IoT Neighborhood Watch: device monitoring for anomaly detection

Pedro Carmo
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
pedro.carmo@tecnico.ulisboa.pt

Abstract—Recent developments in wireless device technology has allowed our home networks to reach simple everyday objects, becoming themselves sensors or actuators, connected to the Internet. However, as these devices leverage Internet connections to provide their services, they can also fall prey to cyberattacks and compromise privacy in vital environments such as our workplace or our home. In this work, we propose IoT Neighborhood Watch (IoT-NW), a Wireless Intrusion Detection System for the Internet of Things (IoT), hosted on devices that are part of the environment, enabling them to keep watch over their neighbor devices. Each device hosting the system can sniff packets in the network and perform feature extraction, at packet and flow level. These features are used to build behavior patterns of each device, and further used to detect any deviations that may be cause for alarm. An IoT-NW prototype was implemented. The prototype performance was evaluated in order to analyze how resources are used and how timely the detections are. The analysis used external datasets as well as data collected from a customized test-bed with specified attack scenarios. The results show that the IoT-NW approach is feasible and is able to detect the majority of the performed Denial of Service (DoS) attacks, with reasonably low false positive rates, when using external datasets. Similar detection rates were achieved in the customized test-bed, using the specified attack scenarios, when the system learns from more dynamic datasets but with low volume traffic.

I. INTRODUCTION

The Internet of Things (IoT) has been expanding across multiple domains, as it can improve efficiency in a wide variety of applications, such as Healthcare, Agriculture, Transportation, Infrastructure management and many more [1] [8] [9]. Physical devices can now be used to collect, process and act on data. The consumer electronics market alone has many offerings, including smart light-bulbs, plugs and home appliances. Most devices communicate using the already available Wi-Fi network. Other devices are power-constrained and need to use low power communication protocols, like Zigbee, which overcome reduced communication range using a device-to-device mesh architecture. Zigbee devices connect to a Hub, which can operate as a bridge between devices and different networks. This combination of networking technologies gives the user more flexibility and devices with reduced energy consumption can be installed seamlessly.

Despite such potential, fully secured systems are really hard to achieve and device security emerges as an important requirement in the design of products. However, practice shows that technology companies have been prioritizing lower time-to-market and production costs over good security practices [2]. Therefore, connecting these technologies to the Internet also means bringing its vulnerabilities and threats close to our daily lives.

For example, vulnerabilities in IP cameras allow attackers to penetrate the home network and expose credentials [7]. Some Zigbee devices are vulnerable to both passive and active attacks, allowing attackers to reset them to factory state, decrypt communications or even inject commands [6]. Furthermore, recent events have shown that targeted attacks are not the only concern. Attackers have also devised malware to capture devices and use them as part of botnets (e.g. Mirai [4]) that can then be used to launch massive Denial of Service (DoS) attacks and other exploits [7].

To mitigate such threats, users should be informed about the behavior of their devices. The Princeton IoT Inspector, is a free tool that can analyze device behavior through a graphical user interface. The monitoring can be automated, as shown by Kitsune [5], an online network intrusion detection system (IDS) among others. Kitsune is able to extract contextual features from network traffic as it happens and report anomalies. Despite the detection, the tool report still lacks important insight about the anomaly itself, e.g., what is causing the anomaly.

With this work, we address the following problems: (1) to detect the presence of neighbor Wi-Fi devices; (2) to trace the activity of such neighbor devices; (3) to detect and identify abnormal behavior in each neighbor device; (4) to use already available technologies. The

1https://iot-inspector.princeton.edu/
first can be obtained by scanning the network using ARP requests, the second is done by posing the host device as the gateway and extracting features from collected packets; the third uses one-class classifiers neural network, called autoencoders; and the latter is expressed by selecting an example setup for evaluation with devices already launched in the consumer electronics market. We provide empirical evaluation using external datasets documented by Hamza et al. [3] and datasets collected from a customized test-bed. Finally, we also provide an insight on how the user presence and its interactions with the devices can influence the behavior analysis model and how this can be used to adapt it to the user presence input.

II. SMART HOME TESTBED

We defined a smart home testbed to develop and evaluate our system, IoT-NW. The selected commercial devices have Wi-Fi or Zigbee connectivity.

The Wi-Fi devices are: a Linksys WRT543G home router; a Raspberry Pi running Home Assistant\(^2\) as a Hub, since it is open-source and provides interconnection between both networks and the user; an Android-based IP Camera due to their role on botnet attacks and their critical security issues; a TP-Link Smart Plug that can be used to turn on or shut down connected appliances; a Smartphone to provide interface between the smart home Hub and the user. The Zigbee devices are: Phillips Hue Light and Dimmer for the lighting system; a Trust Motion Sensor to automate lighting control and for surveillance; a Temperature/Humidity Sensor. In both networks, brands were decided according to their wide availability in the consumer market. The system is depicted in the Figure 1.

![](image)

![Fig. 1. Devices integrated in a smart home with a Zigbee and a Wi-Fi network.](image)

\(^2\)https://www.home-assistant.io/

A. Attacker Model

IoT-NW is intended to protect smart home owners against potential harmful actions from outsiders. Therefore, an attacker needs to access the Local Area Network (LAN) or the device specific cloud services, which require the user credentials. In this model, we only consider the situation where an attacker is able to penetrate the LAN with an unknown device to perform malicious actions in the network through two different scenarios: (S1) where a legitimate user is not present in the smart environment and (S2) a legitimate user is present in the smart environment. We do not intend to detect specific penetration strategies, rather we focus on detecting malicious interactions between devices.

We consider all attacker capabilities have to be remotely executed and they are presented as follows: (A1) send and receive packets, to and from any device in the network; (A2) forge ARP or IP packets; (A3) access the smart home hub.

Capability A1 can be acquired as soon as the enters the network. Meaning the attacker has only access to the information provided by the compromised device, as if he owned it.

Capability A2 can be used to perform ARP scanning, making the device capable of identifying the devices present in a Wi-Fi network. It also allows the attacker to send IP packets that comply with the API of a device to its IP address. Some devices, like the TP-Link smart plug do not authenticate commands sent by local devices, meaning the attacker can potentially shut down the smart plug or get state information from it.

The Hub needs to be properly configured to be interfaced with every smart home device. This configuration is placed in a file, which may gather important credentials for accessing other local devices. A3 assumes the attacker to be able to connect himself using Secure Shell Host (SSH) service, having access to the configuration file. Which, in turn, will enable the attacker to connect himself to the IP Camera web service, without needing to sign in the Hub web service.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Home Assistant</th>
<th>IP Camera</th>
<th>TP Link Smart Plug</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>{A1, A2, A3}</td>
<td></td>
<td>{A1}</td>
</tr>
<tr>
<td>Web Login IPCam</td>
<td>{A1}</td>
<td></td>
<td>{A1, A2}</td>
</tr>
<tr>
<td>ARP Scan</td>
<td>{A1, A2}</td>
<td>{A1, A2}</td>
<td>{A1, A2}</td>
</tr>
<tr>
<td>Plug On</td>
<td>{A1, A2}</td>
<td></td>
<td>{A1, A2}</td>
</tr>
<tr>
<td>Plug Off</td>
<td>{A1, A2}</td>
<td></td>
<td>{A1, A2}</td>
</tr>
</tbody>
</table>
III. IoT-NW

The proposed system is an online Wireless Intrusion Detection System (WIDS), designed to actively monitor the behavior of different Wi-Fi devices and detect anomalous device behavior. IoT-NW is an Wireless Intrusion Detection System (WIDS), designed to actively monitor the behavior of different IoT devices and detect whether a device is under an anomalous behavior or not. Therefore, the system is organized into different components: Packet Capturer (PC), Feature Extractor (FE), Preprocessor (PP), Behavior Analysis (BA) and Statistical Analysis (SA), as illustrated in Figure 2.

![Figure 2. Architecture of the IoT-NW.](image)

The system starts by collecting packets from the Wi-Fi interface of the host device. Then, for each packet, FE extracts relevant features, which we refer to as raw features. From those raw features, PP will relate them with the respective device (present in the WLAN) and generate new features at a packet and flow level, which we will further refer as network features and flow features, respectively. Each set of the new processed features is sent to the BA, responsible for processing them in order to construct a model during a phase we call the learning phase). Afterwards, during the detection phase, new collected and processed features are used to evaluate how much they deviate from the constructed model.

A. Packet Collection

Though there is a shared medium in wireless networks, promiscuous mode is not enough to read packets from other devices. Therefore, our system launches ARP spoof attacks using the aprsloop tool for each target device, in order to place the IoT-NW between our target and the Home Router). This allows the system to receive packets from the target devices on its Wi-Fi interface.

B. Feature Extraction

The received packets are then accessed by the Scapy module, which parses the binary frame into a Python-readable object and extracts raw features. These are

\[ RMSE(x, x') = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x'_i)^2}{n}} \]

C. Preprocessor

Each extracted feature set, being network or flow related, is then sent to the PP module, where each feature \( x \) is scaled to a value \( s \in [0, 1] \subset \mathbb{R} \) using the maximum value \( (x_{max}) \) to define the range in which \( x \) operates. If a larger value is found, \( x_{max} \) is updated. The minimum value \( x_{min} \) is considered to be 0. The scaling function \( f \) is:

\[ f(x) = \frac{x - x_{min}}{x_{max} - x_{min}} \]

D. Learning

When running the learning phase, the system will use scaled features to analyze the behavior (on the BA) of each device and build a behavior model, one for each kind of processed features. These behavior models have the structure of an Autoencoder. An Autoencoder works as an Artificial Neural Network (ANN), composed by multiple layers of neurons, where each neuron in a layer is connected to all neurons in the forward layer through connections called synapses. Each synapse has an associated weight and defines the concepts learned by the model. For a more extensive background on Autoencoders, please refer to [5].

In this application, each Autoencoder has 3 layers: input, hidden and output. The purpose of the learning phase is to force the weights of all neurons to adapt, in order to minimize the error between the input and output layers. To calculate the error we use the root mean squared error (RMSE) defined in 2, where \( x \) is the input feature vector and \( x' \) the output.

The obtained RMSE values (one for each feature instance), are then sent to the SA, where a mean and standard deviation are obtained for network and flow features separately. These values compose the network and flow statistical models, for each device.

\[ RMSE(x, x') = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x'_i)^2}{n}} \]

\[ \text{Identified by a 5-tuple key \{IPv4 source address, IPv4 source port, IPv4 destination address, IPv4 destination port, IPv4 protocol code\} and aggregates traffic from both directions.} \]
E. Detection

After there is two behavior and statistical models, per feature type (network and flow), four in total per device, the system is able to run the **detection phase** for that specific device. Similar to the learning phase, the scaled features are analyzed in the **BA**. However, these will only be fed to the Autoencoder input layer and the RMSE calculated, with no weight adaptation performed. For each feature instance, the RMSE is sent to the **SA**, where the statistical model (mean and standard deviation) for its respective feature type is loaded. Then a normal cumulative distribution function (CDF) is built and a probability for the RMSE is extracted from it. The average of all probabilities obtained in a timeframe is performed in the end and the obtained result is the probability of a timeframe having an anomaly. This probability is then compared to a threshold. If above it, an alert is launched.

IV. Experimental Results

The system evaluation was performed at the operation level, which analyzed how resources are used and how timely and accurate the detections are. As memory is an important resource for low power devices, we ran the system on the host for around 490 minutes while having YouTube videos on to simulate the user’s behavior. During this time, we were also measuring the memory used by the system with the **Python** tool `mprof`. In Figure 3, we can see the result of such experiment. the system peaked its memory usage at 350 MiB, 4 minutes after it started. Then at 25 minutes after it started it reached its minimum memory usage at 257 MiB. From then on it went on slightly increasing over time, reaching 278 MiB at the end of the test, having increased its memory around 21 MiB after processing 13587 timeframes. This experiment shows that the system can continue to operate for long periods with a stable memory utilization. However, the memory required is still high and does not enable the system to run on low resource devices. For example, a Raspberry Pi 3 model A+ only has around 536 MiB available.

Furthermore, each detection is only useful if detected in conditions that allow some reacting mechanism to act on it in time. Therefore, the system was placed over stress conditions with high traffic volumes sent to its host. Running in single thread makes its own collection too low, the process of extracting collecting packets and extracting its raw features were separated, from all the processing steps (process features, learn model or comparing traffic with model), in different threads. The loss from close to 99%, in single thread, to 0% at 1500 pps, in multi-thread can be seen from Figure 4(a) to Figure 4(b), respectively.

![Fig. 3. Memory usage over time of the main thread, ticking job and GUI thread.](image)

![Fig. 4. System packet loss, in percentage, by packet rate during detecting phases for single thread and multi-thread solutions.](image)
Experiments with high traffic volumes, around 2000 pps, were simulated using the hping tool, which allows a user to send multiple ping requests, at very high rate, to a specific device. The results suggest, as seen in Figure 5, the system still needs improvements when performing the detection phase.

Moreover, additional experiments with lower traffic volumes (around 20 pps) in learning and detection phases were also performed. The box plots of such results can be seen in Figure 6 and in all experiments the timeframe was set to one second. Considering lower traffic volumes, the system is only able to perform most of its behavior analysis under one second for the detection phase experiment. Results suggest that the system cannot follow the timeframe compromise for the learning phase under allegedly acceptable traffic volumes and for the detection phase under high traffic volumes.

Regarding the anomaly detection accuracy, we used datasets with benign traffic from Hamza et al. [3], to train our models. In particular, we used the dataset labeled as October 29th, 2018, with our system running on learning phase. Furthermore, we launched the system on the detection phase over a dataset with attack instances, from June 1st of 2018, from the same source. These attacks target a TP Link Smart plug and are resumed in the Table II. More details can be found in their paper [3].

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Rate</th>
<th>Attack scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ARP</td>
<td>1</td>
<td>L2D</td>
</tr>
<tr>
<td>2</td>
<td>spoofing</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TCP SYN</td>
<td>100</td>
<td>L2D</td>
</tr>
<tr>
<td>4</td>
<td>flooding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>L2D2L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L2D2L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>I2D2I</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>L2D2L</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>L2D2L</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>L2D2L</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7(a) presents the plot of the obtained RMSE values during the detection phase on the dataset from June 1st of 2018. The threshold is set to 0.87 and we can see the detections obtained from the learnt model for the TP Link Smart Plug in black crosses. The detections above the threshold are highlighted with a plus red sign, and indicate the considered anomalies that are reason to alarm. Additionally, we plot in Figure 7(b) the True Positive Rate (TPR) across the False Positive Rate (FPR), both obtained by changing the threshold multiple times. This is called Receiver Operating Characteristics (ROC) curve and is a typical performance measurement in machine learning algorithms. On the best fitting result, threshold is at 0.84, FPR reaches 3.2% and TPR 62%, with a precision, an accuracy and f1-score around 0.99, 0.69 and 0.76 respectively.

Finally, we used the scenario defined in Section II to construct multiple datasets with benign traffic with and without user interactions. Likewise, multiple datasets with all attacks defined in the attacker model were performed.

The benign datasets were used to train devices in two different situations: (a) only benign datasets with no traffic related to user interactions were used and (b) benign datasets with and without traffic related to user interactions were used. Then, for such situations, different datasets containing abnormal traffic behavior were used. The attack datasets were collected with and
without the user behavior. Table III provides the results of the best fitting threshold for the different combinations between benign datasets and attack datasets, and the respective area under curve obtained by plotting the ROC curves, similarly to Figure 7(b). The color red or green in the values for the situation (b) means a negative or positive change, respectively, when compared to the situation (a). As it can be seen, results demonstrate that having models trained with and without user behavior enhances detection performance measures in the IP Camera and Smart Plug when detecting on attack datasets without user behavior. Even if FPR increases or TPR decreases, they both occur with little significance when compared to the remaining measurements. The Home Assistant only performs better when detecting on the attack datasets containing user interactions, as opposed to the other devices. This suggests that the models trained without user interactions are less adapted to the situations containing user behavior, hence more susceptible to user behavior variations. Likewise, if the model is trained with user interactions, the model will be more adapted to more dynamic alike behaviors, meaning they will be less susceptible to variations related to the attacks, as well as to the user interactions.

**Table III**

<table>
<thead>
<tr>
<th>Attack Datasets</th>
<th>Threshold</th>
<th>AUC</th>
<th>TPR</th>
<th>FPR</th>
<th>FPRF</th>
<th>Precision</th>
<th>f1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP Link Smart Plug with benign datasets without user behavior</td>
<td>0.995</td>
<td>0.967</td>
<td>0.907</td>
<td>0.982</td>
<td>0.270</td>
<td>0.330</td>
<td>0.200</td>
</tr>
<tr>
<td>TP Link Smart Plug with user interactions</td>
<td>0.870</td>
<td>0.970</td>
<td>0.963</td>
<td>0.980</td>
<td>0.130</td>
<td>0.200</td>
<td>0.110</td>
</tr>
<tr>
<td>TP Link Smart Plug without user interactions</td>
<td>0.995</td>
<td>0.967</td>
<td>0.907</td>
<td>0.982</td>
<td>0.270</td>
<td>0.330</td>
<td>0.200</td>
</tr>
</tbody>
</table>

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented a Wireless Intrusion Detection System (WIDS) capable of being hosted in a computing device capable of running Tensorflow and provide anomaly detection in Wi-Fi devices through traffic behavior analysis. The memory required by the system to run peaks at 350 MiB and has a stable consumption over time. However, the system is not ready to be deployed in a low resource host. This problem can be improved by reducing dependencies from Python like Tensorflow and Scapy. Moreover, the system is able to provide a response under the defined timeframe, but behavior analysis takes longer when traffic volumes are raised. Also, this increase in behavior analysis time compromises the packet collection efficiency as well. Though using multithreading increases the packet collection efficiency, such parallelization is still limited by the Python multithreading mechanisms that only allows partial multithreading.

Regarding anomaly detection, the system is able to detect most of the attacks performed in the used third party datasets, having a significant TPR around 60% at a low FPR (3%). However, such FPR still presents a high
inconvenience for future users. Furthermore, we evaluated the system in a customized scenario, considering different attacks, including benign and anomalous traffic with and without user interactions. Results indicate that the best fitting threshold differs in every combination of different situations with and without user interactions. This suggests user presence is a contextual fact that can be used to enhance detection. In our case, this contextual knowledge can be used to define different thresholds for different contexts. However, we consider important to test the system with broader datasets to clarify the statistical significance of these results. Additionally, it is important to test the system with larger learning phases, to infer how limited is the system adaptation with the features currently being used or if additional features are needed. Finally, expanding the behavior analysis to the low power devices using Zigbee would increase the anomaly detection surface and expand the possibilities on new contextual knowledge. For example, using temperatures inside and outside the smart environment to perceive if heating or cooling system should actually be acting or not.

VI. ACKNOWLEDGEMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

REFERENCES


