of detecting mistakes. As IoT is made up of a variety of software components, APKs, and firmware, analyzing and detecting security flaws in these components is critical. In the past years, a substantial amount of study has been focused on firmware in general [74]. Static analysis techniques usually suffer from various limitations [75]. Although static analysis techniques are more scalable than dynamic analysis approaches, researchers are increasingly combining the two approaches as they both have their own set of constraints.

2.3.1.1 discovRE Authors in [5] have developed and implemented a framework called discovRE, that supports four instruction set architectures which include ×86, ×64, ARM, and MIPS. It is a cross-architecture bug search framework for binaries. It works by matching a known vulnerable binary function with target firmware binaries typically compiled for different architectures, that contain the same vulnerable function. Two types are features are extracted prior to matching which are structural features and numerical features. Structural features are used to build a CFG (Control flow graph) of the binary. While numerical features represent the information about the number of instructions or number of basic blocks of a function. However, these CFG-based bug search approaches are far from being scalable to handle an enormous amount of IoT devices in the wild, due to their expensive graph matching overhead. This framework was evaluated on three firmware images and bugs like Poodle or Heartbleed were detected. Figure 6 shows the main process of this approach.

2.3.1.2 Genius A bug identification approach [76] that increases search accuracy while addressing the scalability challenge in existing tools like discoverRE. It constructs the attributed control flow graph using statistical and structural fac-

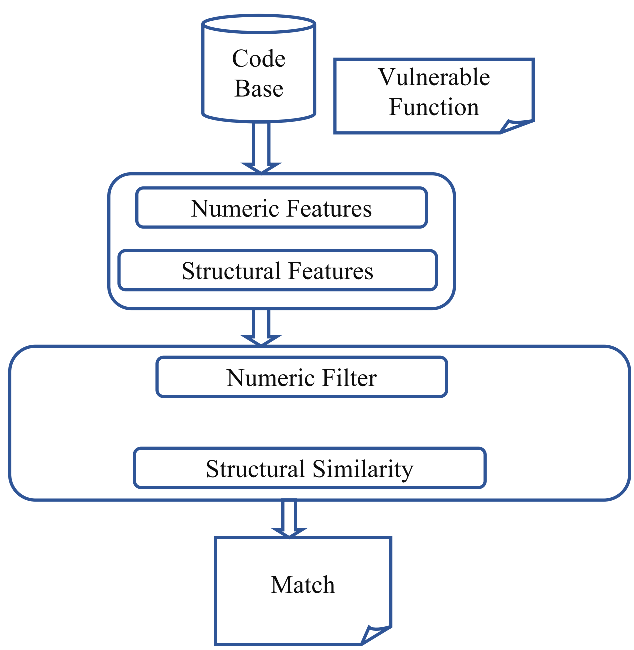


Fig. 6 Architecture of discovRE [5]

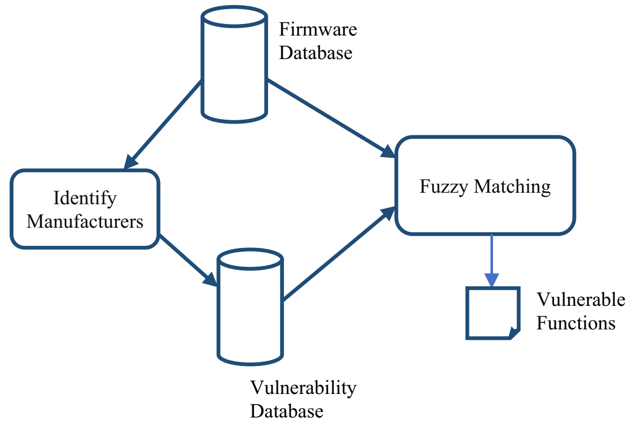


Fig. 7 Proposed Binarm [77]

tors that are consistent across various CPU architectures and labels each basic block in a CFG with the set of attributes (ACFG). The ACFGs are transformed into codebooks using spectral clustering in order to do a more efficient search. However, it is stated by the authors that the creation of a codebook is computationally expensive.

2.3.1.3 BinArm Authors in [77] presented a vulnerability detection technique called BinArm for smart grid IED firmware. It is a multistage detection engine that performs coarse to fine-grained detection as shown in Fig. 7. In the first stage, dissimilar functions having heterogeneous features are discarded. The second stage discards function based on different execution paths. The third stage identifies candidate functions using fuzzy graph matching based on weighted Jaccard similarity and Hungarian algorithm. It is proposed to be efficient in identifying vulnerabilities in IEDs in a smart grid system. However, the authors state that this system only performs analysis of ARM-based intelligent electronic devices and it fails to detect runtime exploits.

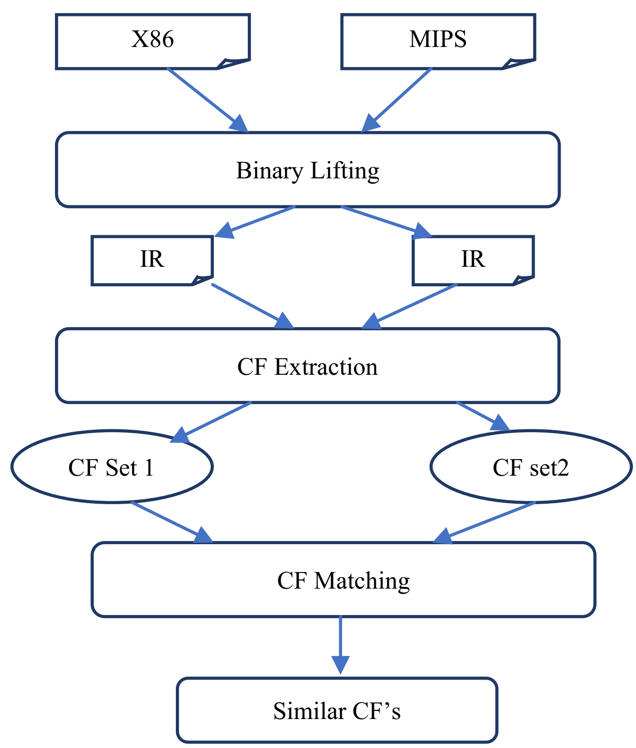


Fig. 8 XMatch approach overview [80]

2.3.1.4 FirmUp This method is given by David et al. [78]. It identifies the vulnerable procedures by considering procedure-based relationships in firmware images. It establishes a correspondence between a set of procedures in a given binary and a target binary. An algorithm called Ehrenfeucht-Fraïssé [79] is used to establish a pairwise similarity between sets of procedures. This approach is tested on about 2000 firmware images and 373 vulnerabilities were discovered out of which 147 appeared in the latest firmware images.

2.3.1.5 XMATCH This is a cross platform analysis framework given by Feng et all [80]. In this framework as shown in Fig. 8, three stage process is used for analysis. In the first stage, binary lifting is performed which produces an intermediate representation of the two binaries using usingMcSema [81] translator. In the next stage, conditional formulas are constructed from the lifted binaries. CF's are used to capture two main factors of a bug, erroneous data dependency, and invalid conditional checks. Irrelevant variables are also discarded. In the third stage function matching is done using the already extracted conditional formulas and by employing integer programming techniques. After that one to one mapping is performed between the CF's in addition to similarity scores.

2.3.1.6 VulSeeker Vulseeker is given by Gao et al. [82]. It is also a cross-platform approach based on function matching. The target function is compared with a vulnerable function and based on the similarity score the output is decided. Labelled semantic flow graph's (LSFG) is constructed from the two binary functions then 8 types of instruction features are extracted as a numerical vector for each block of LSFG. After this, the numerical vector is fed into a DNN model to generate function semantics. The output is then decided based on the Cosine similarity score. The architecture is shown in Fig. 9.

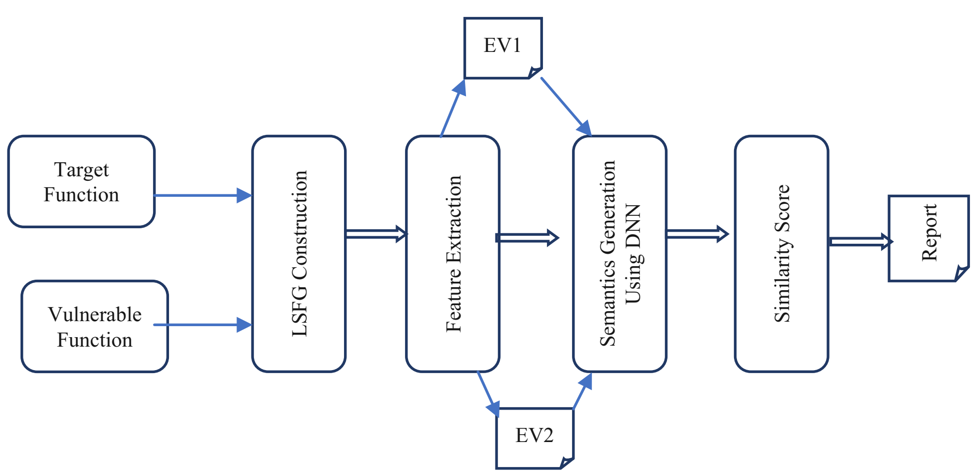
2.3.1.7 aDiff This approach [83] extracts three types of features from binaries which are intra function, inter function, and inter-module features. The CNN and a Siamese network are used for the extraction of semantic features. After extraction of these features from two binaries, a distance measure is calculated between each pair of features of the two binaries. An overall similarity score is then obtained based on the three calculated distances.

2.3.2 Dynamic analysis frameworks

Dynamic analysis approaches rely on the firmware's real execution on hardware devices or emulators. By providing appropriate test inputs to analyze programme behavior, all of the firmware execution pathways are covered. In this part, we look at some of the most advanced dynamic analysis techniques for IoT and embedded device firmware. A comparative study is also supplied at the conclusion for a more in-depth comparison of the approaches mentioned.

2.3.2.1 Avatar Jonas et al. have given a dynamic analysis framework for embedded devices called Avatar [13]. This framework shown in Fig. 10 works by a tight integration of an emulator with an embedded device for helping in various security tasks which include vulnerability discovery, vulnerability analysis, malware analysis, backdoor detection,

Fig. 9 Vulseeker Workflow [82]



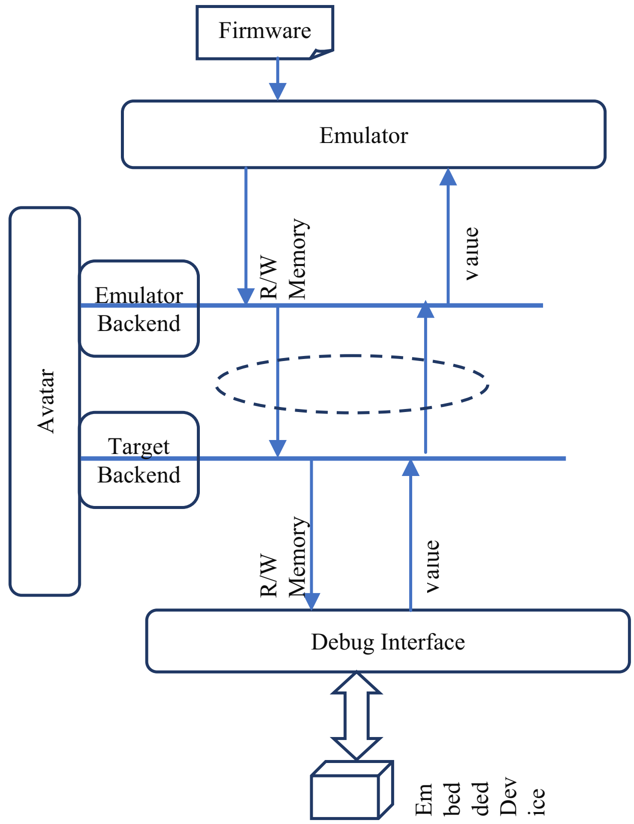


Fig. 10 Avatar Architecture [13]

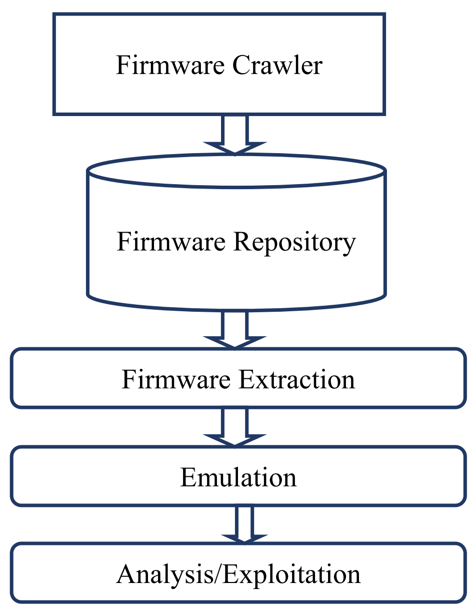


Fig. 11 Flow diagram of firmadyne [94]

and reverse engineering. An emulated firmware forwards I/O accesses to the real embedded devices thus completely emulating a full system behavior. Debug interfaces such as JTAG together with OpenOCD were used for communication with the real hardware device. The authors performed the analysis of three devices: a gsm-based phone, a hard disk bootloader, and a sensor node. Avatar supports all the major hardware architectures which include × 86–64, ARM, MIPS.

2.3.2.2 Firmadyne Chen et al. [84] have developed an automated dynamic vulnerability analysis system that supports full system emulation through QEMU. It specifically supports Linux-based devices. Firmadyne as shown in Fig. 11 consists of three major components which are Firmware Crawler for downloading firmware images from vendor websites, Firmware Extractor for extracting the downloaded file system, System Emulator for performing the initial emulation, and Dynamic Analyzer for running the exploits. Three types of architectures are supported by this framework which are MIPS-BE, MIPS-LE, and ARM-LE. The authors performed an extensive analysis in terms of the firmware count on about 9486 firmware. However, the dynamic analysis performed is rather simple in nature. The dynamic analyzer module consisted only of predefined exploits from the Metasploit framework and some custom-made exploits. These exploits are great in identifying the known vulnerabilities but are not effective in identifying zero-days.

2.3.2.3 Automatic analysis framework Costin et al. presented a dynamic analysis framework in [85]. In this framework, authors have used COTS tools for performing static and dynamic vulnerability analysis in the web interfaces of the embedded devices. Full-Scale emulation of 246 firmware images has been performed to test the web interfaces. The authors have found 225 high-impact vulnerabilities in around 24% of the emulated firmware. Tools such as RIPS, shodan, and ZMap were used for performing analysis. However, these tools have a limitation of producing high false negatives and false positives.

2.3.2.4 IoTFUZZER IoT fuzzer is an automatic blackbox texting framework given by Chen et al. [86]. This framework aims at finding memory corruption vulnerabilities in firmware images by analyzing supporting apps. Dynamic analysis is performed on the app to reveal the logic that is used to construct the messages for communication with an IoT device. This framework has four main phases. In the first phase UI of the app is analyzed for the identification of components that trigger network connections. The second phase analyses the app for various strings and values which are required to construct a network-based protocol message. Then in the third stage, all the recorded protocol fields are used to construct a new message to be sent to the IoT device. The final stage monitors the status of the IoT devices and records any crashes or memory corruptions. The authors have evaluated this framework on 17 IoT devices and identified 15 memory corruption vulnerabilities. However, this framework provides only the input data that triggers the vulnerability and not the location of the vulnerability in the firmware.

2.3.2.5 Pretender A dynamic analysis model called PRETENDER based on firmware re-hosting is given by Gustafson et al. [87]. In this model as shown in Fig. 12 interactions between firmware and hardware are recorded and then modelled using machine learning and pattern recognition techniques. After the completion of modeling, hardware is completely replaced with a virtualized environment. Virtualized environment is realized using QEMU [88] and for carrying out effi-

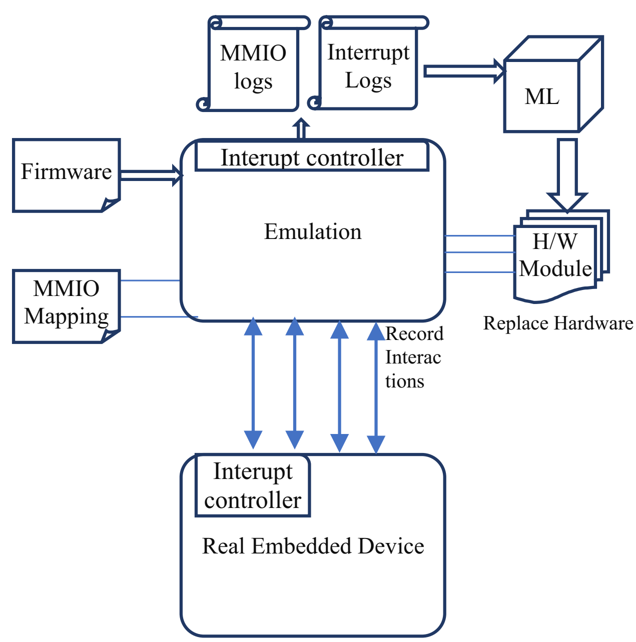


Fig. 12 PRETENDER workflow diagram [87]

cient program analysis angr is used. PRETENDER was evaluated on six firmware images of three different hardware types. PRETENDER was developed to provide an advanced approach for performing dynamic analysis on firmware images.

2.3.2.6 FirmFuzz Srivastava et al. [89] developed a dynamic vulnerability analysis framework of Linux-based IoT devices called FirmFuzz. It uses QEMU tool for carrying out emulation of the MIPS and ARM-based IoT firmware. There are three phases used in the analysis which are, Information gathering, Preparation, and Fuzzing. Firmware Fuzzing is the main technique used in identifying the vulnerabilities. It utilized the web interface of devices as entry points for fuzzing the firmware images. FirmFuzz managed to discover seven unknown vulnerabilities in six different devices by analyzing 32 images of 27 devices.

2.3.3 Hybrid analysis frameworks

The hybrid analysis is the combination of static analysis and dynamic analysis approaches. While designing hybrid analysis frameworks researchers frequently use various deep learning and machine learning methods to automate the process to a certain level. Very little work has been done in this area as a combination of both of the approaches presents some serious challenges. In this section, we have reviewed some of the existing work that has been done on hybrid approaches.

2.3.3.1 DTaint It is a framework [90] to analyse taint style vulnerability in the embedded devices firmware. These types of vulnerabilities are weaknesses due to improper or no sanitization of input data. It has an input source, a specific data flow path, and a data sink that is sensitive in nature. A vulnerability such as the heartbleed [91] bug in the OpenSSL library is an example of a taint-style vulnerability. This framework as shown in Fig. 13 uses both static analysis and dynamic analysis techniques for the identification of vulnerability. In this framework firmware images are taken as input and outputs data flows from these images by using four components which are data structure, functional analysis, pointer aliasing and intraprocedural data flow components. The author applied this framework over 6 firmware images of four manufacturers and identified about 21 vulnerabilities including 13 zero days. However, this approach only identifies taint style vulnerabilities.

2.3.3.2 PATCHECKO PATCHECKO is a state-of-the-art hybrid vulnerability analysis framework given by Sun et al. [92]. The architecture of PATCHECKO is shown in Fig. 14. It works in three phases: (1) It uses a deep learning technique to train the vulnerability detector. (2) Target firmware is of IoT/embedded devices is statically analyzed using the vulnerability detector. (3) The vulnerable functions identified during the second phase are dynamically analyzed to remove any false positives. Patchecko compares the functions with known CVE vulnerable functions and associated patches. Then vulnerable

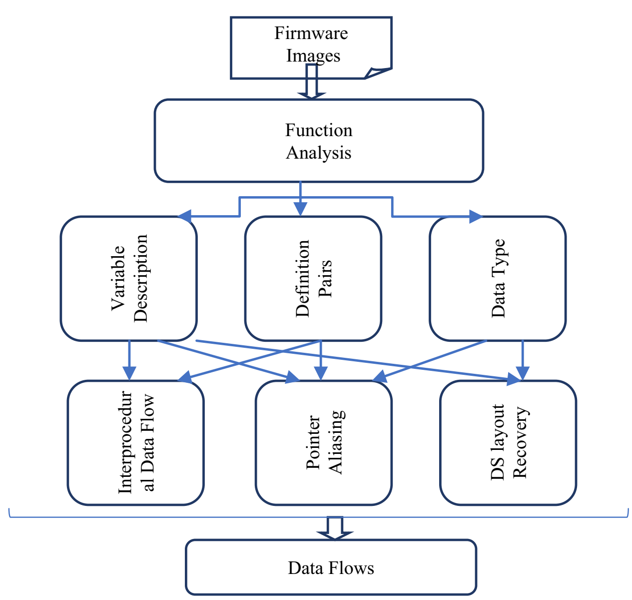


Fig. 13 Dtaint Architecture [90]

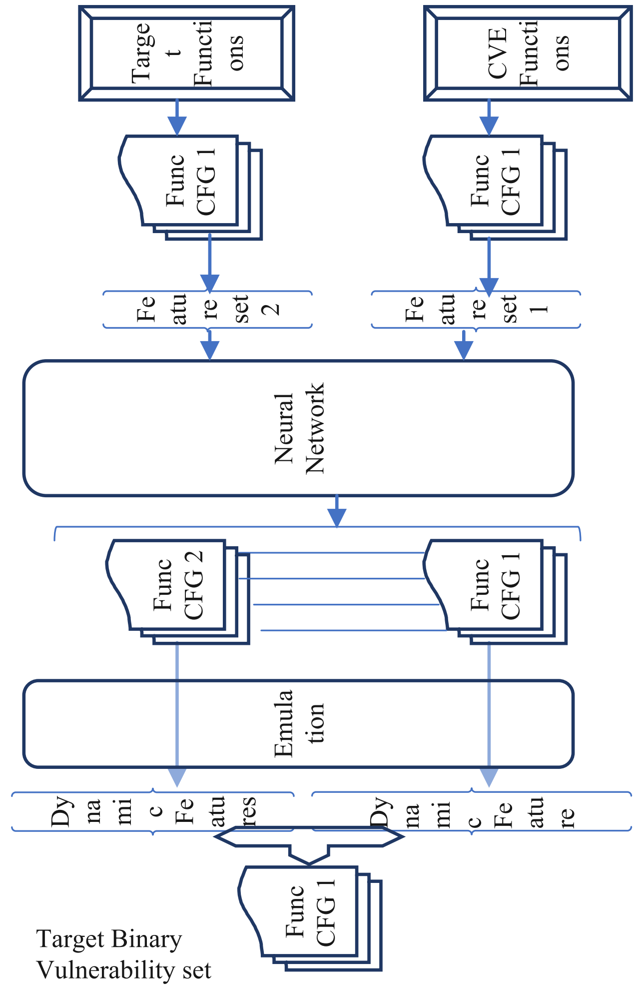


Fig. 14 PATCHEKO workflow diagram [92]

functions are produced as output with associated CVE numbers. Static analysis is used to convert each binary function into a feature vector. A deep learning-based model is used to compare two binary functions based on these feature vectors. After that, a more in-depth dynamic analysis is performed to remove any false positives. It has an accuracy of 93% for properly discovering known vulnerabilities, however, this framework does not identify any unknown vulnerabilities.

2.3.4 Comparative study

In this section, we compare the existing approaches for embedded systems as well as the traditional approaches that can potentially be applied to embedded systems. We further discuss our key observations from this comparative study. As per the comparative study of the frameworks, semantic and structural features based detection produces better results. The number of vulnerabilities produced by semantic and structural features is very high as compared to other techniques. Machine learning which includes deep learning-based approaches shows the best results for the detection of vulnerabilities in cross-architecture platforms. The frameworks are mainly evaluated on the major processor architectures which include x-86, MIPS, and ARM, which constitute the majority of the embedded and IoT devices. QEMU platform is mainly used in the dynamic analysis for runtime evaluation of the embedded firmware. Most of the static analysis tools employ function or pattern matching techniques to mainly detect known vulnerabilities and show poor performance in detecting unknown vulnerabilities. Among the static analysis frameworks, the FirmUp framework has detected a significant number of vulnerabilities across different architectures, whereas in dynamic analysis the framework given by Costin et all [85] has shown better results but across only two major architectures which include ARM, and MIPS. Hybrid analysis frameworks still need to improve in terms of vulnerability detection rate. Different machine learning methods may be explored for improving the hybrid analysis frameworks (Fig. 15).

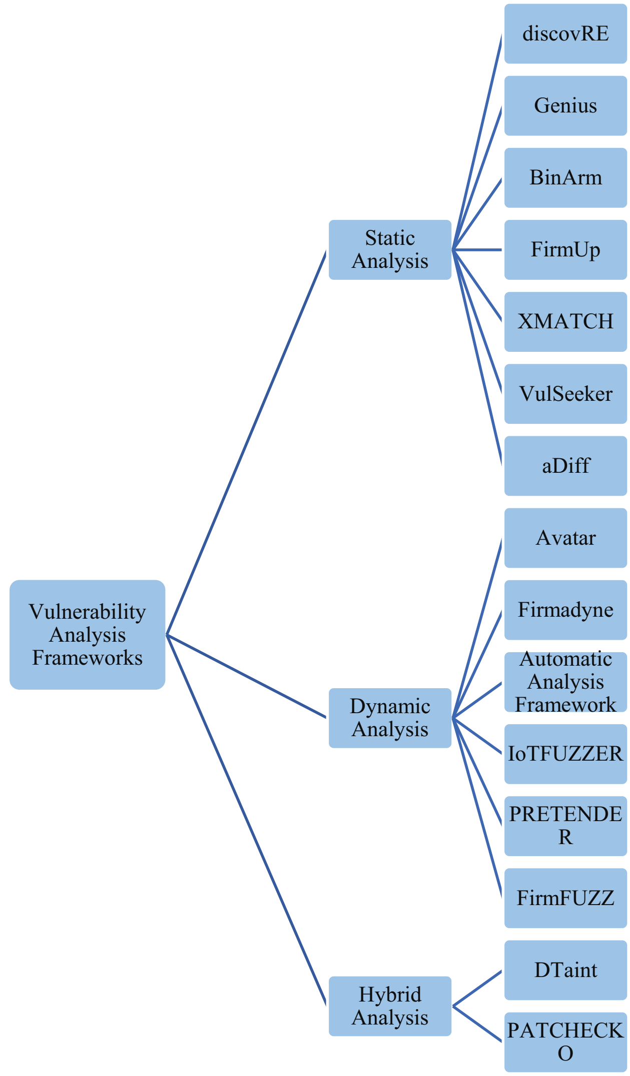


Fig. 15 Hierarchy diagram of vulnerability analysis frameworks

The Table 8 provides an overview of various firmware analysis tools and their characteristics from 2014 to 2020. Several trends can be observed from the data. Over the years, there has been a shift from static analysis to dynamic analysis, including full system emulation and machine learning-based approaches. The number of supported architectures has also increased, accommodating a wide range of devices. Additionally, the number of firmware/devices analyzed has grown significantly, indicating the expanding scope of firmware analysis. Tools like "IoT FUZZER" and "FirmUp" have been designed for dynamic analysis, while "BinArm" and "VulSeeker" focus on static analysis. The development of machine learning and deep learning techniques is evident in tools like "Automatic Analysis," "Genius," and "PATCHECKO." This comprehensive analysis landscape showcases the growing importance of firmware analysis in addressing cybersecurity challenges in the IoT and embedded device domain.

2.3.5 Vulnerability proritization

Vulnerability prioritization plays a pivotal role in crafting an effective cybersecurity strategy, as it empowers organizations to allocate their resources judiciously while addressing the most imminent threats. This process entails a comprehensive evaluation of vulnerabilities, taking into account both their potential impact and exploitability. This systematic approach

Table 8 Comparative analysis of the discussed vulnerability analysis frameworks

Year	Name	Authors	Tools used	Analysis type	Techniques used	Extracted features	Supported architectures	#of firmwares/devices analysed	Vulnerability identification type	Devices/firmware analysed	#Vulnerabilities identified
2014	Avatar	Zaddach et al. [13]	S^2E, QEMU, GDB, OpenOCD	Dynamic Analysis	Memory Access Delegation, Partial Emulation	NS	ARM	3	Known Type	GSM phone, hard disk bootloader, wireless sensor nod	NS
2016	discovRE	Eschweiler et al. [5]	IDA Pro	Static Analysis	Machine Learning, Graph Encoding	Structural, Numerical	X86, ×64, ARM, MIPS	3	Known Type	DD-WRT r21676, Android 4.1.1 ReadyNAS v6.1.6	NS
2016	Firmadyne	D.Chen et al. [84]	QEMU, Metasploit	Dynamic Analysis	Full System Emulation	NS	ARM-LE, MIPS	9486	Known, Unknown Type	NS	14
2016	Automatic Analysis	Costin et al. [85]	RIPS, QEMU	Dynamic Analysis	Full System Emulation	NS	ARM, MIPS	1925	Known, Unknown	NS	225
2016	Genius	Feng et al. [80]	IDA Pro, Near py, Mongo DB	Static Analysis	Machine Learning, Graph Encoding	Structural, Statistical	X86, ARM, MIPS	2	Known Type	DD-WRT r21676, ReadyNAS v6.1.6	154
2017	XMATCH	Feng et al. [76]	IDA Pro	Static Analysis	Conditional Formula Matching	Semantic Features	X86, MIPS	2	Known Type	OpenSSL, BusyBox	5
2018	IoT FUZZER	Chen et al. [86]	Xposed, TaintDriod, Monkeyrunner, EdgeMiner	Dynamic Analysis, Taint Analysis	Fuzzing	NS	NS	17	Known, Unknown Type	Belkin WeMo, TP-Link HS110, D-Link DSP-W215, WD My Cloud (NAS), QNAP TS-212P (NAS), Brother HL-L5100DN, Philips Hue Bridge, WD My Passport Pro (NAS), POVOS PW103 (Humidifier)	15
2018	BinArm 92%	Shirani et al. [77]	IDA Pro, Docker	Static Analysis	Graph Encoding	Structural, Statistical, Instruction-Level	ARM	5	Known Type	NI PMU1_0_11, Honeywell. RTUR150, Schneider Link150, Schneider M251, Ready NAS v6.1.6	93

Table 8 (continued)

Year	Name	Authors	Tools used	Analysis type	Techniques used	Extracted features	Supported architectures	#of firmwares/devices analysed	Vulnerability identification type	Devices/firmware analysed	#Vulnerabilities identified
2018	FirmUp	David et al. [93]	Angr, IDA Pro, Binwalk	Static Analysis	Pairwise Procedure Similarity	Semantic Features	MIPS32, ARM32, PPC32, X86	2000	Known Type	Asus, Netgear, Dlink Routers	373
2018	VulSeeker	Gao et al. [82]	IDAPython	Static Analysis	Deep Neural Networks, Cosine Distance	Numerical Features	X86, X64, MIPS, ARM	4643	Known Type	NS	83
2018	aDiff	Liu et al. [83]	IDA Pro	Static Analysis	Convolution neural network (CNN)	Intra function features, Inter function features, Inter module features	ARM, MIPS and × 86	NS	Known Type	NS	NS
2018	DTaint	Cheng et al. [90]	Binwalk, Angr	Hybrid Analysis	Data Flow Identification	NS	MIPS, ARM	6	Known, Unknown	Dlink DIR-645 1.03, Dlink DIR-890L 1.03, Netgear DGN1000-V1.1.00.46, Netgear DGN2200-V1.0.0.50, Uniview IPC 6201, Hikvision DS-2CD6233F	21
2019	PRETENDER	Gustafson et al. [87]	QEMU	Dynamic Analysis	Re-Hosting, Full System Emulation, Machine Learning	NS	ARM	6	NS	ST Nucleo L152RE, Maxim MAX32600MBED, STM Nucleo F072RB	NS
2019	FirmFuzz	Srivastava et al. [89]	QEMU	Dynamic Analysis	Fuzzing	NS	MIPS, ARM-LE	32	Unknown Type	Trendnet TEW-673GRU, 634GRU, 632BRP, Trendnet TV-IP110WN, IP121WN, Netgear DG834	7
2020	PATCHECKO 93%	Sun et al. [92]	IDA Pro	Hybrid Analysis	Deep Learning	Static Function Features, Dynamic Features	X86, amd64, ARM	2	Known Type	Android Things 1.0, Google Pixel 2 XL 8.0	25

allows for the pinpointing of high-risk areas that demand immediate attention and remediation efforts. Well-established methodologies such as the Common Vulnerability Scoring System (CVSS) serve as a standardized framework for the assessment of vulnerabilities, factoring in elements like base score, temporal score, and environmental score [94]. Furthermore, emerging strategies harness the power of machine learning algorithms and threat intelligence to refine the accuracy of prioritization [95, 96]. Recent scholarly investigations highlight the necessity for dynamic and context-aware vulnerability management tactics [97, 98]. These advancements underscore the evolving landscape of vulnerability prioritization and underscore the paramount importance of integrating state-of-the-art methodologies into an organization's cybersecurity endeavors.

3 Research challenges and open issues

In this section, we discuss the various challenges and issues that are faced in performing vulnerability detection on embedded device binaries.

3.1 Reverse engineering

Reverse engineering of firmware is a very complex task that involves a series of steps with appropriate tools and expertise. Reverse engineering consists of firmware extraction, firmware unpacking, and firmware disassembly. One of the main problems in the extraction of firmware is the use of appropriate hardware and software. Moreover, these extraction tools are very costly and are often very complex and buggy. Embedded devices are designed without any common standards. Lack of standardization in hardware architectures across the wide range of IoT & embedded devices presents a big challenge in the extraction of their firmware.

3.2 Firmware disassembling

Software programs are cross-compiled and deployed on various architecture platforms which puts a huge challenge on the analyst to disassemble and make sense of the different binary instruction formats of specific architectures which have been compiled from the same source code. It is a very challenging task for the researchers to look at all the binary formats for common vulnerabilities.

3.3 Detection accuracy

Obtaining higher accuracy for the detection of vulnerabilities and reducing the false positives is very critical in vulnerability analysis. Out of all the framework types Automatic analysis by Costin et all [85] provides better results in identifying both known and unknown types of vulnerabilities. Accuracy can be improved by tailoring ML & DL algorithms for such problems.

3.4 Scalability

Vulnerability detection at a large scale is a major challenge. IoT & Embedded devices are growing exponentially due to this the vulnerability detection frameworks have to accommodate these ever-growing devices. Testing the embedded firmware in runtime weather on real devices or through emulation tools is very slow and error-prone. The deployment of different architectures and software programs presents a major challenge in vulnerability analysis.

3.5 Vulnerability verification

Verification of the identified vulnerabilities is another problem that researchers are facing. Verifying requires determining the execution path in a firmware that triggers the vulnerability. Due to the limited information of the vulnerability many times it becomes complicated to verify the vulnerability by reproducing the behavior of the system.

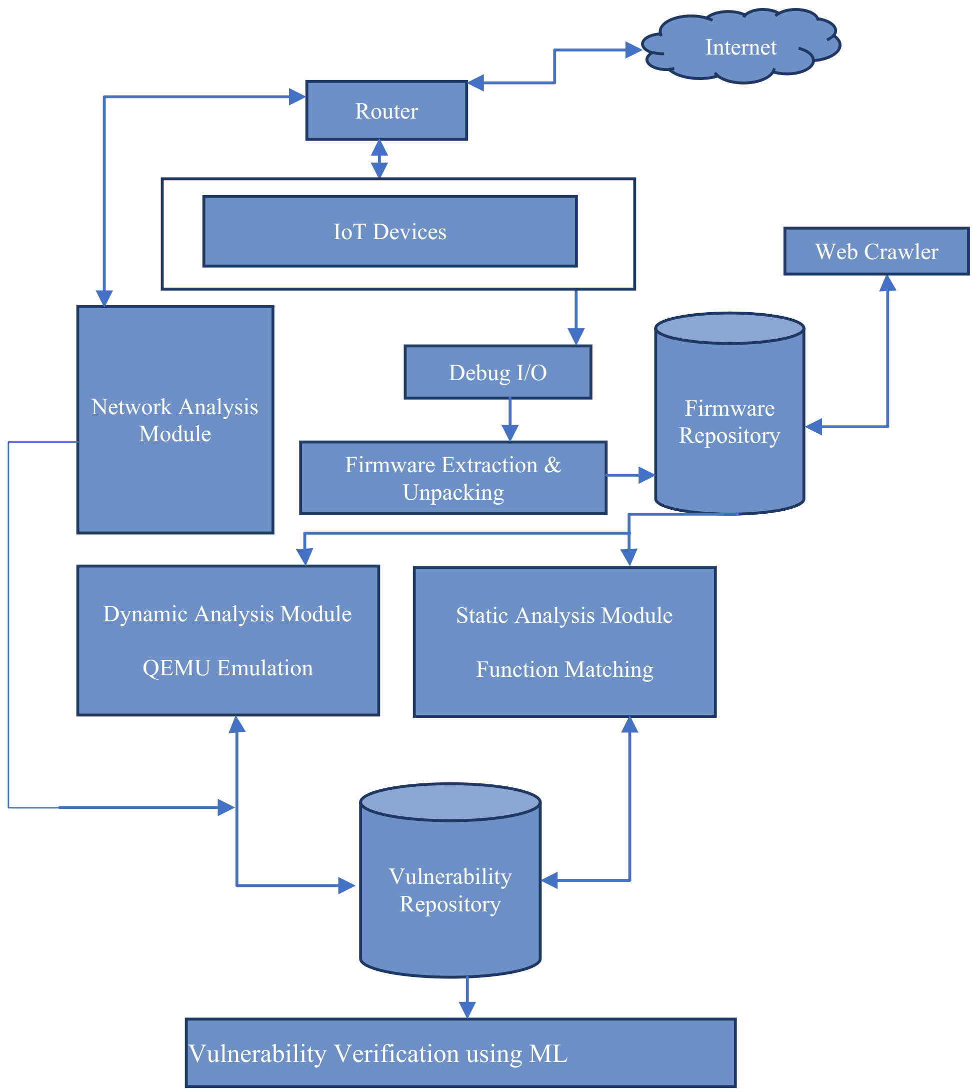


Fig. 16 Proposed vulnerability analysis model

4 Proposed model

We have designed a hybrid vulnerability analysis framework including the testbed. This proposed framework addresses some of the issues present in the already existing frameworks. It addresses the scalability issue by utilizing both the firmware collection methods which include the web crawler approach and extraction using onboard debug ports. Our approach utilizes both dynamic and static analysis techniques for the identification of known and unknown vulnerabilities. QEMU emulator will be used in run-time dynamic analysis of already extracted firmware stored in the firmware repository. If any problem is faced during extraction of firmware using onboard JTAG/UART ports then the firmware will be downloaded using a web crawler as utilized in firmadyne framework. Network Analysis module will be used to check protocol vulnerabilities in IoT devices in runtime using tools that include Wireshark and Metasploit scripts. All the identified vulnerabilities will be stored in the vulnerability repository. The verification process of the vulnerabilities will be carried out by using Machine Learning techniques for the generation of test cases and executing the sequence on emulated firmware or on real devices whichever is feasible. The proposed model will be implemented on a developed testbed.

The proposed testbed as shown in Fig. 16 is a four-layered architecture model. The four layers are the Internet Layer, Control and Monitoring layer, Access Layer, and the Device layer. Internet Layer provides internet connectivity through LAN network using appropriate switches. The monitoring and control layer consists of workstations and a high-performance

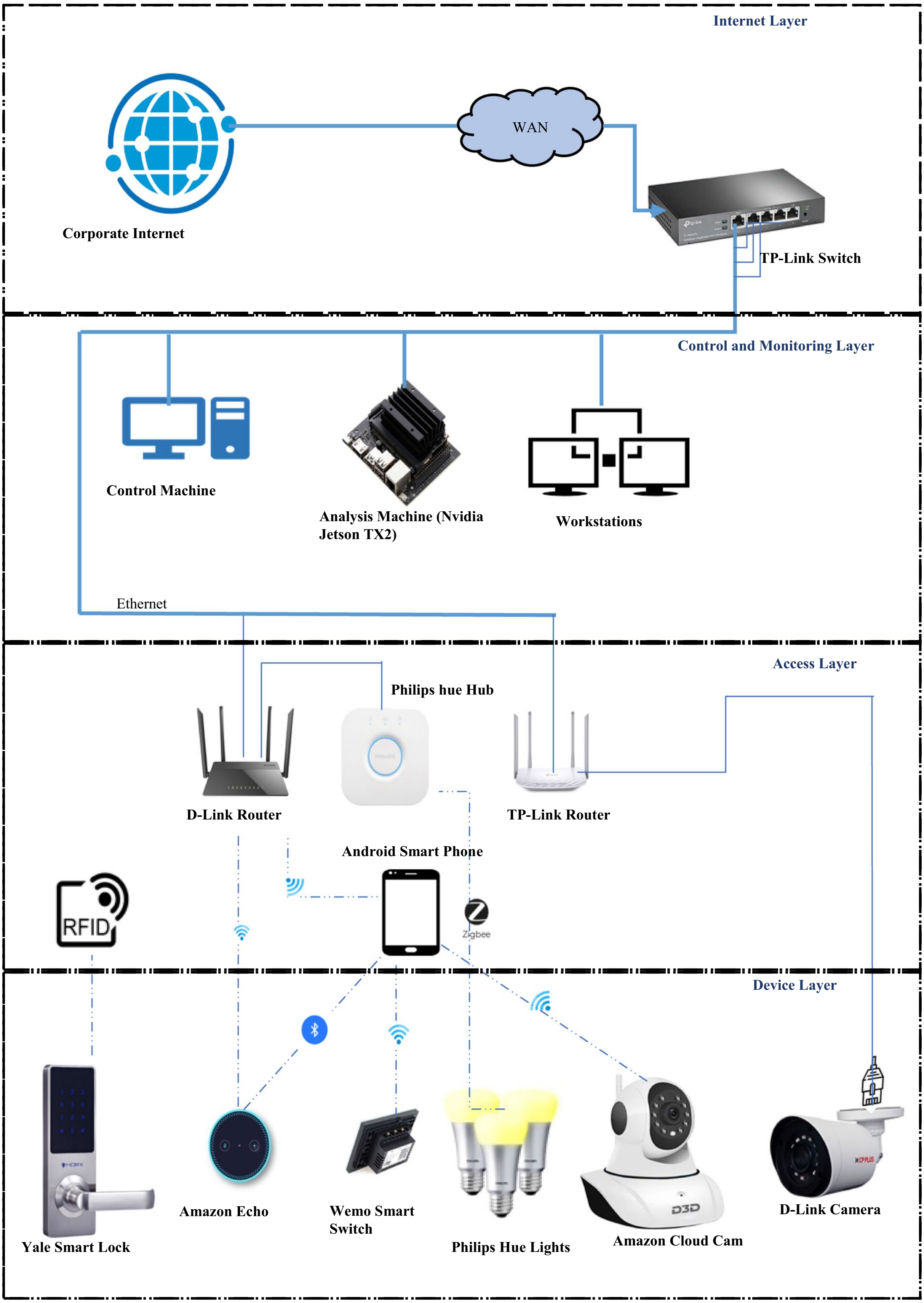


Fig. 17 Proposed vulnerability analysis testbed for IoT and embedded devices

analysis machine that would be used to perform computationally intensive analysis. It also consists of a control machine that would be used to launch scripts and programs necessary for analysis. All the test results will be stored on this machine itself. The Access layer consists of various hubs and routers that connect wirelessly to the IoT devices. The device layer consists of various IoT and embedded devices that are connected to their appropriate hubs and wifi routers (Fig. 17).

5 Conclusion & further work

In this article, we surveyed various types of architectural elements, firmware extraction methods, and various types of vulnerability analysis frameworks for IoT & Embedded devices. We surveyed the major processor architectures of embedded devices. Techniques used to implement static, dynamic, and hybrid analysis was surveyed. A detailed comparison of the vulnerability analysis framework was presented based on various qualitative and quantitative parameters. Finally, we discussed the various challenges in performing vulnerability analysis of IoT devices. A vulnerability analysis model for overcoming some of the challenges is also proposed at the end. As further work, we intend to develop the proposed framework in the lab using various COTS modules. The proposed model will be evaluated based on various parameters on various IoT & Embedded devices available in our institute IoT lab.

In order to solve the lack of standardisation in hardware architectures, future research should focus on the creation of user-friendly and affordable methods for firmware extraction. Innovative methods for deconstructing cross-compiled software programmes and the improvement of machine learning algorithms for vulnerability research are other crucial areas for development. As the IoT ecosystem expands, researchers should concentrate on developing scalable frameworks and procedures for efficient vulnerability verification.

Author contributions SUH and YS wrote the main manuscript, SUH prepared the figures. All authors reviewed the manuscript.

Data availability There is no data to disclose.

Declarations

Competing interests The authors declare no competing interests.

References

1. Antonakakis M et al. (n.d.). Understanding the Mirai Botnet | USENIX. Retrieved September 30, 2021, from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis
2. The reaper botnet could be worse than the internet-Shaking Mirai Ever Was|WIRED. https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/. Accessed 27 Nov 2017.
3. Van Den Broek F, Hond B, Cedillo Torres A. Security testing of GSM implementations. In: Engineering secure software and systems. Springer International Publishing; 2014. p. 179–95.
4. Eschweiler S, Yakdan K, Gerhards-Padilla E. discovRE: efficient cross-architecture identification of bugs in binary code. NDSS; 2017. https://doi.org/10.14722/ndss.2016.23185.
5. Cui, A., Costello, M., & Stolfo, S. J. When Firmware Modifications Attack: A Case Study of Embedded Exploitation; 2013. https://doi.org/10.7916/D8P55NKB
6. Vulnerabilities in FOSCAM IP cameras 2 vulnerabilities in FOSCAM IP cameras. http://www.gartner.com/newsroom/id/3598917. Accessed 14 Sept 2021.
7. Gauthier F, Lavoie T, Merlo E. Uncovering access control weaknesses and flaws with security-discordant software clones. In: Proceedings of the 29th annual computer security applications conference; 2013. p. 209–18. https://doi.org/10.1145/2523649.2523650.
8. Gui Z, Shu H, Kang F, Xiong X. FIRMCORN: vulnerability-oriented fuzzing of IoT firmware via optimized virtual execution. IEEE Access. 2020;8:29826–41. https://doi.org/10.1109/ACCESS.2020.2973043.
9. Chen J, Diao W, Zhao Q, Zuo C, Lin Z, Wang X. I O TF UZZER : discovering memory corruptions in IoT through app-based fuzzing. No. February 2018, 2020.

10. Vasile S, Oswald D, Chothia T. Breaking all the things—a systematic survey of firmware extraction techniques for IoT devices, vol. 11389 LNCS. Cham: Springer International Publishing; 2019. https://doi.org/10.1007/978-3-030-15462-2_12.
11. Abu Waraga O, Bettayeb M, Nasir Q, Abu TM. Design and implementation of automated IoT security testbed. Comput Secur. 2020. https://doi.org/10.1016/j.cose.2019.101648.
12. Costin A, Zaddach J, Francillon A, Balzarotti D. A large-scale analysis of the security of embedded firmwares. In: Proceedings of the 23rd USENIX security symposium; 2014. p. 95–110.
13. Chen DD, Egele M, Woo M, Brumley D. Towards automated dynamic analysis for linux-based embedded firmware; 2017. https://doi.org/10.14722/ndss.2016.23415.
14. Arias O, Wurm J, Hoang K, Jin Y. Privacy and security in internet of things and wearable devices. IEEE Trans Multi-Scale Comput Syst. 2015;1(2):99–109. https://doi.org/10.1109/TMSCS.2015.2498605.
15. Cyr B, Horn W, Miao D, Specter M. Security analysis of wearable fitness devices (fitbit). Massachusetts Institute of Technology; 2014. p. 1–14.
16. Wurm J, Hoang K, Arias O, Sadeghi AR, Jin Y. Security analysis on consumer and industrial IoT devices. In: Proceedings of the Asia and South Pacific design automation conference, ASP-DAC, vol. 25–28; 2016. p. 519–24. https://doi.org/10.1109/ASPDAC.2016.7428064.
17. Li S, Choo KKR, Sun Q, Buchanan WJ, Cao J. IoT forensics: amazon echo as a use case. IEEE Internet Things J. 2019;6(4):6487–97. https://doi.org/10.1109/JIOT.2019.2906946.
18. Ronen E, Shamir A. Extended functionality attacks on IoT devices: the case of smart lights. In: Proceedings—2016 IEEE European symposium on security and privacy, EURO S and P 2016; 2016. p. 3–12. https://doi.org/10.1109/EuroSP.2016.13.
19. OpenWrt Forum Archive. (n.d.). Retrieved June 21, 2021, from https://forum.archive.openwrt.org/viewforum.php?id=10&p=1.
20. iot-fw-extraction/phillips_hue. Retrieved June 20, 2021, from https://github.com/david-oswald/iot-fw-extraction/tree/master/phillips_hue.
21. Hardware Hacking of Accu-Chek Performa Insight. (n.d.). Retrieved June 30, 2021, from https://hackaday.io/project/41162-hardware-hacking-of-accu-chek-performa-insight/details.
22. iot-fw-extraction/accuchek. Retrieved August 20, 2021, from https://github.com/david-oswald/iot-fw-extraction/tree/master/accuchek/
23. Vasile, S., Oswald, D., & Chothia, T. (2019). Breaking all the things—a systematic survey of firmware extraction techniques for IoT devices. Lecture Notes in Computer Science, 11389 LNCS, 171–185. https://doi.org/10.1007/978-3-030-15462-2_12/COVER
24. tencentbladeteam/Exploit-Amazon-Echo. (n.d.). Retrieved June 10, 2021, from https://github.com/tencentbladeteam?tab=repositories.
25. iot-fw-extraction/amazon_echo. Retrieved June 20, 2021, from https://github.com/david-oswald/iot-fw-extraction/tree/master/amazon_echo.
26. Adithyan A, Nagendran K, Chethana R, Gokul Pandy D, Gowri Prashanth K. Reverse engineering and backdooring router firmwares. In: 2020 6th international conference on advanced computing and communication systems, ICACCS 2020; 2020. p. 189–93. https://doi.org/10.1109/ICACCS48705.2020.9074317.
27. Crockett, E. Top IoT Devices. Retrieved October 1, 2021, from https://www.datamation.com/mobile-wireless/75-top-iot-devices-1.html.
28. Most Popular IoT Devices. Retrieved October 1, 2021, from https://www.softwaretestinghelp.com/iot-devices/.
29. Siboni S, et al. Security testbed for internet-of-things devices. IEEE Trans Reliab. 2019;68(1):23–44. https://doi.org/10.1109/TR.2018.2864536.
30. Angrishi, K. (2017). Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets. https://arxiv.org/abs/1702.03681v1.
31. Notra S, Siddiqi M, Habibi Gharakheili H, Sivaraman V, Boreli R. An experimental study of security and privacy risks with emerging household appliances. In: 2014 IEEE conference on communications and network security; 2014. p. 79–84. https://doi.org/10.1109/CNS.2014.6997469.
32. OpenWrt Project: Belkin F7C027. (n.d.). Retrieved September 17, 2021, from https://openwrt.org/toh/belkin/f7c027#bootloader.
33. WebHome U-Boot. Retrieved October 7, 2021, from https://www.denx.de/wiki/U-Boot.
34. Defcon. "All your things are belongs to us". Retrieved July 20, 2021, from https://infocondb.org/con/def-con/def-con-25/all-your-things-are-belong-to-us.
35. Exploitee.rs. Retrieved May 16, 2022, from https://exploitee.rs/.
36. Shwartz O, Mathov Y, Bohadana M, Elovici Y, Oren Y. Opening Pandora's box: effective techniques for reverse engineering IoT devices. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol. 10728 LNCS. Cham: Springer International Publishing; 2018. p. 1–21.
37. JTAGulator®|Grand Idea Studio. http://www.grandideastudio.com/jtagulator/. Accessed 29 Sept 2020.
38. Etemadieh, Z., Heres, C. J., & Hoang, K. (2014). Hacking Hardware With A $ 10 SD Card Reader. 1–17. https://bh2017.exploitee.rs
39. The Shikra|int3.cc. https://int3.cc/products/the-shikra. Accessed 25 Sept 2020.
40. Introduction to attify badge : hacking IoT hardware. https://blog.attify.com/hack-iot-device/. Accessed 25 Sept 2020.
41. Adafruit FT232H breakout—general purpose USB to GPIO, SPI, I2C [USB C & Stemma QT] ID: 2264—$14.95 : adafruit industries, unique & fun DIY electronics and kits.
42. HydraBus v1.0 Specifications|HydraBus. https://hydrabus.com/hydrabus-1-0-specifications. Accessed 23 Sept 2020.
43. Keil ULINK2 Debug Adapter. https://www.keil.com/arm/ulink2/. Accessed 26 Sept 2020.
44. Flyswatter 2|Tin Can Tools. https://www.tincantools.com/product/flyswatter2/. Accessed 15 Sept 2020.
45. Bus Pirate—v3.6a—TOL-12942—SparkFun Electronics. https://www.sparkfun.com/products/12942. Accessed 12 Sept 2020.
46. 1BitSquared—Black Magic Probe V2.1. https://hackerwarehouse.com/product/black-magic-probe-v2/. Accessed 27 Sept 2020.
47. Attify Store—JTAGulator|Attify Store. https://www.attify-store.com/products/jtagulator. Accessed 11 Sept 2020.
48. AVR Dragon. https://www.microchip.com/en-us/development-tool/atavrdragon. Accessed 22 Sept 2020.
49. OpenOCD—Open On-Chip Debugger download|SourceForge.net. https://sourceforge.net/projects/openocd/. Accessed 24 Sept 2020.
50. Universal JTAG library, server and tools download|SourceForge.net. https://sourceforge.net/projects/urjtag/. Accessed 25 Sept 2020.

51. AVRdude GUI download|SourceForge.net. https://sourceforge.net/projects/avrdudegui/. Accessed 24 Sept 2020.
52. EasyJTAG Plus Software|EasyJtag—fastest memory programmer in the word! https://easy-jtag.com/easyjtag-plus-software/. Accessed 24 Sept 2020.
53. Binwalk|Firmware Extraction|ReFirm Labs. https://github.com/ReFirmLabs/binwalk. Accessed 22 Sept 2020.
54. Ghidra. https://github.com/NationalSecurityAgency/ghidra. Accessed 24 Sept 2020.
55. IDA Pro—Hex Rays. https://hex-rays.com/ida-pro/. Accessed 24 Sept 2020.
56. QEMU. https://www.qemu.org/docs/master/. Accessed 24 Sept 2020.
57. Home rampageX/firmware-mod-kit Wiki GitHub. https://github.com/rampageX/firmware-mod-kit. Accessed 24 Sept 2020.
58. radare. https://github.com/radareorg. Accessed 23 Sept 2020.
59. firmadyne: Platform for emulation and dynamic analysis of Linux-based firmware. Retrieved June 19, 2021, from https://github.com/firmadyne/firmadyne.
60. Cortesi, A. binvis.io. Retrieved July 30, 2021, from http://binvis.io/#/.
61. firmwalker: Script for searching the extracted firmware file system for goodies! (n.d.). Retrieved July 26, 2021, from https://github.com/craigz28/firmwalker.
62. FWAnalyzer: a tool to analyze filesystem images. (n.d.). Retrieved August 30, 2021, from https://firmwaresecurity.com/2019/08/07/fwanalyzer-a-tool-to-analyze-filesystem-images/.
63. Fernandes E, Jung J, Prakash A. Security analysis of emerging smart home applications. In: 2016 IEEE symposium on security and privacy (SP); 2016. p. 636–54. https://doi.org/10.1109/SP.2016.44.
64. Ramljak M. Security analysis of open home automation bus system. In: 2017 40th international convention on information and communication technology, electronics and microelectronics (MIPRO); 2017. p. 1245–50. https://doi.org/10.23919/MIPRO.2017.7973614.
65. Hassanzadeh A, Modi S, Mulchandani S. Towards effective security control assignment in the Industrial Internet of Things. In: 2015 IEEE 2nd world forum on internet of things (WF-IoT); 2015. p. 795–800. https://doi.org/10.1109/WF-IoT.2015.7389155.
66. Johnson, C. Securing the participation of safety-critical SCADA systems in the industrial internet of things.(2016). 11–13. https://eprints.gla.ac.uk/130828/.
67. Sajid A, Abbas H, Saleem K. Cloud-assisted IoT-based SCADA systems security: a review of the state of the art and future challenges. IEEE Access. 2016;4:1375–84. https://doi.org/10.1109/ACCESS.2016.2549047.
68. Sachidananda V, Bhairav S, Ghosh N, Elovici Y. PIT: a probe into internet of things by comprehensive security analysis. In: 2019 18th IEEE international conference on trust, security and privacy in computing and communications/13th IEEE international conference on big data science and engineering (TrustCom/BigDataSE); 2019. p. 522–9. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00076.
69. Ferrara P, Mandal AK, Cortesi A, Spoto F. Static analysis for discovering IoT vulnerabilities. Int J Softw Tools Technol Transfer. 2021;23(1):71–88. https://doi.org/10.1007/s10009-020-00592-x.
70. GitHub—nccgroup/VCG: VisualCodeGrepper—code security scanning tool. https://github.com/nccgroup/VCG. Accessed 16 Sept 2021.
71. Cppcheck—a tool for static C/C++ code analysis. https://cppcheck.sourceforge.io/. Accessed 16 Sept 2021.
72. PMD. https://pmd.github.io/. Accessed 16 Sept 2021.
73. USENIX Association. Proceedings of the seventeenth Large Installation Systems Administration Conference (LISA XVII) : October 26–31, 2003 San Diego, CA, USA. USENIX Association; 2003.
74. Qasem A, Shirani P, Debbabi M, Wang L, Lebel B, Agba BL. Automatic vulnerability detection in embedded devices and firmware: survey and layered taxonomies. ACM Comput Surv. 2021. https://doi.org/10.1145/3432893.
75. Feng Q, Zhou R, Xu C, Cheng Y, Testa B, Yin H. Scalable graph-based bug search for firmware images. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security; 2016. p. 480–91. https://doi.org/10.1145/2976749.2978370.
76. Shirani P, Collard L, Agba BL, Lebel B, Debbabi M, Wang L, Hanna A. BINARM: scalable and efficient detection of vulnerabilities in firmware images of intelligent electronic devices. In: Detection of intrusions and malware, and vulnerability assessment; 2018. p. 114–38.
77. David Y, Partush N, Yahav E. FirmUp: precise static detection of common vulnerabilities in firmware. SIGPLAN Not. 2018;53(2):392–404. https://doi.org/10.1145/3296957.3177157.
78. Rocha TA, Martins AT, Ferreira FM. Synthesis of a DNF formula from a sample of strings using Ehrenfeucht-Fraïssé games. Theor Comput Sci. 2020;805:109–26. https://doi.org/10.1016/j.tcs.2019.08.015.
79. Feng Q, Wang M, Zhang M, Zhou R, Henderson A, Yin H. Extracting conditional formulas for cross-platform bug search. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security; 2017. p. 346–59. https://doi.org/10.1145/3052973.3052995.
80. McSema: Static Translation of X86 Instructions to LLVM. www.cs.umd.edu/~awruef
81. Gao, J., Yang, X., Fu, Y., Jiang, Y., & Sun, J. (2018). Vulseeker: A semantic learning based vulnerability seeker for cross-platform binary. ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 896–899. https://doi.org/10.1145/3238147.3240480
82. Liu B et al. αDiff: cross-version binary code similarity detection with DNN. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering; 2018. p. 667–78. https://doi.org/10.1145/3238147.3238199.
83. Zaddach J, Bruno L, Balzarotti D. Avatar: a framework to support dynamic security analysis of embedded systems' firmwares; 2014. http://www.arm.com/community/partners/silicon.php
84. Costin, A., Zarras, A., & Francillon, A. (2016). Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. ASIA CCS 2016 - Proceedings of the 11th ACM Asia Conference on Computer and Communications Security, pp. 437–448. https://doi.org/10.1145/2897845.2897900.
85. Chen J, et al. IoTFuzzer: discovering memory corruptions in IoT through app-based fuzzing. NDSS; 2018. https://doi.org/10.1007/978-3-319-75208-2_1.
86. Gustafson E et al. Toward the analysis of embedded firmware through automated re-hosting. https://github.com/ucsb-seclab/pretender
87. Bellard, F, QEMU, a fast and portable dynamic translator. In USENIX annual technical conference, FREENIX Track (Vol. 41, p. 46). 2005, April. https://www.usenix.org/legacy/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf.

88. Srivastava, P., Peng, H., Li, J., Okhravi, H., Shrobe, H., & Payer, M. (2019). FirmFuzz: Automated IoT Firmware Introspection and Analysis. IoT S and P 2019 - Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things, pp. 15–21. https://doi.org/10.1145/3338507.3358616.
89. Cheng, K., Li, Q., Wang, L., Chen, Q., Zheng, Y., Sun, L., & Liang, Z. (2018). DTaint: Detecting the Taint-Style vulnerability in embedded device firmware. Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, pp. 430–441. https://doi.org/10.1109/DSN.2018.00052
90. Kyatam S, Alhayajneh A, Hayajneh T. Heartbleed attacks implementation and vulnerability. In: 2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT); 2017. p. 1–6. https://doi.org/10.1109/LISAT.2017.8001980.
91. Sun P, Garcia L, Salles-Loustau G, Zonouz S. Hybrid firmware analysis for known mobile and IoT security vulnerabilities. In: 2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN); 2020. p. 373–84. https://doi.org/10.1109/DSN48063.2020.00053.
92. David Y, Partush N, Yahav E. FirmUp: precise static detection of common vulnerabilities in firmware. In: Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems; 2018. p. 392–404. https://doi.org/10.1145/3173162.3177157.
93. GitHub—firmadyne/firmadyne: platform for emulation and dynamic analysis of Linux-based firmware. https://github.com/firmadyne/firmadyne. Accessed 28 Sept 2020.
94. FIRST "Common vulnerability scoring system version 3.1 specification document revision 1"; 2019. p. 1–24. https://www.first.org/cvss/.
95. Fang Y, Liu Y, Huang C, Liu L. FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. PLoS ONE. 2020;15(2): e0228439.
96. Charmanas K, Mittas N, Angelis L. Exploitation of vulnerabilities: a topic-based machine learning framework for explaining and predicting exploitation. Information. 2023;14(7):403.
97. Hashmat F, Abbas SG, Hina S, Shah GA, Bakhshi T, Abbas W. An automated context-aware IoT vulnerability assessment rule-set generator. Comput Commun. 2022;186:133–52.
98. Jung B, Li Y, Bechor T. CAVP: a context-aware vulnerability prioritization model. Comput Secur. 2022;116: 102639.