Passwords in the Air: Harvesting Wi-Fi Credentials from SmartCfg Provisioning

Changyu Li
Shanghai Jiao Tong University
lichangyu@sjtu.edu.cn

Quanpu Cai
Shanghai Jiao Tong University
cpeggsjtu@sjtu.edu.cn

Juanru Li
Shanghai Jiao Tong University
jarod@sjtu.edu.cn

Hui Liu
Shanghai Jiao Tong University
ice_wisdom@sjtu.edu.cn

Yuanyuan Zhang*
Shanghai Jiao Tong University
yyjess@sjtu.edu.cn

Dawu Gu
Shanghai Jiao Tong University
dwgu@sjtu.edu.cn

Yu Yu
Shanghai Jiao Tong University
Westone Cryptologic Research Center
yyuu@sjtu.edu.cn

ABSTRACT

Smart devices without an interactive UI (e.g., a smart bulb) typically rely on specific provisioning schemes to connect to wireless networks. Among all the provisioning schemes, SmartCfg is a popular technology to configure the connection between smart devices and wireless routers. Although the SmartCfg technology facilitates the Wi-Fi configuration, existing solutions seldom take into serious consideration the protection of credentials and therefore introduce security threats against Wi-Fi credentials.

This paper conducts a security analysis against eight SmartCfg based Wi-Fi provisioning solutions designed by different wireless module manufacturers. Our analysis demonstrates that six manufacturers provide flawed SmartCfg implementations that directly lead to the exposure of Wi-Fi credentials: attackers could exploit these flaws to obtain important credentials without any substantial efforts on brute-force password cracking. Furthermore, we keep track of the smart devices that adopt such Wi-Fi provisioning solutions to investigate the influence of the security flaws on real world products. Through reversely analyzing the corresponding apps of those smart devices we conclude that the flawed SmartCfg implementations constitute a wide potential impact on the security of smart home ecosystems.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; *Mobile and wireless security*;

KEYWORDS

Smart devices, Wi-Fi provisioning

ACM Reference Format:
Changyu Li, Quanpu Cai, Juanru Li, Hui Liu, Yuanyuan Zhang, Dawu Gu, and Yu Yu. 2018. Passwords in the Air: Harvesting Wi-Fi Credentials from SmartCfg Provisioning. In *WiSec '18: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks, June 18–20, 2018, Stockholm, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3212480.3212496

1 INTRODUCTION

Wireless smart devices nowadays facilitate various aspects of our daily life. The applications of smart home, smart city, and smart automation significantly benefit from the wireless connection capability of those devices. Nonetheless, the wide use of wireless smart devices also introduces new attack vectors and increase the risk. A typical security threat is the weak authentication of the wireless network. The authentication is often the crucial prerequisite of many subsequent attacks. If the attacker could circumvent the authentication and log in the protected wireless network, devices in the same network area are exposed and face severe threats.

Although recent years have witnessed a series of attacks against the authentication of the Wi-Fi standards, most of the attacks aim to circumvent the authentication through cracking the login credentials (e.g., passwords) using a brute-force search [28], or utilizing weaknesses of the authentication protocols to decrypt the encrypted network traffics [29] [21]. In this paper, we present a new type of security threat caused by a recently introduced application scenario–the provisioning of smart devices. We observe that many smart devices are often headless devices that have been configured to operate without classical input devices such as monitor, keyboard, or mouse. To help those smart devices without an interactive UI to obtain the Wi-Fi credentials for wireless network authentication, wireless chip vendors design auxiliary provisioning solutions. However, these provisioning solutions less regulate the protection scheme, which often leads to the leak of credentials.

In this paper, we pay particular attention to SmartCfg, a provisioning technology designed to provide Wi-Fi credentials for smart

*Corresponding author.

WiSec '18, June 18–20, 2018, Stockholm, Sweden

ACM ISBN 978-1-4503-5731-9/18/06…$15.00
https://doi.org/10.1145/3212480.3212496

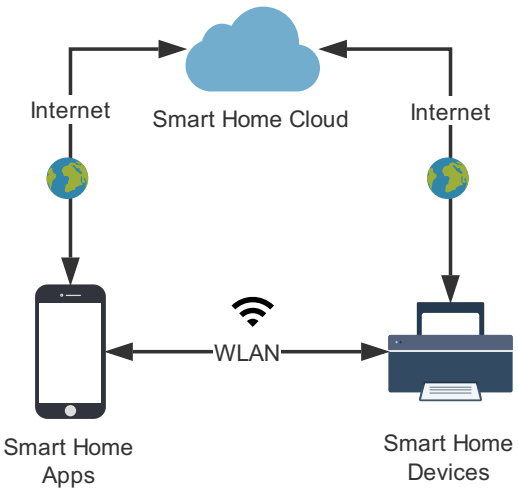

Figure 1: A typical structure of Smart Home Wi-Fi Configuration Process

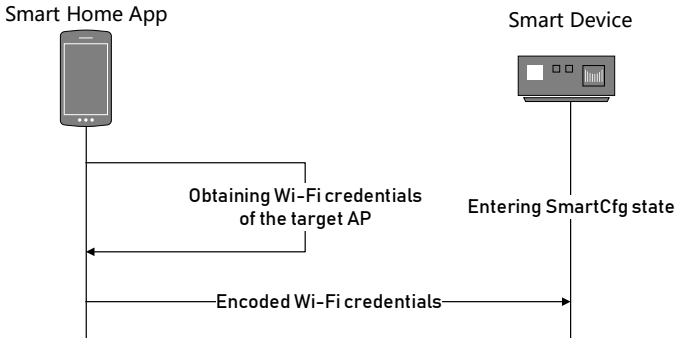

Figure 2: An illustration of SmartCfg provisioning process

home devices (e.g., plug, washer, refrigerator). For a SmartCfg provisioning solution, the credential information is encoded and piggybacked on specific broadcasting packets. Typically, a SmartCfg solution leverages an auxiliary mobile app to encode and broadcast the credentials (Wi-Fi passwords and their corresponding SSIDs). This broadcast is then captured by the smart device and the credentials are transmitted to it. To study the threats against current SmartCfg provisioning solutions, we first investigated eight widely used SmartCfg solutions of popular Wi-Fi chip vendors. Then we conduct a large-scale security analysis of real world products adopting those solutions. We leverage information extracted from the corresponding smart home app to identify which solution is used by a particular smart device. Furthermore, we propose two types of analysis to recover the proprietary credential encoding scheme of the SmartCfg solution used between the mobile app and smart home devices.

Through checking 821 apps related to real world smart home devices, we find that 64 apps integrate at least one SmartCfg solution. A further security analysis demonstrates that 42 apps contain severe security vulnerabilities due to either the insecurely designed SmartCfg solution (six out of eight analyzed solutions are insecure) or the incorrectly implementation caused by developers, and attackers could exploit such vulnerabilities to obtain the authentication credentials of wireless network.

In summary, this paper makes the following contributions:

- We study popular SmartCfg Wi-Fi provisioning solutions and analyze the security by auditing the SDKs provided by Wi-Fi chip vendors. We find that six out of eight studied solutions contain severe security issues resulting in the leak of Wi-Fi credentials.
- We propose a systematic security analysis against real world smart home devices using SmartCfg solutions. The analysis consists of how to identify SmartCfg solutions deployed in a smart device, and how to recover credentials encoding scheme using either binary code reverse engineering or differential traffic analysis.
- We present an overview of the ecosystem through analyzing 821 smart home apps. We find that 64 apps use SmartCfg provisioning solutions and among them, 42 apps are vulnerable. This indicates that a wide range of smart home devices are affected.

2 BACKGROUND

In this section, we first present a typical application scenario, namely, smart home with SmartCfg solution in place. Then we give a detail description of the SmartCfg technology.

2.1 Smart Home

Smart home technology, also known as home automation, provides homeowners the ability to remotely control smart devices at home. A smart home is a residence that applies related techniques on Internet-connected devices to enable remote monitoring and management of appliances and systems, such as controlling the switch, monitoring indoor temperature and humidity [3]. Typically, a smart home consists of three types of components: a set of smart devices, a smart home app, and a smart home cloud. Figure 1 depicts the relations between those components. Typical smart devices in a smart home include smart light bulbs, smart locks, smart plugs, etc. Those devices are often controlled through a smart home app on user's mobile phone (Android or iOS), which allows the user to control smart devices remotely. The smart home cloud often plays the role of bridging the smart home app and smart devices if they are not in the same LAN, and helps store and manage data collected from those smart devices.

Most smart devices nowadays adopt wireless network connection. To connect to the wireless network, a device integrates and utilizes a Wi-Fi module. However, most smart home devices are headless devices. That is, they have been configured to operate without a monitor, keyboard, or mouse. Therefore, a process of preparing and configuring the network, defined as the Wi-Fi provisioning, is required at the first time a smart home device is deployed.

Generally, a smart device cannot accomplish the provisioning by itself. Information about the network (e.g., the SSID and the corresponding password) should be provided via other devices with

interactive UI (e.g., a smartphone or a tablet). This provisioning procedure is a new application scenario and thus may introduce new attack vectors.

2.2 SmartCfg Provisioning

2.2.1 Overview. First introduced by Texas Instruments (TI) in 2012 [15], SmartCfg is a provisioning technology designed to provide Wi-Fi credentials for smart devices without an interactive UI (e.g., plug, washer, refrigerator). After five years of development, it is widely accepted by various wireless solution providers. Wireless chip manufacturers such as Realtek, MediaTek, MXCHIP, and Espressif have also implemented their variants. In this paper we use SmartCfg to refer to all those variants.

A SmartCfg provisioning is used to configure a smart device and help it connect to the wireless network. Typically, a SmartCfg solution possesses three typical features: First, it relies on a mobile app to encode and send authentication credentials to the smart devices through broadcasting. Second, smart devices passively listen to the encoded information without knowing the identity of the sender. Third, the credentials (e.g., passwords) are encoded as part of the metadata of 802.11 packet (e.g., packet length) rather than the content of the packet. As a result, even though the data field in the 802.11 packet frame is encrypted with either WEP or WPA2 [30], devices listening to the data in the air could intercept the information of metadata regardless of the actual data content.

2.2.2 Provisioning Process. One typical process of SmartCfg provisioning is illustrated in Figure 2. Before the provisioning, a smart device in promiscuous mode continuously captures all the packets in the network. When a provisioning procedure starts, the mobile app encodes the Wi-Fi credentials (both SSID and password) into several packets and broadcasts those packets. Once those packets are captured and decoded by the smart device, it uses this information to connect to the wireless network

(1) A smart device that supports SmartCfg technology enters its SmartCfg state, in which the Wi-Fi module of the device is enabled in a sniffer mode to receive broadcast information.
(2) A user use the smart home app on a smart phone to input the credentials (i.e., passwords). Then the smart home app encodes the SSID and password as 802.11 packets of special format, and sends them into the wireless network it connects.
(3) The Wi-Fi module of smart device captures all 802.11 data packets in a few seconds and tries to decode them using certain algorithm to obtain the SSID and password. After obtaining Wi-Fi credentials, the device then connects to the wireless network.

2.2.3 Data Encoding Mode of SmartCfg. SmartCfg solutions encode the credentials into the metadata of 802.11 packets. However, different solutions adopt different encoding modes. In general, there are three popular encoding modes as Table 1 illustrates:

s

- **Data in Multicast Addresses (DMA)** In this mode, the information is encoded into the last 23 bits of the *Destination Address* field of the packets. The DMA row of Table 1 gives a concrete example. In this example every two bytes of the payload are encoded into the last 2 bytes of the destination address, while the ante-penultimate byte of the address is used as the index.
- **Data in Packet Length (DPL)** In this mode, the information is encoded into the length field of each packet (the content is randomly filled). As illustrated in the DPL row of the Table 1, messages are encoded into the length of the packet sequence and then are broadcasted.
- **Hybrid** In this mode, both the *Destination Address* field and the packet length are used to encode the information. The *Destination Address* is usually used as the index and the message is often stored at the length field. The advantage of Hybrid mode is that one packet contains more information and it is thus more efficient.

Table 1: Examples of different types of SmartCfg

Mode	Source Address	Destination Address	Length
DMA	00:90:4c:17:1a:9b	01:00:5e:*01:49:6f*	43
	00:90:4c:17:1a:9b	01:00:5e:*02:54:36*	43
	00:90:4c:17:1a:9b	01:00:5e:*03:36:36*	43
	00:90:4c:17:1a:9b	01:00:5e:*04:37:38*	43
	00:90:4c:17:1a:9b	01:00:5e:*05:39:cc*	43
DPL	00:90:4c:17:1a:9b	FF:FF:FF:FF:FF:FF	*47*
	00:90:4c:17:1a:9b	FF:FF:FF:FF:FF:FF	*67*
	00:90:4c:17:1a:9b	FF:FF:FF:FF:FF:FF	*47*
	00:90:4c:17:1a:9b	FF:FF:FF:FF:FF:FF	*67*
	00:90:4c:17:1a:9b	FF:FF:FF:FF:FF:FF	*96*
Hybrid	00:90:4c:17:1a:9b	01:00:5e:*01:01:01*	*556*
	00:90:4c:17:1a:9b	01:00:5e:*02:02:02*	*555*
	00:90:4c:17:1a:9b	01:00:5e:*03:03:03*	*554*
	00:90:4c:17:1a:9b	01:00:5e:*04:04:04*	*291*
	00:90:4c:17:1a:9b	01:00:5e:*05:05:05*	*338*
	00:90:4c:17:1a:9b	01:00:5e:*06:06:06*	*198*

According to our observation, one important feature of SmartCfg data encoding is that it introduces a preamble. A preamble is a sequence of packets (e.g., three packets) with same length and content. The purpose of this preamble is to help devices locate the data payload. We find that in most provisioning procedures the app will first issue a preamble (or synchronization code) to the network. The advantage of sending the preamble sequence first is that the device could locate, by the preamble, the beginning of a series of packets that matches a particular protocol. Then it is able to extract data from these packets.

Another usage of the preamble is to help measure the padding length introduced by the encryption. Since the app does not know what is the exact length of an encrypted data packet of WPA or WPA2, it can only prepare a set of preamble packets and broadcast them. Then the device capturing and identifying the preamble could calculate the length of padding data. Therefore, the use of preamble helps devices adjust the packet length.

3 SECURITY ANALYSIS OF SMARTCFG

In this section, we highlight the security analysis against SmartCfg provisioning solutions with a focus on their Wi-Fi credential encoding schemes. Before introducing the concrete analysis procedure, we first illustrate the threat model and challenges. Then we detail

the process of SmartCfg security analysis, which aims to recover proprietary credentials encoding schemes used in real world smart home devices and identify those insecure ones.

3.1 Threat Models and Challenges

3.1.1 Threat Models. Since the most valuable data in a provisioning procedure is the password (and the corresponding Wi-Fi SSID), we assume that this credential is the main target for attackers. However, an attacker is assumed to be blocked outside the wireless local area network (WLAN) and cannot physically or remotely connect to the device (otherwise he already has access to the internal network). To obtain this credential, the attacker can only recover Wi-Fi credentials from the wireless data in the air (even though the data may be encrypted). We also assume that the attacker does not know in advance the used smart home device and mobile apps. But he can collect typical SmartCfg solutions beforehand and then conduct reverse engineering to recover the used Wi-Fi provisioning solution and its credentials encoding scheme.

In addition, a crucial requirement for a successful attack is that it must be fulfilled at the same time the device is executing the provisioning. Although the provisioning procedure does not last for a long time, we assume the attacker is always monitoring the wireless data and he can discover such behavior with a sniffing attack: the attacker only needs to prepare a device with a Wi-Fi module in the promiscuous mode; the device would be placed near the wireless network and continuously monitor 802.11 data packets in the air, attempting to decode potential provisioning packets using several decoding functions prepared in advance. Once some packets are successfully decoded, the attacker will then use the obtained credential to connect to the target Wi-Fi.

3.1.2 Challenges. To analyze the security of SmartCfg solutions used by those real world devices, the challenges we face may include:

- **How to identify the specific SmartCfg solution used by the smart home device**. Knowing which SmartCfg solution is used by the analyzed target significantly facilitates the subsequent security analysis. However, almost all analysis targets are real world smart home devices rather than development boards. Despite the knowledge of existing SmartCfg solutions (in the form of specifications and documentations), device manufacturers often do not label which solution is used for a certain device. Therefore, analyst should utilize different kinds of side-channel information (e.g., the model of used wireless chip) to help deduce the specific SmartCfg solution in use.
- **How to recover the used provisioning protocol, in particular, the credentials encoding scheme**. Even with the knowledge of which SmartCfg solution is adopted by the analyzed device, we still cannot determine the provisioning protocol. We found that most SmartCfg solutions provided by the chip vendors are only templates for the reference of device manufacturers. A manufacturer may modify the template to implement its own provisioning protocol (although this protocol is similar to the original one). To detail the credentials encoding scheme, analysts are required to conduct reverse engineering against the code of the smart home device (i.e., firmware and app) since the source code is often not provided by the manufacturer. In this case, an analyst may face binary code of different architectures (ARM, MIPS, Tensilica, etc.). It is challenging to employ an effective reverse engineering against the commercial off-the-shelf binary code. To make things worse, sometimes even the binary code of a device may not be obtained and the only available data for analysis is the network traffic. In this situation, recovering the credentials encoding scheme requires a more complex analysis.

3.2 SmartCfg Solution Identification

We utilize the software development kits (SDKs) of existing SmartCfg solutions as the prerequisite in our identification. Specifically, SDKs of SmartCfg can be divided into two categories: **device SDK** and **mobile app SDK**. Device SDKs are provided to facilitate the development of device firmware, while mobile app SDKs are integrated by the smart home app for establishing the device-smartphone communication. If a device (and its corresponding app) adopts a certain SmartCfg solution and integrates SDKs of this solution, we can leverage the feature of SDKs to identify the used solution.

We find that most chip vendors typically provide SDKs with both source code and documentations on their websites. Hence we could directly download them as the reference of our analysis. Although the collected SDKs include both the device SDKs and the mobile app SDKs, our reverse engineering only focuses on the mobile app SDKs for two reasons: First, a device SDK is usually used to help analyze the device firmware. However, if the manufacturer does not provide the firmware image and the flash memory is protected from being read, it is often infeasible to obtain the firmware. Moreover, firmwares of most smart devices are closed-source, and the reverse engineering of such binary images is often very time-consuming [8]. Second, protocol can be recovered through analyzing the mobile app SDK only. According to our observation, the communication protocol between the mobile app and the device can be recovered by either analyzing the device firmware or the mobile app. Obviously, the analysis of mobile app has the benefit of a series of well-developed reverse engineering tools.

Briefly, the identification of SmartCfg solutions starts with collecting apps from the app market and downloading mobile app SDKs from the websites of chip vendors. In order to check whether a collected app contains certain mobile app SDKs, we utilize libscout [4], a third-party library detector for Java/Android apps, to run a similarity test between the target app and the *jar* packages of the SDKs. We also use the native code library as important references. A native code library is usually compiled as a shared library (.so) and its exported functions are invoked via a Java Native Interface (JNI). We collect the exported function names in shared libraries of the mobile app SDKs as the references (note that we exclude those short function names and frequently used names such as `getSize`). Once the same function names are found in a native code library of the mobile app, a solution is identified.

3.3 Credential Encoding Scheme Recovering

3.3.1 Code Analysis. After we identify the used SmartCfg solution in an app, we further conduct a code reverse engineering to

recover the credentials encoding scheme. The reverse engineering consists of two steps:

Locating Credential Input Activity: To understand the logic of credential encoding, the `Activity` (`Activity` is a component of Android application, acting as an user interface.) of SSID and password input in an app is an important index. If this `Activity` can be located, the provisioning can then be triggered and the credential encoding functions can be located. Typically, this `Activity` obtains the currently used SSID through the API of Android, and obtains the Wi-Fi credential with a manual input. Therefore, we utilize these features to find such `Activity` in the app. We can search APIs for SSIDs in Activities which use `WifiManager` and `WifiInfo` to get the current Wi-Fi connection information.

Locating Credential Encoding Functions: Next, we aim to understand how the app encodes the data after it has obtained credential. Thus we conduct a forward data flow analysis to track the transformation of the credential with a modified Android emulator. We start from the credential variable from the located credential input `Activity`, keeping track its propagation until the app sends a tainted data to external environment (e.g., network). All functions involved in the data propagation are labeled as encoding functions. In particular, our analysis checks both Java code and native binary code. We set the sink of this data flow analysis as the member function `send` of `DatagramSocket/MulticastSocket` classes in Java code, and `send/sendto` in native code. After the locating of encoding function, a manual reverse engineering is then conducted by the analyst to recover the encoding algorithms.

3.3.2 Network Traffic Analysis. Interestingly, we find that, for many SmartCfg solutions, the credential can be recovered through even a simple network traffic analysis. The attacker does not have to obtain the SDK to understand the credential encoding. This is significant since some smart home device manufacturers modify the SDK to adopt their proprietary encoding schemes. In the following, we demonstrate an extraction of credential using network traffic data only, which indicates that an attacker is still able to conduct successful credential sniffing even though he cannot obtain the SDK.

(1) **Step-I: Sniffing**. To sniff the broadcast data in the air, we utilize a TL-WN722N USB wireless adapter produced by TP-Link to capture all 802.11 protocol packets. Then we use the `Scapy` python library to parse and record the captured packets. For the captured data, we use a triplet (source mac address, destination mac address, data length) to format them. Then we only keep the relevant packets with specific addresses. At the same time, we record the existing `SSID` and its corresponding `BSSID`.
(2) **Step-II: Data Payload Locating**. For different provisioning solutions, the data is encoded with several schemes. By observing the variances between different packets we can deduce the adopted encoding mode. By observing the captured triplets (source mac address, destination mac address, data length) we can guess the adopted mode. If the data length field in the triple is changing, then it is considered as the payload. Otherwise the payload is stored in the destination mac address field. For instance, if the destination mac address field is in the form of `01:00:5e:xx:xx:xx`, then we can guess the data is encoded in the changing part of the destination mac address field.
(3) **Step-III: Credential Differentiating** Since we have already located the data payload, next we conduct a differential traffic analysis to locate the position of the used credential in the packets. We test the app with two passwords of same length and trigger the provisioning, respectively. By comparing the changing part of the payloads, we can obtain a differential. However, the differential often not only contains the password field, but also a checksum field. To distinguish, we leverage the fact that if the password is changed slightly (e.g., one character is changed), the differential of the password field will also changes accordingly (since no encryption is used to protect the password field). However, the checksum field will change significantly (i.e., each byte is changed). This features helps us identify the password field effectively.

4 EXPERIMENTAL RESULTS

In this section, we present the results of our security analysis against a wide range of SmartCfg solutions.

4.1 Solution Identification

The prerequisite of our solution identification is the knowledge of typical SDKs of SmartCfg solution. We have collected eight representative SmartCfg solutions from the official websites of the chip vendors. The summary of those collected SDKs including SDK for mobile app, SDK for smart device, and corresponding documents, are listed in Table 2. In Model column of Table 2, we list out the mainstream chip models of the vendor using SmartCfg solution. Despite different names used by the solutions but have the same semantics, such as SmartConfig, SimpleConfig, EasyConfig, SmartConnection, EasyLink, they are actually implementations of SmartCfg (up to certain variance and adaption). A mobile app SDK provides a demo app and Java source code of the app with *jar* packages or shared libraries. An SDK for smart device includes some pre-compiled binary files that provide a basic implementation of the protocol, including the Wi-Fi provisioning process. Also, we collect documents related to the SDK, which help understand details of Wi-Fi provisioning.

After collecting these representative SmartCfg solutions, we further extract their features and utilize them to identify specific solutions in real world apps. A typical feature is the existence of certain library files, which can be found in the *jar* package or the *lib* directory. Since developers seldom change these solution files, it is feasible to use it as a feature to judge whether one specific solution is used by a real world product.

With the downloaded SDKs from the official websites, we filter out apps related to smart home from the largest app market in China–Tencent MyAPP app market [2]. We choose this app market because China has the most active smart home ecosystem and it is expected to find more smart home apps here. We developed a web crawlers to find smart home apps according to an app's keyword description. Obviously, the quality of the filtering results depends on the accuracy of the keyword. Hence we classified the keywords into

Table 2: An overview of eight popular SmartCfg solutions and their SDKs

Solution	Model	Chip Vendor	SDK for Mobile App	SDK for Smart Device
EasyLink	EMW3060	MXCHIP [18]	`EasylinkAndroid_Demo` [18]	`MiCOSDK`
BroadLinkConfig	BL3332-P	BroadLink [1]	`libBroadLinkConfig.so` [6]	-
Esptouch	ESP8266	Espressif [9]	`EsptouchForAndroid` [10]	`ESP8266_RTOS_SDK`
SimpleConfig	RTL8723BS	Realtek [23]	`Release_SimpleConfigWizard` [22]	`SDK_Ameba`
SmartConnection	MTK7681	MediaTek [17]	`LinkIt Connect 7681 SDK` [17]	`LinkIt Connect 7681 SDK`
SmartConfig	CC3200	TI [24]	`SmartConfigCC3X` [27]	`simplelink-cc3220-sdk`
DirectConfig	NL6621	NuFront [19]	`NL6621_StandardSDK` [20]	`NL6621_StandardSDK`
SmartLink	HF-LPX30	Hi-Flying [14]	`SmartLinkV7 (Android)` [14]	`HF-LPX30-HSF-SDK`

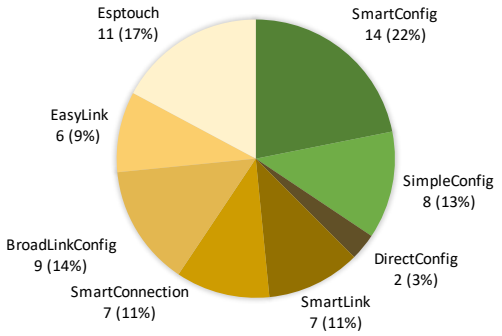

Figure 3: A percentage breakdown of the SmartCfg app

three categories: (1) smart home products, e.g., smart-plug, smart-hub, and smart-clock; (2) chip vendors, e.g., Espressif, MXCHIP; (3) known keywords, e.g., wulianwang (the Chinese pinyin that corresponds to "Internet of Things"), smarthome, and smartthings.

In order to further improve the keyword search method, we count the word frequency statistics of the app's package name and exclude some irrelevant words. Then we use the top 10 words as new keywords and iterate the process for three times. This brings us more relevant apps. Through the process, we obtain 821 smart-home related apps.

After setting up the crawler and downloading a large number of apps, we use apktool to decompress the app. Through checking the unzipped files and comparing them with the features collected from the SDK in advance, we can determine whether an app contains a SmartCfg solution mentioned above. A certain feature found in the app indicates the use of corresponding SmartCfg solution. Interestingly, we also find that many apps integrate more than one SmartCfg solutions. Among the 821 smart-home apps, we find 64 apps use SmartCfg solutions mention in Table 2. The distribution of different SmartCfg solutions among these apps is shown in Figure 3.

4.2 Encoding Scheme Analysis

We analyze the encoding schemes used in the collected SmartCfg apps especially focusing on their security issues. Since the encoded information is broadcasted by a smart home app and all smart devices could capture and decode it, an attacker who is eavesdropping the broadcast can also obtain such information. Hence we check whether the encoding scheme of the SmartCfg solution protects the credentials against illegal packet capturing.

We conduct a network traffic analysis against all collected apps to recover the encoding mode for each SmartCfg solution. Table 3 lists the result. We find only the SmartConnection solution developed by MediaTek adopts the DMA encoding mode. Three solutions (BroadLinkConfig, DirectConfig, and SmartConfig) adopt the DPL encoding mode while the left four solutions (EasyLink, Esptouch, smartLink, and SimpleConfig adopt the Hybrid encoding mode.

In addition, we examine the security of each encoding scheme through conducting a binary code reverse engineering against each collected app. The purpose of this reverse engineering is to verify whether the implementation of a concrete SmartCfg solution adopts a secure encoding scheme. Note that even if the SDK provided by the vendor does implement a secure (or insecure) encoding scheme, device manufacturers may modify it into an insecure (or secure) one. For instance, in samples of many SDKs the encryption function is implemented using a constant key. However, a manufacturer may re-implement this with a randomly generated key. Thus we should detail the implementation for each app to judge whether the adopted encoding scheme is secure.

According to our analysis, six out of the eight SmartCfg solutions provided by different chip vendors adopt insecure encoding schemes. Table 3 gives the detail of the insecure protection. In particular, four solutions (BroadLinkConfig, DirectConfig, Esptouch, SmartLink) do not adopt any encryption and directly broadcast the encoded data, two solutions (EasyLink, SmartConnection) encrypt the encoded data but they misuse the cipher, and only two solutions correctly encrypt the encoded data. Hence we guess the flawed solutions may significantly affect the security of those smart home apps. We then decompile each app with `apktool` and manually verify the encoding scheme used by each app.

Based on the code analysis presented in Section 3.3.1, we locate the encoding function and manually check whether the `SSID` and the password are encrypted. If they are sent without being encrypted, the host app is considered as an insecure one. If they are encrypted, we further check whether the cipher used to protected credential is correctly invoked. We check typical crypto misuses (e.g., constant crypto key, key reuse, key leaking) to find potentially flawed protection.

Unfortunately, among all analyzed apps, we find that every app utilizing an SDK with insecure encoding scheme does not enhance it and thus suffers from a credential eavesdropping attack. Among

the 64 collected apps, 42 apps adopt insecure encoding scheme of the original SDKs. There are 29 apps (45%) affected by four solutions (BroadLinkConfig, DirectConfig, Esptouch, SmartLink), which encode data without encrypting it. Seven apps adopt the SmartConnection solution provided by MediaTek, and execute the encryption with hard-coded AES keys. Six apps (9%) use the RC4 cipher to encrypt the payload regulated by the EasyLInk solution of MXCHIP, but they suffer from the key reuse issue.

What is worse, we find that even the adopted SmartCfg solution uses a secure encoding scheme, the implementation of the app may still misuse it. For instance, eight apps that adopt the SimpleConfig solution use fixed AES keys. We investigate the original solution and find the key used by the native library of the SDK relies on a `Pin code`, which is identical on different devices. However, some implementations may fail to acquire this `Pin code` and then they will use a default code as the key material. In this case, the generated key is constant and predictable.

In summary, 42 out of 64 analyzed apps (66%) contain insecure credential encoding schemes and any attacker is able to recover Wi-Fi credentials from their provisioning procedures. We demonstrate more details in the following subsection.

Table 3: Encoding modes and protection mechanisms for SmartCfg solutions

Solution	Encoding Mode	Protection
SmartConnection	DMA	AES (hard-coded key)
BroadLinkConfig	DPL	None
DirectConfig	DPL	None
SmartConfig	DPL	AES
EasyLink	Hybrid	RC4 (key reuse)
Esptouch	Hybrid	None
SmartLink	Hybrid	None
SimpleConfig	Hybrid	AES

4.3 Case Studies

4.3.1 Esptouch. ESP8266 is a low-power, highly integrated Wi-Fi chip with a Tensilica L106 32-bit microcontroller unit (MCU). It adopts the Esptouch SmartCfg solution to help users connect ESP8266EX-embedded devices to a Wi-Fi network. All of the encoding algorithms used in this solution are implemented in the package `com.espressif.iot.esptouch.protocol` [11].

We describe the original (i.e., prior to encoding) payload format of Esptouch in Table 4. The `apSsidCrc` and the `apBssidCrc` are the checksums of the `SSID` and `BSSID` respectively. The `totalXor` is the *XOR* sum of all the other bytes. The `ipAddress` is the IP address of smart home app, to which the device will return the message. The cumulative error correction algorithm is applied to the concatenated data fields such that all bytes and their respective checksums form the payload.

Esptouch encodes the credentials through a hybrid mode, in which it uses the *Destination Address* as the index, and then sends them through the length fields of a sequence of 802.11 packets. As a result, any attacker can sniff the data packet in the air and obtain the transmitted data to recover the credentials.

4.3.2 EasyLink. EasyLink is an SDK provided by MXCHIP for manufacturers and developers. It is designed to facilitate the connection of the devices to the Wi-Fi network. EasyLink uses pre-share key stored in devices and apps to encrypt the data during Wi-Fi Provisioning.

We download the project `EasylinkAndroid_Demo` from MXCHIP's Github repository [18]. By decompiling the provided Java package `easylinkv3-0.2.1-sources.jar` package we can know all the details about EasyLink. Unfortunately, we find that even though EasyLink adopts cryptographic algorithms to protect credentials, the credential can still be decrypted due to the cryptographic misuse. The code of EasyLink (version 3) is showed in Figure 4, we can see that after receiving `SSID` and password from user input, the RC4 cipher is used (in line 3 of the program) to encrypt SSID and key before the data is sent. However, in this case there is a typical crypto misuse–key reuse for stream ciphers. In particular, EasyLink reuses the same RC4 key to encrypt `SSID` and password. As a result, the attacker could simply conduct the *XOR* operation on the ciphertexts of `SSID` and password to obtain the differential of the plaintext. Since the plaintext of `SSID` is publicly known, password could be recovered easily with the *XOR* of differential and `SSID` plaintext when the `SSID` is longer than the password.

4.3.3 HiSmartWifiSet. Developed by Bayit Home Automation [5], the Bayit Cam smart home app allows users to keep an eye on their houses or offices remotely. We identify the native library (`libHiSmartWifiSet.so`) from the app package developed by Hichip, which is a variant of the MediaTek SDK since they have multiple function names in common (`InitCtrlPassword`, `SetCtrlPassword`, *etc.*). Although we cannot confirm whether this app adopts the same data encoding scheme from the MediaTek SDK, we find that this app uses a DMA encoding mode to send the `SSID` and password. Then we disassemble the binary code and find that it uses the AES cipher to encrypt the encoded data.

Interestingly, we notice that the sample SDK originally provided by MediaTek uses a hard-coded AES-128 key (`MfcwAnwCass2_78p`) for the encryption. We deduce that the sample may mislead third party developers and repeat the similar mistake. The analysis validates our hypothesis: the Bayit Cam app also uses a hard-coded key (`012345678abcdef`) as shown in Figure 5 (which is the pseudocode of the `StartSmartConnection` function decompiled by the Hex-rays decompiler).

4.3.4 Alink. Alink is a widely used smart home app in China. Developed by Alibaba Cloud, the leading cloud provider in China. This app integrates several popular solutions (e.g., SmartLink of Hi-Flying, BroadLinkConfig of BroadLink) to support as many smart devices as possible. In this case study, we demonstrate that even without knowledge of the used encoding algorithm, we can still recover the credential with a differential analysis of network traffic.

Table 5 gives a concrete example of this network traffic analysis. To employ the analysis, we first trigger the Wi-Fi provisioning process using three different credentials. We determine the transmitting mode by observing the changes of encoded data. Since the *Destination Addresses* of the filtered packets are always fixed and only the length field changes, the data encoding mode is DPL. Then

Table 4: Encoding protocol used by the Esptouch solution

Data field	totalLen	apPwdLen	apSsidCrc	apBssidCrc	totalXor	ipAddress	apPwd	apSsid
Data Length (byte)	1	1	1	1	1	4	< 32	< 32

Table 5: An example of the encoded data of the Alink Solution

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Length	0x4e8	0x4e8	0x4e8	0x11d	0x189	0x20d	0x292	0x354	0x3d7	0x44b	0x4cb	0x3e9	0x3e9	0x15b
Data	Preamble			15	Fixed	5	10	l	o	c	c	Preamble		s
Index	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Length	0x199	0x21a	0x29b	0x31c	0x39d	0x41e	0x49f	0x3ea	0x3ea	0x120	0x1a1	0x218	0x28c	0x360
Data	'1'	'2'	'3'	'4'	'5'	'6'	'7'	Preamble		'8'	'9'	'0'	CRC	

we try to locate the preamble. As shown in Table 5, the groups of packet 0-2, 11-12, 21-22 are packets in succession of the same length and this feature holds for all the three credentials. This is a very useful feature to determine the preamble. After we locate the preamble, which indicates the start of the data payload, we then determine the corresponding credential field and its semantics using differential traffic analysis. Through changing the length of SSID and password, we observe that the length of packet packet 3 is the sum of a fixed value (0x10e) and the total length of SSID and password. In addition, the length of packet 5 is the sum of a fixed value (0x208) and the length of SSID, and the length of packet 6 is the sum of a fixed value (0x282) and the length of the password. The actual data is encoded and placed after the packets of encoded length. We refer to some aforementioned encoding schemes and confirm that the encoding scheme should work as follows. Packets with indexes 0 and numbers ending with 1 and 2 are reserved for the preamble and the rest (those numbered 3-10, 13-20 and so on) are used to encode the SSID and the password. SSID concatenated with the password are parsed as an array of bytes, and they are encoded by their ASCII values plus the respective offset values from some arithmetic sequence, starting with 0xe8 and incrementing by 0x80 at a time.

4.4 Responsible Disclosure

We have reported discovered flaws to relevant security response departments prior to submitting our work. According to the responses, several vulnerabilities have been fixed before this submission. The purpose of this paper is not to demonstrate concrete attacks against certain SmartCfg solutions, but to demonstrate how improper design of the SmartCfg solution affects the security and privacy of smart home devices. We expect our work could help security analysts and SmartCfg solution designers conduct more effective assessment of SmartCfg solutions and thereby provide better privacy protection.

5 DISCUSSIONS

In this section, we discuss how to implement a secure SmartCfg solution to defend typical attacks against credential eavesdropping.

Figure 4: Code In EasyLink

```
if (ComHelper.checkPara(rc4key)){
   // encrypt
   this.ssid = SinRC4.encry_RC4_byte(Ssid, rc4key);
   this.key = SinRC4.encry_RC4_byte(Key, rc4key);
   if(null != Userinfo){
      issendip = true;
      this.user_info = SinRC4.encry_RC4_byte(Userinfo,
           rc4key);
   }
}
```

Figure 5: Pseudo-code in libHiSmartWifiSet.so

```
int StartSmartConnection(const char *_ssid, const char *_pwd
     , char _encypt){
  ...
  strcpy((char *)&gSsid, _ssid);
  strcpy((char *)&gPwd, _pwd);
  gEncypt = _encypt;
 ...
  memset(&psk, 0, 0x20u);
  strcpy((char *)&psk, _pwd);
  v9 = aes_encrypt_init((int)"012345678abcdef", 16);
  v10 = aes_decrypt_init("012345678abcdef", 16);
  aes_encrypt(v9, &psk, &e_psk);
  aes_decrypt(v10, &e_psk, &d_psk);
 ...
}
```

To ensure the confidentiality of the credential, device manufacturers, cloud platforms, and developers should cooperate to guarantee the security goal. Hence, we design a SmartCfg solution with three participants to enhance the provisioning security. In general, our solution requires the device manufacturers to distribute a unique

identity information (e.g., `Device ID`) for each device and this information can be obtained by the smart home app through a physical contact (e.g., with a QR code scanning). This identity information is distributed in the firmware of the smart device when the device is shipped. Then, a symmetric key (e.g., `Device Secret`) is derived from this information and thus both the device and the smart home app share the key. During the provisioning, the key is used to protect the credentials. In addition, a cloud server is introduced to help the device validate the smart home app. We consider that the identity information may be leaked and thus an attacker utilizing such information could also connect to the device. However, when the cloud server is introduced, the attacker could not forge an authentication token and thus the device could refuse such connection.

In detail the provisioning procedure of our proposed SmartCfg solution prototype is depicted in Figure 6. It utilizes the following steps to guarantee the security of credential broadcast:

(1) The smart home app obtains a `Product Key` (to identify the products) and a `Device ID` (to identify the devices) of the smart device through a physical contact (e.g., by scanning the QR code attached to the device). Using the combination of the `Product Key` and the `Device ID` proves that an attacker could not enumerate information of other devices since the `Device ID` is randomly distributed.
(2) The app authenticates itself to a web service deployed by the manufacturer on the cloud, then sends the obtained device information (i.e., `Product Key` and `Device ID`) to the cloud server. After that, the cloud server sends back a device `UUID` to the app. This `UUID` is used in the following steps to help device authenticate the smart home app.
(3) The smart home app generates the encryption key (e.g., `Device Secret`) with `HMAC_SHA256` (using the `Product Key` and the `Device ID` as the parameters). Although this crypto key can be derived by simply scanning the QR code on the surface of the device, if the attacker could not get close to the device (e.g., the device is placed in the user's house), he cannot eavesdrop the encrypted information. In addition, the user of the smart device could mask this QR code to protect the secret physically.
(4) The smart home app encrypts the credentials (password and the `SSID`) and the `UUID`, and uses an encoding scheme (e.g., in DMA or DPL mode) to encode the ciphertext. Then the encoded information is sent with several 802.11 packets.
(5) The device in SmartCfg state captures relevant 802.11 packets in the air and decode them. Then it tries to decrypt the information using its shared encryption key(e.g., `Device Secret`) stored in the secure storage. If the decrypted information is a legal message according to the protocol specification, the device obtains the Wi-Fi credential and connects to corresponding wireless network. Finally, the device verifies the received `UUID` by querying the cloud server. If the `UUID` is illegal, the device will refuse to be remotely controlled by the smart home app.

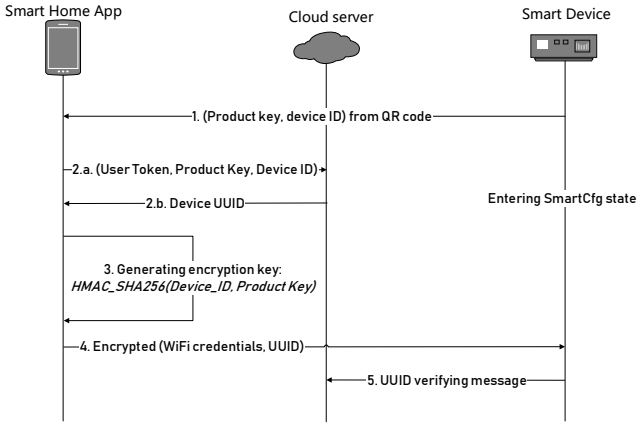

Figure 6: A prototype of security enhanced SmartCfg solution

6 RELATED WORK

6.1 Provisioning Solutions

With the rapid development of the Internet of things, more and more home begin to introduce intelligent systems and equipment. After the smart device connect to the Internet, the user can remotely control the device through the mobile app, which is a typical smart home network model. An issue to be addressed is how to configure the network of the smart device without a UI. In this section, we review some existing Wi-Fi provisioning technologies in the real product from the market along with their pros and cons:

(1) **AP mode** is another common provisioning technology. In AP mode, the credential of AP is defined by the manufacturer and an embedded web server is included. Then users browse into the device's web site and apply the Wi-Fi network configuration via pre-defined local URL or IP address. The device stores the network credentials and switches AP mode to Station mode, which indicates that it begins to connect to home network. However, this brings some problems: during the configuration process, the mobile Wi-Fi network needs to be switched, some of the data may be out of date and trigger error messages. Moreover, recently released smartphones check whether the Wi-Fi network is actually connected to the Internet. If not, it will disconnect from the given network and enforce cellular data connection. All the things above could complicate the user experience, which contradicts the purpose of good usability.

(2) **Wi-Fi Protected Setup (WPS)** was introduced by the Wi-Fi Alliance in 2006 as an easy and secure method to provision devices without knowing the network name and without typing long passwords. In a standard setup, users cannot connect a wireless device to a wireless network unless they know the network name and its password (also known as WPA-PSK key). Assume that the user wants to connect the device, e.g., a smart plug, to the user's wireless network, he must first pick the network to connect to and then enter the correct password on the device. WPS is enabled only when being supported by both the

device and the router, and this technology is commonly used in router provisioning.

(3) **Other side-channel methods** Many provisioning schemes leverage side-channel information to send credentials. Sending the credentials through BLE (Bluetooth Low Energy) requires devices and cell phones support Bluetooth, but this increases the production cost and could bring new problems. The voice-activated method enables mobile phone apps to transmit Wi-Fi credentials through a specific frequency of sound. The QR code method allows the mobile phone apps to generates a QR code that encodes the Wi-Fi credentials, and the device scans the QR code to get access to this device.

6.2 Security Analysis Techniques

We have summarized a number of works in analyzing IoT system security. In this section, we classify these works into five categories.

6.2.1 Information Collection and Data Mining. First, we need to collect sufficient relevant information, including the cloud (e.g., IoT solutions, communication protocols, security measures), devices (chip features, instruction architecture, memory layout, Flash model), and Apps. Zhang et al. [31] built a database that searches through a large amount of online data using semantic analysis to identify over 3000 IoT-related articles. Then, using machine learning methods to cluster the collected data, they are able to identify the the potential IoT security risks and problems that need further attention.

6.2.2 Device and App SDKs Code Audit. According to our survey, the IoT cloud platform will usually provide SDKs (e.g., Devices and Apps SDK) for third-party developers.In the work of [16], the authors illustrated how the security and privacy of smart home devices are affected by the device SDK. They demonstrated a concrete analysis mainly using source code audit to show that the design of smart home solutions they considered was flawed. The key idea of IoTFUZZER [8] is based upon the observation that most IoT devices are controlled through their official mobile apps, and those apps often contain detailed information about the protocol it uses to communicate with its device. SMARTGEN [32] focuses on the constraints by using selective symbolic execution and solves them to trigger the networking APIs. It features a new runtime app instrumentation technique that is able to more efficiently instrument an app and perform an in-context analysis.

6.2.3 Device Firmware Collection. (1) Download from the manufacturer's website or forum. Many IoT device manufacturers provide the latest firmware for download at the official website so users can manually update the device. Chen et al. [7] developed a web crawler using the Scrapy framework, with spiders respectively for each of the 42 vendors whose products include IP cameras, routers, access points, NAS, smart TV, cable modems, satellite modems, and even third-party or open-source firmware. (2) Extract the automatically updated package from traffics. Some IoT devices provide a one-click update or auto-update feature, and we can capture traffics by `Wireshark`. Giese and Wegemer analyzed the `Xiaomi` IoT ecosystem [13] and mounted a Man-in-the-middle attack between the cloud and device during the processing of firmware update. (3) Hardware Extraction and Dumping from Devices. Dumping binary data in memory via serial communication (UART or JTAG) or reading directly from FLASH. Fereidooni [12] connected ST-LINK with the smartphone's chip JTAG port to dump the firmware.

6.2.4 Reverse Engineering for cryptographic misuse. After getting the firmware, we are more concerned about whether the firmware contains the backdoor, as well as the hardcoded password. Firmalice [25] is a binary analysis framework to support the analysis of firmware running on embedded devices. They built a state-of-the-art symbolic execution engine to reveal the presence of backdoors in a number of embedded devices available on the market. Shao et al. [26] investigated the cryptographic misuse problem and finally they concluded the typical cryptographic misuse include the misuse of cryptographic algorithms, mismanagement of crypto keys and inappropriate use of encryption modes.

6.2.5 Traffic Analysis for Protocols. To intercept the communication between the device and the remote server, we usually deploy an MITM proxy acting as a wireless Internet gateway. After capturing the package, we intend to analyze the semantics of important fields in private protocols. AUTHSCOPE's [34] key idea is to use differential traffic analysis to recognize the protocol fields and then automatically substitute the fields and observe the server response. In proprietary protocol analysis, we can also use this method to determine the semantic information of each field. AUTOFORGE [33] can forge a valid cryptographically consistent message which can be consumed by the server. It contains a set of techniques to automatically infer protocol fields, label response messages, replay cryptographic function execution, and regenerate request messages.

7 CONCLUSION

This paper studies the security of SmartCfg, a popular technique for Wi-Fi provisioning. By auditing the SDKs provided by Wi-Fi chip vendors and analyzing the smart home apps, we conduct a systematic security analysis against real world smart home devices equipped with SmartCfg solutions and present an overview of the ecosystem through analyzing 821 smart home apps. We find that although SmartCfg effectively facilitates the Wi-Fi configuration, existing SmartCfg solutions seldom take the protection of credentials into serious consideration and therefore introduce security threats. We hope that our study can raise the awareness of the security issues and guide developers of both chips and devices to build more secure SmartCfg provisioning solutions.

8 ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This paper is partially supported by the Key Program of National Natural Science Foundation of China (Grant No.U1636217), the National Key Research and Development Program of China (Grant No.2016YFB0801200), and a research grant from the Ant Financial Services Group.

REFERENCES

[1] Broadlink official website. http://www.broadlink.com.cn/pageihc.html. Accessed February 28, 2018.

[2] Myapp app market. http://android.myapp.com/myapp/searchAjax.htm?kw=smartconfig&pns=3. Accessed March 5, 2018.

[3] Smarthome - home automation systems, products, kits, hubs. https://www.smarthome.com/. Accessed March 5, 2018.

[4] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 356–367. ACM, 2016.

[5] bayitHome. bayithomeautomation. www.bayithomeautomation.com. Accessed February 26, 2018.

[6] BroadLink. Library of libbroadlinkconfig. https://github.com/ruifeng2357/Breeze/blob/master/libs/armeabi/libBroadLinkConfig.so. Accessed February 26, 2018.

[7] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS*, 2016.

[8] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing.

[9] Espressif. Espressif document. https://www.espressif.com/en/support/download/documents. Accessed February 27, 2018.

[10] EspressifApp. Esptouchforandroid. https://github.com/EspressifApp/EsptouchForAndroid/. Accessed February 26, 2018.

[11] EspressifApp. Esptouch_protocol. https://github.com/EspressifApp/EsptouchForAndroid/blob/master/src/com/espressif/iot/esptouch/protocol/DatumCode.java. Accessed February 26, 2018.

[12] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. Breaking fitness records without moving: Reverse engineering and spoofing fitbit. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 48–69. Springer, 2017.

[13] Dennis Giese and Daniel Wegemer. Reversing iot xiaomi ecosystem. https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Reversing-IoT-Xiaomi-ecosystem.pdf. Accessed March 5, 2018.

[14] High-Flying. Hf low power wifi module user manual. http://www.hi-flying.com/download-center-1/user-guide-1/download-item-hf-lpx30-user-manual. Accessed November 23, 2017.

[15] Texas Instruments. Simplelink wi-fi smartconfig technology. http://www.ti.com/tool/SMARTCONFIG?keyMatch=smartconfig&tisearch=Search-. Accessed February 28, 2018.

[16] Hui Liu, Changyu Li, Xuancheng Jin, Juanru Li, Yuanyuan Zhang, and Dawu Gu. Smart solution, poor protection: An empirical study of security and privacy issues in developing and deploying smart home devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 13–18. ACM, 2017.

[17] MediaTek. Mediatek linkit connect 7681 developer's guide. https://docs.labs.mediatek.com/resource/linkit-connect-7681/en/documents. Published Mar 12, 2016.

[18] MXCHIP. Easylinkandroid_demo. https://github.com/MXCHIP/EasylinkAndroid_Demo/. Accessed February 26, 2018.

[19] NufrontIOT. Nl6621 sdk user manual. https://github.com/NufrontIOT/NL6621_StandardSDK/blob/master/Document. Accessed February 26, 2018.

[20] NufrontIOT. Nl6621_standardsdk. https://github.com/NufrontIOT/NL6621_StandardSDK. Accessed February 26, 2018.

[21] Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. Plaintext recovery attacks against wpa/tkip. In *International Workshop on Fast Software Encryption*, pages 325–349. Springer, 2014.

[22] Realtek. Android_simpleconfigwizard. https://www.amebaiot.com/cn/standard-sdk-simple-config/. Accessed February 26, 2018.

[23] Realtek. Realtek ameba user manual. https://www.amebaiot.com/cn/ameba-sdk-download/. Accessed February 26, 2018.

[24] Gil Reiter. A primer to wi-fi® provisioning for iot applications. *Texas Instruments White Paper*, 2014.

[25] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. In *NDSS*, 2015.

[26] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. Modelling analysis and auto-detection of cryptographic misuse in android applications. In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, pages 75–80. IEEE, 2014.

[27] TI. Smartconfigcc3x. http://www.ti.com/wireless-connectivity/simplelink-solutions/wi-fi/tools-software.html. Published February 26, 2018.

[28] Mathy Vanhoef and Frank Piessens. All your biases belong to us: Breaking rc4 in wpa-tkip and tls. In *USENIX Security Symposium*, pages 97–112, 2015.

[29] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1313–1328. ACM, 2017.

[30] wikipedia. Wi-fi protected access. https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access. Accessed February 28, 2018.

[31] Nan Zhang, Soteris Demetriou, Xianghang Mi, Wenrui Diao, Kan Yuan, Peiyuan Zong, Feng Qian, XiaoFeng Wang, Kai Chen, Yuan Tian, et al. Understanding iot security through the data crystal ball: Where we are now and where we are going to be. *arXiv preprint arXiv:1703.09809*, 2017.

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

[33] Chaoshun Zuo, Wubing Wang, Zhiqiang Lin, and Rui Wang. Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services. In *NDSS*, 2016.

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