Game of Hide-and-Seek: Exposing Hidden Interfaces in Embedded Web Applications of IoT Devices

Wei Xie, Jiongyi Chen*, Zhenhua Wang, Chao Feng,
Enze Wang, Yifei Gao, Baosheng Wang, Kai Lu

{xiewei,chenjiongyi,wzh15,chaofeng,wangenze18,gaoyf,bswang,kailu}@nudt.edu.cn
National University of Defense Technology

ABSTRACT

Recent years have seen increased attacks targeting embedded web applications of IoT devices. An important target of such attacks is the hidden interface of embedded web applications, which employs no protection but exposes security-critical actions and sensitive information to illegitimate users. With the severity and the pervasiveness of this issue, it is crucial to identify the vulnerable hidden interfaces, shed light on best practices and raise public awareness.

In this paper, we present IoTScope, a new approach that automatically exposes hidden web interfaces of IoT devices. Specifically, IoTScope constructs probing requests through firmware analysis to test physical devices, and narrows down the scope of identification by filtering out irrelevant requests and interfaces through differential analysis. It pinpoints hidden interfaces by attaching various device-setting parameters in the probing requests and matching keywords of sensitive information. Evaluated on 17 IoT devices, IoTScope successfully identified 44 vulnerabilities, including 43 previously unknown ones. IoTScope also demonstrates surprising efficiency: on average, it delivered 151438 probing requests, taking only 47 minutes on each target device.

CCS CONCEPTS

• **Security and privacy → Web application security**.

KEYWORDS

Vulnerability detection; authentication; web security; Internet of Things

ACM Reference Format:
Wei Xie, Jiongyi Chen, Zhenhua Wang, Chao Feng, Enze Wang, Yifei Gao, Baosheng Wang, Kai Lu. 2022. Game of Hide-and-Seek: Exposing Hidden Interfaces in Embedded Web Applications of IoT Devices. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3512213

* Corresponding author

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512213

1 INTRODUCTION

With the rapid evolution of Internet-of-Things technologies, recent years have seen broad adoption of IoT devices and applications[22]. Embedded web applications (EWAs for short) play an essential role in managing and configuring vast amounts of devices. They provide a universal and convenient way to interact with end-users. Nevertheless, although the EWAs are increasingly deployed, the protection of EWAs is lagged[1, 3, 26]. Embedded developers do not raise enough awareness about security and do not also follow best practices during development. Consequently, many EWAs are made available without protection and can be easily exploited by unauthenticated attackers.

Hidden interface of IoT device. The most feasible way to attack EWAs is probably through hidden interfaces. A hidden interface allows unauthenticated users to remotely access it without any permission. In reality, developers intentionally or accidentally leave hidden interfaces, which often expose sensitive information like revealing admin passwords or security-critical operations like changing network settings. As an example, CVE-2019-14984 [12] reports that some smart home control devices allow unauthenticated attackers to run system commands by accessing an undocumented web interface "`exec.cgi`". Furthermore, a hidden interface itself is not only vulnerable but also acts as a front door to the exploitation of other vulnerabilities. For instance, in the case of CVE-2018-11510 [11], there exists a command injection in the web interface "`/portal/apis/aggrecate_js.cgi`" of an IoT device. Exploiting this vulnerability requires the presence of a hidden interface to bypass authentication. Those cases are only the tip of the iceberg. According to the statistics from OWASP [25], broken access control has moved up from the fifth position to the category with the most serious web application security risk.

Previous studies of identifying vulnerabilities in IoT devices have been traditionally focused on memory corruptions [6, 14, 40], taint-style vulnerabilities [8, 39], and domain-specific vulnerability types[18, 31]. On the other hand, automatic detection techniques of broken access control focus on particular targets such as the cloud backends of mobile services [2, 43–45] and visible interfaces of general web applications [15, 27, 33]. Given that there lacks a tool applicable to exposing hidden interfaces in IoT devices, the worrisome situation of broken access control is apparently underestimated in the IoT domain.

Our approach. This paper introduces IoTScope, a new approach that automatically exposes hidden interfaces in IoT devices. In particular, we extract filenames and pathnames through static firmware analysis to construct probing requests. Each probing request is sent

out twice (with a minor difference), and the similarity of responses is compared to judge whether they correspond to an invalid request. Based on that, we filter out invalid requests and narrow down the scope of identification with the aim of keeping unprotected interfaces. Finally, IoTScope pinpoints two types of hidden interfaces: (1) for the hidden interfaces that allow manipulation of device settings, we extract various device-setting parameters from the frontend of embedded web services; (2) for the hidden interfaces that expose sensitive information, we match the content on the interfaces with a dictionary of keywords extracted from NVRAM parameters and configuration files.

We evaluated IoTScope by testing 17 devices from 11 vendors. The devices belong to 8 different device types. To our surprise, IoTScope identified 44 vulnerabilities, including 43 of them that were previously unknown. Those identified vulnerabilities could lead to severe consequences like remote code execution, exposure of security-critical actions, sensitive information disclosure, etc. We reported all the vulnerabilities to the corresponding vendors. They confirmed the vulnerabilities and assigned 8 CVE IDs. In the experiments, IoTScope demonstrated impressive performance: the testing time spent on each device is from 101 seconds to 3.59 hours, with an average of 47 minutes per device. With a full implementation and a comprehensive evaluation, IoTScope makes the first step towards automated and quantitative measurement for the identification of hidden interfaces in IoT devices.

The main contributions of this paper are summarized as follows:

- **New approach.** We design and implement IoTScope[1], a new tool that can automatically expose hidden interfaces in embedded web applications of IoT devices. With static firmware analysis to enumerate all possible interfaces and narrow down the scope of identification step-by-step, IoTScope can pinpoint two common types of vulnerable hidden interfaces, namely hidden device-setting interfaces and hidden information-disclosure interfaces.
- **Real-world impact.** We evaluated IoTScope with 17 real-world IoT devices. To our surprise, it successfully identified 44 vulnerabilities, including 43 of them that were previously unknown. We responsibly reported all the identified vulnerabilities to "`cve.mitre.org`" and the corresponding vendors, and received 8 CVE IDs.

We organize the rest of the paper as follows: Section 2 provides the background about embedded web applications and hidden interfaces, as well as the technical challenges in developing the tool. Section 3 details the design of IoTScope. Section 4 gives the evaluation of IoTScope. Section 5 reviews related works, and Section 6 gives the conclusion.

2 BACKGROUND AND MOTIVATION

2.1 Embedded Web Applications

Embedded web applications are widely adopted in IoT devices. They typically serve as the administration panel for the easy configuration of embedded devices. EWAs are different from traditional web applications. Traditional web servers like Apache, IIS, and Nginx are relatively heavy and tailored for high performance

[1]IoTScope would be open-sourced via Github after the conference.

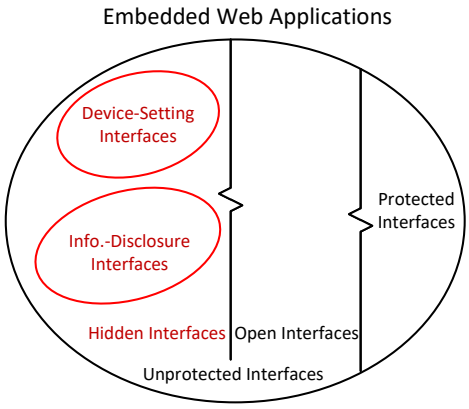

Figure 1: Relationship of Interfaces in EWA

and quick response. On the contrary, embedded web servers like mini_httpd [23], boa [4] and lighttpd [21] that host EWAs, are lightweight and open-sourced for easy development and quick customization. As a result, unlike traditional web applications hosting web files like `.PHP`, `.ASP` and `.JSP`, many EWAs host binary-based `CGI` files. Without semantics at the source code level, it is more challenging to analyze `CGI` files than those scripts of traditional web applications. Even worse, cross-architectural analysis [17, 28, 36] is often involved, as a large number of IoT devices are based on Reduced Instruction Set Computing (RISC) architectures like ARM and MIPS.

2.2 Hidden Interfaces of Embedded Web Applications

EWAs are protected by authentication and authorization. Unfortunately, lacking security awareness, embedded developers do not always follow best practices and intentionally or accidentally leave hidden interfaces. Some EWA interfaces do not require login credentials, allowing unauthenticated users to access security-critical actions or sensitive information. We call them *hidden interfaces*, as they are different from: (1) *open interfaces* such as login/welcome pages and resource files that are directly exposed to unauthenticated users; (2) *protected interfaces* that can only be accessed after authentication or authorization. The relationship of those interfaces is shown in Figure 1. The hidden interfaces contain vulnerabilities inadvertently caused by developers or backdoors left on purpose. Given that an EWA interface is mainly designed either for configuring or displaying settings of a device, there are mainly two types of vulnerabilities:

- **Manipulation of device setting:** If a hidden interface allows configuration of device settings, it leads to unauthorized manipulation, e.g., manipulating the IP address of a DNS server by unauthenticated attackers.
- **Information disclosure:** If a hidden interface is designed to display device settings to users, an information disclosure vulnerability could be caused, for example, leaking a user's login password to unauthenticated attackers.

In practice, we believe hidden interfaces would be the headmost attack surface of IoT devices due to three reasons. First, accessing the hidden interfaces or triggering vulnerabilities on them does

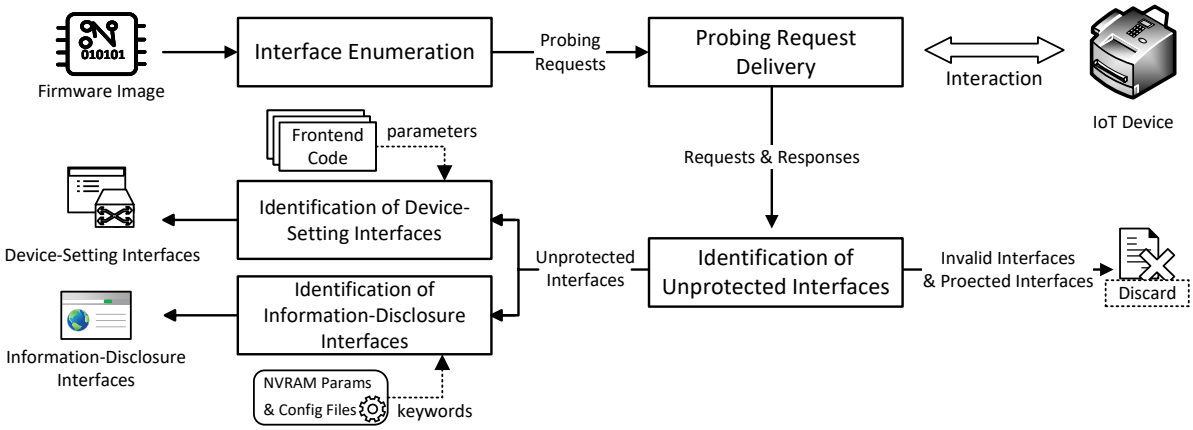

Figure 2: Workflow of IoTScope

not require any authentication and authorization. Second, the gain is high for attackers. Once abused, the hidden interfaces expose sensitive information or allow security-critical actions. Third, given that web servers often listen on HTTP/HTTPS ports, accessing hidden web interfaces would not be prevented by network layer firewalls [35].

2.3 Challenges in Exposing Hidden Interfaces

Given the stealthiness and severe consequences of hidden interfaces of IoT, it is imperative to discover them automatically and assess their impact in the real world. Unfortunately, we found that there is no existing solution that serves this purpose. To this aim, our approach first enumerates possible interfaces, narrows down the scope step by step and eventually identify hidden interfaces, which brings three specific challenges:

- **Challenge 1: enumerating possible interfaces.** Web interfaces are presented in different forms and reside in various locations, making it difficult to enumerate all possible interfaces. On the one hand, they have different extensions of filename like `.PHP`, `.ASP`, `.JSP`, and `.CGI`. Some of the interfaces are scripts, while the others are binaries. On the other hand, they could be either independent files or functions of servers. For instance, `FIRMADYNE` [5] failed to identify the vulnerability CVE-2017-5521 [10], as the tool only enumerates independent files in the "`www`" directory. But the vulnerable interface is presented as a function in the binary executable "`httpd`".
- **Challenge 2: identifying unprotected interfaces.** Once the probing requests are sent out, we collect a bunch of unconfirmed request-response pairs. The next step is to first identify valid interfaces and then identify unprotected interfaces among them. Typically, a web server should reply with a 404 status code to the client when the HTTP request specifies a nonexistent URL. Unfortunately, embedded developers do not always follow the standards of the HTTP protocol. As a result, EWAs would answer the client with responses having various HTTP status codes, if the requests access valid interfaces and unprotected interfaces. For example, some EWAs reply with the `400` status code to indicate a general bad request; Some EWAs answer any request with the status code `200`, then details the error in the response body. Thus, we need to handle the mess of informal programming conventions and automatically identify unprotected interfaces without leveraging the semantics of responses.
- **Challenge 3: identifying hidden interfaces.** Unlike the triggering of memory corruptions that often lead to program crashes, there is no indicator to confirm whether a valid interface is a hidden interface. For information disclosure, some hidden interfaces display security questions related to user privacy or leak login passwords to unauthenticated users, resulting in authentication bypass attacks. Regarding the vulnerability of manipulation of device settings, hidden interfaces could allow unauthenticated attackers to change the DNS settings of the device, or allow the manipulation of router's Wi-Fi settings, causing denial-of-service attacks to wireless users. The technical challenge lies in how to systematically identify the hidden interfaces given the diverse forms and behaviors.

3 DESIGN

The architecture of IoTScope is shown in Figure 2. The input is an IoT device and its firmware image. IoTScope first extracts filenames and pathnames by statically analyzing firmware, and assembles them to construct HTTP probing requests. The probing requests are then used to interact with the physical IoT device or an emulator that hosts the firmware. Then IoTScope collects the requests and responses in order to filter out invalid interfaces and protected interfaces that are not of our interest. In the end, the identification of two types of hidden interfaces is conducted respectively on the remaining unprotected interfaces. The outputs of the system are the vulnerable hidden interfaces within the firmware.

3.1 Enumerating Interfaces

As illustrated in Figure 3, IoTScope extracts structural strings of pathnames and filenames from firmware images, and aggressively concatenates pathnames and filenames to construct syntactically valid URLs as probing requests.

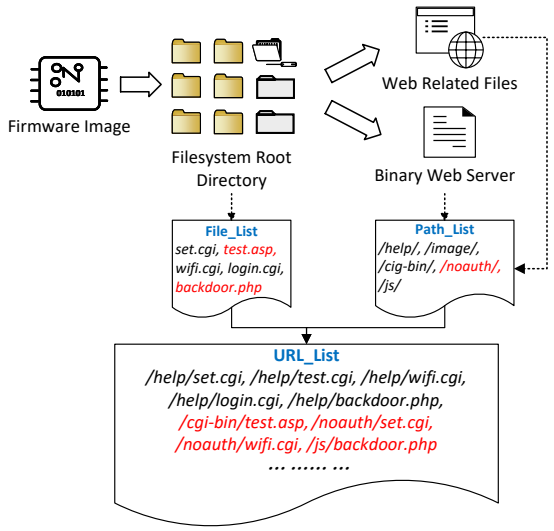

Figure 3: Illustration of Interface Enumeration

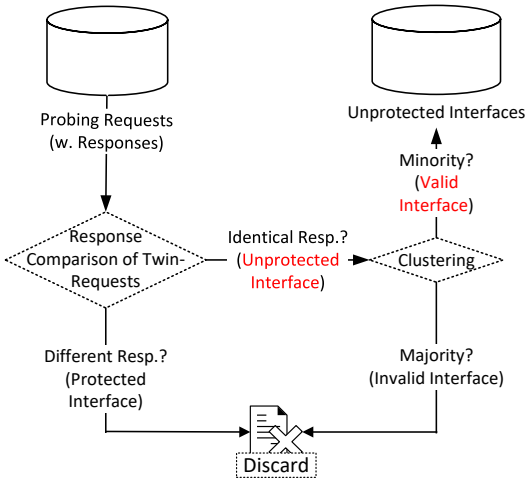

Figure 4: Identification of Unprotected Interface

First of all, we gather searchable information and use regular expressions to enumerate filenames. To make the string data such as filenames of executables and text information within executables (e.g., function names, debugging symbols, and printable strings) consistent and searchable, we decompress the firmware image and repack the root directory of the entire firmware filesystem into a single file. After that, we extract all the strings from the single file and use regular expressions to extract all possible filenames of web interfaces, namely the strings ended with `.cgi`, `.php`, `.asp`, `.xml`, `.htm` and `.html`.

Next, we repack web-related scripts and executables of web servers into a single file, and search for possible pathnames using regular expressions. By repacking only a subset of the whole filesystem, we can exclude the paths that are irrelevant to web services (e.g., "`/etc/`" and "`/proc/`"). Finally, by concatenating filenames and pathnames according to a customizable policy, IoTScope produces a list of probing requests.

Note that even if the web server constructs a probing request at runtime, the sub-strings of the probing requests are already contained in the executable. Thus, our approach can still reconstruct the URL that can only be constructed dynamically. Suppose a web server constructs a specific URL at runtime by concatenating three paths "`/cgi-bin/`", "`/image/`", and "`/auth/`", and there is no single path "`/cgi-bin/image/auth/`" in the firmware. In that case, our customizable policy allows users to increase the number of probing requests by concatenating three paths as one single path or treat one sub-string along (i.e., "`/cgi-bin/`", "`/image/`", or "`/auth/`") as the prefix of filenames.

3.2 Delivering Probing Requests

This component is responsible for sending and receiving HTTP packets to/from a target device/emulator. Our approach does not require the instrumentation of the target firmware to collect execution feedback. Instead, it only observes feedback from response messages. As a result, the target can be an emulator that hosts the firmware or a physical device. Furthermore, IoTScope runs a single thread to avoid concurrency issues where the following request would not be sent out until the reception of the previous response.

IoTScope sends each probing request twice. The first request carries a certificate in the authenticated HTTP head, while the second request does not carry a certificate. We call them twin-requests. The twin-requests with their corresponding responses are all fed to the next component.Since we own the tested device, there is no need to extract the default certificate from firmware. Instead, we can set it by ourselves and sniff its encoded version from a web proxy like Burpsuit. Typically, the certificate would be sent in the header of an authenticated HTTP request, either in the COOKIE field or the AUTHENTICATION field. We only need to set and capture it once when testing each device.

3.3 Identifying Unprotected Interfaces

Figure 4 illustrates the design of this component. At first, it identifies and filters out protected interfaces that require user authentication by observing the difference of the twin-requests' response. A pair of twin-requests (with/without a certificate) targeting the same protected interface would result in two different responses. The request with a certificate would get an indicator of successful access. The other request that does not carry a certificate would get a warning of unauthenticated or unauthorized access. However, when a pair of twin-requests are both invalid, identical responses are replied to. For example, both of the responses indicate the targeted URL does not exist.

To further filter out invalid probing requests, we cluster the remaining responses into different groups based on the content of response body. The responses of invalid requests are the majority and can be clustered into several groups, corresponding to the several errors handling cases in the webserver. The outliers are unprotected interfaces, namely open interfaces and hidden interfaces.

The detailed clustering algorithm is shown in Algorithm 1. It is lightweight and does not require a pre-defined number of groups. We define the similarity of two elements a and b, namely the content of two responses as: $Similarity(a, b) = 2 * M_{ab}/(L_a + L_b)$, where L_a and L_b are the lengths of the two elements. M_{ab} is the length of common characters that are shared by a and b. By definition,

this similarity function returns a value between 0 to 1. Similar responses are often highly-structured and share a large portion of common string characters. So similar strings tend to gather in the sample space. Using this similarity measurement can effectively differentiate responses of common invalid requests and responses of scarce valid requests. Given the large gap between the amount of invalid requests and the amount of valid requests, the value of the threshold is relatively insensitive to the clustering results.

Algorithm 1: Response Clustering Algorithm

```
for i in [1, N] do
    if E_i not in any group then
        create GROUP_i;
        put E_i in GROUP_i;
        for j in [i + 1, N] do
            if E_j not in any group then
                compute S = Similarity(E_i, E_j)
                if S > threshold then
                    put E_j in GROUP_i;
```

After clustering, a response group whose size is significantly larger than others would be marked as "majority" and be treated as responses of invalid requests. The intuition is that most probing requests are invalid, and the amount of requests to hidden interfaces and open interfaces is relatively small. For example, to identify the hidden interface with URL suffix "`/noauth/set-dns.cgi`", hundreds of candidate directories would be enumerated to generate probing requests. However, only one of the candidates, namely "`/noauth/`", would result in a response answered by the function "`set-dns_cgi`" of the EWA server. Other requests would be answered by the function "`load-URL_cgi`", which returns error messages like "`/XXX/set-dns.cgi` does not exist". Those responses with error information would be clustered as a large group and be discarded.

3.4 Identifying Hidden Interfaces

Once the protected interfaces and invalid interfaces are discarded, the goal of this step is to identify hidden interfaces that pose threats to users. As described in Section 2, there are two types of hidden interface: the hidden interface that allows manipulation of device setting, and the other type of hidden interface that discloses sensitive information about the IoT device.

Identification of hidden device-setting interfaces. IoTScope probes each unprotected interface by adding parameters extracted from firmware. As the backend server exposes functionalities to users through frontend code, nearly all of the device-setting parameters are hardcoded in the frontend. To extract the parameters, we scan the frontend code in the firmware images. If the communication with the backend server is based on AJAX, we directly extract parameters in the "`data`" field of AJAX requests and the arguments of the "`$.post()`" function. If the parameters are submitted to the backend using HTML forms, we use regular expressions to extract the field of action attribute and input label. The extracted key-value pairs are stored in a database.

Table 1: Evaluated Devices

Device ID	Vendors	Models	Device Types	Firmware Versions
1	Amcrest	IP2M841	Camera	V2.800.0000000.1.R
2	Asus	4G-AC55U	4G router	3.0.0.4.380_8102
3	D-link	Dir-868L	Wi-Fi router	2.03 B1
4	D-Link	DIR-412	Wi-Fi router	A1-1.14WW
5	D-Link	DIR-816	Wi-Fi router	A1 1.06
6	D-Link	DAP-1320	Repeater	A2-V1.21
7	H3C	MAGIC	Wi-Fi router	V100R006
8	Mercury	MIPC372-4	Camera	1.0.1
9	Mercury	MNVR408	Video recorder	1.0.9
10	Netcore	G1	4G router	V3.0.4.156
11	Netgear	PLW1000	Powerline adapter	1.0.1.6
12	Netgear	W104	Repeater	1.0.4.13
13	Netgear	WNDR4000	Wi-Fi router	V1.0.2.2_9.1.8468
14	Qihoo360	F5C	Firewall router	V3.1.1.65150
15	Tenda	G103	GPON modem	V1.0.0.5
16	TP-Link	GP110	GPON modem	3.2.2.1 build 141119 Rel.74551n
17	Wavlink	AC1200/A42	Wi-Fi router	1.27.6 (201806221623)

To automatically detect whether the probing request with a certain parameter takes effect, we send out two requests: the first request attaches a parameter that probably changes the device setting; the subsequent request does not attach the parameter. If the responses of the two requests are different, it indicates that the request with the parameter takes effect on the device side, by either changing the device setting or enquiring the status of the device. As it is difficult to automatically determine whether the request is about device setting or status enquiry, we manually analyze the contents of the candidates' responses as the last step.

Identification of hidden information-disclosure interfaces. To identify information-disclosure interfaces, we build a dictionary of keywords and match the content of the interfaces with the keywords in our dictionary. The keywords come from the following sources:

- **NVRAM parameters**. NVRAM values are a type of parameter that reside in devices rather than firmware. They are often about device setting or user-specific sensitive information, like "`lan_hwaddr`", "`wlan1_psk_cipher_type`", "`wlan1_psk_pass_phrase`", etc. We collect NVRAM-related keywords from NVRAM libraries like NVRAM faker [16] that contains NVRAM parameters to facilitate firmware emulation.
- **Configuration files.** We also extract key-value pairs about web server configuration from configuration files of the web servers such as "`host.conf`", "`lighttpd.conf`", and "`resolv.conf`". A set of configuration-related keywords are extracted, like "`root`", "`username`", "`groupname`", "`mod_auth`", "`accesslog`", etc.

After deduplication, we build up a dictionary with over 50 keywords. For the text of the response content, when at least two keywords are matched in the dictionary, IoTScope reports it as an information-disclosure interface. This procedure introduces false positives, which is discussed in Section 4.

4 EVALUATION

IoTScope is evaluated on a PC with an Intel i7 processor and 16GB RAM, running Ubuntu 20.04. All the testing targets are physical IoT devices that we purchased. The target devices are representative, including 17 models from 11 well-known vendors, running firmware with the latest version. Device types include cameras,

Table 2: Confirmed Vulnerabilities with CVE-ID

Device ID	CVE	Zero-day Vulnerabilities	Identified by		Description
			IoTScope	Firmadyne	
3	CVE-2019-7642	√	✓		Leak intranet users' DNS query logs and login logs.
	CVE-2019-17506	√	✓	✓	Leak the router's device information include username and password of the admin account.
4	CVE-2019-17511	√	✓		Access the router's log file about the intranet network structure.
	CVE-2019-17512	√	✓		Clear the router's log file to erase attack traces.
5	CVE-2019-17507	√	✓		Access management pages of the router by removing redirection code.
6	CVE-2019-17505	√	✓	✓	Leak the router's Wi-Fi SSID and password.
13	CVE-2017-5521		✓		Leak the admin account of the router to bypass authentication.
	CVE-2019-17372	√	✓		Disable the router's authentication protection of all pages.
14	CVE-2019-3404	√	✓		Query and modify a range of firewall configurations.

video recorders, GPON modems, powerline adapters, repeaters, 4G routers, Wi-Fi routers, and firewall routers. IoTScope does not have particular requirements on device types as long as they have web interfaces with firmware available. Table 1 lists details of the target devices.

4.1 Overall Results

Identified Vulnerabilities. With the help of IoTScope, we confirmed 44 vulnerabilities in total, including 43 previously unknown vulnerabilities and 1 known vulnerability (the firmware has not been updated for a long time). After responsibly reporting the newly-found vulnerabilities to the corresponding vendors, 8 of them have been confirmed and assigned CVE IDs. Table 2 gives a brief description of each vulnerability having a CVE ID. Without any protection, those vulnerabilities cause serious consequences, ranging from obtaining device logs to completely controlling the device.

Statistics and accuracy. As shown in Table 3, regarding the enumeration of possible interfaces, IoTScope on average extracts 117 paths and 326 files from each device's firmware, producing an average of 62357 URLs as probing requests. After filtering the invalid and protected interfaces, the rest of the probing requests are clustered into groups. On average, we obtain 4, 7, and 15 response clusters for one device and the results of vulnerability identification remain the same, when setting the similarity threshold to 0.4, 0.6, and 0.8, respectively. This is attributed to the fact that the clustering algorithm can tolerate false negatives and false positives when the threshold is set to different values. For example, if the threshold is set to a large value, there are more requests belonging to the "minority". The false positives can be eliminated during the identification of hidden interfaces because the difference between the responses of twin-requests (with/without parameters) is verified for hidden device-setting interfaces and the contents of responses are matched with the keywords for hidden information-disclosure interfaces, which directly exclude invalid interfaces. The final results would not be affected when the threshold is empirically set to a value larger than 0.4.

The results of hidden interface identification are provided in Table 4. IoTScope reports 20 cases of unauthenticated device setting and 48 cases of sensitive information disclosure. Nevertheless, after manual triage, we confirmed that there are 44 vulnerable interfaces in total. The false positives occur due to the following reasons: (1) for device-setting interfaces, the difference in the responses does not always indicate a successful device setting operation. It could be enquiry of device status; (2) for information-disclosure interfaces, the interface that matches the keywords does not always leak sensitive information. It could be an open interface that allows users to login with user account or reset the password.

Table 3: Statistics of Req. Generation and Res. Clustering

Device ID	# Path	# File	# URL	# Clusters		
				thd=0.4	thd=0.6	thd=0.8
1	540	240	129600	7	15	49
2	24	97	2328	1	1	1
3	598	1028	614744	7	14	37
4	201	433	87033	4	10	13
5	16	266	4256	2	6	19
6	43	164	7052	6	8	12
7	73	538	39274	4	5	9
8	22	86	1892	1	2	2
9	28	100	2800	3	5	5
10	104	470	48880	7	16	21
11	51	140	7140	6	11	42
12	32	640	20480	5	6	6
13	50	563	28150	2	6	16
14	121	471	56991	7	13	20
15	32	92	2944	2	2	3
16	15	93	1395	1	1	1
17	44	116	5104	4	5	7
Average	117	326	62357	4	7	15
Total	1994	5537	1060063	69	126	263

Table 4: Results of Hidden Interface Identification

Device ID	Device-Setting Interfaces	Confirmed	Info.-Disclosure Interfaces	Confirmed	Vulnerability
1	0	0	4	2	2
2	0	0	0	0	0
3	0	0	15	13	13
4	3	2	5	4	6
5	0	0	2	1	1
6	0	0	2	1	1
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	3	3	5	5	8
11	0	0	4	1	1
12	0	0	1	1	1
13	11	2	5	3	5
14	3	3	3	2	5
15	0	0	0	0	0
16	0	0	0	0	0
17	0	0	2	1	1
Total	20	10	48	34	44

Table 5: Comparison of IoTScope and Firmadyne

Device ID	# Path		# File		# URL		# Vulnerability	
	IoTScope	Firmadyne	IoTScope	Firmadyne	IoTScope	Firmadyne	IoTScope	Firmadyne
3	598	1	1028	175	614744	175	13	8
4	201	1	433	287	87033	287	6	0
6	43	1	164	73	7052	73	1	1
13	50	1	563	339	28150	339	5	0
15	32	1	92	82	2944	82	0	0
16	15	1	93	33	1395	33	0	0
Average	157	1	396	165	123553	165	4.2	1.5
Total	939	6	2373	989	741318	989	25	9

Table 6: Performance of IoTScope

Device ID	Generation (s)	Interaction (s)	Filtering (s)	Identification (s)	Total (s)	
					IoTScope	Firmadyne
1	4.23	12883.04	43.07	0.02	12930.36	-
2	9.86	144.59	0	0.01	154.46	-
3	9.39	5933.52	54.52	0.38	5997.81	861.54
4	5.63	1751.18	1.43	1.66	1759.9	626.14
5	1.19	274	6.6	0.03	281.82	-
6	1.26	3252.05	0.32	2.2	3255.83	362.24
7	3.39	3943.71	68.3	0.01	4015.41	-
8	1.71	166.98	0.08	0.01	168.78	-
9	2.22	288.32	0.14	0.02	290.7	-
10	2.22	4481.66	1.67	3.45	4489	-
11	7.94	837.57	7.23	3.31	856.05	-
12	2.68	1922.4	5.56	1.36	1932	-
13	2.99	947.92	352.71	18.23	1321.85	612.29
14	2.11	10030.24	12.39	7.12	10051.86	-
15	1.87	364.62	0.03	0.26	366.78	756.94
16	0.98	100.33	0.02	0.57	101.9	374.33
17	1.41	491.37	0.31	0.01	493.1	-
Average	3.59	2813	32.61	2.27	2848.76	598.91
Total	61.08	47813.5	554.38	38.65	48428.96	3593.48

4.2 Performance

Table 6 shows the time spent on each procedure for each device. The least time to test a target is only 101 seconds, while the longest is 12,930 seconds (3.59 hours). It indicates that IoTScope is fast and efficient in testing IoT devices. On average, the procedures of request generation, request filtering, and hidden interface identification occupy 0.12%, 1.14%, and 0.08% of the total time cost, respectively. However, the communication procedure, which includes sending requests and waiting for responses, costs 98.66% of the testing time. It is because the experiments were conducted with physical devices, which were relatively slower than emulator-based testing regarding processing requests and responses. Still, this brings the benefit that we do not need to set up an emulation environment that is hard-to-build [24, 29]. Also, the testing can be automatically resumed in the presence of certain device states (e.g., reboot) and communication states (i.e., the presence of tokens to maintain certain states). Although IoTScope can be used in emulator-based testing, firmware emulation is out of the scope of this research.

Comparison with Firmadyne. Firmadyne is a recent work that can be used to identify web-related vulnerabilities in IoT devices. Table 5 details the comparison between IoTScope and Firmadyne [5], given the same set of target devices/firmware. The comparison of performance is shown in Table 6. We also leverage FirmAE [20], a recent work that fixes partial emulation issues of Firmadyne. Unfortunately, as an emulator-based solution, Firmadyne (improved by FirmAE) is less scalable and can only emulate and test 6 out of 17 devices (35%). On average, IoTScope can cover 157 paths for each device, while Firmadyne only deals with the root web paths. IoTScope extracts 396 filenames, which is 2.4 times than the number of filenames extracted by Firmadyne. The main reason is that Firmadyne does not cover interfaces specified in functions of executables. In the end, compared with totally 9 vulnerabilities identified by Firmadyne, 25 vulnerabilities in total were reported by IoTScope.

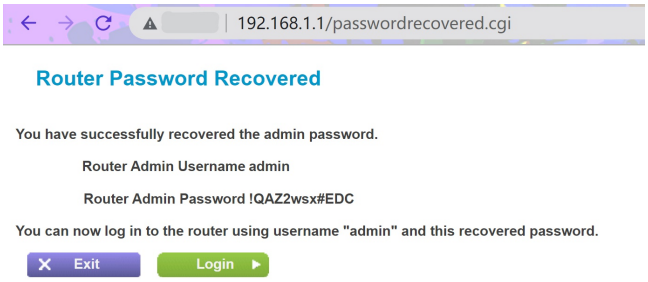

Figure 5: Vulnerable Hidden Page of CVE-2017-5521

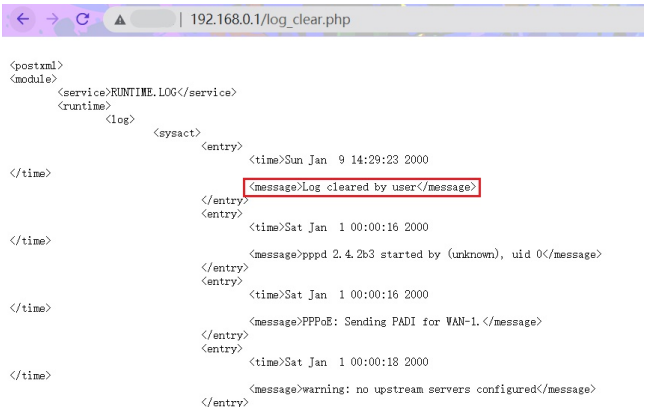

Figure 6: Vulnerable Hidden Page of CVE-2019-17512

4.3 Case Studies

Case study 1: a series of netgear devices. CVE-2017-5521 [10] reported that an unauthenticated attacker could get the admin password of a router by accessing the hidden interface "`passwordrecovered.cgi`", shown in Figure 5. this vulnerability affected 13 Netgear device models. The hidden interface "`passwordrecovered.cgi`" is not a web file but a function within the binary web server. IoTScope discovered this vulnerability as

it extracts strings inside binary web servers to generate probing requests. The matched keywords are "admin", "username", and "password". In the experiments, we found that this vulnerability affects the WNDR4000 device model, which is not included in the report of CVE-2017-5521.

Case study 2: a D-Link router. We reported CVE-2019-17512 [13], a vulnerability affecting DIR-412 router of D-Link. IoTScope extracted the parameters of page "`log_clear.php`": {"act": "clear", "logtype": "this.logType", "SERVICES": "RUNTIME.LOG"}. The parameters reside in JavaScript code which sends Ajax requests. In the first request, no content is received in the response as no parameters are attached. For the second time, when IoTScope sends the request with the parameters, the router's log file is cleared and the information shown in Figure 6 is returned in the response.

4.4 Discussion

Scope of detection. IoTScope can only detect hidden interfaces based on URLs. Although we believe that covers most cases, there are also a few cases that can be triggered only by submitting particular HTTP parameters. This is related to other authorization vulnerabilities like insecure direct object references. Besides, certain device states (e.g., reboot) and communication states (i.e., the presence of tokens to maintain certain states) may have some impacts on the testing of IoTScope. However, recent works have addressed this issue [34, 37, 38], therefore, not the focus of this paper.

Manual verification. Hidden interfaces are a type of logic vulnerabilities. There lacks a unified criterion or indicator, as the vulnerabilities often lead to various behaviors or consequences. Therefore, manual verification to triage the identification outcome is unavoidable. Luckily, after keywords matching or differential analysis of responses, manual triage is simple and straightforward for the outcome of IoTScope. Analysts only need to examine the reported cases by taking a glance at the text information on the web interfaces and determine whether they are vulnerable. Moreover, since the responses are clustered, we only need to verify one response for each cluster.

Legality & ethicality. This research does not cause any legal or ethical concerns. We have ownership of all devices that we purchased and have responsibly reported all vulnerabilities to "cve.mitre.org" and the corresponding vendors. We confirmed that the vendors have already patched the vulnerabilities in the case studies we detailed in this paper.

5 RELATED WORK

In this section, we first review the related work on IoT device testing. Then we discuss how prior works automatically detect authentication and authorization problems.

5.1 IoT Device Testing

IoTScope is based on dynamic analysis. Recently there is a line of research that leverage dynamic techniques to identify vulnerabilities in IoT devices [5, 6, 19, 30, 32, 40]. For example, Chen et al. [5] proposed FIRMADYNE, an automated framework to identify memory corruptions and web-related vulnerabilities by emulating firmware. With the help of this tool, the authors confirmed 14 previously unknown vulnerabilities that affected 69 firmware images. A similar work is proposed by Costin et al. [9] which focuses on emulating embedded web servers and identifies web-related vulnerabilities such as XSS and CSRF. However, those two tools are not designed for identifying hidden interfaces and therefore do not directly suit our case. Srivastava et al. [32] proposed Firmfuzz, an automated device-independent emulation and dynamic analysis framework for Linux-based firmware images. It leverages a greybox fuzzing approach coupled with static analysis and system introspection. Zheng et al. [40] proposed FirmAFL, a high throughput greybox fuzzer for IoT firmware. It leverages a novel technique called augmented process emulation, which combines system-mode emulation and user-mode emulation to improve the throughput of fuzzing. Chen et al. [6] proposed IOTFUZZER, a framework that aims at finding memory corruption vulnerabilities in IoT devices through over-the-air fuzzing. It reuses program-specific logic to mutate test cases to probe IoT devices. However, the mentioned work only targets memory corruptions and is not suitable for exposing hidden interfaces.

5.2 Detection of Broken Access Control

There are some related works that aim to detect broken access control in cloud services [2, 7, 41–45]. Zhou et al. [41] and Chen et al. [7] assessed the interactions among IoT apps, IoT devices and IoT clouds. They systematically decomposed the process of IoT device binding and exposed authentication and authorization problems of the protocols through manual analysis. AutoForge [44] is a tool that automatically forges valid request messages from mobile apps to test whether the server side of an app has ensured the security of user accounts with sufficient checks. AuthScope [45] automatically executes a mobile app and pinpoints the vulnerable access control implementations, particularly the vulnerable authorizations, in the corresponding online service. It uses differential traffic analysis to recognize the protocol fields and automatically substitute the fields and observe the server response. Although the above works aim to identify broken access control, they are only applicable to particular targets like mobile and IoT cloud backends. On the technical side, Zuo et al. [42] proposed SmartGen, which leverages symbolic execution to construct URLs from mobile apps, whereas IoTScope constructs URLs from firmware with string analysis. Comparing with tools [15, 27, 33] that analyze general web applications, they focus on authentication and authorization problems of visible and protected interfaces, rather than hidden interfaces.

6 CONCLUSION

In this paper, we have presented the first automated tool IoTScope that exposes hidden interfaces in embedded web applications of IoT devices. We designed a principled solution to achieve our goal, by constructing probing requests through firmware analysis to test physical devices, narrowing down the scope of identification by filtering out irrelevant requests, and pinpointing two types of hidden interfaces. By conducting experiments in the real environment, IoTScope successfully identified 44 vulnerabilities in 17 real-world IoT devices.

REFERENCES

[1] [n. d.]. Smart Yet Flawed: IoT Device Vulnerabilities Explained. [Online]. Avaliable: https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/smart-yet-flawed-iot-device-vulnerabilities-explained.

[2] Omar Alrawi, Chaoshun Zuo, Ruian Duan, Ranjita Pai Kasturi, Zhiqiang Lin, and Brendan Saltaformaggio. 2019. The betrayal at cloud city: An empirical analysis of cloud-based mobile backends. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 551–566.

[3] Anastasios Arampatzis. [n. d.]. Top 10 Vulnerabilities that Make IoT Devices Insecure. [Online]. Avaliable: https://www.venafi.com/blog/top-10-vulnerabilities-make-iot-devices-insecure.

[4] boa. 2005. Boa Webserver. [Online]. Avaliable: http://www.boa.org.

[5] Daming D. Chen, Manuel Egele, Maverick Woo, and David Brumley. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *Network and Distributed System Security Symposium*.

[6] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, Xiao Feng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing. In *Network and Distributed System Security Symposium*.

[7] Jiongyi Chen, Chaoshun Zuo, Wenrui Diao, Shuaike Dong, Qingchuan Zhao, Menghan Sun, Zhiqiang Lin, Yinqian Zhang, and Kehuan Zhang. 2019. Your iots are (not) mine: On the remote binding between iot devices and users. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 222–233.

[8] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. 2018. DTaint: detecting the taint-style vulnerability in embedded device firmware. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 430–441.

[9] 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*. 437–448.

[10] CVE-2017-5521. 2017. A vulnerability of password disclosure affecting a series of Netgear devices. [Online]. Avaliable: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5521.

[11] CVE-2018-11510. 2018. An unauthenticated RCE (Remote Code Execution) vulnerability affecting a NAS device. [Online]. Avaliable: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11510.

[12] CVE-2019-14984. 2019. Some smart home central control units allow unauthenticated attackers to run system commands by accessing an undocumented web interface. [Online]. Avaliable: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14984.

[13] CVE-2019-17512. 2019. A vulnerability affecting a router of D-Link. [Online]. Avaliable: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17512.

[14] Drew Davidson, Benjamin Moench, Somesh Jha, and Thomas Ristenpart. 2013. FIE on firmware: finding vulnerabilities in embedded systems using symbolic execution. In *Usenix Conference on Security*. 463–478.

[15] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. 2020. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1953–1970.

[16] NVRAM Faker. 2021. A Common Library of NVRAM Parameters for Firmware Emulation. [Online]. Avaliable: https://github.com/zcutlip/nvram-faker.

[17] 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*. 480–491.

[18] Grant Hernandez, Farhaan Fowze, Dave Tian, Tuba Yavuz, and Kevin RB Butler. 2017. Firmusb: Vetting usb device firmware using domain informed symbolic execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2245–2262.

[19] Yikun Jiang, Wei Xie, and Yong Tang. 2018. Detecting Authentication-Bypass Flaws in a Large Scale of IoT Embedded Web Servers. In *Proceedings of the 8th International Conference on Communication and Network Security*. ACM, 56–63.

[20] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. 2020. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In *Annual Computer Security Applications Conference*. 733–745.

[21] lighttpd. 2021. Home - Lighttpd - fly light. [Online]. Avaliable: https://www.lighttpd.net.

[22] Knud Lasse Lueth. [n. d.]. Top 10 IoT applications in 2020. [Online]. Avaliable: https://iot-analytics.com/top-10-iot-applications-in-2020.

[23] mini httpd. 2018. mini-httpd - small HTTP server. [Online]. Avaliable: https://acme.com/software/mini_httpd.

[24] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, and Davide Balzarotti. 2018. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices.. In *NDSS*.

[25] OWASP. 2021. The OWASP Top 10 2021. [Online]. Avaliable: https://owasp.org/Top10/.

[26] Danny Palmer. [n. d.]. These new vulnerabilities put millions of IoT devices at risk, so patch now. [Online]. Avaliable: https://www.zdnet.com/article/these-new-vulnerabilities-millions-of-iot-devives-at-risk-so-patch-now/.

[27] Giancarlo Pellegrino and Davide Balzarotti. 2014. Toward Black-Box Detection of Logic Flaws in Web Applications.. In *NDSS*.

[28] 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*. IEEE, 709–724.

[29] Abdullah Qasem, Paria Shirani, Mourad Debbabi, Lingyu Wang, Bernard Lebel, and Basile L Agba. 2021. Automatic Vulnerability Detection in Embedded Devices and Firmware: Survey and Layered Taxonomies. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–42.

[30] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2021. DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices. In *42nd IEEE Symposium on Security and Privacy 2021*.

[31] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2017. Bootstomp: on the security of bootloaders in mobile devices. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 781–798.

[32] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. 2019. FirmFuzz: automated IoT firmware introspection and analysis. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. 15–21.

[33] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli. 2017. Large-scale analysis & detection of authentication cross-site request forgeries. In *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 350–365.

[34] Enze Wang, Baosheng Wang, Wei Xie, Zhenhua Wang, Zhenhao Luo, and Tai Yue. 2020. EWVHunter: Grey-Box Fuzzing with Knowledge Guide on Embedded Web Front-Ends. *Applied Sciences* 10, 11 (2020), 4015.

[35] Wei Xie, Chao Zhang, Pengfei Wang, Zhenhua Wang, and Qiang Yang. 2021. ARGUS: Assessing Unpatched Vulnerable Devices on the Internet via Efficient Firmware Recognition. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 421–431.

[36] 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*. 363–376.

[37] Qidi Yin, Xu Zhou, and Hangwei Zhang. 2021. FirmHunter: State-Aware and Introspection-Driven Grey-Box Fuzzing towards IoT Firmware. *Applied Sciences* 11, 19 (2021), 9094.

[38] Hangwei Zhang, Kai Lu, Xu Zhou, Qidi Yin, Pengfei Wang, and Tai Yue. 2021. SIoTFuzzer: Fuzzing Web Interface in IoT Firmware via Stateful Message Generation. *Applied Sciences* 11, 7 (2021), 3120.

[39] Li Zhang, Jiongyi Chen, Wenrui Diao, Shanqing Guo, Jian Weng, and Kehuan Zhang. 2019. CryptoREX: Large-scale Analysis of Cryptographic Misuse in IoT Devices. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*. 151–164.

[40] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. 2019. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1099–1114.

[41] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. 2019. Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1133–1150.

[42] Chaoshun Zuo and Zhiqiang Lin. 2017. Smartgen: Exposing server urls of mobile apps with selective symbolic execution. In *Proceedings of the 26th International Conference on World Wide Web*. 867–876.

[43] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. 2019. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1296–1310.

[44] Chaoshun Zuo, Wubing Wang, Zhiqiang Lin, and Rui Wang. 2016. Automatic Forgery of Cryptographically Consistent Messages to Identify Security Vulnerabilities in Mobile Services.. In *NDSS*.

[45] Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin. 2017. Authscope: Towards automatic discovery of vulnerable authorizations in online services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 799–813.