Smart Solution, Poor Protection: An Empirical Study of Security and Privacy Issues in Developing and Deploying Smart Home Devices

Hui Liu
Shanghai Jiao Tong University
Minhang Qu, Shanghai Shi, China

Changyu Li
Shanghai Jiao Tong University
Minhang Qu, Shanghai Shi, China

Xuancheng Jin
Xidian University
Xi'an, Shaanxi, China

Juanru Li
Shanghai Jiao Tong University
Minhang Qu, Shanghai Shi, China

Yuanyuan Zhang*
Shanghai Jiao Tong University
Minhang Qu, Shanghai Shi, China

Dawu Gu
Shanghai Jiao Tong University
Minhang Qu, Shanghai Shi, China

ABSTRACT

The concept of Smart Home drives the upgrade of home devices from traditional mode to an Internet-connected version. Instead of developing the smart devices from scratch, manufacturers often utilize existing smart home solutions released by large IT companies (e.g., Amazon, Google) to help build the smart home network. A smart home solution provides components such as software development kit (SDK) and relevant management system to boost the development and deployment of smart home devices. Nonetheless, the participating of third-party SDKs and management systems complicates the workflow of such devices. If not meticulously assessed, the complex workflow often leads to the violation of privacy and security to both the consumer and the manufacturer. In this paper, we illustrate how the security and privacy of smart home devices are affected by *JoyLink*, a widely used smart home solution. We demonstrate a concrete analysis combined with network traffic interception, source code audit, and binary code reverse engineering to evince that the design of smart home solution is error-prone. We argue that if the security and privacy issues are not considered, devices using the solution are inevitably vulnerable and thus the privacy and security of smart home are seriously threatened.

CCS CONCEPTS

• **Security and privacy** → **Network security**; **Web protocol security**;

KEYWORDS

IoT; smart home solution; security; privacy

*Corresponding author: yyjess@sjtu.edu.cn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IoTS&P'17, November 3, 2017, Dallas, TX, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5396-0/17/11...$15.00
https://doi.org/10.1145/3139937.3139948

1 INTRODUCTION

The rapid growth of Internet of Things (IoT) promotes the popularity of smart home, a home system in which all the home appliances, sensors and services can be connected through the communication network, and can be remotely monitored and controlled. Despite the prosperity, the smart home space is heavily fragmented. Devices are made by different manufacturers and industry standard is lacked, thus they are often incompatible with each other due to the communication diversity. To accelerate the development of smart home devices and integrate diverse devices into a unified network, **smart home solutions** are proposed by large IT companies such as Apple, Amazon, and Google. A smart home solution often provides two components: one **software development kit (SDK)** with common functions and protocols implemented to support multiple IoT architectures (e.g., ARM, AVR, PIC), and a **management system (a gateway or a web-based platform)** to connect different devices in a smart home system. Manufacturers could simply make use of the SDK and configure the device to communicate with the management system, then consumers could access and manage the device through the management system.

Although smart home solution facilitates the developing and deploying of smart devices, it also introduces potential security and privacy risks from two aspects: First, the use of SDK enlarges the code base of smart devices. Since the SDK is not designed specifically for one but for a variety of devices. It often contains multiple functions that are not always necessary and even vulnerable. More seriously, developers do not always understand the usage of the SDK correctly. Recent years have witnessed the misuse of third-party SDKs of many platforms [10, 11] and the IoT platform is no exception. Second, smart home solution providers often introduce a cloud platform for data collecting and transmitting. Although this helps achieve a centralized management of devices, it adds an extra remote server between the device and its user. The uploaded data is sometimes not well-protected and thus raises privacy concerns.

Even though specific concerns about the smart home have been expressed, the problem is often not blamed on the smart devices themselves. It is usually the ill-designed smart home solutions that lead to the violation of security and privacy. To alleviate those concerns, smart home solution should be assessed thoroughly. The assessment is, however, hindered by several issues. First, manufacturers sometimes modify the SDK to conform to their devices and thus the workflow for each device may be slightly different. Second,

some logics of the workflow are implemented on the web server and is unknown to analysts. Third, solutions are not always well-documented and many details of the workflow are only regulated by the source code of the SDKs. In a word, the assessment of smart home solutions requires an in-depth security analysis.

To reveal how smart home solution could affect a wide range of devices from different manufacturers. This paper conducts an empirical study on a representative smart home solution–*JoyLink* [1] and its relevant smart devices. *JoyLink* is a widely used smart home solution in China and supports more than 6100 smart devices including air conditioner, washing machine, camera, etc. Although the high-level process of the deployment and management of *JoyLink* smart home solution are described by its published documents, many details related to security and privacy are opaque and uncertain. To thoroughly comprehend its workflow and find potential flaws, we propose an analysis combined with network traffic interception, source code audit, and binary code reverse engineering. Particularly, we focus on the deployment procedure, which is the most vulnerable link of the entire lifetime of the smart device. Our analysis reveals that the workflow adopted by *JoyLink* is inherently vulnerable due to the following design issues: 1) incorrect crypto key management; 2) unnecessarily uploading of sensitive data (lacking end-to-end encryption); 3) insecure authentication mechanism. As a result, smart devices with JoyLink protocol can be hijacked or impersonated, and the sensitive private data such as WiFi password is leaked.

Responsible Disclosure. We have reported discovered flaws to relevant security response centers (SRC) prior to submitting our work. According to the response, several vulnerabilities including WiFi credential uploading have been fixed before the submitting of our findings. The purpose of this paper is not to demonstrate some concrete attacks against certain smart home solution, but to demonstrate how improper design of the smart home solution and its relevant network infrastructure affect the security and privacy of smart home devices. We expect our work could help security analysts and smart home solution designers conduct more effective assessment of smart home solutions and thereby providing better the device protection.

2 ANALYSIS OF JOYLINK SOLUTION

The *JoyLink* solution is a smart home solution proposed by *JD.com* (a.k.a 360buy), the largest E-commerce company in China (rank by revenue). The solution provides an SDK that supports a wide range of IoT devices, an app to manage all JoyLink-compatible devices, and a smart cloud platform as the management system to receive and store device data. Unlike other smart home solutions, *JoyLink* only supports devices shipped with a WiFi or BLE module. Therefore, it does not require a hub or a gateway as the management system. A typical architecture and relevant workflow of a *JoyLink* smart home system are depicted in Figure 1. There are four entities involved: the **smart home device** (192.168.2.5) to be included; the **JoyLink app** (192.168.2.2) running on a smart phone (Android or iOS) and playing the role of managing and controlling; the **WiFi hotspot** (192.168.2.1) that the phone has connected to and that the device is to be configured to connect to; and the **cloud** (113.xx.xx.xx) that both the phone and the device communicate with to complete registration, device binding, remote controlling, etc. To join the *JoyLink* smart home system, all smart devices must first integrate with *JoyLink* SDK and conform to the same protocol. The protocol mainly consists of three steps: 1) bringing the device online, 2) binding the device with the user account, and 3) setting the device to be controlled locally and remotely. To discover how these steps are implemented and whether the security and privacy are well protected, we build an experimental environment with several JoyLink-compatible devices, a rooted Android smart phone, and a network traffic analyzer based on a Raspberry-Pi to monitor the workflow. The analysis results are detailed as follows:

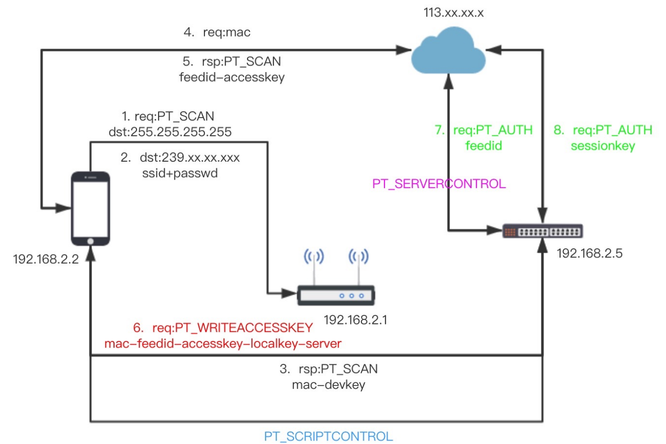

Figure 1: The architecture of the JoyLink solution and the basic workflow of a specific device.

magic	optlen	payloadlen	version	type	total	index	enctype	resevered	crc

Figure 2: Message Header.

We first manually review the source code of *JoyLink* SDK and relevant documents to understand the communication process. Throughout the entire communication process, the communication between the app and the cloud always adopts HTTPS, with the client correctly verifying certificates. We name the communication between the app and the device as *local communication*, and the device-cloud communication as *remote communication*. Both of them are home-made protocols. The protocol message header format, shown in Figure 2, is followed by the *optdata* field and *payload* field. The *magic* field is a protocol indicator, which contains two possible values respectively representing the local and remote communication. The *type* field indicates the packet type, whose value can be the ones listed in Figure 3. The *enctype* field indicates the encryption type, whose possible values including `ET_NOTHING` (no encryption), `ET_ACCESSKEYAES` (use accesskey as the AES key), `ET_ECDH`, `ET_SESSIONKEYAES`, etc. The *crc* field is a two-byte checksum, presumably used to detect data corruption. The *opt* data field is present only when the packet type is `PT_SCAN`, and the content is the EC public key of the sender. Both the header and the *optdata* field are in plain text. The payload can be plaintext or encrypted depending on the *enctype* field.

Packet Type	Function	Key Parameters
PT_SCAN	Device discovery	ECDH pubkey, MAC address
PT_AUTH	Establishing remote communication	accesskey, sessionkey
PT_HEARTBEAT	Maintain connection with the cloud	firmware version
PT_SCRIPTCONTROL	Device local control	-
PT_SERVERCONTROL	Device remote control	feedid
PT_WRITEACCESSKEY	Device initialization	feedid, localkey, accesskey

Figure 3: Mainly used packet type.

Payload is encrypted with AES/CBC by default, and four keys are involved: 1) the key negotiated between the app and the device during `PT_SCAN`, which we name as *tmpkey*; 2) the *localkey* generated by the app and used to encrypt rest of the local communication; 3) the *accesskey* which is generated by the cloud, passed to the device by the app and used by the device to authenticate itself to the cloud; 4) the *sessionkey* which is generated by the cloud and used in remote communication.

After the audit of source code of *Joylink* SDK, we further verify whether the smart devices adopted the SDK follow the suggested workflow. In our analysis we obtain the firmware of a smart plugin through capturing the traffic of its upgrading procedure. We find the used microcontroller is a QCA4010 [3] with Xtensa [2] architecture. Thus we also download the source code of *Joylink* SDK and compile the corresponding library to conduct a code similarity comparison that matches the function in the firmware. Now we detailed those important steps in the workflow especially the device deployment procedure.

2.1 Preparation

The app obtains a *product_uuid* by scanning the QR code on the device or choosing the right product name listed in the app and sends it to the cloud. Afterwards, the app broadcasts the `PT_SCAN` message indicating the *product_uuid* and the EC public key of the app. Meantime, the app asks user to input WiFi credential and starts a WiFi Provisioning process.

2.2 Device WiFi Provisioning

Several schemes are commonly used when completing WiFi configuration of a headless IoT device using a mobile application (e.g., Access Point Mode, WiFi direct, TI's SmartConfig). *JoyLink* adopts the method that the app encodes the SSID and password into a sequence of IP addresses and sends each IP a null character. At the same time, the app continues broadcasting the `PT_SCAN` message. This process repeats until the device successfully observes the traffic pattern, extracts the WiFi credential, gains access to the network, and finally sends a `PT_SCAN` response to the app. The response includes the device MAC address and devkey, which is the EC public key of the device.

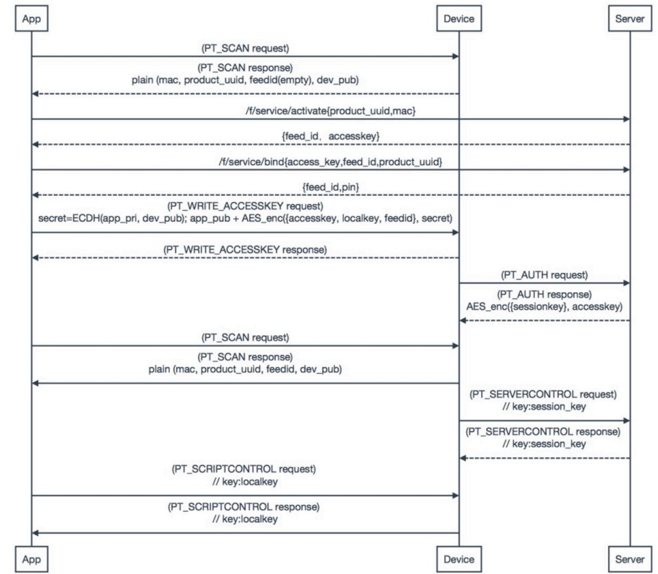

Figure 4: Summary of device setup

2.3 Device Initialization

Once the app gets the MAC address, it sends a request to the cloud containing the MAC and other information like *product_uuid* and user account. The cloud confirms the binding relationship and responds with a message containing *feedid* and *accesskey*. The app generates the *localkey* out of *accesskey* and sends it together with *feedid* and *accesskey* to the device in a `PT_WRITEACCESSKEY` message, which is encrypted with the key negotiated out of ECDH (tmpkey) during `PT_SCAN`. The device saves the {feedid, accesskey, localkey} and sends the `PT_AUTH` request to the cloud. The `PT_AUTH` request includes feedid and is sent to authenticate itself and obtain the key used in the normal remote communication. Once receive the request, the cloud generates a *sessionkey*, encrypts it with the *accesskey*, and sends the result in the `PT_AUTH` response message.

2.4 Remote Control

The device decrypts the `PT_AUTH` response, gets the *sessionkey*, and launches a long connection to the cloud. From this point, the message payload is encrypted with the *sessionkey*. Messages sent between the device and the cloud can be categorized by function into heartbeat, remote control and status report message. The device sends a `PT_HEARTBEAT` packet every 15 seconds. And it reacts immediately once the cloud sends a control message `PT_SERVERCONTROL` and responds with the status report packet.

In all, the whole device setup process is summarized in Figure 4. What we describe above illustrates the normal procedure as an ordinary user adding a new the device and controlling it from the app. In the following two sections, by introducing a locally resided attacker, we describe how the improper design of the *JoyLink* solution endangers security and privacy.

3 SECURITY ISSUES

3.1 WiFi Provisioning

As we presented in Section 2.2, the app encodes the WiFi credential into a sequence of IP addresses. However, this encoding is nothing more but simply putting one character a time to the last byte of the IP address, which makes the credential available to anyone in the vicinity. Besides, a local attacker can fake the traffic indicating credential of the WiFi she controls. The transmitted WiFi credential has no binding relationship with the user account. The device does not report the WiFi it connects. Therefore, the whole configuration and control process seem normal to everyone except that the innocent user will find the device abnormally go offline once the attacker shuts down the fake WiFi.

3.2 Communication Security

The whole crypto key management is vulnerable. The four keys mentioned above have an dependence relationship as: *sessionkey* → *accesskey* → *localkey* → *tmpkey*. Except for *tmpkey*, the rest keys are all generated by a single party. The security of the key on the left when delivered to the counter-party is dependent on the right key. Although *tmpkey* is generated through negotiation, the ECDH key negotiation process has no MITM protection. A local adversary can launch a MITM attack, and all these keys are exposed to the attacker.

Another serious issue comes in as the *enctype* is not constrained by any party of the communication. As such, some of the encrypted communication as we described above, can be downgraded to plaintext communication by an attacker who constructs an `ET_NOTHING` packet. Attacks can be carried out using several approaches and result in different severity of consequences. We demonstrate these attacks as follows:

Traffic Decryption. The attacker resides in the middle when the normal user tries to bind the device. Once detected the `PT_SCAN` message, she acts as the man in the middle, replacing EC public keys in the corresponding messages. After that, she can decrypt the payload of the `PT_WRITEACCESSKEY` message, obtain the *accesskey* and the *localkey*, send these information in `PT_WRITEACCESSKEY` message using the `ET_ECDH` encryption, which is feasible since the device is sharing key with the attacker. She can also simply send these information with `ET_NOTHING` encryption. Since the attacker knows the *accesskey*, she can decrypt the `PT_AUTH` response and obtain the *sessionkey*. Therefore, the normal user perceives nothing, and the attacker can decrypt all the traffic afterwards.

Device Hijacking. The device will react to the `PT_SCAN` message whenever it receives one, and respond immediately with the `PT_SCAN` response indicating the device MAC and other information. An attacker who has connected to the same WiFi as the device does, can send a `PT_SCAN` message and thus obtaining the device MAC. Then the attacker logins into his own the cloud account and sends an activate request containing the obtained MAC to the cloud with the *feedid* field set to null. The cloud will return *feedid* and *accesskey* no matter whether or not the MAC has been binded to an existed *feedid*. Now the attacker can generate a *localkey* and write {accesskey, localkey, server IP} to the device with the `PT_WRITEACCESSKEY` message. And the attacker can leave the device communicating with the cloud and establishing a long connection. In this way, the attacker successfully hijacks the device to her own account, and can control the device remotely.

Out-of-band Device Control. The attacker can control the device without the involvement of the cloud and the app. The attacker can achieve this either by simply sending a `PT_SCRIPTCONTROL` message of `ET_NOTHING` encryption type, instructing the state change of the device locally, or generating a new `PT_WRITEACCESSKEY` message with the server IP field filled with her own IP, which makes the device establish the long connection with the fake server and be controlled by the fake server remotely via the `PT_SERVERCONTROL` message.

Device Impersonation. The attacker can login her cloud account, make up a MAC address, and send the cloud an activate request containing the MAC. With the *accesskey* acquired, she will get all the data needed to forge the existence of the device. Therefore, the attacker can have her account binding as many devices as she want.

3.3 Firmware Modification

Since the SDK allows a firmware update procedure, many smart home devices we analyzed support both local and remote firmware update. After the device binding, the URL of a firmware will be sent from the cloud to the app if the device firmware has a new version. And the app will pass the URL to the device locally. The URL can also be delivered directly from the cloud to the device via the remote communication. After receiving the URL, the device makes an TCP connection to the URL to fetch the binary file of the firmware. However, the verification of the downloaded file is lacked or incorrect, which allows attackers to modify the firmware and inject malicious code. Due to the incorrect sample given by the SDK, we found many devices conduct no verification of the downloaded firmware other than checking the CRC32 value gotten together with the URL. This is a severe security vulnerability, since both of the local and remote communication can be taken control of by the attacker as we describe above. What's more, we encounter the situation that the app kept prompting fail message on the app when we try to update the firmware as the normal user does. Through traffic sniffing, we observe that the URL the device receives can not be successfully used to obtain the firmware. The reason is that the URL the device requests is added with an additional slash compared with what the device receives, which the server does not respond. This typo makes many devices outmoded and may suffer from attacks with published vulnerabilities.

4 PRIVACY ISSUES

Throughout the entire process of communication, all the control commands and device uploaded data are visible to the cloud. We argue that a better design should be based on end-to-end encryption, considering the large amount of user-related data involved. The cloud should only forward all the encrypted data between the device and the app without knowing the meaning of the transmitted data.

Another serious privacy violation we find in the protocol workflow is that during the WiFi provisioning process the app also sends WiFi credential to the cloud. This happens before the app talking to those list of IPs. Actually, this step is done only to get rules of the WiFi credential encoding, which is totally unnecessary and

serious infringement of user privacy. Particularly, we find that the app get an update after being informed of the inappropriateness of the WiFi upload behavior. However, The update conducts an encryption before sending the credential out instead of removing the WiFi credential upload process as we expected. After analysis, we find that the adopted encryption scheme is another typical cryptography misuse. The message in cipher text consists of two parts. The data field is the result of AES/CBC encryption of the credential, and the key field is the encrypted AES key. The AES key is encrypted with a hard-coded public key in the app using RSA/ECB/PKCS1Padding encryption. Everything seems fine until we find out that the value of the timestamp is used as the AES key. We can deduce the key by enumerating all the possible timestamps since the space is small enough. And the one that can decrypt the data field into a sequence of readable characters is the right key with extremely high probability. Nonetheless, even though this encryption scheme works correctly, we insist that the WiFi credential should never be upload to the cloud.

5 DISCUSSION

The attacks described in the previous section is made possible by a combination of design and implementation issues. 1) Critical keys used to secure the communication are determined by single party, which makes them easier to be forged. And the dependency relationship between these keys make the `tmpkey` the most important position. Although the `tmpkey` is negotiated through ECDH, no MITM protection still opens the door to the attacker. 2) The cloud does not check existing binding relationships before responding to an activate request. 3) The device always responds to any request that satisfies the message format. The protocol state machine is poorly maintained. 4) The communication protocol supports different type of encryption and enforces no constraint like when to use the encryption mode.

The SDK is used in every device that support *JoyLink*, making the security of the SDK itself crucial to the entire ecology. We give the following suggestions for mitigation:

- **Add MITM protection during ECDH key negotiation.** Presetting each device a key pair and maintaining the mapping relations on the cloud can prevent the MITM attacks. However, probably because the cost of assigning each device a unique key pair is too high, distributing the same key to the same type of products is more possible to happen when deploying [8]. On the other way around, presetting the public key of the app in devices can also achieve the goal, on the condition that the private key is properly protected and used.
- **Bind the encryption type with packet type.** The `ET_NOTHING` type should only be used in the `PT_SCAN` message.
- **Make keys dependent on both parties.** All the keys used should be generated by negotiating through a secure channel or with confidence of the identity of the communicator. Avoid the direct dependency relationship between keys, and introduce some randomness if the relationship must exist.
- **Maintain the protocol state machine correctly.** The device should only accept `PT_SCAN` and `PT_WRITEACCESSKEY` message when it is in the configuration mode, which is launched merely at the first use indicated by the non-existence of some parameters such as `feedid`, `localkey` and `accesskey`, or by manually resetting the device.

As for how to audit a smart home solution and locate the vulnerable links. The analysis approach we used in *JoyLink* can be applied to other smart home solutions, since their application senarios and the information available to analysts are highly similar. A critical aspect needs to pay attention to is the device bootstrapping procedure. Violation of security and privacy may arise as a lot of options existing. Each of the options meets different security requirements and satisfies different use-cases. [7] provides a structured classification of the available mechanisms and their security considerations. The implementation of these IoT device bootstrapping methods can be complex and error-prone, and the analysis is effort-consuming.

6 RELATED WORK

IoT security and privacy is an emerging area and in recent years numerous works on the security of IoT devices and protocols were published. Current smart home security analyses are centered around two themes: devices and protocols. On the device front, Ronen *et al.* [8] presents a comprehensive attack against Philips Hue smart lamps combined with side-channel analysis and firmware reverse engineering. Fereidooni *et al.* [4] give an in-depth security analysis of the operation of fitness trackers commercialized by Fitbit. However, these researches have largely focused on certain devices and often ignore the security and privacy vulnerabilities caused by smart home solutions. On the protocol front, researchers demonstrated flaws in both ZigBee [8] and BLE [6] protocol implementations for smart home devices. These works focus on the security aspect and discuss less about privacy issues.

Similar to our work, a security analysis of several smart home hubs is performed by Veracode [9]. The security analysis focuses on infrastructure protection such as SSL/TLS deployment, replay attack protection, and password strength. Our study is more fine-grained since we conduct SDK source code audit and firmware reverse engineering to reveal flaws in proprietary *JoyLink* protocol. Fernandes *et al.* [5] presents an in-depth empirical security analysis of Samsung SmartThings smart home platform. The focus of this work, however, is SmartThings apps. In comparison, our work concerns more about the SDKs and management systems introduced by the smart home solutions, and reveals intrinsic design flaws of popular *JoyLink* smart home solution.

7 CONCLUSION

In this paper, we discuss the security and privacy threats introduced by using smart home solutions. A comprehensive analysis combined with source code audit, binary code reverse engineering, and network traffic interception is demonstrated to reveal the detailed workflow of devices using *Joylink* smart home solution. The analysis reveals that due to the design flaws in the SDK and relevant network communication protocols, devices are inevitably vulnerable, and both the security and the privacy are violated. We suggest that developers and analysts scrutinize existing smart home solutions to avoid such issues.

ACKNOWLEDGMENTS

The authors 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 under Grant No.: U1636217, the National Key Research and Development Program of China under Grant No.: 2016YFB0801200 and Major Program of Shanghai Science and Technology Commission under Grant No.: 15511103002.

REFERENCES

[1] 2017. JoyLink 2.0. (2017). Retrieved September 16, 2017 from http://devsmart.jd.com/dev/apiDocDir

[2] 2017. Linux Xtensa. (2017). Retrieved September 16, 2017 from http://www.linux-xtensa.org/

[3] 2017. Qualcomm QCA4010 SoC. (2017). Retrieved September 16, 2017 from https://www.qualcomm.com/products/qca4010

[4] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. 2017. Breaking Fitness Records without Moving: Reverse Engineering and Spoofing Fitbit. *arXiv preprint arXiv:1706.09165* (2017).

[5] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (SP)*.

[6] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. 2016. Mind the Tracker You Wear: A Security Analysis of Wearable Health Trackers. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC)*.

[7] Network Working Group Internet-Draft. 2017. Secure IoT Bootstrapping: A Survey. (2017). Retrieved September 16, 2017 from https://tools.ietf.org/html/draft-sarikaya-t2trg-sbootstrapping-03

[8] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP)*.

[9] Veracode. 2015. The Internet of Things: Security Research Study. (2015). Retrieved September 16, 2017 from https://www.veracode.com/sites/default/files/Resources/Whitepapers/internet-of-things-whitepaper.pdf

[10] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. 2013. Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In *Proceedings of the 22nd USENIX Security Symposium*.

[11] Wenbo Yang, Yuanyuan Zhang, Juanru Li, Hui Liu, Qing Wang, Yueheng Zhang, and Dawu Gu. 2017. Show Me the Money! Finding Flawed Implementations of Third-party In-app Payment in Android Apps. (2017).