Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device

Mahmudur Rahman, Bogdan Carbunar, Madhusudan Banik

School of Computing and Information Sciences
Florida International University, Miami, FL
Email: {mrahm004, carbunar, mbani002}@cs.fiu.edu

Abstract. The fusion of social networks and wearable sensors is becoming increasingly popular, with systems like Fitbit automating the process of reporting and sharing user fitness data. In this paper we show that while compelling, the integration of health data into social networks is fraught with privacy and security vulnerabilities. Case in point, by reverse engineering the communication protocol, storage details and operation codes, we identified several vulnerabilities in Fitbit. We have built FitBite, a suite of tools that exploit these vulnerabilities to launch a wide range of attacks against Fitbit. Besides eavesdropping, injection and denial of service, attacks can also lead to financial rewards. We have built FitLock, a lightweight Fitbit extension that defends against these attacks. Our experiments on BeagleBoard and Xperia devices show that FitLock's end-to-end overhead is only 2.4%.

1 Introduction

Recent advances in wearable, user-friendly devices equipped with smart sensors (e.g., pedometers, heart rate and sleep monitors) and wireless technologies, are facilitating the emergence of *social sensor networks* (SSNs): sites that collect and share not only regular social networking information (e.g., status updates, location reports, friend lists) but also user health-centric data.

Fitbit [1], a popular social sensor network centers its existence on fitness sensor data. Fitbit (see Figure 1) consists of (i) *trackers*, wireless-enabled, wearable devices that record their users' daily step counts, distance traversed, calories burned and floors climbed as well as sleep patterns when worn during the night, (ii) an online social network (called *webserver* in the following) that automatically captures, displays and shares fitness data of its users and (iii) user USB base stations that act as bridges between trackers and the webserver. Trackers communicate to bases in a wireless fashion over the ANT [2] protocol.

While popular and useful in its encouragement of healthy lifestyles, the combination of health sensors and social networks makes social sensor networks the source of significant privacy and security issues. In this paper we show that Fitbit is vulnerable to a diverse set of attacks. Besides standard social networking problems, including infiltration attacks [3] and private data leaks to general

account holders [1], Fitbit is made vulnerable by the wireless nature of tracker communications and poor security practices.

In order to expose Fitbit's vulnerabilities, in a first contribution, we have reverse engineered the semantics of tracker memory banks, the command types and the tracker-to-social network communication protocol. Furthermore, we have built FitBite, a suite of tools that exploit Fitbit's design, and used it to prove the feasibility of several attacks. For instance, we show that FitBite allows attackers to capture and modify the data stored on any tracker situated within a radius of 15 ft. This is an important privacy breach, as information accessible from a tracker includes personally identifiable information (user name, zipcode, city, height, weight) and fitness data.

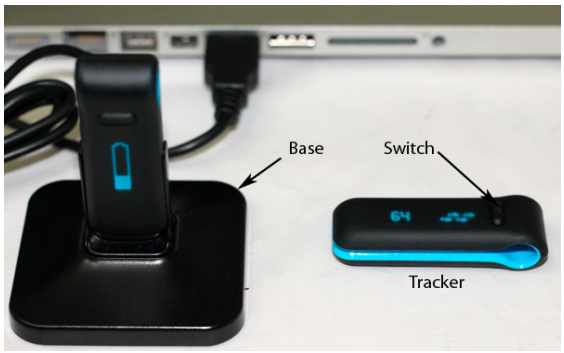


Fig. 1. Fitbit system components: trackers (one mounted on the base), the base (arrow indicated), user laptop. The arrow pointing to the tracker shows the switch button, allowing the user to display various fitness data.

In a second contribution, we propose FitLock, a lightweight extension that uses efficient cryptographic tools to secure the Fitbit protocol. We show that FitLock prevents the FitBite attacks. Our end-to-end implementation on both BeagleBoard [4] and Xperia devices shows that the computation and communication overhead imposed by FitLock on Fitbit is only 2.4%. The project website containing the source code of FitBite and FitLock is publicly available at [5].

The paper is organized as follows. Section 2 reverse engineers Fitbit and Section 3 introduces FitBite, the suite of attack tools. Section 4 introduces FitLock, our defense extension and proves its security. Section 5 describes our implementation results. Section 6 describes related work and Section 7 concludes.

[1] Fitbit has suffered criticism due to its initial default access control settings: The reported sensor information was made publicly available on Fitbit's social network.

2 Reverse Engineering Fitbit

We reverse engineered the Fitbit communication protocol, including the message communication format among the participating devices, and the structure and data format of each memory bank. A tracker has both *read banks*, containing data to be read by the base and *write banks*, containing data that can be written by the base.

Fitbit uses *service logs*, files that store information concerning communications involving the base. On the Windows installation of the Fitbit software, daily logs are stored in cleartext in files. Data retrieved from the tracker to be uploaded to the social network is encoded in base64 format. We have exploited Fitbit's lack of encryption in the messages sent between the base and the tracker to implement a USB based filter driver that creates separate logs of the data flowing to and from the base. The captured logs reveal that during the upload session, the webserver reads data from 6 memory banks, writes on 2 write memory banks and clears data from 5 memory banks by sending requests to the tracker through the base. The read bank #1 stores the daily user fitness records while the write bank #0 stores 64 bytes concerning the Device Settings and Profile Settings as specified on the user's Fitbit account. The communication between the webserver and a tracker through a base is embedded in XML blocks, that contain base64 encoded opcodes – commands for the tracker. All opcodes are 7 bytes long and vary according to specific type of instructions (e.g., read, write, erase).

2.1 The Fitbit Communication Protocol

In the following, for brevity, we use the notation "HOME" to denote the full URL `http://client.fitbit.com`. The data flow between the tracker, base and the webserver during the data upload operation, is divided into 4 phases, beginning at steps 2, 3, 5 and 7:

1. Upon receiving a beacon from the tracker, the base establishes a connection with the tracker.
2. **Phase 1:** The base contacts the webserver at the `HOME/device/tracker/uploadData` and sends basic client and platform information.
3. **Phase 2:** The webserver sends the tracker id and the opcode for retrieving tracker information (TRQ-REQ).
4. The base contacts the specified tracker, retrieves its information TRQ-INFO (serial number, firmware version, etc.) and sends it to the webserver at the `HOME/device/tracker/dumpData/lookupTracker`.
5. **Phase 3:** Given the tracker's serial number, the webserver retrieves the associated tracker public id (TPI) and user public id (UPI) values. The webserver sends to the base the TPI/UPI values along with the opcodes for retrieving fitness data from the tracker (READ-TRQ).
6. The base forwards the TPI and UPI values and the opcodes to the tracker, retrieves the fitness data from the tracker (TRQ-DATA) and sends it to the webserver at the `HOME/device/tracker/dumpData/dumpData`.

7. **Phase 4:** The webserver sends to the base opcodes to WRITE updates provided by the user in her Fitbit social network account (device and profile settings, e.g., body and personal information, time zone, etc). The base forwards the WRITE opcode and the updates to the tracker, who overwrites the previous values on its write memory banks.
8. The webserver sends opcodes to ERASE the fitness data from the tracker. The base forwards the ERASE request to the tracker, who then erases the contents of the corresponding read memory banks.
9. The base forwards the response codes for the executed opcodes from the tracker to the webserver at the address `HOME/device/tracker/dumpData/clearDataConfigTracker`.
10. The webserver replies to the base with the opcode to CLOSE the tracker.
11. The base requests the tracker to SLEEP for 15 minutes, before sending its next beacon.

3 FitBite: Attacking Fitbit

FitBite consists of two modules. The Tracker Module (TM) reads and writes the tracker data. The Base Module (BM) retrieves/injects data from/to the tracker and uploads it into the account of the tracker's owner on the webserver. FitBite implements the following attacks:

Tracker Private Data Capture (TPDC). The TM module is used to discover any tracker device within a radius of 15 ft and capture the fitness information it stores. This attack can be launched in public spaces(e.g., parks, sports venues).

Fig. 2. Outcome of TI attack on tracker: 167,116 steps recorded within 1 day.

Tracker Injection (TI). The TM module is used to modify any of the fitness data stored by nearby trackers. FitBite reads the selected data from a specified memory bank and modifies the target bytes. The TM can simultaneously modify multiple fitness records (memory banks).

Figure 3 shows an example of a victim tracker, displaying an unreasonable value for the daily number of steps taken by its user.

User Account Injection (UAI). The BM module is used to inject data on the Fitbit social network accounts of the owners of nearby trackers. It sends to the webserver, fabricated data replies embedding the desired fitness data, encoded in base64 format. The webserver does not authenticate the request message and does not check for data consistency, thus accepts the data.

Figure 3 shows a snapshot of one account where we have successfully injected number of steps taken by the "account owner", while keeping the other values intact. This shows that (i) FitBite can inject an unreasonable daily step count

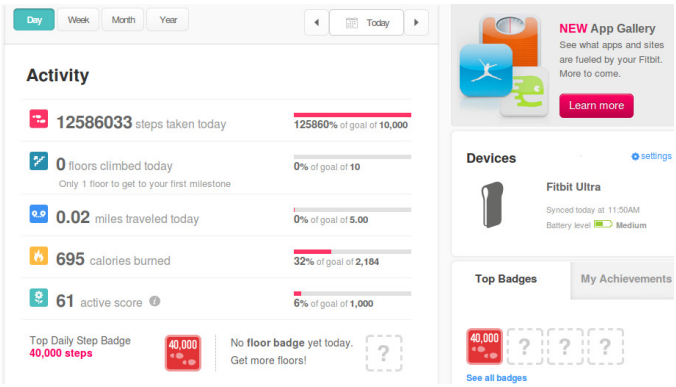


Fig. 3. Snapshot of Fitbit user account data injection attack.

(12.58 million) into the account of any tracker owner located in its vicinity and (ii) Fitbit does not verify the consistency of the data: the 12.58 million steps are shown to correspond to 0.02 traveled miles.

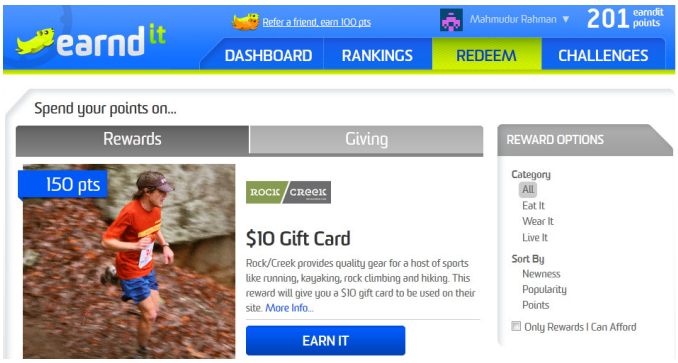


Fig. 4. Earndit points and available gift cards

Free Badges and Financial Rewards. By successful injection of large values in their social networking accounts, FitBite enables attackers to achieve special milestones and acquire Fitbit provided "merit" badges, without doing the required work. Figure 3 shows that the injected value of 12.58 million steps, being greater than 40,000, enables the account owner to acquire a "Top Daily Step" badge. Fitbit users can link their social networking accounts to systems that reward users for exercising, e.g., Earndit [6]. We have shown through experiments that attackers can accumulate undeserved financial rewards. Figure 4 shows an example where we have accumulated 200 Earndit points, that can be redeemed to a $20 gift card.

Battery Depletion Attack. FitBite allows the attacker to continuously query trackers in her vicinity and drain their batteries at a faster rate. Through experiments, we have shown that when forcing trackers to upload data 4 times per

minute, FitBite drains tracker batteries 21 times faster than the regular upload mode.

4 FitLock: Protecting Fitbit

FitLock considers an adversarial model where attackers can impersonate system participants, and snoop, inject and jam existing communications. We assume however that attackers do not have physical access to the victim trackers.

FitLock consists of a *bind* procedure (BindUserTracker), where the user associates a new tracker to her online social network account and an *upload* procedure (UploadData), where the tracker reports information upon demand from the social network. Each tracker T has a unique serial number id_T and a secret symmetric encryption key sk_T, shared with the webserver. These values are stored in a write-once-read-many (WORM) area of the tracker's memory banks. The tracker never reveals (e.g., displays or communicates) the secret key. The webserver stores a database Map that associates a tracker id to tracker related data, including symmetric key, user id and session id. Initially, Map only maps tracker ids into corresponding symmetric encryption keys.

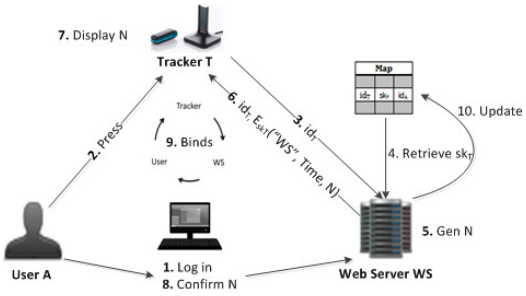


Fig. 5. The BindTrackerUser protocol between the user, tracker and the (Fitbit) webserver

Let Id_A denote the unique user id of the account that user A has on the Fitbit social network. In the following, we use the notation $F(P_1(args_1), .., P_n(args_n))$ to denote a protocol F running between participants P_1,..P_n, each with its own input arguments. For instance, the following BindTrackerUser protocol involves user A, with her account id and a time interval s as input arguments, her tracker T, with its id and secret key as arguments, her base B with no arguments and the (Fitbit) webserver WS, with its Map structure as input argument. The BindTrackerUser protocol allows user A to bind her new tracker to her social network account (illustrated in Figure 5).

BindTrackerUser(A(Id_A,s),T(id_T, sk_T),B(),WS(Map)). User A logs in into her account on the Fitbit social network (step 1 in Figure 5). A presses T's switch button for s seconds (step 2). Upon this action, the tracker T reports its identifier id_T in cleartext to WS, through the user's base (step 3). WS uses the Map structure to retrieve the symmetric key associated with the id_T, i.e., sk_T (step 4). It then generates a 6 digit long random value, N (step 5). WS

sends to T the request value

$$id_T, E_{sk_T}(\text{“}WS'', Time, N),$$

where $Time$ is WS's current time (step 6). WS keeps track of all requests sent to trackers and pending responses, indexed under the tracker id and the nonce value. WS associates an expiration time with each entry, and removes entries as they expire without being answered.

Upon reception of this message, T uses its symmetric key, sk_T, to decrypt it. It verifies the freshness (the $Time$ value) and authenticates WS through its ability to have encrypted this message, containing the string "WS", using the key sk_T. If the verifications succeed, the tracker displays the 6 digit random nonce N (step 7). User A reads and enters this nonce into a confirmation box in her Fitbit social network account (step 9). Then, if WS finds any pending (not expired) request matching the value entered by the user, WS associates Id_A to id_T and sk_T in the Map structure (step 10). WS removes this request from the list of pending requests.

The following procedure, UploadData, is used to secure the Fitbit communication protocol described in Section 2. It involves a tracker T (taking as arguments its id id_T, secret key sk_T, stored fitness $data$, expiration interval δt and retry counter r), a base B (with no arguments) and the webserver WS (with its Map structure and the same expiration intervals and counter as T).

All communication between T and WS is encrypted with their shared key sk_T. Each communication session between WS and T has a monotonically increasing session id S_{wst}. T and WS do not accept messages with older session id numbers.

UploadData(T(id_T,sk_T,data,δt,r),B(),WS(Map,δt,r)). A new session starts only after the tracker's beacon is received by the base and the base sets up a connection with the tracker (step 1). Within each session, the communication between WS and T starts with a request from WS followed by a response from T. Each request contains a request type REQ $\in$ {TRQ-REQ, READ-TRQ, WRITE, ERASE, CLOSE} and a counter C_{ws} encoding the number of times this particular request has been re-transmitted. Within a session, T stores the latest C_{ws} received from WS for any request type, or -1 if no request has been received yet. Thus, a request from WS to T has the format

$$id_T, E_{sk_T}(REQ, S_{wst}, C_{ws}),$$

where S_{wst} is the current session id and C_{ws} is set to 0 for the first transmission of the current REQ type. Upon receiving such a message, the base B uses id_T to route the packet to the correct tracker T in its vicinity. T uses its secret key to decrypt the packet and authenticate WS: verify that the first field is a meaningful request type, the second field contains the current session id and the value of the third field exceeds its currently stored value for REQ. If either verification fails, T drops the packet. Otherwise, T stores the received C_{ws} value, associated with the REQ type for the current session, and replies to this request with

$$id_T, E_{sk_T}(RESP, S_{wst}, C_T),$$

where RESP $\in$ {TRQ-INFO, TRQ-DATA, CLEAR} denotes T's response type and C_T is its counter (initialized to 0).

WS waits a predefined interval δt to receive the reply RESP from T. If it does not receive it in time, WS repeats the request, with an incremented counter C_{ws}. If WS's re-transmission counter reaches a maximum value, r, and no corresponding RESP is received within the δt interval, WS increments the session id S_{wst}. Similarly, if C's re-transmission counter reaches the maximum value r and the next request is not received from WS, T increments S_{wst}. This means that T and WS consider themselves to have been disconnected and their next communication needs to start from the beginning with a new session id. If T receives a REQ from WS that has a session id larger (by 1) than its current session id, T drops the data associated with the current session, and begins a new session with the incremented session id.

At the successful completion of a session, both T and WS increment the session id S_{wst}. WS stores this value in Map indexed under id_T.

4.1 Analysis

We introduce now several properties of FitLock.

Claim. Without physical access to the tracker, an attacker cannot hijack the tracker during the $BindTrackerUser$ procedure.

Proof. (Sketch) A *tracker hijack* attack, takes place during a normal execution of the $BindTrackerUser$ procedure by a victim user for her tracker T. The adversary attempts to bind the victim tracker T to another user account, potentially controlled by the attacker. Let M denotes the Fitbit account owned by the adversary. Without physical access to the tracker, the adversary cannot read the 6 digit random nonce displayed on the tracker and upload it in M. However, the adversary is able to capture packets exchanged by WS and T during a $BindTrackerUser$ procedure. The adversary could then attempt launch a *rush* attack. In a rush attack, the adversary decrypts a captured packet, recovers the nonce N sent by WS to T, and uploads it in M, before the valid user. Rush attacks are prevented by the semantic security of the encryption scheme of FitLock – the adversary cannot recover the nonce. □

FitLock prevents the TPDC attack through the use of semantically secure encryption. The non-malleability of the encryption also prevents injection TI, UAI and ensuing free badge and financial rewards attacks, generated from previously captured (encrypted) messages. The use of session identifiers and re-transmission counters prevents replay attacks.

Claim. FitLock prevents Battery Depletion attack.

Proof. (Sketch) FitLock's use of semantically secure symmetric encryption to protect communications, prevents attackers from obtaining a response from trackers. The attacker cannot replay requests with old session ids or old counters

(for the current session id): Upon receiving invalid requests or requests with old session ids or old counter values, the tracker drops them, thus does not consume power to answer them. □

5 Evaluation

5.1 Experimental Setup

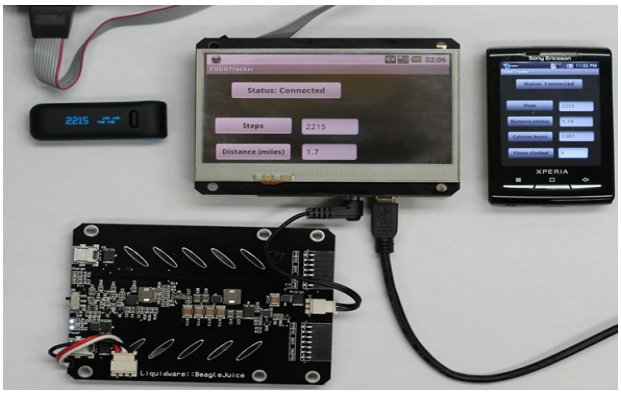


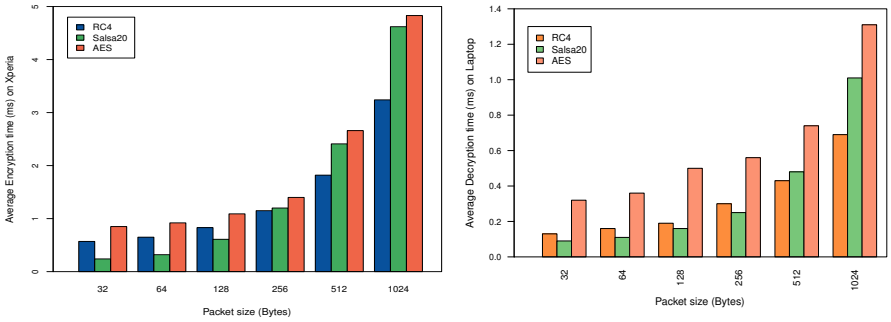
Fig. 6. Snapshot of testbed for FitLock, consisting of BeagleBoard and Xperia devices used as Fitbit trackers.

We implemented FitLock in Android. We choose the following resource-constrained hardware and system setup to compare FitLock and Fitbit. We have tested the tracker side on two devices, (i) a Revision C4 of an OMAP 3530 DCCB72 720 MHz BeagleBoard system [4] and (ii) a Sony Ericsson Xperia X10 mini smartphone (ARM 11 CPU@600 MHz, 128MB RAM). We used a Dell laptop featuring a 2.4GHz Intel Core i3 processor and 4GB of RAM, for the webserver (built on the Apache webserver 2.4). We also implemented a client-server Bluetooth socket communication protocol between the tracker (Xperia smartphone) and the base using PyBluez [7] python library.

For connectivity between the base and the webserver, the laptops use their own 802.11b/g Wi-Fi interfaces. Figure 6 shows a snapshot of our testbed. For encryption we experimented with RC4, AES and the Salsa20 [8] stream cipher, selected in the final eSTREAM portfolio [9].

5.2 Results

In the following, all reported values are averages taken over at least 10 independent protocol runs. A potential bottleneck of FitLock is in the encryption of packets by the tracker. We compared then the performance of RC4, Salsa20 and AES, with a key size set to 128 bits, on the BeagleBoard and the Xperia



(a) (b)

Fig. 7. FitLock overhead. (a) Encryption time overhead on Xperia. (b) Decryption time overhead on webserver (Dell laptop).

devices. The packet size ranged from 32 bytes to 1024 bytes. Figure 7(a) shows the execution time of the three protocols on the Xperia smartphone. For small packet sizes, Salsa20 performs the best. As the packet size increases, RC4 performs slight better than Salsa20. Both RC4 and Salsa20 outperform AES for any packet size. Even for a packet size of 1024 bytes, the average encryption times for RC4, Salsa20 and AES are only 3.24ms, 4.62ms and 4.83ms respectively.

We further examined the packet decryption overhead on the webserver using the above mentioned protocols. Figure 7(b) shows the dependence of the decryption time on the packet size. RC4 and Salsa20 perform better than AES. Even for 1024 byte packets, the average decryption overheads for RC4, Salsa20 and AES are 0.69ms, 1.01ms and 1.31ms respectively.

Finally, we report the measured end-to-end performance of FitLock and compare it against the performance of Fitbit. We have implemented and tested both Fitbit and FitLock on our testbed. Figure 8 shows the results split into the times of each of the 4 phases of the webserver-to-tracker communication protocol. We have set the secret key size to 256 bits. The end-to-end FitLock overhead is 1518ms. The total time of Fitbit is 1481ms. Thus, FitLock adds an overhead of 37ms, accounting for 2.4.% of Fitbit's time.

6 Related Work

Halperin et al. [10] were the first to demonstrate the practicality of security attacks on pacemakers and implantable cardiac defibrillators (ICDs), by reverse engineering an ICD's communications protocol with an oscilloscope and a software radio. They used RF power harvesting for authentications and key exchange to protect Implantable Medical Devices (IMD). Rasmussen et al. [11] proposed proximity-based access control for IMDs, to verify the distance of the communicating peer before initiating wireless communication, thereby limiting

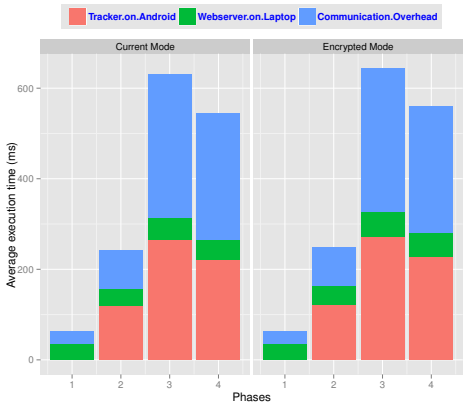


Fig. 8. End-to-end delay, Fitbit and FitLock.

attackers to a certain physical range. Li et al. [12] demonstrated successful security attacks on a commercially deployed glucose monitoring and insulin delivery system and provided two defenses based on rolling-code cryptographic protocols and body-coupled communication. They used a software radio board to intercept radio communications within a frequency band and generate wireless signals. While similar, our work does not require additional hardware to intercept or protect communications, it applies to social sensor networks, and uses a different methodology: we reversed engineered Fitbit's communication protocol using service logs captured through a USB based filter driver. Furthermore, our defense mechanisms take advantage of the different system model and capabilities present in trackers and Fitbit's social sensor network.

Marti et al. [13] described the requirements and implementation of the security mechanisms for MobiHealth, a wireless mobile health care system. MobiHealth relies on Bluetooth and ZigBee link layer security for communication to the sensors. Barnickel et al. [14] targeted security and privacy issues with MobiHealth for HealthNet, a health monitoring and recording system. They proposed a security and privacy aware architecture, relying on data avoidance, data minimization, decentralized storage, and the use of cryptography. While similar in goals, our approach does not require additional costly unique hardware (e.g., wearable sensor shirts and constant Bluetooth connection between smartphone and the body sensor network).

Lim et al. [15] analyzed the security of a remote cardiac monitoring system relying on a Body Area network (BAN). The defense mechanism proposed relies on an asymmetric cryptosystem, making it unsuitable for Fitbit trackers. Muraleedharan et al. [16] proposed two types of possible denial-of-service attacks including Sybil and wormhole attacks in a health monitoring system using wireless sensor networks. They proposed an energy-efficient cognitive routing algorithm to address such attacks. Kulkarni and Öztürk [17] survey security and privacy issues for IMDs.

7 Conclusion and Future Work

In this paper, we studied security and privacy issues of Fitbit, a popular fitness tracking system. We have developed FitBite, a software module that relies on reverse engineering of Fitbit's data communication protocol, to launch both passive and active attacks on Fitbit. We have proposed FitLock, a Fitbit extension that defends against FitBite. We have implemented FitLock and we have shown that FitLock introduces a negligible end-to-end overhead on Fitibit (2.4%).

In future work, we will consider physical, *mule* attacks, where attackers attach trackers to various moving objects (e.g., car wheel, fan). We will explore correlation approaches, studying relations (or lack thereof) between pedometer and other sensor readings (e.g., GPS, heart rate).

References

1. Fitbit. `http://fitbit.com/`.
2. Ant message protocol and usage. `http://www.sparkfun.com/datasheets/Wireless/Nordic/ANT-UserGuide.pdf`.
3. Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: when bots socialize for fame and money. In *ACSAC '11*, 2011.
4. G. Coley. *Beagleboard system reference manual.* BeagleBoard.org, December 2009.
5. FitBite and FitLock: Attacks and defenses on Fitbit Tracker. `http://users.cis.fiu.edu/~mrahm004/fitlock`.
6. Earndit: We reward you for exercising. `http://earndit.com/`.
7. Pybluez. `http://code.google.com/p/pybluez/`.
8. Daniel J. Bernstein. The salsa20 family of stream ciphers. In *New Stream Cipher Designs*, pages 84–97. Springer-Verlag Berlin, Heidelberg, 2008.
9. eSTREAM: the ECRYPT stream cipher project. `http://www.ecrypt.eu.org/stream/`.
10. D. Halperin, T. H. Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *IEEE S & P*, 2008.
11. K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun. Proximity-based access control for implantable medical devices. In *ACM Conference on Computer and Communications Security*, pages 410–419, 2009.
12. Chunxiao Li, A. Raghunathan, and N.K. Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *IEEE Healthcom*, 2011.
13. R. Marti, J. Delgado, and X. Perramons. Security specification and implementation for mobile e-health services. In *IEEE CEC*, 2004.
14. J. Barnickel, H. Karahan, and U. Meyer. Security and privacy for mobile electronic health monitoring and recording systems. In *WoWMoM*, 2010.
15. S. Lim, T.H. Oh, Y. Choi, and T. Lakshman. Security issues on wireless body area network for remote healthcare monitoring. In *SUTC*, 2010.
16. Rajani Muraleedharan and Lisa Ann Osadciw. Secure health monitoring network against denial-of-service attacks using cognitive intelligence. In *CNSR*, 2008.
17. P. Kulkarni and Y. Öztürk. Requirements and design spaces of mobile medical care. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2007.