A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps

*

Davino Mauro Junior
Federal University of
Pernambuco, Brazil
dmtsj@cin.ufpe.br

Luis Melo
Federal University of
Pernambuco, Brazil
lhsm@cin.ufpe.br

Hao Lu
University of
Michigan, USA
harveylu@umich.edu

Marcelo d'Amorim
Federal University of
Pernambuco, Brazil
damorim@cin.ufpe.br

Atul Prakash
University of
Michigan, USA
aprakash@umich.edu

***Abstract*—Security of Internet of Things (IoT) devices is a well-known concern as these devices come in increasing use in homes and commercial environments. To better understand the extent to which companies take security of the IoT devices seriously and the methods they use to secure them, this paper presents findings from a security analysis of 96 top-selling WiFi IoT devices on Amazon. We found that we could carry out a significant portion of the analysis by first analyzing the code of Android companion apps responsible for controlling the devices. An interesting finding was that these devices used only 32 unique companion apps; we found instances of devices from same as well as different brands sharing the same app, significantly reducing our work. We analyzed the code of these companion apps to understand how they communicated with the devices and the security of that communication. We found security problems to be widespread: 50% of the apps corresponding to 38% of the devices did not use proper encryption techniques; some even used well-known weak ciphers such as Caesar cipher. We also purchased 5 devices and confirmed the vulnerabilities found with exploits. In some cases, we were able to bypass the pairing process and still control the device. Finally, we comment on technical and non-technical lessons learned from the study that have security implications.**

***Index Terms*—Security; Internet of Things; Android Apps; Companion Apps**

I. Introduction

The number of Internet of Things (IoT) devices [1] worldwide is predicted to reach 20 billion by 2020 [1]. Their security is a huge concern. As a concrete example, in October 2016, the Mirai malware compromised millions of IoT devices around the world and used them to launch the largest DDoS attack ever recorded [2]. In the scenario of a smart-home, security vulnerabilities in IoT devices could compromise safety [3] and availability of systems.

IoT devices are often compatible with multiple cloud-based software stacks (e.g., SmartApps in the SmartThings cloud [4], Alexa Skills [5], etc.). Prior work has found security vulnerabilities introduced by some of these stacks [6], [7]. Unfortunately, devices may have vulnerabilities out-of-the-box that are independent of security of high-level software stacks.

Given the attention that security of IoT devices has already received, one would assume that vendors of popular devices (and their customers) take security seriously. To assess how vendors incorporate security in their IoT products in the real-world, this paper presents an emperical study of security of 96 popular smart devices on Amazon. To make the analysis scalable, this paper uses an indirect way of assessing security of IoT devices by analyzing their companion apps, i.e., apps available for the Android platform that enable users to control the devices directly from their smartphones. Our hypothesis is that the analysis of these apps can throw substantial light on potential vulnerabilities in devices and even help security analysts develop proof-of-concept exploits to induce the device manufacturers to verify the vulnerabilities and fix them. To validate this hypothesis, we analyze multiple apps for vulnerabilities and, from that, created proof-of-concept attacks.

An alternative technique for analyzing the security of these devices would have been to purchase them, extract their firmware and analyze it to understand their behavior and, indeed, such techniques have been previously used [8], [9]. But, such techniques can be difficult to scale to a large number of devices and require significant analysis of low-level systems. We found that the indirect way of analyzing companion apps provides a complementary security analysis technique that can still offer interesting security insights even without analyzing a device's firmware.

This study makes the following contributions:

1) We analyzed smartphone apps for 96 top-selling WiFi devices on Amazon. We found that companion app analysis can be a useful step in uncovering potential vulnerabilities. Somewhat to our surprise, we found that not just one vendor but different vendors sometimes shared the same app to control the devices; for 96 devices, we found only 32 unique Android companion apps. For example, during our analysis we found that devices from *eWeLink*, *Tuya*, and *YI* all used the same app. We found security problems to be widespread among these popular devices and their companion apps: 50% of the apps corresponding to 38% of the devices did some critical communication without using proper encryption

This material is based upon work supported by RNP and the National Science Foundation under Grant Nos. 1740897 and 1740916.

[1] We may refer to IoT devices as smart devices (or just devices) and the apps that control these devices as companion apps (or just app).

DOI 10.1109/SPW.2019.00042

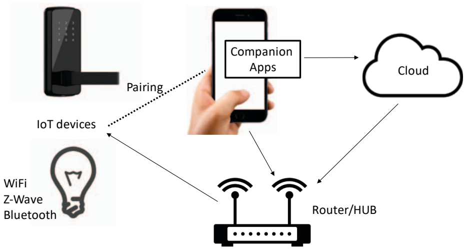


Figure 1: Example of IoT setup.

techniques; some even used well-known weak ciphers such as Caesar cipher.

2) We did a detailed analysis of companion apps responsible for controlling 5 of the devices that we purchased from Amazon. These confirmed the findings from the companion app analysis using dynamic tests and allowed us to test potential exploits. We confirmed, for example, that an Amazon top-seller smart plug from TP-Link [10] shares the same hard-coded encryption key for all the devices of a given product line and that the initial configuration of the device is established through the app without proper authentication. Using this information, we were able to create a spoofing attack to gain control of this device. A video illustrating a counterfeit app in action can be found here [11]. We found that the attack also worked a different TP-Link device that shared the same app.
3) We discuss the lessons learned. We expected to find some vulnerabilities, but the extent of prevalence of vulnerabilities in devices that are top-selling suggest larger challenges around inadequacy of customer awareness and transparency of vendor disclosures related to security risks. We also found some devices that are designed with better security and discuss two strategies used in them for securing communication, as they may offer some guidance towards developing design patterns for vendors to better secure their IoT devices and companion apps.

II. Context, Goal, and Questions

A. Context

Most manufacturers of IoT devices provide a smartphone application and cloud services to monitor and control their devices. Communication between the device and its companion app is often established over the local network using some wireless protocol such as Zigbee and Z-Wave [12]. Some WiFi-enabled devices we analyzed require password-protected WiFi networks as a security measure. Unfortunately, password-protected WiFi networks alone are not an adequate defense. For example, WiFi passwords are often shared and users may also inadvertently download malicious programs or scripts that can send packets on the same network. *It is thus important to analyze whether the communication between a companion app and the device is properly secured.*

One method for such communication is via the cloud [2]. An IoT app sends the message intended for a device to the cloud server and the cloud server relays the message to the device. Similarly, a message from a device can be relayed to the IoT app through the cloud. Figure 1 illustrates an example setup involving a router/hub and the cloud. Another method is to use local communication. In this context, pairing is the process of establishing a communication channel between an app and a device. A reasonable assumption that a customer should be able to make is, once paired, such a channel is secure. Part of our work tests this assumption.

B. Goal and Questions

To help analyze the security of app-device communication, we analyzed the companion apps with the following initial questions to help guide a deeper security analysis:

Q1) Are encryption key(s) hardcoded? A malicious developer could counterfeit messages if she has access to secret keys. In some case, those keys could be mined from code (even when it is obfuscated) and then used to control the device without pairing with it.

Q2) Does the app use local communication? When the IoT app and the corresponding device are in the same network, local communication may be used. Unfortunately, we found potential problems in that scenario. For example, local communication is often implemented with protocols such as UDP over broadcast, enabling an attacker to eavesdrop communication and possibly replaying packets using the same packet structure. In contrast, cloud communication usually involves HTTPS/SSL certificates. Consequently, an attacker would have to make a bigger effort to forge certificates of the parties involved.

Q3) Does the app send broadcast messages? Broadcast messages are frequently used in IoT setups to discover devices and to enable direct app-device communication when there is no hub/gateway in the setup. Their use, unfortunately, can put a smart home at risk. Adversaries can, for instance, sniff[3] the response of devices to broadcast messages, which often include sensitive data such as the internal state of the device.

Q4) Does the app use any well-known protocol with vulnerabilities? Different protocols tailored to IoT deployments exist and some of these protocols are known to be vulnerable. Table I shows the number of reported issues and the ID of an example CVE issue on four highly-vulnerable protocols. For instance, the UPnP vulnerability CVE-2016-6255 allows remote attackers to write arbitrary files to the device file system [14].

Table I: CVE vulnerabilities in major IoT protocols implementations.

Protocol	# Vuln.	Example
MQTT	13	CVE-2017-9868
SIP	59	CVE-2018-0332
UPnP	346	CVE-2016-6255
SSDP	17	CVE-2017-5042

[2] Tuya Smart [13] is an example framework where the companion app can only communicate with the device through the cloud.

[3] This ability to sniff WiFi messages depends on the distance to the router.

III. Finding and Confirming Vulnerabilities

This section details the process of discovering and confirming vulnerabilities in IoT apps.

A. App Selection Criterion

We started with the top-100 most popular smart hubless devices from the Amazon website and then restricted the resulting set to devices that use WiFi for communication. We found 96 such devices. We then restricted them to devices from the categories smart plugs, bulbs, or IR controllers that use Wi-Fi. The rationale is affordability and because Wi-Fi is popular and provides a potential attack surface if an attacker executes code anywhere on the same network (e.g., using an app or downloaded executable code as attack vector). A total of 54 devices satisfied this criterion. From these 54 devices, we randomly selected (and purchased) 5 devices to run our analysis. We found that two of the devices we selected use exactly the same app (both devices belonged to the same manufacturer, TP-Link).

We first present results from vulnerability analysis of these four companion apps. Analysis suggested vulnerabilities in each. We confirmed the vulnerabilities with proof-of-concept exploits on the 5 devices, thus validating that analysis of companion apps can offer significant insights into security of corresponding devices. We then present findings from the analysis of entire set of companion apps corresponding to the 96 devices.

B. Vulnerability Analysis

This section describes the method we used to analyze vulnerabilities. We first analyzed each app with respect to Section II-B's questions. Then, for each app, we looked for a potential attack path and confirmed it by creating a proof-of-concept exploit. We detail this process below.

1) Basic toolset functionality: We used a toolset to help answering the questions from Section II-B. The key components of the toolset are described next.

Encryption Discovery. The encryption discovery component looks for functions in the app that encrypt or decrypt the data exchanged with the smart device. Those functions can be the first line of attack for adversaries [15] and could be used to infer the layout of the messages and send unauthorized commands to the device. This component uses two complementary heuristics to discover these encryption functions. The first heuristic applies to the case where developers use existing Java encryption APIs. The second heuristic covers the case where developers implement custom crypto functions instead of building on existing ones; similar to previous work [16], [17], this component detects custom encryption functions by computing, for every function declared in the app, the ratio between the number of arithmetic and bitwise operations over the total number of instructions.

Network Protocol Discovery. This component extracts information about the smart device-companion app communication protocol. It looks for calls to functions related to communication protocols. For example, for UDP, it looks for calls to functions from the class `java.net.DatagramSocket` and, for TCP, it looks for calls to functions from `java.net.Socket`. The output of this component consists of a mapping from app classes to protocols.

Table II: Potential Threats to Selected Apps.

App	Avoid Hardcoded Keys?	Avoid Local Communication?	Avoid Broadcast Messages?	Secure Protocol?
Kasa Mobile	no	no	no	yes
LIFX	no encryption	no	no	yes
WeMo	no encryption	no	yes	no
e-Control	no encryption	no	no	yes

2) Answering the questions: The method used to answer each of the questions from Section II-B is described next.

Q1) Are encryption key(s) hardcoded? The search for hardcoded keys initiates from the output of the encryption discovery component, which reports function likely related to encryption. When using standard encryption libraries, we are able to automate the search for secret keys by looking for `javax.crypto.SecretKey`, which is the class denoting a key in the Java standard API. For custom encryption, however, we manually inspect each method returned by the encryption discovery, checking if the key is present inside the method body or in uses of the method. *Q2) Does the app use local communication?* In this case, the protocol discovery component acts as guidance for manual analysis. Based on the function calls and protocol report, we inspect the code to find whether or not the app uses local communication. *Q3) Does the app send broadcast messages?* Identifying whether broadcast messages are sent from the app to the smart device is done by inspecting the classes responsible for making network calls and looking for well-known broadcast addresses, e.g., `255.255.255.255`. *Q4) Does the app use any well-known protocol with vulnerabilities?* Using the network protocol discovery component, we can also check if the app uses some vulnerable protocol from the CVE database [18].

Results. Table II shows the answers to these questions for the four selected apps. For each question, we used the labels yes or no to indicate a positive answer or a negative answer, and no encryption for the first question when the app uses no encryption. The label yes indicates good practice whereas the labels no and no encryption indicate a potential vulnerability. To sum up, all four apps are found to use local communication with the device and three of the apps use broadcast communication. Three out of the four apps do not use any encryption to secure communication. One of the selected apps (WeMo) uses an insecure version of a protocol and does not use encryption.

C. Exploits

To create exploits, we first look for a vulnerable path in the code and then try to materialize that path. Below, we detail how we looked for paths and show the exploit created for the "Kasa for Mobile" app. For space reasons, we do not show exploits for the other apps. A technical report describes those exploits, including other devices, in more detail [4].

[4] http://arxiv.org/abs/1901.10062

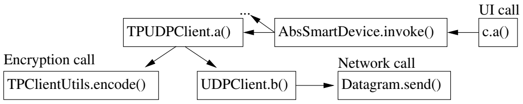


Figure 2: Path (simplified) from UI function to a network call.

```java
1 public static byte[] encode(byte[] data) {
2  byte seed = (byte) -85;
3  for (int i = 0; i < data.length; i++) {
4   data[i] = (byte) (data[i] ^ seed); seed = data[i];
5  } return data; }
```

Listing 1: TP-Link Kasa encryption function.

1) Finding Vulnerable Paths: To craft the exploit, we need to find a path i.e., a sequence of function calls that connects an UI call (the source) to a sink, e.g., a network method call. Figure 2 illustrates a vulnerable path for the Kasa app. To find this vulnerable path, we start by analyzing the output of our toolset, i.e., classes and functions related to encryption, authentication and network protocols. These elements are potential sources of vulnerabilities. Considering the Kasa app, for example, we start by inspecting the classes containing usages of the UDP protocol (related to Q3). We discover that the `UDPClient` class declares the network-related method `b`, which calls `datagramPacket.send()`, a method from the standard Java API to send UDP packets. As the method `b` includes a network call, it could be flagged as a sink. Our analysis shows that this class contains usages of broadcast addresses, representing a potential attack surface. Then, we analyze the call chain leading to this function looking for an UI method. In the process, we found that another method present in the output of our toolset (related to Q1), `TPClientUtils.encode`, contains hardcoded keys that could also be exploited. We also identify the function `TPUDPCLient.a`, responsible for building the UDP packet. This function, while not showing a vulnerability by itself, is responsible for building the UDP packet to be sent and reveals the structure of the message. Finally, we discover the calls to the UI obfuscated method `c.a`, which is the starting point of this path. That is possible because of the programming conventions of Android. More specifically, class `c` declares several (button-related) event callback methods.

2) The "Kasa for Mobile" exploit: TP-Link Kasa is the official app for controlling TP-Link-manufactured devices from the Kasa smart home product line [19]. Our exploit consists of a rogue app that enables an attacker to take control of a TP-Link smart plug as well as other TP-Link devices. Albeit the app targets a smart plug, it can control all devices from TP-Link given that they are on the same network as the smartphone.

Answering the questions. *Q1) Are encryption key(s) hardcoded?* The Kasa app uses a custom encryption function, Caesar cipher [20], that is known to be easy to break. Listing 1 shows this function as it appears in the app. Line 2 shows the hardcoded seed to encrypt the data. Identifying the encryption function and its hardcoded seed gave us hope of replicating the function in a rogue app on the same network to control the device arbitrarily. *Q2) Does the app use local communication?* By using the network discovery component and manually inspecting the code, we identified classes containing calls to UDP-related methods and confirmed that the methods that make these calls are involved in the discovery and control of the TP-Link devices on the local network. For instance, Listing 2 exhibits the function that discovers TP-Link devices in the local network. *Q3) Does the app send broadcast messages?* During our analysis, we found the Kasa app uses broadcast messages to discover and control the TP-Link devices. Line 1 from Listing 2 declares a constant variable holding a well-known broadcast IPv4 address. This variable is then used in Line 4 to discover TP-Link devices on the network. *Q4) Does the app use any well-known protocol with vulnerabilities?* For this case, we did not find uses of protocols with documented vulnerabilities.

```java
1 public static final String UDP_ADDRESS="255.255.255.255";
2 public void discoverLocal() {
3  String requestId = DiscoveryUtils.a();
4  tpDiscovery.broadcastDiscovery(...,UDP_ADDRESS,...);
5  ... }
```

Listing 2: TP-Link Kasa function (simplified) used to discover devices on the local network.

Confirming Vulnerabilities. We designed a proof-of-concept exploit to confirm vulnerabilities; the exploit consists of a rogue app running on the same network. To create the attack we followed these steps: 1) find a vulnerable path and encryption function, 2) discover the structure of exchanged messages, 3) discover what protocol is used to exchange messages, and 4) implement pairing.

From a vulnerable path of the Kasa app, we obtained access to the app's encryption function. We created a test script that trivially broke the cipher and monitored the network traffic, reading the contents of messages, extracting their structure, and the IP addresses used. We also found that broadcasting was used through a single address. Finally, it was necessary to analyze the pairing process. To our surprise, we found by inspection that a pairing process was only used to maintain the profile of users on TP-Link devices, but not for their control.

Monitoring the Network. We used the popular traffic analyzer Wireshark [21] to monitor the packets exchanged between the Kasa app and the device. As the traffic was encrypted we needed to implement a script to decrypt the monitored messages; the script uses the symmetric cipher function from Listing 1. This monitoring tool was used in two important stages: (i) during the app-device pairing process and (ii) while the app interacted with the device, e.g., turning the plug "on" and "off". During the pairing process, we found that broadcast messages were exchanged while the app was connected to the hotspot created by the device. We also monitored the network when interacting with the device through the app's UI. Specifically, we repeated the "Turn Off" and "Turn On" operation multiple times, observing that the contents of the network packets did not change, confirming the use of a hardcoded key (with a poor encryption method). We also observed the use of broadcast messages during device usage after pairing. We also found that the app uses the following message to discover and obtain the current status of the device– `{"system":{"get_sysinfo":}{}}`. Likewise, we found that `{"system":{"set_relay_state":{"state":0}}}` was the message used to turn the device on/off.

Based on the info we collected, we created a rogue app to control the TP-Link smart plug. To sum up, static analysis assisted in the discovery of vulnerable paths whereas dynamic analysis helped in understanding the communication protocol and the messages exchanged. Recall that, during our analysis, we noticed that the pairing process was not needed to control the device. This is a severe flaw as the user would not even be aware of an attack—the official app would still work as intended even with a rogue app controlling the device simultaneously. A video demonstrating the exploit is available online [11].

Vulnerability disclosure. We disclosed the vulnerabilities, along with scripts for exploits, to the manufacturers of the five devices exploited in October 2018. All of them acknowledged the disclosures but, to the best of our knowledge, have not released patches to address the disclosures.

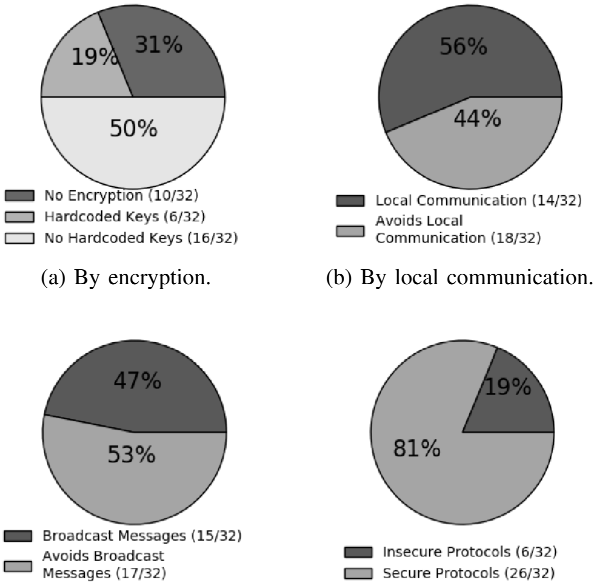
D. Vulnerability Analysis on a Larger Set of Apps

This section describes findings from the study of all the apps corresponding to the 96 devices we have originally identified, as discussed at the beginning of Section III-A. In this case, we did *not* purchase devices, so these findings are indicative of the potential extent of vulnerabilities, but require further confirmation. IoT devices included cameras, locks, and alarms. including the 5 devices that we previously purchased and analyzed in Section III-A. These 96 devices only correspond to 32 companion apps, saving us significant analysis effort compared to analysis of devices themselves.

Figure 3 shows, as pie charts, the distribution of answers to the questions for the apps analyzed. Of the 32 apps, we found only 4 apps using encryption without hardcoded keys, not using local communication, not using broadcasts, and not using known insecure protocols. All their communication was via the cloud service, likely over SSL. The four apps include the popular Nest app. With respect to attacks considered in this paper, this is a relatively secure way to communicate. But it does have a privacy tradeoff in that the cloud service has access to the commands and data sent to the device. Consequently, a potential long-range privacy and security risk exists if the cloud service is ever compromised [22].

IV. Limitations of Our Study

Both static and dynamic analysis techniques are fundamentally limited. They could have failed to detect use of encryption that is obfuscated in some way, for example, a custom Java implementation or implemented in native libraries (JNI). We did look for both custom implementations and calls to crypto functions via JNI when inspecting the code manually along potentially vulnerable paths to the extent feasible. The custom crypto implementations were generally worrisome – the ones we found were implementing weak ciphers, for example, the Caesar cipher. We found 5 of the apps that appeared to invoke encryption functions via JNI. Further analysis would be required to determine if those encryption functions were used properly and what exactly they did.



(a) By encryption.

(b) By local communication.

(c) By broadcast messages.

(d) By security in protocols.

Figure 3: Distributions of apps by features.

V. Discussion and Lessons Learned

We were somewhat surprised to find the extent to which insecure devices were among top selling IoT devices. For example, at the time of our analysis, TP-Link's smart plug was a top-seller with over 12K customer reviews on Amazon [23] and a rating of 4.4 out of 5 stars.

It is apparent that consumers do not have a good way to shop for devices based on security considerations. Kasa Smart Plug does require a password-protected WiFi network, but one customer commented on that on Amazon: *"Forced to use a password on WiFi router. Live in the middle of nowhere and no need for this security. No reason for this manufacturer to force this security."* We clearly have a long road ahead in securing IoT systems – not just technically but also in creating user awareness around security and in developing appropriate disclosure policies for vendors to follow.

The app-device communication strategies that survived our checks may offer some guidance. Nest thermostat's companion app does not talk directly to the device; instead, the user creates a free account on the Next cloud service and then signs into that using the companion app over SSL. Furthermore, the thermostat and the cloud service can also mutually authenticate each other and establish a shared secure link. No shared keys between the companion app and the thermostat are required since, from then on, the communication between the companion app and the thermostat happens over SSL links to the cloud service. The EZVIZ uses a different strategy. Unlike Nest, it supports local communication between the companion app and the device over the local network. The shared encryption key is enclosed in the box in the form of a QR code and must be scanned by the companion app. This strategy is better than hardcoded keys provided the key in the QR code is of sufficient length, random, and strong crypto library is used.

VI. Related Work

Denning et al. [3] presented potential security attacks against smart home devices, pointing that common attacks to traditional computing platforms, like denial-of-service and eavesdropping on network, could also be used in a smart home context. Komninos et al. [24] also analyzed smart home devices, presenting a survey that categorized potential threats in this domain, e.g., device impersonation.

Focusing on IoT platforms, Fernandes et al. [6] analyzed over 499 apps on SmartThings and found out that 55% of those are over-privileged largely due to design flaws in the privilege model of the platform. The authors also demonstrated how to take advantage of this with four proof-of-concept attacks, both remote and local. Jia et al. proposed a context-based permission system for appified IoT platforms with fine-grained context identification and runtime prompts [25].

Android apps have been analyzed for a variety of security-related issues, such as cryptographic misuse [15], [26]. For example, Egele et al. [15] analyzed the violation of six rules including the use of ECB mode and constant keys. Wei et al. [26] designed a static analysis tool for security vetting of Android apps and used it to detect the use of the weak ECB mode for encryption; the analysis is intra-procedural and thus limited in scope.

In 2015, the Veracode team published a white paper on security analysis of six IoT devices to examine vulnerabilities such as non-use of cryptography and lack of strong passwords [27]. They used both network monitoring and reverse engineering techniques. Our work differs in that it focuses on a different set of vulnerabilities, presenting a detailed analysis of companion apps to show how such vulnerabilities can be discovered.

VII. Conclusions

Securing communication between IoT devices and the mobile apps responsible for controlling them is crucial for security and even safety, depending on the types of IoT devices on a network. In this study, we showed that analyzing the smartphone companion apps that are released for the device can provide important clues for potential vulnerabilities in the devices. We analyzed 32 companion apps corresponding to 96 popular IoT devices to assess whether the communication between the devices and their communication app is properly secured. We found significant concerns. For instance, we found that 31% of the apps do not appear to use any crypto to protect the device-app communication and that 19% use hardcoded keys. We also purchased five devices. From insights offered by analysis of their companion apps, we were successful in creating exploits for all five devices and able to control them. The study suggests that there may be a long road ahead in securing IoT systems – issues are not just technical but also non-technical, such as creating mechanisms for consumer awareness of security features and risks when they purchase IoT devices.

References

[1] Gartner Group, “Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016,” 2016. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[2] J. Scott and D. Spaniel, “Rise of the Machines,” http://icitech.org/wp-content/uploads/2016/12/ICIT-Brief-Rise-of-the-Machines.pdf, Institute for Critical Infrastructure Technology (ICIT), 2017.

[3] T. Denning, T. Kohno, and H. M. Levy, “Computer security and the modern home,” *Commun. ACM*, vol. 56, no. 1, pp. 94–103, Jan. 2013.

[4] “SmartThings website.” [Online]. Available: https://www.smartthings.com/

[5] “Echo & Alexa - Amazon Devices - Amazon Official Site.” [Online]. Available: https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011

[6] E. Fernandes, J. Jung, and A. Prakash, “Security analysis of emerging smart home applications,” in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 636–654.

[7] X. Lei, G. Tu, A. X. Liu, C. Li, and T. Xie, “The Insecurity of Home Digital Voice Assistants - Amazon Alexa as a Case Study,” *CoRR*, vol. abs/1712.03327, 2017.

[8] VERACODE, “Use Veracode to secure the applications you build, buy, & manage,” https://www.veracode.com, 2018.

[9] “Reverse Engineering the TP-Link HS110,” https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/, SoftScheck GMBH, 2018.

[10] TP-Link, “WiFi Networking Equipment for Home & Business - TP-Link.” [Online]. Available: https://www.tp-link.com

[11] D. Mauro Junior and L. Melo, “Kasa video exploit,” 2018. [Online]. Available: https://figshare.com/s/d5bc439a7527df358f5f

[12] T. Ambient, “Zigbee vs Z-Wave: Two big smart home standards explored,” https://www.the-ambient.com/guides/zigbee-vs-z-wave-298, 2018.

[13] “Tuya Smart - World’s leading IoT platform.” [Online]. Available: http://www.tuya.com/

[14] CERT, 2013. [Online]. Available: https://www.us-cert.gov/ncas/current-activity/2013/01/29/CERT-Releases-UPnP-Security-Advisory

[15] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An Empirical Study of Cryptographic Misuse in Android Applications,” in *CCS*, 2013, pp. 73–84.

[16] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, “Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-engineering,” in *CCS*, 2009, pp. 621–634.

[17] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, “ReFormat: Automatic Reverse Engineering of Encrypted Messages,” in *Computer Security – ESORICS*, 2009.

[18] MITRE, “CVE - Common Vulnerabilities and Exposures.” [Online]. Available: https://cve.mitre.org/

[19] TP-Link, “Kasa smart home products.” [Online]. Available: https://www.tp-link.com/us/kasa-smart/kasa.html

[20] W. Chapman, “Caesar cipher.” [Online]. Available: https://courses.physics.illinois.edu/cs125/su2017/mp3_caesarcipher.php

[21] “Wireshark •go deep.” Wireshark Foundation. [Online]. Available: https://www.wireshark.org

[22] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, “Decentralized Action Integrity for Trigger-Action IoT Platforms,” in *22nd Network and Distributed Security Symposium (NDSS 2018)*, Feb. 2018.

[23] “Kasa Smart Wi-Fi Plug by TP-Link.” [Online]. Available: https://www.amazon.com/Kasa-Smart-Wi-Fi-Plug-TP-Link/dp/B0178IC734?keywords=tp-link&qid=1539182867&sr=8-7&ref=sr_1_7#customerReviews

[24] N. Komninos, E. Philippou, and A. Pitsillides, “Survey in smart grid and smart home security,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1933–1954, 2014.

[25] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, “ContexIoT: Towards providing contextual integrity to appified IoT platforms,” in *NDSS*, 2017.

[26] F. Wei, S. Roy, X. Ou, and Robby, “Amandroid: A precise and general inter-component data flow analysis ramework for security vetting of Android apps,” in *CCS*, 2014, pp. 1329–1341.

[27] Veracode, “The Internet of Things: Security Research Study,” https://www.veracode.com/sites/default/files/Resources/Whitepapers/internet-of-things-whitepaper.pdf.