IoT Neighborhood Watch: device monitoring for anomaly detection

Pedro E. Carmo[0000−0003−4100−2662] and Miguel L. Pardal[0000−0003−2872−7300]

Instituto Superior Técnico, Universidade de Lisboa, Portugal
{pedro.carmo,miguel.pardal}@tecnico.ulisboa.pt

Abstract. Recent developments in wireless device technology allow simple everyday objects, like plugs and locks, to become sensors/actuators connected to the Internet. These smart things can make environments aware of user needs and be used to improve accessibility and efficiency. However, these devices can also fall prey to cyber-attacks and compromise privacy in personal environments such as our home.
We propose *IoT Neighborhood Watch*, an attack detection system, hosted on devices that are part of the environment which are able to keep watch over their neighbor devices. Each device can sniff packets in the network and perform feature extraction, from data gathered both at packet and flow levels, along with device states and user presence detection. All these value can be used to build behavior patterns, which are then used to detect any deviations that may be cause for alarm. The proposed system is currently under development in a test-bed containing a diverse set of devices, with Wi-Fi and Zigbee connectivity.

Keywords: Wireless Security · Internet-of-things · Network Monitoring · Anomaly Detection.

1 Introduction

Physical devices can be used to collect, process and act on data. This Internet of Things (IoT) has been expanding across multiple domains, as it can improve efficiency in a wide variety of applications [1][7][9]. 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 use low power communication protocols, like Zigbee, which overcomes reduced communication range using a mesh architecture (device-to-device). Zigbee devices connect to a *Hub*, which can operate as a bridge between both networks. This combination of networking technologies gives the user more flexibility and devices with reduced energy consumption can be installed seamlessly.

Technology companies have been prioritizing time-to-market and production costs over good security practices [2]. For example, vulnerabilities in IP cameras allow attackers to penetrate the home network and expose credentials [6]. Some Zigbee devices are vulnerable to both passive and active attacks, allowing attackers to reset them to factory state, decipher their communications or even inject

commands [5]. 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 [3]) that can then be used to launch massive distributed denial-of-service (DDoS) attacks and other exploits [6].

To mitigate such threats, users should be informed about the behavior of their devices. The Princeton IoT Inspector[1], is a free tool that can analyze device behavior through a graphical user interface. The monitoring can be automated, as shown by Kitsune [4], 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?

The problem we address in this paper is twofold: (1) to provide a near real-time detection of anomalous device interactions and (2) to improve the system detection reliability with contextual knowledge of each device state.

The first (1) can be obtained by using two one-class classifier neural networks, called *autoencoders*. Each of the classifiers is statistically modeled after one device's traffic behavior and then used to test whether there is any deviation from its model. The latter (2) uses contextual data from each device to perceive relevant facts, for example, if a user at home or not. With this information, the detection system can adapt its sensitiveness to different expectations about device behaviors.

2 Smart Home Testbed

We defined a smart home testbed to develop and evaluate our system, *IoT Neighborhood Watch*. 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 Figure 1.

2.1 Attacker Model

IoT Neighborhood Watch is intended to protect smart home owners against potential harmful actions from outsiders. Therefore, an attacker needs to access

[1] https://iot-inspector.princeton.edu/
[2] https://www.home-assistant.io/

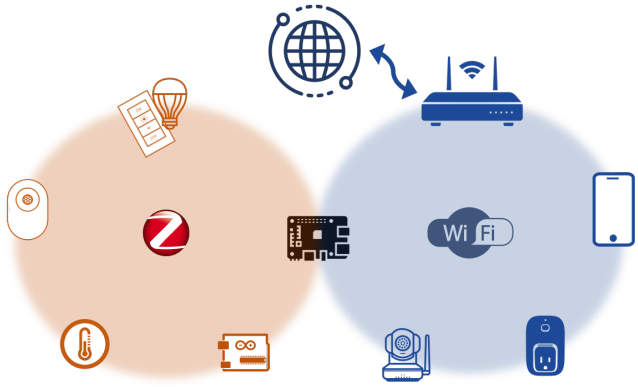


Fig. 1: Devices integrated in a smart home with a Zigbee and a Wi-Fi network.

the LAN (Local Area Network) or the device-specific cloud services, which require the user credentials for access. In this model, we only consider the situation where an attacker is able to penetrate the LAN and compromises a previously trusted device. We do not intend to detect specific penetration strategies, rather we focus on detecting malicious interactions between devices.

In the attacker model, We consider the following remote capabilities: (A1) access the state information of the compromised device; (A2) record every packet between a given source and destination and replay them; (A3) send packets to any address; (A4) access the smart home hub.

Capability A1 can be acquired as soon as the device is compromised. If the device is compromised at a system level, capability A2 can be obtained by performing ARP spoofing[8], making the device act as a gateway. Capability A3 is acquired by accessing the network interface, to forge packets and send them to devices that do not require local network authentication. Capability A4 assumes the attacker is able to connect using Secure Shell (SSH) service, having access to the Hub configuration file that contains credentials in plain text. These will, in turn, enable the attacker to connect to the IP Camera without the need to sign in the Hub web service.

3 IoT Neighborhood Watch

The proposed system is an online Network Intrusion Detection System (NIDS), designed to passively monitor the behavior of different IoT devices and detect anomalous device behavior. It uses machine learning to model behavior in a *learning phase*, which takes extracted features, considers them normal behavior and adapts the behavior model to them. On the *detection phase* it collects the current features and computes the deviation between them and the built behavior model. The system is presented in Figure 2, where **PC** stands for Packet

Capturer, **FE** for Feature Extractor, **PP** for Preprocessor, **AD** for Anomaly Detector and **PD** for Presence Detector.

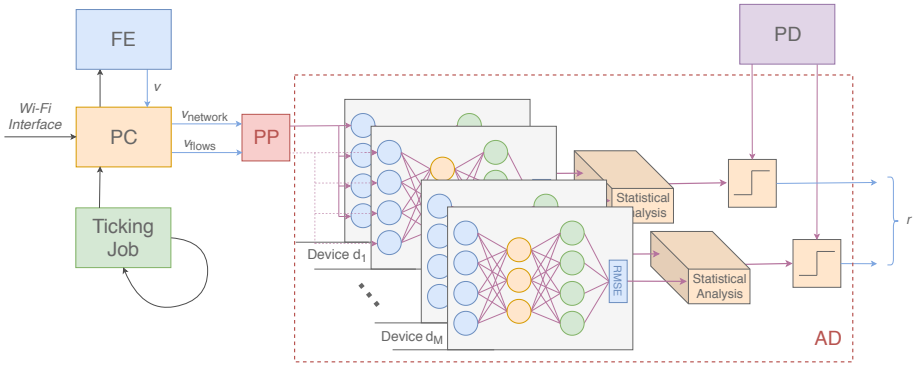


Fig. 2: Architecture of the system.

The AS component)not shown in the figure) is performed by using the *arpspoof*[3] tool, and the PC uses *scapy*[4] as an external library. Both AS and PC run on background, with PC calling FE at each packet. The FE only extracts the features when a ticking thread, with a given time period, signals PC. This makes the feature extraction periodical. After extraction, the features are sent to the PP module, which then provides the scaled features to the AD module. The PD module, provides the indication of whether a user at home or not.

3.1 Monitoring and Learning

Our work monitors devices through their traffic in the network. The analysis is made for each packet, where base features are extracted in order to compute two different sets of features: network and flow. Network features are extracted from the transmission unit frame of each collected packet. Flow features are extracted from TCP and UDP packets, since these are typically used to handle data communication between devices in the Internet.

As features are aggregated in each time window, we construct an instance collection $v = \{v_{network}, v_{flows}\}$ for each time window, where $v_{network}$ is a vector of size M (number of devices), with each value of the vector being the network features for a device m and the v_{flows} a matrix of size MxJ, where J stands for the number of detected flows in a collection v.

The extracted network features are: the **MAC address** of the device that generated the frame, the **Ethernet Type**[5] which identifies the network layer

[3] https://su2.info/doc/arpspoof.php
[4] https://scapy.readthedocs.io/en/latest/
[5] Only ARP, IPv4 and IPv6 ethernet types are considered.

protocol; the corresponding **Packet Count** for that type; and the **Flow Count** at each time window. Each device is identified by its MAC address.

We identify each flow with a 5-tuple hashed key: {IPv4 source address, IPv4 source port, IPv4 destination address, IPv4 destination port, IPv4 protocol code}. All these fields are directly extracted from TCP and UDP packets with no required processing. The features for each flow require processing, as they result from a contextual analysis. Therefore, we extract: the **flow duration** in milliseconds; the **packet count**; the **total octets count** in both directions; the **TCP synchronize flags count**; the **TCP reset flags count**; and the **average time between two packets** in the flow. All these features are extracted from both incoming and outgoing packets.

The instance collection 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}) found 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:

$$f(x) = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

After scaling, the scaled network features $s_{newtork}$ and the scaled flow features s_{flows} are introduced in the AD component, which has one one-class classifier, named Autoencoder, for each set of features. 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 [4]. The purpose of an Autoencoder is to model the relations between each inserted feature in the input layer, by adapting the network weights to the function that minimizes the difference between the input features and the output result. This is referred to as the learning phase, where each Autoencoder tries to reproduce the introduced scaled feature vector in the output of each one. To calculate the error we use the root mean squared error (RMSE) defined in 2, where x is the input feature vector and x' its output.

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

The learning phase computes multiple feature instances and the mean RMSE for each feature after reducing the error. It also gathers the RMSE deviation in this period. Both are obtained for each Autoencoder. Finally, a Gaussian distribution is built for each Autoencoder.

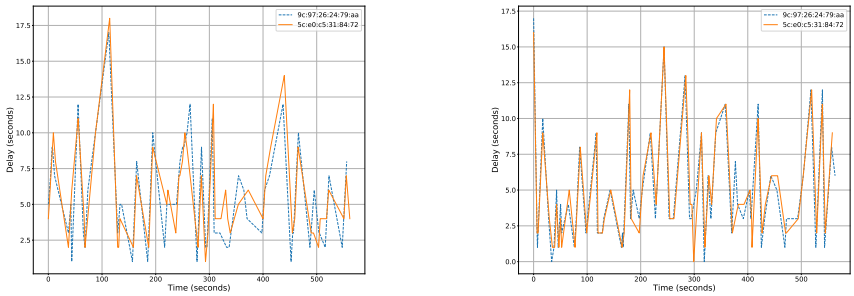
3.2 Detection

After running the training phase, we have a statistical model of the expected values for the RMSE of each Autoencoder. At this point, when in the detection phase, the system performs the same monitoring operation. The difference is in

the Autoencoders, where the RMSE is calculated for each feature instance, but the error is not minimized for that instance. Instead, the computed RMSE is inserted in the statistical model and gets the probability of each network and flow instance being normal, for each device. The smaller the probability, the larger the deviation is expected to be. As different contexts require different thresholds, we use the PD module to indicate whether there is someone inside the home or not, and choose corresponding thresholds. We typically want more sensitiveness when the user is away from home.

4 Experimental Results

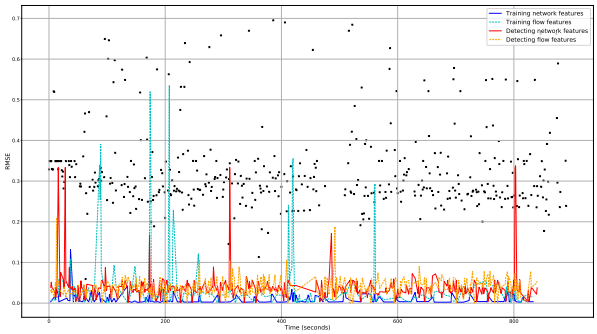
We measured the delay between the feature extraction and the outcome computation. For this experiment, only two devices were in the network, one being the monitor device itself (*5c:e0:c5:31:84:72*). By analyzing Figure 3, we can see that both devices can reach to significant delays (close to 20 seconds) when using a time window of 1 second. As the number of devices increases, the response time is expected to achieve larger delays, since there will be more packets to analyze and more devices to train and detect.



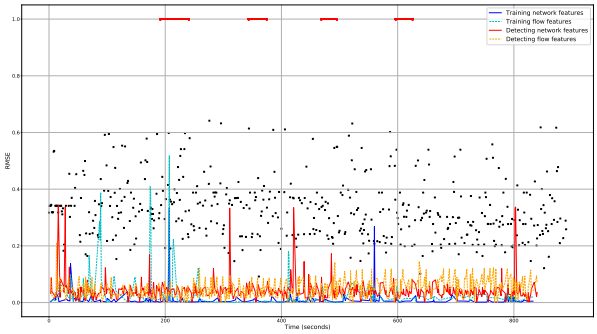
(a) (b)

Fig. 3: Measuring the delay between the appearance of an instance and the respective response during the learning phase (a) and the detection phase (b).

We also profiled each Wi-Fi device mentioned in the testbed section with all devices integrated. No commands were issued commands between them, and no user interaction was made to any application in the Internet. Figure 4 (a) presents the RMSE (root mean squared error) over time for each feature type (network and flow), for both learning and detection phase running on the same dataset with no issued commands between them as mentioned before. We can see that after training, there are some RMSE peaks, even though the dataset is the same, which can lead to False Positive detections even after defining a statistical



(a)



(b)

Fig. 4: RMSE measures (coloured lines) and Anomaly probability (black crosses) over time for training and detecting phases each pair of red dots united by a line correspond to attacks with capabilities {A1, A3, A4}, {A1, A2, A3}, {A1, A3}, respectively.

pattern. Figure 4 (b) presents the same information, but using different datasets in each phase. For the learning we use the previously mentioned dataset, but for the detection we use a dataset with different attacks (each with different capabilities). Here, we can see high RMSE values outside each two pair of points united by a line (time range between which the attacks were performed), meaning possible False Positive detections. We can also see True Negative detections as well (system is not able to properly detect anomalies when they happen). By inserting contextual information about the user, like the location, some devices could have their thresholds adapted for the respective contextual occasion. The defined threshold would be adapted to allow the statistical analysis being more or less flexible with the resulting anomaly probability.

5 Conclusion and Future Work

In this paper, we have presented a NIDS capable of detection with some delay and still without the capability to use contextual information in the traffic analysis. To *reduce the detection delay*, we will increase code efficiency and implement multiple threads, on both learning and detecting phase. Increasing the time window would also reduce the number of operations required but increase the response time. To *reduce the False Positive Rate (FPR)*, we will adjust the thresholds, taking into account contextual information available, starting with user presence at home. Future work in the experimental analysis is also required. Besides the detection of differences in the behavior on a previously trusted device that turned malicious, we will also want to detect the appearance of new devices that may be malicious.

6 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

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys and Tutorials **17**(4), 2347–2376 (2015). https://doi.org/10.1109/COMST.2015.2444095
2. Alrawi, O., Lever, C., Antonakakis, M., Monrose, F.: SoK : Security Evaluation of Home-Based IoT Deployments. IEEE Oakland (2019)
3. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**(7), 80–84 (2017)
4. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089 (2018)
5. Morgner, P., Mattejat, S., Benenson, Z., Müller, C., Armknecht, F.: Insecure to the touch: attacking zigbee 3.0 via touchlink commissioning. In: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 230–240. ACM (2017)
6. Seralathan, Y., Oh, T.T., Jadhav, S., Myers, J., Jeong, J.P., Kim, Y.H., Kim, J.N.: Iot security vulnerability: A case study of a web camera. In: 2018 20th International Conference on Advanced Communication Technology (ICACT). pp. 172–177. IEEE (2018)
7. Vyas, D.A., Bhatt, D., Jha, D.: IoT : Trends , Challenges and Future Scope. International Journal of Computer Science & Communication **7**(1), 186–197 (2016)
8. Whalen, S.: An introduction to arp spoofing. Node99 [Online Document], April (2001)
9. Whitmore, A., Agarwal, A., Da Xu, L.: The Internet of Things-A survey of topics and trends. Information Systems Frontiers **17**(2), 261–274 (2015). https://doi.org/10.1007/s10796-014-9489-2