Deploying Data-Driven Security Solutions on Resource-Constrained Wearable IoT Systems

Hang Cai*, Tianlong Yun†, Josiah Hester‡, Krishna K. Venkatasubramanian*
*Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA
{hcai, kven}@wpi.edu
†Department of Computer Science, Dartmouth College, Hanover, NH
{tyun@cs.dartmouth.edu}@cs.dartmouth.edu
‡School of Computing, Clemson University, Clemson, SC
{jhester@g.clemson.edu}@g.clemson.edu

***Abstract*—Wearable Internet-of-Things (WIoT) environments have demonstrated great potential in a broad range of applications in healthcare and well-being. Security is essential for WIoT environments. Lack of security in WIoTs not only harms user privacy, but may also harm the user's safety. Though devices in the WIoT can be attacked in many ways, in this paper we focus on adversaries who mount what we call *sensor-hijacking attacks*, which prevent the constituent medical devices from accurately collecting and reporting the user's health state (e.g., reporting old or wrong physiological measurements). In this paper we outline some of our experiences in implementing a data-driven security solution for detecting sensor-hijacking attack on a secure wearable internet-of-things (WIoT) base station called the Amulet. Given the limited capabilities (computation, memory, battery power) of the Amulet platform, implementing such a security solution is quite challenging and presents several trade-offs with respect to detection accuracy and resources requirements. We conclude the paper with a list of insights into what capabilities constrained WIoT platforms should provide developers so as to make the inclusion of data-driven security primitives in such systems.**

I. Introduction

Wearable IoT platforms have demonstrated great potential in a broad range of applications in healthcare and well-being. Figure 1 shows a typical architecture of a wearable IoT environment. It consists of various types of low-cost medical devices (i.e., *sensors*) that form a distributed wireless network around the user. These sensors monitor various types of personal health information from the user and wirelessly forward them to a *base station*. The base station is a always-present, safety-critical WIoT device that is designed to act on the data received from the sensors, such as perform closed-loop control of specific ailments that the user may have. It is designed to have a very long battery-life and also be secure in terms of apps that it runs. The base station further forwards the data to a *sink entity*. The *sink* is resource-rich device responsible for providing expensive but non safety-critical operations such as local storage of historical patient information, visualization tools, and cloud connectivity. Typically the sink can be implemented on a more generic smartphone or a tablet, which may not be secure given the eco-system that it may be a part of.

Security is essential for WIoT environments. Lack of security in WIoTs not only harms user privacy, but may also harm the user's safety. Though devices in the WIoT can be attacked in many ways, in this paper we focus on adversaries who mount what we call *sensor-hijacking attacks*. We *define* sensor-hijacking attacks as attacks that prevent sensors from accurately collecting or reporting their measurements. In this paper, we focus on the health and wellness applications,

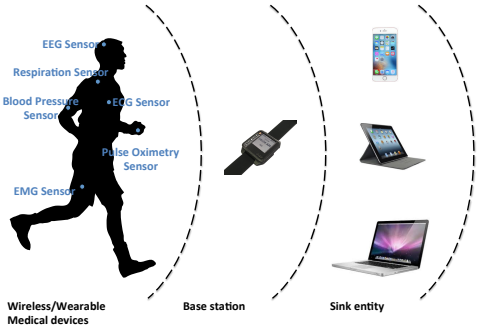

Fig. 1: **Wearable IoT Environment**

where sensor-hijacking attacks typically manifest by preventing medical devices from correctly gathering or reporting sensor data about the user's health state (e.g., reporting old or wrong physiological measurements). Sensor-hijacking in WIoT systems presents itself due to vulnerabilities in four basic areas of the system (each of which has been exploited): (1) through the communication channel [1], [2], (2) through the software and firmware update process [3], (3) through the unprotected sensory-channel [4], [5] and (4) through direct physical compromise.

In our efforts we focused on developing an *attack-agnostic way* to secure the WIoT systems against sensor-hijacking attacks. We do this by observing that the goal of any security solution with respect to WIoT should be to ensure that the user is protected from harm. This means, that the adversary should not be allowed to introduce incorrect user state information into the system such that wrong diagnosis and treatment are made. Therefore, if we can analyze the data being received from the sensors we should be able to determine if the data has been legitimately measured from the user or has been tampered by someone. In this regard, our work has focused on detecting attacks on electrocardiogram (ECG) sensors in WIoT. We focus on ECG sensors for several reasons: (1) they are one of the most important vital signs collected by a variety of WIoT; (2) ECG is a representation of the cardiac process, which is very important to monitor for any user; and (3) ECG sensors have already been demonstrably compromised to measure incorrect user state in a variety of contexts [1].

We have developed a novel methodology for detecting alteration of electrocardiogram (ECG) measurements in a WIoT device due to sensor-hijacking called **SIgnal Feature-correlation-based Testing (SIFT)** [6]. SIFT leverages the fact that different physiological signals generated by the same underlying physiological process are inherently related,

i.e., they share similar features among them. For example, electrocardiogram and arterial blood pressure are different manifestations of the cardiac process and the two signal time-series track each other. Consequently, SIFT uses a machine learning-based model that detects sensor hijacking attacks on ECG sensors capturing its interrelationship with arterial blood pressure sensor measurements. So far we have only validated the performance of our work through simulation using MATLAB, where it has shown promise [6]. However, SIFT is designed to be ultimately implemented on the base station of a WIoT. This is easier said than done because SIFT uses complex feature extraction and sophisticated machine learning algorithms, which computationally and memory-wise expensive.

Consequently, the goal of this paper is to evaluate the feasibility of implementing the SIFT approach on a WIoT base station. We utilize the Amulet platform [7], a wrist-mounted device, as our base station of choice. We chose the Amulet as our base station because: (1) it is designed to be a secure base station for WioT environment, (2) it is a platform that has strict controls on the apps deployable on it; (3) it is a low-power device designed to have a battery lifetime measured in weeks; and (4) it supports multiple apps executing on it simultaneously. In all, we implemented three different versions of SIFT by using different feature extraction algorithms, to deal with the trade-offs between detection performance and resource consumption. Overall, we were able to successfully implement the SIFT approach for ECG sensor-hijacking detection on the Amulet platform with minimal computational, memory, and battery overhead. Our analysis shows that we achieved a detection performance comparable to our gold-standard version on MATLAB. Finally, we provide some insights into what capabilities, constrained WIoT platforms should provide developers so as to make the inclusion of data-driven security primitives on such systems manageable.

II. Preliminaries

The goal of this paper is to understand the challenges in implementing and deploying a complex approach for detecting sensor-hijacking attacks – SIgnal Feature-correlation-based Testing (SIFT) – on a resource-constrained wearable IoT platform (WIoT). Our platform of choice is the Amulet platform [7]. Before we proceed to the deployment details, we provide a short overview of SIFT approach applied to detecting ECG sensor hijacking attacks, and the Amulet platform including its capabilities.

A. SIFT for detecting ECG sensor hijacking

There exists several physiological signals that measure the same underlying physiological process, e.g., electrocardiogram, blood pressure, blood volume pulse etc. are measures of the cardiac process. SIFT approach aims to detect sensor hijacking attack by leveraging the idea that multiple physiological signals of the same underlying physiological process are inherently related to each other [6]. In this sense, SIFT utilizes *signal-level redundancy* to detect attacks, as opposed to redundancy at the device level, which would require redundant sensors as part of the WIoT. SIFT's strategy is particularly useful in WIoT context as deploying redundant sensors is not always feasible for wearability and usability reasons. Specifically, in this paper we focus on using SIFT for detecting the hijacking of ECG sensors and the resulting measurement alterations by using arterial blood pressure (ABP) as reference. We assume that the ABP signal is trustworthy and cannot be tampered by adversaries. Therefore, any unilateral change in ECG signal can be detected. Figure 2 shows an overview of SIFT for detecting ECG sensor hijacking. SIFT has three steps, which we describe below.

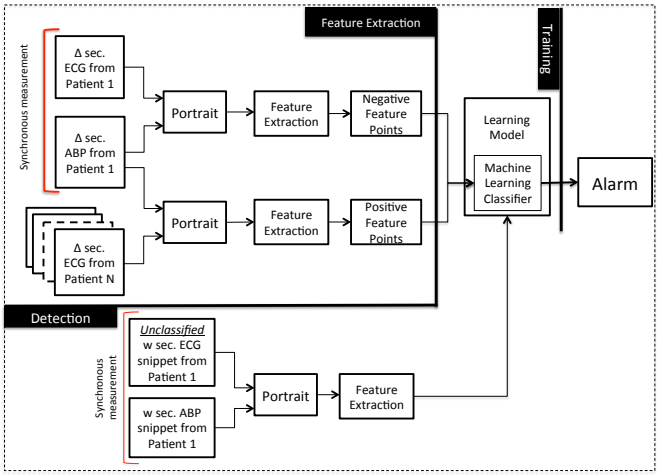

Fig. 2: **Detecting Alterations of ECG Sensor Measurements**

Feature Extraction: In the feature extraction step, w time-units synchronously measured ECG and ABP signals are first transformed into a two-dimensional normalized form called a *portrait*. The portrait captures the shape of the ECG and ABP signals measured in tandem. Let $a(t)$ and $e(t)$ be the normalized ABP and ECG signals at time t, where $0 \leq t \leq w$. Then a 2-dimensional portrait P is generated through the function $f(t) = (a(t), e(t))$. Once the portrait is built, we extract a total of eight features from it. We categorize these eight features into two classes: *matrix features* and *geometric features*. Matrix features are generated based on viewing the portrait as an $n \times n$ grid and counting the number of points from the portrait that fall into each element in the grid. Formally, we view the grid as a $n \times n$ matrix, C, in which each element $c(i, j)$ is the number of points in the corresponding grid element (i, j), where $i, j \leq n$. We chose $n = 50$ for generating the matrix C. Geometric features, on the other hand, describe the absolute and relative location of certain characteristic points (like R peaks in ECG and Systolic peaks in ABP) of the signals in the portrait. Table I shows the 8 features that are used and described in [6].

TABLE I: Original Feature Summary

Type	Feature
Matrix Features	Spatial Filling index of matrix C
	Standard deviation of column averages of the matrix C
	Area Under the Curve (AUC) formed by the matrix C column averages
Geometric Features	Average of the angles between R peaks on the portrait
	Average of the angles between Systolic peaks on the portrait
	Average distance between R peaks and the Origin on the portrait
	Average distance between Systolic peaks and the Origin in the portrait
	Average distance between R peaks and the corresponding Systolic peaks in the portrait

Training step: The next step is the training step where we build a user-specific model for the wearer of the WIoT. In this regard, we collect Δ time-units of synchronously measured

ECG and ABP signals from the user for whom we are training our model. The feature extraction for the negative class points is done by a sliding window of size $w < \Delta$, over the time-series of the ECG and ABP signals. Each w time-units window of data produces one portrait, and one 8-dimensional feature point is then generated from this portrait. We categorize our feature points into two classes, positive and negative. The negative class feature are obtained from portraits obtained from Δ time-units of ECG and ABP signals from the user. On the other hand, the positive class points are generated using portraits from Δ time-units of the wearer's ABP and ECG belonging to several different users, and then extracting 8-dimensional features by sliding a window of size w over the time-series. We use Support Vector Machine (SVM) as the machine learning algorithm of choice for training the user-specific model. We chose SVM as it performed the best among the algorithms we tried. Further, it is also well-understood and has excellent tool support [6].

Detetction step: Once the model is trained for a particular user, then it can decide if any newly received snippet of ECG measurements have been maliciously altered. For every newly received w time-units ECG and ABP signals from the user, it generates a portrait and extracts the 8-dimensional feature point from this portrait. Then, this feature point is fed into the user-specific model. The model will then output a positive or negative label for this feature point. If the feature point is deemed to be positive, then this w second ECG signal snippet is considered to be altered and an alert will be generated.

B. Amulet platform

We briefly introduce the Amulet platform [7] on which we deployed our SIFT detector. Currently, the Amulet platform consists of a prototype device in a smart watch form factor with operating system and toolchain support, for energy efficient applications. We selected Amulet platform as our platform as it meets several requirements of a base station. (1) The Amulet platform allows multiple applications from different third party developers to be deployed on the same device. (2) It runs a secure operating system called Amulet OS. The Amulet OS provides capabilities for isolating applications from each other and from the system, which protects sensitive user information (such as health data) and prevents applications from interfering with the system or other applications. (3) It is designed to operate in a resource-constrained environment and has a small footprint, which allows it to operate for weeks without recharging. (4) It comes with an OS and associated tool-chain that provide compile-time predictive analysis of resource usage, including energy and memory.

Currently, the Amulet wearable prototype is mostly a single-board system. It comes with a main board, battery, haptic buzzer, and secondary storage board are all housed in a custom-designed 3D-printed case that fits a standard 22mm off-the-shelf watchband. Texas Instruments (TI) MSP430FR5989 micro-controller with 2 KB of SRAM and 128 KB of integrated FRAM serves as the main computational device. The wearable is equipped with internal sensors for use by developers: an Analog Devices (AD) ADMP510 microphone, an Avago Tech APDS-9008 light sensor, a TI TMP20 temperature sensor, an STMicroelectronics L3GD20H gyroscope and an AD ADXL362 accelerometer.

AmuletOS is implemented on top of the QM event-based programming framework [8]. It provides a low-power, event-driven programming model, an API, and efficient app isolation and optimization through compile-time techniques. Each application is represented as a state machine with memory. Therefore, there are no processes or threads, all application code runs to completion without context-switching overhead. App code, state, and variables are kept in persistent storage. Applications are written in a custom variant of C that removes many of C riskier features: access to arbitrary memory locations (pointers), arbitrary control flows (goto statements), recursive function calls, and in-line assembly. Array access in C is implemented using equivalent pointer operations, however, in AmuletOS, the array syntax is modified so that arrays can be passed to functions explicitly by reference (not as pointers). Furthermore, arrays also have an associated length that allows for run-time bounds checking whenever access behaviors cannot be adequately checked statically.

Each Amulet application includes: (1) a state machine, (2) event handlers (written in Amulet version of C), and (3) attributes specifying the app global variables. The QM framework combines application information into XML-formatted QM files. Amulet Firmware Toolchain translates the Amulet version of C code to safe C code using a modified C grammar and ensures that array and other memory accesses are valid, that problematic integer operations do not occur (e.g., division by zero), and that programming techniques such as recursion, goto statements, and pointers are not employed. After these steps, applications are merged together in a single QM file, which is then converted to C using QM. This code is compiled and linked using Texas Instrument open-source GCC for MSP430. This firmware image can then be installed onto the application chip (MSP430) of Amulet platform.

III. Implementing SIFT Approach on Amulet Platform

In this section, we give a detailed description of the implementation of our detector on the Amulet platform. The goal of the implementation is two fold. (1) demonstrate the feasibility of our detector, a complex data-driven application, on a resource-constrained wearable IoT system, and (2) help improve the design of Amulet to facilitate more diverse, sophisticated applications.

Given that the Amulet platform is a low-capability system, we implemented three versions of the detector with different memory and energy overhead by using the QP framework. We call these three versions as: (1) ***Original version***, which was the full implementation of the detector, as described in the previous section; (2) ***Simplified version***, which simplified the feature extraction algorithms so that it did not utilize the standard C math library; (3) ***Reduced version***, which implemented only the geometric features as part of its feature extraction algorithm. There are several reasons for us to implement the three versions of our detector: (1) To establish the *effectiveness* of our detector on resource-constrained WIoT platforms. (2) To accomodate the possibility of *adaptive security* in WIoT, where we can switch between different versions of the detector depending upon available computational resources.

We use QM event-based programming framework to develop our detector, and each version of our detector consists of three states: (1) *PeaksDataCheck state*; (2) *FeatureExtraction state*; (3) and *MLClassifier state*. Only the Peaks Check state is the same across all the three versions of our detector, while FeatureExtraction state and MLClassifier state are different across all the three versions. We now describe each of the the states.

PeaksDataCheck State: is responsible for fetching ECG and ABP data snippets from the memory that Amulet platform received every $w = 3$ seconds and then displaying them on the LED screen. In our current implementation, we pre-stored ECG and ABP data and their corresponding peak indexes into the memory of the Amulet platform for ease of testing. It is a simple extension to perform these tasks at run-time based on live data. Once the PeaksDataCheck state is done, the 3 seconds ECG and ABP snippets are transmitted to the FeatureExtraction state, which is described below.

FeatureExtraction State: We customized the feature extraction algorithm for each version of our detector. The *original* feature extraction algorithm is a full implementation which is described in Section II. The *reduced* feature extraction algorithm only uses the geometric features from the simplified case. Consequently, we focus on the *simplified case* in our description of the feature extraction state.

In the simplified case, all three matrix features are generated based on the matrix C. The calculation of the *Spatial Filling Index* feature is the same in both simplified and original case. For the remaining two matrix features, we simplified the way that we generated them. Instead of calculating the standard deviation of column averages of matrix C, we use the variance of column averages of matrix C, which avoids using the square root computation. Further, to compute the AUC of the column averages in matrix C, we originally performed numerical integration via the trapezoidal method. To implement this in C code, we simplified the process and used the equation $\int_a^b f(x)dx = \frac{b-a}{2N}\sum_{n=1}^{N}(f(x_n)+f(x_{n+1}))$ to calculate the the integral of the curve formed by the the column averages of matrix C.

For geometric features, instead of calculating the angles and the distance for the characteristic points on the portrait, we used the slope and the square of the distance as the features. Consequently, our five new features in the simplified case are: (i) *Average of the slope for the R peaks*, given by $\frac{1}{m}\sum_{i=1}^{m}\frac{y_{r_i}}{x_{r_i}}$, where (x_r, y_r) denotes R peaks in a portrait. (ii) *Average of the slope for the Systolic peaks*, given by $\frac{1}{n}\sum_{i=1}^{n}\frac{y_{s_i}}{x_{s_i}}$, where (x_s, y_s) denotes Systolic peaks in a portrait. (iii) *Average squared distance between R peaks and the origin*, given by $\frac{1}{m}\sum_{i=1}^{m}(x_{r_i}^2 + y_{r_i}^2)$. (iv) *Average squared distance between Systolic peaks and the origin*, given by $\frac{1}{n}\sum_{i=1}^{n}(x_{r_i}^2 + y_{r_i}^2)$. (v) *Average squared distance between R peaks and the corresponding Systolic peaks*, given by $\frac{1}{m}\sum_{i=1}^{m}(x_{r_i}-x_{s_i})^2+(y_{r_i}-y_{s_i})^2$, where (x_{s_i}, y_{s_i}) denotes the corresponding systolic peak for a R peak (x_{r_i}, y_{r_i}). Above, m denotes the total number of R peaks in a portrait and n denotes the total number of Systolic peaks in a portrait.

MLClassifier State: As described in Section II, to build each user-specific model, we fed a set of positive and negative feature points into the SVM classifier with a linear kernel for training purpose. Once the offline training phase is done, we then translate the prediction function of the trained model into C code and implemented the MLClassifier state. Note that each version of the detector has a different feature extraction algorithm, which results in a different set of positive and negative points that are used to train the machine learning model. Thus, for a given user, the MLClassifier are different across all the three versions of detector. The MLClassifier state, therefore, uses a learned user-specific model to predict the label of the generated feature point, which is the output of the FeatureExtraction state. If the label is positive, then it will generate an alert on the LED screen of the Amulet platform.

IV. Performance & Results

In this section, we evaluate the performance of our three versions of our detector on the Amulet platform in detecting ECG sensor hijacking attack that result in the alteration of the ECG measurements. We evaluate our approach w.r.t. two aspects: (1) the performance of our detector in detecting ECG measurement alteration attack with respect to our analysis results on MATLAB; (2) the memory consumption and the battery lifetime of the detector, when implemented on the Amulet platform. All three versions of SIFT achieve above 86% accuracy rate in detecting the malicious alteration of the ECG measurements, while consuming minimal memory and energy.

Dataset: In order to train and test our model, we used data belonging to 12 users from the MIT PhysioBank Fantasia database [9]. We chose these particular subjects from these databases because the availability of both ECG and ABP signals for them. The average age of our users was 46.5 years (with a standard deviation of of 25.5 years). We chose these particular users because of the availability of both ECG and ABP signals for them. We simulated ECG measurement alteration due to sensor hijacking by replacing a user's ECG with someone else's.

Metrics: We use the following metrics to train for our validation: false positive rate, false negative rate and accuracy rate. We define *false positive rate* (FP) as the fraction of the cases in which an unaltered ECG sensor measurement is misclassified as altered. Similarly, we define *false negative rate* (FN) as the fraction of the cases where an altered ECG sensor measurement is misclassified as unaltered. Finally, accuracy rate is the fraction of the cases where an altered or unaltered ECG sensor measurement is classified as such.

A. Performance Analysis

In order to evaluate our detector, for each subject from our dataset, we trained a user-specific model using original, reduced, or simplified features and SVM as our classifier or choice. we chose training time to be 20 minutes as it works best for us based on our original work as shown in [6]. The training is done offline and therefore, need not be done on amulet platform itself. Once the models were trained, we then loaded each of them on the amulet platform as an *app* for evaluation.

In order to simulate the reception of both altered and unaltered ECG measurements at the Amulet platform, we pre-stored 2 minutes of unseen ECG and ABP signals and their peaks indexes into the memory. **By unseen ECG and ABP snippets we mean measurement that were not used for training our model**. Within these 2 minutes of unseen ECG measurements, about 1 minute worth (i.e., 50%) of measurement were altered by replacing it with someone else's ECG snippets. The alteration was done in random locations within the 2 minute snippet. As our detector only need 3 seconds of ECG and ABP snippets to generate alerts, we ended up with 40 test examples in total for each subject. The detector app then fetches 3 seconds worth of unclassified ECG and ABP snippets from our 2 minute-long test data-set for a given subject, extracts features from it and then uses the trained SVM model to identify if the 3 seconds ECG snippet has been altered or not.

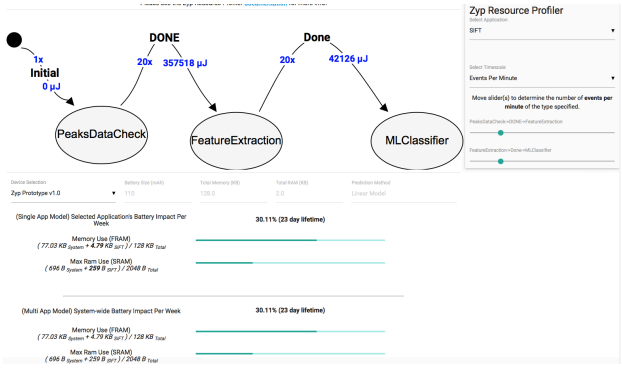

Fig. 3: **Resource Consumption of SIFT app**

Table II shows the comparison of three versions of detector implemented on Amulet platform along with the results for our MATLAB implementation, which forms the gold-standard of the detector. We can see that our original version of the detector reaches 93.06% accuracy, which is very close to the MATLAB implementation. For the simplified version of the detector, the detection accuracy only drops 0.2% for our implementation. This indicates that our simplified features are a good approximation of the original features. However, there is a bigger decrease in detection accuracy when we used the reduced version of the detector. This is simply a function of the fact that we are only using a portion of the features in the reduced cases. Furthermore, we can see that our app implemented on Amulet platform performs as well as or even slightly better than the simulation on MATLAB, which demonstrates that our implementation is accurate.

TABLE II: Performance Evaluation for Three Versions of Detector

Version	Platform	Avg. FP	Avg. FN	Avg. Acc	Avg. F1[1]
Original	Amulet	0.83%	12.50%	93.06%	92.77%
	MATLAB	5.83%	10.23%	91.97%	91.97%
Simplified	Amulet	6.67%	7.58%	92.86%	93.43%
	MATLAB	5.00%	12.88%	91.06%	90.28%
Reduced	Amulet	12.08%	15.15%	86.31%	87.10%
	MATLAB	22.08%	14.39%	81.76%	84.04%

B. Energy Analysis

To evaluate the resource usage of three versions of the detector app, we used the Amulet Resource Profiler front end, ARP-view, to gain insight into the energy and memory efficiency of our SIFT detectors. ARP-view presents developers a graphical view of the resource profile and sliders that allow them to see the battery-life impact when they adjust application parameters. ARP-view leverages the Amulet Resource Profiler's fine-grained data about the structure and behavior of applications. Amulet Resource Profiler captures information about each app's code space and memory requirements, using a combination of compiler tools and static analysis. To profile energy, Amulet Resource Profiler builds a parameterized model of the app's energy consumption. Fig 3 shows a snapshot example of the resource consumption of original version of detector app. Table III shows the memory usage and expected lifetime with the 110mAh battery for three versions of detector apps. We can see that the simplified version consumes 16% less memory than the original version for the detector alone with a nominal reduction in system memory usage. The reduced version, on the other hand, consumes almost 50% less memory than the other original version for the detector. In terms of the expected lifetime, the reduced version of our detector lasts the longest among the three versions, where the expected lifetime is 55 days compared to the original and simplified models which have about half the lifetime. Overall, it can be seen that we were quite successful in implementing our complex data-driven detectors on a limited capability Amulet base station in our WIoT environment.

[1]F1 score can be interpreted as a weighted average of the precision (the number of true positives divided by the total number of elements labeled as belonging to the positive class) and recall (the number of true positives divided by the total number of elements that actually belong to the positive class).

TABLE III: Resource Usage of Three Versions of Detector

Version	Resource Type	Measurements
Original	Memory Use (FRAM)	77.03 KB_{System} + 4.79 $KB_{detector}$
	Max Ram Use (SRAM)	696 B_{System} + 259 $B_{detector}$
	Expected Lifetime	23 days
Simplified	Memory Use (FRAM)	71.58 KB_{System} + 4.02 $KB_{detector}$
	Max Ram Use (SRAM)	694 B_{System} + 259 $B_{detector}$
	Expected Lifetime	26 days
Reduced	Memory Use (FRAM)	56.29 KB_{System} + 2.56 $KB_{detector}$
	Max Ram Use (SRAM)	694 B_{System} + 69 $B_{detector}$
	Expected Lifetime	55 days

V. Experiences & Insights

One of the main goals of this work was to implement our detector on low capability WIoT platforms and suggest potential improvements to developers of such platforms based on our experiences. In this section we list some of the main insights and suggestions for the WIoT platforms when it comes to implementing data-driven, complex security "apps" on them.

Insight #1: *Have efficient sensor data pipelines:* Many wearable systems require their sensors to continuously measure the user's various physiological signals at a high sampling rate and transmit the measured data to the base station for processing. Further, many apps on WIoT base station may need to initialize and use large sized arrays. However, the resource-constrained base station in the WioT environment may have small non-volatile memory available (e.g., 128KB for the Amulet), which presents a considerable impediment to designing sensor-data rich apps. For instance, to be able to execute our detection algorithm, the 3 seconds ECG and ABP data had to be stored into two floating type arrays (each has a size of 1080) temporarily in the memory. One major problem we encountered with the Amulet platform was that it does not allow large array size nor did it support 2D arrays. Further, the QM software did not provide the functionality of initializing a global array. One possible solution is to this problem is to use in-built SD card on base station. However, this solution will negatively impact both the energy consumption and I/O delays. Another possible solution is performing the feature extraction on sensors. Therefore, the data stream transmitted to base station are features, which are compressed form of the raw data [10]. However, this solution requires the physiological sensor has the ability to process the data, which may not be possible. Consequently, efficient sensor data pipeline and management needs to be provided by the WIoT base station platform so that we may carefully buffer real-time sensor measurements and allocate appropriate memory for it.

Insight #2: *Provide basic data processing capabilities:* The main focus of low capability WIoT platforms is often to process the sensor data, and give a decision to the user (such as fall detection, security breach alerts, etc.). In the Amulet platform, many basic data processing capabilities were not available, often for security reasons, which makes it very hard to develop apps like ours. For instance, we wrote our own APIs for the AmuletOS that convert the string to float, float to string. In addition, data-driven based solutions like ours are heavy users of mathematical operations. Earlier versions of the Amulet platform did not support C math library nor provide any kind of math APIs, which made our task very difficult. Further, with the Amulet platform we also faced the problem that the mathematical operation between array elements were often calculated incorrectly. To solve this problem, we first stored the value of the array into a temporary variable. Then, instead of directly using the array to do the calculation, we use this temporary variable to do the calculation, which was very inefficient. Consequently, the operating systems that runs on such WIoT base station needs to provide built-in support for FFT or audio processing API, mathematical operations, and simplifying display of formatted number strings.

Insight #3: *Provide efficient debugging tools:* Implementing complex apps on WIoT platform requires that we are able to debug them effectively. The lack of good debugging tools seriously reduces the efficacy of the app developer. For instance, to debug the application code for Amulet, one can either use GDB or manually use the provided API to show the variable values on the LED display. The current debug mode (GDB) for Amulet is unstable, and it got crashed very frequently. In our implementation, we had to use the second method to debug. However, this proved to be very inefficient as to be able to see the value on the LED screen one had to compile the code and repeatedly flash it into the Amulet. Consequently, platform developers need to provide good debugging tools, for instance, showing the resource consumption of the application, showing where and how the sensor data is being transformed, providing a desktop based simulator that emulates the screen writing.

Insight #4: *Enable adaptive security in WIoT:* Unlike the traditional mobile computing devices (such as smartphone), resource-constrained Wearable IoT base station like the Amulet have limited capabilities such as computational power, memory space and battery power. To accommodate the diversity in hardware and software of the low-capability WIoT platforms, we developed three different versions of SIFT. However, currently only one version of SIFT is manually flashed into the Amulet device. This is not really practical because: (1) the decision of deploying which version of SIFT has to be made beforehand; (2) the computational resource availability changes overtime, and the Amulet device has to be flashed every time when switching to another version of SIFT is needed. Thus, we envision an adaptive security model with the ability to automatically adjust the security level by switching between different versions of one security app based on the available resources. This model considers two types of resource constraints: 1) *static constraints*, which exists in the compile time, such as the memory, available library, available API and etc. 2) *dynamic constraints*, which exists in the runtime, such as the memory, CPU cycle, battery power and etc. The core of this model is a *decision engine*, which can automatically detect any types of constraints during compile time and runtime, and decide which version of security app to run based on the detected resource constraints. Therefore, in our future work, there are essentially two questions that need to be answered: (1) How to detect the static and dynamic constraints during compile time and run time, respectively? (2) Based on the detected resource constraints, how to decide which version of the security app to switch to? The answer to these two questions will require that the WIoT base station be capable of providing information about it's energy and computational state to the app at runtime.

VI. Conclusion

In this paper we presented our experiences while implementing a data-driven security solution, SIgnal Feature-correlation-based Testing (SIFT), for detecting electrocardiogram sensor-hijack attack on a constrained wearable internet-of-things (WIoT) platform called the Amulet. We showed the inherent trade-offs between the performance and detection accuracy that we had to navigate during the implementation. To deal with this trade-off, we totally developed three different versions of our detector. We demonstrated that each version of our detector could be implemented on a constrained wearable platform (Amulet), was energy efficient, and was accurate. We believe that the challenges and trade-offs we faced are generalizable to the realization of other complex security solutions in resource-constrained WIoT environments. In this regard, we provided some insights into the capabilities that the WIoT systems should provide developers to make the inclusion of data-driven security solutions easier.

Acknowledgements

We would like to thank David Kotz and Ron Peterson from Dartmouth College, Jacob Sorber from Clemson University, along with Alex Witt, and Ahmad Moghimi from Worcester Polytechnic Institute who helped us with this work.

References

[1] D. Halperin, T. Kohno, T. Heydt-Benjamin, K. Fu, and W. Maisel, "Security and privacy for implantable medical devices," *Pervasive Computing, IEEE*, vol. 7, no. 1, pp. 30–39, Jan 2008.

[2] Dan Goodlin, "Insulin pump hack delivers fatal dosage over the air," http://www.theregister.co.uk/2011/10/27/fatal_insulin_pump_attack/, October 2011.

[3] "Advisory (ICSA-15-090-03), Hospira MedNet Vulnerabilities," https://ics-cert.us-cert.gov/advisories/ICSA-15-090-03.

[4] N. Brown, N. Patel, P. Plenefisch, A. Moghimi, T. Eisenbarth, C. Shue, and K. K. Venkatasubramanian, "Scream: Sensory channel remote execution attack methods," in *Usenix Security Symposium*, August 2016.

[5] D. Foo Kune, J. Backes, S. S. Clark, D. B. Kramer, M. R. Reynolds, K. Fu, Y. Kim, and W. Xu, "Ghost Talk: Mitigating EMI signal injection attacks against analog sensors," in *Proceedings of the 34th Annual IEEE Symposium on Security and Privacy*, May 2013. [Online]. Available: https://spqr.eecs.umich.edu/papers/fookune-emi-oakland13.pdf

[6] H. Cai and K. K. Venkatasubramanian, "Detecting signal injection attack-based morphological alterations of ecg measurements," in *Distributed Computing in Sensor Systems (DCOSS), 2016 International Conference on*. Springer, 2016.

[7] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord, R. Halter, D. Kotz, and J. Sorber, "Amulet: An energy-efficient, multi-application wearable platform," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16, 2016, pp. 216–229.

[8] L. Quantum Leaps. Qp/c framework. [Online]. Available: http://www.state-machine.com

[9] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[10] K. Lorincz, B.-r. Chen, G. W. Challen, A. R. Chowdhury, S. Patel, P. Bonato, M. Welsh *et al.*, "Mercury: a wearable sensor network platform for high-fidelity motion analysis." in *SenSys*, vol. 9, 2009, pp. 183–196.