BurnFit: Analyzing and Exploiting Wearable Devices

Dongkwan Kim, Suwan Park, Kibum Choi, and Yongdae Kim(✉)

Korea Advanced Institute of Science and Technology (KAIST),
291 Daehak-ro, Daejeon, Republic of Korea
{dkay,skyhwen,kibumchoi,yongdaek}@kaist.ac.kr

Abstract. Wearable devices have recently become popular, and more and more people now buy and wear these devices to obtain health-related services. However, as wearable device technology quickly advances, its security cannot keep up with the speed of its development. As a result, it is highly likely for the devices to have severe vulnerabilities. Moreover, because these wearable devices are usually light-weight, they delegate a large portion of their operations as well as permissions to a software gateways on computers or smartphones, which put users at high risk if there are vulnerabilities in these gateways. In order to validate this claim, we analyzed three devices as a case study and found a total 17 vulnerabilities in them. We verified that an adversary can utilize these vulnerabilities to compromise the software gateway and take over a victim's computers and smartphones. We also suggest possible mitigation to improve the security of wearable devices.

Keywords: Wearable device security · Communication channel analysis

1 Introduction

Wearable device, a representitive example of the *Internet of Things (IoT)*, have been increasing their market share recently. Researchers expect that this market share will reach 13 billion dollars by 2018 [23]. As increasing numbers of people are wearing these devices, adversaries may turn their attention to steal customers' private information. Despite their popularity, research on the security of wearable devices has been lacking. One can consider wearable devices as an embedded device, whose security has been actively studied recently [5,8,11,21,28]. These studies mainly focus on (semi) automatic analysis of their firmware.

In order to prepare for the IoT era, we have to understand the current state of art in designing wearable devices, as they represent the first batch of IoT devices. To this end, we decided to comprehensively evaluate the security of wearable devices through multiple steps. First, we classify the attack vectors on these devices. To save energy, wearable devices push their computational overhead and many permissions to their corresponding application on computers or smartphones, so called *software gateways*. These software gateways take the

H. Kim and D. Choi (Eds.): WISA 2015, LNCS 9503, pp. 227–239, 2016.
DOI: 10.1007/978-3-319-31875-2_19

role of medium between wearable devices and servers for updating the device or storing user data for customer services. Wearable devices utilize Bluetooth Low Energy (BLE) to communicate with the software gateways, and software gateways connect to the servers as usual.

We chose three popular wearable devices from a top 10 list [22], and analyzed and classified attack vectors for their update, data, and BLE channels as well as those for the device itself. From this analysis, we discovered a total of 17 vulnerabilities in the devices and showed that a user's private information can be exposed in plaintext. We successfully exploited them to take control of the software gateways as well. Finally, we present mitigations in the design stage to prevent wearable devices from being compromised.

To summarize, we make the following contributions in this study:

- We classified possible attack vectors related to the wearable devices that can be applied to most low-power IoT devices.
- We performed an analysis on our target devices and found 17 vulnerabilities. An adversary can take the control of the software gateways as well.
- We discuss mitigations for wearable devices in their operational channels to securely protect a user's private.

The rest of this paper is organized as follows: Section 2 outlines existing security research related to wearable devices and embedded systems. Section 3 overviews wearable device system as a whole and introduces our threat model. In Sect. 4, we analyze the vulnerability of wearable devices and their exploitation is described in Sect. 5. We discuss the analysis result and mitigation of these vulnerabilities in Sect. 6, and we conclude the paper in Sect. 7.

2 Related Works

Attacks on Wearable Devices. When no wearable device was available on the market in the early 2000s, there were a few studies forecasting security and privacy issues of wearable devices [3,7,13], but they were not really materialized until now. Recently, Ryan exposed vulnerabilities in the BLE interface, a dominant communication channel for wearable devices, and showed that sniffing is possible [19]. After that, Barcena at el. applied this research to actual wearable devices [6]. In particular, the authors proved that they can extract user information from the BLE interface between the target device and smartphone. In contrast to these previous studies, we perform a comprehensive study to classify possible attack vectors of the operational channels on wearable devices and analyze all.

Attacks on Embedded Devices. Security research on embedded devices is also relevant. Zaddach at el. introduced various ways to analyze the vulnerability of embedded devices [28]. In addition, Costin et al. performed a large-scale security analysis on embedded devices [10]. The authors improved their research and proposed an analysis platform for embedded devices called Avatar [27]. With this platform, firmware of embedded devices can be emulated and debugged along with the peripherals.

In addition, there have been several other approaches to embedded devices. AEG [5], Mayhem [8], FIE [11], and Firmalice [21] are frameworks for detecting vulnerabilities in firmware utilizing symbolic execution and taint analysis. These analysis methods on embedded devices can also be applied to determine vulnerabilities in the wearable devices. However, in this study, we focus on operational channels rather than their firmware. Furthermore, these studies are limited to Intel or ARM platform.

3 Background

3.1 Wearable Device Overview

A wearable device is a device that users can wear in their daily lives. Because wearable devices are light-weight and use low energy, they offload their computational overhead to software gateways. Therefore, almost all wearable devices are connected to a computer or smartphone, and this connection is usually based on Bluetooth. In this case, computers and smartphones take the role of software gateways to bridge the wearable devices to update and database servers through the Internet. Figure 1 illustrates an overview of the operational procedure of wearable devices.

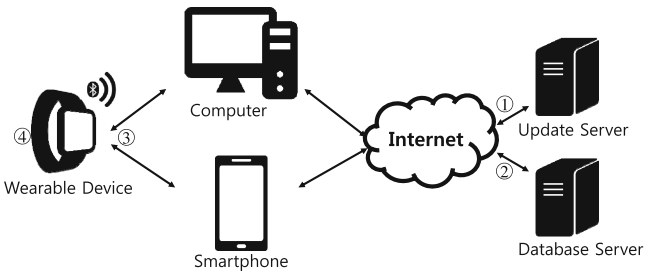

Fig. 1. General operational procedure of wearable devices

3.2 Bluetooth Low Energy (BLE)

Bluetooth Smart, also called BLE, is a relatively new protocol proposed after Bluetooth 4.0. Over the years, the security of classic Bluetooth has been improved. But BLE is a different protocol from classic Bluetooth, and, therefore, its security has to be analyzed almost from scratch.

Because BLE is designed to reduce power consumption, the randomness of the frequency hopping sequence, which protects the classic Bluetooth signal from sniffing, is replaced with an insecure modular addition. Furthermore, in classic Bluetooth, the secure simple pairing (SSP) technique effectively protects it from Man-in-the-Middle (MitM) attacks. However, this feature was only introduced in Bluetooth 4.1, meaning that devices implementing the earlier versions

of Bluetooth would be vulnerable. Instead of using SSP, BLE utilizes six-digit pin numbers as a Temporary Key (TK), but this can be brute-forced within a second.

Ryan identified the vulnerabilities of the BLE interface pairing procedure in [19]. However, because many devices use customized BLE stack, their communcation should be analyzed to determine the exact meaning of the payload data in each BLE packet.

3.3 Threat Model

Wearable devices offload their operation to software gateways (i.e., applications on computers and smartphones). This includes access to various information such as health information, SMS message, calendar, email, and call history. Therefore, if an adversary can compromise the software gateway, she can take over the whole information that the wearable device can access. Furthermore, as wearable devices use more communication channels than a computer, they are likely to have more vulnerabilities.

① **Gateway to Update Server.** If there is a vulnerability in the update channel, this would be critical. An adversary can replace a benign update file to her own malicious one to compromise the software gateway as well as the wearable device. Usually this update channel is in a content management system managed by third party environment; hence, it could have vulnerabilities with respect to a link or URL access.

② **Gateway to Database Server.** As many wearable devices operate with health applications, they collect health information and send it to the database server. Based on this collected data, companies provide health-related services to their customers. However, if this channel is vulnerable, private user information can be exposed to an adversary.

③ **BLE Connection.** Wearable devices are usually connected to their software gateway through BLE. Because the BLE interface aims for low energy consumption, it is highly likely to have vulnerabilities in the pairing or communication procedure. Furthermore, the data encapsulated in the BLE signals may not be encrypted. In addition, if the channel is not authenticated, an adversary can directly send a request to the device and extract private user information.

④ **Device Analysis.** If there are vulnerabilities on the device itself, an adversary can exploit them to compromise the device itself. In this case, the adversary can control the device arbitrarily: sending multiple alarms or vibration to wake up the victim at night. Additionally, the adversary could extract a user's health information through the BLE interface.

We assume an adversary may have her own device for the analysis, but she does not have a physical access to the victim's device. In addition, we assume

an adversary has access to the same or upper layer of the network so that she can perform DNS spoofing to redirect packets to her server. In addition, the adversary could be located close to the target so that she can send and receive BLE packets from the device.

4 Vulnerability Analysis of Wearable Devices

4.1 Methodology

We targeted three devices: A-fit, B-fit, and C-fit, all of which are fitness trackers that are on the list of the most popular wearable devices [22]. As we mentioned in Sect. 3, the software gateways bridging the wearable devices to the servers through the Internet may have many vulnerabilities.

We analyzed the devices as well as the operational channels: update, data, BLE. To analyze the update channel, we disassembled the software gateways using IDA Pro [14], and checked whether an adversary could substitute updates and device firmware with malicious ones. For the application analysis on Android smartphones, we utilized decompiling tools such as ApkTool [24], Smali/Baksmali [15], and dex2jar [18]. For the data channel, we analyzed the security of the health information by determining whether it is exposed as plaintext. For the analysis of BLE channel, it was determined whether sniffing or spoofing over-the-air data was possible. We sniffed BLE packets with Ubertooth [17], and analyzed them with Wireshark [9]. For device analysis, we searched for hardware debug points that could be utilized to extract firmware or other device information directly. From this analysis, we discovered total 17 vulnerabilities among three devices, as listed in Table 1.

Table 1. Vulnerabilities found on A-fit, B-fit, and C-fit. (âŰş means there was only partial obfuscation.)

Channel	Attacks	A-fit	B-fit	C-fit
Update channel	No obfuscation on app	✓	âŰş	✓
	DNS spoofing	✓	✓	✓
	App substitution	✓	✘	✓
	Firmware substitution	✘	✘	✓
Data channel	Plaintext data transfer	✓	✘	✘
BLE channel	Sniffing	✓	✓	–
	Plaintext data transfer	✓	✓	–
Device analysis	No obfuscation on firmware	✘	✘	✓
	Hidden function	✘	✘	✓
	Hidden protocol	✘	✘	✓
	Hardware debug point	✘	✘	✘

4.2 Update Channel Analysis

We performed both traffic and application analysis by respectively monitoring the network and disassembling the application to find vulnerabilities in the update channel. The A-fit has a couple of software gateways: applications on a smartphone and computer. The other devices only have a smartphone application.

A-fit. The A-fit can be connected to either a smartphone or computer using the provided Bluetooth connector. The smartphone loads its data using HTTPS, and this application was heavily obfuscated. Therefore, we chose to analyze the application on the computer, which was not obfuscated. As a result, we could easily figure out the firmware update procedure.

When a user first connects the A-fit to a computer, the device determines whether there is an update for the application on the computer. In addition, the user can directly send an update request message to the server by clicking an update button on the side of the application screen. From this analysis, we discovered that there was a hardcoded URL for the update server. Additionally, the A-fit application utilizes HTTP, which has no protection mechanism, so users are exposed to sniffing or even spoofing.

Fig. 2. XML file contents in A-fit update procedure. (Critical information was redacted to protect device.)

To update itself, the application first accesses to an XML file located at the update server of A-fit, as shown in Fig. 2. It then downloads an update file from the secondary server following the link written in the body of the XML file after checking its version. Because the A-fit application runs on Windows, this downloaded update file is an executable (.exe). After downloading it, the A-fit automatically executes the file to continue its update. From the analysis of the application we found that there is neither authentication of the downloaded file nor integrity check/verification of it. Therefore, if an adversary can perform DNS spoofing and make the A-fit application access her spoofed XML, it is possible

to substitute a benign update file with a malicious one so that the application will execute it and compromise the victim's computer.

B-fit. In case of B-fit, there was only partial obfuscation on the application, and this obfuscation was not applied to the update routine. Therefore, we were able to analyze the application in a similar way to A-fit.

B-fit has similar vulnerability as A-fit; it first accesses to an XML file that contains the link of the update file. Even though it utilizes HTTPS to protect the connection from spoofing and verifies the certificate of the update server, there was no protection for the link of the update file in the XML file. The link was HTTP, which enables an adversary to perform a DNS spoofing attack to change the original update to a malicious one.

However, B-fit utilizes the update channel only for the firmware update. Because the application on the smartphone is updated from the app store, it was not able to modify the application. Furthermore, the update file was encrypted and decrypted on the B-fit device. Therefore, even though an adversary could perform DNS spoofing, she cannot create properly functioning firmware without the key.

C-fit. C-fit was the most vulnerable device in our test. Similar to B-fit, it only supports a connection with an Android smartphone that has the BLE feature. When C-fit is first connected to a smartphone, the application on the smartphone checks the firmware version as well as the application version in a similar way to A-fit. It accesses an XML file from the update server, follows the URL within it, and downloads an update file. We found that the application contains the new version of firmware in its *assets* folder. The application was not obfuscated, and there was neither integrity check nor authentication of the application and firmware, just as for A-fit. Therefore, an adversary could perform DNS spoofing and manipulate the application as well as the C-fit firmware.

4.3 Data Channel Analysis

We analyzed the security of the data channels by investigating the capability of sniffing. When A-fit transfers a user's data to the database server, it utilizes HTTP which is vulnerable to sniffing. As shown in Fig. 3, the application transfer the victim's private information in plaintext, which is only encoded in Base64. This private information includes the version of the victim's operating system, victim's private information in plaintext, health information, and device model, which is only encoded in Base64. This private information includes the version of the victim's operating system, health information, and device model.

By monitoring the network, an adversary can easily collect the private information of the users. Based on this collected data, the adversary could perform targeted attack in the real world. For example, an adversary could determine whether the target is sleeping or not before breaking into a house, or could recognize whether the target is sick. We also found that not only health information, but also the user's account information (i.e., ID and password) was exposed in plaintext. Therefore, from this analysis, we determined that the A-fit

Fig. 3. Plaintext health information of a user. (The box shows user's private information and critical information of device was redacted.)

was vulnerable to sniffing because it utilized HTTP. However, in case of B-fit and C-fit, they encrypted their data using HTTPS with certificate verification, which made it difficult to extract information.

4.4 BLE Channel Analysis

As all these devices are known to use BLE, we analyzed the BLE channel to check if an adversary could access a user's health information. First, we needed to determine which version of Bluetooth was utilized. This information can be found in the BLE advertisement packets; there is a *Flags* field that indicates the configuration of BLE discoverable mode as well as the rates for data transferring.

As A-fit and B-fit only utilize BLE for communication, we were able to sniff the communication process via a pairing step. Moreover, these two devices did not have any option for BLE encryption. Because most BLE devices communicate using their own protocol implementation to increase energy efficiency, we could not understand the captured wireless packets. However, we analyzed the Android applications and figured out the data structure of the BLE packets. If an adversary is within BLE range, she can read the health information of the devices and even send fake data to the software gateways.

The C-fit supports classic Bluetooth. During our repeated experiments, we could not find any connection request in BLE protocol. This is because C-fit utilizes classic Bluetooth to send its actual data, and BLE is only used for the device ID advertisement to reduce power consumption.

4.5 Device Analysis

We also analyzed the devices to explore possible hidden functions. In the case of C-fit, the firmware consists of a file system, main executable binary, and binary for group of library functions. The firmware was not encrypted, so we could analyze it to find hidden functions and protocols.

The C-fit has an engineering mode to provide debugging at the service center. By analyzing the firmware, we could find all of the hidden functions, as listed in Table 2. Some hidden functions were published in [2], but this study did not report the full list.

Table 2. Codes for hidden functions in the C-fit engineering mode (bold codes were published in [2])

Codes for hidden menu					
#0	*#737425	*#1111	*#250	*#2222	*#251
***#1234**	*#350	*#12580*369	*#0228	***#232337**	***#2663**
*#2664	*#7353	***#7284**	***#9900**	*#22228378	*#232331
*#533881	*#533883	*#737425			

In addition, we found that C-fit has a modem feature that accepts AT commands similarly to a smartphone if it is connected to a computer with a micro USB interface. The C-fit supports multiple AT commands to configure or check device information. We discovered that there was a buffer overflow vulnerability in the AT commands. For example, if we entered an extremely long string as a parameter of the AT command, the C-fit displays the text 'Hardware Fault' on its screen and reboots itself. If an adversary can physically access the device, she can exploit this vulnerability to compromise the device. After it is compromised, she can then send fake health information to the smartphone through the Bluetooth interface and make multiple false alarms.

Furthermore, we searched for hardware debugging points that we could utilize to extract device information or firmware. Because we could not obtain the firmware of the A-fit and B-fit, we need to utilize these debugging points. We focused on exploiting the Universal Asynchronous Receiver/Transmitter (UART), Joint Test Action Group (JTAG), and Serial Wire Debug (SWD) interfaces [25,26], as many embedded devices generally contain these debugging features.

We utilized JTAGulator [16] and J-link [20] to determine if one of debugging interfaces exists. JTAGulator automatically generates signals to find debugging pins for UART and JTAG, but we could not find any such pins. In the case of SWD, we needed to input every single pin to find the debugging interface. However, as there were too many combinations of pins, we could not explore all pins. We also used a logic analyzer from Quant Asylum to directly analyze the signals from each pin, but the signal was too complicated to analyze. Therefore, we were not able to find one on the devices.

5 Exploiting Wearable Devices

We verified the exploitability of the update channel from the discovered vulnerabilities because this is the most critical part that can harm users. We prepared one phone as a victim, our own update server, and a laptop for DNS spoofing with [1]. As a result, the software gateways of the A-fit and C-fit were successfully compromised, but we were not able to take over the B-fit.

A-fit. We first performed DNS spoofing to substitute the A-fit application with our own. As we mentioned in Sect. 4.2, the application checks for updates when a user clicks on a button. We placed a malicious XML file and executable file on our server, and redirected the update traffic to it. As a result, the application on the victim's computer downloaded our file and executed it, as shown in Fig. 4. As a proof-of-concept, we utilized the Windows calculator, but any application could be executed here.

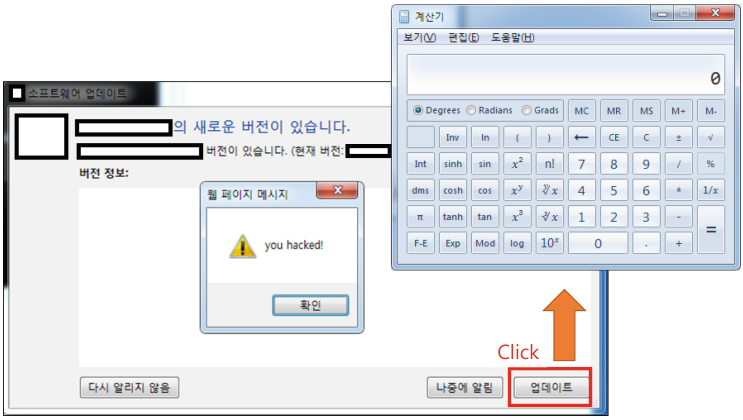

Fig. 4. Successful compromise of the victim's computer after DNS spoofing on the A-fit application. (Critical information of device was redacted. Note that words in the figure are Korean due to the OS language pack.)

C-fit. We also tried DNS spoofing the C-fit. In the case of C-fit, we were able to compromise both the Android application and the C-fit device firmware. We could install our malicious application successfully on the victim's phone. Furthermore, the installed application inherited the permissions from the original C-fit application, which includes many permissions such as accessing the phone book, extracting SMS messages, connecting to the Internet, and so on. Therefore, we could perform almost all behaviors on the victim's smartphone.

6 Discussion

6.1 Failure to Debug Hardware

Most hardware developers make debugging points to help them to remove bugs. However, these debugging points are eliminated before shipping or are located

in a hidden space that only developers can recognize. Therefore, it takes a large amount of time and effort to find such points. In this experiment, we could not find any hardware debugging points among the three devices because the devices were too small and there were no public datasheets. However, even though we could not find hardware debugging interface in our target devices, we found 17 vulnerabilities from the operational channels that could enable an adversary to compromise the victim's computer and smartphone.

6.2 Improving the Security of Wearable Devices

To protect wearable devices securely, the update, data, and BLE channels should be securely protected as well as the device. Currently, the update and data channels from the software gateway to the servers are easily spoofed because they do not use encryption. Even the BLE interface has no encryption, and hence an adversary could exploit these channels. Because these channels are necessary, as light-weight wearable devices cannot directly communicate with servers, the channels should operate using SSL or TLS with certificate verification. This can prevent an adversary from manipulating packets to inject malicious update files or to extract private information from the users. Furthermore, the software gateways as well as the devices should be protected; integrity checks should be performed to prevent and detect modification of the firmware or application as in Sect. 5. In addition, TrustZone and secure boot should be applied to prohibit an adversary from bypassing the integrity check [4,12].

7 Conclusion

As wearable devices are currently becoming popular, development should be accompanied by security research. Therefore, we classified possible attack vectors related to some wearable devices. We analyzed three devices and found a total 17 vulnerabilities among them. Utilizing these vulnerabilities, we verified the exploitability of the software gateways. We also discussed ways to protect these wearable devices.

However, these issues are not only a matter for wearable devices, as other IoT devices might have similar problems. As more and more IoT devices are developed and appear on the market, researchers should take their security into account and take steps to resolve their vulnerabilities at the design stage.

Acknowledgement. This research was supported by (1) Next-Generation Information Computing Development Program through the NRF (National Research Foundation of Korea) funded by the MSIP (Ministry of Science, ICT and Future Planning) (No. NRF-2014M3C4A7030648), Korea, and by (2) the MSIP, Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-R0992-15-1006) supervised by the IITP (Institute for Information and Communications Technology Promotion).

References

1. Cain & Abel. http://www.oxid.it/cain.html. Accessed 8 June 2015
2. Redacted to protect the device
3. Al-Muhtadi, J., Mickunas, D., Campbell, R.: Wearable security services. In: 2001 International Conference on Distributed Computing Systems Workshopp, pp. 266–271. IEEE (2001)
4. Alves, T., Felton, D.: TrustZone: Integrated hardware and software security. ARM White Pap. **3**(4), 18–24 (2004)
5. Avgerinos, T., Cha, S.K., Hao, B.L.T., Brumley, D.: AEG: automatic exploit generation. In: NDSS, vol. 11, pp. 59–66 (2011)
6. Barcena, M.B., Wueest, C., Lau, H.: How safe is your quantified self. Symantech, Mountain View (2014)
7. Campbell, R.H., Al-Muhtadi, J., Naldurg, P., Sampemane, G., Mickunas, M.D.: Towards security and privacy for pervasive computing. In: Okada, M., Babu, C.S., Scedrov, A., Tokuda, H. (eds.) ISSS 2002. LNCS, vol. 2609, pp. 1–15. Springer, Heidelberg (2003)
8. Cha, S.K., Avgerinos, T., Rebert, A., Brumley, D.: Unleashing mayhem on binary code. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 380–394. IEEE (2012)
9. Combs, G., et al.: Wireshark, pp. 12–02 (2007). http://www.wireshark.org/lastmodified
10. Costin, A., Zaddach, J., Francillon, A., Balzarotti, D., Antipolis, S.: A large-scale analysis of the security of embedded firmwares. In: USENIX Security Symposium (2014)
11. Davidson, D., Moench, B., Ristenpart, T., Jha, S.: FIE on firmware: finding vulnerabilities in embedded systems using symbolic execution. In: USENIX Security, pp. 463–478 (2013)
12. Davis, D.L.: Secure boot , US Patent 5,937,063 (1999)
13. Di Pietro, R., Mancini, L.V.: Security and privacy issues of handheld and wearable wireless devices. Commun. ACM **46**(9), 74–79 (2003)
14. Eagle, C.: The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. No Starch Press, San Francisco (2011)
15. Freke, J.: Smali. https://code.google.com/p/smali. Accessed 7 June 2015
16. Grand, J.: JTAGulator: assisted discovery of on-chip debug interfaces. In: 21st DefCon Conference, Las Vegas (2013)
17. Ossmann, M.: Project ubertooth, p. 23 (2012). Accessed 18 Nov 2012
18. Pan, B.: dex2jar. https://github.com/pxb1988/dex2jar. Accessed 7 June 2015
19. Ryan, M.: Bluetooth: with low energy comes low security. In: WOOT (2013)
20. SEGGE: Debug Probes - J-Link and J-Trace. https://www.segger.com/jlink-debug-probes.html. Accessed 6 June 2015
21. Shoshitaishvili, Y., Wang, R., Hauser, C., Kruegel, C., Vigna, G.: Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. In: NDSS (2015)
22. Stables, J.: Best fitness trackers 2015: Jawbone. Misfit, Fitbit, Garmin and more, April 2015. http://www.wareable.com/fitness-trackers/the-best-fitness-tracker
23. Statista: Wearable device market value from 2010 to 2018 (2015). http://www.statista.com/statistics/259372/wearable-device-market-value. Accessed 9 June 2015

24. Tumbleson, C., Wisniewski, R.: Apktool. http://ibotpeaches.github.io/Apktool. Accessed 7 June 2015
25. Wikipedia: Joint test action group – wikipedia, the free encyclopedia (2015). http://en.wikipedia.org/w/index.php?title=Joint_Test_Action_Group&oldid=663324599. Accessed 5 June 2015
26. Wikipedia: Universal asynchronous receiver/transmitter – wikipedia, the free encyclopedia (2015). http://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver/transmitter&oldid=663120875. Accessed 5 June 2015
27. Zaddach, J., Bruno, L., Francillon, A., Balzarotti, D.: Avatar: a framework to support dynamic security analysis of embedded systems firmwares. In: Symposium on Network and Distributed System Security (NDSS) (2014)
28. Zaddach, J., Costin, A.: Embedded devices security and firmware reverse engineering. Black-Hat USA (2013)