

# Is Your Kettle Smarter Than a Hacker?

## A Scalable Tool for Assessing Replay Attack Vulnerabilities on Consumer IoT Devices

Sara Lazzaro, Vincenzo De Angelis

*Mediterranea University of Reggio Calabria  
University of Calabria*

Anna Maria Mandalari

*University College London*

Francesco Buccafurri

*Mediterranea University of Reggio Calabria*

**Abstract**—Consumer Internet of Things (IoT) devices often leverage the local network to communicate with the corresponding companion app or other devices. This has benefits in terms of efficiency since it offloads the cloud. ENISA and NIST security guidelines underscore the importance of enabling default local communication for safety and reliability. Indeed, an IoT device should continue to function in case the cloud connection is not available. While the security of cloud-device connections is typically strengthened through the usage of standard protocols, local connectivity security is frequently overlooked. Neglecting the security of local communication opens doors to various threats, including replay attacks. In this paper, we investigate this class of attacks by designing a systematic methodology for automatically testing IoT devices vulnerability to replay attacks. Specifically, we propose a tool, named REPLIoT, able to test whether a replay attack is successful or not, without prior knowledge of the target devices. We perform thousands of automated experiments using popular commercial devices spanning various vendors and categories. Notably, our study reveals that among these devices, 51% of them do not support local connectivity, thus they are not compliant with the reliability and safety requirements of the ENISA/NIST guidelines. We find that 75% of the remaining devices are vulnerable to replay attacks with REPLIoT having a detection accuracy of 0.98-1. Finally, we investigate the possible causes of this vulnerability, discussing possible mitigation strategies.

**Index Terms**—Internet of Things, replay attack, security, privacy, IoT device

### I. INTRODUCTION

Consumer Internet of Things (IoT) devices, such as smart TVs, speakers, surveillance cameras, and appliances, offer numerous advantages to their users [1]–[3]. These devices can be managed remotely by the users through smartphone apps connected to cloud platforms. However, in most cases, to optimize the performance, when the smartphone is in the same network of the device, local communication between the app and the device is enabled. In this way, the latency time is reduced [4] and the device still works even if the remote connection is not available. This latter point is crucial for reliability and safety reasons. This principle is reported in the ENISA security guideline GP-TM-17 [5], which states that IoT devices should continue to work if the cloud back-end

fails. Indeed, despite a user may be in the same network of the IoT device, physical access to the latter may still be hard (e.g., for impaired users or in case the device is difficult to reach because of its position). In these cases, the loss of connectivity to the Internet should not limit the possibility for the user to control a device that may have a safety impact (e.g., to turn off a smart oven, open a smart lock, start a camera recording, etc.).

On the other hand, enabling local connectivity requires proper security measures as highlighted by the security guidelines of ENISA [5] and NIST [6]. As a matter of fact, each device should not rely on the security of firewalls or other security practices implemented in the access point but should have its own security mechanisms. As witnessed recently by Girish et al. [7], despite its importance, the security of IoT local connectivity has not been deepened in the scientific literature.

In this paper, we focus on one of the main threats arising in the local network, i.e., replay attacks [8], [9]. Replay attacks are performed by eavesdropping the network traffic, intercepting it, and replaying the packets after a given time.

In smart homes [10], this class of attacks is very relevant for two reasons. First, replay attacks can be easily performed by a single malicious device inside the local network [11]–[13]. We expect this scenario to become increasingly likely given the proliferation of IoT devices [14], [15]. Second, replay attacks may be part of a more complex attack strategy, serving as a means to furnish adversaries on insights into the controllability of a device [16]. While the prior knowledge of the target device may enable ad hoc solutions to perform replay attacks [17], a challenging question is whether they can be automatically performed on generic devices. To the best of our knowledge, no papers in the scientific literature pursue this ambitious goal.

In this work, we propose REPLIoT, a tool for automatic testing replay attacks in a smart home environment. The tool is designed to be *agnostic* and works with any type of device and communication protocol. Our innovative methodology is based on black-box processing of the sniffed traffic to increase the chances of making the attack successful. Being our tool device-agnostic, it can also be used in a household environment by non-technical users [18]. To this aim, we equip REPLIoT with a detection module that works on the

*The two lead authors contributed equally to this work.*

device's responses to automatically detect whether the attack is successful or not. This module is particularly relevant in the case in which the user launching the tool has no physical access to the device to observe the effect of a replayed command. Furthermore, there exist commands that may not trigger an external change in the device (e.g., a change in the sensitivity level of a motion detection sensor).

Our key research contributions include:

- We develop an automated methodology for large-scale testing replay attack vulnerabilities on IoT devices;
- We demonstrate the feasibility of detecting the success of the attack;
- We assess the (in)effectiveness of 41 popular IoT devices in preventing such attacks;
- and finally, we examine the potential causes behind these vulnerabilities, shedding light on the factors contributing to these security weaknesses.

We find that 21 IoT devices do not support local connectivity in contrast to the reliability and safety guidelines of ENISA [5]. Out of the 20 remaining devices, 15 are vulnerable to replay attacks through our tool. We perform thousands of automated experiments to validate REPLIOT and show the effectiveness of the designed detection module. The code of our tool and the data collected in our experiments are publicly available at: <https://github.com/SafeNetIoT/ReplayAttack>.

**Responsible Disclosure.** We responsibly disclosed our results with the IoT manufacturers in this study. We received responses from one manufacturer. We include these responses (with permission) in Section VI.

## II. ASSUMPTION, GOALS AND NON-GOALS

In this section, we set the assumptions, the goals, and the non-goals for this work.

### A. Threat Model

We consider the following threat model.

**Victim.** The victim is any person who uses or benefits from consumer IoT devices.

**Adversary.** The adversary is any party that can access the local IoT device traffic. Examples include internal privacy and security threats, and malicious IoT devices, placed within the local network, with the ability to sniff promiscuously [19]. Remote attackers can also exploit router vulnerabilities (e.g., default passwords) to access the local network [20]. We observe that when our tool is adopted for defensive purposes, it assumes the role of an adversary (without leading to threats). It can be placed directly on the access point or a device connected to the same network.

**Threat.** The adversary can trigger commands on smart home IoT devices, thus instructing them to perform some actions without the victim's will.

**Plausibility.** Several security guidelines of ENISA [5] and NIST [6] (Logical Access to Interfaces) witness the importance of protecting the local network segment. Furthermore, also the scientific literature acknowledges this problem [19], [21], [22]. As highlighted by Miettinen et al. [19], also

remote attacks through "NAT hole punching" [23], may be effective by compromising a device (e.g., a smartphone) with access to the local network. As a result, devices should not depend on the security provided by firewalls or other measures implemented in the access point. This aligns with the GP-TM-43 of ENISA [5].

### B. Goals

The main goal of this work is to design a methodology to automatically verify whether an IoT device might be vulnerable to replay attacks. In particular, this work answers the following research questions (RQ):

**RQ1:** *Can replay attacks against IoT devices be automated without prior knowledge of the device-specific features?* To answer this question we build a tool (REPLIOT) able to actively perform replay attacks against generic IoT devices. We also design a methodology to employ this tool for a large-scale study of replay attack vulnerabilities of IoT devices.

**RQ2:** *Is it possible to automatically detect whether a replay attack is successful?* Since also non-technical users can adopt our tool in a real environment, we design a methodology to automatically detect whether a replay attack is successful on a given device. Overall, our tool is conceived to minimize human intervention.

**RQ3:** *Does the attack apply to a variety of IoT devices?* To answer this question, we employ commercial IoT devices spanning various vendors and categories.

### C. Non-Goals

In this work, we do not consider the following as goals.

**Development of an Intrusion Detection System** [24]. The goal of our tool is not to verify that a device is undergoing a replay attack. On the contrary, our tool acts preemptively by identifying potential device vulnerabilities, regardless of whether these devices may be in a potentially protected network segment (see GP-TM-43 of ENISA [5]).

**Design of ad hoc procedures for vulnerability testing.** Our tool is based on network traffic analysis, thus it does not require any knowledge of the device under test.

**Usability testing.** We do not perform usability testing on our tool. Even though REPLIOT is deployed to be used in the home environment and not just in a lab environment, some refinements are needed to enhance its user-friendliness. However, addressing these aspects is beyond the scope of this paper.

## III. METHODOLOGY

We answer our research questions by proposing a tool to automatically test the replay attack vulnerabilities of an IoT device. Our tool works with consumer IoT devices that are managed through a proper companion app provided by vendors. REPLIOT is based on network traffic analysis. At a high level, it works as follows. First, we sniff the local traffic (if any) exchanged between the companion app and the target device. Then, we replicate in a proper fashion the sniffed traffic and monitor the (possible) responses received by the device.

Finally, we analyze these responses to automatically check the success or failure of the performed attack.

REPLIoT presents three modules: Training Module, Attack Module, Detection Module. Overall, each of the three modules corresponds to a different phase in which the tool works. In the following sections, we describe the three modules in detail.

#### A. Preliminary Considerations

1) *Machine Learning Algorithms Selection*: REPLIoT implements a detection module to automatically check the success or failure of the performed attack. This module includes a machine learning (ML) algorithm that falls in the class of anomaly detection, specifically novelty detection [25]. We train the algorithm using the responses sent by the device to the companion app. When REPLIoT performs the attack, it also detects its success or failure by feeding the ML algorithm with the obtained responses.

The intuition is that, when the replay attack works, the responses received by REPLIoT are similar to those exchanged by the device with its companion app. We refer to this type of response as *regular response*. On the other hand, when the attack does not work, REPLIoT does not receive any response or the device sends an error message. Such error messages should appear different from legitimate responses (with which the ML algorithm is trained). We refer to the responses carrying the error messages as *irregular responses*. The ML module distinguishes between regular and irregular responses. We observe that, in the ML terminology, an irregular response is considered a novelty. Novelty detection requires that the training instances do not contain any anomalies. This fits our scenario in which no error message is expected during the legitimate use of the device with the companion app.

2) *Observed Types of Responses*: From a preliminary study, we observe four types of responses in the local traffic.

- **Full Cleartext Responses**: words from natural language, mostly organized in a structured format (e.g., JSON).
- **Standard Encrypted Responses**: responses encrypted through standard protocols, e.g., TLS, QUIC, etc.
- **Non-standard Encrypted Responses**: responses encrypted through non-standard protocols, which can also carry cleartext metadata along with encrypted data.
- **Encoded Responses**: responses that are neither encrypted nor expressed in natural language, but they contain fixed bytes encoding some messages of proprietary protocols.

When a device executes a command (triggered by REPLIoT), if it replies with cleartext or encoded messages, we expect they present some similarity with the responses provided by the device when it is triggered by the legitimate companion app. Indeed, when no encryption mechanism is adopted, the responses to the same command are exactly the same or present minor changes (e.g., identifiers or timestamps). Then, our ML algorithm can detect them as regular responses. On the other hand, when the devices do not execute a command (the replay attack does not work), we expect that the responses (if any) include error messages not included in

the legitimate responses. Then, the ML algorithm can detect these responses as irregular.

Regarding standard encrypted responses, we expect that the replay attack does not work and the ML module is not applicable, due to the fact that the encryption of two identical cleartext messages results in different encrypted payloads.

Finally, when a device uses non-standard encrypted responses, we cannot predict a well-defined behavior. The response may present insufficient randomness, suffer from key reuse, or contain some cleartext metadata. Then, it is possible that the responses to the same command present a given degree of similarity, and the replay attack may work. Otherwise, similarly to standard encrypted responses, we expect the replay attack does not work.

#### B. Training Module

The training phase is devoted to the collection of some responses of the target device to train the ML model. We obtain the responses by triggering some functions on the device through its companion app and sniffing the traffic. We collect all network traffic traversing the testbed using *tcpdump*.

We set as a usability requirement the fact that the ML model should perform well with a few training data.

Once collecting them, REPLIoT processes the responses, via *Pyshark* [26], a Python tool adopting Wireshark dissectors. Specifically, we extract the payloads of the transport level (TCP or UDP) messages to train the ML model.

#### C. Attack Module

We trigger the attack when a function (e.g., switching on/off the light of a smart bulb, watching live from a camera, etc.) on the target device is performed. The aim is to understand if our tool can trigger the same function at a later time.

The first step of this phase consists of sniffing the traffic exchanged between the legitimate companion app and the target device via *tcpdump* when this function is triggered.

We process the captured traffic, through *Pyshark*, by extracting the transport level payloads. However, differently from the training phase, we collect both the requests and the responses. Subsequently, we organize the extracted payloads in *flows*. Each flow is represented by a list of consecutive requests from the companion app to the device and a list of consecutive responses from the device to the companion app. Once the flows are collected, the responses can be discarded, and we are ready to launch the attack.

The attack consists of replaying the requests of each flow and storing the received responses (if any). The list of flows is organized as a stack so that the last flow will be the first to be replayed. The rationale behind this empiric design choice is that, for some devices, it may happen that, to trigger a specific function, several requests and responses (and then several flows) are exchanged. However, the first flows may not contain the actual command (that triggers the function) but just accessory information (e.g., the information needed to exchange some secret or synchronization messages). On the other hand, it is more likely that the actual command

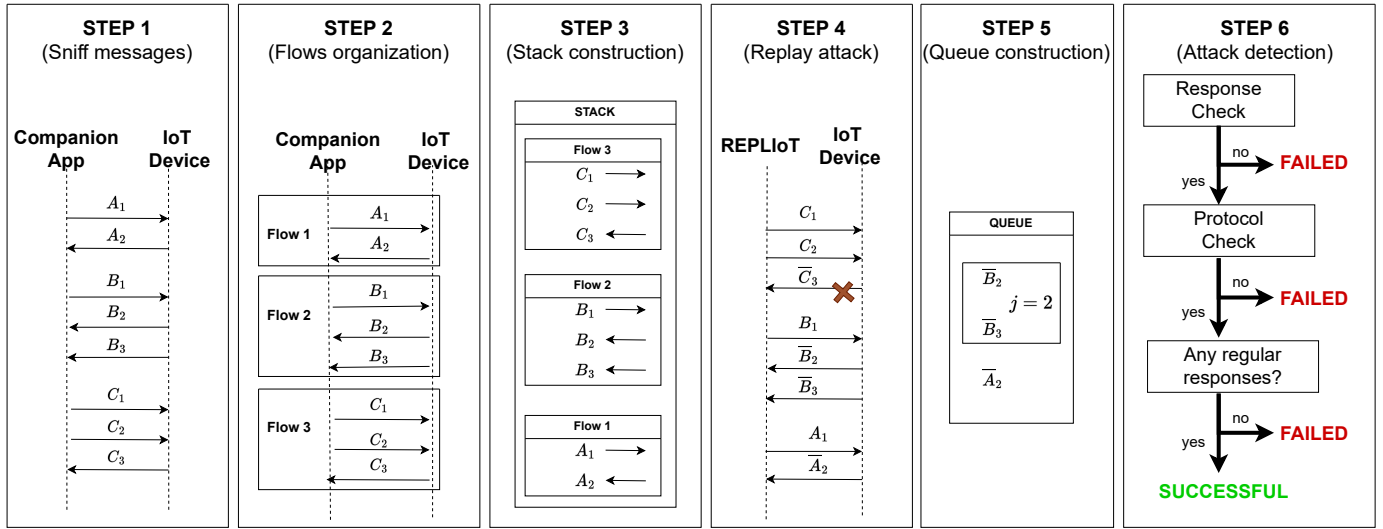


Fig. 1. REPLIoT Operation Steps.

is contained in the last flows. Moreover, we observe that starting the attack by the first flow may change the state of the target device. Thus, when the tool replicates the flow containing the actual command, the attack may fail because of the state change of the device and not because of an effective countermeasure implemented on the device. However, being our tool device-agnostic there is a lack of prior knowledge about which specific flow contains the actual command. Hence the need to replicate all flows rather than solely focusing on the last one.

#### D. Detection Module

The aim of the detection module is to automatically detect whether the replay attack is successful.

We perform two preliminary checks: (i) response check and (ii) protocol check.

**Response Check:** If our tool receives no response during the attack phase, the attack is considered **FAILED**.

**Protocol Check:** If the companion app and the device communicate through standard security protocols (i.e., TLS, QUIC, etc.), the attack is considered **FAILED**.

In all the other cases, we assume a list of responses organized in a queue (the first response received during the attack phase is the first response scheduled in the detection phase). We recall that, the flows during the attack phase are scheduled in reverse order with respect to the order of arrival. We then expect that the response of an actual command (triggering a function) is contained in the first response stored by the queue. In particular, as a heuristic approach, we consider the first  $j$  responses received.

The ML algorithm receives as input these responses to check if they are detected as regular or irregular. If **all** the  $j$  responses are detected as irregular, we consider the attack **FAILED**. Otherwise, we consider the attack **SUCCESSFUL**. We assume that the fact that a device sends at least one regular response to a replayed request may be a security issue. This is the reason

for which we consider the attack **SUCCESSFUL** when at least one response is detected as regular.

On the other hand, if in place of  $j$ , we feed all the responses sent by the target device to the ML algorithm, the chance of having false positives could be very high. To explain this, we consider an example. We suppose that, before sending a command, the companion app performs an initial handshake with the IoT device (e.g., exchanging keys). Then, by replaying the first request sent by the app (i.e., a request belonging to the initial handshake), the target device would most likely send a regular response. Then our tool would consider the attack as successful based on this response only, while it may not succeed in triggering the desired command. This is the reason why we limit the detection to the last  $j$  responses.

#### E. REPLIoT Operation

In this Section, we provide an example of the application of REPLIoT. Figure 1 shows how REPLIoT operates. We consider 6 steps, described below.

**Step 1:** We assume that a series of requests/responses are exchanged between the smartphone and the IoT device in the order shown in Step 1 of Figure 1.

**Step 2:** We sniff the traffic and organize it in flows. As an example, we consider that Flow 1 includes a single request ( $A_1$ ) and a single response ( $A_2$ ). Flow 2 includes a request ( $B_1$ ) and two responses ( $B_2$  and  $B_3$ ). Flow 3 includes two requests ( $C_1$  and  $C_2$ ) and a response ( $C_3$ ).

**Step 3:** We perform the attack by scheduling the Flows in reverse order.

**Step 4:** We replicate the captured requests to the device. In this example, we take into account that the device may not provide some responses (e.g., because the state of the device is changed and the requests sent are not meaningful anymore). Specifically, in this example, the response to the requests  $C_1$  (or  $C_2$ ) of Flow 3 is lost. We observe that, in general, the received responses during the attack are not

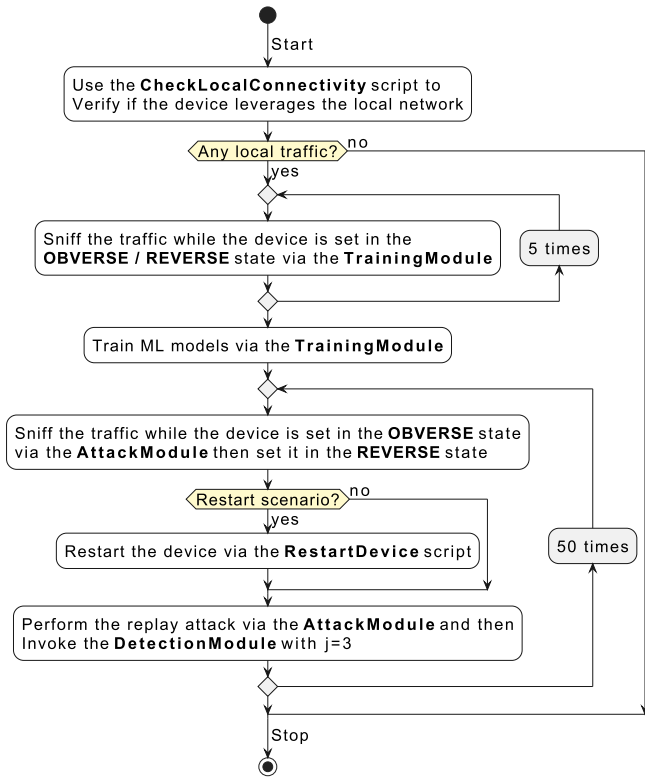


Fig. 2. Procedure to Validate REPLIOT.

identical to the original responses (for example, they may contain different identifiers or timestamps). To highlight this, we use the overline notation ( $\overline{C_3}, \overline{B_2}, \overline{B_3}, \overline{A_2}$ ).

**Step 5:** We store the responses in the order we receive them for creating the queue.

**Step 6:** Finally, we use the responses to feed the detection module. If the **Response Check** and the **Protocol Check** do not output **FAILED**, then the first  $j$  responses are fed into the ML algorithm. In this example, we consider  $j = 2$ .

#### IV. VALIDATION

In this section, we describe the procedure we follow to validate REPLIOT and assess the effectiveness of 41 popular IoT devices in preventing replay attacks. Figure 2 shows the steps of our validation.

##### A. Testbed

In order to have a controlled environment for testing the vulnerability to replay attacks of IoT devices, we build an IoT testbed. Our testbed consists of: (1) 41 consumer IoT devices, (2) a smartphone in which we install the companion apps to control the IoT devices, (3) an access point (installed on a server) to which the smartphone and the target IoT devices are connected, (4) a set of support scripts to automatically turn on and off an IoT device, trigger a function, and determine whether the device has local connectivity.

TABLE I  
OBVERSE AND REVERSE STATES PER IOT DEVICE

OBVERSE-REVERSE State	Device
ON - OFF	Arlo pro 4 camera, Govee lightstrip, Furbo camera, Lepro bulb, Lifx bulb, Meross smartplug, Nanoleaf triangle, Tapo smartplug, Wiz lightbulb, Wyze cam pan, Yee-light bulb, Yeeligh lightstrip
START - STOP	Coffee maker Lavazza, Cosori airfrier, Eco-vacs vacuum, Eufy roborvac, iRobot roomba i7, OKP vacuum, Swan Alexa Smart Kettle, WeeKett Kettle, Xiaomi blender
PLAY - PAUSE	Blink camera, Blink video doorbell, Bose Speaker, Boyfun Baby monitor, Ring doorbell, Ring spotlight camera, Simplisafe camera, Sonos Speaker, Vtech baby monitor
ACTIVATE - DISACTIVATE	Aqara hub, Eufy doorbell, Meross Garage Opener, Switchbot hub mini
MAX - MIN VOLUME	Echodot 4, Echodot 5, Eufy doorbell chime
POSITIVE - NEGATIVE OFFSET	Netatmo Weather Station
SET KG-SET LB	Withings smart scale
SET °C-SET °F	Withings thermo
LOCK-UNLOCK	Petsafe Automatic Feeder

1) *IoT Devices:* The devices we consider are consumer IoT devices typically deployed in a smart home. We have chosen devices under different categories (see the second column of Table I). The devices are simultaneously connected to the network. We distinguish them by their MAC address.

2) *Smartphone:* We use a standard smartphone (Google Pixel 3a) to manage the IoT devices through their apps.

3) *Access point:* The access point offers IP connectivity to the IoT devices under test, and the ability to capture and control the network traffic exchanged between the IoT devices and the apps. REPLIOT is installed on the access point.

4) *Support Scripts:* We deploy the following 4 support scripts to automatize the validation procedure. The `CaptureCoordinates` script records the coordinates of the screen taps when we trigger a function to set the device in the **OBVERSE** or **REVERSE** state. The `ProbeCommand` takes such coordinates as input and uses them to automatically set the device in the **OBVERSE** or **REVERSE** state. The `CheckLocalConnectivity` script verifies whether the device and the companion app communicate through the local network. Finally, the `RestartDevice` script physically restarts a device by turning off a plug powering the device, and then turning it on.

In the following, we describe the steps followed to conduct the validation procedure.

##### B. State Phase

This phase consists of finding the state in which the IoT device can be set in the companion app. We assume that an IoT device cannot be in two states at the same time (e.g., the ON and OFF states). We call one of these states as **OBVERSE** (i.e., smart light on) state and the other state as **REVERSE** (i.e., smart light off).

Table I shows the state **OBVERSE** and **REVERSE** considered for each IoT device in the testbed. For example, for the vacuum cleaners, the air fryer, the kettles, and the blender we consider the START-STOP state. The speakers and several cameras offer the PLAY-PAUSE state. For Withings smart scale and Withings thermometer, the states are represented by the unit of measurement for weight and temperature.

Since our experiments require to trigger the states several times, we adopt an automatic procedure for triggering them inspired by Mandalari et al. [27]’s study. Specifically, we leverage the Android Debug Bridge (ADB) tool to programmatically trigger the function of the device by emulating the user’s taps on the companion app. At first, we follow a one-time procedure. We manually perform some taps in the companion app to trigger the **OBVERSE** state and the **REVERSE** state. Meanwhile, the `CaptureCoordinates` script records the coordinates of the screen taps. These coordinates are the input of the `ProbeCommand` script. We then leverage this script to automatically set the device to the desired state.

To validate the correct execution of the `ProbeCommand` script (setting the **OBVERSE** or **REVERSE** state on the device), we employ the following procedure. Each time we invoke the `ProbeCommand` script, we capture the screenshot of the companion app and compare it to a previously retrieved screenshot where we know that the command was correctly executed. For example, we invoke the `ProbeCommand` script to turn on the smart light bulb and we capture the screenshot on the app to check whether the bulb is actually in state ON.

### C. Check Local Connectivity

In this phase, we filter out the devices that do not communicate with the companion app in the local network but only via the cloud. In general, some devices may work by default through the cloud even though the companion app and the IoT device are in the same network. To verify this, we leverage the `CheckLocalConnectivity`. This script forces the devices to communicate with their companion app via the local network (i.e., we prevent the devices from connecting to the Internet) by setting firewall rules at the access point. This script starts using `tcpdump` to capture the traffic in the local network. Meanwhile, we invoke the `ProbeCommand` script to set the device in the **OBVERSE** and the **REVERSE** state. The `CheckLocalConnectivity` script checks the presence of packets during this phase. If no packets are detected then the validation procedure ends (see Figure 2).

### D. Training Phase

This phase starts by invoking the `Training Module`. We recall that this module first launches `tcpdump` to sniff the traffic sent from the device to the companion app when the latter issues some commands. We automatically trigger these commands via ADB through the `ProbeCommand` script. Specifically, we first invoke this script to set the device in the **OBVERSE** state and then to set the device in the **REVERSE** state. We repeat this procedure five times.

Overall, the training set is composed of the responses to ten commands sent by the companion app (setting of **OBVERSE** and **REVERSE** states five times), along with possible control messages (e.g., synchronization messages or handshake messages). In the home environment, this procedure (i.e., the trigger of some commands) is performed manually by the users. We intentionally choose a small number of times (five) in which the two commands are repeated with the aim to easily get the dataset in the real-life application of the tool by users. Finally, the `Training Module` feeds the responses collected during the sniffing time to an ML model. For our validation, we select 3 standard models of ML for novelty detection [25], [28], [29]: Isolation Forest, Elliptic Envelope, and Local Outlier Factor.

### E. Attack Phase

During this phase, we first invoke the `ProbeCommand` script to set the device in the **OBVERSE** state. The `Attack Module` of REPLIOT sniffs and processes the traffic exchanged by the device and the companion app (as described in Section III-C). We then invoke again the `ProbeCommand` script to set the device in the **REVERSE** state.

At this point, we perform the replay attack. This attack is successful if REPLIOT manages to switch the device from the **REVERSE** state to the **OBVERSE** state. On the other hand, if the device remains in the **REVERSE** state, we assume the attack fails. As ground truth, to check the final state of the device (**OBVERSE** or **REVERSE**), we perform the screenshot of the companion app screen and compare it with a previously captured screenshot taken when the device is known to be in the **OBVERSE** state.

We perform the above procedure in two different scenarios.

- **Non-Restart Scenario:** in this scenario, we first set the device in the **REVERSE** state and then we perform the replay attack.
- **Restart Scenario:** in this scenario, we first restart the device via the `RestartDevice` script, and then we set the device in the **REVERSE** state. Finally, we perform the attack.

The two scenarios capture two different situations. The **Non-Restart Scenario** represents a real-life scenario, in which we simulate a replay attack after a user legitimately set the device in the **REVERSE** state. In this case, the effect of the replay attack is to change the state of the device against the will of the user. We point out that this is not the simplest scenario in which we test REPLIOT. Indeed, a *Real-time* attack, in which we replicate a command immediately after capturing it, may have more chances to be successful. However, the impact of such an attack would be minimal since the device would be set in the same state desired by the user. On the other hand, the **Restart Scenario** is a more challenging scenario in which to test our tool. Indeed, the restart forces the device to clear its working memory (possibly negotiated temporary keys, tokens, agreed ports, etc.). When the **REVERSE** state is set, there are more chances that the device would possibly renegotiate some values (i.e., keys, tokens, ports, and so on), thus making the



attack ineffective. In our validation, we perform the attack 30 seconds after the restart to give the device the time to run the initial configuration procedure.

#### F. Detection Phase

In this phase, we invoke the Detection Module of REPLIoT. We recall that this module takes as input the packets (properly pre-processed) sniffed by the Attack Module. Moreover, this module requires a parameter  $j$  to be set representing the number of responses to provide to the ML algorithm. In our validation procedure, we set  $j = 3$ . This value is set empirically by testing manually a subset of devices.

To evaluate the performances of the Detection Module, we invoke the Attack Module 50 times. For each of these, we invoke the Detection Module that returns a boolean output: whether the replay attack is successful or not. We observe that the ground truth remains the same for all 50 experiments, i.e., the attack either always works or does not. To measure the Detection Module performance, for a given device, we consider the *accuracy* measured as the number of times in which the Detection Module successfully classifies the attack over the total number of experiments. The *accuracy* coincides with the *recall* while the *precision* is not significant because it is always 1. Indeed, by considering as positive instance the result of the ground truth, there are no false positive and true negative instances. Then, the *accuracy* (i.e., the recall) is the only meaningful metric to consider.

#### G. Validation through API

In case no local traffic is exchanged or the replay attack is not successful, we consider an additional step. We check whether the IoT device maintains some open ports for developing personal applications and communicates with the device through some APIs [30]. This interaction is referred to as local APIs. The main goal of our investigation remains the interaction between the companion app and the target device.

To test the robustness of devices when APIs are adopted, we implement ad hoc scripts for these devices, emulating the behavior of the smartphone through APIs. Then, we repeat the methodology described in Figure 2, with the difference that, this time, the device is not set in the **OBVERSE/REVERSE** state through the `ProbeCommand` script but through customized scripts using the available APIs.

### V. RESULTS

We answer our research questions by applying the methodology described in Section III and IV to understand whether it is possible to automate replay attacks against IoT devices and automatically detect whether the attack is successful.

#### A. Local Connectivity

By leveraging the procedure described in Section IV-C, we find that out of the 41 devices of our testbed, 22 do not communicate with the related app through the local network. Among the devices that do not use local connectivity, one (the Govee lightstrip) uses local APIs. This means that 21 devices

TABLE II  
REPLAY ATTACK RESULTS. ✓ INDICATES WHETHER THE REPLAY ATTACK IS SUCCESSFUL OR NOT (X).

Device (*Tested via APIs)	Non-Restart Scenario	Restart Scenario
Bose Speaker *	✓	✓
Boyfun Baby monitor	X	X
Eufy robovac	✓	✓
Furbo camera	X	X
Govee lightstrip *	✓	✓
iRobot roomba i7	X	X
Lepro bulb	✓	✓
Lifx bulb	✓	✓
Meross smartplug	✓	✓
Meross Garage Opener	✓	✓
Nanoleaf triangle *	✓	✓
OKP vacuum	✓	✓
Sonos Speaker *	✓	✓
Tapo smartplug	✓	X
Vtech baby monitor	X	X
WeeKett Kettle	✓	✓
Wiz lightbulb	✓	✓
Wyze cam pan	X	X
Yeelight lightstrip	✓	✓
Yeelight bulb	✓	✓

out of 41 are not compliant with the principle stated in the ENISA security guideline (GP-TM-17 [5]), stating that an IoT device should continue to function if the cloud back-end fails.

**Takeaways.** The finding that, among 41 devices, 21 lack local connectivity, is inherently significant. It underscores the pressing need for substantial improvements in the reliability of IoT devices. In addition, allowing devices to work independently of the cloud enables users to have greater control over their data and reduces the risk of data exposure during cloud outages or breaches. Finally, local connectivity reduces the dependency on the cloud, thus resulting in lower latency and energy saving.

#### B. Replay Attack

Table II shows the result of the performed replay attacks on the 20 devices that leverage local connectivity in the two considered scenarios (i.e., **Non-Restart** and **Restart**). To fairly validate our results, besides using the screenshot procedure as described in Section IV-E, we manually verify whether the replay attack is working or not for each device. As reported in Section IV-G, in case no local traffic is exchanged or the replay attack is not successful, we test whether IoT devices present vulnerable local APIs. This happens for 4 devices (i.e., Govee lightstrip, Bose speaker, Sonos speaker, and Nanoleaf triangle). We denote these devices by \* in Table II. Specifically, as reported in Section V-A, the Govee lightstrip does not leverage the local network when communicating with the companion app. On the other hand, the Bose speaker and the Sonos speaker leverage TLS to communicate with their companion apps. Then, our tool is not able to perform the attack. Finally, the Nanoleaf triangle uses a secure proprietary protocol based on HTTP to communicate with its companion app, and again the attack is not successful. Despite this, we found that these devices present local APIs vulnerable to replay attacks.

TABLE III  
RESPONSE DISTRIBUTION

Response	Device
<b>Cleartext</b>	Bose Speaker, Govee lightstrip, Meross Garage Opener, Meross smartplug, Nanoleaf triangle, Sonos Speaker, Wiz lighbulb, Yeelight bulb, Yeeligh lightstrip
<b>Standard Encrypted</b>	iRobot roomba i7
<b>Non-Standard Encrypted</b>	Boyfun Baby monitor, Furbo camera, Tapo smartplug, Vtech baby monitor, Wyze cam pan
<b>Encoded</b>	Eufy robovac, Lepro bulb, Lifx bulb, OKP vacuum, WeeKett Kettle

TABLE IV  
EXAMPLES OF RESPONSES' PAYLOAD

Device	Payload
<b>WeeKett Kettle</b>	ER;ñü@§EDlyÿ:ÇÆFilyB6PU <sup>o</sup> Q <sup>u</sup> U
<b>Tapo smartplug</b>	{{"error_code":0,"result":{"response":"CO07WBT2xBhRL05oIZbhAEuf/FjQEEa596JE3+X1ubE="}}}
<b>Meross Garage opener</b>	"header":{"messageId":"08..6c","namespace":"Appliance.System.DNDMode","triggerSrc":"Android-Local","method":"SETACK","payloadVersion":1,"from":"/appliance/22..1b/publish","timestamp":16..90,"timestampMs":814," <b>sign</b> ":"41..3d"},"payload":{}}

Surprisingly, among the 20 devices, 15 are found to be vulnerable to the replay attack. This attack successfully exploits all the devices in the **Non-Restart Scenario** and is effective against 14 of them in the **Restart Scenario**.

To investigate the reasons behind our results, we manually study the response payloads of the 20 devices during the replay attack. As reported in Section III-A2, we observe four types of responses in the local traffic: **full cleartext**, **standard encrypted**, **non-standard encrypted**, **encoded**. Table III describes the type of responses adopted by each device. We observe that 9 devices adopt cleartext responses, 5 devices use encoded responses, 5 devices adopt non-standard encrypted responses, and only 1 device (i.e., the iRobot roomba i7) employ standard encrypted responses (i.e., TLS).

Table IV describes response payloads interesting to discuss.

In the **Restart Scenario**, the 14 devices vulnerable to replay attacks all communicate through protocols that adopt either encoded or cleartext responses. In Table IV, we give an example of an encoded response for a smart kettle. This response (and also the associated request) always presents the same payload. These devices do not perform any authentication with the companion app after rebooting.

The Tapo smartplug is vulnerable only in the **Non-Restart Scenario**. This device leverages a proprietary protocol that includes partially encrypted responses (see Table IV). However, this protocol is not secure since an encrypted packet remains valid over time until the device is rebooted (**Restart Scenario**). In this case, the device exchanges a new key with the companion app and thus the attack is not successful. We point out that the **Non-Restart Scenario** is the most plausible in real-world situations.

Two devices that deserve attention are the Meross smartplug and the Meross Garage opener. These devices leverage clear-

text messages that include a signature to prevent commands from being altered. However, the signature remains valid when the tool replicates the same message so that the devices are vulnerable. We report in Table IV an example of a response that includes such a signature (highlighted in red).

We now focus on the remaining 5 devices in which the replay attack is not successful. Four of the devices are smart cameras. A manual inspection reveals that the cameras examined use (on the local network) proprietary protocols that do not appear to be vulnerable to REPLIoT. We observe that this does not guarantee that they are not vulnerable to replay attacks as they do not adopt standard security protocols. As such, our findings represent a lower-bound of such vulnerability, using an approach that can be automated. The iRobot roomba i7 communicates with the companion app via the TLS protocol. Generally speaking, the adoption of standard security protocols should be preferable to counter replay attacks.

**Takeaways.** Our findings demonstrate the effectiveness of REPLIoT. Despite its agnostic nature, REPLIoT can automatically identify 15 out of 20 devices vulnerable to replay attacks, thus positively answering the research questions **RQ1** and **RQ3**. In addition, our manual investigation highlights that IoT devices use proprietary protocols with weak (or not) security measures. This makes them vulnerable to replay attacks. Finally, out of 20, 14 devices are still vulnerable to replay attacks in the **Restart Scenario**. This denotes a lack of an authentication procedure. To prevent the attack from working, well-known security protocols with mutual authentication mechanisms (e.g., TLS with embedded certificates) should be adopted.

### C. Detection Module

As reported in Section IV-D, we investigate three different ML algorithms. We find that the Local Outlier Factor algorithm outperforms the other two algorithms. Specifically, with the Local Outlier Factor model, the accuracy of the Detection Module ranges (over all the devices) from 0.98 to 1 for both the **Restart** and **Non-Restart** scenarios. Concerning the Isolation Forest model, we have an accuracy of 0% for a single device, while for the other devices the accuracy ranges from 0.98 to 1. Regarding the Elliptic Envelope model, we have an accuracy of 0% for two devices, while for the other devices the accuracy ranges from 0.78 to 1.

We recall that the Detection Module performs two preliminary checks, before (possibly) feeding the responses to the ML algorithm. The **Response Check** consists of verifying if the device under test responds to our tool during the attack phase. It outputs **FAILED** when our tool receives no responses. The **Protocol Check** consists of verifying if the device adopts standard security protocols for local communication. If so, it outputs **FAILED**. If none of the two checks output **FAILED**, the Detection Module feeds the received responses to the ML algorithm.

Table V shows how the Detection Module works for each device. Specifically, we observe that for 4 devices the **Response Check** outputs **FAILED** and for 1 device the



TABLE V  
DetectionModule RESULTS

Detection	Device
Response check	Boyfun Baby monitor, Furbo camera, Vtech baby monitor, Wyze cam pan
Protocol check	iRobot roomba i7
ML intervention	Bose Speaker, Eufy robovac, Govee light-strip, Lepro bulb, Lifx bulb, Meross Garage Opener, Meross smartplug, Nanoleaf triangle, OKP vacuum, Sonos Speaker, Tapo smartplug, WeeKett Kettle, Wiz lighbulb, Yeelight bulb, Yeelight lightstrip

**Protocol Check** outputs FAILED. There are two main reasons to explain the high accuracy values of the `DetectionModule`. First, the two preliminary checks (avoiding the intervention of the ML algorithm) allow us to effectively detect that the replay attack does not work. This is the case for 5 devices where either our tool receives no response or the devices use secure communication protocols. Secondly, when the ML algorithm is invoked (for the remaining 15 devices), the responses fed to it have many similarities. Indeed, by manual investigation, we observe that for some devices the responses are exactly the same while for other devices just a few fields (e.g., identifiers and timestamps) change. Hence, the ML algorithm readily identifies this similarity.

**Takeaways** The performance achieved by the `DetectionModule`, included in REPLIOT, shows an accuracy value ranging from 0.98 to 1, thus positively answering the research question **RQ2**. Accuracy is crucial for determining the efficacy of our tool for autonomously detecting vulnerabilities to replay attacks in IoT devices. This is fundamental since the tool’s intended adoption is by non-technical users in domestic settings. The detection module is particularly relevant in the case in which the user launching the tool has no physical access to the device to observe the effect of a replayed command, or the effect of the command may not trigger any visible changes in the device status.

The results obtained encourage us to believe that our tool can be adopted (beyond the lab) by non-technical users to detect whether the IoT devices they own are vulnerable to replay attacks.

## VI. DISCUSSION

In this section, we discuss the implications of our findings, possible mitigation, limitations, and ethical considerations.

**Safety and Reliability implications.** We highlight that the absence of local connectivity (detected in more than 50% of our devices) may harm the safety and reliability of an IoT device. This principle is stated in the ENISA security guideline GP-TM-17 [5], stating “*essential features should continue to work with a loss of communications and chronicle negative impacts from cloud-based systems ... a loss of communications shall not compromise the integrity of the device, and the device should continue to function if the cloud back-end fails*”.

Safety problems arise also when local connectivity is supported but replay attacks are effective. For example, among

vulnerable devices, we found two different smart plugs, a garage opener, and a smart kettle. The impact of the attack on these kinds of devices is particularly critical in terms of safety. Indeed a smart plug can be used to power a generic device (it may be a medical device or a security camera). Thus a vulnerable plug may directly affect the user safety. Similarly, a replayed “open” command sent to a garage door opener may enable a domestic violation. Also, if a heat-related device (such as a smart kettle) is turned on without any authorization, there might be a risk of overheating, thus potentially leading to a fire. This could pose a threat to the safety of the home occupants. It is also worth noting that unauthorized activation of a generic smart device could lead to unplanned and excessive energy consumption.

**Security and Privacy implications.** The effectiveness of our tool denotes a lack of proper security measures on the IoT devices in the local network. This violates several guidelines of ENISA and NIST (see Section II). Our results clearly highlight that devices rely on the security provided by firewalls and access points, rather than offering security mechanisms themselves (in contrast with GP-TM-43 of ENISA).

While the impact in terms of security is evident (devices can be managed without users’ will), we observe that replay attacks may lead to privacy issues. For example, the activation of a recording function on a smart speaker may be used to record the voice of a user without their explicit consent, opening the door to new potential IoT tech-abuse phenomena [31].

**Mitigation.** Possible mitigations could include the adoption of a mutual authentication procedure. This would ensure that only authorized parties can issue commands to the IoT device. This solution should always be adopted in conjunction with standard security protocols. Indeed, we can find an attempt at an authentication procedure in the Tapo smartplug. Unfortunately, it is not performed via a security protocol, as it suffers from a lack of randomness and key reuse. Hence this device is still vulnerable to replay attacks. We suggest the adoption of WPA3 that prevents traffic interception from a malicious device in the same network also in the case the network password is known. However, it is not effective in the case of a compromised access point. In addition, we point out that WPA2 is still widely deployed, thus the threat we identified is very likely.

**Limitations.** While we have made our best effort to investigate how IoT devices react to replay attacks, as a first attempt at this space, this work has a few limitations.

*Assessing Device Vulnerability.* The agnostic nature of our tool does not allow us to leverage customized vulnerabilities of a target device. As a consequence, when our tool fails to perform a replay attack on the device, it is not a guarantee that the device is not vulnerable. As such, our findings represent a lower bound of such vulnerability, using an approach that can be automated, i.e., automatically detecting the effectiveness of the attack. On the other hand, the fact that out of 20 potentially vulnerable devices, REPLIOT succeeds in performing the attack against 15 devices, confirms the efficacy of our tool.

*Scalability.* While every step of the validation of our approach

is fully automated, including the execution of function trigger scripts, the creation of these scripts is a manual process that needs to be repeated individually for every function tested on each device. One mitigating factor is that devices within the same categories can often reuse existing scripts with minimal modifications. Lastly, our study is limited to the number of popular devices in our testbed, and we do not investigate all IoT devices on the market. However, REPLIoT is designed to be device-agnostic and will easily scale to other devices. Testing more devices and performing longitudinal studies will be a valuable next step in future works; hence we make publicly available our tool and data, to ease the reproducibility of our experiments.

**Non-Observable Functions.** Our approach is designed to operate exclusively with device functions that can be assessed using trigger scripts. Certain functions, such as device maintenance or synchronization tasks, cannot be initiated directly. To test whether the attack works with these functions, one can follow manual steps.

**Ethical Considerations.** In our experiments we do not cause any real threat on the Internet. All experiments are contained within our own testbed. No traffic related to human activity was collected during the experiments. When conducting the experiments, we fully respected the ethical guidelines defined by our affiliated organization, and we received approval. In testing the functionality of REPLIoT, we exclusively test our own devices, eliminating the privacy risk to others.

**Feedback from Vendors.** To this date, only TP-Link, the manufacturer of the Tapo smart plug, has acknowledged the identified issue and taken proactive steps by releasing a new firmware to address the vulnerability. This corrective action extends its positive impact to millions of Tapo devices, underscoring the significance of our work in enhancing the security and resilience of IoT devices.

## VII. RELATED WORK

In this section, we discuss the scientific literature related to replay attacks in the IoT domain. To the best of our knowledge, no proposal in the literature pursues the same goals of this paper. The problem of replay attacks in IoT is acknowledged in the literature [32]. As a consequence, several scientific works [33]–[35] propose solutions to mitigate it. However, no commercial device currently implements them.

Other papers are devoted to the detection of replay attacks against IoT devices [36]–[38]. Similarly, [39]–[41] focus on detecting replay attacks against voice-activated services. However, the primary purpose of these works is to identify whether a device is currently under a replay attack and these solutions are integrated as part of an intrusion detection module, thus proposing mitigation techniques for replay attacks. In contrast, as explained in Section II-C (Non-Goals), our paper addresses the active automated execution of replay attacks with the goal of assessing the vulnerability of devices to replay attacks.

Wara and You [42]’s study is the closest to our work since it shows how replay attacks can be performed on IoT devices supporting ZigBee. However, the tool proposed is not

agnostic and the experimental evaluation is quite limited (the experimental validation is conducted on three devices only).

Other works [17], [43] actively perform replay attacks on smart speakers. However, again, they are not device-agnostic and the replay attack refers to record and replay the voice of users to trigger commands, not testing generic functions of a consumer IoT device.

Finally, our work is closely aligned with a relevant technique known as IoT fuzzing [44]. However, there are some inherent distinctions. Fuzzing is not primarily designed for detecting or preventing replay attacks. Fuzzing is a software testing technique that involves sending intentionally malformed or random data to a target to discover vulnerabilities. As a matter of fact, our tool aims to replay legitimate messages to verify whether possible authentication mechanisms put in place by an IoT device can be bypassed. Indeed, our tool preserves the original messages including potential signatures, authentication tokens, nonces etc., thus it has more chances than a fuzzer to craft messages that will be accepted by IoT devices.

## VIII. CONCLUSION

Due to the proliferation of IoT devices in smart home, the protection of the local network segment is a crucial aspect to take into consideration. In this paper, we focused on replay attacks that can be performed by an adversary with access to the local network to trigger functions on the devices without the user’s will. We developed a tool, called REPLIoT, for automatically testing replay attacks on consumer IoT devices. The tool is designed to be device-agnostic, thus not requiring prior knowledge of the specific devices. We employed this tool to perform a large-scale experiment involving 41 devices spanning different vendors and categories.

Our experiments reveal the existence of several vulnerable devices or not compliant with safety guidelines, thus demonstrating that this threat is real and can potentially harm users’ households.

As future work, we plan to extend our tool with new features. Specifically, we intend to leverage Natural Language Processing (NLP) techniques, enabling us to differentiate the diverse commands associated with devices, facilitating the precise triggering of specific actions.

To support further research, all software and data we produced as part of this work are publicly available at <https://github.com/SafeNetIoT/ReplayAttack>.

## ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our shepherd Urs Hengartner for their constructive feedback. This work is partially funded by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and the EPSRC PETRAS (EP/S035362/1).

## REFERENCES

- [1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
- [2] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of cleaner production*, vol. 140, pp. 1454–1464, 2017.
- [3] M. Q. Aldossari and A. Sidorova, "Consumer acceptance of internet of things (iot): Smart home context," *Journal of Computer Information Systems*, vol. 60, no. 6, pp. 507–517, 2020.
- [4] H. Lee, H. Mun, and Y. Lee, "Comparing response time of home iot devices with or without cloud," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 1–6.
- [5] European Union Agency for Cybersecurity (ENISA). (2021) Baseline security recommendations for iot. Accessed: October 8, 2023. [Online]. Available: <https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot/@/download/fullReport>
- [6] NIST. (2021) Cybersecurity for iot program. Accessed: October 8, 2023. [Online]. Available: <https://www.nist.gov/itl/applied-cybersecurity/nist-cybersecurity-iot-program/consumer-iot-cybersecurity>
- [7] A. Girish, T. Hu, V. Prakash, D. J. Dubois, S. Matic, D. Y. Huang, S. Egelman, J. Reardon, J. Tapiador, D. Choffnes, and N. Vallina-Rodriguez, "In the room where it happens: Characterizing local communication and threats in smart homes," in *Proc. of the 2023 ACM on IMC*. ACM, 2023, p. 437–456.
- [8] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proceedings The Computer Security Foundations Workshop VII*. IEEE, 1994, pp. 187–191.
- [9] M. Singh and D. Pati, "Countermeasures to replay attacks: A review," *IETE Technical Review*, vol. 37, no. 6, pp. 599–614, 2020.
- [10] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 618–623.
- [11] F. Shaikh, E. Bou-Harb, N. Neshenko, A. P. Wright, and N. Ghani, "Internet of malicious things: Correlating active and passive measurements for inferring and characterizing internet-scale unsolicited iot devices," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 170–177, 2018.
- [12] H. Sinanović and S. Mrdovic, "Analysis of mirai malicious software," in *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2017, pp. 1–5.
- [13] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.
- [14] E. Fritsch, I. Shklovski, and R. Douglas-Jones, "Calling for a revolution: An analysis of iot manifestos," in *Proc. of the CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 1–13.
- [15] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices: A multi-dimensional, network-informed measurement approach," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 267–279.
- [16] R. Gurunath, M. Agarwal, A. Nandi, and D. Samanta, "An overview: Security issue in iot network," in *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, 2018, pp. 104–107.
- [17] S.-H. Yoon, M.-S. Koh, J.-H. Park, and H.-J. Yu, "A new replay attack against automatic speaker verification systems," *IEEE Access*, vol. 8, pp. 36 080–36 088, 2020.
- [18] G. Anselmi, A. M. Mandalari, S. Lazzaro, and V. De Angelis, "Copsec: Compliance-oriented iot security and privacy evaluation framework," in *Proc. of the International Conference on MobiCom*, 2023, pp. 1–3.
- [19] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, 2017, pp. 2177–2184.
- [20] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis et al., "Understanding the mirai botnet," in *26th USENIX security symposium*, 2017, pp. 1093–1110.
- [21] M. Capellupo, J. Liranzo, M. Z. A. Bhuiyan, T. Hayajneh, and G. Wang, "Security and attack vector analysis of iot devices," in *SpaCCS 2017 International Workshops*. Springer, 2017, pp. 593–606.
- [22] M. Serror, M. Henze, S. Hack, M. Schuba, and K. Wehrle, "Towards in-network security for smart homes," in *Proc. of the 13th international conference on availability, reliability and security*, 2018, pp. 1–8.
- [23] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local iot devices," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 29–35.
- [24] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home iot devices," *IEEE Internet of Things Journal*, vol. 6, pp. 9042–9053, 2019.
- [25] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.
- [26] Pyshark, "Python wrapper for tshark, allowing python packet parsing using wireshark dissectors," <https://pypi.org/project/pyshark/>, accessed: January 17, 2024.
- [27] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. Choffnes, "Blocking without breaking: Identification and mitigation of non-essential iot traffic," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 369–388, 2021.
- [28] S. Shriram and E. Sivasankar, "Anomaly detection on shuttle data using unsupervised learning techniques," in *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, 2019.
- [29] P. Burlina, N. Joshi, I. Wang et al., "Where's wally now? deep generative and discriminative embeddings for novelty detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 507–11 516.
- [30] Mark\_equiawatt, "New kasa api is now available for all," 2023, <https://community.equiawatt.com/t/new-kasa-api-is-now-available-for-all/507> [Accessed: January 17, 2024].
- [31] S. Parkin, T. Patel, I. Lopez-Neira, and L. Tanczer, "Usability analysis of shared device ecosystem security: Informing support for survivors of iot-facilitated tech-abuse," in *Proceedings of the new security paradigms workshop*, 2019, pp. 1–15.
- [32] Y. Lu and L. Da Xu, "Internet of things (iot) cybersecurity research: A review of current research topics," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2103–2115, 2018.
- [33] Y. Feng, W. Wang, Y. Weng, and H. Zhang, "A replay-attack resistant authentication scheme for the internet of things," in *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, vol. 1. IEEE, 2017, pp. 541–547.
- [34] F. De Rango, G. Potrinio, M. Tropea, and P. Fazio, "Energy-aware dynamic internet of things security system based on elliptic curve cryptography and message queue telemetry transport protocol for mitigating replay attacks," *Pervasive and Mobile Computing*, vol. 6, 2020.
- [35] F. Farha, H. Ning, S. Yang, J. Xu, W. Zhang, and K.-K. R. Choo, "Timestamp scheme to mitigate replay attacks in secure zigbee networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 342–351, 2020.
- [36] A. A. Elsaedy, N. Jagannath, A. G. Sanchis, A. Jamalipour, and K. S. Munasinghe, "Replay attack detection in smart cities using deep learning," *IEEE Access*, vol. 8, pp. 137 825–137 837, 2020.
- [37] A. A. Elsaedy, A. Jamalipour, and K. S. Munasinghe, "A hybrid deep learning approach for replay and ddos attack detection in a smart city," *IEEE Access*, vol. 9, pp. 154 864–154 875, 2021.
- [38] W. Song, H. Jia, M. Wang, Y. Wu, W. Xue, C. T. Chou, J. Hu, and W. Hu, "Pistis: Replay attack and liveness detection for gait-based user authentication system on wearable devices using vibration," *IEEE Internet of Things Journal*, vol. 10, no. 9, pp. 8155–8171, 2023.
- [39] K. M. Malik, A. Javed, H. Malik, and A. Irtaza, "A light-weight replay detection framework for voice controlled iot devices," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 982–996, 2020.
- [40] K. M. Malik, H. Malik, and R. Baumann, "Towards vulnerability analysis of voice-driven interfaces and countermeasures for replay attacks," in *2019 IEEE conference on multimedia information processing and retrieval (MIPR)*. IEEE, 2019, pp. 523–528.
- [41] S. Pradhan, W. Sun, G. Baig, and L. Qiu, "Combating replay attacks against voice assistants," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 3, pp. 1–26, 2019.
- [42] M. S. Wara and Q. Yu, "New replay attacks on zigbee devices for internet-of-things (iot) applications," in *IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2020, pp. 1–6.
- [43] M. Shirvanian, S. Vo, and N. Saxena, "Quantifying the breakability of voice assistants," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2019, pp. 1–11.
- [44] M. Eceiza, J. L. Flores, and M. Iturbe, "Fuzzing the internet of things: A review on the techniques and challenges for efficient vulnerability discovery in embedded systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 390–10 411, 2021.