A Large-Scale Empirical Analysis of the Vulnerabilities Introduced by Third-Party Components in IoT Firmware

Binbin Zhao*
Zhejiang University
Georgia Institute of Technology
USA
binbin.zhao@gatech.edu

Shouling Ji†
Zhejiang University
China
sji@zju.edu.cn

Jiacheng Xu
Zhejiang University
China
stitch@zju.edu.cn

Yuan Tian
University of Virginia
USA
yuant@virginia.edu

Qiuyang Wei
Zhejiang University
China
weiqiuyang@zju.edu.cn

Qinying Wang
Zhejiang University
China
wangqinying@zju.edu.cn

Chenyang Lyu
Zhejiang University
China
puppet@zju.edu.cn

Xuhong Zhang
Zhejiang University
China
zhangxuhong@zju.edu.cn

Changting Lin
Zhejiang University
Binjiang Institute of Zhejiang University
China
linchangting@zju.edu.cn

Jingzheng Wu
Institute of Software, Chinese Academy of Sciences
China
jingzheng08@iscas.ac.cn

Raheem Beyah
Georgia Institute of Technology
USA
rbeyah@ece.gatech.edu

ABSTRACT

As the core of IoT devices, firmware is undoubtedly vital. Currently, the development of IoT firmware heavily depends on third-party components (TPCs), which significantly improves the development efficiency and reduces the cost. Nevertheless, TPCs are not secure, and the vulnerabilities in TPCs will turn back influence the security of IoT firmware. Currently, existing works pay less attention to the vulnerabilities caused by TPCs, and we still lack a comprehensive understanding of the security impact of TPC vulnerability against firmware.

To fill in the knowledge gap, we design and implement FirmSec, which leverages syntactical features and control-flow graph features to detect the TPCs at version-level in firmware, and then recognizes the corresponding vulnerabilities. Based on FirmSec, we present the first large-scale analysis of the usage of TPCs and the corresponding vulnerabilities in firmware. More specifically, we perform an analysis on 34, 136 firmware images, including 11, 086 publicly accessible firmware images, and 23, 050 private firmware images from TSmart. We successfully detect 584 TPCs and identify 128, 757 vulnerabilities caused by 429 CVEs. Our in-depth analysis reveals the diversity of security issues for different kinds of firmware from various vendors, and discovers some well-known vulnerabilities are still deeply rooted in many firmware images. We also find that the TPCs used in firmware have fallen behind by five years on average. Besides, we explore the geographical distribution of vulnerable devices, and confirm the security situation of devices in several regions, e.g., South Korea and China, is more severe than in other regions. Further analysis shows 2, 478 commercial firmware images have potentially violated GPL/AGPL licensing terms.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

IoT firmware, Third-party component, Vulnerability

ACM Reference Format:
Binbin Zhao, Shouling Ji, Jiacheng Xu, Yuan Tian, Qiuyang Wei, Qinying Wang, Chenyang Lyu, Xuhong Zhang, Changting Lin, Jingzheng Wu, and Raheem Beyah. 2022. A Large-Scale Empirical Analysis of the Vulnerabilities Introduced by Third-Party Components in IoT Firmware. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22), July 18–22, 2022, Virtual, South Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3533767.3534366

*This work was partially conducted when Binbin Zhao was at Zhejiang University.
†Shouling Ji is the corresponding author.

ISSTA '22, July 18–22, 2022, Virtual, South Korea

ACM ISBN 978-1-4503-9379-9/22/07.
https://doi.org/10.1145/3533767.3534366

1 INTRODUCTION

The Internet of Things (IoT) has become ubiquitous and offers great convenience to our daily lives [35, 40]. According to a recent report [20], Gartner forecasts that the number of IoT devices will triple from 2020 to 2030. Inevitably, the booming of IoT devices also raises the public's concern about their security risks [50, 52, 58] and several real-world attacks further aggravate this panic. For example, Mirai has compromised millions of IoT devices including IP cameras, DVRs, and routers, to form a botnet [19, 49, 53] and launch DDoS attacks against various online services [17, 38].

Firmware is an integrated software package that booted in IoT devices which serve the essential roles. Nowadays, firmware widely adopts TPCs, e.g., *BusyBox* [4] and *OpenSSL* [14], to accelerate and simplify the development process. However, the wide usage of TPCs is a double-edged sword since a significant number of TPCs have known vulnerabilities that will open up many new attack surfaces to IoT firmware. For example, the Heartbleed vulnerability [31] in *OpenSSL* has greatly affected at least millions of IoT devices. To make things worse, vendors may reuse a set of the same TPCs in different kinds of firmware, which accelerates the spread of potential vulnerabilities. Therefore, it is vital to recognize the vulnerable TPCs used in firmware.

Although a series of research [21–24, 27, 28, 33, 34, 48, 54] has adopted static or dynamic approaches to evaluate firmware security, they are still limited since they pay less attention to the vulnerabilities caused by TPCs in firmware, lack the consideration of non-Linux based firmware, and/or are unscalable on large-scale firmware security analysis. Specifically, first, they lack the analysis of N-days vulnerabilities introduced by TPCs, which may result in more serious problems than unknown vulnerabilities in reality [25]. Second, most of them are limited to analyze the Linux-based firmware, but short of the analysis of monolithic firmware, which is widely adopted in new ubiquitous, low-power embedded systems, e.g., smart homes. Last but not least, their scalability is a challenge when given large-scale firmware tests. For instance, several approaches require real IoT devices in analysis or lots of manual work for configuration, which greatly limits their scalability. Therefore, we still lack a comprehensive understanding of the usage of TPCs in multiple kinds of firmware and the vulnerabilities introduced by them, with a scalable and practical method.

1.1 Challenges

To obtain a comprehensive overview of the TPCs used in firmware and their corresponding vulnerabilities, we have the following key challenges.

Firmware Dataset Construction. To enable our study, the first challenge is to construct a large-scale and comprehensive firmware dataset covering different kinds of firmware from various vendors. Thus we can obtain convincing results of the current security issues of the firmware. Nevertheless, there is no publicly accessible firmware dataset for research. Besides, more and more vendors begin to prohibit the public from downloading firmware and adopt anti-scraping techniques [51] against the hostile web crawler, which in turn greatly increases the data collection difficulty.

Firmware Processing. There are two major challenges in firmware processing. (1) Extract the contained objects, which have essential information for TPC detection, from Linux-based firmware. Though existing tools, e.g., *binwalk* [36], can be used to unpack firmware, they have limitations on dealing with the firmware which has adopted the latest or customized filesystems. (2) Deal with the monolithic firmware, which is widely used in lower-power embedded systems, e.g., smart homes. Monolithic firmware usually lacks a traditional operating system or metadata, e.g., RAM/ROM start address, needed for analysis. Besides, its code, libraries, and data are highly intermixed. Regarding these challenges, yet, existing tools, e.g., *IDA*, cannot deal with the monolithic firmware without extra configuration.

TPC Detection and Vulnerability Identification. We have two major challenges in TPC detection and vulnerability identification. (1) Detect the TPCs at version-level. To accurately identify the corresponding vulnerabilities of TPCs used in firmware, we need to detect them at version-level rather than only at TPC-level since the vulnerabilities of different versions of TPCs might be different. Nevertheless, it is hard to distinguish the same TPCs at version-level since the code difference of different versions might be tiny. Besides, without source code, we can only obtain limited features from firmware for TPC identification. Previous works for firmware analysis [27, 29, 34] cannot meet our requirements since they do not support detecting the TPCs at version-level in firmware. (2) Construct a TPC database. We need a comprehensive and easily usable TPC database that indicates the possible TPCs used in firmware and the vulnerabilities of each version of TPCs. To the best of our knowledge, there is no previous work has built such a TPC database for IoT firmware. It is a challenge to collect as many TPCs as possible and map the vulnerabilities to the TPC versions.

1.2 Methodology

In this paper, we dedicate to providing a comprehensive understanding of the usage of TPCs in firmware and the potential security risks caused by TPCs. Toward this, we develop a scalable and automated framework FirmSec to conduct a large-scale analysis of IoT firmware. Our analysis method is as follows.

First, to solve the challenge of constructing a firmware dataset, we collect firmware images from public sources and private sources, e.g., official website and private firmware repository. **Second**, we customize several plugins for existing tools, e.g., *binwalk* and *IDA*, to address the problem in firmware processing. The customized tools support unpacking and disassembling both Linux-based firmware that utilizes popular and uncommon filesystems, and monolithic firmware. **Third**, we propose a new detection strategy to identify the TPCs used in firmware. The main idea behind our strategy is to leverage two kinds of features, syntactical features, and control-flow graph features, that are extracted from the TPC and firmware. We then search the corresponding vulnerabilities based on versions check [18, 56, 57], which relies on our TPC database. Based on the results, FirmSec will generate a report for each firmware image which indicates its potential risks. Moreover, the report will provide a series of suggestions for fixing the vulnerabilities.

To provide more in-depth insights, we conduct further studies to explore the security status quo of the IoT ecosystem from four aspects. First, we evaluate the vulnerability of the firmware of different kinds and from different vendors. Second, we explore the geographical difference of the security of IoT devices. Next, we

explore the outdated TPC problem in firmware. Finally, we analyze the TPC GPL/AGPL license violations in firmware.

1.3 Contributions

We summarize our main contributions below.

• We construct so far the largest firmware dataset, which includes 11,086 publicly accessible firmware images and 23,050 private firmware images. It contains 35 kinds of firmware, with most of them rarely studied in previous works. To facilitate future research, we will open-source this dataset at https://github.com/BBge/FirmSecDataset.

• We propose FirmSec, a scalable and automated framework to analyze the TPCs used in firmware and identify the corresponding vulnerabilities. FirmSec has 91.03% of precision and 92.26% of recall in identifying the TPCs at version-level in firmware, which significantly outperforms state-of-the-art works, e.g., *Gemini* [54], *BAT* [37], *OSSPolice* [30].

• We conduct the first large-scale analysis of the vulnerable TPC problem in firmware. We identify 584 TPCs and detect 429 CVEs in 34,136 firmware images. According to the results, we reveal the widespread usage of vulnerable and outdated TPCs in IoT firmware. Moreover, we confirm the geographical difference in the security of IoT devices where several regions, e.g., South Korea and China, are in a more severe situation than other regions. Finally, we discover 2,478 firmware images potentially violate GPL/AGPL licensing terms, which may lead to lawsuits.

2 DESIGN AND IMPLEMENTATION

FirmSec is designed to automatically identify the TPCs used in firmware and detect the corresponding vulnerabilities. As shown in Figure 1, FirmSec mainly contains three components: *firmware collection*, *firmware preprocessing* and *firmware analysis*. The firmware collection module is mainly designed to collect firmware from different sources. Next, the firmware preprocessing module will process the collected firmware in three steps: 1) filter out the non-firmware files from the raw dataset; 2) identify the necessary information of firmware; and 3) unpack and disassemble firmware. Finally, the firmware analysis module will detect the TPCs at version-level in firmware according to the syntactical features and control-flow graph features extracted from the TPC and firmware. The corresponding vulnerabilities will be recognized through versions check according to our TPC database. The implementation behind each component is discussed in the following subsections.

2.1 Firmware Collection

The firmware collection module is responsible for collecting firmware to construct a large-scale raw dataset. To solve the problem of limited firmware resources, we implement a web crawler to collect firmware from three sources: 1) Official website; 2) FTP site; 3) Community. We collect a part of firmware images of several vendors, e.g., Xiaomi, from the community, including the related forums and GitHub repositories; 4) Private firmware repository. We obtain the permission to access the private firmware repository of a world-leading company that mainly focuses on smart homes. In this paper, we use TSmart [1] to represent this company. TSmart's

[1]The anonymized name TSmart represents a world-leading company that mainly focuses on smart homes.

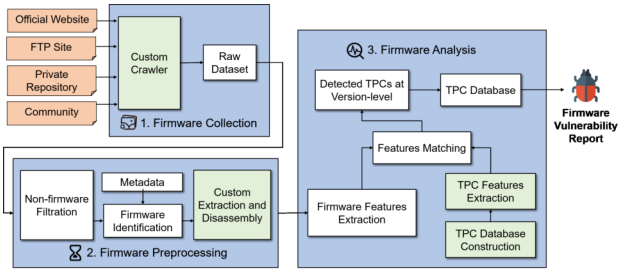


Figure 1: Architecture of FirmSec. Green submodules require some manual work during the initialization of FirmSec, while other submodules can be automatically executed.

firmware repository contains tens of thousands of firmware images of smart homes manufactured by hundreds of vendors, e.g., Philips, and never be studied.

2.2 Firmware Preprocessing

In this part, we mainly preprocess the collected firmware by three steps: firmware filtration, firmware identification, and firmware extraction and disassembly.

Firmware Filtration. The first step is to exclude the non-firmware files from the raw dataset. First, we filter the obvious non-firmware files, e.g., .txt files, through suffix matching. Second, we adopt Binary Analysis Next Generation (BANG) [3], which supports recognizing 136 kinds of files, to get rid of other non-firmware files, e.g., Android Dex. We apply it to the remaining collected files and remove the non-firmware files based on the returned file types.

Firmware Identification. The second step is to identify the necessary information of firmware, e.g., operating systems, architectures, and filesystems, which is crucial for further analysis. We obtain the missed information of firmware based on two steps. First, we extract the information from the metadata files, which are crawled with the firmware. The metadata file describes the firmware, including its corresponding device model, category, and so on. The metadata file is usually in the same archive with firmware. Second, we adopt *binwalk* to scan all firmware images to obtain their operating systems, architectures, and filesystems.

Firmware Extraction and Disassembly. The third step is to unpack and disassemble the firmware for further analysis.

•*Firmware extraction.* The contained objects in firmware have many syntactical features, which are vital for further TPC detection. Besides, the linked libraries extracted from firmware are the main sources for creating the TPC database. Therefore, it is necessary to conduct firmware extraction before we perform further analysis. Nevertheless, existing tools more or less have issues, e.g., incomplete extraction, when extracting the firmware which adopts the latest or customized filesystems. The main challenge here is to extract the filesystems in firmware thoroughly without recursively extracting compressed data. To address this challenge, we equip *binwalk* with a series of plugins. First, we analyze the filesystems used in the collected firmware and find the original *binwalk* mainly has issues on dealing with three popular systems: SquashFS, JFFS2, and YAFFS. Then, we utilize *sasquatch* [1] and *jefferson* [11] to replace the *unsquashfs* and *jffsdump* in *binwalk*. These two plugins support

many vendor-specific SquashFS and JFFS2 implementations. Finally, we implement a new plugin based on *yaffshiv* [2] for *binwalk* to unpack YAFFS, which supports the brute-forcing of YAFFS build parameters, e.g., page size, and extracting the contained objects even after the filesystem has run many rounds.

•*Firmware disassembly.* The control-flow graph (CFG) in firmware contains the essential information to identify the TPCs in firmware. To obtain the CFGs, we need to disassemble the firmware first. However, existing disassemblers are unable to handle the monolithic firmware without extra configuration. For instance, both native *IDA* and *GHIDRA* [16] cannot deal with many monolithic firmware images since they adopt uncommon processors and discard some necessary information, e.g., the RAM/ROM start address, required for analysis. The main challenge here is to recover the missing information based on the known limited information of the monolithic firmware. To solve this challenge, we first analyze the processors used in monolithic firmware to recover the missing information and then implement a series of customized plugins for *IDA* to load and disassemble the monolithic firmware. More specifically, first, we analyze the processor types, e.g., ESP8266 [47], used in the collected monolithic firmware. Second, we collect the corresponding reference manual or datasheet for each processor. The reference manual and datasheet can provide us three kinds of useful information: the core, memory map, and interrupt vector table of the processor. We can figure out the exact instruction sets to disassemble the firmware according to the core, and retrieve the RAM/ROM start address used for loading firmware from the memory map. Moreover, we can obtain the firmware start address from the interrupt vector table. Finally, we implement 7 plugins, which correspond to 7 kinds of processors used in monolithic firmware, for *IDA* based on the recovered information. The plugins enable *IDA* to load and disassemble the monolithic firmware automatically.

2.3 Firmware Analysis

It is a common practice to use the versions check [18, 56, 57] to identify the corresponding vulnerabilities of TPCs. In FirmSec, we also adopt this strategy to analyze firmware. The firmware analysis module includes three submodules: 1) *TPC database construction*, 2) *TPC detection*, and 3) *vulnerability identification*.

TPC Database Construction. Our analysis strategy requires a TPC database that contains the possible TPCs used in firmware and the detailed vulnerability information for each TPC version. Nevertheless, unlike the Maven Repository [12] that indicates the TPCs used in Android apps, there is no such a publicly accessible database for IoT firmware. The main challenge here is to collect as many TPCs as possible from reliable sources and map the vulnerabilities to the TPC versions. To overcome this challenge, we first collect the possible TPCs used in IoT firmware from four sources: 1) linked libraries extracted from the firmware; 2) open-source IoT projects; 3) SDKs from multiple IoT platforms, e.g., AWS IoT; and 4) a shortlist of TPCs from TSmart. Second, we leverage the *cve-search* [7], a professional CVE search tool, to query the TPCs from the CVE database [5], and implement a script to query the NVD [13] and CVE Detail [6] to collect the TPC CVEs. With these two methods, we can get the CVEs that correspond to different versions of TPCs. The constructed TPC database for IoT firmware has the following fields: TPCs, licenses, versions, CVEs, CVSS score [8], and CVE description. We finally collect 1,191 TPCs.

Table 1: Block-level and Function-level Attributes.

Type	Attribute Name	FirmSec	*Gemini* [54]
Block-level	String Constants	✓	✓
	Numeric Constants	✓	✓
	No. of Transfer Instructions	✓	✓
	No. of Calls	✓	✓
	No. of Instructions	✓	✓
	No. of Arithmetic Instructions	✓	✓
	No. of Offspring	✓	✓
	Betweenness	✓	✓
Function-level	No. of Basic Blocks	✓	✗
	No. of Edges	✓	✗
	No. of Variables	✓	✗

TPC Detection. To precisely detect the TPCs used in firmware at version-level is vital in our analysis since we will use the exact version of the detected TPC to confirm its vulnerabilities. Nevertheless, it is difficult to distinguish the same TPCs at version-level since we can only obtain limited information from firmware, and the code difference of different versions might be tiny. To address this challenge, we implement a novel TPC detection method based on two kinds of features: syntactical features and CFG features, which are hardly changed between the source files and binaries. We first extract the above two features from TPCs and firmware. Next, we leverage the edit distance [10, 56], a widely used method to measure the similarity between two strings, and ratio-based matching to calculate the similarity of syntactical features and use customized *Gemini* [54] to compare the CFG features. Based on the comparison results, we finally confirm the usage of TPCs at version-level. The workflow is as follows.

•*TPC feature extraction.* **First**, we implement a parser to extract the syntactical features from the C/C++ source files of TPCs. The syntactical features include the string literals (e.g., unique string), function information (e.g., function names and function parameter types). For each kind of TPC, we summarize the common syntactical features in its all versions, which are regarded as sharing syntactical features. We then identify the specific syntactical features in each version of the TPC, which are regarded as unique syntactical features. **Second**, we extract the attributed control-flow graphs (ACFGs) [34, 54] from each version of TPCs. Each vertex in an ACFG is a basic block labeled with a set of attributes. Except for the block-level attributes used in *Gemini* [54], we also use three function-level attributes, as shown in Table 1. The function-level attributes provide more detailed information on the structure of CFGs, which are ignored by *Gemini*.

•*Firmware feature extraction.* **First**, we extract the same types of syntactical features as the previous step from firmware. Though we successfully disassemble the Linux-based firmware and monolithic firmware, many function names are unrecognized by *IDA*, especially in monolithic firmware. To address this problem, we equip *IDA* with a large number of signature files of TPCs to identify the function names in disassembled files. The signature files are mainly collected from the Internet, e.g., GitHub projects, and we also manually

generate a part of signature files. **Second**, we extract the ACFGs from the disassembled firmware with the same attributes as those extracted from TPCs. However, the original extraction tool used by *Gemini* cannot extract the ACFGs from monolithic firmware since it cannot disassemble monolithic firmware. Therefore, we customize the extraction tool by integrating our firmware disassembly module.

•*Syntactical feature matching.* In this step, we conduct syntactical feature matching to identify the TPCs. Unfortunately, the direct features matching, e.g., regex, will cause low precision and recall. Design a new matching method with high precision and recall is the main challenge in this step. To address this challenge, we utilize the edit distance and ratio-based matching to measure the similarity between the TPC and firmware. We use $D(S_{TPC}, S_{Firmware})$ to represent the edit distance between the syntactical features from TPCs and firmware. If $D(S_{TPC}, S_{Firmware})$ exceeds the given threshold α, we regard the features are matched. We then calculate the distance of each extracted syntactical feature and record the number of matched features. Next, we calculate the ratio of matched features to all features extracted from the TPC, which can be represented as $\frac{S_{TPC} \cap S_{Firmware}}{S_{TPC}}$. If $\frac{S_{TPC} \cap S_{Firmware}}{S_{TPC}}$ exceeds the given threshold β, we regard the TPC is matched. Given some firmware may adopt the partially built TPCs, the ratio-based matching can improve the precision under these circumstances. According to the evaluation results in Section 3, we finally set the threshold of $D(S_{TPC}, S_{Firmware})$ as 0.74 and the threshold of $\frac{S_{TPC} \cap S_{Firmware}}{S_{TPC}}$ as 0.52. Based on the above matching strategy, we then leverage the sharing syntactical features for TPC-level identification and use the unique syntactical features for version-level identification. We mark the matched results by syntactical features as R_{Syntax}.

•*CFG feature matching.* In this step, we leverage the customized *Gemini* [54] to conduct the CFG feature matching. The original *Gemini* is designed to detect specific vulnerabilities, e.g., *OpenSSL* Heartbleed vulnerability, in firmware. It first extracts CFGs from the vulnerability and firmware respectively. Then, *Gemini* converts all CFGs to ACFGs, which are noted as Vul_{ACFG} and $Firm_{ACFG}$. The ACFG is represented as an eight-dimensional vector, which consists of eight block-level attributes as shown in Table 1. Next, it utilizes an embedding network, which is based on *struct2vec* [26], to calculate the cosine similarity between these ACFGs, which is noted as $Cosine(Vul_{ACFG}, Firm_{ACFG})$. Finally, it lists the top-K similar functions in firmware according to $Cosine(Vul_{ACFG}, Firm_{ACFG})$.

Nevertheless, the original *Gemini* cannot directly apply to TPC detection. In our task, we want to confirm the similarity between the TPC and firmware rather than the similarity of individual functions. Given each TPC has many ACFGs, the similarity of a single ACFG cannot determine the similarity between the TPC and firmware. The main challenge here is to design a method to aggregate the similarity of each ACFG in the TPC to represent the final similarity between the TPC and firmware. To address this challenge, we normalize and aggregate the similarity of each ACFG based on the weight of the corresponding CFG. We consider that if the CFG is more complicated, its internal logic will also be more complicated. Therefore, the high-complexity CFGs in the TPC are more likely to have a larger difference from the CFGs in other TPCs than low-complexity CFGs, which are more suitable as features to identify TPCs. Based on the above analysis, we give a high weight to the complex CFGs. We use cyclomatic complexity (CC) [9] to evaluate the complexity of a CFG. CC is calculated as follows:

$$CC = e - b + 2 \tag{1}$$

where e is the number of edges in the CFG, and b is the number of basic blocks in the CFG. Next, we calculate the ratio of the complexity of each CFG to the complexity of all CFGs in the TPC as the weight of each CFG, which is defined as follows:

$$Weight(CFG) = \frac{CC(CFG)}{\sum_{i=0}^{N} CC(CFG_i)} \tag{2}$$

where N is the number of CFGs in the TPC. Finally, we aggregate the similarity of each ACFG to represent the similarity of the TPC and firmware, which is denoted as follows:

$$Sim(TPC, Firmware) = \sum_{i=0}^{N} Sim(ACFG_i) * Weight(CFG_i) \tag{3}$$

where $Sim(ACFG_i)$ is the similarity between each ACFG in the TPC and the most similar function in firmware. With the above strategy, we successfully apply the customized *Gemini* to our task. We regard the TPC as matched if $Sim(TPC, Firmware)$ exceeds the threshold γ. We train the customized *Gemini* on Dataset I and then set threshold as 0.64, which are described in Section 3. We mark the matched results by CFG features as R_{CFG}.

We finally take the union of R_{Syntax} and R_{CFG} as our final results, which can be represented as $R_{Syntax} \cup R_{CFG}$. For instance, the identification result of sample.bin in R_{Syntax} is: *OpenSSL unknown, uClibc 0.9.32.1*, and in R_{CFG} is: *OpenSSL 0.9.8, uClibc 0.9.32.1*. Then, $R_{Syntax} \cup R_{CFG}$ should be *OpenSSL 0.9.8, uClibc 0.9.32.1*.

Vulnerability Identification. In this step, we leverage versions check to identify the vulnerabilities of the detected TPCs in firmware based on our TPC database. The database includes the CVEs that correspond to different versions of TPCs. Therefore, we implement a script to automatically query the database with the TPCs and the corresponding versions (e.g., *OpenSSL 0.9.8*), and record the returned vulnerability information. We need to clarify that some vulnerabilities may cannot be exploited since it is possible that some of the vulnerable code cannot be reached due to other remedies, such as disabling certain configuration options or performing some checks to prevent its use. Therefore, we regard identified vulnerabilities as potential vulnerabilities. We finally generate a report for the tested firmware, which indicates its potential risks and provides a series of suggestions for fixing the vulnerabilities.

3 SYSTEM EVALUATION

In this section, we first introduce the composition of our dataset. Next, we evaluate the performance of FirmSec, and compare it with three state-of-the-art works: *Gemini* [54], *BAT* [37], *OSSPolice* [30].

3.1 Dataset Composition

Based on the firmware collection module, we initially collect a total of 35, 978 firmware images varying from 13 vendors which involve 35 kinds of devices. More specifically, 12, 913 firmware images are crawled from the Internet and 23, 065 firmware images are obtained from TSmart. Our dataset actually involves hundreds of vendors since the private firmware images from TSmart are manufactured by hundreds of vendors. TSmart provides a platform that can enable the devices from various vendors to become smart products. We use TSmart as the vendor of private firmware for convenience.

It's not fair comparison because those tools didn't support most of the improvements done in the paper. They could have compared by adding their system improvements to this tool then it would be interesting to see if how their approach is effective?

Table 2: Dataset Composition.

Vendor	Category	# Download Firmware	# Valid Firmware	SquashFS	CramFS	JFFS2	Unknown Filesystems	ARM	MIPS	X86	Unknown ARCH	Linux-based	Non-Linux based
Xiongmai	Camera	1,038	520	-	45	-	475	326	-	-	194	520	-
Tomato-shibby	Router	642	230	230	-	-	-	168	-	-	62	230	-
Phicomm	Router	107	107	73	-	-	34	30	25	-	52	107	-
Fastcom	Router	200	149	13	-	-	136	53	34	-	62	149	-
	Unknown	10	10	6	-	-	4	-	4	-	6	10	-
Trendnet	Camera	492	477	8	44	-	425	274	100		103	477	-
	Router	463	336	261	-	-	75	119	91	-	126	336	-
	Switch	162	162	93	-	18	51	116	17	-	29	162	-
	Unknown	168	106	12	18	4	72	81	2	-	23	106	-
Xiaomi	Router	21	21	21	-	-	-	8	9	-	4	21	-
TP-Link	Camera	384	319	242	-	35	-	170	145	-	4	319	-
	Router	996	820	709	-	19	92	143	638	-	39	820	-
	Switch	270	270	22	-	-	248	110	43	-	117	270	-
	Unknown	48	48	20	-	1	27	19	16	-	13	48	-
D-Link	Camera	360	360	17	11	-	332	192	106	-	62	360	-
	Router	555	552	401	-	2	149	268	109	6	169	552	-
	Switch	695	545	198	-	-	347	312	18	8	207	545	-
	Unknown	91	91	4	8	5	74	25	7	1	58	91	-
Hikvision	Camera	158	139	-	47	1	91	131	-	-	8	139	-
Foscam	Camera	113	113	-	-	-	113	53	-	-	60	113	-
Dahua	Camera	419	419	9	19	-	341	260	2	1	156	419	-
TSmart	Camera	326	326	88	123	43	72	191	45	-	90	296	30
	Smart Switch	2,053	2,053	1	-	-	2,052	267	2	-	1,784	29	2,024
	Sweeper	33	33	-	-	-	33	10	2	-	21	25	8
	Light	8,089	8,089	-	-	1	8,088	617	-	-	7,472	12	8,077
	General	871	856	2	9	44	801	212	2	-	642	776	80
	Plugin	8,294	8,294	-	-	-	8,294	478	-	-	7,816	14	8,280
	Heater	361	361	-	-	-	361	-	-	-	361	-	361
	Blueteeth Light	1,000	1,000	-	-	-	1,000	637	-	-	363	-	1,000
	Air	517	517	-	-	-	517	96	-	-	421	-	517
	Curtain	195	195	-	-	-	195	8	-	-	187	-	195
	Lock	68	68	-	-	-	68	-	-	-	68	-	68
	Freezer	44	44	-	-	4	40	4	-	-	40	4	40
	Air Purifier	24	24	-	-	-	24	-	-	-	24	-	24
	Humidifier	12	12	-	-	-	12	-	-	-	12	-	12
	Dehumidifier	57	57	-	-	-	57	-	-	-	57	-	57
	Heat Controller	49	49	-	-	-	49	-	-	-	49	-	49
	Fan	14	14	-	-	-	14	-	-	-	14	-	14
	Washer	4	4	-	-	-	4	4	-	-	-	-	4
	Gateway	60	60	3	-	-	57	14	14	-	32	3	57
	Others	994	994	42	21	22	909	337	6	-	651	97	897
OpenWrt	Router	5,585	5,292	3,903	-	41	1,348	2,415	194	10	2,673	5,292	-

We list the detailed composition of our dataset as shown in Table 2. After data filtration, we get rid of 1,842 non-firmware files and finally obtain 34,136 valid firmware images across 13 vendors including 11,086 publicly accessible firmware images and 23,050 private firmware images from TSmart. Our dataset includes 35 kinds of known IoT devices and a part of unknown IoT devices. 2,694 (7.9%) firmware images are used in camera, 7,293 (21.3%) firmware images belong to router, 1,191 (3.5%) firmware images are deployed on switch, 23,050 (67.5%) firmware images from TSmart have been equipped on smart homes, and 255 (0.7%) are unknown. Apart from the vendors and devices mentioned above, our dataset also covers several instruction sets, of which ARM (23.9%) takes the majority and MIPS follows (4.9%). SquashFS, CramFS, and JFFS2 are three popular filesystems included in the dataset. Most of the unknown filesystems actually belong to the above three filesystems based on our further analysis. Vendors customize the above filesystems in firmware, which also changes the magic number of the original filesystems used for identification. Moreover, 12,342 (36.2%) firmware images are Linux-based and 21,794 (63.8%) firmware images are non-Linux based.

3.2 Evaluation

Ground-truth Dataset Construction. We first build three ground-truth datasets: (1) Dataset I for training the customized *Gemini* and evaluating the accuracy of the model; (2) Dataset II for choosing the appropriate thresholds used in FirmSec; (3) Dataset III for evaluating the accuracy of FirmSec in detecting the TPCs at TPC-level and version-level in firmware. Dataset I includes the ACFGs we extracted from 1,192 TPCs in our TPC database. To be consistent with the settings of *Gemini*, we compile each TPC into 12 different versions, including three different platforms (MIPS, X86, ARM) and four different optimization levels (O0-O3). Dataset II and Dataset III have a labeled mapping of firmware to TPCs usage for ground truth. To the best of our knowledge, there are no such datasets available from previous works. Given it is difficult to know the specific TPCs used in commercial firmware, we finally use Tomato-shibby and OpenWrt firmware images to build our datasets since

Table 3: Comparison of FirmSec, *Gemini*, *BAT*, and *OSSPolice*.

Tools	TPC-level		Version-level	
	Precision	Recall	Precision	Recall
FirmSec	92.09%	95.24%	91.03%	92.26%
Syntax-based	92.38%	86.29%	91.47%	81.66%
CFG-based	93.72%	82.76%	94.65%	80.90%
Gemini [54]	89.60%	74.19%	90.78%	71.73%
BAT [37]	70.74%	56.38%	NA	NA
OSSPolice [30]	86.63%	71.85%	82.51%	67.05%

Did selecting a firmware images from a vendor for Dataset 2 and 3, which are both linux based, affected the results? It would have been nice to see some diverse firmware images used for threshold setting. Because of this we can't clearly tell how good is their approach for non-linux firmwares?

they have source code and clearly indicate the adopted TPCs with exact versions in the configuration files. For Dataset II, we randomly collect 200 Tomato-shibby firmware images from our dataset, which include 17,918 TPC-version pairs (73 different TPCs with 211 different versions). For Dataset III, we randomly select 300 OpenWrt firmware images from our dataset, which include 19,645 TPC-version pairs (92 different TPCs with 194 different versions). We use the two datasets to perform the threshold selection and evaluation respectively to avoid bias.

Model Accuracy. We split Dataset I into three subsets for training, validation, and testing respectively according to the ratio of 6:2:2. We train the customized *Gemini* for 100 epochs based on the original *Gemini*'s training process. The model that achieves the best AUC (Area Under the Curve) on the validation set during the 100 epochs is saved. We finally test the model on the testing set and the AUC is 0.953. We also retrain the original *Gemini* with the same process, which AUC is only 0.912.

Threshold Selection. Our final results are the union of the results that matched by syntactical features and CFG features. We do not directly use the thresholds when the respective method achieves the optimal performance since the union results may not reach the best at this time. We take three thresholds as a whole and utilize the true positive rate (TPR) at version-level as the metric to select the appropriate thresholds. We combine the three thresholds and their corresponding TPR as a four-dimensional vector: [α, β, γ, TPR]. Each threshold ranges from 0.01 to 1.00. We select the thresholds when the TPR reaches the highest. Based on our experiment, FirmSec achieves the highest TPR (91.47%) when the thresholds of $D(S_{TPC}, S_{Firmware})$, $\frac{S_{TPC} \cap S_{Firmware}}{S_{TPC}}$, and $Sim(ACFG_{TPC}, ACFG_{Firmware})$ are 0.74, 0.52, and 0.64, respectively. We leverage the above thresholds for the following evaluation and analysis.

Evaluation Results. We evaluate the detection accuracy of FirmSec on Dataset III at TPC-level and version-level with two evaluation metrics: precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$). The false positives represent the TPCs and versions that we wrongly identified, and the false negatives represent the TPCs and versions that we do not find. Though we have constructed the ground-truth dataset, we still manually check the true positives, false positives, and false negatives to verify the precision and recall. Because we consider there may exist some corner cases that will influence our results. As shown in Table 3, FirmSec achieves 92.09% precision, 95.24% recall at TPC-level, and 91.03% precision, 92.26% recall at version-level. We also list the syntax-based matching results and the CFG-based matching results when the TPR of each method reaches the highest. We find that the precision of the union results is very close to the respective methods, but the recall rate is much higher. We explore the reasons that why the union results have a great performance than the separate methods. First, the true positives of the two methods have non-overlapping parts, thus reducing the union results' false negatives. For instance, the syntax-based method is hard to detect the TPCs which have limited syntactical features (e.g., without unique string literals), but they can be identified through CFG features. Second, their false positives have overlapping parts, which will not greatly increase the false positives of the union results. We further analyze the false positives and false negatives. First, the false positives are mainly caused by two reasons: 1) TPCs overlapping. Several TPCs reuse the code of other TPCs with minor changes. Under these circumstances, FirmSec will report all matched TPCs. 2) High similarity between different versions. Some versions of the same TPC have little difference or even no difference in their code. Second, the false negatives are due to two reasons: 1) Uncommon TPCs used in firmware. FirmSec cannot detect the TPCs that are not included in our database. 2) Insufficient features. Some firmware only use partially built TPCs which have limited features that can be used for identification.

Comparison Results. We compare FirmSec with three state of the arts: *Gemini* [54], *BAT* [37] and *OSSPolice* [30]. The original *Gemini* uses 8 block-level attributes to conduct code similarity comparison. We enhance it with our CFG weight-based method to detect the TPCs in firmware. *BAT* leverages string literals to identify the TPCs in binaries, but cannot detect the exact versions. *OSSPolice* is originally designed to detect the open-source software usage at version-level in Android apps. Given it supports finding TPCs in C/C++ native libraries, we successfully apply it on IoT firmware and compare it with FirmSec. Before we conduct comparison, we generate the corresponding TPC database for *BAT* and *OSSPolice* based on their instructions respectively. Table 3 presents the comparison results of FirmSec with *Gemini*, *BAT*, and *OSSPolice*. We do not report the results of *BAT* at version-level since it does not support version-level identification. For TPC-level identification, FirmSec is far better than other tools. FirmSec reports more TPCs at a higher precision and recall rate. For version-level identification, FirmSec outperforms *Gemini* and *OSSPolice* on both metrics. We further explore the reasons that why FirmSec is better than other tools. First, *Gemini* ignores the function-level attributes which offer much useful information on the structure of CFGs. Second, *BAT* mainly utilizes string literals to identify the TPCs. It uses the direct feature matching to compare the string literals extracted from TPCs and firmware, which causes a low precision and recall rate. Finally, *OSSPolice* utilizes a hierarchical matching strategy, which relies on the package structures of TPCs, to identify TPCs. Nevertheless, the package structures of the same TPC in different versions can be easily changed. Besides, its feature extraction tool does not perform well on IoT firmware since it is not optimized for firmware.

4 DATA CHARACTERIZATION

4.1 TPC Usage

In this subsection, we deploy FirmSec on the dataset to first identify the TPCs used in firmware. FirmSec successfully unpacks and

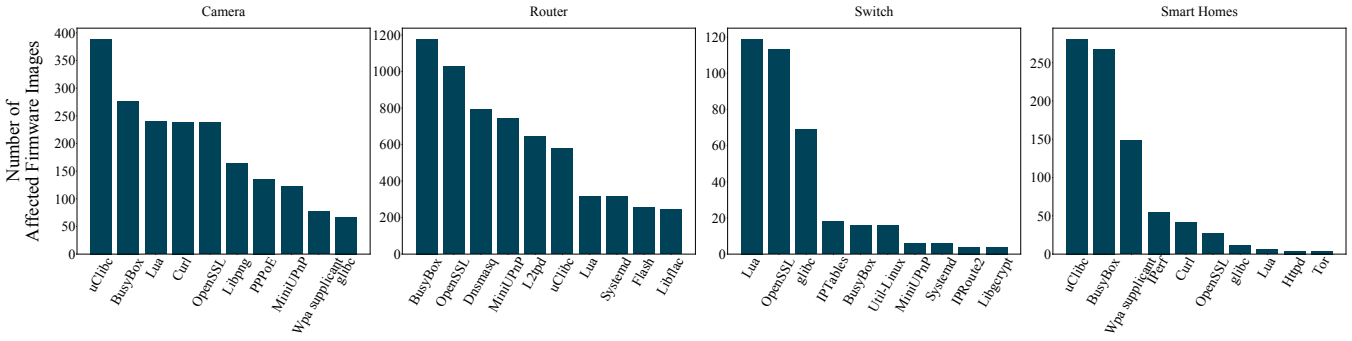


Figure 2: Top 10 TPCs Used in Each Kind of Firmware.

Table 4: Analysis Results of Our Dataset. # $\overline{TPC}$ and # $\overline{Vul.}$ represent the average number of TPCs and vulnerabilities of each firmware image respectively.

Vendor	Category	# Firmware	# TPC	# $\overline{TPC}$	# Vul.	# $\overline{Vul.}$
Xiongmai	Camera	520	232	0.45	313	0.60
Tomato-shibby	Router	230	2,088	9.08	11,948	51.95
Phicomm	Router	107	405	3.79	1,818	16.99
Fastcom	Router	149	274	1.83	1,849	12.41
	Unknown	10	0	0	0	0
Trendnet	Camera	477	136	0.28	1,395	4.32
	Router	336	1,762	5.24	7,903	23.52
	Switch	162	366	2.26	3,157	19.49
	Unknown	106	164	1.54	158	1.52
Xiaomi	Router	21	251	11.95	2,440	116.19
TP-Link	Camera	319	1,981	6.21	27,001	84.64
	Router	606	4,222	6.97	30,612	50.51
	Switch	484	77	0.16	795	1.64
	Unknown	48	67	1.40	639	13.31
D-Link	Camera	360	113	0.31	737	2.04
	Router	552	2,823	5.11	14,495	26.26
	Switch	545	80	0.15	1062	1.95
	Unknown	91	30	0.33	266	2.92
Hikvision	Camera	139	8	0.06	127	0.91
Foscam	Camera	113	0	0	0	0
Dahua	Camera	419	43	0.10	430	1.03
TSmart	Smart Homes	23,050	856	0.04	4,353	0.19
OpenWrt	Router	5,292	300,020	56.69	13,486	2.55

disassembles 96% firmware images in the whole dataset. We summarize the reasons why the remaining 4% firmware images cannot be analyzed: 1) 123 firmware images are encrypted; 2) 972 firmware images from TSmart have unknown processors; thus, we fail to implement plugins for *IDA* to process them; and 3) We do not support the unknown filesystems used in 269 Linux-based firmware images. We have excluded the remaining 4% firmware images in further analysis in Section 5.

As shown in Table 4, FirmSec identifies 584 different TPCs used in 34, 136 firmware images. Since there are lots of identified TPCs, we decide to present the results of top 10 TPCs used in each kind of firmware, as shown in Figure 2. Based on the results, we have the following findings. (1) Routers from OpenWrt contain the most TPCs, reaching an average of 56.69 per firmware image. It is reasonable that OpenWrt utilizes many TPCs since OpenWrt is an open-source project for embedded systems. (2) Smart homes from TSmart contain fewer TPCs which have 0.04 TPCs per firmware image. Most of the smart homes leverage the monolithic firmware.

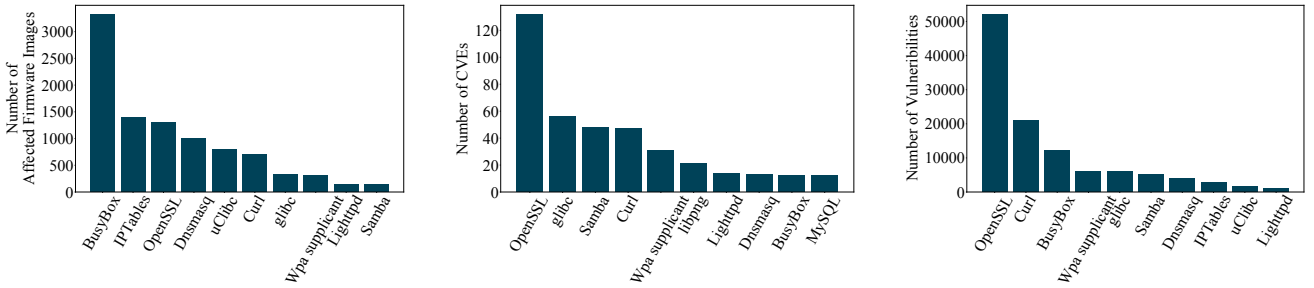
Table 5: Top 10 CWE Software Weaknesses.

	CWE ID	Weakness	# CVEs
1.	399	Resource Management Error	87
2.	119	Buffer Overflow	84
3.	310	Cryptographic Issues	47
4.	20	Improper Input Validation	39
5.	264	Access Control Error	36
6.	200	Information Disclosure	31
7.	189	Numeric Errors	20
8.	-	Insufficient Information	18
9.	94	Code Injection	8
10.	362	Race Condition	7

According to our further analysis, we find the monolithic firmware used in smart homes usually adopts their own implementations to replace the TPCs. Besides, some monolithic firmware is composed of a piece of code and data that achieves simple logistic functions. (3) 10 unknown firmware images from Fastcom and 113 cameras from Foscam are encrypted. For these encrypted firmware images, FirmSec cannot obtain useful information to identify the TPCs used in them. Currently, there is no effective method that can analyze the encrypted firmware automatically. (4) The same kind of firmware from different vendors adopts similar TPCs. For instance, all routers adopt *BusyBox*, *OpenSSL*, *Dnsmasq*, *MiniUPnP*, *uClibc*, *L2tpd* and *Util-linux* in their firmware. (5) Different kinds of firmware have commonalities in adopting TPCs. For instance, *BusyBox*, *Lua*, *OpenSSL* are all in the top 10 TPCs usage list of each kind of firmware. **In addition**, we have a counterintuitive finding. We find that only <1% firmware images are using *MbedTLS*, which is a popular lightweight TPC designed to replace *OpenSSL* in embedded systems. We propose two possible reasons why more firmware images are using *OpenSSL* rather than *MbedTLS*. (1) *OpenSSL* has more features compared to *MbedTLS*. When computing power is allowed, there are more reasons for developers to utilize *OpenSSL*. (2) Though *MbedTLS* is used in many popular IoT frameworks, e.g., FreeRTOS, many vendors have their own frameworks to develop the firmware which do not use *MbedTLS*.

4.2 Introduced Vulnerabilities Overview

After we obtain the TPCs used in firmware, we then search the corresponding vulnerabilities in our vulnerability database. As shown in Table 4, we detect a total of 128, 757 potential vulnerabilities, which involve 429 CVEs, in 34, 136 firmware images. In this paper,



(a) Number of Affected Firmware Images by Top 10 TPCs.

(b) Number of CVEs Caused by Top 10 TPCs.

(c) Number of Vulnerabilities Caused by Top 10 TPCs.

Figure 3: Number of Affected Firmware Images, CVEs and Vulnerabilities Caused by the Top 10 TPCs.

we count each CVE found in a firmware image as a separate vulnerability. Table 5 lists the top 10 CWE software weaknesses by the number of CVEs, accounting for 88% of all the CVEs we detected. The results indicate that the vulnerabilities in firmware involve a wide variety of security issues. Based on the results, we find IoT devices are suffering from substantial security risks since most of them contain a significant number of vulnerabilities. The routers from Xiaomi are in a very critical situation with 116.19 mean vulnerabilities, the most of all vendors. Moreover, routers produced by other vendors also have a great number of vulnerabilities. TP-Link, D-Link and Trendnet are well-known IoT vendors and all have multiple kinds of IoT devices. However, their products all have lots of vulnerabilities. The cameras from Xiongmai, Hikvision and Dahua all have nearly one vulnerability per firmware image. What's more, though OpenWrt contains the highest average number of TPCs per firmware image, we find comparatively few vulnerabilities in it, which has an average of 2.55 vulnerabilities for each firmware image. Next, FirmSec detects a comparative few vulnerabilities from TSmart's firmware in consideration of its largest scale. TSmart has the best performance among the involved vendors, which only contains 0.19 vulnerabilities per firmware image.

Besides, though we detect a large number of TPCs, most of the vulnerabilities are concentrated on a few TPCs. As shown in Figure 3, we list the top 10 TPCs from three aspects: the number of affected firmware images, the number of caused CVEs and the number of caused vulnerabilities. *OpenSSL* contains the most CVEs, totaling 132, that affects 1,304 firmware images and causes 52,135 vulnerabilities totally. *Busybox* is the most widely used TPC that has been identified in 3,326 firmware images. We detect 12 CVEs of *Busybox* that cause 12,072 vulnerabilities, which reaches an average of 1,006 vulnerabilities per CVE. Though *IPTables* has been identified in a great number of firmware images, it just causes a few vulnerabilities. Moreover, as shown in Figure 3(b), we have identified a total of 386 CVEs from these 10 TPCs, accounting for 90% of all the CVEs we detected.

5 ANALYSIS RESULTS

In this section, we aim to answer the following research questions.

- **RQ1:** How vulnerable are firmware images of different kinds and from different vendors?
- **RQ2:** What is the geographical distribution of the devices using vulnerable firmware?
- **RQ3:** Does the firmware adopt the latest TPCs at the time when it was released?
- **RQ4:** Are there any TPC license violations?

Table 6: Vulnerability of Different Kinds of Firmware. $\overline{Critical}$, $\overline{High}$, $\overline{Medium}$ and $\overline{Low}$ represent the average number of critical, high, medium and low vulnerabilities.

Category	$\overline{Vul.}$	$\overline{Critical}$	$\overline{High}$	$\overline{Medium}$	$\overline{Low}$
Router	22.92	1.48	2.73	17.59	1.12
Camera	9.81	0.32	1.92	7.20	0.37
Switch	5.29	0.22	0.62	3.98	0.47
Smart Homes	0.19	0.01	0.05	0.11	0.02

Table 7: Vulnerability of Firmware From Different Vendors.

Vendor	$\overline{Vul.}$	$\overline{Critical}$	$\overline{High}$	$\overline{Medium}$	$\overline{Low}$
Xiaomi	116.19	2.86	18.52	78.43	10.67
Tomato-shibby	51.95	2.77	8.46	35.49	1.84
TP-Link	39.20	1.37	6.97	28.59	2.26
Phicomm	16.99	0.41	3.88	11.28	0.54
D-link	11.87	0.55	1.95	8.03	1.34
Trendnet	11.02	0.29	1.85	7.98	0.90
Fastcom	9.13	0.44	1.35	6.81	0.53
OpenWrt	2.55	0.00	0.46	1.58	0.00
Dahua	1.03	0.03	0.14	0.66	0.16
Hikvision	0.91	0.05	0.17	0.60	0.03
Xiongmai	0.60	0.00	0.21	0.32	0.07
TSmart	0.19	0.01	0.05	0.11	0.02

Results look skewed better toward the non-linux firmwares, which is the major improvement and questions their results if their improvement actually worked?

5.1 Firmware Vulnerability

This subsection answers **RQ1**. Though we have identified many vulnerabilities in firmware, we still lack an understanding of how vulnerable are firmware images of different kinds and from different vendors. To answer this question, we evaluate the security of firmware based on the average number of vulnerabilities of different severity in firmware. We also discover some critical vulnerabilities still threaten the security of firmware.

First, we explore the vulnerability of different kinds of firmware. As shown in Table 6, we list the average number of different severity vulnerabilities in each kind of firmware. Since the number of routers

Table 8: The Firmware Affected by Two Vulnerabilities.

Vendors	Heartbleed (CVE-2014-0160)	GHOST (CVE-2015-0235)
Fastcom	2	1
Trendnet	36	87
Tomato-shibby	24	-
TP-Link	301	91
D-Link	3	45
Hikvision	1	-
Dahua	5	-
TSmart	8	-

from OpenWrt is much higher than other vendors, which may bring bias to our results, we finally randomly select 500 OpenWrt routers for analysis. We notice that the router is more vulnerable to attacks than the other three kinds of firmware since it has the most vulnerabilities of different severity. The router is also the only category that has more than one critical vulnerability per firmware image on average. The camera and switch have a similar number of critical vulnerabilities and low vulnerabilities per firmware image, while the camera has more high vulnerabilities and medium vulnerabilities than the switch. Smart home is the least vulnerable category since it has few vulnerabilities of different severity. **Second**, we study the vulnerability of firmware from different vendors. As shown in Table 7, Xiaomi has the most vulnerabilities of different severity, which causes Xiaomi more vulnerable to attacks. Both Xiaomi and Tomato-shibby have nearly three critical vulnerabilities per firmware image on average, which is significantly higher than the other vendors. The vulnerabilities detected in TP-Link, Phicomm, D-Link, Trendnet, and Fastcom are mainly at the medium severity level. OpenWrt, Dahua, Hikvision, Xiongmai, and TSmart have very few vulnerabilities per firmware image. Besides, both OpenWrt and Xiongmai have no critical vulnerability in each firmware image.

During the above analysis, we surprisingly find that some critical vulnerabilities are still having severe impact on the firmware in our dataset. We take two representative critical vulnerabilities: *OpenSSL* Heartbleed and *glibc* GHOST for instance. We find these two vulnerabilities have affected 604 firmware images which account for 1.8% of the dataset, as shown in Table 8. More specifically, 380 firmware images from 8 vendors are vulnerable to the Heartbleed, and 224 firmware images from 4 vendors contain the GHOST.

5.2 Geographical Distribution

This subsection answers **RQ2**. We regard that geographical distribution of vulnerable devices can reflect the potential imbalance between regions in terms of IoT devices' threat level. To answer this question, **first**, we map firmware to its exact device model, which is necessary for searching IoT devices in the IoT search engine. We successfully find the corresponding device models with exact firmware versions for 1,247 firmware images. For instance, we confirm the firmware DIR827A1_FW103.bin is actually used in D-Link Amplifi HD Media Router 2000 (DIR-827), whose firmware version is 1.0.3. **Second**, we utilize *Shodan* [15] to find the distribution of these possible vulnerable IoT devices, and choose the

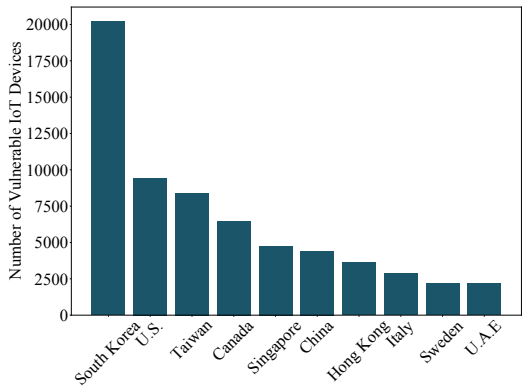


Figure 4: Top 10 Regions with the Most Vulnerable Devices.

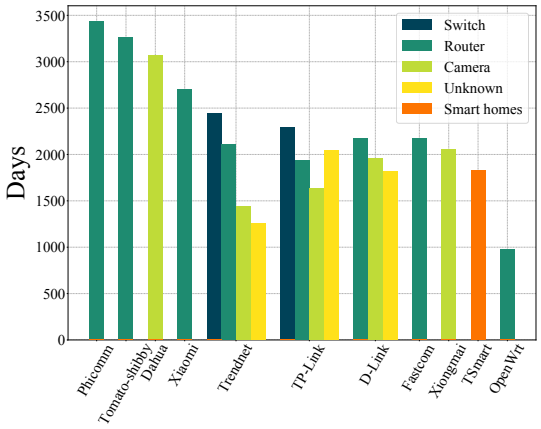


Figure 5: Delay Time of TPCs.

devices whose version is the same as the vulnerable firmware. For example, we use **DIR-827** as the keyword to search the DIR-827 routers. Then, we select the routers whose version is 1.0.3 that is shown in their banner. Based on the search results, we list the top 10 regions with the most deployed vulnerable IoT devices, as shown in Figure 4. There are six regions: South Korea, Taiwan, Singapore, China, Hongkong, and the United Arab Emirates, located in Asia. South Korea contains the most vulnerable IoT devices all over the world, which reaches 20,191. The two countries in North America - the United States and Canada both have a large number of vulnerable IoT devices. Europe contains relatively few vulnerable IoT devices.

We propose three possible reasons for the difference in the distribution of vulnerable devices. (1) The number of devices sold by vendors varies from region to region. (2) The security of the devices on factory mode is different in different regions. The same devices sold in some regions may have already been equipped with the updated firmware on factory mode. (3) Vendors have different firmware update mechanisms in different regions. Some regions may get firmware updates preferentially.

5.3 Delay Time of TPCs

This subsection answers question **RQ3**. Most of the firmware vulnerabilities caused by TPCs are due to that firmware is still utilizing outdated TPCs. However, the latest TPCs may not be suitable for

firmware, since they may lower the performance and influence the stability. Vendors may ignore the vulnerabilities and still use the old version of TPCs. Considering this situation, we want to explore the delay time of TPCs used in firmware. The delay time represents the days from the release date of the TPC that the firmware was using to the latest version of the TPC when the firmware was released.

We first obtain the release dates of TPCs based on their versions. Then, we obtain the release dates of the corresponding firmware. Based on this, we infer the latest versions of the TPCs used in firmware at the time when the firmware was released. Finally, we calculate the delay time by the difference between these two release dates. Figure 5 shows the average delay time of the used TPCs for each vendor. The TPCs used in Phicomm have the longest delay time, which reaches 3457.2 days. In other words, Phicomm is still using the TPCs that were released ten years ago. OpenWrt has the shortest delay time, which is less than two years. It could be a reason that OpenWrt has few vulnerabilities, as shown in Table 4. Xiaomi also has a long delay time, nearly seven years. What's more, we also notice that Xiaomi contains the most number of vulnerabilities per firmware image, which also confirms the relationship between the delay time and the number of vulnerabilities. D-Link, TP-Link, and Trendnet all have a couple of different kinds of IoT devices. These vendors have a common phenomenon that the router has a longer delay time than the camera. The average delay time of TPCs for all involved firmware images is 1948.2 days, which shows that the TPCs they used have fallen behind by five years. Our results reveal the widespread usage of outdated TPCs in IoT firmware.

5.4 License Violations

This subsection answers question **RQ4**. The use of TPCs in firmware can lead to complex license compliance issues. For instance, Cisco has involved in a lawsuit since it did not adhere to the license requirements [45]. We mainly study the license violations caused by two highly restrictive licenses: General Public License (GPL) and Affero General Public License (AGPL), since they are widely used licensing terms and have a basic requirement that developers should provide the source code if they distribute the programs that use the TPCs licensed under GPL/AGPL. According to our in-depth study, we first discover 2,478 commercial firmware images that have potentially violated GPL/AGPL licensing terms, as shown in Table 9. We then study the open-source policy of the involved vendors. We notice that four vendors (TP-Link, D-Link, Trendnet, and Hikvision) have provided distribution sites for downloading the source code of firmware. Nevertheless, we find the source code of some firmware, which utilizes the GPL/AGPL-licensed TPCs, cannot be found on their sites. We further contact the involved vendors, except for the TSmart, to request the source code for some firmware but do not get any response yet.

We summarize three possible reasons why vendors do not open-source the firmware according to license requirements. **First**, vendors disregard the restrictions of licenses. Currently, there are no strong measures to enforce the GPL/AGPL compliance since the lawsuit is complicated and may not apply to some countries. **Second**, open-sourcing the firmware may lead to new attacks. Attackers can find vulnerabilities through auditing the source code, which may affect more firmware images if vendors reuse the vulnerable code in other firmware. **Third**, the firmware has license conflicts. Vendors may have some commercial licenses that applied to the firmware simultaneously, which conflicts with the open-source licenses.

Table 9: Potential License Violations.

Vendors	# Firmware	Source Code Available
Xiongmai	195	✗
Phicomm	96	✗
Fastcom	17	✗
Trendnet	433	✓
Xiaomi	20	✗
TP-Link	847	✓
D-Link	487	✓
Hikvision	2	✓
Dahua	11	✗
TSmart	370	✗

6 DISCUSSION

Ethics. Our large-scale vulnerability analysis of IoT firmware may raise serious ethical concerns. To avoid these potential hazards, we pay special attention to legal and ethical issues. First, all firmware images are collected and treated legally. For the publicly accessible firmware, we collect it from legal sources, e.g., official websites, and adhere to the Robots Exclusion Protocol (REP) [39]. For the private firmware, we only use it for research purposes. Besides, responsible disclosure is also a basic requirement for us. We have actively contacted the related vendors and reported our results to them as detailed as possible. Finally, we have a legal and ethical issues-free plan to open-source our dataset. For the publicly accessible firmware images, we will provide their official download links. For the private firmware images, we have full authorization from TSmart to open-source them after desensitizing.

Limitations and Future Work. In the future, we plan to improve our work in three aspects. First, we will continue to collect more firmware images to extend our dataset. Though we have collected 34,136 firmware images, we still lack the firmware of some current popular devices, e.g., smart assistant devices. Second, we will continually enrich our TPC database. Currently, FirmSec can only detect the TPCs included in the database. During our analysis, FirmSec has some false positives at TPC-level identification since our database does not record some uncommon TPCs used in IoT firmware. Finally, we will adopt new techniques to conduct a more in-depth analysis of IoT devices. FirmSec has an outstanding performance in finding N-days vulnerabilities caused by TPCs. However, it is hard to detect unknown vulnerabilities. We plan to combine fuzzing techniques [41, 42] with FirmSec to find new vulnerabilities.

7 RELATED WORK

TPC Detection. Several works have been proposed to discover the TPCs used in Android apps [18, 44, 56, 57]. However, these works more or less require Android features support. Therefore, it is difficult to apply them to IoT firmware analysis. Duan et al. [30] proposed *OSSPolice* to detect the open-source software license violations and identify the open-source software at version-level in Android apps. Though *OSSPolice* supports C/C++ native binaries,

its feature extraction tool does not perform well on IoT firmware. Besides, its hierarchical matching strategy heavily relies on the correctness of the package structures of TPCs. Hemel et al. [37] proposed *BAT* to detect the usage of TPCs in binaries based on string literals. Nevertheless, the direct feature matching strategy adopted by *BAT* will cause a low precision and recall rate.

Static Analysis. Costin et al. [23] conducted the first large-scale analysis of firmware. However, they did not delve into the vulnerabilities introduced by TPCs in firmware. Other works utilized more robust features from I/O behavior [46] and control flow graphs [32, 34] of an image to discover vulnerabilities. Nevertheless, these methods cannot be applied for large-scale firmware vulnerability search since they only support firmware using a few kinds of architectures and require a lot of manual work. Several works focus on using similar code detection to find vulnerabilities in firmware. Xu et al. [54] proposed a neural network-based approach to compare the similarity of binary codes. David et al. [27] proposed *FirmUp* to conduct a precise static detection of common vulnerabilities in firmware via matching similar procedures in the context of executables. Ding et al. [29] proposed *Asm2Vec* for assembly clone detection based on learned representation. However, these works are designed for comparing the similarity between the individual functions rather than the similarity between the entire TPCs with firmware.

Dynamic Analysis. Zaddach et al. [55] designed *Avatar*, a dynamic analysis framework for firmware security analysis by forwarding I/O access between the emulator and real device. Further, Muench et al. [43] described how to orchestrate the execution among different testing environments. Chen et. al [21] presented *FIRMADYNE* to automatically analyze Linux-based firmware. Costin et al. [24] performed the security analysis of web interfaces within embedded devices leveraging several off-the-shell analysis tools. Feng et al. [33] proposed P^2IM to perform fuzz-testing on firmware from MCU devices in a fully emulated fashion. Nevertheless, it is hard to apply them tn a large-scale analysis since they require real devices or a lot of manual work to configure for each firmware image.

8 CONCLUSION

In this paper, we conduct the largest-scale analysis of the TPC issues in IoT firmware at present. We propose FirmSec which dedicates to finding the vulnerabilities in firmware caused by TPCs. Based on FirmSec, we identify 584 TPCs and detect a total of 128,757 security vulnerabilities caused by 429 CVEs in 34,136 firmware images. Our analysis reveals the widespread usage of vulnerable and outdated TPCs in IoT firmware. Moreover, we present a global view of the geographical difference in the security of IoT devices. Further analysis discloses the GPL/AGPL license violations widely exist in firmware. We believe our work will shed light on the further study of the security of IoT devices.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable comments to improve our paper. This work was partly supported by NSFC under No. U1936215, the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under No. LR19F020003, the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCHA202001, the Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform), Google Research Scholar Award, Facebook Faculty Award, NSFC under No. 62102363, and the Zhejiang Provincial Natural Science Foundation under No. LQ21F020010.

REFERENCES

[1] 2015. Sasquatch. https://github.com/devttys0/sasquatch/
[2] 2016. yaffshiv. https://github.com/devttys0/yaffshiv.
[3] 2022. Binary Analysis Next Generation (BANG). https://github.com/armijnhemel/binaryanalysis-ng
[4] 2022. BusyBox. https://busybox.net/.
[5] 2022. Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/.
[6] 2022. CVE Details. https://www.cvedetails.com/.
[7] 2022. CVE-search. https://github.com/cve-search/cve-search.
[8] 2022. CVSS: Common Vulnerability Scoring System SIG. https://www.first.org/cvss/.
[9] 2022. Cyclomatic Complexity. https://en.wikipedia.org/wiki/Cyclomatic_complexity.
[10] 2022. Edit Distance. https://en.wikipedia.org/wiki/Edit_distance.
[11] 2022. JFFS2 filesystem extraction tool. https://github.com/sviehb/jefferson.
[12] 2022. Maven Repository. https://mvnrepository.com/.
[13] 2022. NATIONAL VULNERABILITY DATABASE. https://nvd.nist.gov/.
[14] 2022. OpenSSL. https://www.openssl.org/.
[15] 2022. Shodan. https://www.shodan.io/.
[16] National Security Agency. 2019. GHIDRA. https://github.com/NationalSecurityAgency/ghidra.
[17] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis
[18] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable Third-Party Library Detection in Android and its Security Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 356–367. https://doi.org/10.1145/2976749.2978333
[19] Ulrich Bayer, Imam Habibi, Davide Balzarotti, and Engin Kirda. 2009. A View on Current Malware Behaviors. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '09, Boston, MA, USA, April 21, 2009*, Wenke Lee (Ed.). USENIX Association. https://www.usenix.org/conference/leet-09/view-current-malware-behaviors
[20] Thomas Bittman, Bob Gill, Tim Zimmerman, Ted Friedman, Neil MacDonald, and Karen Brown. 2021. Predicts 2022: The Distributed Enterprise Drives Computing to the Edge. https://www.gartner.com/en/documents/4007176.
[21] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/towards-automated-dynamic-analysis-linux-based-embedded-firmware.pdf
[22] Nassim Corteggiani, Giovanni Camurati, and Aurélien Francillon. 2018. Inception: System-Wide Security Testing of Real-World Embedded Systems Software. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 309–326. https://www.usenix.org/conference/usenixsecurity18/presentation/corteggiani
[23] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, 95–110. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin
[24] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. 2016. Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang (Eds.). ACM, 437–448. https://doi.org/10.1145/2897845.2897900
[25] Ang Cui. 2018. The Overlooked Problem of 'N-Day' Vulnerabilities. https://www.darkreading.com/vulnerabilities---threats/the-overlooked-problem-of-n-day-vulnerabilities/a/d-id/1331348

[26] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 2702–2711. http://proceedings.mlr.press/v48/daib16.html

[27] Yaniv David, Nimrod Partush, and Eran Yahav. 2018. 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, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 392–404. https://doi.org/10.1145/3173162.3177157

[28] Drew Davidson, Benjamin Moench, Thomas Ristenpart, and Somesh Jha. 2013. FIE on Firmware: Finding Vulnerabilities in Embedded Systems Using Symbolic Execution. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Washington, D.C., 463–478. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/davidson

[29] Steven H. H. Ding, Benjamin C. M. Fung, and Philippe Charland. 2019. Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 472–489. https://doi.org/10.1109/SP.2019.00003

[30] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. 2017. Identifying Open-Source License Violation and 1-day Security Risk at Large Scale. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 2169–2185. https://doi.org/10.1145/3133956.3134048

[31] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, Carey Williamson, Aditya Akella, and Nina Taft (Eds.). ACM, 475–488. https://doi.org/10.1145/2663716.2663755

[32] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. 2016. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/discovre-efficient-cross-architecture-identification-bugs-binary-code.pdf

[33] Bo Feng, Alejandro Mera, and Long Lu. 2020. P2IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1237–1254. https://www.usenix.org/conference/usenixsecurity20/presentation/feng

[34] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. 2016. Scalable Graph-based Bug Search for Firmware Images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 480–491. https://doi.org/10.1145/2976749.2978370

[35] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* 29, 7, 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[36] Craig Heffner. 2022. Binwalk. https://github.com/ReFirmLabs/binwalk.

[37] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. 2011. Finding software license violations through binary code clone detection. In *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings*, Arie van Deursen, Tao Xie, and Thomas Zimmermann (Eds.). ACM, 63–72. https://doi.org/10.1145/1985441.1985453

[38] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey M. Voas. 2017. DDoS in the IoT: Mirai and Other Botnets. *Computer* 50, 7, 80–84. https://doi.org/10.1109/MC.2017.201

[39] Martijn Koster, Gary Illyes, Henner Zeller, and Lizzi Harvey. 2022. Robots Exclusion Protocol. https://datatracker.ietf.org/doc/html/draft-koster-rep.

[40] Shancang Li, Li Da Xu, and Shanshan Zhao. 2015. The internet of things: a survey. *Inf. Syst. Frontiers* 17, 2, 243–259. https://doi.org/10.1007/s10796-014-9492-7

[41] Peiyu Liu, Shouling Ji, Xuhong Zhang, Qinming Dai, Kangjie Lu, Lirong Fu, Wenzhi Chen, Peng Cheng, Wenhai Wang, and Raheem Beyah. 2021. IFIZZ: Deep-State and Efficient Fault-Scenario Generation to Test IoT Firmware. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 805–816. https://doi.org/10.1109/ASE51524.2021.9678785

[42] Chenyang Lyu, Shouling Ji, Xuhong Zhang, Hong Liang, Binbin Zhao, Kangjie Lu, and Raheem Beyah. 2022. EMS: History-Driven Mutation for Coverage-based Fuzzing. In *29rd Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society. https://www.ndss-symposium.org/wp-content/uploads/2022-162-paper.pdf

[43] Marius Muench, Dario Nisi, Aurelien Francillon, and Davide Balzarotti. 2018. Avatar^2: A Multi-target Orchestration Platform. In *Workshop on Binary Analysis Research (BAR)*. http://s3.eurecom.fr/docs/bar18_muench.pdf

[44] Duc Cuong Nguyen, Erik Derr, Michael Backes, and Sven Bugiel. 2020. Up2Dep: Android Tool Support to Fix Insecure Code Dependencies. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*. ACM, 263–276. https://doi.org/10.1145/3427228.3427658

[45] Ryan Paul. 2009. Cisco settles FSF GPL lawsuit, appoints compliance officer. https://arstechnica.com/information-technology/2009/05/cisco-settles-fsf-gpl-lawsuit-appoints-compliance-officer/.

[46] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz. 2015. Cross-Architecture Bug Search in Binary Executables. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 709–724. https://doi.org/10.1109/SP.2015.49

[47] Marco Schwartz. 2016. *Internet of Things with ESP8266*. Packt Publishing Ltd.

[48] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/firmalice-automatic-detection-authentication-bypass-vulnerabilities-binary-firmware

[49] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard A. Kemmerer, Christopher Kruegel, and Giovanni Vigna. 2009. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis (Eds.). ACM, 635–647. https://doi.org/10.1145/1653662.1653738

[50] Hui Suo, Jiafu Wan, Caifeng Zou, and Jianqi Liu. 2012. Security in the Internet of Things: A Review. In *2012 International Conference on Computer Science and Electronics Engineering*, Vol. 3. 648–651. https://doi.org/10.1109/ICCSEE.2012.373

[51] HongRu Wang, ChunFang Li, LingFei Zhang, and MinYong Shi. 2018. Anti-Crawler strategy and distributed crawler based on Hadoop. In *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*. 227–231. https://doi.org/10.1109/ICBDA.2018.8367682

[52] Qinying Wang, Shouling Ji, Yuan Tian, Xuhong Zhang, Binbin Zhao, Yuhong Kan, Zhaowei Lin, Changting Lin, Shuiguang Deng, Alex X. Liu, and Raheem Beyah. 2021. MPInspector: A Systematic and Automatic Approach for Evaluating the Security of IoT Messaging Protocols. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 4205–4222. https://www.usenix.org/conference/usenixsecurity21/presentation/wang-qinying

[53] Qinge Xie, Shujun Tang, Xiaofeng Zheng, Qinran Lin, Baojun Liu, Haixin Duan, and Frank Li. 2022. Building an Open, Robust, and Stable Voting-Based Domain Top List. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity22/presentation/xie

[54] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 363–376. https://doi.org/10.1145/3133956.3134018

[55] Jonas Zaddach, Luca Bruno, Aurélien Francillon, and Davide Balzarotti. 2014. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. https://www.ndss-symposium.org/ndss2014/avatar-framework-support-dynamic-security-analysis-embedded-systems-firmwares

[56] Xian Zhan, Lingling Fan, Sen Chen, Feng Wu, Tianming Liu, Xiapu Luo, and Yang Liu. 2021. ATVHUNTER: Reliable Version Detection of Third-Party Libraries for Vulnerability Identification in Android Applications. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1695–1707. https://doi.org/10.1109/ICSE43902.2021.00150

[57] Jiexin Zhang, Alastair R. Beresford, and Stephan A. Kollmann. 2019. LibID: reliable identification of obfuscated third-party Android libraries. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, Dongmei Zhang and Anders Møller (Eds.). ACM, 55–65. https://doi.org/10.1145/3293882.3330563

[58] Binbin Zhao, Shouling Ji, Wei-Han Lee, Changting Lin, Haiqin Weng, Jingzheng Wu, Pan Zhou, Liming Fang, and Raheem Beyah. 2022. A Large-Scale Empirical Study on the Vulnerability of Deployed IoT Devices. *IEEE Trans. Dependable Secur. Comput.* 19, 3, 1826–1840. https://doi.org/10.1109/TDSC.2020.3037908