Toward Predicting Secure Environments for Wearable Devices

Charles Walter, Ian Riley, Xinchi He, Ethan Robards, Rose F. Gamble
Tandy School of Computer Science, The University of Tulsa, Tulsa, OK
{charlie-walter, ian-riley, xinchi-he, ethan-robards, gamble}@utulsa.edu

Abstract

Wearable devices have become more common for the average consumer. As devices need to operate with low power, many devices use simplified security measures to secure the data during transmission. While Bluetooth, the primary method of communication, includes certain security measures as part of the format, they are insufficient to fully secure the connection and the data transmitted. Users must be made aware of the potential security threats to the information communicated by the wearable, as well as be empowered and engaged to protect it. In this paper, we propose a method of identifying insecure environments through crowdsourced data, allowing wearable consumers to deploy an application on their base system (e.g., a smart phone) that alerts when in the presence of a security threat. We examine two different machine learning methods for classifying the environment and interacting with the users, as well as evaluating the potential uses for both algorithms.

1. Introduction

There has been a marked increase in the demand for wearable devices by the average consumer, from fitness bands and blood pressure monitors to sweat sensors and headphones. These devices communicate through Bluetooth to a base station, often a phone, to transmit data. Their proliferation by a multitude of companies and the speed at which they are entering the market means that security mechanisms may be deferred to later product releases. The lack of security consideration means these devices and/or their connections are more likely to be attacked or have data stolen.

The security of Bluetooth devices primarily relies on attackers being unable to follow the communication pattern. However, if an attacker can capture the initial pairing messages between a wearable and its base station, they may be able to follow the full hop pattern. The pairing packets also include the keys used to decrypt additional data packets, if the devices use encryption. Acquiring the keys allows hackers to intercept all data communication, assuming they can get the hop pattern. Ryan [12] was able to overcome this constraint, showing that the encryption that Bluetooth uses can be bypassed nearly instantly and that the hop pattern could be calculated with only minimal packets intercepted by chance.

Consumers are largely unaware of security risks, widely adopting a somewhat childish “If I can’t see you, you can’t see me” approach to securing their information [6]. This inattention becomes a much larger problem when data is being transmitted via Bluetooth. Most Bluetooth devices use only the security measures already in the specification itself, such as the default encryption and the rapid change of communication frequency. However, there are mechanisms that can sniff Bluetooth packets and provide a method for man-in-the-middle and denial of service attacks. [12].

Because wearables collect significant amounts of data, from accelerometer data to medical information such as heartrate, they can provide an attacker with very detailed information about the wearer. For instance, it has been shown that accelerometer data generated by hand movements as captured by a wearable (e.g., a smart bracelet or watch), could be intercepted by an attacker and potentially calculate a user’s PIN [16]. Because of the possibility of intercepting Bluetooth communication, it is increasingly important to have a method to prevent data leakage. Ideally, new security mechanisms should be compatible with existing Bluetooth devices, as well as become deployable on future devices.

As there are large numbers of Bluetooth devices currently in the hands of consumers, with 3 billion being sold in 2014 [1], there exists the potential to use crowdsourcing techniques to collect obfuscated user data. This data can provide researchers with the ability to discover patterns based on wearable usage given the basic information that the devices already collect. When dealing with large datasets, machine learning is often used to discover patterns quickly

URI: http://hdl.handle.net/10125/41321
ISBN: 978-0-9981331-0-2
CC-BY-NC-ND

and effectively. Utilizing such techniques could lead to a greater understanding of user environments and the ability to adapt to unexpected security concerns of new environments.

We previously investigated methodologies to secure wearable connections by preventing data from being sent between devices when the devices are in an insecure location [15]. This research involved examining passive and active wearables and the different ways in which they submit data to the associated base station. Given the differences, we determined that the best overall method was to force the wearable to send empty packets. To illustrate the methodology, we deployed an iPhone app with embedded rules for pre-defined insecure environments, such as using the GPS coordinates of a university office. This allowed us to ensure that, in a potentially insecure environment, the app could automatically cause the wearable to send empty packets and not disconnect. The app deployed only static rules, whose modification had to be performed programmatically. While the manual construction and deployment of such rules was not meant for a wearable consumer, it did provide us with a means to communicate with the wearable using this form of technology.

In this paper, we expand on the original concept of using an adaptive base station app, by exploring the use of a cloud service and two machine learning algorithms using data that is simulated as if crowdsourced from users. One algorithm generates a series of rules, allowing users to receive updated rules automatically to adapt the app, and consequently, their wearables, to recognize new, potentially insecure situations as they are identified. The second algorithm classifies ranges of sensors and their combinations as potentially insecure in real-time and can respond to an application's query as to the probability of the wearable information being insecure. We detail the algorithms and their integration with an extended app. We then evaluate the benefits and disadvantages of each machine learning algorithm, including how often they are correct when given additional testing data.

2. Background

Existing Bluetooth security has been shown to be lacking in many key areas. Ryan [12] created a method to brute-force the key exchange protocol after intercepting the pairing packets. This allowed him to decrypt all additional communication sent between the devices, illustrating a need for additional security which relies on something beyond standard encryption.

Diallo, et al. [4] attempted to secure the pairing process by creating a table of trusted MAC addresses for known devices which included a temporary private key to encrypt only the pairing process. Because this method requires both devices to keep their own table, device manufacturers would have to incorporate it into their design process. This method is effective at securing all communication, though it is unreasonable to expect manufacturers of Bluetooth devices to universally adopt this strategy.

Two examples illustrate the need for wearable security awareness. Pan, et al. [11] were able to intercept and use data from a Bluetooth mouse to recreate passwords input through an onscreen keyboard. Wang, et al. [16] used the internal accelerometer on a smart bracelet to recreate PINs input at an ATM. Using Bluetooth keylogging and password recreation in this way makes it impossible to check if your information is being intercepted, implying the need for embedding technology that can stop wearable communications from being intercepted.

Our previous work [15] investigated how this security might be added by forcing Bluetooth devices to send empty packets when in a pre-defined (static) environment known by the base station to possibly be insecure. This method relied on embedding static rules which could not be modified by the user. This investigation led us to the current research presented in this paper in which we design a more robust system by incorporating machine learning techniques and by crowdsourcing data collection.

Machine learning algorithms have been shown to achieve accuracy with relatively small datasets. Moreno, et al. [10] were able to combine labels from crowdsourced data with an accuracy of 89% or above with datasets of only 500 labels and 200 users. This method would allow us to discover new clusters as they appear with relatively few reports of issues, which is important for any security application.

Machine learning algorithms have been deployed directly on wearables and smartphones. Lane, et al. [8] examined four algorithms which used minimal processing power to not adversely affect battery life. They found that these algorithms were able to get good results, but not as good as more processing intensive algorithms which can be used when a device is connected to the internet.

Parallelizing machine learning algorithms allows for more powerful algorithms to be used. Chen, et al. [3] showed that, when using a custom-designed architecture for running neural networks on parallel processors, they could achieve significant improvements in energy requirements and speed. This method is ideal for a cloud-based service which can be run on multiple servers or across multiple processors more easily.

Crowdsourcing and machine learning go hand in hand for many use cases. Because crowdsourcing can deliver large quantities of data quickly and machine learning can identify often unseen groupings in the data, it is natural to combine the two. For example, Minoda, et al. [9] used Amazon mTurk to collect preferred lighting levels of images within a room from a wide range of users of different ethnic backgrounds. Their study was able to determine estimated age and ethnicity of users based on their responses. As another example, Saxe, et al. [13] used code help forums, such as StackOverflow, to map terms found in malware to train a machine learning algorithm to detect the capabilities of specific malware program. They were able to determine if a specific piece of malware contained a given capability, with approximately a 20% false positive or false negative rate.

Crowdsourcing can also be used to improve existing machine learning algorithms by identifying where errors were made. Georgescu, et al. [5] used an algorithm designed to identify important information from academic papers, such as the author, title, and abstract. They found that, despite the added process of humans checking their work, they were unable to get above 90% accuracy. This shows the need, when crowdsourcing data, to understand that humans can be flawed, which can lead to imperfections in the data over which the machine learning algorithms are applied.

Two independent researcher groups investigated crowdsourcing users' smart home preferences in an attempt to improve the experience for all users. Shahriar and Rahman [14] and Bourelos, et al. [2] looked at energy management by crowdsourcing heat sensors and environmental data to optimize heating and cooling and minimize electricity used in homes. Shahriar and Rahman used machine learning to discover optimal clusters. Bourelos, et al. used an algorithm that attempted to minimize each home's electrical requirements without compromising the comfort of the users in the house.

3. Adapting Communication with the Wearables

Bluetooth devices communicate using the Adaptive Frequency Hopping Spread Spectrum. This method makes it difficult to intercept messages from the devices, as the channel they are communicating on changes after every packet. Hence, many Bluetooth devices do not incorporate additional forms of security. However, if an attacker is able to follow the hop pattern, they can intercept all data sent between the two devices.

When wearables pair with a base station, the initial pairing packets are unencrypted, as they must communicate with a device that has not received a public key. This allows an attacker to gain information stored in the pairing packets, such as the hop pattern. Since the main packets are encrypted, it is possible to obtain the key used to encrypt the keys of both devices. Thus, an attacker can eavesdrop and insert packets into the connection.

The issue is typically combatted by the devices remaining paired so they do not send out additional pairing packets. However, remaining paired does not completely inhibit an attacker who has already intercepted the initial pairing packets from eavesdropping or inserting malicious packets. Our experimental application forced a small set of wearables to send empty data packets to an iPhone when in environments that were pre-specified as being insecure [15]. By sending empty data packets, and subsequently halting all data requests made by the phone, the devices maintained the connection, disallowing an adversary to capture the pairing information or obtain any information from eavesdropping.

The original adaptive behavior was triggered only when the device was in an environment that was identified as insecure. The conditions defining an insecure environment were predefined and statically placed within the app to demonstrate its viability in preventing an attacker from connecting to the device themselves or eavesdropping on the packet transmissions. The experimental application successfully prevented data from being sent between the iPhone and selected devices, which included a Metawear R, a Fitbit Charge HR, a Pebble Time Round, and a pair of LG Tone Ultra Bluetooth headphones. However, the application it was unable to adapt to new potential threats or insecure environments. These threats and environments had to be manually inserted, which is impractical for wearable consumers.

Validating the concept led to investigating a system that would allow for adaptation "in-the-wild." There are multiple methods by which this adaption could be accomplished. The simplest method is to allow users to define rules from within the application. This method relies on users being aware of the security issues in their everyday lives and, therefore, is likely not to be adopted. A more feasible approach is to automatically adapt to the potentially changing environments that consumers may find themselves in. In this paper, we apply machine learning algorithms to snapshots that are crowdsourced from security conscious users in order to classify environments as being secure or insecure.

4. Insecure Environment Snapshot Generation

To improve on our experimental application, we focus on creating a system that can connect multiple user experiences and learn what made specific situations insecure. We incorporate a cloud service to aggregate user input and machine learning algorithms to predict if a situation is insecure or not.

To simulate crowdsourcing and evaluate our learning algorithms, we create a service that generates snapshots of sensor values representing an insecure environment. The snapshots are passed to the learning algorithms to represent users telling other users that they believe the environment as described by their sensor values to be insecure. The concept is similar to WAZE, a community-based app for traffic and navigation, where users communicate to a cloud service regarding traffic delays and the alternative routes they are taking. Similarly, the snapshot generation allows the algorithms to learn sensor ranges and combinations potentially related to insecure situations that can then be communicated to other users.

4.1. User defined snapshots

The application normally generates snapshots in 5 minute increments. The only exception to this timing is if a user explicitly tells the app that they feel their data may be insecure. At the time this action is taken, the app will generate a snapshot explaining that it is in an insecure environment, and will remain in this state until the user tells the app they are again in a secure environment.

Each snapshot is labeled as pertaining to a secure or insecure environment at the time of generation. This information can be used to both train the model to improve its accuracy and evaluate the current model's effectiveness. The snapshots are stored locally until the user is in a position where they can upload the snapshots to the cloud service. Users may choose to not send data when in an insecure environment, as well as to only send data when their device is connected to Wi-Fi to avoid using cellular data. The cloud service stores snapshots from all users who use the application. By crowdsourcing their data, we aim to acquire a large number of data points for each potentially insecure environment. Users are not burdened with defining rules to secure their devices. Additionally, users allow other security conscious users to help define insecure conditions and situations with just the press of a button.

```
{
 "Devices": "Pebble",
 "Heart Rate": "136.0",
 "Time": "1377.0",
 "Speed": "20.724903281615532",
 "Latitude": "35.72388775811967",
 "Longitude": "-95.94220940565249",
 "Temperature": "4.9E-324",
 "Insecure": "true"
}
```

Figure 1: Sample Snapshot

An example snapshot is depicted in Figure 1. Note the value for temperature, which is used when a Bluetooth device which provides that data is unavailable. We have limited our snapshots to only include data that can be taken from devices we have access to. However, it is simple to add additional variables to these parameters when new information becomes available. In the case of the above snapshot, the user has said that they are insecure in their current situation, so the application is sending all available data to attempt to learn the reason.

This approach relies heavily on a user with an understanding of their Bluetooth device communication safety, which may not always be possible. We expect that a user of our application initially will be security conscious enough to be fairly accurate with their choice of secure/insecure environments. As more users are added, the existing rule base should help give those users an understanding of secure and insecure environments, allowing them to become more aware of their wearable data security in their everyday lives.

By crowdsourcing, we are able to generate an extensive list of potentially insecure environments. To beat this system, an attacker could change locations in order to set up in an environment that may currently be considered secure. However, there are other conditions that an attacker relies on that can identify potential insecure environments, such as the presence of public WiFi. Thus, if an attacker was to relocate, application users eventually would begin reporting this new environment as insecure, making the move only a temporary solution for the attacker.

4.2. Automatic generation

To test our approach and to obtain sufficient initial data for the machine learning algorithms, we needed to create a snapshot generator that takes a set of basic insecure environments and generates a significant number of snapshots, both secure and insecure, with loosely clustered data points representing insecure environments.

We defined insecure ranges for each variable. For the purposes of this test, these variables had consistent but somewhat arbitrary values. In a real world situation, these values would be very carefully controlled such that they mimic an actual known insecure environment. The algorithm generates a value for that variable. As some of the variables will not always be available to the device, such as temperature and heart rate, we only generate those values in approximately 30% of the snapshots.

Next the algorithm checks if that value is within the insecure range we had previously defined. If so, the chance that the snapshot will be defined as insecure increases. To better simulate human interaction with the system, we include a base chance that each snapshot that is generated is insecure. This is a low value to allow for snapshots which have values within the predefined secure ranges to be more commonly defined as secure. The chance that a snapshot is insecure can be modified prior to generation as needed to better simulate real environments or to force much clearer clustering.

For initial testing, automatic snapshot generation was necessary to ensure there were enough data points for evaluation in Section 6. Each algorithm was trained on identical datasets generated in this manner. However, automatic snapshot generation also presents the possibility of introducing known insecure situations by generating multiple copies of the same insecure environment. This possibility can lead to the rapid response of the system in the event of a sudden and clear insecure environment.

As in the user generation of snapshots, each snapshot includes a value if the snapshot is within a secure environment.

5. Machine Learning

We investigated two different machine learning algorithms to provide situational awareness with respect to wearable data security. The first algorithm, a rule-based approach, generates a tree from which rules can be derived that dictate when the wearable should send only empty packets. These rules can be pushed to our application as periodic updates. This method allows users to decide when their applications are updated, an important consideration for a phone application with limited data or service. The second algorithm, a cloud-based approach, performs machine learning using the Microsoft Azure Machine Learning Studio. This method requires the application to be in constant communication with the server, since the application needs to send each snapshot to the service to determine if it is in a safe environment. We discuss in more detail the training and results of these two methods in Section 6.

5.1. Rule-based Approach

For the rule-based approach, we created a web service to collect data from multiple users into a single repository comprised of snapshots of the wearable sensors. These snapshots are used to train the machine learning algorithm towards identifying insecure environments. The web service also stores the resulting rules, which are sent to devices to adapt them to notify wearables when in an insecure setting. The architecture of this method is shown in Figure 2. We provide more of the programmatic details below.

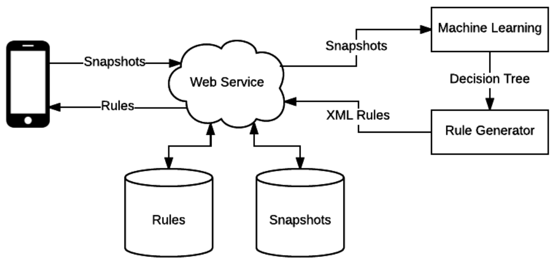

Figure 2: Rule-Based Architecture

The web service was entirely written in-house in Python, relying on a single R script to perform statistical calculations on the data that has been saved to disk. Our mobile app generates snapshots via user input from presumably insecure environments, as well as in known secure environments, which are sent to the web service. The web service parses the request and converts the resulting data into a single CSV file to be used by the learning algorithm. This method potentially limits the number of simultaneous requests the service can currently accept, but the bottleneck can be reduced by incorporating a database which is periodically exported to a CSV file. Using a single CSV file reduces the application complexity as the file can be directly accessed by the machine learning algorithm.

Once sufficient data has been collected, we train the model using the full CSV file. After initial training, if there are minimal requests by users for updated rules or if there are not enough new snapshots being generated, the learning algorithm will suspend processing until a predefined threshold is reached so that it can properly retrain and not introduce rules which may be temporary or may lower the effectiveness of important rules.

When training the model, our web service calls an R script to execute the Random Forest learning algorithm given either the created CSV file or a CSV file with training snapshots as input. The program learns which label and range is able to predict an insecure environment within the snapshots and uses that to generate a conditional inference tree, or ctree [7], as output. A ctree is a two-class decision tree which partitions the data such that the leaf nodes of the tree are the probability that the branch is insecure and that each internal node describes the features of the snapshot. The tree is created using a genetic algorithm, set to terminate when the data can no longer be subdivided. This allows us to generate rules by simply reading the branches of the tree. For numeric values choosing the left branch will result in a rule which is less than or equal to that value, while taking the right branch results in a rule which is greater than the value. An example tree can be seen in Figure 3. Note in this tree, the node labeled by *[2]* is connected to a leaf node of probability .226. The rule generated by this can be seen in Figure 4. Figure 5 shows the branch *[3][4][6]* converted to rules.

The resulting ctree can be used to give a prediction of whether or not a given snapshot is insecure by following the tree nodes until a leaf node is encountered and then comparing the confidence values against a pre-specified threshold. If the values are greater than the threshold, the snapshot being checked is most likely insecure. Otherwise the snapshot is considered secure. We test a variety of thresholds in Section 6.

We chose to use a ctree because we wanted to create a set of rules to determine whether or not a user is insecure rather than to have a predictive model which would require input data to make an evaluation of whether or not the user is actually in an insecure situation. This approach was chosen to prevent users from needing to constantly poll a trained model to determine if their data is insecure. Ctrees are also more resistant to noise. Any data points that fall outside of the insecure clusters cannot affect the cluster so the rules do not change. The only way to break up a cluster is to have a significant number of safe points to be plotted inside of the cluster, which would cause the integrity of the cluster to come into question anyway.

```
[1] root
|   [2] Speed <= 10.3458: 0.226
|   [3] Speed > 10.3458
|   |   [4] Temperature <= 50
|   |   |   [5] Heart.Rate <= 140: 0.119
|   |   |   [6] Heart.Rate > 140: 0.234
|   |   [7] Temperature > 50: 0.191
```

Figure 3: Ctree format

```
<Policy>
    <Device id="All">
        <Rule>
            <Type>Speed</Type>
            <Op>LE</Op>
            <value>10.3458</value>
            <priority>.226</priority>
        </Rule>
    </Device>
</Policy>
```

Figure 4: Sample XML Rule

```
<Policy>
    <Device id="All">
        <Rule>
            <Type>Speed</Type>
            <Op>GT</Op>
            <Cond>and</Cond>
            <value>10.3458</value>
            <priority></priority>
        </Rule>
        <Rule>
            <Type>Temperature</Type>
            <Op>LE</Op>
            <Cond>and</Cond>
            <value>50</value>
        </Rule>
        <Rule>
            <Type>HeartRate</Type>
            <Op>GT</Op>
            <value>140</value>
            <priority>.234</priority>
        </Rule>
    </Device>
</Policy>
```

Figure 5: Combined Rules from Ctree

A ctree is a not a traditional decision tree. It has all of the advantages and disadvantages of a traditional decision tree, but the features are selected using statistical inferences at each step to make a reasonable educated guess for the features it chooses, as opposed to the traditional algorithm which guesses which dimension to divide on at each level, then checks its accuracy and retries as needed. This speeds up the training time, while retaining the same level of accuracy.

Once the model has been trained and the tree has been created, the tree is iterated over using a traditional depth-first search algorithm. At each leaf node, we get a certainty from 0-1 for the decision present in that branch. If the certainty is high enough and the decision is 1, the path is translated to an intermediate language that we have designed for the purpose of representing a set of conditions which must all be true for a decision to be made. The lower bound of certainty is determined beforehand based on the risk we are willing to allow our users to take. This value can be adjusted by the user from within the app as needed. We look at optimal values from our initial tests in Section 6.

The resulting language is then parsed according to the original XML rule schema specified in [15] that works within the existing app. An example of the algorithm output can be seen in Figure 3. Each line is parsed by the algorithm individually, then turned into a rule. In the case of the tree in Figure 3, this means creating a rule based around Speed first, then taking both branches. The left branch is a leaf node, which gives us the rule seen in Figure 4. The right branch leads to two temperature options, one of which has additional children. An example of one of these completed branches can be seen in Figure 5.

After parsing, the rules are stored in a database on the web server. When a request is made to the service, the list of rules is returned to the requester. This allows users to get updated rules at their convenience. As previously stated, the tree and, thus, the rules are resistant to noise, so the integrity of the rules is preserved regardless of how long the user reports snapshots without receiving the latest update.

5.2. Cloud-based Approach

The previous approach required the app to be periodically updated to reflect potential insecure environments by pushing the rules to the app. An alternative approach is to poll a web service to determine if the current state is insecure. In this method, snapshots are sent to the Microsoft Azure cloud where the Machine Learning studio is used to deploy an algorithm to determine whether or not an environment is insecure. Because of the cloud nature of the algorithm, it can use snapshots that it receives to continue to learn new, potentially insecure environments constantly, adding and retraining on each new snapshot that it receives.

The more simplistic architecture of this approach can be seen in Figure 6.

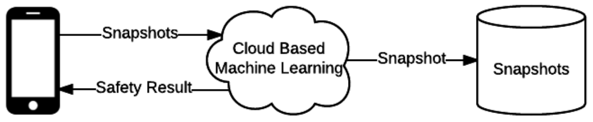

Figure 6: Cloud-based Architecture

Using Azure ML studio, we trained a Decision Forest Regression model to evaluate a set of snapshots to determine how likely each snapshot is to be from an insecure environment. The Decision Forest Regression algorithm was chosen because it is similar in function to the methods used in the rule-based approach, is considered to have good accuracy, and is fairly fast to train. This is because the regression algorithm uses multiple decision trees, all trained concurrently, which results in a model that aggregates over all the trees to find a Gaussian distribution close to the distribution of all the trees taken individually. Ideally, these results could be taken from Azure ML Studio and converted into rules in much the same way as the rule-based approach. However, Azure ML Studio does not support the retrieval of the tree, preventing us from taking the results and converting them to a series of rules. Azure ML Studio makes it very simple to create a web service to handle individual requests for data to be scored, allowing us to use the system to evaluate snapshots as they are generated by a user.

Azure ML Studio requires a CSV file for input, which allowed us to use the same file to train this model as was used in the rule-based approach. Unlike the rule-based approach, Azure ML Studio requires data to use for testing the model and requires the output of the algorithm to be in CSV format.

The web service takes in a snapshot and reports whether or not the snapshot is insecure, with a certainty ranging from 0 to 1 which represents how likely the algorithm thinks the snapshot is insecure. To train the web service, we uploaded a training CSV file. This file is automatically parsed by Azure ML Studio into an internal format that Azure ML Studio uses. This data is then fed into a tuning module which takes the machine learning algorithm and trains a model. The resulting model is then deployed via another web service, as seen in Figure 7. This service takes the model and compares it with the input from the web, outputting the result to the requester. Azure ML Studio requires that a sample set of snapshots is included so that it knows the format the model is trained on, and this is included with snapshots2.csv.

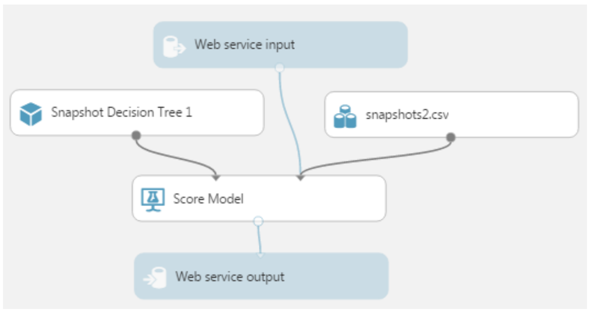

Figure 7: Employing Web Services

Similar to the rule-based approach, this model predicts if a snapshot given to it is insecure by checking if its confidence in that choice is above a specified threshold. We test a variety of thresholds in Section 6.

6. Evaluation

To evaluate our solution, we trained both machine learning approaches on the same set of one hundred thousand generated snapshots. The trained models were then tested against an additional one hundred thousand generated snapshots, and scored at different levels of confidence. To start, we trained each model on a base set of snapshots which had predefined clusters. These clusters were arbitrarily chosen, but would appear in a real life setting. The generated training set had a 10% chance that every snapshot would be insecure. For each snapshot that was within a cluster range, the chance was raised to at most 55%. We feel that this approximates the type of data we would get from an initial set of security conscious users.

The results from this initial training set can be seen in Table 1a. Our test set, which consisted of an additional one hundred thousand snapshots, was then scored by both learning algorithms. In the rule-based approach, we checked how often the rules would trigger based on the confidence values we set as a threshold. We are able to achieve fairly stable results with confidences between 25% and 50% on both algorithms, though the rule-based approach provides slightly lower values for all confidences. This implies both learning algorithms are accurate. Note that, at 0%, when all locations are considered insecure, the learning algorithm is still correct 38.8% of the time on both algorithms. This is because the scored data set had around 38.8% of the snapshots generated as being insecure.

After the initial training set, we generated a new set of training and test data, each with one hundred thousand (100,000) snapshots, which introduced two new clusters into the data. The first based on location and the second based on temperature. The addition of these two new clusters raised the total percentage of the insecure environments to 48.9%. This simulated additional attackers appearing after the initial training, showing resilience in the training model and the ability of the system to learn new clusters as they arise.

The results of this dataset can be seen in Table 1b. Again, we see the best results from confidences between 25% and 50%, and a lower accuracy from the rule-based approach. While both performed well at 50% confidence, the cloud-based service outperformed by 1.6-12.9%. This is most likely because of the choice of the decision forest regression as the training model, as this model is fairly accurate.

Table 1. (a) Initial Training Set Results (b) New Cluster Results

	% Confident	% Correct
Rule-Based	0	38.8
	25	78.9
	50	80
	75	70.8
	95	66.3
Cloud-Based	0	38.8
	25	81.6
	50	81.5
	75	71.1
	95	69.5

	% Confident	% Correct
Rule-Based	0	48.9
	25	68.9
	50	75.8
	75	67.9
	95	52.1
Cloud-Based	0	48.9
	25	80.3
	50	80.2
	75	67.7
	95	65

7. Conclusion and Future Work

In this paper, we extended the functionality of previous research on our existing mobile application which prevents Bluetooth communication between a base station and a device in an insecure environment. We introduce a system that promotes the use of two machine learning algorithms, trained on simulated crowdsourced data, to predict if a user is in an insecure environment. We found that both machine learning algorithms were accurate, with the rule-based method being accurate up to 80% and the cloud-based method being accurate up to 81.6%. Both options allow for increased security in potentially insecure environments.

This method could be implemented by device manufacturers and application creators using preexisting machine learning packages and cloud services. Similar to WAVE's need for modified street maps, it would require modification only as new sensors were configured on wearables and how the data was formatted. The app creators could determine how often to retrain the models based on their security needs, which might provide a competitive advantage. The main limitation for a third-party app is whether device manufacturers are willing to allow developers access to what the wearable communicates, which could restrict how the device can be adapted.

Future work will focus on both improving the machine learning algorithms and increasing the features which can be learned on. Additionally, we would like to begin testing our app with a small user base to see if our generated dataset is similar to the real world data that users would generate. Since our app relies on users to be aware of the locations and situations where they are actually insecure, we plan to analyze how often these users are correct. An important consideration may be to identify if there are other factors that can be assessed to automatically detect insecure environments that users are in and provide information on a more public scale.

8. References

[1] https://www.bluetooth.org/en-us/Documents/Annual_Report_2014.pdf

[2] P. Bourelos, G. Kousiouris, O. Voutyras, and T. Varvarigou, "Heating Schedule Management Approach through Decentralized Knowledge Diffusion in the Context of Social Internet of Things", *Proceedings of the 19th Panhellenic Conference on Informatics*, ACM, Athens, Greece, 2015, pp. 103-108.

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer", *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, Cambridge, United Kingdom, 2014, pp. 609-622.

[4] A. Diallo, W. Al-Khateeb, R. Olanrewaju, and F. Sado, "A Secure Authentication Scheme for Bluetooth Connection", *Proceedings of the International Conference Computer and Communication Eng.*, 2014, pp.60-63.

[5] M. Georgescu, D. D. Pham, C. S. Firan, U. Gadiraju, and W. Nejdl, "When in Doubt Ask the Crowd: Employing Crowdsourcing for Active Learning", *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics*, ACM, Thessaloniki, Greece, 2014, pp. 12:1-12:12.

[6] J. Hong, "Research Issues for Privacy in a Ubiquitously Connected World", *NITRD Research Strategy on Privacy*, 2015, pp. 1-20.

[7] T. Hothorn, K. Hornik, and A. Zeileis, "ctree: Conditional Inference Trees", http://cran.nexr.com/web/packages/partykit/vignettes/ctree.pdf, 2015.

[8] N. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An Early Resource Characterization of Deep Learning on Wearables, Smartphones, and Internet-of-Things Devices", *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*, 2015, pp. 7-12.

[9] Y. Minoda, I. Ohama, and E. Muramoto, "A Machine Learning Approach for Lighting Perception Analysis via Crowdsourcing", *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, Osaka, Japan, 2015, pp. 1355-1360.

[10] P. Moreno, A. Artex-Rodriguez, Y. W. Teh, and F. Perez-Cruz, "Bayesian Nonparametric Crowdsourcing", *Journal of Machine Learning Research*, JMLR.org, 2015, pp. 1607-1627.

[11] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, and X. Fu, "How privacy leaks from Bluetooth mouse?", *Proceedings of the ACM Conference on Computer and Communications Security*, 2012, pp. 1013-1015.

[12] M. Ryan, "Bluetooth: with low energy comes low security", *7th USENIX conference on Offensive Technologies*, USENIX, Washington, D.C., 2013, pp. 4-4.

[13] J. Saxe, R. Turner, and K. Blokhin, "CrowdSource: Automated Inference of High Level Malware Functionality from Low-Level Symbols Using a Crowd Trained Machine Learning Model", *Proceedings of the 9th International Conference on Malicious and Unwanted Software*, IEEE, Fajardo, PR, 2014, pp. 68-75.

[14] M. S. Shahriar and M. S. Rahman, "Urban Sensing and Smart Home Energy Optimizations: A Machine Learning Approach", *Proceedings of the 2015 International Workshop on Internet of Things Towards Applications*, ACM, Seoul, South Korea, 2015, pp. 19-22.

[15] C. Walter, M. L. Hale, and R. F. Gamble, "Imposing Security Awareness on Wearables", *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, ACM, New York, NY, USA, 2016, p. 29-35.

[16] C. Wang, C. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or Foe?: Your Wearable Devices Reveal Your Personal PIN", *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ACM, New York, NY, USA, 2016, pp. 189-200.