Securing the Smart Home via a Two-Mode Security Framework *

Devkishen Sisodia[1], Samuel Mergendahl[1], Jun Li[1], and Hasan Cam[2]

[1] University of Oregon,
Eugene OR 97403, USA,
{dsisodia, smergend, lijun}@cs.uoregon.edu
[2] United States Army Research Lab,
Adelphi MD 20783, USA,
{hasan.cam.civ}@mail.mil

Abstract. The growth of the Internet of Things (IoT) is contributing to the rise in cyber attacks on the Internet. Unfortunately, the resource-constrained IoT devices and their networks make many traditional security systems less effective or inapplicable. We present TWINKLE, a framework for smart home environments that considers the unique properties of IoT networks. TWINKLE utilizes a two-mode adaptive security model that allows an IoT device to be in regular mode for most of the time which incurs a low resource consumption rate and only when suspicious behavior is detected, switch to vigilant mode which potentially incurs a higher overhead. We show the efficacy of TWINKLE in two case studies that address two types of attacks: distributed denial-of-service (DDoS) and sinkhole attacks. We examine two existing intrusion detection and prevention systems and transform both into new, improved systems using TWINKLE. Our evaluations show that TWINKLE is not only friendly to resource-constrained devices, but can also successfully detect and prevent the two types of attacks, with a significantly lower overhead and detection latency than the existing systems.

Keywords: Internet of Things, smart home, security, resource consumption

1 Introduction

The Internet of Things (IoT) continues to pervade our lives. In 2016, 6.4 billion devices were connected to the Internet [12]. This number is expected to increase to 30 billion by 2020 [15]. However, as IoT devices are connected by the Internet,

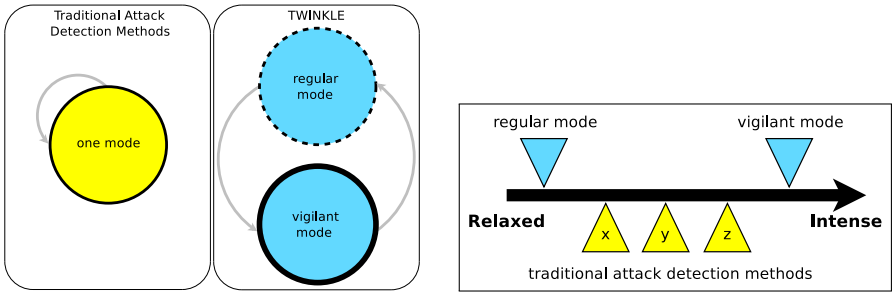
* This project is in part the result of funding provided by the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00204. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security or the US Government.

they also suffer from the same types of attacks that plague traditional Internet-connected machines. In October 2016, for example, the Mirai IoT botnet, which comprised of up to 100,000 infected IoT devices, launched multiple large-scale distributed denial of service (DDoS) attacks [7]. This botnet created a 1.2 terabits per second attack which resulted in the inaccessibility of many popular websites, such as Twitter, Reddit, Netflix, GitHub, and Airbnb.

While IoT devices and traditional machines suffer from the same types of attacks, IoT devices tend to be harder to secure due to some unique properties. IoT devices are often harder to patch and update due to largely non-existent automatic update systems. Also, they tend to have scarce CPU and memory resources, and limited battery capacity, if not plugged into an external power source. IoT devices can have anywhere from a few gigabytes to a few kilobytes of memory. Furthermore, with many different types of IoT devices, IoT networks are far more diverse and heterogeneous than traditional networks. These unique properties, which differentiate IoT devices from traditional machines, hinder the deployment of existing security mechanisms in IoT environments.

Cryptographic protocols and intrusion detection/prevention systems (IDSes/IPSes), developed for the traditional Internet, are designed without the assumption of extremely limited resource and computing power. Even systems that are considered extremely lightweight cannot be installed on memory-constrained devices that have less than 1 MB of available memory [21]. For example, Sehgal et. al. [19] show that many IoT devices struggle to run the cryptographic protocol TLS, a traditional Internet security standard. If a security solution needs to probe devices they protect, most devices in an IoT environment may either lack the power or network bandwidth to respond to every probe, or simply wish to stay dormant most of the time. Sometimes a security solution may impose some minor penalties on benign devices while mitigating an attack (e.g., dropping traffic from devices to mitigate a DDoS attack). These minor penalties, when moved to an IoT environment, can become a significant hindrance to those benign devices.

In this paper, we focus on the smart home environment where security and privacy are especially important, and address the ineffectiveness of traditional security mechanisms in the smart home. We introduce a security framework called TWINKLE that supports individual security applications that handle specific attacks in the smart home. By enabling each security application to run in two distinct modes, TWINKLE not only preserves the salient features of classic security solutions, but also addresses the resource limitations that IoT devices face. Every security application, while plugged into TWINKLE, will be running in regular mode for the most time and incur a minimal amount of resource consumption, but when it detects any suspicious behavior that an attack must display, it can readily switch to vigilant mode and engage in sophisticated routines for a short time window during which to cope with the suspicious behavior with strong competence. By only running the heavyweight routines when needed, TWINKLE saves precious resources over methods that run these routines either continuously or periodically.



(a) Operational state diagrams **(b) Resource consumption**

Fig. 1: Comparison of TWINKLE and traditional defense methods

We further apply the TWINKLE framework to transform two prior attack solutions for the smart home environment. We convert the D-WARD solution [13] that handles DDoS attacks from source networks to D-WARD+; unlike D-WARD, D-WARD+ does not drop packets from benign devices while still effectively keeping the DDoS traffic to an unharmful level. We also convert the SVELTE solution [17] that detects sinkhole attacks to SVELTE+, which does not consume network and power resources unless a suspicious behavior is detected and further adds a routine to remove a sinkhole node once it is detected. Our evaluation further demonstrates that D-WARD+ and SVELTE+ incur much less overhead than D-WARD and SVELTE, respectively, while achieving equal or better efficacy in handling the attacks.

The rest of the paper is organized as follows. In Section 2, we describe the TWINKLE framework, including its two modes and its architectural design. In Section 3, we describe how D-WARD addresses the DDoS attack from the source and how we design D-WARD+ to address the drawbacks of D-WARD in the smart home environment. In Section 4, we describe the sinkhole attack in 6LowPAN networks and the prior SVELTE solution, and present how we convert SVELTE into the SVELTE+ solution running on TWINKLE. We present the evaluation results of both case studies in Section 5, showing that the TWINKLE framework can help reduce the resource consumption in the smart home, compared to D-WARD and SVELTE. In Section 6, we discuss the feasibility of deploying TWINKLE, present possible extensions to TWINKLE, and consider open issues that will be addressed in future work. Lastly, we survey related work in Section 7 and conclude the paper in Section 8.

2 TWINKLE: A Two-Mode Security Framework for the Smart Home

Many security solutions developed for the traditional Internet, if deployed in an IoT environment such as a smart home, would require more computing power, resources, and energy than what IoT devices can provide. We design a two-mode security framework called **TWINKLE**, **TW**o-mode **IN**-home framewor**K**

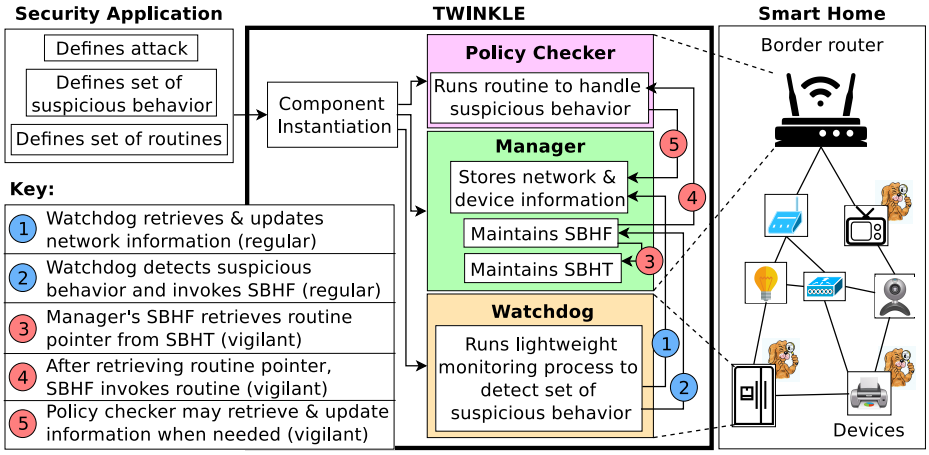


Fig. 2: The basic architecture of TWINKLE

toward **L**ightweight S**E**curity, that will not only preserve the salient features of classic security solutions, but also address the resource limitations that IoT devices face. We describe our design in this section.

2.1 Basic Design with Two Modes

A smart home requires many types of security applications. It may face various malicious attacks such as an eavesdropping attack that can spy on the traffic between the smart home devices, a sinkhole attack that can misdirect traffic of devices to a sinkhole, a wormhole attack that can reroute data from the smart home to an attacker outside, or an attack that compromises devices at the smart home and turns them into nodes of a botnet. Worse, a smart home may also initiate attacks, such as launching a distributed denial-of-service (DDoS) attack or a phishing campaign through compromised devices at home. The TWINKLE framework thus aims to support various security applications for the smart home, where every security application handles a specific type of attack. For every security application, the user can plug it into the framework when needed, or remove it when it is no longer necessary.

The central dilemma facing these security applications is that they must address the inadequacy of computing power and resources available to smart home devices without compromising their efficacy. If a security application runs directly on a smart home device, it may demand resources from a device that are unavailable; otherwise, a security application may still need devices to respond to its requests, sometimes causing a stretch in the resources at those devices.

We therefore design the TWINKLE framework to address this dilemma. It supports any security application to operate in two distinct modes: *regular mode* for most of the time which has a low resource consumption rate and *vigilant mode* that potentially incurs a high overhead but is infrequent. In regular mode, a security application invokes functions from TWINKLE to detect suspicious behavior that an attack, if occurring, must display, where those functions must also

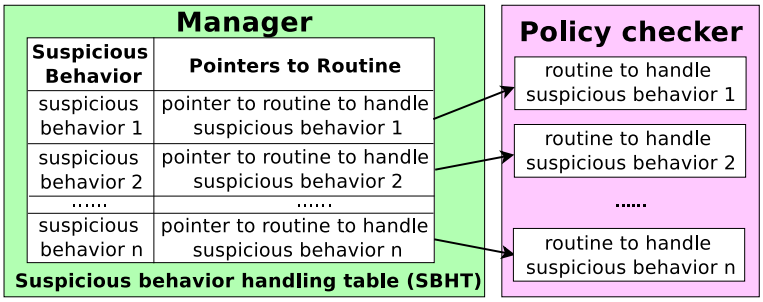


Fig. 3: Diagram of the suspicious behavior handling table (SBHT)

be lightweight. Once it detects suspicious behavior (i.e., an attack *may be* occurring), the security application will enter vigilant mode to inspect closely whether an attack is *indeed* occurring and if so conduct other security operations such as sending an alert of the attack, mitigating the attack, or recovering from the attack. After the attack is handled or the smart home is no longer under this attack, the security application goes back to regular mode. As shown in Figure 1a, this two-mode design differs from many traditional attack detection methods which either run continuously or periodically in one mode. Regular mode is less resource-consuming than a traditional one mode system, while vigilant mode may be more resource-consuming.

TWINKLE thus supports every security application to switch between these two modes. By staying in regular mode most of the time, the security application will incur a minimal amount of resource overhead. By transitioning into vigilant mode for a short period only when needed, the security application can engage in sophisticated operations, including those that may be resource-consuming, to detect or handle an attack in question. This concept is depicted in Figure 1b, which represents the resource consumption of the two modes as compared to traditional methods.

2.2 Architecture of TWINKLE

As shown in Figure 2, TWINKLE is composed of three main components: *manager*, *policy checker*, and *watchdog*. In general, the manager and policy checker will be running at a central node, such as the border router of a smart home, and the watchdog can be running at every device.

The manager maintains the information of the smart home network, such as the network topology, routing information, or allowed bandwidth of each out bound connection. More importantly, it supports a function to handle suspicious behavior, or the **suspicious behavior handling function (SBHF)**. It maintains a **suspicious behavior handling table (SBHT)**, in which for each suspicious behavior it points to a specific routine for handling that suspicious behavior, as shown in Figure 3.

The policy checker maintains routines for handling suspicious behavior. Such routines are usually heavyweight and should only be running in vigilant mode when invoked on demand.

The watchdog is a lightweight running process that monitors the smart home for suspicious behavior. Multiple watchdogs can also be running at multiple devices. Whenever a watchdog detects a suspicious behavior, it invokes the function above to process the suspicious behavior. As soon as the function begins its execution, the system will enter vigilant mode. Depending on the security application, a watchdog may perform signature-based detection, behavior-based detection, or a combination of both. In fact, some security applications may not require the watchdog to run on devices in the network. In these cases, off-the-shelf intrusion detection systems (IDSes) can be utilized as the basis for the watchdog that is installed on the border router.

When the TWINKLE framework supports a security application, it will instantiate the manager, the policy checker, and the watchdog according to the security application. The security application must define the attack it targets and the suspicious behavior that its watchdog should monitor. Furthermore, it needs to develop routines to handle each suspicious behavior, plug these routines into the policy checker, and populate the suspicious behavior handling table with every suspicious behavior that the security application is concerned about and the routine that handles the suspicious behavior. Additionally, the security application needs to provide the manager with the necessary information so that the suspicious behavior handling routines can refer to as a basis for their operations.

TWINKLE also provides a dynamic mechanism for a security application to install its watchdog at any device needed. Unlike the manager or policy checker which can run at the central node, depending on the security application in question, the watchdog may need to run on arbitrary devices in the smart home. To cater to this need, TWINKLE deploys a lightweight process called **elf** at each device that may be a candidate for running a watchdog of a security application. When the TWINKLE framework deploys a new security application and needs to run the watchdog code of the application at a device, TWINKLE can communicate with the elf on the device to ship, install, and eventually run the watchdog code on the device. While the watchdog is lightweight, especially compared to traditional methods, installing it on extremely resource-constrained devices may not be possible. This situation is discussed in more detail in Section 6.

2.3 Jamming Attack Scenario

As stated previously, the TWINKLE framework can support various security applications which can be used to handle different types of attacks. In Figure 4, TWINKLE is being used to handle a jamming attack where the link between devices A and B is being jammed by an unknown attacker. In the first two steps (box 1 and box 2), TWINKLE is running in regular mode. In the first step, the watchdog detects suspicious behavior defined by the security application. In this case, device B is not receiving traffic from device A, which is abnormal. Therefore, the watchdog notifies the manager of the suspicious behavior by invoking the manager's SBHF. As the function begins its execution, the manager switches the security application to vigilant mode (box 3). The manager's SBHT

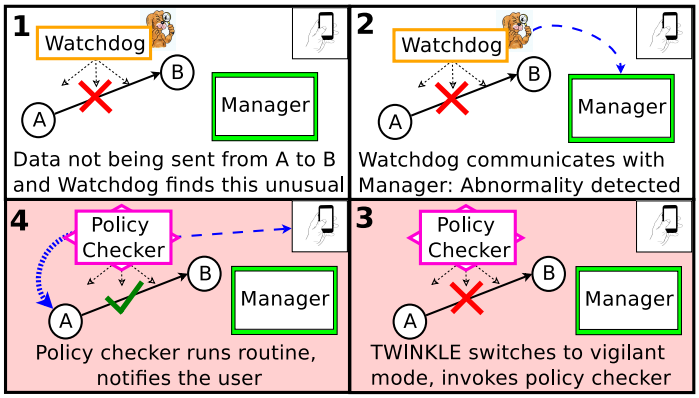


Fig. 4: Jamming Attack

will match the suspicious behavior detected with the routine to handle that suspicious behavior, and the manager will then invoke the policy checker to run that routine. By running the routine, the policy checker will first command A to change its frequency to possibly alleviate the jammed link. The policy checker will also notify the user of the jamming attack (box 4). After mitigating the attack, the policy checker would notify the manager, which returns the security application to regular mode.

3 DDoS Attack Detection By Transforming D-WARD

In this case study, we transform D-WARD, a classic security system for detecting and mitigating DDoS attacks at the source-end of the DDoS traffic, into D-WARD+, a new DDoS defense solution as a security application on TWINKLE.

3.1 DDoS Attacks with IoT Devices

In a DDoS attack, an attacker sends a victim, such as a web server, an overwhelming amount of traffic to make it unavailable. The attacker usually employs a botnet, or a network of compromised devices, to send the traffic. Due to their abundance and the ease to be compromised, IoT devices are easy targets to be recruited by a botnet. As shown in the Mirai attack [7], recent DDoS attacks have been launched from compromised IoT devices and networks.

3.2 Prior Art: D-WARD Against DDoS Attacks

A DDoS defense system placed near the victim may struggle with high volume attacks, but because links closer to the attack sources are less likely to be overwhelmed, filtering attack traffic becomes more feasible for source-end defense systems. One source-end solution example is D-WARD [13]. Deployed at the border router of a policed network, D-WARD consists of an observation module, a rate-limiting module, and a traffic-policing module. The observation module

classifies each aggregated flow, or **agflow**, from all devices in the policed network to an entity outside, **receiver**, as good, suspicious, or attack, based on the ratio of sent packets to received packets of each agflow. Also, each agflow consists of multiple connections where each connection is the traffic from a specific device to the receiver, and for each attack agflow, D-WARD classifies each individual connection as good, transient, or bad, also based on the ratio of sent packets to received packets of the connection. The rate-limiting module applies to each bad and transient connection in an agflow, and it cuts the allowed sending rate of each of these connections to a fraction, f_{dec}, of its current amount. If the device complies with the rate-limit, the rate-limit is increased linearly and eventually removed. The traffic-policing module drops all traffic that surpasses the rate-limit.

While D-WARD is primarily designed for DDoS attacks launched from traditional end-hosts on the Internet, when deployed in a smart home environment, it could hurt benign devices if their connections are labeled as transient connections since their traffic, if over the rate limit, will then be dropped. While a traditional benign end-host can recover from the accidental loss of their packets, in a IoT environment such as a smart home, a benign device could instead suffer significantly from such a loss, due to unnecessary retransmissions of lost packets and increased latency, as shown in subsection 5.1.

3.3 D-WARD+: A Two-Mode Approach Against DDoS Attacks

We therefore transform D-WARD to D-WARD+ that runs on TWINKLE. To overcome the aforementioned drawback of D-WARD, when detecting a DDoS attack from a policed network, D-WARD+ leverages the *fast retransmit* mechanism in TCP congestion control to reduce the sending rate of transient connections, rather than literally dropping their packets as done in D-WARD. Since these connections could be from benign devices, doing so will not cause their packets to be dropped, while still lowering the amount of DDoS traffic departing from the network.

The manager, watchdog, and policy checker of D-WARD+, all running at the border router, are designed as follows. The manager keeps track of the rate-limit of every connection in every attack agflow. The policy checker consists of an agflow monitoring routine. The watchdog monitors the suspicious behavior of each agflow and has the agflow monitoring routine invoked if it detects an attack agflow.

As a security application of TWINKLE, D-WARD+ handles DDoS attacks by switching between the two modes. Beginning with regular mode, if the watchdog of D-WARD+ detects an attack agflow, it will invoke the manager's function for handling suspicious behavior, including passing the handler a suspicious behavior description block (SDB). The SDB will include the identifier of the attack agflow and other meta-data of the agflow. D-WARD+ then executes this function and enters vigilant mode. In doing so, based on the SDB, the function will determine to invoke the agflow monitoring routine to handle the agflow in question. The routine will then monitor each transient connection of the attack agflow; it will

send three duplicate TCP acknowledgments to the device of the connection, which, by following the TCP congestion control design, will reduce its congestion window by half, thus halving its sending rate and mitigating the ongoing DDoS attack. Here, we call the three duplicate TCP acknowledgments a *signal*. In case the device ignores the signal and continues to send its traffic at the original rate, the routine will detect it and label the connection as a bad connection. (Note that if a DDoS device follows the signal in the same way as a benign device, it lowers its sending rate and effectively mitigates the DDoS attack.) Furthermore, if the traffic volume of the connection is still above certain threshold after sending a signal, the routine can send another signal and observe the volume change of the connection, and it can repeat this procedure until the connection is no longer overwhelming its receiver.

Based on the two-mode design above, D-WARD+ is more suitable to a smart home environment than D-WARD. By not literally dropping packets as in D-WARD, D-WARD+ instead informs devices to transmit more slowly. Doing so avoids retransmissions of packets from benign devices, thus lowering network overhead and power consumption.

4 Sinkhole Attack Detection By Transforming SVELTE

In this case study, we transform SVELTE, an IDS for detecting sinkhole attacks in 6LoWPAN networks, into a more resource-efficient security application on TWINKLE.

4.1 Sinkhole Attack in 6LoWPAN Networks

6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) is a wireless technology that combines IPv6 and Low-power Wireless Personal Area Networks (LoWPAN) to enable low-powered devices to communicate using an Internet protocol. A 6LoWPAN network uses RPL (Routing Protocol over Low Powered and Lossy Networks) as its routing protocol [8]. For each destination in a 6LoWPAN network to reach, RPL creates a graph called *Destination Oriented Directed Acyclic Graph* (DODAG) where every node is a device in the network and the destination is the root. Each node in a DODAG has a set of parents, including a preferred parent, where every parent is a potential next hop to reach the root. Moreover, every node in a DODAG has a *rank* to represent the distance between the device and the root (the distance can be calculated in a number of ways, the simplest being hop-count).

Each device periodically sends out a *DODAG Information Object* (DIO) message to advertise its rank. An entering device, upon the receipt of DIO messages from its neighboring devices, will create its set of parents, choose the preferred parent, and calculate its own rank (which is greater than the rank of each of its parents).

The 6LoWPAN network is subject to the sinkhole attack. In such an attack, a compromised device announces a short path toward a destination node to attract

traffic from other nodes to the destination, therefore intercepting or dropping the traffic and creating a sinkhole. A sinkhole attack via RPL can happen when a device sends to its neighbors a DIO message to lie that the device has a low rank. It has been shown that RPL's self-healing and repair mechanisms are not resilient against the sinkhole attack [22].

4.2 Prior Art: SVELTE Against the Sinkhole Attack in 6LoWPAN

SVELTE detects sinkhole attacks in 6LoWPAN networks that occur through RPL rank manipulation. It has three main modules running on the border router (6BR) of a 6LoWPAN network: 6LoWPAN Mapper (6Mapper) that gathers information about the network and determine the DODAG rooted at 6BR, an intrusion detection module that checks the rank inconsistency in data obtained by 6Mapper to detect sinkhole attacks, and a distributed mini-firewall that filters unwanted traffic before it enters the network. 6Mapper sends *probing* messages to nodes in the entire network at regular intervals (e.g., 2 minutes). Each node then sends a *response* message to 6Mapper, which includes its node ID, node rank, parent ID, and all of its neighbors' IDs and ranks.

Unfortunately, SVELTE's probing mechanism can increase the network overhead, device power consumption, and the latency of detecting sinkhole attacks. Every probe from the 6BR will increase the network overhead. Every response from a device will consume more power. Worst of all, SVELTE has a dilemma in choosing the probing interval: a short interval will lead to a low latency in detecting sinkhole attacks, but a large overhead due to frequent probing and responding; a long interval will result in a low overhead, but a high latency in detecting sinkhole attacks.

4.3 SVELTE+: A Two-Mode Approach Against Sinkhole Attacks

To be more resource-efficient, we transform SVELTE to SVELTE+ that runs on TWINKLE. The essential difference between SVELTE+ and SVELTE is that the 6Mapper in SVELTE+ will not probe the entire network periodically and correspondingly, the intrusion detection component will not run periodically, either.

When SVELTE+ is plugged into TWINKLE, its manager, watchdog, and policy checker are as follows. The manager will consist of the 6Mapper module from SVELTE which runs on the central node and the distributed mini-firewall which may run on devices. The policy checker, which also runs on the central node, will include two suspicious behavior handling routines: (1) a sinkhole detection routine (i.e., the intrusion detection module from SVELTE) that inspects the ranks of nodes in the DODAG graph to determine if a sinkhole attack is occurring; and (2) a sinkhole mitigation routine that SVELTE+ newly introduced to mitigate a detected sinkhole attack. Finally, for each device that originally runs a 6Mapper client, we instead equip it with a SVELTE+ watchdog through TWINKLE; it monitors the RPL ranks of its neighbors and alerts the manager of a suspicious behavior when it receives a new rank advertisement; the manager

in turn determines how to handle the suspicious behavior, such as invoking the sinkhole detection routine.

SVELTE+ detects sinkhole attacks by switching between the two modes. It begins in regular mode. Each time a node advertises a new rank, the watchdogs that are within the range of the advertisement will treat the node as a suspect and detect a suspicious behavior. Each watchdog then invokes the manager's function for handling suspicious behavior, including passing to the function a suspicious behavior description block (SDB). The SDB will include the rank of the suspect and the rank of the watchdog itself. More importantly, as soon as the function begins its execution, SVELTE+ enters vigilant mode, allowing it to invoke the corresponding routine to handle the suspicious behavior. Based on the SDB, the function will inspect the behavior and further decides to invoke the sinkhole detection routine inside the policy checker to handle the behavior. The sinkhole detection routine first queries the 6Mapper in the manager for an up-to-date DODAG; then, if the watchdog is a parent (child) of the suspect and its rank is lower (greater) than the rank of the suspect as expected, the routine then has verified the consistency between this watchdog and the suspect. If it has verified the rank consistency with all of the parents and children (or a threshold number of each) of the suspect, it will treat the suspect as a benign node and invoke the 6Mapper to add the node to the DODAG, or simply update its rank if it is already in the DODAG. In case the sinkhole detection routine cannot establish the rank consistency between the suspect and its parents and children, it will detect a sinkhole attack, label the suspect as a sinkhole attacker, and further invoke the sinkhole mitigation routine as described below. The sinkhole detection routine then finishes its execution, followed by the function for handling suspicious behavior, and SVELTE+ returns to regular mode.

The sinkhole mitigation routine's main purpose is to remove a sinkhole node from not only the DODAG, but also the records of any device. Specifically, every parent of the attacker will remove it as their child. Every child of the attacker will remove it as its parent; it may also add a new parent as well as choose a new preferred parent. As a result, the attacker is isolated and can no longer successfully reach any other node.

SVELTE+ outperforms SVELTE in multiple ways. SVELTE+ can reduce the latency in detecting sinkhole attacks to a negligible amount because the watchdog immediately invokes the suspicious behavior handler whenever a new rank is advertised, without having to wait for the next probing interval, as in SVELTE. SVELTE+ also decreases the network overhead and device power consumption as compared to SVELTE; SVELTE+ may incur more overhead in the beginning as nodes join the network, but as the network stabilizes, the amount of times SVELTE+ switches to vigilant mode will be low. An exception here is that a malicious node may frequently advertise a new, legitimate rank, causing SVELTE+ to repeatedly process the suspicious behavior; SVELTE+ sets up an upper bound at which a benign node would advertise a new rank and labels a node as malicious if it advertises a new rank too frequently (it can further remove the node using the sinkhole mitigation routine).

5 Evaluation

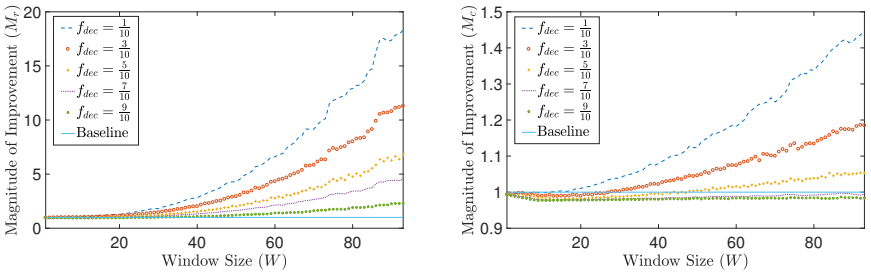
We evaluated TWINKLE's two-mode design by showing how D-WARD+ outperformed D-WARD in source-end DDoS defense and how SVELTE+ outperformed SVELTE in sinkhole attack detection. The metrics we focused on were retransmissions and connection duration for the DDoS case study and network overhead and detection latency for the sinkhole case study. For the DDoS case study, we additionally compared the effects of D-WARD+ and D-WARD on a simple TCP flooding attack versus a smart TCP flooding attack. Note that we did not compare D-WARD+ to D-WARD, and SVELTE+ to SVELTE, in terms of detection accuracy because D-WARD+ and SVELTE+ use the same detection modules as D-WARD and SVELTE, respectively.

We implemented D-WARD+, D-WARD, SVELTE+, and SVELTE in Java on a 2015 Dell XPS with a 2.2 GHz Intel Core i5 processor and 8GB of RAM. Specifically, for the evaluation of D-WARD+ and D-WARD, we constructed a Bluetooth Personal Area Network (PAN) in which a client device transfers 2.5 MB of data to the server through a router on which D-WARD+ and D-WARD are implemented. For the router, we used a 2015 Dell XPS with the same specifications as mentioned previously. In addition to behaving normally, the client device was able to perform simple and smart TCP flooding attacks. Both the client and server utilized TCP New Reno for congestion control. Additionally, for the evaluation of SVELTE+ and SVELTE, we randomly generated mesh IoT network topologies of varying size, which is explained in more detail in subsection 5.2.

5.1 D-WARD+ vs. D-WARD

The main difference between D-WARD+ and D-WARD is that D-WARD+ utilizes the fast retransmit mechanism instead of dropping packets from transient connections. The fast retransmit mechanism allows D-WARD+ to throttle DDoS traffic that leaves the source network it polices and avoid resource penalties on benign traffic. In this section, we analyzed the attainability of these goals in a smart-home network that utilizes D-WARD+. Specifically, we analyzed the following:

1. the ratio of retransmissions D-WARD requires of a benign transient connection over the amount required by D-WARD+;
2. the difference in connection duration of a benign transient connection under D-WARD compared to that of D-WARD+;
3. the maximum length of time that D-WARD+ allows a malicious transient connection to perform a TCP flooding attack; and
4. the maximum length of time that D-WARD+ allows a malicious transient connection to perform a "smart" TCP flooding attack by following TCP congestion control.



(a) Retransmissions **(b) Connection Duration**

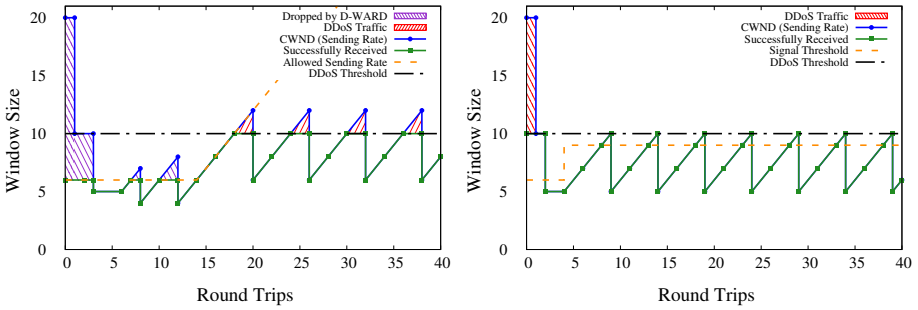
Fig. 5: Comparison of retransmissions and connection duration under D-WARD and D-WARD+

Retransmissions In order to calculate the ratio of retransmissions D-WARD requires of a benign transient connection over the amount required by D-WARD+, we examined the number of retransmissions required of a benign transient connection that attempts to send 2.5 MB of data outside of the policed network under both D-WARD and D-WARD+.

Figure 5a presents the average number of retransmissions that D-WARD+ and D-WARD requires of benign transient connection over two main parameters: the sender's congestion window size, W, at the time D-WARD or D-WARD+ detects an attack agflow, and the pre-set fraction of traffic, f_{dec}, that D-WARD or D-WARD+ allows to leave the source network during a suspected DDoS attack. An f_{dec} of $1/2$ is set as default by Mirkovic et al. [13]. Upon detection of an attack agflow, D-WARD only allows $W * f_{dec}$ segments to the sender each RTT to mitigate any DDoS attacks. Therefore, when the benign transient devices follow TCP congestion control, D-WARD drops $W - W * f_{dec}$ segments every two RTTs. Thus, as W increases or f_{dec} decreases, D-WARD drops more segments which causes more retransmissions. With a large window size and a strict pre-set fraction of allowed traffic, D-WARD may require more than 15 times the number of retransmissions than D-WARD+. Even when the window size is less than 40 and the pre-set fraction of allowed traffic is greater than 0.5, D-WARD still requires more retransmissions than D-WARD+, but to a lesser degree.

Connection Duration We further compared how long a benign transient connection may last under D-WARD and D-WARD+. Clearly, when transmitting the same amount of data, a shorter duration is desired. We examined the duration of a benign transient connection that attempts to send 2.5 MB of data outside of the policed network at a maximum bandwidth of 250 Kb/s under both D-WARD and D-WARD+.

Figure 5b shows the average difference in connection duration between D-WARD and D-WARD+ over the two main parameters W and f_{dec}. When f_{dec} is set low, D-WARD may punish a transient connection too heavily which leads to long connection durations. However, in cases where f_{dec} is set high and W is



(a) D-WARD **(b) D-WARD+**

Fig. 6: Behavior of a smart attacker under D-WARD and D-WARD+

large, a transient connection's duration under D-WARD can be slightly faster (at most 3 seconds) than if it were under D-WARD+.

Simple TCP Flooding Attack A "simple" TCP flooding attack is one in which the attacker ignores TCP congestion and flow control. We formulate the maximum length of time that D-WARD+ allows a malicious transient device to perform a simple TCP flooding attack as follows. Upon detection of an attack agflow, D-WARD+ provides all transient connections belonging to the attack agflow a window of D seconds to prove the benevolence of their traffic. If a transient device performs a simple TCP flooding attack, it will not follow TCP congestion control which D-WARD+ notices within two RTTs. At this point, D-WARD+ now has high confidence the transient connection is malicious, and begins to drop this connection's traffic, thus ending the DDoS attack.

Smart TCP Flooding Attack A "smart" TCP flooding attack is one in which the attacker follows TCP congestion control. We formulate the maximum length of time that D-WARD+ allows a malicious device to perform a "smart" TCP flooding attack by following TCP congestion control. Upon detection of an attack agflow, D-WARD+ sends s signals every RTT for the next D seconds to each transient connection belonging to the attack agflow so that after D seconds, each transient connection's congestion window will be below $RECW$.

Figure 6 compares how D-WARD and D-WARD+ handle a smart attacker in TCP Reno. This figure shows two transient connections that belong to an attack agflow, which was detected at 0 RTT. For simplicity, the victim has a constant $RECW$ of 10. Any traffic that surpasses the $RECW$ of 10 is considered DDoS traffic. Lastly, the circle-dotted (blue) line represents the amount of traffic sent by the attacker and the square-dotted (green) line represents the amount of traffic that is successfully received by the victim.

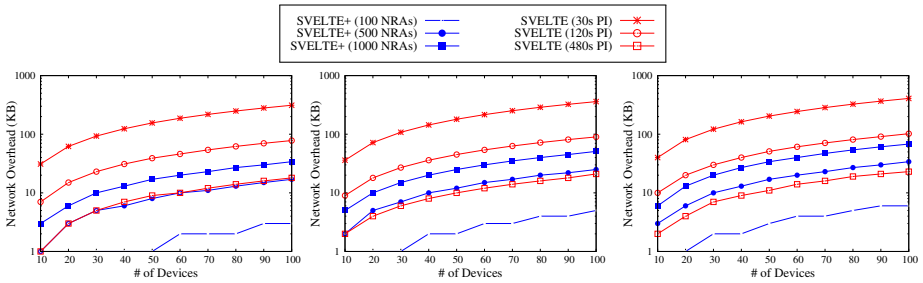
In 6a, D-WARD drops all traffic that surpasses the allowed window size of 6, which in this case means D-WARD set f_{dec} to 0.3. D-WARD initially throttles the smart attacker which forces the attacker to send at the allowed rate (from

0 RTT to 14 RTT), but once the congestion window settles below this initial allowed rate, and because the smart attacker follows TCP congestion control, D-WARD continues to linearly increase the allowed amount even past an amount the receiver can manage (which can be seen at 20 RTT, 25 RTT, 32 RTT, and 37 RTT). This provides the smart attacker an opportunity to successfully send DDoS traffic in the future.

In 6b, D-WARD+ allows all traffic to be sent to the victim, but sends signals to the smart attacker to force it to send less than or equal to the allowed rate. After 4 RTTs, D-WARD+ will have gained enough information about the victim's *RECW* and increase the signal threshold (the threshold at which 3 duplicate ACK packets will be sent to the attacker) to just below *RECW*. While D-WARD+ allows DDoS traffic for a short period initially (0 RTT to 1 RTT), because the smart attacker follows TCP congestion control, D-WARD+ can continue to preemptively restrict the smart attacker's congestion window preventing any further DDoS attacks.

5.2 SVELTE+ vs. SVELTE

The main difference between SVELTE+ and SVELTE is that SVELTE+ utilizes on-demand probing while SVELTE utilizes periodic probing. In this section, we explored how this difference affected network overhead and detection latency for both security applications.



(a) Average degree of 2 **(b) Average degree of 3** **(c) Average degree of 4**

Fig. 7: Difference in network overhead between SVELTE and SVELTE+ based on probing intervals (PI) and number of new rank advertisements per device (NRA/D)

Network Overhead This metric measures the number of extra bytes that are sent using SVELTE as compared to SVELTE+. Sending traffic requires power consumption and therefore, a security system that sends less extra traffic by power-constrained devices is highly desired.

Figure 7 depicts the difference between SVELTE and SVELTE+ in network overhead. We compare SVELTE with three different probing intervals (30, 120,

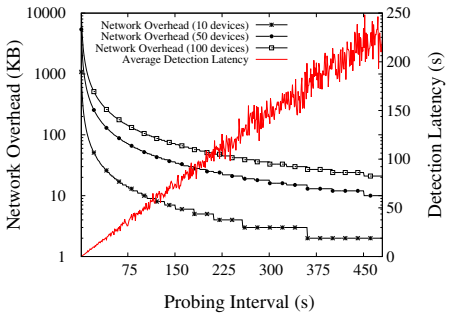


Fig. 8: The effect of probing interval on network overhead and detection latency for SVELTE

and 480 seconds) to SVELTE+ with three different new rank advertisement frequencies (1, 5, and 10 new rank advertisements per device). SVELTE+ incurs less network overhead than SVELTE with probing intervals of 120s and 30s. At an average degree of 3 and 4, SVELTE+ with 5 and 10 new rank advertisements per device incur more network overhead than SVELTE with a probing interval of 480s. However, if a network contains 10 devices and each device advertises a new rank 10 times in a one-hour period, there would be more than 1 new rank advertised each minute. This would be highly unlikely in a stable and stationary environment, such as a smart home. Also, note that while a 480s probing interval for SVELTE incurs a relatively small amount of overhead, the detection latency will be relatively high.

Detection Latency The metric measures the difference in the amount of time it takes SVELTE to detect an attack as compared to SVELTE+. When considering detection latency in our simulations, we did not take into account the negligible round-trip time (RTT) between the border router and each device or processing time at the border router.

Figure 8 illustrates the relationship between network overhead and detection latency for SVELTE. As the probing interval increases, network overhead decreases exponentially and detection latency increases linearly. Unlike SVELTE+, SVELTE must strike a balance between network overhead and detection latency.

The detection latency with SVELTE+ instead is negligible since it immediately responds to a sinkhole attack on demand. It incurs only some communication delay and processing time for the watchdog to report the manipulation of rank by an attacker and for the policy checker to process the report. Here, the on-demand probing method employed by SVELTE+ is a clear advantage over the periodic probing of SVELTE.

6 Discussion

One factor in the feasibility of deploying TWINKLE is the potential difficulty of installing components on a smart home's border router. We assume that the

border router has enough resources to run TWINKLE's manager and policy checker components. While this may be a safe assumption to make for many commercial home routers, we have yet to evaluate this claim. Also, the feasibility of running TWINKLE on routers is highly dependent on the security application. We can estimate the memory consumption of D-WARD+ and SVELTE+ from evaluation done on D-WARD and SVELTE. For example, D-WARD consumes at most 37.581 KB of RAM at the router in a smart home with 100 outbound connections, while SVELTE consumes at most 4.724 KB of RAM (49.924 KB of ROM) at the router in a smart home containing 16 devices.

Another issue is the feasibility of running watchdog code on devices in the smart home. We assume that a watchdog device can run an elf process which can be used to transfer watchdog code onto the device. However, some devices, such as legacy and extremely resource-constrained devices, may not have the ability to install even lightweight processes like elf. Therefore, in some cases, additional devices need to be added to the network to act as watchdogs. Furthermore, watchdogs are required to have enough resources to run the lightweight algorithms of regular mode. Again, the feasibility of running these algorithms is dependent on the security application. SVELTE consumes at most 0.350 KB of RAM (1.414 KB of ROM) at each device while D-WARD does not run any code on the devices.

TWINKLE is a security framework that addresses the resource limitations of IoT devices by giving security applications the ability to run in two modes. However, TWINKLE can be extended to include more than two modes of operation. Certain security applications need to invoke a wide range of functions in order to mitigate an attack. Allowing for multiple levels of granularity in which applications can invoke functions of varying resource-consumption intensity may further reduce resource consumption.

In future work, we plan on studying, and eventually addressing, the aforementioned issues. Specifically, we will evaluate multiple different security applications on TWINKLE and implement TWINKLE on a real IoT testbed to present a more comprehensive study on the feasibility of deploying components on border routers and devices.

7 Background & Related Work

We organize the related work into three sections. The first is on work that provides analysis on smart home security and goals for securing smart home environments. The second is on work that introduces security frameworks and systems targeted towards IoT environments. The last section is on work that motivates different components of our two-mode framework.

7.1 Smart Home Security Analysis

Recent work, [5], [6], and [16], explores the current state of smart home security and provide suggestions on improvements in this environment. Denning et al. [5]

group security and privacy goals into three categories: device goals (device privacy, device availability, command authenticity, and execution integrity), data goals (data privacy, data integrity, and data availability), and environment goals (environment integrity, activity pattern privacy, sensed data privacy, sensor validity, and sensor availability). Notra et al. [16] report vulnerabilities in various household devices, such as the Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm. The main contribution of [6] is the discovery of security-critical design flaws in the SmartThings capability model and event subsystem. These papers give insight into the vulnerabilities and open issues that need to be addressed by smart home security frameworks and systems. Of the three papers, only [16] provides a security solution. However, this solution only provides protection via access control rules deployed at the gateway router to prevent unauthorized in-bound and out-bound traffic. Our framework, not only monitors traffic leaving and entering the network, but also monitors device to device communication from within the network. This allows our framework to potentially detect and prevent attacks that cannot be detected or prevented solely at the gateway router.

7.2 Frameworks and Systems

In this section, we survey select papers which introduce security frameworks for IoT environments [2], [3], [10], and [20]. In [3], the authors present a security framework based on the Architecture Reference Model (ARM) of the IoT-A EU project. The work in [2], uses game theory and context-aware techniques to create a risk-based adaptive security framework for IoT in an eHealth environment. Both [3] and [2] are proof-of-concept papers that do not provide evidence that the presented frameworks are viable in resource constrained environments. Similar to our framework, the frameworks presented in [20] and [10] are both targeted towards smart home environments. Also, the authors of [20] present a modular security manager which is similar to the Manager component in our framework. However, like [3] and [2], the authors of both papers do not address the limitations of IoT devices nor provide evaluation results for the resource costs of deploying their solutions. In contrast, our framework's primary focus is to reduce resource consumption while maintaining a secure environment. Furthermore, we show that our framework can reduce resource consumption through the evaluation of two concrete case studies.

7.3 Motivation for Certain Components and Policies

Papers [4], [9], [11], and [14] motivate the need of certain components in a security framework for the smart home environment. Instead of introducing new security countermeasures, the authors of [9] attempt to strengthen security for smart home networks by making it easier for non-expert home owners to set up secure networks and intuitively manage trust and access to their devices. The research in [14] attempts to provide adequate mechanisms to control the flow of data and enforce policies based on users' preferences. In [4], the authors

utilize special nodes that monitor traffic within the network to detect certain routing attacks. The work in [9], [14], and [4] show the need of user interaction, adjustable policies set by users, and dedicated watchdog nodes for inspection of in-network communication, respectively. Also, the work in [11] provides motivation for allowing security policies, such as using efficient authentication and key agreement methods. The substantial research in the area of security in wireless sensor networks (WSNs), such as the work presented in [1] and [18], can be leveraged to improve TWINKLE. In summary, work presented in this section can supplement and extend our framework.

8 Conclusion

The staggering growth of the Internet of Things (IoT) brings serious security concerns. However, due to the constrained resources of IoT devices and their networks, many traditional attack detection methods become less effective or inapplicable in an IoT environment. Using the smart home as the battleground, this paper proposes a security framework called TWINKLE that endeavors to address a fundamental dilemma facing any security solution for IoT: the solution has to consume as little resources as possible while still aspiring to achieve the same level of performance as if the resources needed are abundant. It introduces a two-mode design to enable security applications plugged into the framework to handle their targeted attacks in an on-demand fashion. Every security application can simply run lightweight operations in regular mode most of the time, and only invoke heavyweight security routines when it needs to cope with suspicious behavior. By applying TWINKLE to distributed denial-of-service (DDoS) and sinkhole attacks, we can successfully convert prior solutions to more resource-efficient versions, as demonstrated by our evaluations.

References

1. Abduvaliyev, A., Pathan, A.S.K., Zhou, J., Roman, R., Wong, W.C.: On the vital areas of intrusion detection systems in wireless sensor networks. IEEE Communications Surveys & Tutorials 15(3), 1223–1237 (2013)
2. Abie, H., Balasingham, I.: Risk-based adaptive security for smart iot in ehealth. In: Proceedings of the 7th International Conference on Body Area Networks. pp. 269–275. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2012)
3. Bernabe, J.B., Hernandez, J.L., Moreno, M.V., Gomez, A.F.S.: Privacy-preserving security framework for a social-aware internet of things. In: International Conference on Ubiquitous Computing and Ambient Intelligence. pp. 408–415. Springer (2014)
4. Cervantes, C., Poplade, D., Nogueira, M., Santos, A.: Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things. In: IFIP/IEEE International Symposium on Integrated Network Management. pp. 606–611. IEEE (2015)

5. Denning, T., Kohno, T., Levy, H.M.: Computer security and the modern home. ACM Communications 56(1), 94–103 (2013)
6. Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: IEEE Symposium on Security and Privacy. pp. 636–654. IEEE (2016)
7. Hilton, S.: Dyn analysis summary of friday october 21 attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/ (2016)
8. IETF, R.: Routing over low power and lossy networks
9. Kalofonos, D.N., Shakhshir, S.: Intuisec: a framework for intuitive user interaction with smart home security using mobile devices. In: IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications. pp. 1–5. IEEE (2007)
10. Kang, W.M., Moon, S.Y., Park, J.H.: An enhanced security framework for home appliances in smart home. Human-centric Computing and Information Sciences 7(1), 6 (2017)
11. Kumar, P., Braeken, A., Gurtov, A., Iinatti, J., Ha, P.: Anonymous secure framework in connected smart home environments. IEEE Transactions on Information Forensics and Security (2017)
12. van der Meulen, R.: Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. http://www.gartner.com/newsroom/id/3165317 (2015)
13. Mirkovic, J., Reiher, P.: D-ward: a source-end defense against flooding denial-of-service attacks. IEEE transactions on Dependable and Secure Computing 2(3), 216–232 (2005)
14. Neisse, R., Steri, G., Baldini, G.: Enforcement of security policy rules for the internet of things. In: IEEE 10th International Conference on Wireless and Mobile Computing. pp. 165–172. IEEE (2014)
15. Nordrum, A.: Popular internet of things forecast of 50 billion devices by 2020 is outdated. http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated (2016)
16. Notra, S., Siddiqi, M., Gharakheili, H.H., Sivaraman, V., Boreli, R.: An experimental study of security and privacy risks with emerging household appliances. In: IEEE Conference on Communications and Network Security. pp. 79–84. IEEE (2014)
17. Raza, S., Wallgren, L., Voigt, T.: Svelte: Real-time intrusion detection in the internet of things. Ad hoc networks 11(8), 2661–2674 (2013)
18. Roman, R., Zhou, J., Lopez, J.: Applying intrusion detection systems to wireless sensor networks. In: IEEE Consumer Communications & Networking Conference (CCNC 2006) (2006)
19. Sehgal, A., Perelman, V., Kuryla, S., Schonwalder, J.: Management of resource constrained devices in the internet of things. IEEE Communications Magazine 50(12) (2012)
20. Simpson, A.K., Roesner, F., Kohno, T.: Securing vulnerable home iot devices with an in-hub security manager. In: IEEE International Conference on Pervasive Computing and Communications Workshops. pp. 551–556. IEEE (2017)
21. Team, O.P.: Ossec: Open source hids security. https://ossec.github.io/index.html (2010–2017)
22. Wallgren, L., Raza, S., Voigt, T.: Routing attacks and countermeasures in the rpl-based internet of things. International Journal of Distributed Sensor Networks 9(8), 794326 (2013)