Digital Communications and Networks xxx (xxxx) xxx

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Developing a platform to evaluate and assess the security of wearable devices

Matthew L. Hale [a,*,1], Kerolos Lotfy [a,2], Rose F. Gamble [b,3], Charles Walter [b,4], Jessica Lin [b,5]

[a] Department of Cybersecurity, School of Interdisciplinary Informatics, University of Nebraska at Omaha, Omaha, NE, 68135, USA
[b] Tandy School of Computer Science, University of Tulsa, Tulsa, OK, 74104, USA

ARTICLE INFO

Keywords:
Bluetooth LE
Internet of things
Man-in-the-middle attacks
Security
Vulnerability discovery
Wearables

ABSTRACT

Operating in a body area network around a smartphone user, wearables serve a variety of commercial, medical and personal uses. Depending on a certain smartphone application, a wearable can capture sensitive data about the user and provide critical, possibly life-or-death, functionality. When using wearables, security problems might occur on hardware/software of wearables, connected phone apps or web services devices, or Bluetooth channels used for communication. This paper develops an open source platform called SecuWear for identifying vulnerabilities in these areas and facilitating wearable security research to mitigate them. SecuWear supports the creation, evaluation, and analysis of security vulnerability tests on actual hardwares. Extending earlier results, this paper includes an empirical evaluation that demonstrates proof of concept attacks on commercial wearable devices and shows how SecuWear captures the information necessary for identifying such attacks. Also included is a process for releasing attack and mitigation information to the security community.

1. Introduction

Wearables are emerging as the next major ubiquitous computing area. The $9 billion market in 2014 was expected to surge forward to over $30 billion by 2018 and upwards to $70 billion by 2024 [1]. Vast permeation of wearables into the consumer, medical and business electronics markets is coinciding with improved form factors that decrease the size and increase the computational ability and battery life of wearable devices. Wearables operate in *body area networks* using the Bluetooth Low Energy (BLE) protocol to pair with and transmit data to a user's smartphone and often interact with web services on the internet.

Wearable devices come in three varieties: sensors, actuators and hybrids. Wearable *sensors* collect biometrics and/or environmental interaction data, such as pulse rate [2–4], O2 saturation [4], blood glucose levels [4,5], accelerometer data for remote controllers or impact assessments for sport players [2,3,6], exercise metrics (e.g., running distances [2,3,6,7]), visual data [8], and weather information [9] etc. Wearable *actuators* provide application developers and users with feedback mechanisms [6,7], augmented reality capabilities [8], and/or additional User Interfaces (UIs) for software outputs [3,6–8], such as email viewers or media players. Most wearable devices are *hybrids*, containing both sensors and actuators.

There are many challenges with using and applying wearable technologies, e.g. extending battery life, creating rich functionality with

* Corresponding author.
E-mail address: mlhale@unomaha.edu (M.L. Hale).

[1] Matthew L. Hale is an Assistant Professor of Cybersecurity in the School of Interdisciplinary Informatics at the University of Nebraska at Omaha. His research interests lie at the intersection of software engineering and security and include the security of wearables, the internet of things, and web services and the psychology of suspicion (mlhale@unomaha.edu)

[2] Kerolos Lotfy is a graduate student at the University of Nebraska at Omaha. His research interests include cybersecurity, wireless network security, and reverse engineering (klotfy@unomaha.edu)

[3] Rose F. Gamble is the Tandy Professor of Computer Science & Engineering in the Tandy School of Computer Science at the University of Tulsa. Her research interests include web services attack forensics, distributed policy management, and self-adaptive systems (gamble@utulsa.edu)

[4] Charles Walter is a post-graduate researcher in the Tandy School of Computer Science at the University of Tulsa. His research interests are in cyber security and self-adaptive systems. (charlie-walter@utulsa.edu)

[5] Jessica Lin holds a M.S. in Computer Science from the Tandy School of Computer Science at the University of Tulsa. Her research interests are in cyber security and robotics. (suqin-lin@utulsa.edu)

https://doi.org/10.1016/j.dcan.2018.10.009
Received 20 September 2017; Received in revised form 6 July 2018; Accepted 25 October 2018
Available online xxxx

minimal computational resources, minimizing size and breaking other form constraints, and solving problems concerning human-computer interaction. While there are many engineers and researchers focusing on these problems, security is an area often left behind. Despite the large upward trend for the development of wearables, data security and privacy concerns abound [10–12]. Concerns stem from using wearables in insecure environments where body area networks are prone to Man-in-The-Middle (MiTM) attacks such as Bluesnarfing [10], eavesdropping [10] and packet injection [11]. Such attacks allow attackers to co-opt wearables so as to misuse the functionality of actuators and/or steal data from sensors.

Since wearable devices use proprietary hardware and closed APIs, finding some common ground to test, explore and analyze the security of wearables is a difficult task. Researchers seek to explore, exemplify and categorize security vulnerabilities generally for distribution in repositories, such as the National Vulnerability Database (NVD) [13], the Common Vulnerability Exposures and the Common Weakness Enumeration (CVE [14] and CWE [15], respectively), and map these vulnerabilities to specific devices. The goal is to identify vulnerability patterns to prevent future development efforts from repeating existing mistakes and to modify existing wearable applications with effective software/firmware patches.

We previously described a multi-component research platform called SecuWear to take the first steps forward toward addressing this goal [16]. SecuWear is a toolkit that security researchers and application developers can use for testing, analyzing and mitigating vulnerabilities in wearables. It is designed with transparency and openness to give researchers full knowledge and control over the components in the wearable architecture to better understand and assess vulnerabilities. To achieve transparency, SecuWear is built upon five core open source technologies, namely *MetaWear* [17,18] (the device), *Apache Cordova* [19–21] (for rapidly prototyping mobile apps) on top of *Android* [22,23] and iOS [24], *Ubertooth One* [11,25] (for conducting attacks over the air between the smartphone and the wearable), and Django [26] (a web-app framework for logging data from the mobile app).

These technologies span three domains of interest: wearables, mobile devices and web services. SecuWear allows researchers to consider security challenges for wearable applications in each domain individually and across domains. Within the wearable domain, there are questions related to how data is collected and how it is made available to paired mobile devices. In the area of mobile device, there are concerns that data in a certain app may leak out to other less secure apps that may capture and abuse such data (e.g. medical data viewable by a freeware game). In the domain of web service, there are concerns that data may be either attacked, e.g. Cross Site Scripting (XSS) and SQL-injection, or inadvertently reveal personally identifiable information if presented in a poorly aggregated form. In addition, there are concerns over problems between domains. A particular area of interest explored in this paper is the space between the wearable and the mobile app, where the data is transferred using BLE and is open to eavesdropping and other MiTM attacks.

In our prior work, we evaluated the effectiveness of SecuWear platform and solved two core problems. We firstly contrasted the research approach and open source components used by SecuWear against other alternatives and made an evaluation by comparative analysis. The second question looked at the platform's ability to investigate vulnerabilities and capture relevant data needed for analysis. To answer this question, experimentes were conducted and data actually captured were shown and analyzed.

In this paper, we expand our initial research in Ref. [16] by 1) expanding the original study to identify and exemplify additional attack vectors, which are proof of concepts that affect commercial wearable products, and some mitigation suggestions; 2) demonstrating that vulnerabilities associated with the MetaWear component in SecuWear can be generalized to other commercially available consumer wearable products; and 3) providing a template and process for the creation of wearable-centric CVE and CWE entries. For point 2), we specifically capture and analyze packets of several commercial wearable products, including Pebble Watch [6] and Jawbone UP 2 [27], to show that the data structure and transmissions are comparable to (and therefore generalizable from) MetaWear.

The rest of the paper is organized as follows: Section 2 covers the background of wearable applications, mobile security, Bluetooth LE and wearable security. Section 3 overviews the design and components of SecuWear and explores the wearable security in SecuWear. Section 4 juxtaposes the original comparative analysis and empirical study in Ref. [16] with the new extended work on attack vector identification. Section 5 demonstrates the process of applying SecuWear. We show the efficacy of the approach by assessing the results of several commercially available products using SecuWear. Section 6 concludes the paper.

2. Background

2.1. Wearable device applications

Wearable devices and their accompanying smartphone applications include a variety of medical and commercial grade electronics. Fitbit [2] targets fitness, providing a pedometer, accelerometer and other types of fitness tracking sensors. Nike Fuelband series include fitness tracking sensors as well as a heart rate monitor in its Nike + Sportband line [3]. The last major pillar in the fitness wearable market is Jawbone UP [27] which includes motion, sleep and calorie tracking algorithms.

Other commercial grade wearables include Samsung Gear [7], Apple [28] and Pebble Smartwatches [6], etc.. Many of them have the same sensors and tracking capabilities as fitness tracking bands and also offer small touch screen UIs. Smart watch use cases include receiving push notifications and sending user inputs to a controlling smartphone for applications such as mp3 players and email clients. Head-mounted displays are an emerging class of wearables (previously piloted by the Google Glass explorer program [8]). Although discontinued, Glass enabled augmented reality [8] in real time by using video analysis algorithms and UI overlays. Projects like Glass bring about new innovations but also significant security concerns.

The last major type of wearables are medical grade products. These wearables may transmit sensor information to a smartphone app and then the wider internet for the analysis of patient data by medical professionals for home health care purposes such as surgical recovery, geriatrics, diabetes and heart disease. Medical wearables using BLE include: AliveCor ECG [29] that measures electrical activity in a patient's heart and communicates the data to a doctor, the Withings blood pressure monitor [30], and the iHealth pulse oximeter and heart rate monitor [31]. Next generation's pacemakers and insulin pumps may also rely on Bluetooth [5,32]. This prospect raises the bar for security requirements as the price of failure could be death.

2.2. Mobile application security

Reasearch on mobile security falls into several categories. Overviewed here are privacy and policy research [33–35] which focuses on the societal, cultural and individual factors that affect how users trust and use mobile apps; operating system research [35,36] that looks into the capalibilities, vulnerabilities and technical mitigations of the Operating System (OS); and application level research [37,38] that targets the top of the mobile stack to affect processes in web and/or native mobile app development.

In the domain of privacy and policy research, Hurlburt et al. [33] focused on user awareness deficiencies and a range of issues that affect trust decisions when users install new apps. They showed that users are rarely aware of what apps are actually capable of and are increasingly trading conveniences afforded by mobile apps with data and personal privacy protections.

Contrast to this direction, authors of Ref. [34] examined the ways in which application developers can be encouraged to develop apps that

support basic privacy protections. They discussed how the economic pressures and time constraints that encourage app developers to push products to the market can be offset by developments in privacy guidelines, education and platform modifications. Of the three, they suggested that privacy protecting Application Programming Interfaces that force developers to "dig for the more specific privacy-invasive information" are much better than APIs that mix sensitive and public data [34].

Hunt [36] examined OS-level vulnerabilities that could potentially allow for data leakage and/or file stealing by malicious entities or other apps running on a user's smartphone. Specifically, he looked at malware that poses as legitimate software in the google play store, but, when downloaded and executed, runs malicious scripts behind a seemingly legitimate veneer to steal data. One example is a malicious weather app that uses OS level vulnerabilities to bypass access permissions to iterate over the file system to gather and upload the user's photos to a malicious web server.

At the application level, Jain and Shanbhag [38] have identified some big security issues that affect mobile apps, such as client-side injection of malicious content, improper session handling, acceptance of untrusted input, side-channel data leakage, and insecure data storage. Side channels, a problem where one app reads from a space that another app has access to, are particularly relevant to wearables. Secure coding practices were suggested to reduce these problems, including avoiding the storage of data in spaces accessible to other apps, using device-based cryptography, and avoiding OS cut-copy-paste and auto-completion features.

2.3. Bluetooth Low Energy

Bluetooth Low Energy (BLE) [11,39,40], or "Bluetooth Smart", is a scaled down version of the standard Bluetooth protocol designed for communication with small and power-constrained hardware. Largely due to the miniscule power requirements, BLE is the go-to standard for wearable devices to communicate with mobile apps on the higher powered smartphones. BLE is supported by hardwares with Bluetooth running modern OSs (e.g. iOS5+, Android 4.3+, Apple OS X, Windows 8, and most Linux distributions).

Central to the BLE protocol are two concepts: GAP, short for Generic Access Profile [22,40], and GATT, short for Generic Attribute Profile [22, 40]. GAP is an enabling functionality that makes BLE enabled devices, such as wearables, visible to other Bluetooth capable devices. GAP defines two roles for wearables: peripheral, which indicates that the device is small, low powered and resource constrained (i.e., the wearable); and central, which indicates that a device is a more powerful base station, e.g., a smartphone. GAP provides a protocol for pairing [40] a peripheral with a central device. During the process, just as illustrated in Fig. 1, a peripheral will set a specific advertising interval to transmit an advertising packet [40]. If a central device is within the range and interested in the advert, it can send a scan response request [40] to ask for more information or initiate a connection to the peripheral. If additional information is requested, the peripheral may respond with a scan response packet. If a connection is initiated, the two devices will pair following the parameters specified in GAP. This may include cryptographic key formation [11,40] and other handshaking processes [11,40].

Once a connection is established, GATT is used to transfer data back and forth. GATT uses two core concepts: service and characteristics. A service is an object that encapsulates the behavior of a device function using chunks of data called characteristics. Characteristics are objects of the lowest level in GATT. They encapsulate a single variable, such as an array, an integer, or a string. Services are pre-defined formats curated by the Bluetooth GATT Specification [41] for representing different types of device data. For instance, the heart-rate service [41] includes three characteristics: heart-rate-measurement for collecting beats per minute, body-sensor-location for identifying position, and heart-rate-control-point for resetting meta data.

In addition to defining data encapsulation profiles for different types of data, GATT provides a transactional server client protocol [22,40] for transmitting data between the peripheral (server) and central (client) devices. GATT transactions [22,40] communicate service information following a Master (client) to Slave (server) request/response cycle. In addition, a slave (server) can asynchronously notify the GATT client when events occur on the device. This process is shown in Fig. 2.

2.4. State of wearable security and privacy

There are a number of security and privacy conerns with body area networks of devices communicating over BLE channels [10–12]. These concerns can be generally categorized as related to man-in-the-middle enabling vulnerabilities [10,11], privacy and visibility of data to unauthorized apps or users [12], and disclosure of information by apps with poorly configured policies [42].

Ryan [11] highlighted some profound security shortcomings of BLE transmissions. In particular, Ryan developed a BLE sniffing device capable of following a GATT communication process to reveal the contents of transmitted master/slave packets. The device, called Ubertooth One [25,43], is open sourced and comes with a toolset for following connections. He demonstrated that the sniffer can passively eavesdrop or actively inject packets into BLE communications on unencrypted channels. For encrypted channels (default on most device pairings), Ryan was able to make use of a protocol weakness in the BLE key exchange process to bypass encryption. This means that if the sniffer is able to observe the key exchange process during device pairing, then it can passively eavesdrop or actively inject packets into the encrypted BLE channels as well. This indicates the need for policies that ensure all BLE communications are encrypted and to disable pairing in untrusted environments.

In a report by Cyr, Horn, Miao, and Specter [44] from MIT, Ubertooth One was applied to analyze the security features of a Fitbit [2]. They showed that Fitbit uses a proprietary encryption process different from the standard investigated by Ryan in Ref. [11], but still exposes credentials in the clear over BLE during pairing.

3. The SecuWear platform

Combatting security and privacy risks of wearables is an emergent concern. A core motivating design principle of the SecuWear platform is to provide security researchers the ability to step back from particular wearable devices or apps and to conceptualize and investigate vulnerabilities more generally in terms of how and where they present themselves in the multi-domain workflow. SecuWear provides a generalized application testing workflow that supports security investigations spanning three domains and the communication channels between them. The

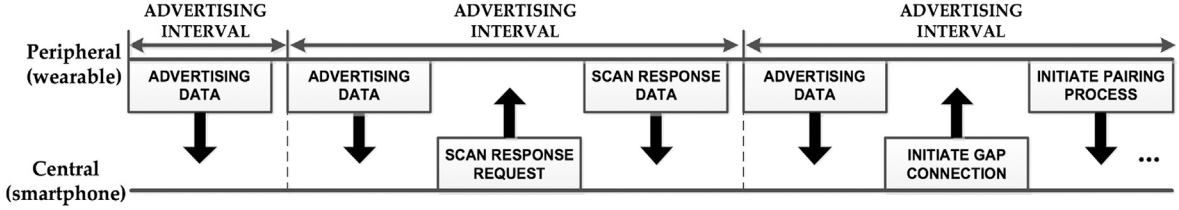

Fig. 1. GAP Bluetooth LE advertising process showing advertising data, scan request, scan response, and gap initiation packets.

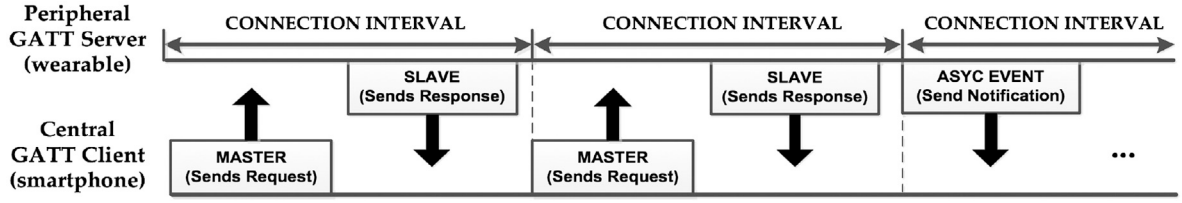

Fig. 2. GATT Protocol showing transactions and intermixed asynchronous event notifications. This process begins after initial pairing is established using the process in Fig. 1.

approach is vendor and product agnostic, making it suitable to a variety of IoT and wearable products. The supported domains are wearable (the software and hardware on the actual wearable device), mobile (the smartphone or tablet and their applications communicating with the wearable), and web (online components that connect and communicate with the mobile domain).

Testing efforts in SecuWear are separable by domain, as shown in Fig. 3, to help researchers isolate problems to particular domains and generalize discovered vulnerabilities to other commercially available products by category. For instance, if a vulnerability or weakness is found in the mobile domain, researchers can publish their results, allowing security engineers at other organizations to determine how relevant the vulnerability or weakness is for their own products. Thus, the guiding principle behind SecuWear is to enable core understanding and analysis of *common weaknesses* of wearables across three domains. This effort is in line with efforts to enumerate and catalog weaknesses and vulnerabilities, e.g. CWE [15] and CVE [14]. Given these goals, the inner workings of the SecuWear platform must be transparent so that researchers can clearly understand how different types of attacks affect different domains.

Addressing this requirement, SecuWear uses five core open-source technologies: Metawear [17,18] (the wearable device), Apache Cordova [19–21] on top of Android [22,23] and iOS [24] (for building a prototypical mobile app), Ubertooth One [11,25] (for conducting attacks over the air between the mobile and wearable domains), and Django [26] (a web-app framework for logging data collected from the mobile app to the web domain). In the spirit of openness, SecuWear will also be open source. Fig. 3 shows the domains (wearable, mobile, web), the communication channels between them, and the attack vectors targeting each. On the left is the wearable domain provided by MetaWear, in the middle is the SecuWear mobile app and on the right is the Django web app. The sections below cover each component in detail, providing specifications and motivations for their usage. Also covered is Ubertooth one, which can sniff BLE data packets between the wearable and mobile domains.

3.1. MetaWear

MetaWear [17,18] is an open-source wearable hardware and software package developed by MbientLab. MetaWear offers a tiny form factor that is roughly the size of a US Quarter with a weight of approximately 4 g including the battery. The MetaWear chip packs a tiny MCU ARM Cortext-M0 processor [18] for handling small instruction sets, and a variety of sensors including: an ultra-bright LED, a temperature sensor, a three-axis accelerometer, a push button, and extensible pins for additional GPIO (General Purpose Input/Output) sensors or devices which can be anything from optical pulse readers to miniature vibrating motors for haptic feedback [18]. Also on the board is a USB charger and a Bluetooth LE antenna. A reference diagram from Ref. [18] is shown in Fig. 4.

MetaWear provides developers and companies with a scalable turnkey hardware platform for creating quick-to-market wearables. Included in the hardware is an onboard firmware package that makes the various sensor data available to smartphones and tablets via an open BLE API available in Android and iOS [17].

SecuWear greatly benefits from both the feature richness and

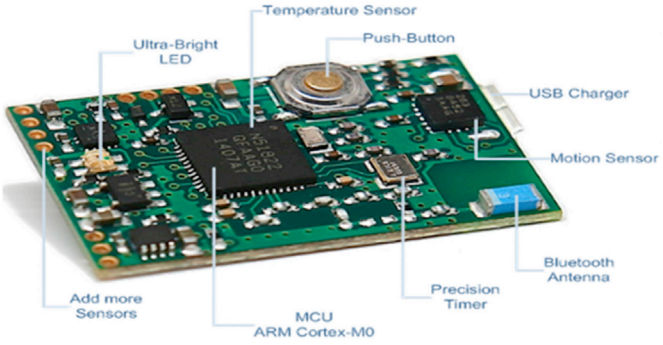

Fig. 4. Annotated MetaWear hardware diagram from Ref. [18].

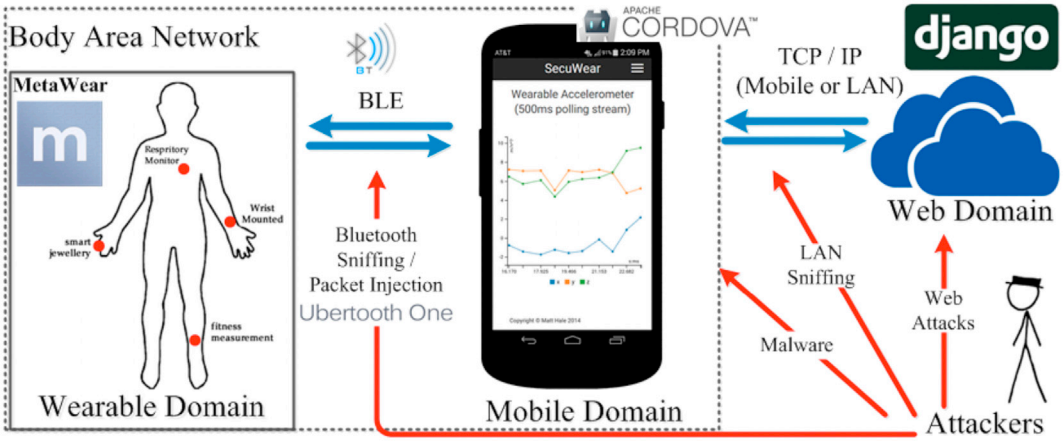

Fig. 3. Domains and open source technologies in SecuWear include the Wearable, Mobile, and Web Domains and the areas between them.

openness of MetaWear. The features allow security researchers to study a variety of different general use cases often seen in commercial and medical wearable applications. The openness, through code release and good documentation, makes the inner workings of wearable applications transparent, a quality that is important in vulnerability assessment and analysis.

3.2. SecuWear mobile app built with Cordova

Acting as a base station for Bluetooth LE wearables to communicate with, mobile apps are often central to many wearable applications. Shown at the center of Fig. 3, the mobile app in SecuWear is built with another open source technology called Apache Cordova [19–21]. Cordova is a framework for building *hybrid mobile apps* (or simply hybrid apps). The beauty of hybrid apps is that they are cross-platform, meaning that they are built only once and then ported to different smartphone platforms (such as Android or iOS) with relative ease. Hybrid apps are built as web applications using HMTL5, CSS, and JavaScript. These web applications are then wrapped in platform-specific code using *thin native containers* that provide all of the device-specific functionality exposed to native apps (such as Geolocation, accelerometer and Bluetooth capabilities, to name a few). The thin native container wrapping process is supported in Cordova for both iOS and Android. It works by using pre-built open-source libraries in objective C (iOS) and Java for Android that provide off-the-shelf application shells with native drivers. In the shell, the hybrid app core (i.e. the developer-written web app) is executed as a WebView (a type of process that runs in both iOS and Android) to run web application code.

Choosing Cordova for SecuWear allows security researchers to build their apps in well-understood web languages and then port them to different platforms to explore how vulnerabilities may present themselves differently across different hardwares and operating systems. To make the SecuWear mobile app work with MetaWear, we have developed modular plugins following established Cordova plugin development standards [19,20] to map the Bluetooth API in MetaWear's Android and iOS interfaces to web apps built in Cordova. This allows security researchers to get first-class object handles on MetaWear's sensors using JavaScript, e.g. *metawear accelerometer*. When completed, these libraries will be open sourced for the benefit of security and MetaWear communities.

In addition to the Cordova plugins developed specifically for exposing the MetaWear API, SecuWear currently supports custom Bluetooth characteristic calls (see Ref. [41]) that allow security researchers to directly form and send raw data using the Bluetooth adapter on smartphones. This functionality is supported by an open-source Cordova plugin available at Ref. [45] and affords researchers a great deal of control that can be used to craft customized packets to help reveal certain types of vulnerabilities or aid in specific assessments. Finally, SecuWear integrates all of the official Apache Cordova plugins [19] that provide access to other device features such as the phones geolocation functionality, accelerometer, file system, camera, etc. Thus, SecuWear can store wearable data in a phone's file storage, allowing side channels (such as another app reading the storage) to be examined. This provides the SecuWear app with access to the Bluetooth log (for Android devices). These components are included in the SecuWear mobile app architecture diagram shown in Fig. 5.

Here the web app core is provided with first-class objects functions and data implemented by the Cordova plugins through the rendering engine. The plugins map to underlying mobile OS-specific libraries (e.g. Android and iOS APIs). The rendering engine wraps the web app into a native executable runnable on the mobile OS.

3.3. Ubertooth One

Whether in this wearable domain or in more traditional environments such as web servers, vulnerability testing requires an understanding of the network that applications are running in. In wearable applications, where data is sent over the air through BLE, there is the potential for an attacker to listen to specific segments of the Radio Frequency (RF) spectrum and sniff (i.e. spy on) data exchanges. Ubertooth One [11,25] is an open source hardware component and software package that provides spectrum analysis of the 2.4 GHz ISM (industrial, scientific and medical) radio band in a USB dongle. In practical terms, it is a Bluetooth antenna that functions like a Wi-Fi spectrum analysis tool (e.g. Wi-Spy [46]) that sniffs IP traffic.

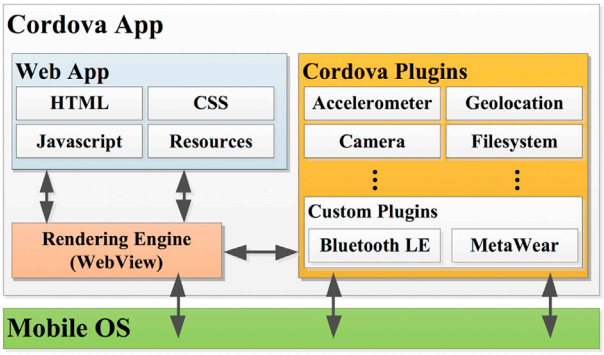

Fig. 5. SecuWear mobile app architecture.

SecuWear requires the use of an Ubertooth within the range of the SecuWear mobile app and the MetaWear hardware. A Ubertooth is connected to a *-nix based system and is configured, following the instructions in Ref. [43], to capture packets into PCAP (Packet Capture) files that can be analyzed in Wireshark. Ubertooth allows security researchers to explore vulnerabilities with the channel between mobile apps and wearables (e.g. eavesdropping) and to conduct active MiTM attacks through packet insertion.

3.4. Django and the web service API

The last component of the SecuWear architecture is an online web service that exposes a RESTful API to allow data from the SecuWear mobile app and PCAPs from Ubertooth to be logged online for viewing and/or analysis. The SecuWear web service uses Django [26], a python-based web application framework, its out-of-the-box admin capabilities, and a third party open source plugin called *Django REST Framework* [48]. The data accepted by the simple API is shown below in JSON (JavaScript Object Notation) format in Fig. 6. The *experiment_id* is a foreign key to an experiment object that groups events and collects PCAPs related to a single test. The other fields are self-explanatory by name. All fields are captured as strings. In Fig. 7, some captured data is shown as viewable in the Django admin panel.

3.5. Exploring wearable security

Many security problems exist in a multi-domain wearable application. This section highlights some interesting data that can be collected by SecuWear in each domain and discusses how it can be used to analyze and investigate security vulnerabilities. A particular focus is placed on the space between the wearable device and the mobile app. Research by Zhou and Chao [49] has called attention to the types of data generated

```
POST /api/event/
{ id, type, data, datetime, device, os, os version,
  experiment id }
POST /api/pcap/
{ id, data, experiment id }[47]
```

Fig. 6. Simple SecuWear JSON API for logging data.

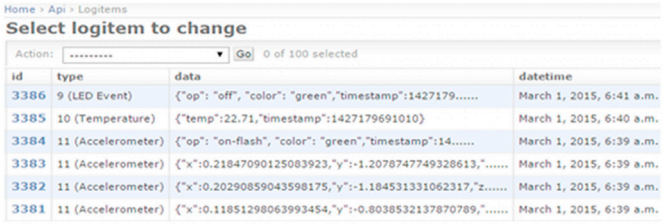

Fig. 7. Sample of collected event data in admin panel.

and processed by different domains in the internet of things. They identified sensor nodes, transmission systems, and service ecosystems as particular areas of interest in common IoT applications. In separate work [50], they also identified performance requirements and scheduling schemes to convey security data of interest to the forensic or analysis platform in use by applications.

SecuWear can capture many types of data such as those identified in Ref. [49] and uses in-application audit hooks to do so, helping developers to have control over the performance of the security data capture elements within their applications. First, the SecuWear mobile app works with the MetaWear API to gather *streaming wearable events*. These events are temporarily stored locally within the SecuWear app before being forwarded on to the SecuWear RESTful web service. Sample events collected via this method are shown in the web service admin panel in Fig. 7. Each event includes the fields in the JSON API from Fig. 6.

In addition to capturing wearable data through this typical application workflow, SecuWear can also capture individual packets sent between the wearable and the mobile app in two different ways. First, if an Android device running Android 4.4+ (Kit Kat) is used, then SecuWear can grab Bluetooth packets from the btsnoop_hci log [23] and send them to the web service as a PCAP file. Second, the main form of capturing involves over-the-air collection using the Ubertooth One component. Ubertooth allows researchers to conduct eavesdropping attacks to analyze the 2.4 GHz Bluetooth spectrum that wearables operate in, examine the packets exchanged during the advertising and pairing processes (shown in Figs. 1 and 2), observe actual data exchanged after pairing, and even perform packet insertion. Fig. 8 shows the state of the Bluetooth spectrum during data exchange. The ubertooth-specan-ui package [43] was used to capture this data. In the next section, Fig. 11 shows actual packets captured during the same time interval.

These different data collections, coupled with the complete control that security researchers have over each SecuWear component, facilitate the investigation of many security vulnerabilities ranging from MiTM attacks between the device and the mobile app, to application vulnerabilities in the wearable, mobile app and web service.

4. Evaluation of the platform

4.1. Comparative component analysis

It is important to critically examine the strengths and weaknesses of using SecuWear compared to other possible options for exploring wearable security. SecuWear seeks to unify different domains in the application of wearables and represents them in a generic way that allows researchers to generalize vulnerability assessment findings. This approach stands in contrast to many vulnerability assessment strategies [51] that emphasize targeting a specific hardware or software, but aligns with efforts to categorize and classify common security issues in CVE [14] and CWE [15]. The list below provides a look at the relative advantages and disadvantages of the SecuWear approach compared to hacking at specific hardware or software components.

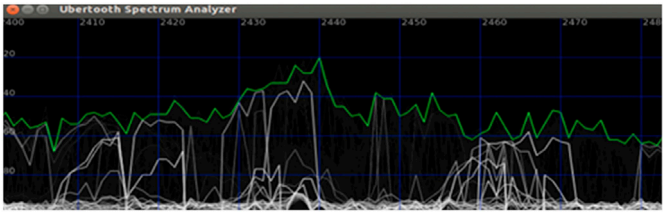

Fig. 8. Spectrum analysis in the 2.4 GHz range during the time when the events in Fig. 7 were captured.

4.2. Advantages

+ It facilitates assessment of vulnerabilities that span different domains.
+ Its results describe categorical (common) issues that may affect multiple hardware or software products.
+ Components are open source and interactions are transparent.
+ It provides multiple data collection streams for a multi-factor analysis and to report to isolate identified issues.

4.3. Disadvantages

- Wearable vendors may be less likely to accept results as their products are not specifically targeted.
- Identified vulnerabilities still need to be investigated against commercial products to determine if they apply.
- Vulnerabilities may be specific to selected open-source components, resulting in false positives when identifying security issues as common problems.

Given these disadvantages, it is important to understand that the role of SecuWear is not to replace specific product assessments, but to act as a first step towards identifying and categorizing wearable security vulnerabilities. The vision is that researchers can examine problems generally with SecuWear and distribute results via a CVE/CWE-like method, and then vendors and other researchers can follow up with additional product-specific assessments to better classify and contextualize the information to their products.

Table 1 weighs the pros and cons of using the selected components in SecuWear against competing alternatives. The range of options available in the wearable and mobile SecuWear domains are listed as columns and each feature is color coded from dark brown (bad) to light brown (decent) to green (good). Variant cells are simply shaded white. Since the web service just provides a REST API, competing options are not analyzed closely. Wearable products examined include: Adafruits Flora [52], an Arduino-compatible electronics board for GPIO functionality; WARP [53], a small IoT and wearable app development platform; and typical consumer commercial products (Comm. Products for short). Typical consumer products lumped together in the table include: Apple Watch [28], Fitbit [2], Pebble [6], Jawbone UP [27], Samsung Gear [7], and Nike + Sportbands [3]. Different mobile app development technologies are also compared below according to the desirability of their features.

A firmware is considered *programmable* if developers are given access to the firmware source code by the device manufacturer; *closed, customizable* if device manufacturers are willing to provide alternative firmware per developer request; but the source is *closed* if its source code is not accessible and device manufacturers are unwilling to customize the firmware for alternative uses aside from those provided by the release. *Community size* is estimated according to the number of mobile app users on Google Play and Apple App stores for each type of device and the number of contributors/downloads/activities on their GitHub repositories, if available. By the time of writing, the mobile app used by Flora, the Adafruit Bluefruit LE connect app, has approximately 10 thousand downloads (note that the app is not exclusive to Flora) on the Play Store and an unknown amount on iOS. MetaWear has approximately 8 thousand across GitHub, Play store and App store and has active source

Table 1
Comparative analysis of SecuWear Components.

Wearable	MetaWear	Flora [52]	WARP [53]	Comm. Products
Firmware	Closed, Customizable	programmable	Closed	Closed
Community	Medium	Medium	Small	Large
Bluetooth	BLE	Planned	BLE	Varies
Simplicity	Easy	Easy	N/A	Varies
# of Sensors	High	High	Medium	Varies
Open source	Yes	Yes	Planned	No
Platforms	iOS, Android	Arduino	All	Single
Cost	$39	$20 + $70 (sensors)	$149	$100+

Mobile App	Cordova	iOS	Android	Web
Cross Platform	Yes	No	No	Yes
Device Independent	Yes	No	No	Yes
Development Time	Medium	High	High	Low
Performance	Moderate	High	High	Slow
Feature Access	All, requires plugins	All	All	Few

code repositories. We could not determine app counts for the Warp at the time of writing and there was little or no activity on GitHub. Commercial wearables ranged from millions of downloads (e.g. 10 million for Fitbit on Play) to hundreds or thousands depending on the product. Simplicity refers to the ease with which developers can use and write code for their apps using the product. Flora and Metawear are easy to use given their well-documented APIs and modular design, whereas commercial products are less simple, depending on the device because of no or little documentation. The *number of sensors* is considered *High* if the device possesses 4 or more sensors, *Medium* if it contains 2 or 3 sensors, and *Low* if it only contains 1 sensor. Commercial products vary widely, with most having medium or high numbers of sensors. The comparisons of Cordova, iOS, Android, and Web technologies are based on prior work in Refs. [16, 47,54].

4.4. Studying attack vector types in SecuWear

This section reports the findings of two empirical studies on attack vectors that can be identified and mitigated in SecuWear. The first study, conducted initially in Ref. [16] and extended in this work, demonstrates that eavesdropping attacks affect three different types of wearable use cases. The second study conducted firstly in this work exemplifies a denial of service attack that can affect wearable devices and demonstrates a mobile app memory attack.

While the attack vectors examined in the studies are interesting in their own right, the goal of both studies is to show that SecuWear has the logging functionality and multi-domain perspective necessary to capture and analyze attack data for use during vulnerability assessment and mitigation strategy synthesis. Below we summarize the attacks investigated in both studies and use these results to enumerate the other types of attacks that can and cannot be investigated in SecuWear.

4.4.1. Eavesdropping attacks on bluetooth

In Ref. [16] we investigated passive eavesdropping attacks in three use case scenarios commonly seen in all wearables that use Bluetooth for wireless communication. The first scenario involves the wearable advertising its availability to BLE enabled smartphones. The second involves the wearable connecting and pairing with an app on a mobile device. The third examines actual BLE GATT data having passed from the wearable to a smartphone. Each scenario, the resulting data captured by SecuWear, along with the new mitigation techniques added in this paper are discussed below. In each case, packets were captured using the *ubertooth-btle f c* command to follow (f) connections and dump (c) data to a FIFO (First In, First Out) pipe. Using this approach, data flows directly into Wireshark in real time, allowing researchers to use Wireshark filters and plugins, such as libbtbb [43], to analyze the data.

The first scenario (advertising) begins when the MetaWear component is powered on and connectable, but not paired. During this time, it broadcasts advertising packets constantly, following the process described in Fig. 1. Consistent with expectations based on results from Ref. [11], Ubertooth was easily able to capture the advertising packets. Fig. 9 shows an advertising packet (type ADV_IND) that includes the MAC address and device name of the wearable. Since ADV_IND packets are broadcasted, this type of capture is not very malicious.

Clearly, SecuWear is able to capture relevant attack data for assessment in this case. In terms of mitigations, there is no compelling reason to counter this type of eavesdropping, for this type of observation is intended by the wearable to broadcast its advertising packet. If one does want to prevent announcements, mitigation would entail: a) pairing the wearable before entering a publicly accessible wireless space (e.g. pairing the device at home); b) putting the device in a faraday cage to prevent all outside communications; or, c) shutting the device off.

The second use case (pairing) involves a smartphone running the SecuWear mobile app and pairing with MetaWear. In this use case, the mobile app gets the advertising packets broadcasted in the first use case and then sends a scan request (SCAN_REQ) packet to ask for more information, such as its Bluetooth address and device clock. When MetaWear gets the request, it sends back a scan response (SCAN_RES) packet using the same channel. Next, the user selects “Connect” in the mobile app. This generates a connection request (CONNECT_REQ) packet which identifies the pairing information, such as hop patterns and data channels to use.

SecuWear, using Ubertooth, captures all three types of packets as shown in Fig. 10. It also captures data (not shown) across the mobile app (Bluetooth log) and the wearable (event data log). This data provides a three-perspective audit log of events allowing researchers to understand the attack from the originator, destination, and attacker perspectives. This particular attack is interesting because given a CONNECT_REQ packet, the attacker can then follow the pairing process and continue to sniff data even during channel hopping (as shown in the next use case scenario). This allows the attacker to follow data sessions and makes this attack particularly harmful.

When pairing, there are three types of Secure Simple Pairing

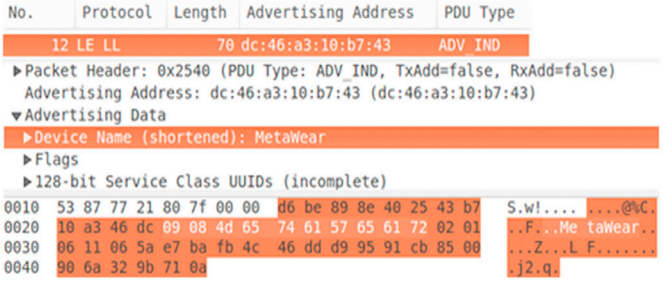

Fig. 9. Packet captured during GAP advertising process.

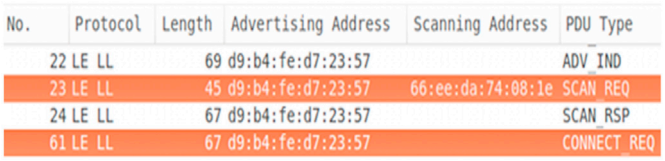

Fig. 10. Packets captured during the pairing process.

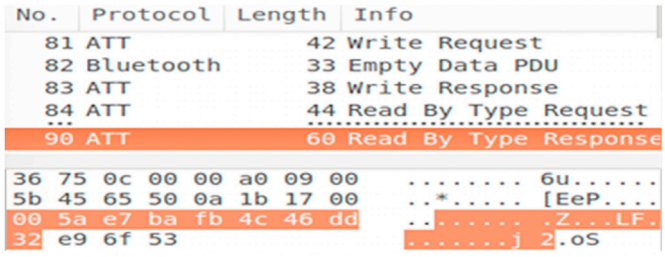

Fig. 11. Packets turning on thermometer and sending data.

strategies (SSPs) [55]. The first, used by MetaWear and a majority of wearable devices, is called *just works pairing*. As its name suggests, this pairing strategy does not require any user interaction and does not provide any MiTM protection. The other two types are called *numerical comparison* and *passkey entry*. They both require a wearable device to have a display. In the former type, both the wearable and the mobile phone generate a 6 digit numeric code and the two are compared before the user pairs the device. In the latter case, one of the two devices generates a code and the user must enter it on the other device. Both pairing strategies claim that they provide MitM protections.

In the original study in Ref. [16], we've only examined just works pairing. In this work we extend this by examining other two SSPs. Since the Bluetooth specs [40,41,56] claim that they provide MiTM protections, we expect that the connection packets would not be discoverable. However, surprisingly, our results show that passive eavesdropping attacks attempting to identify the connection packet are NOT limited to *just works* pairing as suggested by the Bluetooth LE documentation. In fact, we have captured the CONNECT_REQ packets for both Fitbit [2] and Pebble Watch [6] as shown later in Fig. 17 and Fig. 19 respectively. Pebble uses numerical comparison, while Fitbit uses passkey entry.

Given these findings, it is clear that mitigating this type of attacks requires more than simply using an alternative SSP strategy. Existing mitigation approaches focus not on preventing the attacker from viewing the connection packet (and by extension the data packets) but instead on encrypting the data packets to prevent the attacker from actually viewing the contents. This approach can be thwarted if attackers have the CONNECT_REQ, by reverse engineering the encryption method and decrypting the data packets that follow successful pairing. In fact, Mike Ryan has demonstrated such an attack on the core Bluetooth encryption capabilities [11]. For these reasons, encrypting packets after the initial connection is not sufficient. Our results suggest that the Bluetooth LE protocol itself needs to be reexamined for alternative SSP strategies that do not involve in-the-clear CONNECT_REQ packets.

The last eavesdropping attack (data transfer) we examined involves an attacker intercepting and viewing data transmissions after successfully capturing the CONNECT_REQ. In this use case, an end user is sending signals to a wearable (via a paired mobile app) to evoke functionality, or the wearable is capturing data and sending it to the paired mobile app for viewing. Since functions and data on wearables could be life critical, such as in medical applications, it is vitally important that the communication channels are not visible or tamperable.

To examine this, MetaWear is paired with the SecuWear app and several functions are turned on (LED, accelerometer, temperature sensor). These events are logged to the app and then to the web service, as shown previously in Figs. 6 and 7. Fig. 11 shows the actual packets signaling MetaWear to read and send temperature data back to the paired device. Packet 81 specifically sends an opcode using a Write Request to turn on the thermometer. After acknowledging the request in packet 83 and receiving a subsequent *read request* for data (packet 84), MetaWear sends back the temperature data in a *Read by Type* response in packet 90. This data can be decoded by anyone who happens to have seen the connection requests or is listening to the channel when the data is being transmitted on. SecuWear includes, again, three different types of information for security analysts to reason over and to understand the specifics of the attack.

To mitigate this type of passive eavesdropping, vendors should use either numerical comparison or passkey entry SSP techniques and then encrypt all critical or personally identifiable data transmitted after pairing. Currently (as of the time of writing) the standard encryption technique of Bluetooth LE is not secure, as shown by Ryan [11]. For this reason, vendors such as Fitbit [2] are tending to use proprietary encryption techniques. But this is not a feasible or advisable solution in long term. Thus, we suggest that open source modification, such as those proposed by Ref. [11], could be adopted into the Bluetooth LE standard.

In addition to the eavesdropping attacks shown above, attackers could, using the same mechanisms demonstrated here, spoof themselves and inject malicious packets in either direction. This means that an attacker could take control of a wearable and do many bad things, including invoking functionality to cause error states or injecting malicious code/data into the mobile app or the wearable's firmware. In any of these cases, SecuWear offers a valuable set of integrated components to test security vulnerabilities, perform assessments and aid mitigation efforts.

4.4.2. Denial of service attacks on bluetooth

Another type of attack we have explored in addition to eavesdropping is Denial of Service (DOS). We examined two specific types of DOS. The first simply involves spamming connection requests to any advertising device in range to prevent other (legitimate) apps from pairing with the device. The second (more interesting) type involves spamming devices that allow multiple connections until their connection limit is exceeded.

In the first case, we find that a malicious entity with a USB Bluetooth adapter and a python script rapidly scanning and connecting to any device in range could effectively prevent a user from connecting to their device. Fig. 12 shows the connection (as captured by SecuWear) between the intercepting computer and the passive Bluetooth device, a pair of LG HBS800 headphones. The first two lines are beacon packets sent by the headphones, followed by the initial pair of connection request packets

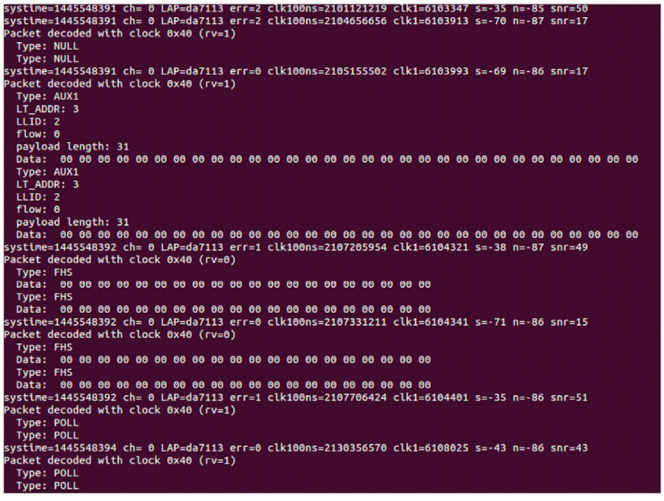

Fig. 12. SecuWear packet capture showing a denial of service attack executed from a python script.

(AUX1). This is followed by the handshake, linking the hop pattern in the Frequency Hop Synchronization (FHS) packets. Once the hop pattern is synchronized, normal operation of the device can begin. At this point an attacker can either stay connected or disconnect and reconnect again.

This type of attack is also of concern for Bluetooth devices that receive firmware updates from their apps. Using this attack, an attacker can prevent legitimate usage and coopt the device, or upload malware-laced firmware after the attacker has connected to the device. Many commercial products, including all of the major fitness trackers (Fitbit, Jawbone UP2, Pebble, etc), accept firmware updates via Bluetooth, usually through their mobile apps.

The second type of attack we've investigated targets at Bluetooth devices that support pairing with multiple apps at once. In this case, an attacker using the single connection approach in the first case can connect once, but is unable to stop the original device from connecting. These types of Bluetooth devices generally support a maximum of 7 active connections (due to the limitation of a 3bit address space) in a single piconet.

While we are able to modify the script used in the first attack to create 7 simultaneous connections, none of the wearables we test it on (Pebble, Jawbone UP2, MetaWear, and Fitbit) support multiple simultaneous connections. Therefore, it is unclear whether or not this type of attack could prevent legitimate usage of a multi-connecting device. Overall, Bluetooth DOSsing is feasible and SecuWear is able to capture relevant information to identify such attacks.

4.5. Limitations of SecuWear investigaive capacity

Up to this point, specific examples of attacks detectable by SecuWear have been shown. This section takes a step back to summarize the domain areas that are investigable by SecuWear and to define the types of attacks that cannot be captured by SecuWear. Going back to Fig. 3, we identify three large domains (the wearable, the mobile app, and the web) and the areas between them (Bluetooth, Wi-Fi/4G). The attacks shown above focus on the wearable and the Bluetooth domains. SecuWear is particularly well suited to identifying these types of attacks because of the level of event logging on the wearable device (MetaWear), the collection capability of the Bluetooth Antenna (Ubertooth), and the built-in logging framework in the SecuWear mobile app.

The limitations of SecuWear start to arise on the paired mobile apps. SecuWear only has access to data in its app and the shared file space on the mobile OS (Android or iOS). Based on this fact, SecuWear can identify attacks targeting shared memory exploitation (for instance, a malicious mobile app is accessing memory that the SecuWear app is using in the clear). However, SecuWear cannot look into the protected file or memory spaces of other apps (that may be malicious). This means that third party malicious apps that connect with paired Bluetooth devices are not entirely logged by SecuWear in the mobile domain.

SecuWear also provides no coverage for attacks targeting the 4G domain. It is illegal in the US to monitor the frequency range involved in wireless cellular communications, so SecuWear cannot do so either for legal reasons. SecuWear is also not currently equipped with a Wi-Fi monitoring component to examine attacks between the mobile app and the web domain over Wi-Fi. Since SecuWear is primarily focused on the wearable and its mobile app, it is also not currently equipped for examining web attacks on the web domain that connects with the mobile app as shown in Fig. 3. While one can certainly extend the SecuWear logging infrastructure to examine such attacks, it is beyond the scope of concern in SecuWear and covered much better by many other web vulnerability and forensics analysis frameworks such as w3af [57], Metasploit [58], sqlmap [59], and others. Table 2 summarizes the coverages and coverage limitations discussed above.

Table 2
Coverage and Limitations of SecuWear Investigative Capacity by Attack Vector Type on each Domain.

Domain	Eavesdropping	DOS	Injection	Overflow
Wearable	Medium	N/A	High	Medium
Bluetooth	High	High	High	N/A
Mobile	Medium	Low	Low	Low
Network: 4G/Wifi	None	None	None	N/A
Web	out of Scope	–	–	–

5. SecuWear in practice

How do the components in SecuWear compare to commercial products in the real world and how are the results useful in practice? After all, if the investigations conducted in SecuWear to explore attack vectors and identify vulnerabilities are not representative of any commercial wearable products, then the efforts are futile and in vain. We address these questions head on to show that a) there is a path from SecuWear analysis to safer end users; and b) vulnerabilities found in SecuWear can also be applied to commercially available products, including Fitbit [2], Pebble Steel [6], and Jawbone UP2 [27], given the structural similarities between the interoperation of components in SecuWear and those of the real world products.

5.1. From lab vulnerability discovery to end-user

Fig. 13 addresses claim a) by laying out a 5-step iterative process that 1) discovers and catalogs wearable vulnerabilities (in one or more domains), 2) investigates and tests mitigations to combat them, 3) disseminates findings as CVEs to vendors, 4) encourages vendors to test their individual products (using SecuWear or internal testing tools), 5) ends with patches being deployed to make end users safer. In the evaluation section (Section 4), we've showed a few example attack investigations and possible mitigations that cover steps 1 and 2.

Step 3 in the 5-step process involves following the MITREs process to create and report CVEs. Since SecuWear users might include independent testing firms, other researchers in academia and firms with commercially or internally available wearable devices, there are two processes involved in creating and reporting a new CVE.

In general, CVEs can only be created by an authorized CVE Numbering Authority (CNA). Hence, SecuWear users operating in a firm with CNA status can create and submit a CVE internally. CNAs include well established firms such as IBM, Microsoft, Apple HP, Google, and large open source product makers such as Ubuntu and Debian GNU.

For most SecuWear users, step 3 means contacting an authorized third-party coordinating CNA in the form of a CERT (Computer Emergency Response Team). There are four options for this: the original CERT/CC [60] that begins as a part of CMUs Software Engineering Institute, US-CERT [61] which coordinates with CERT/CC on vulnerabilities affecting the national security of the United States, ICS-CERT [62] which is primarily concerned with industrial control systems in the US, and JP-CERT/CC [63] which is Japan's analog to CERT/CC. At this point, the choice of a CERT will handle the dissemination and vendors can choose to follow steps 4–6.

5.2. Generalizing SecuWear vulnerabilities

To demonstrate, at least by way of example, that vulnerabilities identified in SecuWear can also be found in commercially available wearable products, we've evaluated a representative sample of products from three different vendors (Fitbit Charge HR, Pebble Steel and Jawbone UP2) to see if they also exhibite the same vulnerabilities described in Section 4. The answer is a resounding yes as we show in the packet captures and collected data below for the devices tested. Note that in all tests we've used the listed device (one of the three above) with its associated mobile app (from the Android App store) and online API to capture data analogous to that of what SecuWear provides. In some cases such as Fitbit and Jawbone, the API does not provide any data on the

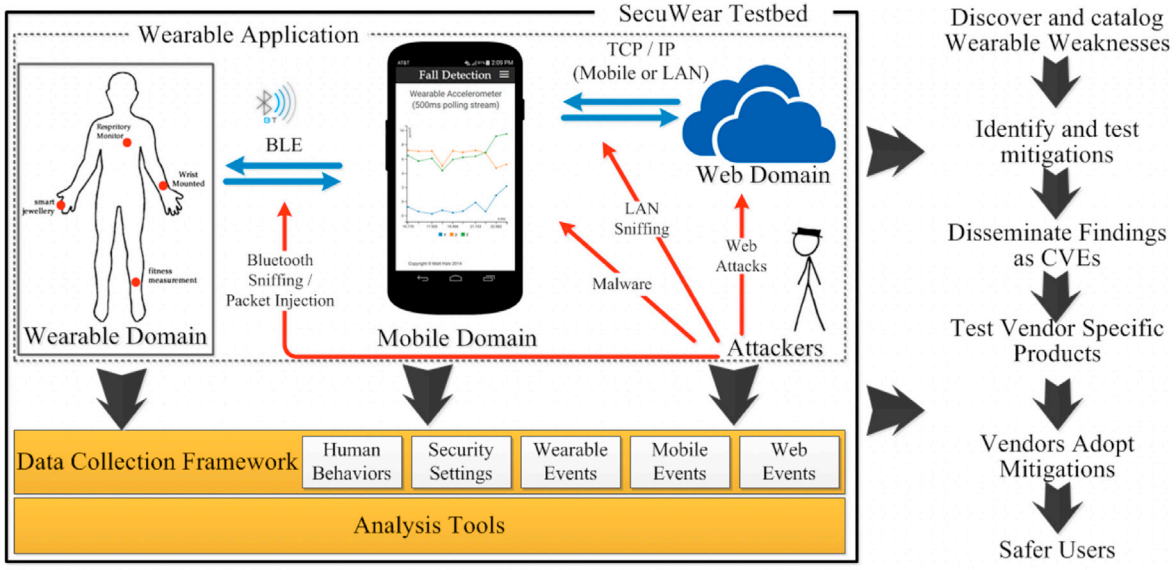

Fig. 13. Overall process for applying SecuWear in practice. This process includes discovering and cataloging wearable weaknesses.

operation of the wearable device, but instead only the data captured. To capture the over-the-air Bluetooth data, we've also used an Ubertooth.

5.2.1. *Similarities of advertising use cases*

All of the three devices tested exhibite advertising packets that look very similar to those produced by MetaWear operating as part of the SecuWear platform. Fig. 14 shows a SCAN_REQ and ADV_IND packet from a Fitbit Charge HR that closely resembles the structure of those sent from MetaWear as shown in Fig. 9. Similarly, Fig. 15 and Fig. 16 show the same types of packets as captured from Jawbone UP 2 and Pebble Steel, respectively. Note that the UP and Fitbit place their info in a SCAN_RSP (i.e. the user must request more information about the ADV_IND) whereas both MetaWear and Pebble place their information directly in an ADV_IND packet, which avoids the scan request/response steps.

From the three captures, one can see that an attacker's ability to observe the advertising packet of each of the devices is the same as what was discovered by MetaWear in SecuWear. The only difference is that in SecuWear a researcher has more information (specifically the information from the device and mobile app perspectives) that is not available on the commercial products. This fact again reinforces the motivating goals behind SecuWear, namely, to explore attack vectors and then let vendors and other third parties (like a CERT) deal with specific commercial products.

5.2.2. *Similarities of pairing use cases*

Just like the first use case, we've examined three products to see how closely the next much more important (from the perspective of security

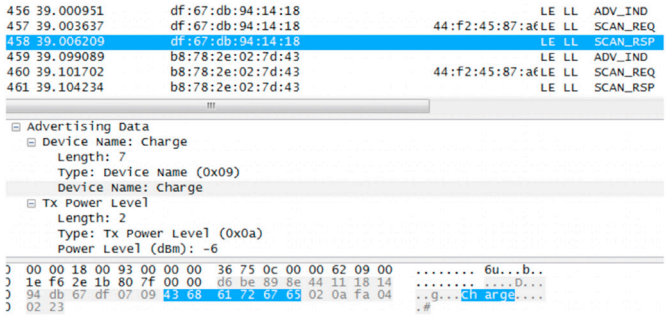

Fig. 14. Advertising packets from Fitbit Charge HR showing very close similarities with MetaWear capture.

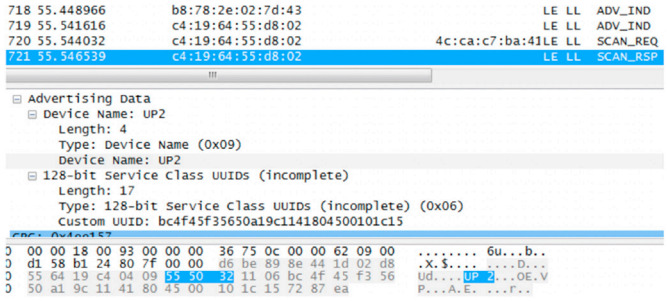

Fig. 15. Same type of packet captured from Jawbone UP 2.

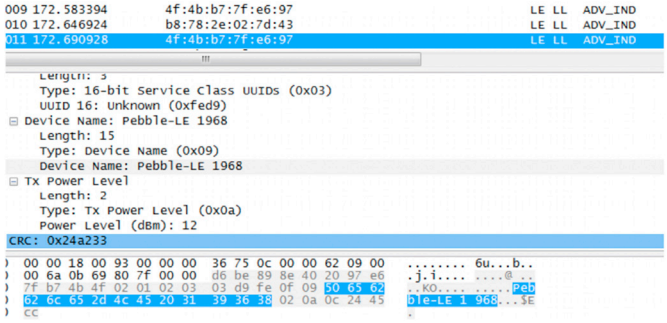

Fig. 16. Similar ADV_IND packet from Pebble.

risk) use case correspond to what we've found in MetaWear. Fig. 17 shows the CONNECT_REQ packet from the Fitbit Charge HR, while Fig. 18 and Fig. 19 show the same packet types captured from Jawbone UP2 and Pebble Steel, respectively.

It is important to note, as we alluded to earlier, that the three devices use different SSP strategies rescpectively: Fitbit uses passkey entry, Pebble uses numerical comparison, and Jawbone uses just works. It is clear from our results that all three expose their connection forming packets when pairing to would-be attackers. To reiterate, these packets allow an attacker to follow a connection after it is initiated. These results, although troublesome, support our claims presented above and encourage vendor and researcher to adopt a process like the one in Fig. 13. Also it is important to note that, at the time of writing, the findings detailed above are being submitted to CERT/CC for

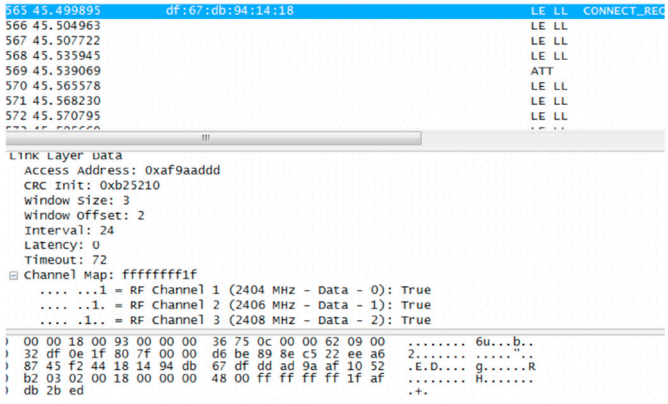

Fig. 17. Fitbit CONNECT_REQ packet captured under the Passkey entry Secure Simple Pairing Strategy.

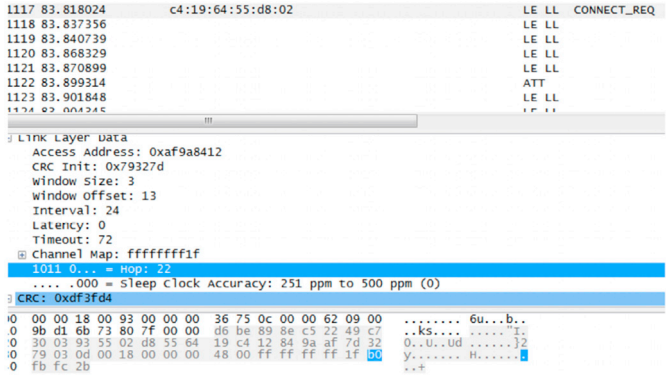

Fig. 18. Similar packet from Jawbone using just works SSP.

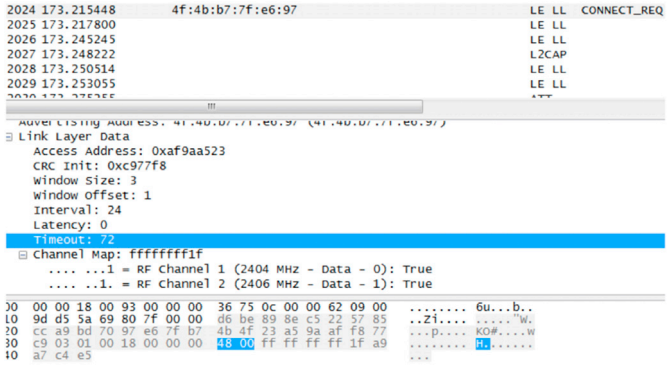

Fig. 19. Same packet from Pebble using numerical comparison.

consideration as a CVE.

5.2.3. *Similarities of data use cases*

Our last examination explored what happened after the connection was established. This step is where the devices vary widely in terms of what is transferred. Fitbit, as investigated by others [44], uses a proprietary encryption protocol after the connection is established. We were able to find specific Bluetooth handles (which operate in a manner similar to registers in low level CPU architectures) where data resided, but the data was encrypted and we did not attempt to reverse the encryption method as this is well out of scope of the work. The captured data shows a handle change notification in Fig. 20.

With Jawbone UP2, we also found that it was using a proprietary encryption protocol. This is consistent with work in the reverse engineering community, such as that by Stefano Brilli [64]. This author also

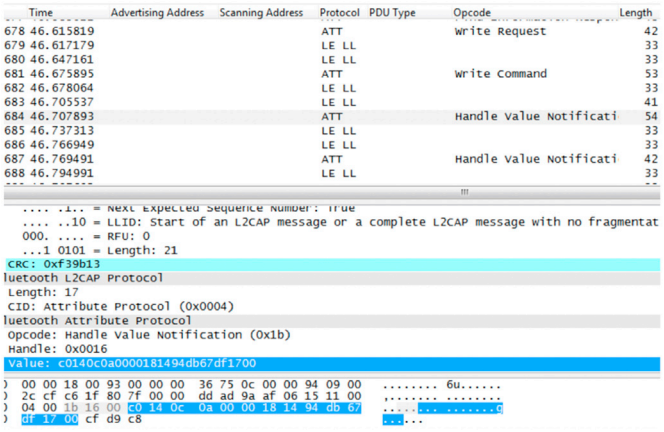

Fig. 20. Handle data being sent by Fitbit to its mobile app.

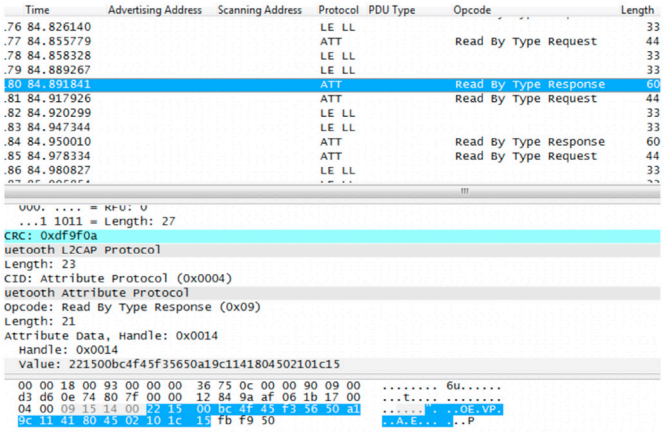

Fig. 21. Jawbone UP2 mobile app reading some kind of data from its paired wearable device.

has demonstrated a hacked mobile app (apart from the app provided by the vendor) that can control the device. For this app to work, he reversed the proprietary encryption protocol. We did not investigate the specifics of his work, but, given his results, it is reasonable to assume the encryption method is not absolutely secure. Fig. 21 shows some data being read on a specific data handle. The data itself was likely encrypted, but we did not investigate it further or attempt to reverse it.

Lastly, Fig. 22 shows a capture of a Pebble Steel post pairing as several pieces of data are established on Bluetooth handles. The Fig. shows a specific packet that is setting the initial value of the device name handle to the device name of the wearable. Like the others, Pebble also has encrypted exchanged data, but still suffered from the same problems of insecure key establishment. With the wide variety of features on Pebble, it was difficult to determine what was occurring after the initial handles were established, but we were able to detect read and write requests post-pairing just as seen in other devices.

Collectively, these results in combination with the results of pairing and advertising use cases suggest that the SecuWear philosophy as espoused in Fig. 13 is a realistic approach for discovering (at a minimum) air vulnerabilities in wearable applications and generalizing them in a way that can be investigated for many different vendor-specific products.

5.2.4. *End-to-end case study*

Demonstrating the entire capability set of SecuWear requires more than just analyzing one particular domain. Therefore, we considered an end-to-end attack case as shown in Fig. 23. In this example, a wearable application includes a MetaWear device being used to gather fitness data (such as gait and heartrate data). The mobile app, created in Cordova, is

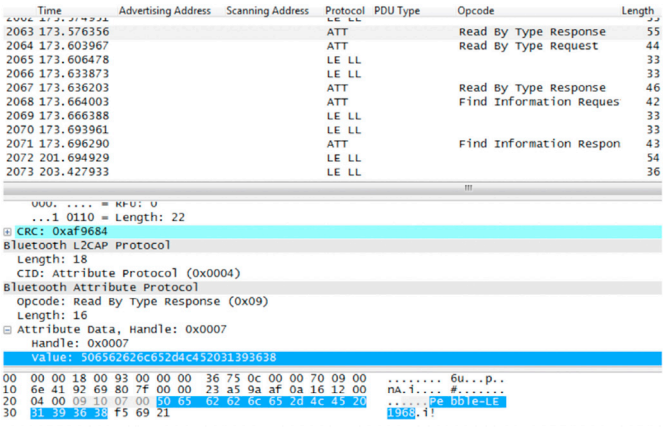

Fig. 22. Initial handle setup established post-pairing on a Pebble Steel.

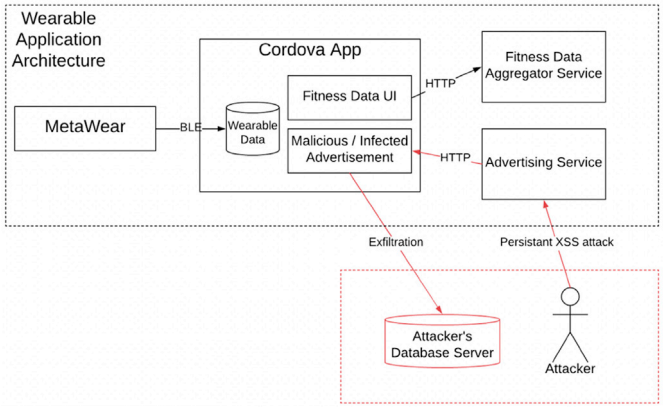

Fig. 23. Code-injection attack allows malicious Javascript to exfiltrate sensor data to an attacker's server.

interacting with a data aggregator service and an advertising service. The fitness data service simply gathers and aggregates data coming from the mobile app and its associated wearable. The advertising service displays ads within the app. In a real-world scenario, the advertising service would likely be a third-party service integration. In this particular case, the advertising service is vulnerable to Cross Site Scripting (XSS). An attacker exploits the vulnerability in the advertising service to persist an attack displayed on the Cordova mobile app. When the user loads the app, their fitness data begins to stream into the local database on the mobile phone and is displayed in the fitness data UI. However, at this point, the app loads an advertisement from the advertising service and gets the attacker's XSS attack. Assume that the Cordova app does not follow the principle of least privilege and has provided the advertisement webview with permissions that allow it to observe and interact with any data in the app. In this case, the XSS attack executes and begins exfiltrating data from the Cordova app and sending it to the attacker's database.

Cases such as this illustrate the usefulness of a tool such as SecuWear. SecuWear collects events from MetaWear, the Mobile App, associated services, and interfaces of communication. Hence, the original XSS attack generates an audit record of an HTTP POST with the XSS attack embedded in it on the advertising service. Next, the platform would capture the network request made by the mobile app for the advertising service with the malicious advertisement payload. It would also capture internal events generated within the Cordova app at any place where an audit hook exists. Next it would see network traffic destined for the attacker's database server. This audit log allows for the detection of domain-spanning security problems.

We investigated this scenario by implementing each component in the architecture and then capturing data in SecuWear to examine the audit log. Simple services were mocked by using WireMock [65] to gather and transmit data as shown in Fig. 23. The same MetaWear app described in 3.2 was modified to include a new webview which makes a request to the advertising server for an advertisement. To implement the case, we gave the webview access to the full scope of the Cordova app, including its data store. A snippet of the resulting audit log gathered by SecuWear is shown in Fig. 24.

One can see a series of events in the audit log capture that highlight abberant behavior in the Cordova app where the advertisement is rendered. Once rendered, the app begins to log wearable data to the attacker's webserver. This is detected by the audit hook in the app and can be used to pinpoint the first time and place where the attack occurrs. This log snippet does not show other events - such as the read/write behavior on the wearable occurring after the XSS attack, but such data is loggable by SecuWear.

6. Conclusion

This paper discusses the design and usefulness of a multi-domain wearable test bed platform called SecuWear. SecuWear facilitates wearable security research by collecting data and enabling researchers to conduct attacks on wearable apps to identify security vulnerabilities in hardwares and softwares. Several types of evaluation are used to assess SecuWear. First, the selected SecuWear components are compared against alternatives. Second, through experiments, we show how several attack vectors present themselves in a generic wearable application and how SecuWear capture the information relevant to identifying and

Origin	Event	Captured By
<attacker IP>	HTTP POST – malicious XSS attack	Web service audit hook on Advertising service
MetaWear	Device advertisement	Ubertooth
Cordova App	Pairing request	Cordova App audit hook, Ubertooth
MetaWear	Pairing response	Ubertooth
Cordova App	Connection successful	Ubertooth, Mobile App audit hook
Cordova App	Turn accelerometer sensor on	Cordova App audit hook, Ubertooth
MetaWear	Sensor on acknowledgement	Ubertooth
MetaWear	Sensor data begins streaming	Ubertooth, Mobile App audit hook
Cordova App	HTTP POST - Authenticate to fitness data aggregator	Cordova App audit hook, network tap, Web service audit hook on Fitness data aggregator service
Fitness data aggregator	HTTP POST Response – return token	Web service audit hook on Fitness data aggregator service, network tap, Cordova App audit hook
Cordova App	HTTP POST - Data streaming to fitness data aggregator	Cordova App audit hook, network tap
Cordova App	HTTP GET to advertising service	Cordova App audit hook, network tap
Advertising Service	HTTP GET Response containing malicious advertisement	Web service audit hook on Advertising service, network tap, Cordova App audit hook
Cordova App	Render Advertisement (XSS attack)	Cordova App audit hook
Cordova App	Access local wearable data store	Cordova App audit hook
Cordova App	HTTP POST to <attacker server>	Cordova App audit hook
…	…	…
Cordova App	Access local wearable data store	Cordova App audit hook
Cordova App	HTTP POST to <attacker server>	Cordova App audit hook

Fig. 24. Audit log of events gathered by SecuWear from the case study scenario.

combating attacks. Lastly, we examine the identified attacks on commercially available wearable products to show that the attack vectors investigated in SecuWear present themselves in the same way as in the commercial products. Overall, the work shows that SecuWear offers a step forward for security vulnerability detection in the current wild-west atmosphere of the wearable computing revolution.

Acknowledgements

This work was supported in part by a Nebraska Research Initiative (NRI) grant entitled "Identifying, assessing, and mitigating wearable security issues in the internet of things". NRI is an investment by the State of Nebraska to provide a research base to enhance economic growth in business and industry, agriculture, social services and health care. The views expressed in this paper are those of the authors and do not necessarily reflect those of NRI or the State of Nebraska.

References

[1] PRNewswire, Wearable Computing: Global Market Set to Grow to $30.2 billion in 2018, 2014. https://www.prnewswire.com/news-releases/wearable-computing-global-market-set-to-grow-to-302-billion-in-2018-279321092.html. (Accessed 15 November 2018).
[2] Fitbit, ChargeHR, 2014. https://www.fitbit.com/chargehr#specs. (Accessed October 2014).
[3] NikePlus, Nike+ Sportband Specifications, 2013. https://secure-nikeplus.nike.com/plus/products/sport_band/. (Accessed 27 October 2014).
[4] M. Patel, J. Wang, Applications, challenges, and prospective in emerging body area networking technologies, IEEE Wirel. Commun. 17 (1) (2010) 80–88.
[5] T. Yilmaz, R. Foster, Y. Hao, Detecting vital signs with wearable wireless sensors, Sensors 10 (12) (2010) 10837–10862.
[6] Pebble, Pebble Smart Watch, 2014. https://getpebble.com/pebbleAccessed. (Accessed 27 October 2014).
[7] Samsung, Samsung Gear Fit Watch, 2014. http://www.samsung.com/us/mobile/wearable-tech/SM-R3500ZKAXAR. (Accessed 27 October 2014).
[8] Google, Google Glass Technical specifications, 2014. https://support.google.com/glass/answer/3064128?hl=en. (Accessed 27 October 2014).
[9] Sensorcon, Sensordrone, 2014. http://sensorcon.com/products/sensordrone-multisensor-tool. (Accessed 27 October 2014).
[10] M. Callaghan, J. Harkin, T. McGinnity, Case study on the Bluetooth vulnerabilities in mobile devices, IJCSNS Int. J. Comput. Sci. Netw. Secur. 6 (4) (2006) 125–129.
[11] M. Ryan, Bluetooth: with low energy comes low security, in: Proceedings of the 7th USENIX conference on Offensive Technologies, 2013.
[12] G.P. Picco, C. Julien, A.L. Murphy, M. Musolesi, G.-C. Roman, Software engineering for mobility: reflecting on the past, peering into the future, in: ACM Proceedings of the on Future of Software Engineering, 2014, pp. 13–28.
[13] National Institute for Standards and Technology, National Vulnerability Database, 2014. https://nvd.nist.gov/. (Accessed 27 October 2014).
[14] The Mitre Corporation, Common Vulnerabilities and Exposures, 2014. https://cve.mitre.org/. (Accessed 27 October 2014).
[15] The Mitre Corporation, Common Vulnerabilities and Exposures, 2014. https://cwe.mitre.org/. (Accessed 27 October 2014).
[16] M.L. Hale, D. Ellis, R. Gamble, C. Waler, J. Lin, Secu Wear: An Open Source, Multi-component Hardware/Software Platform for Exploring Wearable Security, in: IEEE International Conference on Mobile Services, 2015, pp. 97–104.
[17] MbientLab, MetaWear API Documentation, 2015. http://docs.mbientlab.com/. (Accessed 1 March 2015).
[18] MbientLab, MetaWear Product Specification v0.7, 2015. https://www.mbientlab.com/docs/MetaWearPPSv0.7.pdf. (Accessed 1 March 2015).
[19] Cordova plugins on Github, 2015. https://github.com/apache/?query=cordova. (Accessed 1 March 2015).
[20] J.M. Wargo, Apache Cordova 3 programming, Pearson Education, 2013.
[21] Apache Software Foundation, About Apache Cordova, 2013. https://cordova.apache.org/docs/en/latest/. (Accessed 27 October 2014).
[22] Google, Android Bluetooth Low Energy API, 2014. https://developer.android.com/guide/topics/connectivity/bluetooth-le.html. (Accessed October 2014).
[23] Google, Android API Package Index, 2014. http://developer.android.com/reference/packages.html. (Accessed 27 October 2014).
[24] Apple, iOS API and Developer Library, 2014. http://developer.apple.com/library/ios/. (Accessed 27 October 2014).
[25] M. Ossmann, Project ubertooth. Retrieved 18 (2012) 23.
[26] J. Forcier, P. Bissex, W. Chun, Python web development with Django, Addison-Wesley Professional, 2008.
[27] Jawbone, Jawbone Up 2, 2015. http://jawbone.com/store/buy/up2. (Accessed 1 March 2015).
[28] Apple, Apple watch, 2014. https://www.apple.com/watch/. (Accessed 27 October 2014).
[29] AliveCor, AliveCorECG, 2014. http://www.alivecor.com/home. (Accessed 27 October 2014).
[30] Withings, Withings Blood Pressure Monitor, 2014. http://www.withings.com/us/blood-pressure-monitor.html. (Accessed 27 October 2014).
[31] iHealth labs, iHealth Wireless Pulse Oximeter, 2014. http://www.ihealthlabs.com/fitness-devices/wireless-pulse-oximeter/. (Accessed 27 October 2014).
[32] D. Takahashi, Insulin pump hacker says vendor Medtronic is ignoring security risk, Venture Beat, 2011.
[33] G. Hurlburt, J. Voas, K.W. Miller, Mobile-app addiction: threat to security? IT Prof. 13 (6) (2011) 9–11.
[34] R. Balebako, L. Cranor, Improving app privacy: nudging app developers to protect user privacy, IEEE Secur. Priv. 12 (4) (2014) 55–58.
[35] C.S. Gates, C. Jing, L. Ninghui, R.W. Proctor, Effective Risk Communication for Android Apps, IEEE Trans. Dependable Secure Comput. 11 (3) (2014) 252–265.
[36] R. Hunt, Security testing in Android networks - A practical case study, in: 19th IEEE International Conference on Networks, 2013, pp. 1–6.
[37] T. Mikkonen, A. Taivalsaari, Apps vs. open web: The battle of the decade, in: Proceedings of 2nd Workshop on Software Engineering for Mobile Application Development, 2011.
[38] A.K. Jain, D. Shanbhag, Addressing security and privacy risks in mobile applications, IT Prof. 14 (5) (2012) 28–33.
[39] C. Gomez, J. Oller, J. Paradells, Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology, Sensors 12 (9) (2012) 11734–11753.
[40] BluetoothSIG, Bluetooth Core Specification version 4.1, 2014. https://www.bluetooth.com/specifications/bluetooth-core-specification. (Accessed 27 October 2014).
[41] BluetoothSIG, Bluetooth LE Approved Service Specification Formats, 2014. https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx. (Accessed 27 October 2014).
[42] W. Enck, et al., TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones, ACM Trans. Comput. Syst. 32 (2) (2014).
[43] Great Scott Gadgets, Ubertooth Build Gudie Release 2014-02-R2, 2015. https://github.com/greatscottgadgets/ubertooth/wiki/Build-Guide. (Accessed 1 March 2015).
[44] B. Cyr, W. Horn, D. Miao, M. Specter, Security Analysis of Wearable Firness Devices (Fitbit), Massachusets Institute of Technology, 2014.
[45] D. Coleman, Bluetooth Low Energy Apache Cordova Plugin, 2015. https://github.com/don/cordova-plugin-ble-central. (Accessed 1 March 2015).
[46] MetaGeek, Wi-Spy + Chanalyzer 2.4 GHz Spectrum Analyzer, 2015. http://www.metageek.com/products/wi-spy/. (Accessed 1 March 2015).
[47] S. Xanthopoulos, S. Xinogalos, A comparative analysis of cross-platform development approaches for mobile applications, in: Proceedings of the 6th Balkan Conference in Informatics, 2013.
[48] Django REST Framework, 2015. http://www.django-rest-framework.org/. (Accessed 1 March 2015).
[49] L. Zhou, H.-C. Chao, Multimedia traffic security architecture for the internet of things, IEEE Netw. 25 (2011) 3.
[50] L. Zhou, H.-C. Chao, A.V. Vasilakos, Joint forensics-scheduling strategy for delay-sensitive multimedia applications over heterogeneous networks, IEEE J. Sel. Area. Commun. 29 (7) (2011) 1358–1367.
[51] M. Dowd, J. McDonald, J. Schuh, The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities, Pearson Education, 2006.
[52] B. Stern, Getting Started with Flora, 2015. https://learn.adafruit.com/getting-started-with-flora/overview. (Accessed 1 March 2015).
[53] WARP, Wearable Reference Platform, 2015. http://www.warpboard.org/. (Accessed 1 March 2015).
[54] G. Hartmann, G. Stead, A. DeGani, Cross-platform mobile development, Mob. Learn. Environ. 16 (9) (2011) 158–171.
[55] J. Rowberg, Bonding, encryption, and MITM protection with BLE modules, 2013. https://bluegiga.zendesk.com/entries/22882472–REFERENCE-Bonding-encryption-and-MITM-protection-with-BLE-modules. (Accessed 14 October 2015).
[56] C. Mui, Bluetooth pairing machanism - Legacy Pairing and Secure Simple Pairing (SSP), 2013. https://bluegiga.zendesk.com/entries/23078058-Bluetooth-pairing-machanism-Legacy-Pairing-and-Secure-Simple-Pairing-SSP. (Accessed 14 October 2015).
[57] W3af Framework, 2014. http://w3af.org/. (Accessed 14 October 2015).
[58] Metasploit Penetration Testing Software, 2015. http://www.metasploit.com/. (Accessed 14 October 2015).
[59] SQLMap: An Automatic SQL Injection and Database Takeover Tool, 2015. http://sqlmap.org/. (Accessed 14 October 2015).
[60] CERT, The Computer Emergency Response Team Coordination Center. http://www.cert.org/, 2015. (Accessed 14 October 2015).
[61] United States Computer Emergency Response Team. https://www.us-cert.gov/, 2015. (Accessed 12 November 2015).
[62] Industrial Control Systems Cyber Emergency Response Team. https://ics-cert.us-cert.gov/, 2015. (Accessed October 2015).
[63] Japan Computer Emergency Response Team Coordination Center. https://www.jpcert.or.jp/english/, 2015. (Accessed October 2015).
[64] S. Brilli, Reverse Engineering the New Jawbone UP2, 2015. http://stefano.brilli.me/blog/posts/reverse-engineering-the-new-jawbone-up2/index.html. (Accessed October 2015).
[65] WireMock, Mock your APIS for Fast, Robust, and Comprehensive Testing, 2018. http://wiremock.org/. (Accessed July 2018).