# IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT

Markus Miettinen
Tommaso Frassetto
Ahmad-Reza Sadeghi
Technische Universität Darmstadt, Germany
Email: {markus.miettinen,tommaso.frassetto, ahmad.sadeghi}@trust.tu-darmstadt.de

Samuel Marchal
N. Asokan
Aalto University, Espoo, Finland
Email: samuel.marchal@aalto.fi
asokan@acm.org

Ibbad Hafeez
Sasu Tarkoma
University of Helsinki, Finland
Email: {ibbad.hafeez,sasu.tarkoma} @cs.helsinki.fi

***Abstract*—The emergence of numerous new manufacturers producing devices for the Internet-of-Things (IoT) has given rise to new security concerns. Many IoT devices exhibit security flaws making them vulnerable for attacks and manufacturers have difficulties in providing appropriate security patches to their products in a timely and user-friendly manner. In this paper, we present our implementation of IoT Sentinel, which is a system aimed at protecting the user's network from vulnerable IoT devices. IoT Sentinel automatically identifies vulnerable devices when they are first introduced to the network and enforces appropriate traffic filtering rules to protect other devices from the threats originating from the vulnerable devices.**

## I. Introduction

The vision of the so-called Internet-of-Things (IoT) is rapidly becoming a reality, as more and more people are installing IP-connected devices and appliances into their homes. An emerging issue with the adoption of IoT devices is related to their security. New players with limited experience in security engineering of consumer products are entering the market, leading to many new products being shipped with inherent security flaws [1]. Means for providing security updates to rolled-out products are often deficient, as manufacturers often do not have adequate processes in place for providing security updates to their products in a timely manner. Many products often lack easy-to-use means for updating their system software, making it too difficult for average consumers. These factors combined lead to a situation in which devices vulnerable to security attacks are likely to be present in many users' networks. Recently, massive DDoS attacks leveraged hundreds of thousands of compromised IoT devices [2], confirming the tangibility of this threat.

Managing the security of users' IoT devices in the presence of vulnerable devices is challenging. To accomplish this, one would need the possibility to identify vulnerable devices and take appropriate protective measures to protect other devices from potential security attacks utilizing the vulnerable devices.

We have built a prototype system, IoT Sentinel, to prevent vulnerable devices from compromising the security of other devices in the network. IoT Sentinel implements automated techniques, introduced in [3], for identifying vulnerable devices and isolating them from the rest of the user's devices. IoT Sentinel is composed of a local network gateway serving as access point (AP) for IoT devices and a cloud-based security service. It leverages traffic monitoring, machine-learning and software-defined networking (SDN) [4] components to achieve protection of the IoT network.

## II. Overview of IoT Sentinel

The goal of IoT Sentinel is to assess the vulnerability of new IoT devices when they are initially added to a network. According to this assessment, traffic filtering rules are generated and applied to isolate vulnerable devices, so that they will not be exploited or, if they are, not be used to compromise other devices in the network. IoT devices vulnerability assessment relies on the automatic identification of their *device-type*. In IoT Sentinel, a *device-type* refers to the tuple <*model*, *software version*> of a device.

The design of IoT Sentinel is shown in Fig. 1. It consists of a *Security Gateway* and an *IoT Security Service*. The Security Gateway is located in user's local network and acts as a local AP. IoT Security Service is a cloud service operated by an IoT Security Service provider (IoTSSP). The IoTSSP aggregates information about IoT device-types and assesses the required *isolation level* (Sect. III-B2) for each device-type. These assessments can be based on lab-based penetration tests, entries in vulnerability databases like, e.g., CVE [5], and, on cross-correlating crowd-sourced incident reports with device-types present in affected networks.

IoTSSP maintains a mapping between device-types and *isolation profiles*. An isolation profile consists of appropriate enforcement rules for isolation in order to protect other devices in the network from being affected by potentially vulnerable devices. IoT Sentinel's device-type identification relies on monitoring the communication of a device during its typical set-up phase when it is introduced to a network. Security Gateway records its communication with the AP and derives a fingerprint from it. The fingerprint is transmitted to IoT Security Service, which identifies the corresponding device-type and determines the associated isolation profile. Based on the isolation profile, Security Gateway generates traffic filtering (i.e. enforcement) rules that will be enforced in order to protect other devices in the same network from risks of exposure to vulnerable devices that may become compromised.

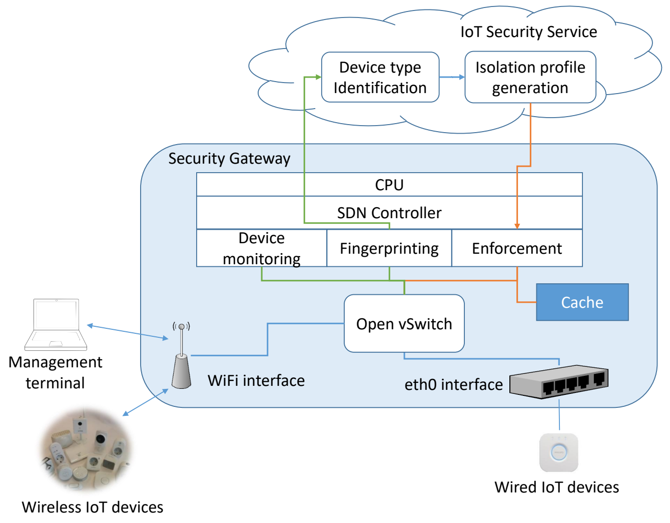


Fig. 1. Internal architecture of Security Gateway.

## III. Implementation

Our prototype implementation of IoT Sentinel consists of a Security Gateway acting as an AP in the user's network, and a cloud-based IoT Security Service. Security Gateway and IoT Security Service communicate using a RestAPI over HTTP/HTTPS.

### A. Security Gateway

The internal architecture of Security Gateway is shown in Fig. 1. The prototype implementation uses a modified version of open-source Floodlight SDN controller v1.2 [6] and supports Open vSwitch (OVS) v2.3 or later [7]. We have written custom modules in Floodlight for detecting new devices in the network, initiating device fingerprinting and communicating with IoT Security Service. Security Gateway is deployed on a Raspberry PI 2 device set up as wireless AP using `hostapd` [8], which runs the SDN controller and OVS.

*1) Device Fingerprinting:* Device-type identification in IoT Sentinel is based on *device fingerprints* extracted from the distinctive communication behaviour of a device when it is introduced for the first time to the user's network. When the SDN controller discovers a new device with a MAC address that has not been observed earlier, it initiates a `tcpdump` packet capture process recording all data packets originating from this device (identified using MAC address) for the duration of at most $k$ seconds, collecting a set $\{p_1, p_2, \ldots, p_N\}$ of $N$ data packets.

After the packet capturing process completes, A *feature extraction module* processes then the recorded packets to derive a set of 23 features $\{f_{1,i}, f_{2,i}, \ldots, f_{23,i}\}$ for each packet $p_i$. All features are based on the analysis of the packet headers only, allowing encrypted packets to be included in the feature set as well. The features indicate the use of particular protocols or encode other packet header properties. (For feature details, please refer to [3]).

```
{
    "id": 123,
    "name": "Ednet gateway",
    "isolation": 2,
    "permitted_ip": ["34.192.121.32", "23.20.224.23"]
}
```

Fig. 2. Example isolation profile sent by IoT Security Service to Security Gateway.

The resulting fingerprint $\mathbf{F}$ after processing $N$ packets is a $23 \times N$ matrix. Each row $i$ represents one of the 23 packet features and each column $j$ represents the $j$th received packet. $\mathbf{F}$ is stored in a CSV file, optionally compressed and uploaded to IoT Security Service for identification.

*2) Traffic Filtering:* Security Gateway uses SDN to dynamically change the network configuration. The wireless interface in Raspberry PI is bridged with a virtual interface in OVS to pass all wireless traffic through OVS, allowing us to manage traffic between clients connected to the same SSID.

When a new device joining the network is detected, it is placed in a quarantine network. The quarantine network consists of devices for which no isolation profile is yet known by Security Gateway. When an isolation profile is received from IoT Security Service (see Fig. 2), the new device is assigned either to the *trusted* or to the *untrusted* network. The SDN controller uses the isolation profiles to dynamically generate OpenFlow enforcement rules (OF-rule), which are then implemented in OVS to control the traffic in the network. A lightweight Enforcement Rule Generation Engine (ERGE) implemented as a Java library assists the SDN controller in generating OF-rules. Isolation profiles and OF-rules are stored locally in Security Gateway as JSON files.

OVS forwards every incoming packet for which there is no matching OF-rule available in OVS to the SDN controller. Our custom module in SDN controller intercepts these packets and extracts information from Layer 2–4 headers including, e.g., source and destination MAC addresses, IP addresses, port numbers, etc. Using this information, it verifies whether or not the given connection request satisfies the criteria specified by isolation profiles of devices involved in the connection. If a connection request is permitted by the matching isolation profile, ERGE generates a matching OF-rule with `action=FORWARD`, which allows the connection establishment. In case the connection is not permitted, ERGE generates an `action=DROP` OF-rule to prevent the connection establishment. In case an isolation profile is updated by IoT Security Service, the existing corresponding OF-rules are removed from OVS and regenerated according to new isolation criteria specified for the communicating devices.

### B. IoT Security Service

IoT Security Service was developed from scratch using the Python-based Flask Framework [9]. It is responsible for device-type identification and for maintaining a device-type to isolation profile mapping.

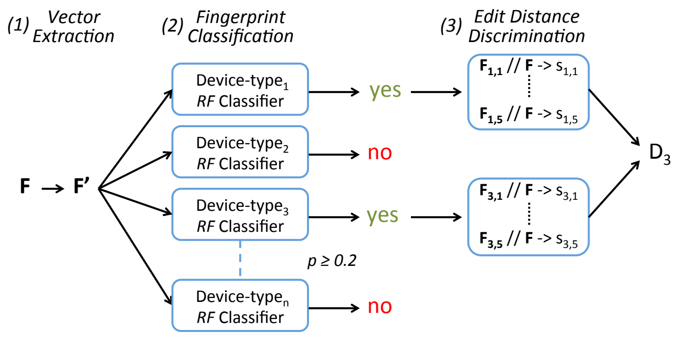

(1) Vector Extraction
(2) Fingerprint Classification
(3) Edit Distance Discrimination
F → F'
Device-type1 RF Classifier
Device-type2 RF Classifier
Device-type3 RF Classifier
Device-typen RF Classifier
yes
no
yes
no
p ≥ 0.2
F1,1 // F -> s1,1
F1,5 // F -> s1,5
F3,1 // F -> s3,1
F3,5 // F -> s3,5
D3

Fig. 3. Device-type identification process.

IoT Security Service can be deployed both locally using a personal workstation or in a public or private cloud service. In our prototype setup, we instantiate the IoT Security Service in Docker containers deployed in a private cloud environment using the Kubernetes platform [10], allowing us to achieve better scalability and fault tolerance.

*1) Device-Type Identification:* The device-type identification process involves two main steps as described in [3] and shown in Fig. 3. *Fingerprint classification* (2) consists in identifying a limited number of candidate device-types for the fingerprint **F** received from the security gateway (cf. Sect III-A1). Then, *edit distance discrimination* (3) breaks ties between candidates and decides on a final device-type for **F**. The whole identification process is implemented in Python.

First, a fixed length vector $\mathbf{F}'$ is extracted from **F** ((1) in Fig. 3) to be used for *fingerprint classification*. $\mathbf{F}'$ is a 276-dimensional feature vector and is the concatenation of the 12 first unique columns of **F**. **F** and $\mathbf{F}'$ are represented as `pandas` dataframes [11].

*Fingerprint classification* ((2) in Fig. 3) relies on $n$ classifiers, each for identifying one device-type and rendering a binary decision: is the device of the given type or not. They are implemented as Random Forest classifiers using the `scikit-learn` Python library [12]. In contrast to the original Random Forest algorithm that uses majority vote of the binary decisions of each tree to reach the final prediction, this implementation averages probabilistic predictions from each tree to predict the class of a sample. Considering 0 as "the device is not of this type" and 1 as "the device is of this type", we selected a discrimination threshold of 0.2 to be applied on the averaged probabilistic prediction. Each classifier is composed of 50 binary trees having a maximum depth of 3. Hence, the number of nodes per tree is between 7 and 15. We chose a moderate number (50) of small trees to speed up the decision process. It allows testing $\mathbf{F}'$ on a high number (>1000) of device-type classifiers in a limited amount of time (<1 second).

During *edit distance discrimination* ((3) in Fig. 3) fingerprint **F** is compared to a set of five reference fingerprints from each candidate device-type identified in (2). The comparison is done by computing the Damereau-Levenshtein distance [13] between **F** and each of the reference fingerprints. Each packet corresponding to a column in **F** is considered as a single character. All 23 packet features must be equal to consider two characters equal. The distance computation is implemented as a double loop over the columns of the two compared fingerprints. Taking two fingerprints of size $23 \times M$ and $23 \times N$, the complexity of one comparison is $O(M \cdot N)$.

*2) Isolation profile generation:* Once a device-type is identified, IoT Security Service retrieves its associated isolation profile from a NoSQL database, i.e. Mongo-DB [14]. The isolation profile (Fig. 2) is a JSON file that consists of the device-type's name, ID and an *isolation level* $\in \{\textit{strict} : 1, \textit{restricted} : 2, \textit{trusted} : 3\}$. *Trusted* devices are assigned to the *trusted network* and have unrestricted access to the Internet. *Strict* and *restricted* devices are assigned to the *untrusted network* and have no, respectively restricted, Internet access. The list of permitted_IP defines the set of remote IP addresses a restricted device can access and typically correspond to the cloud service of the device's vendor. Communications between *trusted* and *untrusted network* are strictly prohibited. The isolation profile is sent to Security Gateway in response to a device identification request, or updated asynchronously if the isolation profile for a device-type is updated by IoTSSP.

## IV. Evaluation

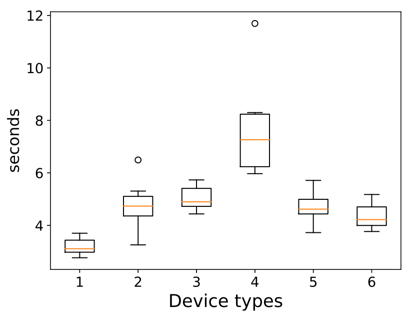
We evaluated the performance of our implementation by measuring the time it takes for the system to fingerprint and classify a newly added device and update the traffic filtering rules enforced by Security Gateway. This was done by repeating the typical set-up procedure of an IoT device for ten times, considering six different IoT device-types, resulting in a dataset of $n = 60$ distinct device set-up measurements. We captured packets for $k = 120$ seconds in each set-up experiment in order capture the typical characteristics of the device's behaviour during set-up. In our analysis, we focus on the performance of the system regarding fingerprinting and classification of the acquired data, which takes place after the packet capturing phase.

### A. Fingerprinting

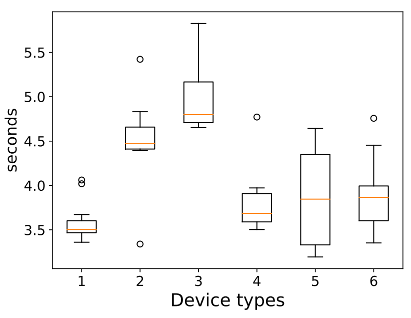
Figure 4a shows the time required for the Security Gateway to process the captured packets and generate the fingerprint **F**. The fingerprinting takes between 2770 and 11690 milliseconds, the average being $4894(\pm1559)$ ms. To reduce communication overhead the fingerprint can optionally also be compressed. The compression takes on the average $4101(\pm627)$ ms. Detailed results are shown in Fig. 4b.

### B. Classification and Enforcement Rule Update

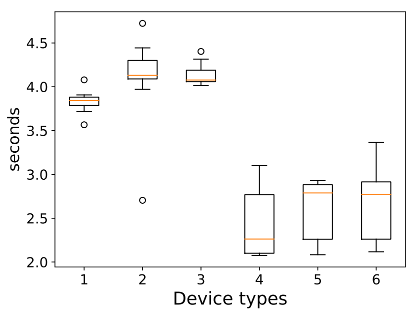
After generating the fingerprint **F**, Security Gateway sends it to the IoT Security Service to obtain a device-type classification for the device. We measured the round trip time it takes from sending the request to getting a reply from IoT Security Service with a device-type classification result. The Security Gateway and IoT Security Service were located in different countries, at approximately 1500 km distance. The results are shown in Fig. 4c. Classification takes between 2075 and 4723 ms, on average $3301(\pm806)$ ms.



(a) Fingerprint extraction



(b) Fingerprint compression



(c) Classification

Fig. 4. Box plots for the duration of fingerprint extraction, compression and classification for several device-types 1: *Fitbit Aria*, 2: *Edimax Plug 1101W*, 3: *Edimax Plug 2101W*, 4: *Ednet Gateway*, 5: *TP-Link Plug HS100*, 6: *TP-Link Plug HS110*

### *C. Overall Performance*

Both fingerprinting and classification are performed in a matter of a few seconds. The majority of the time is taken by the actual set-up procedure of the device. As the classification of the device is done very quickly after data acquisition, appropriate enforcement rules are almost immediately applied after the device setup is completed. This leaves a small window ($\approx 10$ seconds) of opportunity for potential attackers, thereby effectively protecting the user's network from potential threats posed by vulnerable devices.

## V. Related Work

Some commercial products like F-Secure Sense [15], Dojo Gateway [16], or Bitdefender Box [17] follow a similar approach in which a dedicated gateway device is used to protect the user's network. These systems, however, rely on on-line analysis of the network traffic flowing between devices and remote endpoints on the Internet with the aim of identifying potential threats or attacks based on known attack patterns or anomaly detection. In this respect, IoT Sentinel follows a totally different approach, as it aims at *proactively* identifying vulnerable devices and enforcing appropriate countermeasures even *before* potential security vulnerabilities are exploited.

For realizing automatic device identification, IoT Sentinel uses a novel *device fingerprinting* approach. In this respect, earlier device fingerprinting approaches were lacking as they were either too specific, i.e., for identifying specific device instances, or, too coarse, identifying particular combinations of chipsets and drivers that can be potentially shared between many different types of devices. For a detailed discussion of related work in device fingerprinting, please refer to [3].

## VI. Demonstration Setup

**Materials**: IoT Sentinel demo uses a Security Gateway implemented on a Raspberry Pi 2 that serves as wireless access point. The IoTSSP is hosted on a remote server. A Smartphone is used to associate several wireless IoT devices (i.e. EdnetGateway, Edimax Plug, Fitbit Aria, etc.) with Security Gateway. All Security Gateway operations are displayed on a connected monitor.

**Procedure**: The set-up of IoT devices with the Security Gateway is interactively demonstrated by inviting a person from the audience to perform the setup using a provided smartphone and the Security Gateway as access point. The audience can follow the different steps of device-type identification and update of the enforcement rules (Sect. III) on a monitor as they are performed by IoT Sentinel.

## References

[1] Senrio. 400,000 publicly available IoT devices vulnerable to single flaw. [Accessed: 2016-07-07]. [Online]. Available: http://blog.senr.io/blog/400000-publicly-available-iot-devices-vulnerable-to-single-flaw

[2] The New York Times, "Hackers used new weapons to disrupt major websites across u.s." https://www.nytimes.com/2016/10/22/business/internet-problems-attack.html.

[3] M. Miettinen *et al.*, "IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *Proc. 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*, Jun. 2017.

[4] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.

[5] MITRE Corporation. Common vulnerabilities and exposures. [Online]. Available: https://cve.mitre.org/data/downloads/index.html

[6] Big Switch Networks, "Project floodlight - floodlight OpenFlow controller," http://www.projectfloodlight.org/floodlight/, Oct. 2016, [Accessed: 2016-09-17].

[7] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2015, pp. 117–130.

[8] IEEE 802.11 AP, "Hostapd," http://w1.fi/hostapd.

[9] Flask, "Flask-Framework," http://flask.pocoo.org/.

[10] Google, "Kubernetes," https://kubernetes.io/.

[11] Pandas, "pandas," http://pandas.pydata.org.

[12] Scikit Learn, "scikit-learn," http://scikit-learn.org.

[13] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, no. 3, pp. 171–176, Mar. 1964.

[14] K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2010.

[15] F-Secure. Smart security for your smart lifestyle - F-Secure Sense. [Accessed: 2017-01-30]. [Online]. Available: https://sense.f-secure.com/

[16] Dojo Labs. Dojo. [Accessed: 2017-01-30]. [Online]. Available: https://www.dojo-labs.com/product/dojo/

[17] Bitdefender. Bitdefender BOX. IoT security solution for all connected devices. [Accessed: 2017-01-30]. [Online]. Available: http://www.bitdefender.com/box