DEFENDING AGAINST IOT-ENABLED DDOS ATTACKS AT CRITICAL VANTAGE POINTS ON THE INTERNET

by

DEVKISHEN SISODIA

A DISSERTATION

Presented to the Computer and Information Science
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 2022

DISSERTATION APPROVAL PAGE

Student: Devkishen Sisodia

Title: Defending Against IoT-Enabled DDoS Attacks at Critical Vantage Points on the Internet

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Computer and Information Science by:

Jun Li	Chair
Joe Sventek	Core Member
Lei Jiao	Core Member
Ramón Alvarado	Institutional Representative

and

Krista Chronsiter	Vice Provost for Graduate Studies

Original approval signatures are on file with the University of Oregon Division of Graduate Studies.

Degree awarded June 2022

DISSERTATION ABSTRACT

Devkishen Sisodia

Doctor of Philosophy

Computer and Information Science

June 2022

Title: Defending Against IoT-Enabled DDoS Attacks at Critical Vantage Points on the Internet

The number of Internet of Things (IoT) devices continues to grow every year. Unfortunately, with the rise of IoT devices, the Internet is also witnessing a rise in the number and scale of IoT-enabled distributed denial-of-service (DDoS) attacks. However, there is a lack of network-based solutions targeted directly for IoT networks to address the problem of IoT-enabled DDoS. Unlike most security approaches for IoT which focus on hardening device security through hardware and/or software modification, which in many cases is infeasible, we introduce network-based approaches for addressing IoT-enabled DDoS attacks. We argue that in order to effectively defend the Internet against IoT-enabled DDoS attacks, it is necessary to consider network-wide defense at critical vantage points on the Internet. This dissertation is focused on three inherently connected and complimentary components: (1) preventing IoT devices from being turned into DDoS bots by inspecting traffic towards IoT networks at an upstream ISP/IXP, (2) detecting DDoS traffic leaving an IoT network by inspecting traffic at its gateway, and (3) mitigating attacks as close to the devices in an IoT network originating DDoS traffic. To this end, we present three security solutions to address the three aforementioned components to defend against IoT-enabled DDoS attacks.

This dissertation includes published and unpublished co-authored materials.

CURRICULUM VITAE

NAME OF AUTHOR: Devkishen Sisodia

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
The University of Texas at Arlington, Arlington, TX, USA

DEGREES AWARDED:

Doctor of Philosophy, Computer and Information Science, 2022, University of Oregon
Bachelor of Science, Computer Science, 2015, The University of Texas at Arlington

AREAS OF SPECIAL INTEREST:

Network Security
Network Traffic Analysis
Network Measurement

PROFESSIONAL EXPERIENCE:

Research and Teaching Assistant, University of Oregon, 2015 - 2022
Research and Teaching Assistant, University of Texas at Arlington, 2013 - 2015
Research Assistant, Indiana University–Pudue University Indianapolis, 2014
Chief Android Developer and Founding Member, Geronimobile Studios LLC, 2012 - 2015

GRANTS, AWARDS AND HONORS:

NSF REU Best Research Contribution Award, 2014
Oregon Cyber Security Day Outstanding Poster Prize, 2018

PROFESSIONAL SERVICES:

Student Member, CIS Diversity Committee, University of Oregon, 2022
Volunteer, ACM SIGCOMM 2021 Hackathon: ISP-DDoS, 2021
Reviewer, IEEE/ACM Transactions on Networking (ToN), 2021
Reviewer, IEEE/ACM International Symposium on Quality of Service (IWQoS), 2021
Reviewer, IEEE Internet of Things Journal (IoT-J), 2020
Reviewer, IEEE Transactions on Dependable and Secure Computing (TDSC), 2019
Reviewer, IEEE Journal on Selected Areas in Communications (JSAC), 2019
Demonstration and Poster Chair, Oregon Cyber Security Day, University of Oregon, 2017
Technical Program Committee Chair, Research Experience for Undergraduates Symposium, Indiana University–Pudue University Indianapolis, 2014

PUBLICATIONS:

Jun Li, Devkishen Sisodia, Yebo Feng, Lumin Shi, Mingwei Zhang, Samuel Mergendahl, Christopher Early, Peter Reiher (2022). Toward Adaptive Distributed Filtering of DDoS Traffic. **In Preparation**.

Lumin Shi, Devkishen Sisodia, Jun Li, Mingwei Zhang, Alberto Dainotti, Peter Reiher (2022). The Moving Target Attack: On Quantifying Multi-Wave Victims in DDoS Mitigation. **In Preparation**.

Yebo Feng, Jun Li, Devkishen Sisodia, Peter Reiher (2022). On Explainable and Adaptable Detection of Distributed Denial-of-Service Traffic. *IEEE Transactions on Secure and Dependable Computing (TDSC).* **In Submission**.

Yebo Feng, Jun Li, Devkishen Sisodia (2022). CJ-Sniffer: Measurement and Content-Agnostic Detection of Cryptojacking Traffic. *International Symposium on Research in Attacks, Intrusions and Defenses (RAID).* **In Submission**.

Devkishen Sisodia, Jun Li, Samuel Mergendahl, Hasan Cam (2021). A Two-Mode, Adaptive Security Framework for Smart Home Security Applications. *ACM Transactions on Internet of Things (TIoT).* **In Submission**.

Jun Li, Devkishen Sisodia, Shad Stafford (2021). On the Detection of Smart, Self-Propagating Internet Worms. *IEEE Transactions on Secure and Dependable Computing (TDSC).* **In Submission**.

Jelena Mirkovic, Stephen Hayne, Michalis Kallitsis, Wes Hardaker, John Heidemann, Christos Papadopoulos, Devkishen Sisodia (2021). Cybersecurity Datasets: A Mirage (White Paper). *NSF Workshop on Overcoming Measurement Barriers to Internet Research (WOMBIR).*

Devkishen Sisodia, Jun Li, Lei Jiao (2020). In-Network Filtering of Distributed Denial-of-Service Traffic with Near-Optimal Rule Selection. *ACM ASIA Conference on Computer and Communications Security (ASIACCS).*

Devkishen Sisodia (2020). On the State of Internet of Things Security: Vulnerabilities, Attacks, and Recent Countermeasures (Technical Report AREA-202001-Sisodia). *Computer and Information Science, University of Oregon.*

Lumin Shi, Samuel Mergendahl, Devkishen Sisodia, Jun Li (2020). Bridging Missing Gaps in Evaluating DDoS Research. *USENIX Workshop on Cyber Security Experimentation and Test (CSET).*

Yebo Feng, Jun Li, Devkishen Sisodia (2020). Content-Agnostic Identification of Cryptojacking in Network Traffic (Extended Abstract). *ACM ASIA Conference on Computer and Communications Security (ASIACCS).*

Lumin Shi, Samuel Mergendahl, Devkishen Sisodia, Jun Li (2020). Playing in the Sandbox: A Step Towards Sound DDoS Research Through High-Fidelity Evaluation (Extended Abstract). *Passive and Active Measurement Conference (PAM).*

Mingwei Zhang, Lumin Shi, Devkishen Sisodia, Jun Li, Peter Reiher (2019). On Multi-Point, In-Network Filtering of Distributed Denial-of-Service Traffic. *IFIP/IEEE International Symposium on Integrated Network Management (IM).*

Lumin Shi, Devkishen Sisodia, Mingwei Zhang, Jun Li, Alberto Dainotti, Peter Reiher (2019). The Catch-22 Attack (Extended Abstract). *Annual Computer Security Applications Conference (ACSAC).*

Devkishen Sisodia, Samuel Mergendahl, Jun Li, Hasan Cam (2018). Securing the Smart Home via a Two-Mode Security Framework. *EAI International Conference on Security and Privacy in Communication Networks (SecureCom).*

Samuel Mergendahl, Devkishen Sisodia, Jun Li, Hasan Cam (2018). FR-WARD: Fast Retransmit as a Wary but Ample Response to Distributed Denial-of-Service Attacks from the Internet of Things. *International Conference on Computer Communications and Networks (ICCCN).*

Tianchong Gao, Wei Peng, Devkishen Sisodia, Tanay Kumar Saha, Feng Li, Mohammad Al Hasan (2018). Android Malware Detection via Graphlet Sampling. *IEEE Transactions on Mobile Computing (TMC).*

Samuel Mergendahl, Devkishen Sisodia, Jun Li, Hasan Cam (2017). Source-End DDoS Defense in IoT Environments (Extended Abstract). *Workshop on Internet of Things Security and Privacy (IoT S&P).*

Wei Peng, Tianchong Gao, Devkishen Sisodia, Tanay Kumar Saha, Feng Li, Mohammad Al Hasan (2016). ACTS: Extracting Android App Topological Signature Through Graphlet Sampling. *IEEE Conference on Communications and Network Security (CNS).*

ACKNOWLEDGEMENTS

I sincerely thank my advisor, Jun Li, for guiding me throughout my journey to my Ph.D. and providing me with invaluable lessons on how to be an effective leader. With these lessons, I hope that one day, I can lead my own students with kindness, compassion, and empathy.

I would also like to thank my other committee members, Joe Sventek and Lei Jiao, for their guidance as I worked on my final two milestones, and Ramón Alvarado, for his helpful feedback on how to best structure this thesis.

I am forever grateful to the many friends I made during these seven years in Oregon for helping me get through some of the toughest times in my life. The relationships I formed here will last a lifetime.

To my old friends, I can't thank you enough for not only encouraging me to pursue a Ph.D. far from home, but for motivating me to persevere when I wanted to quit.

Last, but not least...

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

CHAPTER I
INTRODUCTION

The Internet of Things (IoT) is a computer networking paradigm that refers to scenarios where network connectivity and computing capability extends to embedded sensors and everyday devices, allowing them to generate, exchange, consume, and act upon data with minimal to no human intervention. This paradigm is possible due to recent advancements in the miniaturization of electronics, networking capabilities, and computing power. Even in its relative infancy, IoT is already impacting our everyday lives in profound ways, from healthcare to home automation. By the beginning of 2021, there were around 11.3 billion IoT devices connected to the Internet, and this number is expected to increase to more than 27 billion by 2025 [129].

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.

With the rise of IoT devices, the Internet is also witnessing a rise in IoT-enabled distributed denial-of-service (DDoS) attacks [68]. However, the lack of attention paid to the ever-growing problem of IoT-enabled DDoS by the IoT

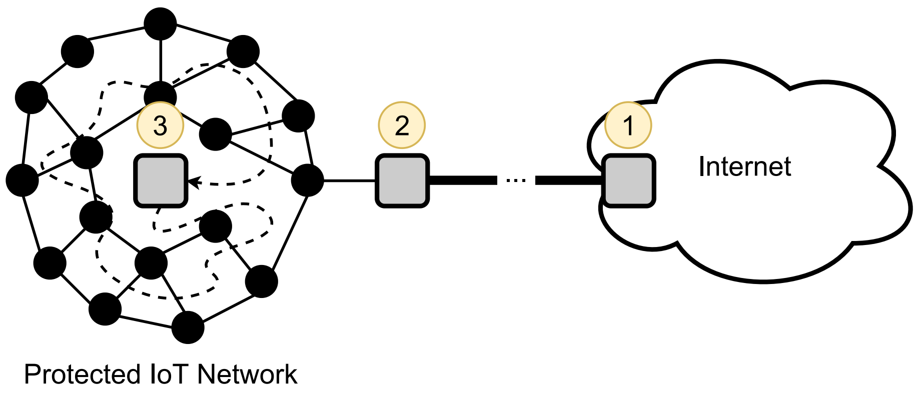


Figure 1. The three vantage points at which we deploy our network-based security solutions to tackle IoT-enabled DDoS attacks. Vantage point 1 is an upstream ISP/IXP to the protected IoT network, vantage point 2 is a gateway of the protected IoT network, and vantage point 3 is inside of the protected IoT network.

security community has lead to a lack of network-based solutions targeted directly for IoT networks to address IoT-enabled DDoS. The solutions that do exist usually require modifications of the IoT devices themselves in order to patch vulnerabilities that allow for devices to be turned into DDoS bots.

The two main issues facing such solutions that require modifications are that they are infeasible and not scalable. With regards to feasibility, many of the devices already connected to the Internet are closed systems, thereby making modifications at any layer almost impossible. Also, the location of IoT devices can also make modifications extremely difficult (e.g., devices implanted in the body). Furthermore, modifying the software or hardware of a device may require a shutdown, which could cause wide-ranging negative effects. With regards to scalability, there are billions of IoT devices currently connected to the Internet [129]; patching each one of them is practically impossible.

1.1 Thesis Statement

Unlike most security approaches for IoT which focus on hardening device security through hardware and/or software modification, we introduce networking-based approaches for addressing IoT-enabled DDoS attacks. We argue that in order to comprehensively defend IoT networks, defense must occur at several important vantage points on the Internet. Thus, the thesis statement is as follows. **In order to effectively defend the Internet against IoT-enabled DDoS attacks, it is necessary to consider network-wide defense at critical vantage points on the Internet. This thesis is focused on three inherently connected and complimentary components: (1) preventing IoT devices from being turned into DDoS bots by inspecting traffic towards IoT networks at an upstream Internet service provider (ISP) or Internet exchange point (IXP), (2) detecting DDoS traffic leaving an IoT network by inspecting traffic at its gateway, and (3) mitigating attacks as close to the devices in an IoT network originating DDoS traffic.** Figure 1, shows the three vantage points at which we deploy our network-based security solutions to tackle IoT-enabled DDoS attacks. We elaborate on each component below.

1.2 Preventing IoT Devices From Being Turned Into DDoS Bots by Inspecting Traffic Towards IoT Networks at an Upstream ISP/IXP

Self-propagating worms, which ultimately can lead to devastating DDoS attacks, can infect millions of computers on the Internet in just several minutes. As witnessed by the recent Mirai [12] worm, worm attacks are real, destructive, and continue to persist. Although many worm detectors exist, most that we studied suffer from three drawbacks: none systematically consider countermeasures from worm authors, potentially causing low effectiveness against evasive worms; all focus on outbound worms leaving a network, leaving their efficacy against inbound worms entering a network unanswered; and many require bi-directional traffic to detect worms, making their placement on the Internet inflexible. We therefore revisit worm detection while avoiding the aforementioned drawbacks of existing work. We describe our design of SWORD, a new worm detector that focuses on the fundamental behavior of worms. It includes two complementary modules to monitor connections from and to a protected network, with one module monitoring burst durations and the other ensuring quiescent periods. While SWORD can be deployed at an IoT network's gateway router, we choose to deploy it at an IoT network's upstream ISP/IXP, which is a more challenging placement due to the likelihood of SWORD only observing partial traffic. However, the main advantage for deploying SWORD upstream is that it is more cost efficient compared to running SWORD at each network downstream. Furthermore, if SWORD is effective at detecting the presence of worms upstream, it will be equally effective, if not more effective, if deployed at an IoT network's gateway router. Via extensive experiments using a real-world Mirai worm trace, we demonstrate that SWORD is superior to existing detectors at detecting inbound worms, especially those that are superspreading or surreptitious.

1.3 Detecting DDoS Traffic Leaving an IoT Network by Inspecting Traffic at Its Gateway

The growth of the IoT is contributing to the rise in cyber attacks on the Internet. Unfortunately, the resource-constrained IoT devices and their networks make many classical security systems ineffective or inapplicable. We present TWINKLE, a security framework for smart home environments that considers the unique properties of IoT networks. TWINKLE utilizes a two-mode adaptive security framework that allows an IoT network to be in regular mode for most of the time, which incurs a low resource consumption rate, and switch to vigilant mode only when suspicious behavior is detected, which potentially incurs a higher overhead. We also describe critical challenges in implementing TWINKLE in an IoT environment and explain in detail how we addressed each challenge. We also provide technical details of the TWINKLE framework, such as the threads that make up each component and the messages that are passed between threads. Furthermore, we show the efficacy of TWINKLE, when deployed on a gateway router, in detecting DDoS traffic leaving an IoT network. Our evaluations show that TWINKLE is not only effective at securing resource-constrained IoT networks, but can also successfully detect and prevent DDoS attacks with a significantly lower overhead than existing solutions.

1.4 Mitigating Attacks as Close to the Devices in an IoT Network Originating DDoS Traffic

In some IoT networks, security solutions cannot rely on a central location, where all the network traffic passes through, to be deployed at. Such networks require security solutions to defend against attacks as close to the devices as possible. In this chapter, we focus on defense close to the devices in an IoT network originating DDoS traffic, and present the mobile firewall system, which leverages the SDN and NFV design paradigms to publish and subscribe security functionality in an IoT network. This system consists of mobile security nodes that monitor the network on-demand, and a security controller that decides which security functions to deploy on the mobile security nodes, given network telemetry data provided to it by the mobile security nodes. In order to detect and mitigate DDoS traffic generated by an IoT network, the mobile firewall does not need to modify devices in the network. Additionally, security functions can be installed on-demand to handle new security threats as they emerge. Through initial simulations, we show the efficacy of the mobile firewall system at mitigating DDoS traffic from an IoT network. The mobile firewall system is an ongoing project that we will continue to work on in the future. We plan on eventually implementing a fully functioning prototype in a real IoT network.

1.5 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter II, we provide a background of IoT-enabled DDoS attacks, common vulnerabilities that lead to IoT-enabled DDoS attacks, and countermeasures for handling IoT-enabled DDoS attacks. In Chapter III, we briefly survey the current state of IoT security, and discuss several challenges facing IoT security research and missing gaps that we believe have not been sufficiently addressed. In Chapter IV, we present SWORD, our solution to preventing IoT devices from being turned into DDoS bots by inspecting traffic towards IoT networks at an upstream ISP/IXP. In Chapter V, we present TWINKLE, our solution to detecting DDoS traffic leaving an IoT network by inspecting traffic at its gateway. In Chapter VI, we present the mobile firewall system, our solution to mitigating attacks as close to the devices in an IoT network originating DDoS traffic. Finally, we discuss future work in Chapter VII, and conclude the dissertation in Chapter VIII.

1.6 Co-authored Materials & Acknowledgment

1.6.1 Co-authored Materials. Most of the content in this thesis is from published and unpublished work. Below we connect each chapter to the material and authors that contributed to it.

- Chapter II and Chapter III:
  - Published as Devkishen Sisodia. On the State of Internet of Things Security: Vulnerabilities, Attacks, and Recent Countermeasures.

  Computer and Information Science, University of Oregon, Technical Report, AREA-202001-Sisodia, 2020

- Chapter IV:
  - Unpublished as Jun Li, Devkishen Sisodia, Shad Stafford. On the Detection of Smart, Self-Propagating Internet Worms. *IEEE Transactions on Secure and Dependable Computing*, 2021. **In submission.**
- Chapter V:
  - Unpublished as Devkishen Sisodia, Jun Li, Samuel Mergendahl, Hasan Cam. A Two-Mode, Adaptive Security Framework for Smart Home Security Applications. *ACM Transactions on Internet of Things*, 2021. **In submission.**
  - Published as Devkishen Sisodia, Samuel Mergendahl, Jun Li, Hasan Cam. Securing the Smart Home via a Two-Mode Security Framework. *International Conference on Security and Privacy in Communication Networks*, 2018.
- Chapter VI:
  - Unpublished as Derek Strobel, Sam Mergendahl, Zhangxiang Hu, Matthew Supan, Devkishen Sisodia, Jun Li. An Open Firewall Ecosystem with On-Demand Security for Internet of Things, 2020. **In preparation.**

1.6.2 Acknowledgment. The material presented in this thesis is partially based upon work supported by the National Science Foundation under

Grant No. 0644434 and 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 National Science Foundation, the Department of Homeland Security, or the US Government.

CHAPTER II

BACKGROUND: IOT-ENABLED DDOS

In the this chapter, we provide an overview of IoT-enabled DDoS attacks, detail vulnerabilities that can lead to IoT-enabled DDoS attacks, and present recent countermeasures to tackle IoT-enabled DDoS attacks.

The chapter is derived in part from the following published work: On the State of Internet of Things Security: Vulnerabilities, Attacks, and Recent Countermeasures [131] by Sisodia, D. I am the leading author of this work, and the content of this chapter was written entirely by me.

2.1 An Overview of IoT-Enabled DDoS

A distributed denial-of-service (DDoS) attack is an attack that attempts to render a service, machine, or network, which we term the victim, inaccessible to benign users by overwhelming the victim with a flood of network traffic, which we term attack traffic. IoT-enabled DDoS attacks are simply DDoS attacks where the attack traffic is generated by infected IoT devices. Over the last decade, the Internet is witnessing a rise in IoT-enabled DDoS attacks [68], and this trend will only continue to grow.

Let us begin by studying how IoT-enabled DDoS attacks are generally launched on the Internet. Before launching an IoT-enabled DDoS attack, an attacker must have access to a botnet of infected IoT devices which can be used to send DDoS traffic to the victim. In order to build an IoT botnet, a malicious entity, which we call the botmaster, will initially set up command and control (C&C) servers around the world, as shown in Figure 2. The botmaster will use these C&C servers to first initiate the infection of IoT devices and later send commands to the infected IoT devices.

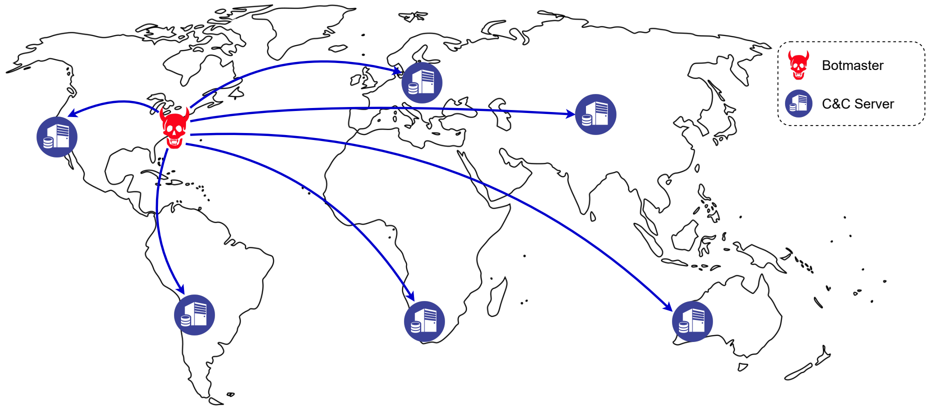


Figure 2. Setting up C&C servers.

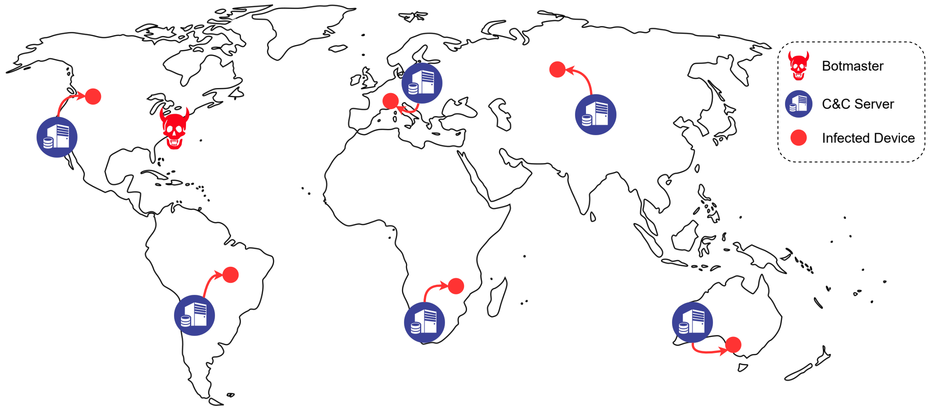


Figure 3. Initial infection.

Once set up, the C&C servers will begin the initial infection of IoT devices through worm propagation, as shown in Figure 3. It does so by scanning the Internet for potentially vulnerable IoT devices susceptible to brute-force attacks. For example, the Mirai worm identified potential victims by sending TCP SYN probes to pseudo-random IPv4 addresses on telnet TCP ports 23 and 2323 [12]. If an IoT device were to respond to a probe, the C&C server would enter into a brute-force login phase, during which the C&C server would try to establish a telnet connection using predetermined username and password pairs from a list of common credentials or default credentials from IoT vendors. Unfortunately, this was a highly effective attack because there are hundreds of thousands of devices

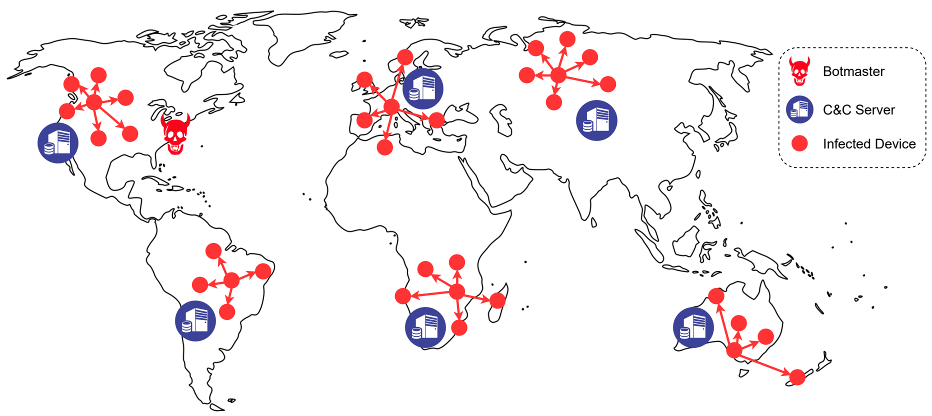


Figure 4. Worm propagation.

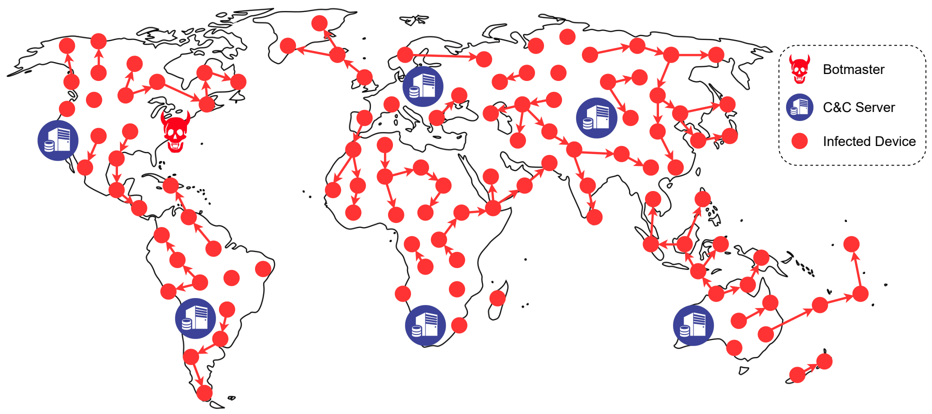


Figure 5. Formation of a large-scale botnet.

on the Internet which use default settings. Once infected, the device, now a bot, will listen for commands from the C&C server. The initially infected IoT devices will continue to spread the worm by rapidly scanning the Internet and launching brute-force attacks themselves, as shown in Figure 4, eventually forming a large-scale botnet that potentially spans the entire globe, as shown in Figure 5.

After the botnet is formed, the botmaster can initiate a DDoS attack by notifying the C&C servers of the victim IP address to target, as shown in Figure 6. The C&C servers will then relay that information to the bots, as shown in Figure 7. Note, depending on the attack, the botmaster may only select a subset of bots to use for the attack. Finally, the subset of attacking bots flood the victim with

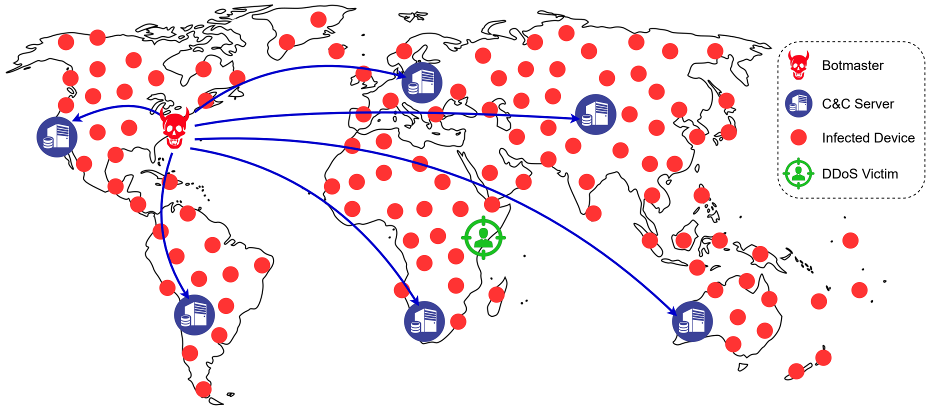


Figure 6. Botmaster initiates a DDoS attack.

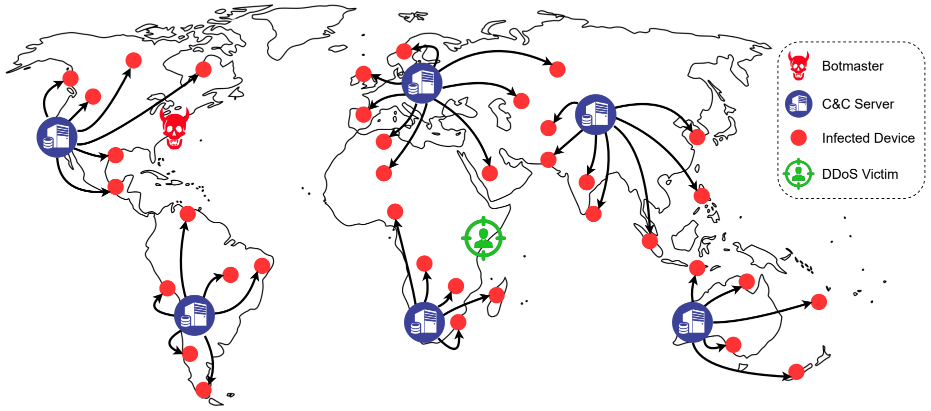


Figure 7. C&C servers command the infected devices to attack.

attack traffic thus causing a DDoS attack, as shown in Figure 8. Most botnets have the capability to launch numerous different types of DDoS attacks. For example, Mirai's DDoS attack capabilities include TCP SYN flooding, UDP flooding, TCP ACK flooding, GRE-flooding, along with HTTP GET, POST and HEAD attacks, among others [12].

2.2 Vulnerabilities That Cause IoT-Enabled DDoS Attacks

2.2.1 Open Ports. A major concern to the security of IoT networks is the significant number of devices with unnecessarily open ports. The first step in infecting an IoT device with malware to turn it into a DDoS bot is to gain access to it by creating a connection to it on an open port. Czyz et al. [40] showed

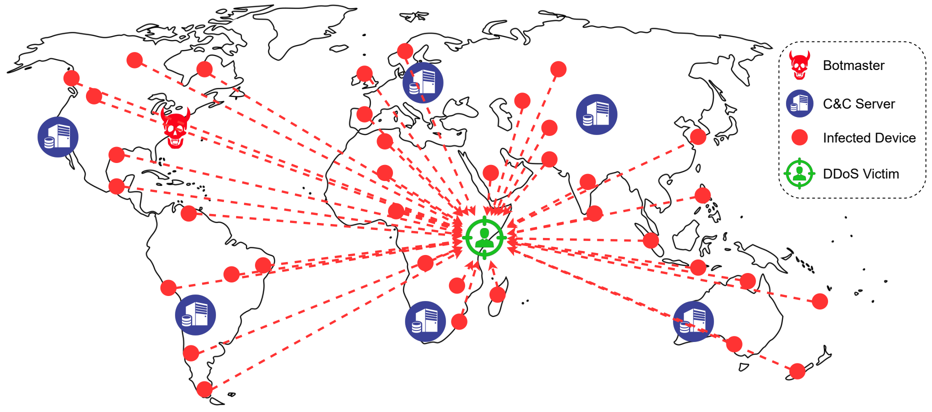


Figure 8. DDoS attack.

that a large number of IoT devices are only reachable over IPv6, and various IoT protocols are more accessible over IPv6 than over IPv4 (e.g., 6LoWPAN). They discovered that a given IPv6 port is almost always more open than the same port is in IPv4. For example, IPv6 had 5% more open SSH ports, and 46% more open Telnet ports as compared to IPv4. They also concluded that there was a systemic failure in organizations to deploy consistent security policies for their devices as it pertains to port blocking. Lastly, the authors debunked the belief that the security threat of open ports in IPv6 is dampened due to the infeasibility of IPv6 network-wide scanning by discovering high-value hosts through scanning alone.

2.2.2 Security Flaws in Firmware Images. Costin et al. [37] performed simple static analysis on 32,000 embedded firmware images, and discovered 38 previously unknown vulnerabilities in almost 700 firmware images. They were also able to verify that some of those vulnerabilities are affecting at least 140,000 devices on the Internet. Specifically, the authors were able to extract private RSA keys and their self-signed certificates used in an estimated 35,000 online devices (many of these devices were surveillance cameras). The authors were also able to extract 100 distinct hard-coded password hashes, and, for 58 of them, recover the original passwords. However, simple static analysis alone is insufficient to discover all of the vulnerabilities in a given firmware image, and therefore the number and severity of the vulnerabilities discovered through only simple static analysis should be very worrying.

2.2.3 Lack of Reliable Patching and Update Mechanisms. In order to minimize the number of attack vectors, operating systems, firmware, and applications should be patched regularly. However, many manufacturers either do not regularly maintain security patches, or do not have automated update mechanisms in place [99]. Some devices themselves may lack software updating capabilities, while others may outlive the limited period for which they receive updates. Even if updates are available, they may be challenging to apply, as some devices may require users to actively install updates, while those that automatically install might need to restart to update their firmware, causing gaps in availability [125]. Furthermore, even available update mechanisms lack integrity guarantees, rendering them vulnerable to man-in-the-middle and modification attacks.

2.2.4 Weak Credentials and Lack of Strong Authentication Mechanisms. Weak credentials and lack of strong authentication mechanisms is a major concern for IoT devices. In 2010, Cui et al. [39] conducted Internet-scale probing and uncovered more than half a million embedded devices with default credentials. Most of these devices belonged to government organizations, large enterprises, ISPs, and educational institutions. Two years later, in 2012, the Carna botnet revealed that there were more than 1.2 million devices online with no or default credentials [92]. Weak credentials have in fact led to large-scale, real-world attacks [92, 12, 89, 62].

Note, brute-force, or dictionary attacks, can easily be launched against IoT devices, especially those using weak credentials. As we discussed, there are a significant number of open IoT devices with no, default, or weak credentials, a vulnerability that can be exploited fairly easily. One of the most notorious examples of such exploitations is the Mirai malware which at one point infected over 300,000 IoT and embedded devices all over the world [12]. The botnet that was formed from the infected devices was used to launch devastating DDoS attacks against multiple targets (Krebs on Security, OVH, and Dyn), where the Krebs on Security attack exceeded 600 Gbps in volume, making it the largest DDoS attack recorded at the time. A study conducted by Cetin et al. [26] showed that while Mirai could be removed by rebooting an infected device, simply rebooting the device will not fix the underlying problem as the device remains vulnerable to infections once it comes back online. As the authors note, removing the underlying problem would require affected users to change default passwords, or update the firmware — measures that are much more complicated than a mere reboot. In recent years, the Mirai malware has been evolving into potentially much more dangerous strands, such as the Hajime malware [62], whose real purpose remains a mystery as it has no attacking code for launching DDoS attacks.

2.3 Countermeasures Against IoT-Enabled DDoS Attacks

2.3.1 Detecting the Execution of Malicious Processes.

Breitenbacher et al. [19] presented a host-based anomaly detection system called HADES-IoT, to detect if unauthorized processes, such as those belonging to IoT malware, are attempting to execute. HADES-IoT has proactive detection capabilities, and provides tamper-proof resistance. It is installed inside a device's kernel space, and can be deployed on a wide range of Linux-based IoT devices.

HADES-IoT is first pre-compiled and delivered to the device. The kernel's initialization file is modified accordingly to ensure that HADES-IoT is always executed when the device is booted. Once executed, HADES-IoT monitors and collects information about all calls to the *execve* system call. It only allows a set of whitelisted processes, which it learns of during a profiling phase, to call the *execve* system call. The authors deployed HADES-IoT on seven IoT devices and achieved 100% effectiveness in detecting recent IoT malware strains, such as VPNFilter and IoTReaper, while on average, requiring only 5.5% of available memory, but in some cases incurs almost 40% extra CPU usage.

2.3.2 Reliable Patching and Update Mechanism. Along with insecure firmware images, one of the major vulnerabilities facing IoT devices is the lack of reliable patching and update mechanisms. Mainly to address the issue of patching in IoT environments, Simpson et al. [125] proposed a central security manager that can be installed on top of a smart home's gateway router. The security manager is aware of the status of all devices in the home, by observing all network traffic leaving and entering the network. By having knowledge of each devices status, and keeping track of reported vulnerabilities, the security manager could potentially intervene as needed to deter or alleviate many types of security risks. Modules can be built on top of the manager to offer convenient installation of software updates, filter traffic that might be malicious, and strengthen authentication for legacy devices. As the authors elaborate, the proposed modules can help improve security for the IoT devices at different stages in the patching process:

1. Stage 1: Vulnerability discovery but no patch yet. In this case, the goals of the security manager are to identify the affected devices, filter attack flows

to these devices, reduce the impact of default passwords, mitigate cross-site request forgeries, secure vulnerable SSL/TLS connections, block vulnerable APIs on the affected devices, and alert the user.

2. Stage 2: Acquiring the patch. In this case, the main goal of the security manager is to apply the patch as quickly as possible by alerting the user. If the device is offline or in use, the security manager will pre-fetch the patch and apply it later, while continuing to protect the device as described in Stage 1.

3. Stage 3: Applying the patch. In this case, the goal of the security manager is to simply apply patches when it is safe to do so.

4. Stage 4: Device compromised. In this case, the goal of the security manager is to prevent the compromised device from harming other devices in the network.

The security manager is a proof-of-concept, as implementation details of the security manager and modules are not discussed in the paper.

2.3.3 Management of Compromised Devices. Instead of focusing solely on device identification, Xu et al. [154] presented an idea more related to IoT device management. The authors argued that large IoT networks, consisting of many nearly identical devices, are especially attractive targets to attackers. However, recovery from compromises by conventional means is costly and slow, especially if the devices are distributed over a large geographical area, where network administrators or operators would have to travel to the devices to manually recover them. The authors presented CIDR, a system that can recover compromised IoT devices within a short amount of time, even if attackers have taken root control of every device in the network. Once a network administrator identifies a compromise and creates an updated firmware image, he/she can direct CIDR to force the devices to reset and to install the patched firmware. The authors call the ability of one computing device (i.e., the server on which CIDR is running) to control the software configuration of another computing device *dominance*. Unfortunately, CIDR requires hardware modifications, making it difficult to be applied in the real-world.

CHAPTER III

THE STATE OF IOT SECURITY

Due to the prevalence of IoT and the security threats facing it, we briefly survey the area of IoT security. The goal of this chapter is to provide a holistic-view of the vulnerabilities, attacks, and recent countermeasures in the Internet of Things. This chapter is particularly focused on countermeasures proposed in the last five years to capture the more recent landscape of IoT security. We hope to provide the reader with a view on what security-related topics the IoT research community has been focusing on, mainly in the last five years. We also discuss several challenges facing IoT security research, and missing gaps that we believe have not been sufficiently addressed.

The chapter is derived in part from the following published work: On the State of Internet of Things Security: Vulnerabilities, Attacks, and Recent Countermeasures [131] by Sisodia, D. I am the leading author of this work and the content of this chapter was written entirely by me.

3.1 Vulnerabilities

In Chapter 2.2, we describe several vulnerabilities that can lead to IoT-enabled DDoS attacks. In this section, we mainly focus on vulnerabilities that have peaked the interest of the IoT security research community in recent years.

3.1.1 Lack of Necessary Power for Cryptographic Primitives.

While it is clear that encryption can help to address some of the vulnerabilities presented in [153], complex cryptographic functions, such as those found in the Advanced Encryption Standard (AES), can result in large overhead for resource-constrained IoT devices. As a result, there is a growing interest in ultra lightweight, but secure encryption algorithms optimized for low-powered hardware. However, as Singh et al. [127] pointed out, hardware-based encryption engines have a significant vulnerability: the power dissipation of the hardware can be measured while performing encryption, and later statistically analyzed to recover the secret key, thus compromising the device. Many countermeasures have been proposed to address this vulnerability in AES engines. Unfortunately, these countermeasures incur significant power and performance overheads, and therefore are not suitable for lightweight cryptographic primitives.

3.1.2 Security Flaws in Various Communication Protocols.

Before we present the vulnerabilities of 6LoWPAN, ZigBee, and BLE, we provide a brief overview of each. 6LoWPAN uses the Constrained Application Protocol (CoAP) [122] as its application layer protocol, UDP with DTLS at the transport layer, IPv6 over LoWPAN with Routing Protocol for Low-Power and Lossy Networks (RPL) [152] as the routing algorithm at the network layer, and finally IEEE 802.15.4 at the data link and physical layers. ZigBee on the other hand uses ZigBee-specific protocols at the application, transport, and network layers, with a routing algorithm similar to Ad hoc On-Demand Distance Vector Routing (AODV) [104], but like 6LoWPAN, also uses IEEE 802.15.4 as the underlying data link and physical layer protocol. Unlike 6LoWPAN and ZigBee, BLE supports its own physical (BLE PHY) and data link layer (BLE MAC) protocols, but like ZigBee, also supports its own application, transport, and network layer protocols.

Granjal et al. [56] showed that Neighbor Discovery (ND) and mesh routing mechanisms in IEEE 802.15.4 environments, such as 6LoWPAN and ZigBee, are susceptible to security threats, such as routing and data leakage attacks. While RPL has a self-healing mechanism to deal with routing topology failures, link failures, and node failures, Wallgren et al. [144] showed that this mechanism cannot self-correct networks under certain selective forwarding attacks.

ZigBee devices use symmetric-key encryption to establish secure communications. However, if keys are not pre-installed on devices that want to communicate, they are transmitted unencrypted, making it feasible for attackers to not only gain sensitive information, but control over the devices, as demonstrated by Vidgren et al. [143]. Furthermore, studies by Zillner [169] and Morgner et al. [95] showed that the ZigBee Light Link (ZLL) protocol, which is used in smart home lighting systems to set up the connection between smart lights and the hubs responsible for controlling them, had a security flaw that allowed control of smart lights to be taken from their hubs and given to adversaries. This vulnerability can be exploited to launch crippling distributed-denial-of-service (DDoS) attacks against smart cities.

BLE also has its share of vulnerabilities. As shown by Ryan [115], a flaw in the key-exchange protocol for Bluetooth allows an attacker to passively recover session keys. Ho et al. [64] showed how door locks could be opened by launching relay attacks on BLE — an attacker can capture a lock's BLE authentication challenge message, relay the message (possibly over Wi-Fi for long-range communication) to an accomplice who is several meters away from the lock's rightful owner; the accomplice would then broadcast the challenge message and capture the response from the rightful owner's device, and relay the response back to the attacker who can now use the response to unlock the door. Even more worrying is that home-based IoT devices have vulnerable legacy versions of low energy protocols implemented in hardware, thereby limiting their mitigation options, as described by Alrawi et al. [9].

3.1.3 Lack of Data Flow Control in Trigger-Action Platforms.

In recent years, researchers have begun studying the potential vulnerabilities of trigger-action IoT platforms. The trigger-action programming paradigm provides a simple and intuitive abstraction for non-technical users to automate IoT devices, and as a result, is commonly used for home automation. Essentially, a trigger-action program contains a set of trigger-action rules that specify that when a certain trigger event occurs, such as motion is detected, one or more actions, such as turn on the lights, should be subsequently executed.

Wang et al. [148] comprehensively analyzed the interactions between trigger-action rules in order to identify inter-rule vulnerabilities. Some of these vulnerabilities, if exploited, could lead to serious damage. For example, an attacker could exploit an action loop vulnerability to continuously turn on and off a light potentially inducing seizures, or the action duplicate to inject medicine multiple times potentially causing a fatal reaction.

Once the authors identified the inter-rule vulnerabilities, they then analyzed 315,393 applications from a popular trigger-action programming platform, IFTTT (acronym for If This Then That), and found that 66% of them had potential for inter-rule vulnerabilities. Due to the popularity of trigger-action platforms, coupled with the aforementioned serious vulnerabilities, there has been a push within the research community to find appropriate countermeasures to these vulnerabilities.

3.1.4 Lack of Verification in Virtual Personal Assistant Services. Lastly, we describe speech recognition and voice-injection vulnerabilities that are present in virtual personal assistant (VPA) services running on IoT devices. Given a voice commands, such as "will it rain today?", VPA services can run a function, or a skill, to respond to the command. Most

VPA services, such as Amazon's Alexa and Google Assistant, allow third-party skills to be uploaded to their skills market to further enrich the user experience when interacting with the VPA services. Unfortunately, Kumar et al. [78] found a vulnerability, which was later termed invocation confusion [165], that when exploited could cause a voice squatting attack (VSA). This vulnerability stems from the fact that VPA services allow for different skills to be initiated by same or similar sounding commands. Therefore, an attacker could exploit this vulnerability to create a malicious skill that can be initiated unknowingly by a victim. The authors give the example of a malicious skill, which attempts to phish credit card information, that is initiated by the words "Capital Won". In this case, the victim may initiate the malicious skill when he/she says the words "Capital One", which in normal circumstances (in absence of the malicious skill) would initiate the legitimate skill (which opens the authentic Capital One application). Speech recognition and voice-injection vulnerabilities in VPA services can cause more interesting attacks, some of which we discuss later in this chapter.

3.2 Attacks

In this section, we describe various attacks common in IoT networks. Most of the attacks we cover in this section exploit the vulnerabilities we described in Chapter 3.1.

3.2.1 Side-Channel Attacks. Attackers can exploit data leakage vulnerabilities to gain sensitive information about users. One such example was a side-channel attack described in 2015 [147]. The goal of the attack was to discover the keystrokes of a victim typing on a keyboard from their wrist position inferred by accelerometer and gyroscope data sensed from a smartwatch. The authors tested the attack on eight real users, each of whom was asked to type 300 different English words from a dictionary of 5000 words while wearing a smartwatch on their left wrist. The results showed that the keystroke detection rates for the left-most keys on the keyboard (Q, W, E, R, T, A, S, D, F, G, Z, X, C, V, B) were from 88% to 99% accuracy, while the keystroke detection rates for the right-most keys on the keyboard (Y, U, I, O, P, H, J, K, L, N, M) were from 3% to 5%. These keystroke detection rates allowed the authors to have a 50% chance to narrow down each possible word typed by 99.5% (in other words, for each word the user typed, the attack had a 50% chance of accurately eliminating 4976 words from the 5000-word dictionary of possible words). Unlike similar previous papers that explored side-channel attacks from sensor data leakage of smartphones, this paper was one of the first to present an attack from sensor data leakage of wearable devices with relative success, without requiring any training from the victim. In 2016, Wang et al. [146] improved the accuracy of the attack and tested it on real ATM machines with over a 93% success rate.

Alternatively, sensitive data can also be leaked through traffic analysis attacks. Copos et al. [36] presented a traffic analysis attack on encrypted smart home network traffic. The authors analyzed the network traffic of Nest Thermostat and Nest Protect (smoke and carbon monoxide detector) devices, and showed that potentially sensitive information about the state of the smart home could be discerned from the encrypted traffic. Specifically, the authors captured traffic to and from the Nest devices within the smart home network, and decrypted the WPA encrypted traffic so that they could identify the hosts that the devices frequently contact. However, the packets are still encrypted with SSL/TLS. By analyzing the types of requests (e.g., HTTPS, NTP, DNS, etc.) and the frequency of requests, the authors were able to discover when the devices switched between *Home* (home is occupied) and *Away* (home is unoccupied) modes. For example, there was a discrepancy in the frequency of NTP requests generated between when the Nest Thermostat was operating in *Home* mode and when it is in *Away* mode. From this information, the authors created a simple Support Vector Machine (SVM)-based learning model that achieved 81% accuracy (with zero false positives) in predicting the mode of the Nest Thermostat.

Apthorpe et al. [13] extended the work done in [36] by demonstrating that ISPs or other network observers could infer privacy sensitive in-home activities by simply analyzing the traffic rates generated from smart homes containing commercially-available IoT devices, even when those devices used encryption. Clearly, an attacker aware of this correlation between smart home traffic rates and device states could easily distinguish user activities within the smart home. The authors present a traffic shaping defense method for preventing such traffic analysis attacks.

Covert channel attacks presented in works [158] and [113] show that an attacker can use IoT devices to bypass security mechanisms, such as firewalls, traffic monitors, and information flow control systems, by routing sensitive data through covert channels. Yang et al. [158] presented a covert communication method called NICScatter, where a mobile device, infected with malware, backscatters surrounding radio frequency (RF) signals at varying intensities to transmit sensitive information retrieved from the device's memory to the attacker's phone. In order to launch this attack, the malware controls the impedance of a device's wireless network interface card (NIC). While the authors only tested their attack on laptop and smartphone NICs, this attack is very much possible in an IoT environment. In fact, Ronen et al. [113] showed that it was possible to create a covert channel with smart LED lights. The authors were able to misuse smart LEDs' APIs to cause the lights to oscillate between different intensities, in such a way as to be indistinguishable to the human eye, to surreptitiously transmit data to a light sensor.

3.2.2 Voice-Command Injection Attacks. In recent years, the IoT security research community has focused much attention on voice-command injection vulnerabilities. In this subsection, we present three papers that exploit such vulnerabilities, one on inaudible command attacks [164], and two on skill squatting attacks [78, 94].

In 2017, Zhang et al. [164] designed a completely inaudible attack, which they called the DolphinAttack, that modulates voice commands on ultrasonic frequencies (e.g., $f > 20$ kHz) using the amplitude modulation technique (AM) to achieve inaudibility. By leveraging the nonlinearity of electret condenser microphone (ECM) and MEMS microphone circuits, the modulated low-frequency audio commands can be successfully demodulated, recovered, and interpreted by the speech recognition systems. The authors showed that with the DolphinAttack, an adversary can inject malicious and inaudible voice commands into major speech recognition systems, including Siri, Google Now, and Alexa.

3.2.3 Selective-Forwarding Attacks. In a selective-forwarding attack, a malicious device, which is located on the route between the sources and destinations of certain connections, launches a denial-of-service (DoS) attack by forwarding only a subset of the packets that it receives to the destination [144]. Although this attack is primarily targeted to disrupt routing paths, it can be used to filter any type of traffic, no matter the protocol. For example, an attacker could forward all RPL control messages (thereby preventing devices from detecting routing inconsistencies), and drop all other traffic. In this case, RPL's self-healing mechanisms will not be able to detect the DoS attack, as the control messages are still being forwarded.

Selective-forwarding attacks can be coupled with sinkhole attacks to become even more powerful. In sinkhole attacks, a malicious device advertises an artificial, but attractive routing path, thereby causing many nearby devices to route traffic through it [144].

It is generally fairly difficult to defend against all selective-forwarding attacks. However, there are some security mechanisms that can minimize the impact of these types of attacks. For example, through encryption, legitimate devices can ensure that an attacker cannot distinguish between different types of traffic, thus forcing the malicious device to either forward all or none of the traffic it receives. To this end, devices can use IPsec to secure RPL control messages. Additionally, analysis of application-layer traffic can help detect if any application traffic is lost, and devices may report such losses to the underlying RPL system in order to improve path quality by finding a route that bypasses the malicious device.

3.2.4 Battery-Draining Attacks. Battery-draining, or energy-depleting attacks are another way attackers can target the availability of IoT devices. Vasserman et al. [141] present various battery-draining attacks, called Vampire attacks, which target wireless ad hoc sensor networks. Vampire attacks use routing protocols to permanently disable networks by depleting devices' battery power. These attacks do not depend on particular protocols or implementations, but rather rely on the properties of many popular classes of routing protocols. These properties include link-state, distance-vector, source routing, and geographic and beacon routing. The authors show through simulation that depending on the location of the adversary in the network, network energy expenditure increases from between 50 to 1000 percent. While the authors present Vampire attacks on ad hoc sensor networks, smart home networks could just as easily fall prey to such attacks, as many smart home devices, many of which are mobile and battery-powered, use routing protocols similar to those used in ad hoc sensor networks, such as AODV (used by ZigBee devices).

Chiariotti et al. [31] also analyze battery-draining attacks from a game-theoretic perspective. The authors use game theory to model a smart IoT device defending itself from a similarly energy-constrained jammer. Both devices are assumed to be rational actors, and their interactions can be modeled as a zero-sum game The mathematical properties of this zero-sum game are exploited to find an effective defense solution.

3.2.5 DDoS Attacks.

Lastly, we describe unique DDoS attacks presented in recent years against IoT devices and networks. In 2018, Soltan et al. [136] and Ronen et al. [114] presented DDoS attacks that could potentially disrupt entire cities or countries. Soltan et al. demonstrated that a botnet of high wattage IoT devices, such as air conditioners and heaters, gives an adversary the unique ability to launch large-scale coordinated attacks on a city's or country's power grid. By synchronously switching on/off compromised high wattage IoT devices, and thereby manipulating the total power demand, an adversary could significantly disrupt a power grid's normal operation. Through simulation of the attack on Poland's power grid, the authors show that this attack, in the extreme case, could cause large-scale, country-wide blackouts. With a botnet of only 210 devices, the attack could initiate a cascading failure resulting in a power outage covering 86% of the country.

Ronen et al. described a new type of malware worm that infects IoT devices that are in close proximity with one other, thereby rapidly spreading over large areas, given that the density of compatible IoT devices exceeds a certain amount (i.e., critical mass). The authors tested the malware on the popular Philips Hue smart lamps. The worm spreads by jumping directly from one lamp to its neighboring lamps, using only the built-in ZigBee wireless mesh connectivity. The infection can begin spreading by a user plugging in a single infected bulb anywhere in a city, and the worm can catastrophically spread throughout the city within minutes. The malware enables an attacker full control of the city's smart lamps – he/she can turn all of the lights on or off, permanently brick them, or use them in a massive DDoS attack. To make such an attack possible, the authors had to find a way to:

1. remotely break the connection between already installed lamps and the hubs controlling them (i.e., remove them from their networks), and

2. perform over-the-air firmware updates.

As we mentioned in Chapter 3.1, the authors exploited the known vulnerability in the ZLL protocol to overcame the first problem. To solve the second problem, they developed a side-channel attack to extract the global AES-CCM key (for each device type) that Philips uses to encrypt, and authenticate new firmware.

Two examples of real-world DDoS attacks targeting IoT devices are the BrickerBot [32] malware, and the attack on Finland's central heating and warm water circulation systems [71]. In the span of about one year (November 2016 to December 2017), the BrickerBot malware managed to permanently destroy over 10 million IoT devices, in the hopes, according to its author, to prevent those bricked devices from being infected with the Mirai malware and launch DDoS attacks themselves. Also in November 2016, a DDoS attack halted heating distribution in two buildings in Finland. While the attack did not target any IoT devices directly, but rather the computers that controlled the devices responsible for central heating and warm water circulation, this attack prevented the devices from functioning normally. Thankfully the attack was easily mitigated by limiting network traffic, and no one was hurt. However, these real-world attacks that target critical infrastructure should make us wary about the potential damage that IoT-based attacks can cause.

3.3 Countermeasures

In this section, we describe various recent countermeasures to the aforementioned IoT vulnerabilities and attacks. Most of the countermeasures we cover in this section attempt to prevent, detect, and/or mitigate the attacks we described in Chapter 3.2.

3.3.1 Access Control. In recent years, researchers have found the need for reenvisioning access control and authentication for multi-user-per-device environments, like the smart home. As He et al. [60] noted in their 2018 study of access control and authentication in the home IoT, current authentication methods for the home IoT appear transplanted from smartphone and desktop paradigms, which for the most part, assume a single-user-per-device environment. However, in IoT environments, such as the smart home, multiple users with complex social relationships may interact with a single device. Therefore, He et al. conducted a 425-participant online user study and found major differences in the participants' desired access-control policies for different capabilities within a single device (e.g., updating software, turning lights on/off, turning cameras on/off, adding new user, etc.), as well as based on who is trying to use that capability (e.g., spouse, teenager, child, visiting family, babysitter, neighbor, etc.). This study allowed them to pinpoint various contextual factors (e.g., time of day, location of user, location of device, who is near by, etc.) that, along with capabilities and relationships, dictate the specification of more complex, yet desired, access-control policies.

One year later, Zeng et al. [163] applied the complex access-control policies derived from [60], among other design principles from other studies, to create an access control system (in the form of a mobile application) for the smart home. The application included four types of access controls:

- Role-Based Access Control: Each user is assigned a role — admin, child, or guest. Only admins are allowed to change access control policies, add new users, and organize the devices.
- Location-Based Access Control: Users can be restricted from using devices if they are not physically near the device.
- Supervisory Access Control: Allows a user who may be restricted from using a device, to use the device, if and only if another (authorized) user is nearby.
- Reactive Access Control: If a user attempts to use a device they do not have permission to use, the application will ask a more privileged user for permission in real-time, by sending a notification asking them to approve or deny the request.

The authors further tested their application in seven households in the Seattle metropolitan area. Although the users expressed that they had a strong need for certain access controls in their homes before the study, only a few ended up using the application. While usability was one factor that went into this, most participants were either unconcerned about restricting access to mundane devices, or that existing social norms and trust in their households checked against bad behavior. For example, children were trusted to follow rules, roommates respected each others' spaces, and for the most part, people were not concerned about information revealed by the smart home when it matched their household's privacy norms. The users were also willing to accept (multi-user) security and privacy risks posed by usage of devices because of the convenience and utility lost by using the access control mechanisms. The study reemphasized the notion that the design of security and privacy features for a smart home must work with, and not limit, a user's primary use case for the smart home.

Hong et al. [65] argued that IoT device communications should be default-off and only explicitly enabled when absolutely needed because IoT applications and devices, unlike traditional applications and computers (e.g., web browser or laptop), serve narrowly defined purposes. The authors proposed, Bark, a policy language and runtime for specifying and enforcing minimal access permissions in IoT networks by whitelisting access for IoT applications when communication is default-off. Bark constructs access control policies through natural questions (e.g., who, what, where, when, and how), and transforms them into transparently enforceable rules for IoT application protocols. However the authors assume that the gateway, where policies are being enforced, can observe all traffic between devices, applications, and remote servers. This means that the system would not be able to enforce rules for device-to-device communication in protocols that allow for mesh networking (e.g., 6LoWPAN, ZigBee, Bluetooth).

3.3.2 Data Flow Control.

As discussed in Chapter 3.1, the lack of data flow control is a significant vulnerability in smart home environments. We analyze five recent works [53, 54, 14, 23, 91] related to controlling how data flows between sources and sinks.

Fernandes et al. [53] introduced a method called FlowFence for restricting an application's access to sensitive IoT data. The authors noted that while IoT frameworks use permission-based access control for data sources and sinks, they do not control *flows* between the authorized sources and sinks. To explain this notion further, let's consider the example of an application that unlocks a door based on the person's face. It extracts features from the person's face and compares them to the features belonging to authorized people, checks the current state of the door, unlocks the door, and sends a notification to the home owner over the Internet. The user may have given this application permission to use the camera, unlock the door, and access the Internet, however, there is no way for the user to ensure that the application does not leak camera data to the Internet. To be supported by FlowFence, an application developer must split his/her application into two parts — one part is code that deals with non-sensitive data, and the other that does. The code that deals with sensitive data is "quarantined". FlowFence then runs the application in a sandbox to generate a data flow graph, which is analyzed to determine if any data flowing through the quarantined code violates any user defined flow rules (e.g., no data flowing through quarantined code should be sent over the Internet).

Fernandes et al. applied rule-based flow tracking and control properties for action integrity for trigger-action IoT platforms [54] as an extension of their work in [53]. They noted that IFTTT, and other trigger-action platforms, can have overprivileged access to smart home devices, which can cause serious attacks if the authorization tokens, which give the platforms the ability to control the devices, are compromised. For example, a user could allow a IFTTT application to change the color of the smart lights in their living room if a friend posts a picture on FaceBook. In order to create this type of application, the setup process would include the user allowing the smart lights' action service (which could be located on the devices themselves, the hub controlling the devices, or in the cloud) to provide an authorization token to the application. Although the application should only control the lights' color, with the authorization token it has full control over the lights. Therefore, if the application accidentally leaks the authorization token or if the platform itself is compromised, this would potentially give an attacker full control over the lights. In [54], Fernandes et al. present a security principle that ensures that an attacker who controls a compromised trigger-action platform can only invoke actions and triggers that are specified in the rules that users have created (e.g., can only control the color of the lights), can invoke actions only if it can prove to an action service that the corresponding trigger occurred in the past within a reasonable amount of time, and cannot secretly tamper with any data passing through the platform.

Similarly to [53] and [54], Bastys et al. [14] and Celik et al. [23] evaluate the runtime execution of applications to enforce policy-based information flow control for trigger-action platforms. Bastys et al. studied a dataset of 279,828 IFTTT applets and found that that 30% of the applets were at risk of violating privacy. To tackle this problem, they argued and demonstrated that information flows should and can be secured in applications by state-of-the-art information flow tracking techniques. Celik et al. [21] presented IoTGuard, a dynamic, policy-based enforcement system for IoT. The purpose of IoTGuard is to protect users from unsafe and insecure device states by monitoring the behavior of trigger-action platform applications. It requires extra logic to be added to an application's source code to collect runtime information. Similarly to [53], that runtime information is used to create a dynamic model that represents the runtime execution behavior of the application. Lastly, it finally enforces relevant safety and security policies on the dynamic model of the individual application or sets of interacting applications.

Most of these solutions [53, 14, 21] are somewhat impractical for the real-world due to the fact that they require applications to be modified. Furthermore, the aforementioned solutions also require applications to be analyzed thoroughly before they are used, in order to discern their runtime execution behavior. However, they do provide an important first-step in addressing the problem of lack of data flow control in IoT environments.

Melara et al. [91] introduced an access control system that did not require the modification of applications. Because most IoT applications include third-party libraries that may contain vulnerabilities, the authors aimed to provide intra-process access control for applications, especially for those that generate and analyze highly privacy-sensitive data. The access control system, called Pyronia, enforces function-granular resource access policies via runtime and kernel modifications. Therefore, while Pyronia does not need to modify the applications themselves, each device looking to deploy Pyronia needs to run a custom Linux kernel.

3.3.3 Recently Proposed Authentication Mechanisms. While conducting this survey, we found that recently proposed authentication mechanisms mostly dealt with biometric factors. For example, many papers [67, 101, 83, 156, 29] developed unique touch-based authentication mechanisms for wearable or smart home devices. Alternatively, Lin et al. [86] presented a continuous authentication system based on geometric and non-volitional features of cardiac motion.

Of all the interesting authentication methods we found, the continuous authentication mechanism for voice assistants presented by Feng et al. [50] is the most relevant to our survey. The authors present VAuth, the only system that we found that provides continuous authentication for voice assistants. VAuth collects the body-surface vibrations of a user, and matches it with the speech signal received by the voice assistant's microphone. VAuth can fit inside things that people normally wear, such as eyeglasses, earbuds, and necklaces, Such a system can guarantee that the voice assistant only executes the commands that originate from the voices of authorized users. The authors evaluated the system on 18 users and 30 voice commands, and achieved a detection accuracy of 97% with less than 0.1% false positives, regardless of VAuth's position on the user's body, the user's language, the user's accent, or the user's mobility. This authentication mechanism would help prevent against the inaudible voice-command injection attacks.

3.3.4 Detecting Side-Channel Attacks. We discussed side-channel attacks in Chapter 3.2, and present a recent solution to this type of attack. Chaman et al. [28] analyzed stealthy eavesdroppers that employ passive receivers that only listen and never transmit any signals. The authors show that even passive receivers leak RF signals over the wireless medium. As a result, they were able to detect passive receivers present in a network, even when the leaked RF signals were buried under ongoing transmissions.

3.3.5 Detecting Routing Attacks. While in the past four decades there has been a plethora of papers focused on the detection and mitigating of routing attacks in wireless sensor networks (WSNs), we focus on a more recent IDS for 6LoWPAN routing attacks. Raza et al. [111] presented SVELTE which detected sinkhole attacks in 6LoWPAN networks that occur through RPL rank manipulation. It has two main modules running on the border router of a 6LoWPAN network: 6LoWPAN Mapper (6Mapper) that gathers information about the network, and an intrusion detection module that checks the rank inconsistency in data obtained by 6Mapper to detect sinkhole attacks. The 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 the 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 border router increases the network overhead. Every response from a device consumes more power. Worst of all, SVELTE has a dilemma in choosing the probing interval: a short interval leads to a low latency in detecting sinkhole attacks, but a large overhead due to frequent probing and responding; a long interval results in a low overhead, but a high latency in detecting sinkhole attacks.

3.3.6 Detecting Voice-Command Injection Attacks. He et al. [61] instead presented a detection and mitigation system to tackle voice-command injection attacks. Specifically, the system they presented detects inaudible voice commands by analyzing the frequency of voice commands picked up by the microphone. Once an inaudible voice command is detected, the authors apply active noise cancellation techniques to negate the frequency at which the inaudible command is carried.

3.3.7 Detecting Attacks via Encrypted Traffic Analysis.

Sensitive data can be leaked through traffic analysis, even if that traffic is encrypted. Zhang et al. [166] employed the same techniques used in traffic analysis attacks to create an anomaly detection system. The goal of the system, called HoMonit, is to detect misbehaving applications. Specifically, HoMonit leverages the leakage of packet size and inter-packet timing to infer device activities. The inferred activities are compared to the expected program logic of the application to identify anomalies. and then compares the inferred event sequences with the expected program logic of the application to identify misbehaviors (e.g., event spoofing). At the core of HoMonit is a Deterministic Finite Automaton (DFA) matching algorithm. The authors argue that every application follows a certain DFA model, in which each state represents the status of the application and the corresponding smart devices, and the transitions between states indicate interactions between the application and the devices. HoMonit includes techniques to extract the program logic from an application's source code, and automate the DFA construction process. HoMonit applies the DFA matching algorithm to compare the inferred device activities with the expected DFA transitions of each application. If the DFA matching fails, a misbehaving application is detected.

3.3.8 Detecting Hidden Inter-Application Interactions.

We previously discussed inter-rule vulnerabilities which are essentially the same as the inter-application interaction vulnerabilities that Ding et al. [45] analyzed. Unlike in traditional networks, in an IoT environment, an application can control a device to change the physical environment, which may in turn trigger another application to command another device to react to that change. Therefore, if an application is not aware of all of its possible interactions with other applications, unexpected interactions could be exploited and triggered by attackers. The authors proposed a framework called IoTMon that captures all potential physical interactions across applications, and allows users to create safe interaction controls. IoTMon performs intra-application interaction analysis using static program analysis and NLP to extract necessary application information, such as conditions, actions, and devices, for generating inter-application interaction chains. After identifying all interaction chains, IoTMon uses a risk analysis mechanism, similar to [148], to evaluate the risk of identified inter-application interaction chains. The authors evaluated their framework on 185 official SmartThings applications, and found that 162 hidden interaction chains exist among these applications, where 37 of them are highly risky and could be potentially exploited to launch serious attacks.

In the same vein, Wang et al. [149] argued that with the rise in popularity of home automation, devices and applications may be chained together in long sequences of trigger-action rules. These chains could be so complex that from an observable symptom (e.g., an unlocked door), it may be impossible to identify the distantly removed root cause (e.g., a malicious app). Thus, the authors presented ProvThings, an auditing system that performs efficient automated instrumentation of IoT apps and device APIs in order to capture complex chains of interdependencies between different apps and devices.

3.3.9 Improvements to 6LoWPAN Security. As we described in Chapter 3.1, 6LoWPAN enables the use of IP in IEEE 802.15.4 low power and lossy wireless networks, and it uses CoAP as its application-layer protocol, which sits on top of UDP with DTLS at the transport-layer. While DTLS supports a wide range of cryptographic primitives for peer authentication and payload protection, it was originally designed for networks where machine resources and message length were not critical constraints. Therefore, Raza et al. [110] proposed Lithe — CoAP with a compressed version of DTLS. They employed the same techniques used in 6LoWPAN to compress IP headers. Through DTLS header compression, the authors argued that they achieved energy efficiency by reducing the overall IP/UDP datagram size (due to the fact that transmitting packets requires more energy than computation). They also argued that they minimize 6LoWPAN fragmentation that is applied when the size of a datagram is larger than the link layer MTU (127 bytes for the case of IEEE 802.15.4), thereby making 6LoWPAN less vulnerable to fragmentation attacks. Similarly, Raza et al. [109] also proposed to use IPsec to secure communication between devices through IPsec compression. However, it is unclear from these papers ([110, 109]) if compressing the respective security protocols reduces the overall security of a network and opens it up to potential attacks. Furthermore, neither of the protocols presented in these papers have been standardized as of yet, and therefore, it is unlikely that they are being used in the real-world.

3.3.10 Recent Encryption Protocols. IoT applications often store and share data in the cloud, and therefore, sensitive data about users could be observed by cloud operators, or leaked if the cloud infrastructure is not appropriately secured. In order to tackle the problem of sensitive data leakage in the cloud, Shafagh et al. [121] designed a data protection framework, called Talos, where the cloud stores encrypted data while allowing applications to perform queries over the encypted data, without having to first download and decrypt the data. The proposed solution utilized crpytographic techniques, such as Partial Homomorphic Encryption (PHE) and order-preserving encryption, to allow computations to be carried out on encrypted data. Furthermore, the authors later extended Talos to not only be able to query encrypted data, but to also share it [120].

Due to the resource constrained nature of some IoT devices, and the resource consumption incurred by cryptographic primitives, Li et al. [80] proposed a key-free communication method for IoT networks, which they called HlcAuth. Essentially, HlcAuth utilized challenge-response mechanisms for mutual authentication between the gateways and smart devices without key management. Through real-world evaluation, the authors showed that HlcAuth can defend against replay attacks, message-forgery attacks, and man-in-the-middle attacks. However, for HlcAuth to work, the authors assumed that attackers are not within a certain range (at least 4.2 meters) of the gateway node.

3.3.11 Identification Through Traffic Analysis. The discovery and identification of IoT devices is a necessary first-step to monitor and protect those devices. Sivanathan et al. [133] analyzed the problem of classification of IoT devices based on their network traffic characteristics. The authors first instrumented a real-world IoT environment consisting of 28 devices. The devices included smart cameras, lights, plugs, motion sensors, appliances, and health-monitors. They then collected data from this environment for a period of 6 months, which they made available to the IoT research community. Next, they identified key statistical attributes such as activity cycles, port numbers, signaling patterns, and cipher suites, which gave them insights into the underlying network traffic characteristics of the IoT devices. The authors further developed a multi-stage machine learning-based classification algorithm, and demonstrated its ability to identify, with high accuracy, specific IoT devices based solely on their network behavior.

Alternatively, as Feng et al. [51] argued, manual device annotation impedes large-scale discovery, and device classification based on machine learning requires large training data with labels. Therefore, the authors proposed an Acquisitional Rule-Based Engine (ARE), which can automatically generate rules for discovering and annotating IoT devices without the need of any training data. ARE creates these rules by using application-layer responses from IoT devices and product descriptions from websites for device annotations. The authors define a transaction as a pair of textual units, consisting of the application-layer data of an IoT device, and the corresponding description of an IoT device from a website. To collect the transaction set, ARE uses the association algorithm to generate rules of IoT device annotations in the form of (type, vendor, and product). In testing ARE in the real-world for a 2-month period, the authors were able to discover nearly 2,000 compromised IoT devices among 12,928 IP addresses.

3.3.12 Traffic Shaping to Prevent Unauthorized Identification.

While identification is an important step in securing IoT devices, unauthorized analysis of IoT traffic for malicious purposes should be prevented. Apthorpe et al. [13], who demonstrated that network observers, such as ISPs, could infer privacy sensitive in-home activities by simply analyzing the traffic rates generated from encrypted traffic of smart home devices, argued that current defenses to such traffic analysis attacks, such as blocking traffic, or tunneling traffic through a virtual private network (VPN), are ineffective. Specifically, developers would be disincentivized to allow users to block all or some traffic from leaving the smart home, and while tunneling smart home traffic through a VPN would increase the attack difficulty, it would not provide the user with any actual proof of privacy protection. In fact, the authors showed that an adversary can still fingerprint devices using the rate of VPN traffic alone. Therefore, the authors suggested that traffic shaping is the only solution that can prevent the leak of rate information, and thus render such an attack impossible. By shaping traffic rates to match a predetermined rate or schedule, users can prevent exposing information about device behavior to an adversary. Unfortunately, slowing down devices' traffic rates to match the predetermined rate can have significant negative performance impacts (e.g., voice assistants not answering questions, devices losing connectivity to their mobile applications, etc.), and adding extra cover traffic to match the predetermined rate can incur significant data usage costs.

3.4 Challenges and Open Issues

While surveying the area of IoT security, we found several missing gaps and challenges yet to be sufficiently addressed, such as a lack of large-scale identification methods for insecure and compromised devices, lack of security protocol standardization, lack of sufficient regulatory frameworks, lack of user interest in or knowledge of security and privacy concerns in their homes, and lack of real-world deployment. However, from our perspective, there are four main challenges that are critical to IoT security, and need to be strongly considered in future work related to this area:

1. **Lack of concern for IoT-enabled DDoS**: Considering the recent rise of large-scale IoT-enabled DDoS attacks, we were surprised to find a lack of interest among the IoT security research community in studying this issue in more depth. We only found a few solutions, such as [19], that focused on detecting recent malware strains specifically targeting IoT devices. We suspect the research community is not interested in this topic because many assume that the large body of work dealing with worm propagation and

botnet detection for the traditional Internet can be applied directly to the IoT environment. However, we believe that the mesh networking capabilities, coupled with energy (and potentially memory) consumption constraints of IoT devices makes direct application somewhat unsuitable.

2. **Strong assumptions related to device and application modification**: Of the few distributed solutions we surveyed, most of them assume that devices or applications running on the devices can be modified. We argue that this is a strong assumption. Many of the devices already connected to the Internet are closed systems, thereby making modifications at any layer almost impossible. Furthermore, the location of IoT devices can also make modifications very difficult, and in some cases, infeasible (e.g., devices implanted in the body).

3. **Lack of concern for energy consumption**: We were surprised to find that almost none of the papers published in the last five years that we surveyed treated energy consumption as a primary constraint when designing their security solutions. This may be because we mainly surveyed papers related to smart home security, and the research community is less concerned with energy consumption in such networks. We disagree. While it is true that many smart home devices may be "plugged-in" and do not rely on a finite energy source (e.g., a battery), there are some devices, such as cameras, smoke detectors, and various wearables, that are not always plugged-in. Also, an ideal security solution should be able to be deployed in various different IoT environments, not just smart homes. Furthermore, a solution that requires a seemingly moderate amount of energy may not be a significant issue for an individual, plugged-in device, but if that solution were to scale to

the billions of IoT devices all over the world, its total energy consumption would be anything but moderate, which could arguably have a negative impact on our environment.

4. **Lack of distributed solutions**: Most of the papers we surveyed proposed defense mechanisms that were placed in a central location, most likely the gateway/border router of the IoT network. Unfortunately, due to the device-to-device communication and mesh networking capabilities of many IoT protocols (e.g., 6LoWPAN, ZigBee, Bluetooth Mesh), centralized solutions cannot capture a large portion of intra-network traffic, and therefore, may be useless against in-network attacks.

We attempt to address each of the aforementioned challenges in this thesis. To address the first challenge, we dedicate the rest of this thesis to tackling the problem of IoT-enabled DDoS attacks. Specifically, in Chapter IV, we present a solution for preventing IoT devices from being infected with worms that can turn them into DDoS bots. In Chapter V, we present a solution for detecting DDoS traffic leaving an IoT gateway. Finally, in Chapter VI, we present a solution for mitigating DDoS traffic originating from an IoT network. To address the second challenge, all of the solutions we present require little to no modification of individual IoT devices. To address the third challenge, a primary focus for the solutions presented in Chapter V and Chapter VI is on limiting resource consumption incurred by IoT devices and networks. Lastly, to address the fourth challenge, the solutions presented in Chapter V and Chapter VI are distributed security systems that are able to handle the mesh networking capabilities of IoT networks.

CHAPTER IV
DEFENSE AT AN UPSTREAM ISP/IXP: PREVENTING IOT DEVICES FROM TURNING INTO DDOS BOTS

In this chapter, we focus on defense at an upstream ISP or IXP to the protected IoT network. The first step in launching IoT-enabled DDoS attacks is to take control of IoT devices by infecting them with a worm, and therefore to detect such worm infections, we present a worm detector called SWORD.

While SWORD can detect both incoming and outgoing worm traffic, we analyze inbound worm detection, or worm traffic originating from the Internet to the protected network. Running SWORD at an ISP/IXP can be more cost efficient compared to running SWORD at each network downstream. However, the drawback to running SWORD upstream is that detection is far more difficult, especially when only partial incoming traffic is observed.

We use a real-world Mirai traffic trace collected at an Internet Exchange Provider (IXP) in September 2016 to evaluate SWORD's performance at detecting incoming worm traffic. SWORD consistently outperforms all other detectors in terms of total number of Mirai IPs detected, false negative rates, and detection latency. We also find that SWORD outperforms every other detector at detecting surreptitious worm IPs with relatively low total scans or low incoming scanning rates, and superspreading worm IPs with relatively high total scans.

The chapter is derived in part from the following unpublished work: On the Detection of Smart, Self-Propagating Internet Worms by Li, J.; Sisodia, D.; Stafford, S. Note, this unpublished work was itself derived from the following published work: Detecting Smart, Self-Propagating Internet Worms [81] by Li, J.; Stafford, S. The content of this chapter is focused on inbound worm detection, of *which I am the primary contributor, and was responsible for conducting all of the presented analyses.*

4.1 Introduction

Worms can infect millions of hosts on the Internet in just several minutes and continue to pose a severe threat to the security of the Internet. In fact, the ground for worms to spread is potentially more fertile than ever. The number of Internet-capable devices continues to rise at a stunning rate [130], and each of these devices is capable of running a diverse range of user-installable software that can be vulnerable to malicious attacks.

While there was a relatively long lull without much worm activity between the Morris worm in 1988 and the big wave of many devastating worms in the late 1990s and early 2000s (e.g.Trinoo, Tribe Flood Network, Code Red, Nimda, SQL Slammer) [82], worm activity has continued in the last two decades [150]. For example, in 2008, Conficker [150] infected over a million machines, and between 2010 and 2012, Stuxnet/Duqu/Flame [33] caused devastating damage to several industrial and energy-producing facilities in several countries. More recently, the world witnessed BASHLITE [34] in 2015 which caused massive spikes in distributed denial-of-service (DDoS) attacks, along with WannaCry and NotPetya in 2017 which caused catastrophic ransomware attacks all over the world [33].

One of the widest spreading worms in recent memory is the Mirai worm. In 2016, the Mirai worm infected over 300,000 Internet of Things (IoT) and embedded devices all over the world, and thus formed a botnet that was used to launch multiple devastating DDoS attacks against companies such as OVH and Dyn [12]. According to the 2020 NETSCOUT Threat Intelligence Report, “Malware king” Mirai continues to “thrive” on the Internet, especially in IoT-rich environments [66]. Furthermore, due to its propagation success, Mirai's scanning techniques have been copied by a plethora of newer worms such as Hajime, IoTroop, and Mozi, among others [11].

All of the existing worm detectors we studied [58, 119, 35, 73, 117] focus solely on detecting outgoing worm traffic from a protected network to the Internet, leaving their efficacy at detecting incoming worm traffic from the Internet to a protected network unknown. Even if a detector is effective against outbound worms, we cannot assume it will be effective against inbound worms. Inbound worm detection is significantly more difficult than outbound worm detection. When detecting outbound worms, a detector can be placed to observe all of the outgoing scans from an infected network. However, when detecting inbound worms, because the scans from every infected host usually target victims all over Internet, no matter where a detector is placed, it may only observe a miniscule portion of all worm scans, making it significantly harder to detect the inbound worms.

We revisit worm detection in this chapter. We treat worm detection as an arms race in which a worm can be smart and evasive, and propose a new worm detector called **SWORD** (Self-propagating Worm Observation and Rapid Detection). Unlike most existing detectors, SWORD is focused on the *fundamental* behavior of worms that is hard for any worm to evade. The only truly fundamental behavior of worms is that of connecting to new destinations. Behavior-based detection systems that do not focus on this fundamental behavior can be evaded successfully by sufficiently smart worms. Furthermore, SWORD is designed to detect both outbound worms from, and inbound worms toward a protected network. It only needs to observe a very small number of connections from an infected host to detect the presence of the worm, either outbound or inbound. In addition, unlike some detectors that require bi-directional traffic to detect worms in either direction, SWORD only requires outgoing traffic to detect outbound worms and incoming traffic to detect inbound worms.

SWORD's working mechanisms are novel. It includes two main modules in detecting violations that a worm will cause in connecting to new destinations, and these two modules complement each other: If a worm does not wish to violate one module when connecting to somewhere, it will inevitably violate the other, leaving little space for a worm to breathe and forcing it to slow down or freeze.

We evaluate SWORD and three detectors, MRW, PGD, and RBS, on incoming worm traffic, where all of the detectors are only given uni-directional incoming traffic as if they were deployed at an upstream ISP or IXP to protect multiple downstream networks. We exclude DSC, TRW, and TRWRBS from comparison because they require bi-directional traffic to detect worms. We leveraged a real-world, Mirai worm traffic trace collected at a major educational IXP [3], along with a trace of background traffic collected at the same IXP [2]. Not only does SWORD outperform every other detector in terms of false negative rates, but it also outperforms them at detecting *surreptitious* worm IPs with low total scans or low incoming scanning rates, and *superspreading* worm IPs with high total scans. Compared to its competitors, SWORD detects significantly more worm IPs that make as low as 5 incoming scans, and unlike any of its competitors, SWORD can detect worm IPs with incoming scanning rates as low as 0.002 scans/s. Furthermore, SWORD detects the first incoming Mirai worm scan the quickest, therefore allowing the fewest incoming worm scans out of all the detectors before alerting of the first worm scan.

4.2 Background and Related Work

4.2.1 Worm Traffic Detection.

A worm running on a host actively scans the network (or the entire Internet) that the host is connected to and looks for new victims to infect. A worm can employ a variety of scanning mechanisms, including random, local preference, sequential, permutation, topological, and hitlist scanning [139]. It infects a remote host by gaining sufficient privileges to copy itself to, and then execute itself on, the remote host.

We categorize worm detection systems into two categories: host-based and network-based. Host-based detection uses information available at the end-host, and example techniques include buffer overflow detection, correlating network data to memory errors, and looking for patterns in system calls (e.g.[85, 38, 100]). But since host-based detection requires deployment on every host to detect if a host is infected, network-based detection became more desirable with less overhead to install and maintain. Network-based systems usually only need a single deployment location, such as a network gateway, to protect an entire network. Network-based detection mainly includes content-based detection and behavior-based detection. Content-based detection observes the content of network traffic to look for byte patterns that match the signature of a worm. Early content-based detectors leveraged simple statistical methods (e.g.[77, 128, 84]), while recent content-based detectors leverage deep learning to detect worms (e.g.[168, 135]). Behavior-based detection observes the network behavior of end hosts and identifies patterns that are indicative of the presence of a worm. Because content-based detection is less capable against zero-day or polymorphic worms and can incur a high overhead to inspect traffic payload, we focus on behavior-based detection in this chapter.

Existing behavior-based worm detection has focused on various types of traffic behaviors, including: how the outgoing connections from a host correlate to the incoming connections to that host, how the connection failure patterns of a host deviate from normal, and what a host's pattern of visiting destinations looks like. As we will need to compare SWORD against state-of-the-art behavior-based worm detectors, we now summarize these detectors below.

DSC [58] detects a worm by correlating an incoming connection on a given port with subsequent outgoing connections on that port. If the outgoing connection rate exceeds a threshold established during training, the alarm is raised.

TRW [117] identifies a host as worm infected if its attempts to connect to new destinations result in a high rate of connection failures. The basic idea is that a worm-infected host that is scanning the network randomly will have a higher connection failure rate than a host engaged in legitimate operations.

The multi-resolution approach [119], which we refer to as MRW, supposes that when there is no worm, the growth curve of the number of distinct destinations over time is concave, but not so when a worm is present since worm scanning will lead to many destinations. This hypothesis can be leveraged by monitoring over multiple time windows with different thresholds for each window. If the number of new destinations for a host within a given window exceeds the threshold, an alarm is raised.

The Protocol Graph detector [35], which we refer to as PGD, is targeted at detecting slowly propagating hitlist or topologically aware worms. It works by building protocol-specific graphs where each node in the graph is a host, and each edge represents a connection between two hosts over a specific protocol. It assumes that during legitimate operation over short time periods, the number of hosts in the graphs is normally distributed and the number of nodes in the largest connected component of each graph is also normally distributed. During a worm infection, however, both numbers will become abnormal, thus indicating the presence of a worm.

RBS [73] measures the rate of connections to new destinations, similar to MRW. It assumes that a worm-infected host contacts new destinations at a higher rate than a legitimate host does. It measures this rate by fitting the inter-arrival time of new destinations to an exponential distribution.

TRWRBS [73] combines the TRW and RBS detectors into a unified scheme, and observes both the connection failure rate and the first contact rate. It performs sequential hypothesis testing on the combined likelihood ratio to detect worms.

All aforementioned detectors are focused primarily on classic worms, without considering the countermeasures that a smart, evasive worm may employ. Research in [138] have also evaluated and compared their performance, but only against classic worms. Although a detector may appear to perform well by only considering classic worms, their performance against sophisticated evasive worms remains to be seen.

Furthermore, all aforementioned detectors are focused solely on detecting outbound worms from a network. While each detector may be applied to also detect inbound worms toward a network without significant changes, each detector was evaluated only against outbound worms.

Lastly, DSC, TRW, and TRWRBS require bi-directional traffic to detect worms. DSC needs to observe both incoming and outgoing traffic in order to correlate the two, TRW needs to observe incoming traffic in order to determine which outgoing connections led to connection failures, and TRWRBS leverages

TRW and therefore also needs to observe incoming traffic. In some scenarios in the real world, the requirement of bi-directional traffic may be impossible to meet, due to the deployment location of a detector, hardware limitations, privacy considerations, and other constraints.

4.2.2 Content-Agnostic Traffic Analysis. The aforementioned behavioral-based worm detectors, including SWORD, are content-agnostic because they do not need to observe the content of the network traffic. There is a plethora of content-agnostic traffic anomaly detection approaches in literature, especially in two areas related to worm detection: bot detection and DDoS detection.

While somewhat similar, bot detection differs from worm detection in that typical bot detection approaches attempt to detect communication between bots and their C&C servers, instead of detecting scanning behavior of the bots. Content-agnostic bot detection approaches leverage flow-level information, instead of deep packet inspection, to identify key features of C&C communication and develop detection frameworks (e.g. [57, 17]). In many cases, worms are used to create botnets that ultimately launch DDoS attacks. Over the last couple decades, content-agnostic DDoS detection has been a fertile ground for network security research (e.g. [46, 106]).

Some of the same shortcomings that apply to the previously investigated behavioral-based worm detectors also apply to the content-agnostic anomaly detection approaches in bot and DDoS detection. All of the investigated approaches require a relatively comprehensive view of the network in which they are deployed, which includes being able to observe bi-directional traffic. While content-agnostic approaches claim to be easier to deploy in the real-world due to not needing

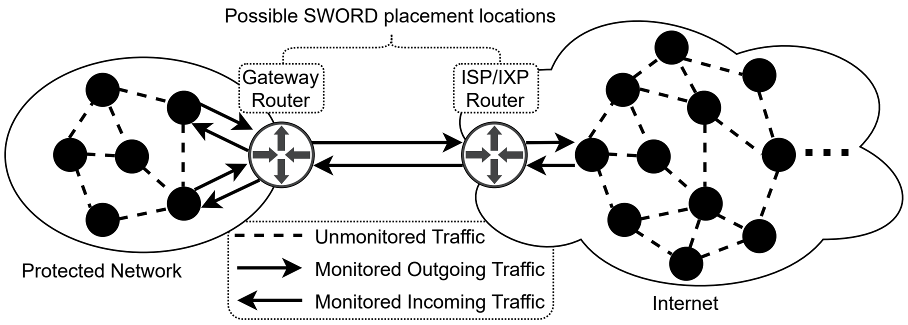

Possible SWORD placement locations
Gateway Router
ISP/IXP Router
Protected Network
Internet
Unmonitored Traffic
Monitored Outgoing Traffic
Monitored Incoming Traffic

Figure 9. Placement of the SWORD detector.

network traffic content, the requirement of comprehensive input data may render them infeasible for many networks on the Internet.

4.3 The SWORD Detector

4.3.1 Placement of SWORD. SWORD can detect both outgoing and incoming worm traffic from and toward a protected network. In other words, to detect outgoing worm traffic, SWORD needs to be placed where it can monitor the network's outgoing traffic; *no* incoming traffic is needed. To detect incoming worm traffic, it needs to be placed where it can monitor the network's incoming traffic; *no* outgoing traffic is needed.

A SWORD detector can be placed either at or away from a protected network. As depicted in Figure 9, a typical deployment position of SWORD is the gateway of a protected network where SWORD can monitor all of the outgoing and incoming traffic. Alternatively, SWORD could also run at an ISP/IXP that is *en route* of the outgoing traffic from and/or incoming traffic to the network. If the network is single-homed (i.e.connected to the Internet with just one ISP/IXP) and SWORD is deployed at its direct ISP/IXP, SWORD can monitor all the outgoing and incoming traffic. However, if the network is multihomed (i.e.connected to the Internet with more than one ISP/IXP), SWORD may not see all of the outgoing or incoming traffic and thus only detect outgoing or incoming worm traffic traversing the ISP/IXP where it is installed. This is also true if SWORD is deployed at an ISP/IXP multiple hops away from the protected network (regardless if the network is single-homed or multihomed). A distributed version of SWORD that runs multiple instances of SWORD at more than one location can also be deployed to collectively monitor all the outgoing/incoming traffic; in this work we focus on the single-instance version of SWORD.

To monitor outbound worms departing from a protected network, it is often preferable to deploy SWORD at a location where it can observe *all* the outgoing traffic, such as the network gateway or its direct ISP/IXP if the network is single-homed, so that the network can minimize its liability of leaking worms to the Internet. Moreover, if SWORD is deployed at a location not able to monitor all the outgoing traffic, the worms that have infected the network may also learn the location of SWORD, and bypass it such that the protected network may not even know the presence of the worm.

To monitor inbound worms toward a protected network, SWORD can also be deployed at the network gateway or its direct ISP/IXP if the network is single-homed such that it can see all the incoming traffic to maximize the detection of all worm traffic. However, as inbound worms can travel toward the network along multiple incoming paths, even if SWORD is only deployed on one of the incoming paths, such as when SWORD is deployed at one of the protected network's ISP/IXPs, SWORD will still be able to detect the presence of the worm, for two reasons: First, even though in such a deployment scenario SWORD may only see incoming traffic to the network, SWORD only needs uni-directional traffic to detect worms. Second, although SWORD is not deployed on every incoming path, worm traffic usually appears on every incoming path toward the network. Running SWORD at an ISP/IXP can be more cost efficient compared to running SWORD at each network downstream, but far more difficult, especially when only partial incoming traffic is observed. ISPs/IXPs may also be interested in running such a worm detection service, that can act as an early warning system, in order to advertise to potential customers that it has the ability to detect the presence of worm traffic towards its customers. Note, if SWORD is effective at detecting the presence of worm traffic upstream, it will also be just as effective, if not more so, if deployed at an IoT network's gateway router.

In this dissertation, we focus solely on inbound worm detection. Below we describe SWORD's two complimentary detection modules, BDD and QPD, in detail.

4.3.2 BDD: Preventing Fast Scanning via the Burst Duration Detector. The behavior of contacting new destinations seeking new victims to infect simply cannot be avoided by a worm that is looking to propagate. So, to detect a worm one should look for anomalies in the rate at which a host contacts new destinations, i.e.the rate of *first-contact connections*. The key is then to determine whether or not a host is making first-contact connections at a rate faster than usual.

Two previous detectors relied on heuristic of this flavor: the MRW detector and the RBS detector. However, they both have their deficiencies. The MRW detector counts the number of first-contact connections in a series of time windows of different length, but it only uses a relatively small set of windows, typically fewer than 10. An intermediate window size might produce a detection window that would detect a worm more quickly than the bigger or smaller sizes in use, but due to the limited number of windows, MRW cannot take advantage of this. RBS, on the other hand, computes a threshold for every different window size, and it uses the number of connections instead of time to describe the window. However, it suffers from sub-optimal thresholds, and thus a poor performance even against classic worms. RBS attempts to fit a single curve to the distribution of inter-connection intervals and uses this curve to generate the thresholds, but in practice the distribution does not map well to a single curve.

BDD avoids the drawbacks in the MRW and RBS approaches. Rather than using a small number of time windows of different sizes like MRW, it uses RBS's method of creating a window for every different size of connection burst. Moreover, BDD derives a threshold for every burst size (from a two-connection burst size to a maximum-connection burst size). We introduce a training process, during which we measure multiple different durations observed for each burst size and use the minimum duration observed for each burst size to determine the threshold for a burst of that size. Different from RBS using a single curve to derive thresholds, this process allows for a more complex distribution of inter-connection interval times for connections in a burst, thus obtaining more accurate thresholds. As a result, BDD has the advantages of supporting a large number of window sizes and obtaining an accurate threshold for every burst of a different size.

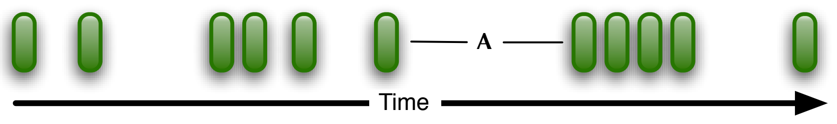
Another advantage of BDD is that even if a worm only makes a small number of connections, these connections will be verified against the thresholds for bursts of a small size, and if any threshold is violated, BDD can detect the worm. This advantage is especially useful in detecting inbound worm traffic from a worm-infected host toward a protected network, since the host, while scanning everywhere on the Internet, may only launch a small number of worm connections toward the network.

The potential drawback to this new method is greater overhead for storing different thresholds and greater computational requirements for examining a recent connection history to determine if it violates any of the thresholds. However, a truism is that computational power and storage space are constantly increasing, and this additional load is of a less concern.

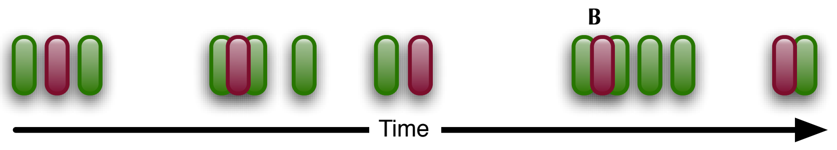
4.3.3 QPD: Ensuring Quiescent Periods via the Quiescent Period Detector. A normal host will exhibit regular *quiescent* periods where it does not make any first-contact connections. In other words, legitimate traffic is typically bursty, with first-contact connections occurring in groups and quiet periods between them. Figure 10a shows an example pattern of legitimate connections. Point **A** in the figure shows a quiescent period with no worm traffic, followed by a burst of connections.

After a worm infects a host and tries to spread itself, if it scans at a fixed rate, it will make connections during the middle of a legitimate burst, which will raise the overall observed connection rate from the host. Figure 10b shows the legitimate traffic with the addition of classic worm traffic. Point **B** indicates a spot of increased connection rate due to the worm connections adding to the burst of legitimate traffic. If BDD is in place, it can detect the worm.

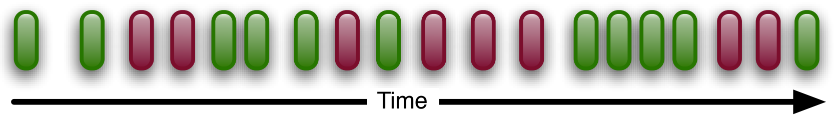
The worm, however, could be adaptive and avoid this additive effect. Specifically, the worm can dynamically adjust its first-contact rate so that it is always lower than the detection threshold. If the host makes bursts of legitimate first-contact connections, the worm can simply slow down to keep from adding too many of its connections to the legitimate connections, thus avoiding exceeding



(a) Legitimate connections



(b) Legitimate connections plus classic worm connections



(c) Legitimate connections plus rate-adaptive worm connections

Figure 10. Examples of observed connections over time.

the detection threshold. When the host is otherwise idle, however, as long as the worm does not exceed the BDD thresholds, it is then free to make first-contact connections. Figure 10c shows the legitimate traffic with an adaptive worm overlaid. By scanning mostly when the host is in the middle of a quiescent period, the adaptive worm avoids having a scanning rate greater than the legitimate traffic, even at a higher scanning rate than the classic worm (with eight worm scans instead of five in Figure 10b).

Preventing or limiting this adaptive behavior of worms would then help to reduce the achievable scan rate of a worm, and is the basis for QPD. Basically, if a host does not display quiescent periods as it typically does, and has been "active" for overly long, QPD then determines that the host is infected by a worm that is scanning the network.

QPD thus detects worms by measuring the duration of active periods during a training phase. An active period is defined as the duration of a period during which first-contact connections happen with *no more than* the specified quiescent period between them. QPD uses a series of different quiescent periods. For every quiescent period size, it measures the mean and standard deviation of all the active periods that are separated by a quiescent duration of at least that length. These values are used to generate a threshold duration for active periods, which is the mean plus β times the standard deviation. β can be tweaked for different environments to fix the false positives at a specific value. If a host has an active period exceeding the threshold duration for any of the quiescent period, it is likely infected with a worm. For example, we can apply QPD to Figure 10c where the host is active all the time and does not exhibit any quiescent period at all to detect the presence of the worm.

Note that, similar to BDD, QPD also has the advantage that it is sensitive to worms that only make a small number of connections, which, again, is particularly useful in detecting incoming worm traffic despite that there may be only a small number of scans from a worm IP. Among the different quiescent periods QPD uses, some of them can be extremely short, and the active period based on a short quiescent period will also be short and contain only a small number of connections. Therefore, even if a worm only makes a small number of connections, it could cause certain active periods to exceed their threshold values, causing QPD to detect the worm.

4.3.4 Clustering.

Existing behavior-based detection systems employ the same threshold for all hosts in a protected network or on the Internet. This is a poor choice because hosts show widely divergent behaviors. As more devices (e.g.IoT devices) connect to the Internet, they also come with even more divergent behaviors [10]. Desktop computers used primarily for web surfing make connections in a different pattern than a department email server would, for example. If a desktop computer started making connections at the same rate as the email server, it is likely an anomalous event and something strange must have happened to that computer. But if the desktop computer applies same thresholds as the email server does, its behavior would not appear to be anomalous because those thresholds must allow it as normal behavior to avoid constantly flagging the email server as infected.

We applied existing clustering techniques to automatically categorize the hosts such that different thresholds can be applied to different groups of hosts. We examined a range of clustering techniques, behavior characteristics to cluster against, and number of clusters to create. We have found that using k-means clustering to separate the hosts into groups allows us to improve overall performance. In our current design we cluster based on a single feature of the hosts, the number of destinations contacted during a training period.

4.3.5 Design of SWORD.

We have combined the above principles into a new worm detector, i.e.SWORD. It uses the BDD and QPD detectors outlined above, and declares a host to be infected with a worm when either BDD or QPD raises an alarm. SWORD observes legitimate network activity for a period of time to cluster hosts into groups and generate thresholds for each cluster.

The co-existence of BDD and QPD makes it extremely hard for a worm to avoid being caught. If a worm wants to escape BDD but still makes new connections, it cannot shorten the duration of a burst of any size; it will then have to lengthen active periods, but doing so will get it caught by QPD. On the other hand, if a worm wishes to escape QPD while still making new contacts, it then has to ensure the quiescent periods; it will then have to insert its connections into active periods, which however will cause certain connection bursts to have a shorter duration than permitted, thus triggering the alarm from BDD. Therefore, this combined, collective detection of SWORD captures the fundamental behavior of worm detection, preventing a worm from quickly spreading to many destinations.

4.4 Experiment Methodology for Inbound Worm Detection

4.4.1 Procedure Overview. Our objective is to evaluate and fairly compare the performance of different detectors, including SWORD, against inbound worms. We chose to place each detector at a protected network's upstream ISP/IXP, a challenging deployment scenario where the detector is not guaranteed to be able to access all the incoming traffic, to study how effective SWORD and the other detectors are under such a realistic scenario. Moreover, such placement does not guarantee a detector to be able to access both incoming and outgoing traffic of a protected network either, a requirement for DSC, TRW, and TRWRBS to detect worms, we thus exclude them from comparison for inbound worm detection. The incoming traffic at the ISP/IXP is composed of (1) background traffic that we collected at a major IXP called FRGP [2] that does not contain worm traffic, and (2) the real-world Mirai worm traffic also collected at FRGP [3].

Different from the evaluation against outbound worms where we evaluated both classic worms and evasive worms, we do not assume inbound worms can be evasive. While an outbound worm, as it originates from a network already compromised, may have the knowledge of the legitimate traffic of the network and/or the parameters of the detector in place to become evasive, we do not expect the same knowledge for inbound worms.

For each worm detector, we first train it on 1-hour worth of background traffic. Specifically, we set the threshold values of each detector as aggressive as possible *without* triggering a false positive during the 1-hour period. Then on a different 1-hour period of background traffic that is mixed with worm traffic, we test each detector and measure its accuracy (total number of detected worm IPs, false positive rate, and false negative rates) and latency.

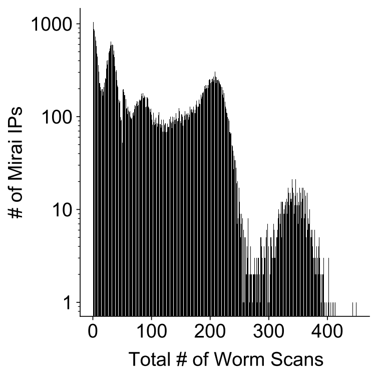
4.4.2 The Mirai Worm. We use the Mirai worm as a case study for this evaluation. As Mirai's code will continue to be the basis of future worms [11], the detection of Mirai traffic should be considered a baseline for worm detectors striving to achieve success in today's Internet. Hosts infected by the Mirai worm are diverse: Some only make a very small number of scans during a period or scan with an extremely slow scanning rate, which we call running a **surreptitious** worm; some scan a large number of targets, which we call running a **superspreading** worm. The ability to detect surreptitious worms potentially can alert a protected network of a worm infection otherwise unnoticed, while superspreading worms are clearly too dangerous not to detect.

4.4.3 Metrics and Parameters. The key metrics we will focus on in this evaluation are **number of Mirai IPs detected**, **false positive rate**, **false negative rate**, and **detection latency**. We denote TP as the number of Mirai IPs correctly detected as Mirai, FP as the number of legitimate IPs incorrectly detected as Mirai, TN as the number of legitimate IPs correctly detected as legitimate, and FN as the number of Mirai IPs incorrectly detected as legitimate. Then the number of Mirai IPs detected is $TP + FP$ and we define the false positive rate as $\frac{FP}{FP+TN}$ and false negative rate as $\frac{FN}{FN+TP}$. The detection latency is in terms of the number of worm scans allowed before detection. A Mirai IP's detection latency is thus the number of incoming scans the Mirai IP made before it is detected. On the other hand, the Mirai worm is detected when detecting the first Mirai IP, so the detection latency of the Mirai worm is the total number of all incoming worm scans made by all the Mirai IPs before detecting the first Mirai IP.

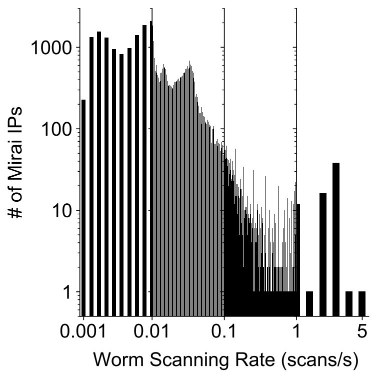
We also define two parameters associated with each Mirai IP: W that is the number of incoming worm scans made by the Mirai IP during the testing period and R that is the worm scanning rate of the Mirai IP, which is particularly useful when we inspect slowly scanning, surreptitious Mirai IPs. If a Mirai IP is detected, say after making m worm scans, assuming its first scan is at time t_1, the m-th scan is at time t_m, $R = \frac{m}{t_m - t_1}$; otherwise $R = \frac{W}{t_W - t_1}$ where t_W is the time of the last scan from the Mirai IP.

4.4.4 Evaluation Environment. The Mirai traffic trace are Argus flow records and was collected at FRGP during four days in September of 2016 (8th, 9th, 10th, 12th) [3]. These days coincide with Mirai's first major growth phase in early September 2016 [12]. The background traffic was also collected at FRGP in September 2016, across the entire month [2]. Both are uni-directional traffic collected at a single router at FRGP towards its downstream customers; as such traffic is not guaranteed to be *all* the incoming traffic toward a customer, it creates a more challenging deployment scenario.

We trained the detectors on 1-hour worth of FRGP background traffic (∼3 GB of records) collected on 9/8/2016 at 6:20 PM—7:20 PM (MST). We verified



(a) With respect to the total number of worm scans (W)



(b) With respect to the worm scanning rate (R)

Figure 11. Number of Mirai IPs in the 1-hour testing period.

that no incoming Mirai connections are present during this time window (there were no incoming connections to destination ports 23/2323 from random source ports). Once trained, the detectors were tested on 1-hour worth of Mirai scanning traffic mixed with FRGP background traffic (totaling ∼8 GB), both collected at FRGP on 9/9/2016 at 6:20 PM—7:20 PM (MST).

There was a total of 45,291 unique Mirai IPs present in the 1-hour testing period, along with 45,903 unique legitimate source IPs making incoming connections to FRGP customer IPs. Figures 11a and 11b show the total number of Mirai IPs (log scale) for different W and R values in the 1-hour testing period, respectively. From Figure 11a we can see that a large portion of the Mirai IPs made fewer than 50 connections to FRGP customer IPs. Detecting such IPs is almost impossible for any detector. Figure 11b shows a large number of worm IPs had extremely slow scanning rates between 0.001 and 0.030 incoming scans per second, also almost impossible to detect.

Table 1. Total number of Mirai IPs detected by each detector.

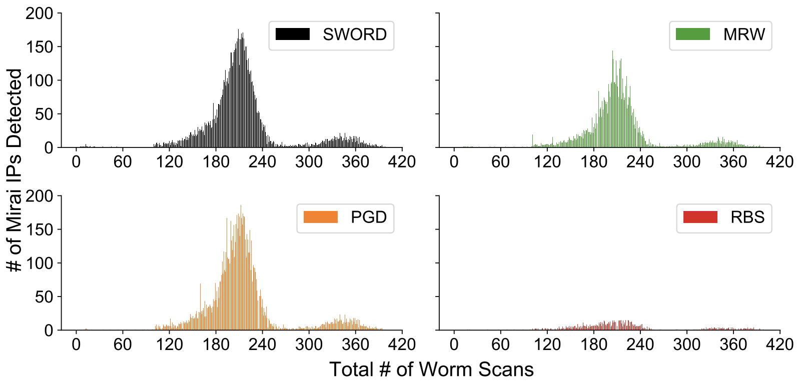
Detector	Mirai IPs Detected
SWORD	8882
MRW	5797
PGD	9100
RBS	1041

4.5 Inbound Worm Detection Evaluation

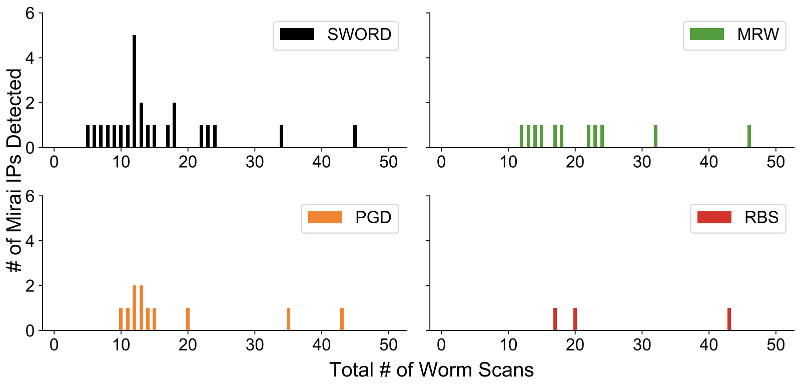
4.5.1 Total Number of Mirai IPs Detected. Table 1 shows the total number of Mirai IPs detected by each detector, By setting the threshold values of each detector *without* triggering a false positive during training, we found during testing that every detector's false positive rate is also 0, and therefore, all detections are true positives. PGD detected slightly more Mirai IPs than SWORD, while MRW and RBS detected around 35% and 88% less Mirai IPs than SWORD, respectively.

Figure 12a provides us with a more detailed view of the number of Mirai IPs detected by grouping the Mirai IPs based on their number of scans, i.e.their W values. First, for all of the detectors, most Mirai IPs that were detected made anywhere from around 120 to 250 scans (120≤W≤250). Second, there is a dip in detections between around 250 and 300 scans (250≤W≤300); this is because there was only a small number of Mirai IPs (less than 10) that fell within this range, as shown in Figure 11a. Third, although there were many Mirai IPs whose number of scans were relatively low with $W \leq 120$, it was difficult to detect them; without enough incoming Mirai traffic, a detector has difficulty in distinguishing between incoming legitimate and malicious connections.

Nonetheless, if we take a closer look at surreptitious Mirai IPs with a low number of incoming scans (e.g., no more than 50 scans or $W \leq 50$), as shown



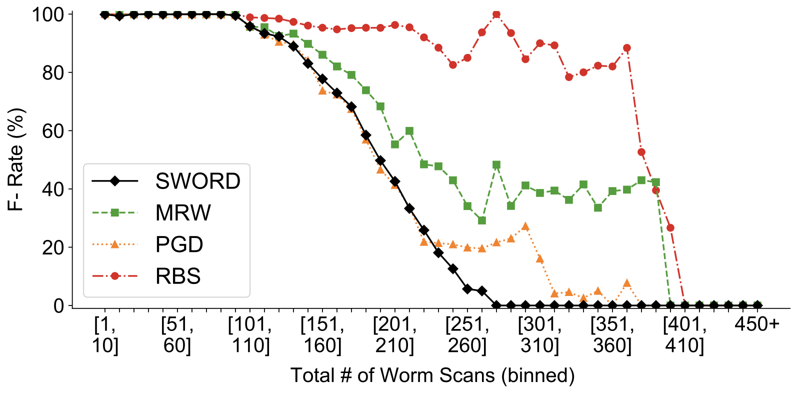
(a) Number of Mirai IPs detected with respect to W



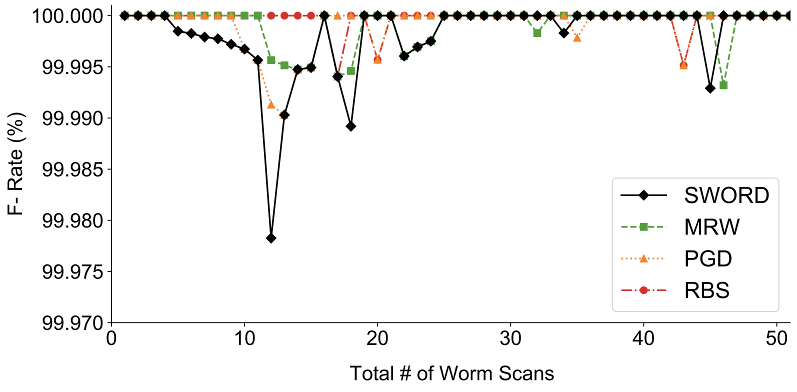
(b) Number of Mirai IPs detected with respect to $W \leq 50$

Figure 12. Number of Mirai IPs detected.

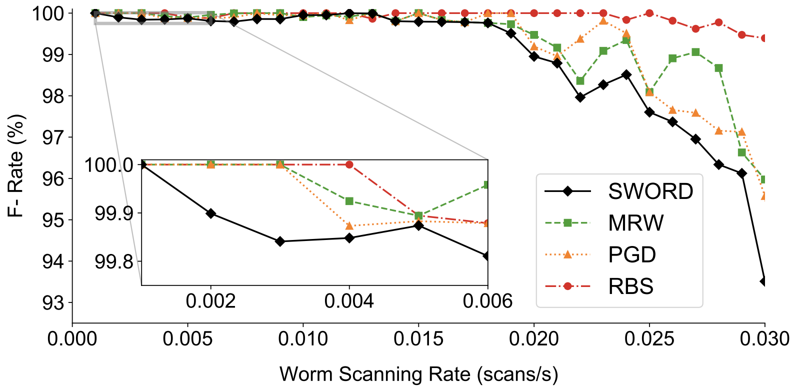
in Figure 12b, we see that SWORD significantly outperformed the other three detectors, and shows that it has the ability to detect some worm IPs with very low number of scans. While other detectors did not detect any Mirai IPs making less than 10 scans in the entire 1-hour period, SWORD was even able to detect a Mirai IP that only made 5 scans in the period. The main reason for SWORD's superior performance here is that even with an extremely low number of scans, some surreptitious worm IPs will exhibit bursty behavior, and will therefore be caught by SWORD's BDD module.



(a) False negative rates with respect to W



(b) False negative rates with respect to $W \leq 50$



(c) False negative rates with respect to $R \leq 0.03$

Figure 13. False negative rates.

4.5.2 False Negative Rate. Figure 13a shows a granular view of the false negative rates for each detector. We first binned Mirai IPs based on their total number of scans, i.e.their W values, at 10-scan increments and then calculated every detector's false negative rate for each bin. Overall, SWORD outperformed MRW, PGD, and RBS. Specifically, while SWORD's false negative rates were lower than MRW and RBS over all the Mirai IPs, PGD had a slightly lower false negative rate than SWORD for Mirai IPs that made from 121 to 240 total scans, which further led PGD to detect slightly more Mirai IPs than SWORD over all Mirai IPs. However, PGD was not as effective as SWORD against Mirai IPs that were superspreading or surreptitious, which are of particular importance for worm detection.

For superspreading Mirai IPs, which are clearly damaging when left undetected, we can observe at which point a detector reached 0% false negative rate to gauge its capability in detecting them. From Figure 13a, we can see for Mirai IPs with more than 240 scans, PGD remained at about a 20% false negative rate, while SWORD approaches 0%. SWORD reached a 0% false negative rate when Mirai IPs made more than 270 scans. PGD, MRW, and RBS, on the other hand, did not reach a 0% false negative rate until 371, 391, and 401 scans, respectively, allowing Mirai IPs 100, 120, and 130 more scans than SWORD, respectively, before they were guaranteed to be detected.

For surreptitious Mirai IPs, we look at every detector's performance against them in two complementary measurements. We first view every detector's false negative rates against Mirai IPs that made no more than 50 scans during the entire 1-hour testing period. Figure 13b shows for the most part SWORD's false negative rates were clearly lower than those of every other detector compared. Note that although the false negative rates here were fairly high, so long as they were not 100%, given the false positive rates were 0%, even if only one Mirai IP was detected, a reliable early warning could be issued against the worm. On the other hand, we also notice the false negative rate curves oscillated as the number of scans increased. This is because a surreptitious worm IP that made less scans than the other surreptitious worm IPs could have just scanned within a smaller time window, thus achieving a higher scanning rate and potentially a lower false negative rate. We therefore also look at every detector's performance against Mirai IPs using every worm's scanning rate. Figure 13c shows every detector's false negative rates against surreptitious Mirai IPs at different scanning rates between 0.001 and 0.030 scans/s. Now, the false negative rate curve decreases as the worm scanning rates increase. Again, clearly SWORD outperformed all of the other detectors at detecting surreptitious worms. For example, SWORD was even able to detect at least some Mirai IPs with scanning rates as low as 0.002 scans/s, while MRW, PGD, and RBS did not detect any worms with scanning rates less than 0.004, 0.004, and 0.005 scans/s, respectively.

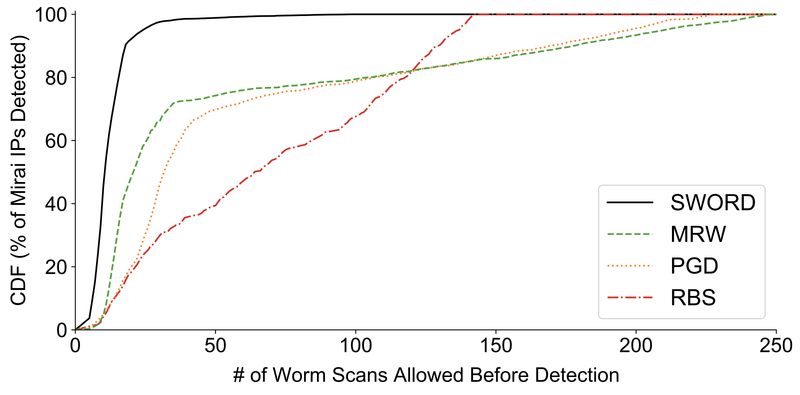
4.5.3 Detection Latency. Table 2 shows for each detector when it detects the first worm scan and the number of unique worm IPs allowed to scan the protected network before the detection. It also shows the total number of worm scans that occurred before the detection, which is the detection latency of a worm detector. For the given 1-hour testing period, SWORD detected the first worm scan more than twice as fast as PGD and MRW, and 1.5 times faster than RBS. Out of a total of 45,291 unique Mirai IPs present in this 1-hour period, SWORD only allowed 102 unique Mirai IPs undetected to scan the protected network before the first worm scan was detected, which accounted for 5,380 total worm scans in a 7.87-second period. SWORD therefore allowed around 57% less worm scans than PGD, 61% less worm scans than MRW, and 38% less worm scans than RBS before

Table 2. Mirai worm detection latency of each worm detector.

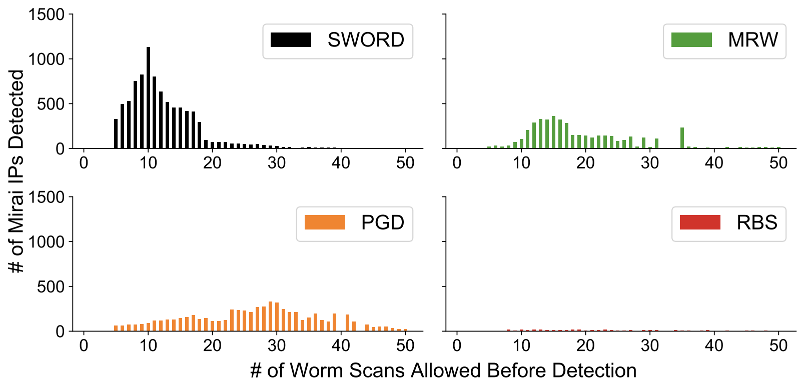
Detector	**First Detection**	**Allowed IPs**	**Allowed Scans**
SWORD	7.87s	102 IPs	5380 scans
MRW	19.77s	272 IPs	13728 scans
PGD	18.20s	235 IPs	12641 scans
RBS	12.50s	169 IPs	8674 scans

detecting the occurrence of the Mirai worm. While 5,380 allowed incoming scans may seem high, this accounts for only 0.23% of all of the incoming Mirai scans in the 1-hour testing period (in other words, SWORD detected the presence of Mirai before 99.77% of the Mirai scans reached the protected network). While PGD detected slightly more Mirai IPs overall, SWORD's ability to detect Mirai IPs faster could be a more valuable attribute to a network operator who may want to perform mitigative and preventative steps as soon as possible to limit the damage of the worm.

We further measured the detection latency of every Mirai IP detected, as shown in Figure 14a. Recall the latency is measured in terms of the number of worm scans allowed before detection. Among all Mirai IPs detected by SWORD (8,882), 98.87% of them were all detected before they could make more than 50 scans. The next best detector in terms of latency was RBS, none of the Mirai IPs detected (1,041) made more than 140 scans. For all the Mirai IPs that PGD and MRW detected (9,100 and 5,797, respectively), they made no more than 230 and 250 scans, respectively. In fact, SWORD was able to detect 80% of all of its detected Mirai IPs each within 20 scans. When compared to MRW, which detected less than 40%, and PGD along with RBS, which detected less than 10% of all of their detected Mirai IPs each within 20 scans, clearly SWORD can detect individual Mirai IPs faster than the other detectors.



(a) Detection latency (number of scans before detection) CDF of the Mirai IPs detected



(b) Number of Mirai IPs detected for low latency values

Figure 14. Detection latency of Mirai IPs detected.

Finally, Figure 14b details how many Mirai IPs each detector detected with very low latency (i.e.50 scans or less). For SWORD, the majority of low latency detections occurred within 10 scans, and a large portion of detections occurred even within just 5 scans. MRW's low latency detections were spread across 10 to 30 scans, and PGD's low latency detections were spread fairly evenly across the 5 to 50 scans range. Both MRW and PGD detected a significantly lower number of Mirai IPs with 5 scans, as compared to SWORD. RBS had such a low number of detected Mirai IPs for each latency that it is difficult to eyeball, but it too follows an even distribution similar to PGD, with its detections spread across the

10 to 30 scans range like MRW; it does not detect any Mirai IPs until 8 scans. Clearly, compared to other detectors, SWORD can detect many more Mirai IPs within a very small number of scans (e.g., 5 allowed scans); this fact is consistent with SWORD's lowest detection latency of the Mirai worm among all detectors evaluated.

4.5.4 Summary. As with outbound worm detection, SWORD again significantly outperformed all of the other detectors in inbound worm detection. While PGD detected slightly more total Mirai IPs than SWORD, SWORD outperformed PGD in terms of false negative rates and detection latency. SWORD also outperformed MRW and RBS in all three metrics. Furthermore, SWORD detected far more superspreading Mirai IPs with relatively high total scans, and surreptitious Mirai IPs with relatively low total scans and relatively low scanning rates than the other three detectors. In fact, SWORD was even able to detect some surreptitious Mirai IPs with extremely low scanning rates, unlike MRW, PGD, and RBS.

4.5.5 Limitations and Open Issues. It is possible that worm traffic may be present in the 1-hour training dataset. While we verified that no incoming Mirai scans were present, it may be possible that other strands of worms are present. Also, we only tested the detectors on one 1-hour period. While the results may vary across different 1-hour periods, we doubt that we will arrive at vastly different conclusions to the ones presented in this section.

4.6 Conclusion

We identify two principles that an effective worm detection solution must follow: (1) Worm propagation and worm detection are in an arms race, and a detector must consider potential countermeasures from worm authors; and (2)

Behavior-based worm detection must focus on the fundamental behavior of worm propagation that worms cannot avoid. Although there are many existing worm detectors, they are inadequate in following these two principles.

In this chapter, we revisited behavior-based worm detection. We identified that the fundamental behavior of worm propagation is that of connecting to new destinations, and designed SWORD, a detector that encompasses two complementary modules that can detect violations that a worm will cause in connecting to new destinations. With one module monitoring burst duration and the other ensuring quiescent periods, SWORD is extremely hard for a worm to evade.

Furthermore, unlike previous behavior-based worm detectors which only focus on outbound worm detection, we designed SWORD such that it not only can detect outbound worms, but also inbound worms, even though detecting inbound worms is difficult given that a detector usually observes only a fraction of scans from worm-infected hosts. Additionally, unlike some worm detectors that require bi-directional traffic to detect worms, SWORD can detect worms solely from uni-directional traffic.

As demonstrated in our evaluation, the SWORD detector significantly outperforms all other detectors. We evaluated SWORD and its competitors against the inbound Mirai worm using only uni-directional real-world incoming traffic, and demonstrated that SWORD is most effective at inbound worm detection, especially in detecting superspreading and surreptitious worm IPs.

CHAPTER V
DEFENSE AT THE GATEWAY: DETECTING DDOS TRAFFIC LEAVING AN IOT NETWORK

In the previous chapter, although we show that SWORD is highly effective at detecting the presence of a worm from incoming worm traffic, completely preventing worm infections is almost impossible, especially given the difficulties in incoming worm detection. Thus, if devices within a network happen to be infected, preventing DDoS traffic from leaving the network is crucial. Therefore, in this chapter, we focus on detecting DDoS traffic leaving an IoT network by inspecting traffic at its gateway.

A fundamental dilemma facing any security solution for IoT (especially as it moves closer to the devices themselves) is that it must consume as little resources as possible while still achieving the same level of performance as classical security systems. To address this dilemma, we present TWINKLE, a two-mode adaptive security framework for IoT environments. This framework allows an IoT network to incur a low resource consumption rate most of the time, and only incur higher overhead when suspicious behaviors are detected. We transform a classic security system for detecting and mitigating DDoS attacks at the source-end of the DDoS traffic (called D-WARD), into a new DDoS defense solution using TWINKLE (which we call D-WARD+).

We show that using TWINKLE leads to the reduction in retransmissions and connection duration, which has the positive side effect of reducing overall energy consumption in the network. Furthermore, only storing limited information in regular mode, allows TWINKLE to significantly reduce its overall memory consumption, as compared to a classic security system.

The chapter is derived in part from the following unpublished work: A Two-Mode, Adaptive Security Framework for Smart Home Security Applications by Sisodia, D.; Li, J.; Mergendahl, S.; Cam, H. Note, this unpublished work was itself derived from the following published work: Securing the Smart Home via a Two-Mode Security Framework [132] by Sisodia, D.; Mergendahl, S.; Li, J.; Cam, H. I am the leading author of these works and was responsible for leading all of the presented analyses.

5.1 Introduction

The Internet of Things (IoT) continues to pervade our lives. By the beginning of 2021, there were around 11.3 billion IoT devices connected to the Internet, and this number is expected to increase to more than 27 billion by 2025 [129]. However, as IoT devices are connected by the Internet, 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 [63]. 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. The landscape of IoT security is growing darker and more precarious as new malware strains, such as Hajime [62], Satori [4], the new Bashlite variants [142], Echobot [70], Silex [20], and many others, are being developed and deployed to exploit the many vulnerabilities of IoT devices.

While IoT devices and traditional machines often 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. More significantly, they tend to have scarce CPU and memory resources, lower network bandwidth, and limited battery capacity if not plugged into an external power source. These properties, which differentiate IoT devices from traditional machines, severely 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 [140]. For example, Sehgal et. al. [118] 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 benign 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 chapter, we focus on the smart home environment where security and privacy are especially important and address the ineffectiveness of classical security mechanisms in the smart home. We introduce a security framework called **TWINKLE**, **TW**o-mode **IN**-home framewor**K** toward **L**ightweight S**E**curity, 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. While plugged into

TWINKLE, every security application runs in regular mode for most of the time and incurs 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 applications that run these routines either continuously or periodically.

We further implement TWINKLE by addressing key implementation challenges and use it to transform two prior attack solutions for the smart home environment. We convert the D-WARD solution [93] 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. Our evaluation further demonstrates that D-WARD+ incurs much less overhead than D-WARD, while achieving equal to better efficacy in detecting and mitigating DDoS attacks.

5.2 Background and Related Work

5.2.1 Smart Home Security Analysis. We begin by studying papers that explore the current state of smart home security and provide suggestions on improvements in this environment. Denning et al. [42] 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. [102] report vulnerabilities in various household devices, such as the Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm. Sivaraman et al. [134] also analyze various smart home devices and rate them in terms of confidentiality, integrity, access control, and their ability to launch reflection attacks. Mare et al. [88] evaluate seven popular smart home platforms, mainly focusing on the extent these platforms support access control, privacy, and automation. Similarly, Celik et al. [22] study the security and privacy of five popular IoT programming platforms through program analysis. The main contribution of [52] is the discovery of security-critical design flaws in the SmartThings capability model and event subsystem. Fernandes et al. [54] introduce a security principle that prevents an attacker from misusing compromised OAuth tokens of trigger-action platforms. These papers give insight into the vulnerabilities and open issues that need to be addressed by smart home security frameworks and systems. Of the seven papers, only [102] and [54] provide security solutions. However, [102] only provides protection via access control rules deployed at the gateway router to prevent unauthorized in-bound and out-bound traffic, and [54] provides a solution to a very specific vulnerability of trigger-action platforms.

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. Our framework is also generic in that a network administrator can plug various security applications into it, allowing it to handle various vulnerabilities (including, but not limited to, trigger-action platform vulnerabilities).

5.2.2 Frameworks and Systems. In this subsection, we survey select papers which introduce security frameworks and systems targeted towards IoT environments. Table 3 summarizes the differences between each of the studied

Table 3. Comparison of frameworks and systems related to TWINKLE.

Paper	**Design Philosophy**	**Environment**	**Evaluation**	**Resource Consumption**
Bernabe et al. [15]	privacy preservation through contextual management	social IoT	proof-of-concept	not studied
Abie et al. [8]	game theory and risk analysis	eHealth	proof-of-concept	not studied
Rahmati et al. [107]	risk-based permission	smart home	real-world	not studied
Celik at al. [24]	policy-based enforcement	trigger-action platforms	real-world	not studied
Simpson et al. [126]	centralized and extensible security manager	smart home	real-world	not studied
Kang et al. [75]	access control techniques	smart home	proof-of-concept	not studied
TWINKLE	two-mode paradigm	smart home	real-world	studied

frameworks and systems, and provides a comparison with TWINKLE. In [15], the authors present a security framework based on the Architecture Reference Model (ARM) of the IoT-A EU project. The work in [8], uses game theory and context-aware techniques to create a risk-based adaptive security framework for IoT in an eHealth environment. Both [15] and [8] are proof-of-concept papers that do not provide evidence that the presented frameworks are viable in resource constrained environments.

Rahmati et al. [107] introduce a secure development methodology which leverages risk-based permissions for IoT networks, instead of permission models used by smart phone operating systems. Unlike TWINKLE, this work attempts to improve smart home security by focusing solely on the permission model aspect of IoT devices.

The authors of [24] present a policy-based enforcement system that prevents insecure device states that may occur in trigger-action platforms. Again, unlike our framework, this system is targeted towards the specific area of trigger-action platforms and cannot be applied to securing the generic IoT smart home. Furthermore, this system does not concern itself with reducing resource consumption which is a key aspect of our framework.

Similar to our framework, the frameworks presented in [126] and [75] are both targeted towards smart home environments. The authors of [126] present a centralized security manager, similar to the security manager component in our framework, whose main purpose is to provide reliable patching and update mechanisms to smart homes. The authors of [75] present a security framework that requires kernel-level modifications to provide authentication and access control mechanisms for smart home appliances. However, like [15] and [8], 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.

5.2.3 Security Mechanisms for Edge Computing. Because IoT and edge computing are closely related, we analyze existing security mechanisms for edge computing. Specifically, we focus on identification and authentication, which are two well-studied problems in edge computing.

In IP networks, the two main roles an IP address serves are as locator and end point identifier. The Host Identity Protocol (HIP) [97] is a host identification protocol that decouples these two roles by introducing a Host Identity (HI) name space as an end point identifier which is based on public key cryptography. One of the security advantages HIP provides is that it prevents machines on the Internet from directly accessing IoT devices without passing the strict security procedure of mutual peer authentication via Sigma-compliant Diffie-Hellman key exchange.

In fact, the preferred way of implementing HIP in edge networks is to use Internet Protocol Security (IPsec) [76] to carry the data traffic. IPsec is a network protocol suite that is used to authenticate and encrypt packets, thereby providing secure communication between two machines in an IP network. Specifically, the only defined method for implementing HIP is to use IPsec's Encapsulated Security Payload (ESP) to carry the data packets. When used in combination, HIP and IPsec not only provides data authentication, integrity, and confidentiality, but allows for secure IP multihoming and mobile computing.

Protocols, such as HIP and IPsec, which enhance the security of communications between IoT devices, are orthogonal to the security benefits that TWINKLE provides. Because HIP and IPsec only deal with the specific problems of host identification, and data authentication, integrity, and confidentiality, TWINKLE, which can handle a plethora of attacks, can be leveraged in combination with HIP and IPsec to provide a more secure environment for IoT networks.

5.2.4 Motivation for the TWINKLE Design and Possible Extensions. Lastly, we analyze papers that motivate certain design choices and components of TWINKLE. Instead of introducing new security countermeasures, the authors of [74] 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 [98] attempts to provide adequate mechanisms to control the flow of data and enforce policies based on users' preferences. Such work motivates the need for TWINKLE's automated component instantiation which allows users to intuitively and easily plug existing security applications into the framework without having any knowledge about the inner-workings of those applications.

The TWINKLE framework can be extended to include additional features that may work well with the two-mode paradigm, and further increase its defense efficacy and resource efficiency. In [25], the authors utilize special nodes that monitor traffic within the network to detect certain routing attacks. The work in [74], [98], and [25] show the need of user interaction, adjustable policies set by users, and dedicated event watchdog nodes for inspection of in-network communication, respectively. Also, the work in [79] provides motivation for allowing security policies, such as using efficient authentication and key agreement methods. He et al. [60] conclude that per-device granular access control policies are not sufficient, and instead, a combination of per-capability, per-relationship, and per-context granular access control policies are needed. TWINKLE can be extended to allow the user to specify these types of access control policies, and how these policies may change depending on the mode. Furthermore, the substantial research in the area of security in wireless sensor networks (WSNs), such as the work presented by Abduvaliyev et al. [7] and Roman et al. [112], where devices are extremely constrained, can be leveraged to further improve TWINKLE's resource efficiency.

5.3 TWINKLE: Design and Architecture

Many security solutions developed for the traditional Internet, if deployed in an IoT environment such as a smart home, may cause a substantial burden on some IoT devices due to their computing power, resource, and energy requirements. We design the TWINKLE framework in such a way as to not only preserve the salient features of classic security solutions, but also address the resource limitations that

IoT devices face. In this section we describe its design, architecture, and how it supports security applications running in a smart home.

5.3.1 Design with Two Modes for IoT-based Security Applications. 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(s) of attack. We also call these security applications **TWINKLE security applications**. For every TWINKLE security application, the network administrator can plug it into the TWINKLE framework when needed.

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 own efficacy. A security application may demand resources from a device, such as CPU utilization, memory consumption, power consumption, and bandwidth consumption, such that it is impossible for the device to meet that demand without sacrificing performance of the security application itself, or other applications running on the device.

To address this dilemma, we design the TWINKLE framework that supports security applications to run on top of TWINKLE and 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 TWINKLE security application invokes functions to detect suspicious behavior that an attack, if occurring, *must* display, whereas those functions must also be lightweight. Note, a legitimate operation may also display a suspicious behavior. Once it detects a suspicious behavior (i.e., an attack *may* be occurring), the security application enters vigilant mode to closely inspect 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.

This two-mode design differs from many classical security applications, which usually run in one mode. Specifically, a classical application usually runs continuously or periodically to monitor security-related events and must minimize both false positives and false negatives. It often employs complicated operations in order to be accurate in detecting attacks, thus consuming resources frequently and heavily. Conversely, a TWINKLE security application, by switching between these two modes but staying in regular mode most of the time, does not consume as much resources as classical applications. In regular mode, it is only concerned about detecting with high sensitivity the suspicious behaviors that attacks in question must demonstrate, even if legitimate operations may also demonstrate such behaviors. In other words, in regular mode it is more concerned with minimizing false negatives but less concerned with minimizing false positives. While every security application defines and handles suspicious behaviors of a different type, nature, or severity, this design choice simplifies the detection of suspicious

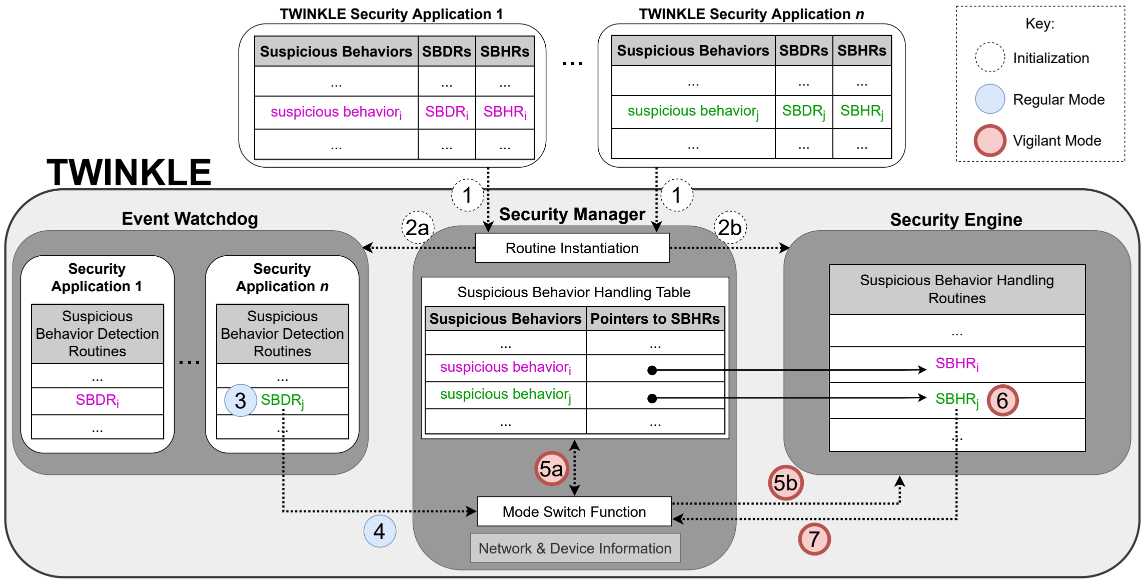


Figure 15. The basic architecture of TWINKLE.

behaviors for every security application, as they all can rely on vigilant mode to further check if a suspicious behavior is indeed from an attack. 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. Since regular mode is usually less resource-consuming than a classical one-mode system, and by only invoking the resource-consuming vigilant mode infrequently and on demand, a TWINKLE security application overall incurs much less resource consumption than classical security applications.

5.3.2 Architecture of TWINKLE. As shown in Figure 15, TWINKLE is composed of three main components: **security manager**, **event watchdog**, and **security engine**. The security manager is TWINKLE's central component and acts like a security operating system. It is responsible for supporting various security applications, maintaining information about the smart home and key data structures for security, instantiating the security functions at the event watchdog and the security engine according to security applications in place, and switching between regular and vigilant mode. The event watchdog is responsible for detecting suspicious behaviors. The security engine is responsible for further handling those suspicious behaviors, such as verifying whether a suspicious behavior is indeed from an attack or not.

In general, the security manager and security engine are deployed at a central node, such as the border router of a smart home, and an event watchdog can consist of more than one instance by running at multiple devices that can detect suspicious behaviors, such as a smart watch running a monitoring routine, a Raspberry Pi devoted to eavesdropping and monitoring IoT traffic, or any selected devices in the smart home. Certain security applications may instead require a single event watchdog to be installed at the network's border router.

Responding to the need of supporting various different security applications for the smart home, the network administrator can plug any security application into TWINKLE as needed (**Step 1** in Figure 15). In doing so, the security manager populates data structures and instantiates routines that run on the event watchdog and security engine, all according to the security application in question. On the other hand, in order to be supported by TWINKLE, a security application that handles an attack must define the suspicious behaviors that may be a sign of the attack in question. And for every suspicious behavior, the security application must define the routine that *detects* the suspicious behavior, which we call a **suspicious behavior detection routine (SBDR)**, and the routine that *handles* the suspicious behavior, which we call a **suspicious behavior handling routine (SBHR)**. A general principle here is that an SBDR should be lightweight while an

SBHR may be more resource-consuming since, as we explain below, the SBDR runs in regular mode and the SBHR runs in vigilant mode.

After a security application is plugged into TWINKLE, the security manager's **Routine Instantiation** module instantiates the event watchdog by having the event watchdog monitor the set of suspicious behaviors defined by the security application (**Step 2a**). Specifically, the event watchdog begins running the lightweight SBDRs defined by the security application.

TWINKLE provides a dynamic mechanism for a security application to install its event watchdog or security engine at any device needed. At start-up, lightweight processes that can receive, consume, and send messages run at each device that may be a candidate for running an event watchdog or security engine of a security application. When the network administrator deploys a new security application on TWINKLE, and needs to run the event watchdog or security engine code of the application on a device, the security manager can communicate with the device to ship, install, and eventually run the code on the device. Note, depending on the security application, an event watchdog may perform signature-based detection, behavior-based detection, or a combination of both. As described in the previously, the event watchdog ensures that detection is lightweight by not concerning itself with minimizing false positives. When tuned to be lightweight, off-the-shelf intrusion detection systems (IDSes), such as Snort [5], Suricata [6], and Zeek [103], could be used as a basis for the event watchdog.

Next, to instantiate the security engine, the Routine Instantiation module injects the SBHRs defined by the security application into the security engine. Furthermore, the Routine Instantiation module ensures that every defined suspicious behavior is mapped to an SBHR (**Step 2b**). It does so by populating a data structure called the **suspicious behavior handling table (SBHT)**. For each suspicious behavior, the SBHT points to a specific SBHR at the security engine for handling that behavior. While, in general, the security engine runs at a central node, such as a border router, some security applications may require the security engine to run at multiple locations in the network. For example, traffic from a malicious device may need to be dropped before it reaches the border router, and therefore the SBHR that is responsible for dropping malicious traffic should be installed in-network between the malicious device and border router.

Note, security applications may define parameters for various SBDRs and SBHRs. Furthermore, security applications may also require data to be tracked which may be needed for certain SBDRs and SBHRs. Parameters (i.e., threshold values that define various suspicious behaviors) defined by security applications that are required by SBDRs and SBHRs are transferred to the event watchdog and security engine during routine instantiation. Data required by security applications' SBDRs and SBHRs are stored in the security manager's **Network & Device Information** module.

Once the routines are instantiated, the application begins in regular mode, with the event watchdog running. In order to detect suspicious behaviors, in some cases the SBDRs may need to retrieve information stored in the security manager's Network & Device Information module, such as network topology or routing information (**Step 3**). As mentioned previously, data required by security applications' SBDRs and SBHRs are stored in this module, as well. Once the event watchdog detects a suspicious behavior, it notifies the security manager's **Mode Switch Function (MSF)** to handle the suspicious behavior (**Step 4**). Upon receiving the notification, the security manager then switches to vigilant mode. The

MSF takes the detected suspicious behavior as input, queries the SBHT (**Step 5a**) to determine which SBHR should be invoked (**Step 5b**), and passes the pointer of that SBHR to the security engine (**Step 5c**). The security engine, in turn, invokes the SBHR in question. Generally, as we said above, the SBHRs are heavyweight and should only be running in vigilant mode when invoked on demand. SBHRs, like the SBDRs in regular mode, may need to retrieve (and update) network information stored at the security manager (**Step 6**) in order to successfully handle a suspicious behavior. Once the SBHR finishes its execution, the security engine notifies the security manager, and TWINKLE returns to regular mode.

5.3.3 TWINKLE Security Applications. Although TWINKLE security applications are not a part of the TWINKLE framework, they affect how each component in TWINKLE operates. Specifically, the SBDRs and SBHRs, which are critical in detecting and mitigating attacks, are defined by the TWINKLE security applications plugged into TWINKLE. A TWINKLE security application must:

1. define a set of suspicious behaviors that may constitute the attack, and

2. define a set of routines to detect and handle these suspicious behaviors.

Handling an attack requires defining a set of suspicious behaviors that must be present if that attack is occurring. As mentioned before, suspicious behaviors are defined and detected via SBDRs, which run on event watchdog nodes in regular mode. Depending on the security applications, suspicious behaviors can be of varying granularity. An example of a coarse-grained suspicious behavior is all outgoing traffic exceeding 100 MB at any given time (thus necessitating inspecting the entire network as a whole to determine the suspicious behavior). An example of a fine-grained suspicious behavior is device X sending more than 100 KB traffic every 10 seconds (thus warranting inspecting a specific device to determine the suspicious behavior). Nonetheless, suspicious behaviors must be measurable and detectable by event watchdog nodes. Some suspicious behaviors can be easily measured, such as the amount of outbound traffic, while others may be more difficult, such as the energy consumption of a particular device.

However, suspicious behaviors, if present, are not sufficient in determining if an attack is actually occurring. Further analysis needs to be performed in order to detect if a suspicious behavior or combination of suspicious behaviors should be considered malicious. If an event watchdog detects a suspicious behavior, it triggers the security manager to enter vigilant mode where the security engine performs detailed inspection to determine if the suspicious behaviors are indeed malicious. In particular, a set of routines or functions must be defined to verify and mitigate an attack. These routines are dependent on the attack and vary across different security applications. For example, for certain suspicious behaviors, a security application may want to send probing signals to gain additional information from devices (SEND-SIGNAL), drop outbound traffic (DROP), change certain paths in the network (CHANGE-PATH), change the encryption protocol between certain devices (CHANGE-ENCRYPTION), or change the channel frequency between certain devices (CHANGE-FREQUENCY).

Security applications may handle any number of attacks. Table 4 shows a taxonomy of common attacks in smart home environments. TWINKLE provides a framework for *any* security application to utilize the two-mode paradigm for reducing resource consumption in IoT environments. However, while any security application can be plugged-into TWINKLE, how effectively the application will

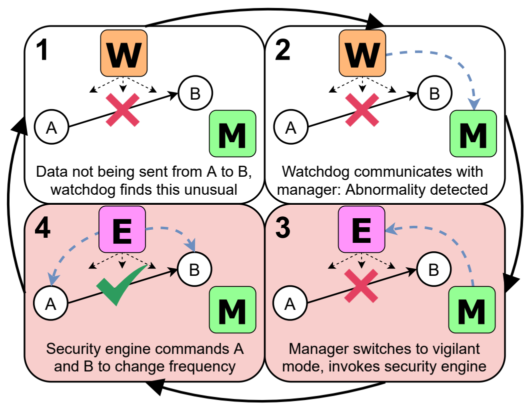
Table 4. Taxonomy of common attacks in smart home environments.

Security Requirements	**Attacks**	**Description**
Targeting Confidentiality	Side-Channel	Exploits data leakage vulnerabilities to gain sensitive information.
	Brute-Force	Exploits weak credentials to gain privileged access to devices.
Targeting Integrity	Voice-Command Injection	Exploits vulnerabilities in speech-recognition systems to inject malicious or inaudible voice commands.
	Event Spoofing	Exploits lack integrity checks in applications to propagate seemingly legitimate events to devices causing them to react in some way that benefits the attacker.
Targeting Availability	Sinkhole/Selective Forwarding	Exploits routing protocol vulnerabilities to launch a DoS attack by forwarding only a subset of packets to the destination.
	Jamming	Exploits how transceivers operate to launch a DoS attack that disrupts data transmission and forces devices to repeatedly retransmit packets.
	Battery-Draining	Exploits routing protocol vulnerabilities to launch a DoS attack by depleting devices' battery power.

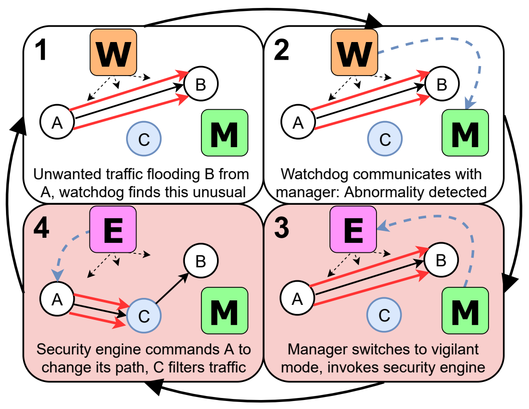
utilize the two-mode paradigm depends on how the security application itself is written. To exhibit TWINKLE's versatility in protecting an IoT network, we present three example TWINKLE applications, as shown in Figure 16, that use the TWINKLE framework to address jamming attacks, flooding attacks, and weak encryption.

5.3.3.1 *Example Application to Address Jamming Attacks.*

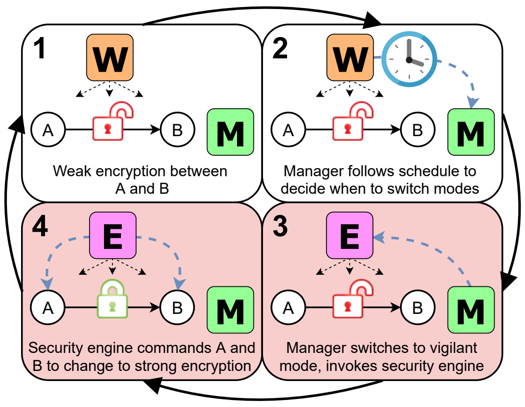
In Figure 16a, TWINKLE is being used to handle a jamming attack where the link between devices A and B is being jammed by an unknown attacker. In the first two steps (box 1 and box 2), TWINKLE is running in regular mode. In the first step, an event watchdog node in the network, while running an SBDR defined by the security application, detects suspicious behavior. In this case, device B is not receiving traffic from device A, which is abnormal. The event watchdog can


1
W
A
B
M
Data not being sent from A to B, watchdog finds this unusual
2
W
A
B
M
Watchdog communicates with manager: Abnormality detected
3
E
A
B
M
Manager switches to vigilant mode, invokes security engine
4
E
A
B
M
Security engine commands A and B to change frequency

(a) Jamming example


1
W
A
B
C
M
Unwanted traffic flooding B from A, watchdog finds this unusual
2
W
A
B
C
M
Watchdog communicates with manager: Abnormality detected
3
E
A
B
C
M
Manager switches to vigilant mode, invokes security engine
4
E
A
B
C
M
Security engine commands A to change its path, C filters traffic

(b) Flooding example


1
W
A
B
M
Weak encryption between A and B
2
W
A
B
M
Manager follows schedule to decide when to switch modes
3
E
A
B
M
Manager switches to vigilant mode, invokes security engine
4
E
A
B
M
Security engine commands A and B to change to strong encryption

(c) Encryption example

Figure 16. Three example TWINKLE security applications.

detect this abnormality by eavesdropping on the communication between A and B by setting its network interface controller (NIC) to promiscuous mode. Because the watchdog does not receive any traffic from A to B for a prolonged period of time (e.g., the period of time without receiving any traffic surpasses the threshold for what is considered normal), it notifies the security manager at the border router of the suspicious behavior, by invoking the MSF. The security manager switches TWINKLE to vigilant mode (box 3). The security manager's SBHT matches the suspicious behavior detected with the proper SBHR (e.g., CHANGE-FREQUENCY), and the security manager then invokes the security engine to run that SBHR. By running the CHANGE-FREQUENCY routine, the security engine commands A and B to change their communication channel to possibly alleviate the jammed link (box 4). In a worst case scenario, where no SBHR can mitigate the attack (e.g., all channels are being jammed), the security manager can notify the network administrator (we assume that the manager is running on a router with wired ethernet connection, thereby bypassing any jamming that may be present). After the attack is mitigated, either by successfully changing the communication channel or by the network administrator manually identifying and removing the malicious device from the network, the SBHR terminates, and regular mode resumes.

5.3.3.2 Example Application to Address Flooding Attacks.

In Figure 16b, we show how TWINKLE handles a flooding attack where device A is flooding device B with unwanted traffic. Note, the unwanted traffic may have originated from a node other than A, who is simply forwarding the traffic to B, and therefore may not be malicious. Similar to the previous example, the first two boxes show TWINKLE's operation in regular mode. First, the event watchdog, running an SBDR, detects an unusually large amount of traffic being sent to device B by device A, which surpasses the normal threshold. The event watchdog then invokes the security manager, which (by invoking the MSF) switches the security application to vigilant mode (boxes 2 and 3). The SBHT at the security manager points to the security engine's SBHR (e.g., CHANGE-PATH) and, as a result, the security manager invokes the security engine. The security engine then commands A to change its path to route its traffic to a node C which can filter the unwanted traffic and route only the wanted traffic to B (box 4). Here, C can be a machine dedicated to detecting and filtering malicious traffic which may be required for this particular security application. If A does not change its path, and continues to flood B, the security engine may notify the network administrator, or attempt to isolate A from the rest of the network by running other SBHRs. Once the attack had been mitigated, the SBHR terminates and the framework returns to regular mode.

5.3.3.3 Example Application to Address Weak Encryption. The TWINKLE framework can be used to proactively secure an IoT network that requires expensive but stronger encryption at certain critical periods. Figure 16c shows an example of how TWINKLE can help. For this IoT network, there are certain times during operation that require stronger encryption. During normal operational times, the network is operating under regular mode when weak or no encryption between certain devices, say A and B, is sufficient (box 1). However, once a critical period begins, the watchdog triggers the MSF (box 2), which switches the security application to vigilant mode. Once in vigilant mode, the security manager invokes the security engine to handle the time change and commands the devices to change to strong encryption (e.g., CHANGE-ENCRYPTION; box 4). During the secure period, the event watchdog nodes in the network can ensure that strong encryption is being used between the critical devices. Once the critical period ends, the framework returns to regular mode.

5.4 TWINKLE Implementation Details

There are three main challenges that must be addressed in order to implement the TWINKLE framework. First, the security application plugged into the framework must be represented in a way as to allow for routine instantiation by the security manager. Second, routine instantiation must be done automatically, without human intervention. Finally, after the routines are initiated, all of the components running the routines must interact with each other in order to detect,

```
11 <behavior>
12     <name> ____________________ </name>
13     <SBDR>
14         <name> ________________ </name>
15         <parameter_name> ______ </parameter_name>
16         <parameter_value> _____ </parameter_value>
17         <devices> _____________ </devices>
18         ...
19     </SBDR>
20     <SBHR>
21         <name> ________________ </name>
22         <parameter_name> ______ </parameter_name>
23         <parameter_value> _____ </parameter_value>
24         <devices> _____________ </devices>
25         ...
26     </SBHR>
27 </behavior>
28 ...
```

(a) Behavior element

```
29 <routine>
30     <name> ____________________ </name>
31     <lang> ____________________ </lang>
32     <code> ____________________ </code>
33     <compile> _________________ </compile>
34 </routine>
35 ...
```

(b) Routine element

Figure 17. A template XML file for representing a TWINKLE security application.

verify, and mitigate attacks. What follows is a detailed look at how the TWINKLE framework may be implemented in actuality and how each aforementioned challenge is addressed.

5.4.1 Representing a Security Application. In order for the TWINKLE framework to support various security applications with minimal effort, each security application must be represented in such a way as to allow for automatic routine instantiation. Generally, either the application's developer or network administrator is responsible for creating such a file. We represent security applications using a markup language such as XML. This allows for the security manager to easily parse the file to find which devices need to run which routines. Figure 17 depicts a template XML file for representing a security application. An XML file represents a single security application, and consists of the following two main type of elements: behavior and routine elements.

The *behavior* element contains information about a suspicious behavior that the security application must detect and handle (lines 11 to 28, as shown in Fig 17a). Note, each suspicious behavior is defined in a separate *behavior* element. The *behavior* element contains a *name* element, which serves simply as a unique identifier to differentiate it from other suspicious behaviors, along with an *SBDR* and an *SBHR* element, which represent the SBDR and SBHR to detect and handle the given suspicious behavior, respectively. The *SBDR* and *SBHR* elements further consist of a *name* element that represents the name of the routine, along with *parameter_name*, *parameter_value*, and *devices* elements. The *parameter_name* and *parameter_value* elements define parameters (e.g., thresholds) that the routine takes as input (as command-line arguments). Note again, there must be separate *parameter_name* and *parameter_value* elements for each routine parameter. The *devices* element specifies the devices, as a list of device IDs, on which the given routine will run.

The *routine* element contains the routine code, and information necessary for compiling and running the code (lines 29 to 35, as shown in Fig 17b). It contains a *name* element that represents the name of the routine. The name of the routine must match the name of an SBDR or SBHR specified in a *behavior* element. The *lang* element specifies the programming language in which the routine is written, which the device responsible for running the routine must know because the command to run the routine may depend on the language. For example, if the routine is written in a language that must be interpreted, such as Python (e.g., "*python routine.py arg0, arg1, ...*") and Java (e.g., "*java routine arg0, arg1, ...*"), the device must invoke the appropriate interpreter. The *code* element contains the actual code for the routine. The code of a routine can be as simple as a single function, or as complex as multiple classes that may need to be split across multiple files. The *compile* element specifies the compilation command for the code if it needs to be compiled.

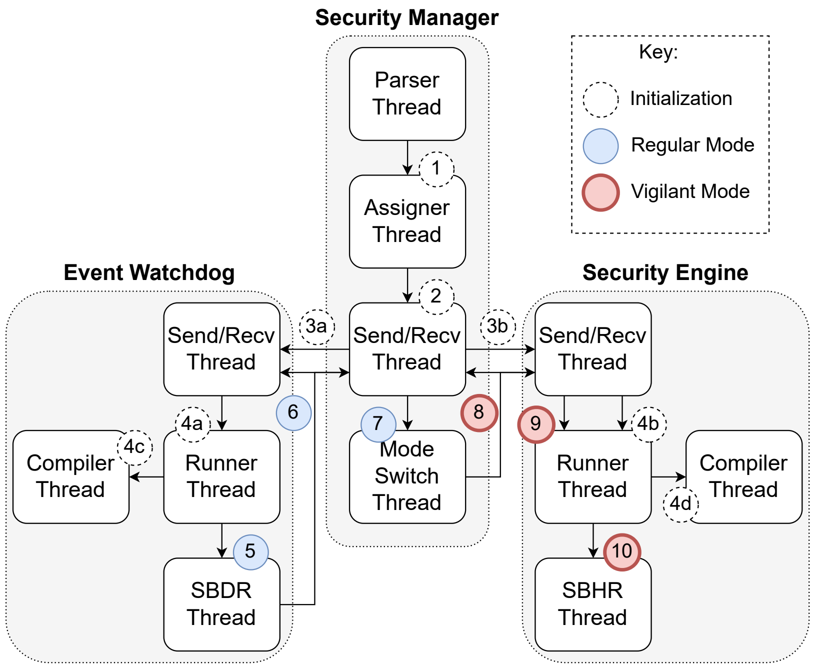


Figure 18. A flow diagram of the interactions between the main threads in the TWINKLE framework.

It is important to note that one can introduce additional elements into the XML file when necessary, especially in order to include additional information required for certain security applications. Minimal change to the TWINKLE framework (mainly the security manager) would be required in order to parse any new elements.

5.4.2 Automated Routine Instantiation.

5.4.2.1 Initialization and component threads. Each TWINKLE component is a lightweight process that contains several threads that are executed at start-up. The network administrator initializes TWINKLE by executing the security manager, event watchdog, and security engine processes. The network administrator provides the security manager process with a configuration file that includes information about devices in the network, such as a device's ID, IP address, and listening port, along with architecture, board, and operating system information, which the security manager process stores in the Network & Device Information module (which can simply be a database). The network administrator provides the event watchdog and security engine processes with a port number on which it should listen on for TWINKLE messages. Each process starts the component threads, which are independent from the security applications plugged into TWINKLE and must be running before the routines are instantiated. We list the threads for each component below and provide a brief description of each. We explain each in detail throughout the rest of this section.

- Security manager threads:
    * parser thread: parses and extracts routines from a security application file
    * assigner thread: assigns routines to the event watchdog and security engine
    * send/recv thread: sends and receives information to and from the event watchdog and security engine
    * mode switch thread: given an alert from the event watchdog, invokes the appropriate SBHR
    * query handler thread: handles queries from the event watchdog and security engine
    * alert thread: sends SMS and email alerts to network administrator
    * cross-compiler thread: cross-compiles a given routine (only when needed)
- Event watchdog and security engine threads:

∗ send/recv thread: sends and receives information to and from the security manager

∗ compiler thread: compiles a given routine

∗ runner thread: runs a given routine

5.4.2.2 Parsing a security application. Once a security application needs to be plugged into TWINKLE, the network administrator uploads its file onto the security manager. Figure 18 shows a flow diagram of the interactions between the main threads in the TWINKLE framework. The **parser thread** reads in the file and parses each element. It parses each *behavior* element, and extracts the necessary information to populate the SBHT and create source code files for the SBDRs and SBHRs. Specifically, it appends a row of two columns to the SBHT, where one element of the row is the name of the suspicious behavior and the other element is a pointer to the SBHR (e.g., the name of the SBHR) that handles that suspicious behavior. Parameter values, as specified in the *parameter_value* elements, are ultimately sent as arguments along with the source code files to event watchdog and security engine devices. In order to correspond parameters to their routines, the parser matches the *name* element in the *SBDR/SBHR* element with the *name* element in the *routine* element. In order to correspond routines to the devices on which they will run, the parser parses the *devices* element. Once all parameters are matched with their routines and all routines are matched with devices, the parser parses the *code* element to generate the source code file(s), along with the *lang* and *compile* elements. Finally, it sends all of the parsed information of the security application to the **assigner thread** (circle 1 in Figure 18).

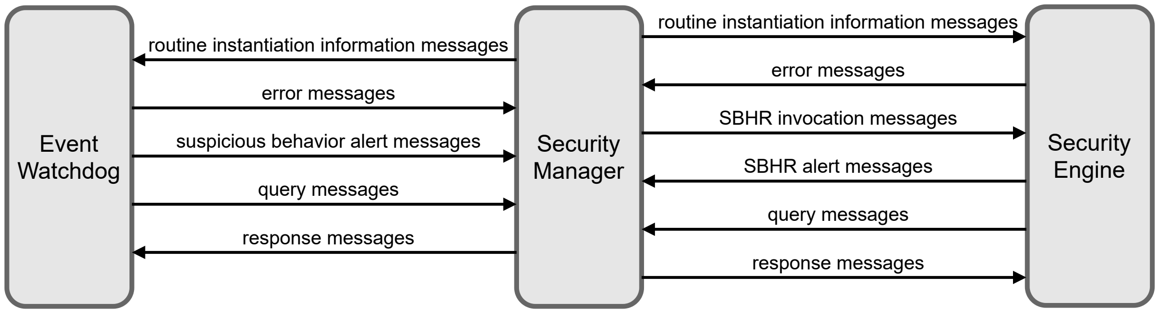


Figure 19. The messages TWINKLE currently supports for inter-component communication.

5.4.2.3 Assigning routines to devices. After the assigner thread receives the parsed information from the parser thread, it assigns the SBDR and SBHR source code to the event watchdog and security engine devices, respectively. It does so by mapping each device's ID to its IP address and port, and appends this information to the original parsed information. Then the assigner thread sends all of the information of the security application to the **send/recv thread** (circle 2 in Figure 18).

5.4.2.4 Sending routines to devices. After the security manager's send/recv thread receives a message from the assigner thread for a device, it simply forwards the message as a *routine instantiation information message* to the device (circles 3a and 3b in Figure 18). The send/recv thread creates and manages a TCP/IP socket for each receiving device, where each socket is connected to a receiving device's IP and port, which it fetches from the Network & Device Information module. Note, TWINKLE currently supports seven types of messages for inter-component communication, as shown in Figure 19.

5.4.2.5 Executing routines. After receiving a routine instantiation information message, the device's send/recv thread forwards it to the **runner thread** (circles 4a and 4b in Figure 18). The runner thread first checks if the routine needs to be compiled or can be interpreted. If the former, it invokes the **compiler thread**, which compiles the source code as specified by the given compilation command into an executable (circle 4c and 4d in Figure 18). It then checks the type of the routine. If the routine is an SBDR, the runner runs the routine in a new SBDR thread (circle 5 in Figure 18). If the routine is an SBHR, it saves the code to be executed when the SBDR thread detects the suspicious behavior in question. Note, by compiling and running routines in such a way, routines can be easily *hot-plugged* into TWINKLE without requiring any of the devices to be restarted or interrupted.

5.4.2.6 Dealing with errors. If an error is encountered at any point in the compiler and runner threads, the compiler or runner thread will send an *error message* that includes the specific compilation or runtime error, to the security manager. The security manager's **alert thread** can then send an alert to the network administrator that an error during routine instantiation occurred.

5.4.2.7 Cross-compiling for embedded systems. The security manager's **cross-compiler thread** cross-compiles routines on behalf of extremely lightweight devices, such as an embedded system without an operating system. To do so, the assigner thread invokes the cross-compiler thread before invoking the send/recv thread. The compiler installed on the security manager must be compatible with the target device's processor architecture. Note, because compiling on the target machine is generally more safe and straightforward, cross-compilation is only done when absolutely necessary; capable devices compile routines themselves, as previously described.

5.4.3 Component Interaction After Routine Instantiation.

Next, we explain how the components interact with each other after routine instantiation is complete. For simplicity, we explain the interaction between a single event watchdog device and security engine device.

5.4.3.1 Generating suspicious behavior alerts. In regular mode, the watchdog device's SBDR thread is running (or multiple SBDR threads are running). Once an SBDR thread detects a suspicious behavior it constructs a *suspicious behavior alert message* that includes the device's ID, name of the suspicious behavior, and a list of parameters to be consumed as arguments by the corresponding SBHR. These parameters are usually identifying features of the entity that caused the suspicious behavior alert (e.g., the source and destination IP:port pairs of a suspicious connection). Note, the SBDR and corresponding SBHR must be coded so that the output of the SBDR can be seamlessly handled as input to the SBHR. The SBDR thread sends the suspicious behavior alert message to the send/recv thread which in turn forwards the message to the security manager's send/recv thread, as shown by circle 6 in Figure 18 (note, the connection established during routine instantiation is kept alive, thus allowing the two devices to communicate over the same connection).

5.4.3.2 Mode switching and invoking the SBHR. The security manager's send/recv thread observes that the message is a suspicious behavior alert message, and sends the message to the **mode switch thread** (circle 7 in Figure 18). The mode switching thread first extracts the name of the suspicious behavior from the message, looks it up in the SBHT, and finds the name of the corresponding SBHR that should be invoked. It then constructs an *SBHR invocation message*, which includes the name of the SBHR, the event watchdog's device ID, and the list of parameters, and sends it to the send/recv thread, which in turn forwards the message to the security engine's send/recv thread (circle 8 in Figure 18).

5.4.3.3 Running the SBHR. The security engine's send/recv thread receives the message, observes the message is an SBHR invocation message, and sends the message to the local runner thread (circle 9 in Figure 18). The runner thread finds the SBHR with the same name as is specified in the message, appends the parameters in the message to the list of arguments, and prepares to run the SBHR. In the same way that the event watchdog's runner thread created a thread for running its SBDR, the security engine creates a thread for running the SBHR (SBHR thread), as shown by circle 10 in Figure 18. During the process of running, the SBHR may need to issue an alert to the network administrator, and it does so by sending an *SBHR alert message*, which includes all information regarding the alert, to the security manager. The security manager's alert thread in turn forwards the entire SBHR alert message to the network administrator.

5.4.3.4 Querying the security manager. Finally, if at any time an SBDR or SBHR thread needs to query the security manager's Network & Device Information module, it sends the query (e.g., an SQL command) encapsulated in a *query message* to the security manager via the local send/recv thread. The security manager's send/recv thread forwards the query message to the **query handler thread**, which simply extracts and executes the query. It then encapsulates the result in a *response message*, and sends the message back to the send/recv thread, which forwards it to the receiving device's send/recv thread. Note, the query message may include update commands, which when executed, will update the Network & Device Information module with new information.

5.5 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.

5.5.1 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 [63], recent DDoS attacks have been launched from compromised IoT devices and networks [69].

5.5.2 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 [93], which we detail in this subsection.

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. The classification is 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. D-WARD classifies each individual connection as good, transient, or bad, also based on the ratio of sent packets to received packets (smoothed over time) of the connection; a connection is classified as good if its smoothed packet ratio is below a threshold defined by a *legitimate TCP connection model*, transient if there is not enough information about the connection to discern its packet ratio, and bad if it is classified as neither good nor transient. D-WARD's observation component stores information in an *agflow table* and a *connection table*. In the agflow table, D-WARD stores the number of bytes sent, received, and dropped for each agflow, which is used to label the agflow and calculate the potential rate-limit for connections in a suspicious or attack agflow. In the connection table, D-WARD stores the smoothed packet ratio for each connection, which is used to determine if connections follow the legitimate connection model. For agflows that are labeled suspicious or attack, D-WARD first checks the smoothed packet ratio of each connection in the agflows to determine which connections need to be rate-limited; only bad and transient connections are rate-limited. Specifically, the rate limiting module cuts the allowed sending rate of all bad and transient connections in a suspicious or attack agflow to a fraction, f_{dec}, of the afglow's current sending rate. The observation module observes the aglow for a certain period of time, called the *observation interval*. The dynamic rate-limit for the next observation interval is determined by the *agflow compliance factor*, which is the ratio of bytes sent to the sum of bytes sent and bytes dropped; a high compliance factor leads to a relaxed rate-limit in the next observation period. Finally, the traffic-policing module decides whether to forward or drop each outgoing packet. It allows all packets from good connections and transient connections that belong to good agflows to be forwarded, but drops packets based on the current rate-limit from bad and transient connections that belong to attack or suspicious agflows.

D-WARD is designed for DDoS attacks launched from traditional end-hosts on the Internet, and therefore, several drawbacks may arise when deploying it in a smart home environment that otherwise would not be noticed in a traditional network. First, the memory consumption caused by storing agflow and connection statistics may be too costly in constrained IoT networks. Second, and most importantly, D-WARD could hurt benign devices if their connections are labeled as transient or mislabeled as bad connections, since their traffic, if over the dynamic rate-limit, is dropped. While a traditional benign end-host can recover from the accidental loss of their packets, in an 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 connection duration. As we show in Chapter 5.6, unnecessary retransmissions and an increase in connection duration directly leads to an increase in energy consumption.

5.5.3 D-WARD+: A Two-Mode Approach Against DDoS Attacks. We therefore transform D-WARD into D-WARD+ that runs on TWINKLE. To overcome the aforementioned drawbacks of D-WARD, D-WARD+ significantly reduces memory consumption by not storing any connection-level information in regular mode, and only storing suspicious and bad connection information in vigilant mode. Furthermore, 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 dropping their packets as done in D-WARD. Since these connections could be from benign devices, leveraging fast retransmit does not cause their packets to be dropped, but does lower the amount of DDoS traffic departing from the network. In this subsection, we present the two-mode design of D-WARD+ and explain in detail why D-WARD+ is better suited for an IoT environment as compared to D-WARD.

The XML representation of the D-WARD+ security application consists of a single behavior element for detecting and handling suspicious agflows, and two routine elements. Within the behavior element, the SBDR and SBHR elements each specify the name and input parameters of a routine. The SBDR element defines an *agflow monitoring routine*, which observes and labels each agflow in the network and detects suspicious agflows. The SBHR element defines a *connection monitoring routine*, which inspects every suspicious aglow more closely, including each connection in the agflow. The agflow monitoring routine has a single input parameter value that defines the threshold for determining if an agflow is suspicious in terms of its ratio of sent packets to received packets, while the connection monitoring routine does not have any input parameters. The routine elements provide the programming language, code, and compilation specifics of the SBDR and SBHR.

The security manager, event watchdog, and security engine of D-WARD+, all running at the border router, are designed as follows. The security manager's Network & Device Information module stores the agflow and connection tables, along with keeping track of a fixed rate-limit based on the receiver's TCP receive window (RWIN) for every suspcious agflow, which is detailed in the following paragraphs. The event watchdog consists of the agflow monitoring routine and invokes the security engine when it detects a suspicious agflow. The security engine consists of the connection monitoring routine.

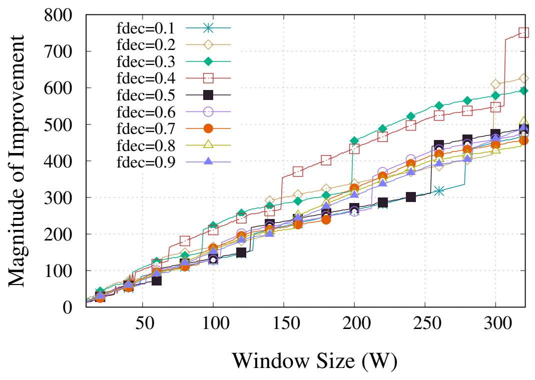
As a security application of TWINKLE, D-WARD+ handles DDoS attacks by switching between the two modes. In regular mode, the event watchdog keeps track of each agflow's sent to received packet ratio and stores this information in the agflow table, located at the security manager. However, unlike D-WARD, the event watchdog does not keep track of any connection-level information in regular mode. If the event watchdog detects a suspicious agflow, it invokes the security manager's MSF to determine what routine to execute at the security engine.

In vigilant mode, the MSF invokes the security engine's connection monitoring routine to handle the agflow in question. The security engine begins keeping track of the smoothed packet ratio for bad and *suspicious connections*, and stores these ratios in the connection table, also located at the security manager. Here, we introduce a new connection class called suspicious connections, which include transient connections and other connections that have only slightly surpassed the smoothed packet ratio threshold (i.e., connections that may be legitimate, but were labeled bad by the legitimate connection model). Introducing this new class of connection allows us to reduce collateral damage, especially since false positives caused by strict legitimate connection models leads to unnecessary network overhead (specifically, an increase in retransmissions and connection durations). In addition to the smoothed packet ratio for each bad and suspicious connection, the security engine periodically keeps track of the receiver's RWIN, for each suspicious agflow. Unlike D-WARD, which attempts to guess the maximum sending rate that the receiver can handle by calculating a dynamic rate-limit, D-WARD+ sets a fixed rate-limit at the beginning of each observation period which is set to the current RWIN. Although D-WARD+ is periodically storing an additional value in memory, it ensures that traffic sent from the IoT network never overwhelms the receiver. Furthermore, we will show that D-WARD+ significantly decreases the overall memory consumption as compared to D-WARD in the evaluation section, and therefore, periodically keeping track of RWIN is feasible. The security engine monitors each suspicious connection of the suspicious agflow; it sends three duplicate TCP acknowledgments to the device of the connection, which, by following the TCP congestion control design, reduces its congestion window by half, thus halving its sending rate. 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 detects it, labels the connection as a bad connection, and drops its packets (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 the static rate-limit 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, thus mitigating an ongoing DDoS attack.

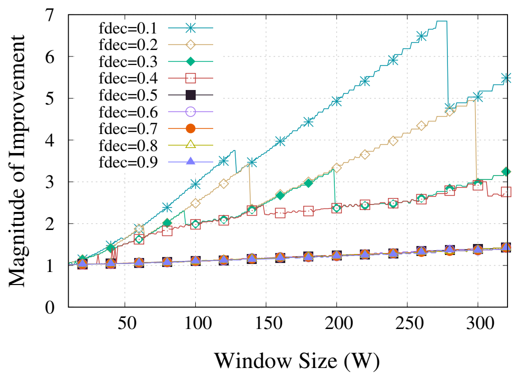
Based on the two-mode design above, D-WARD+ is more suitable to a smart home environment than D-WARD. It significantly reduces memory consumption by being selective with what information is stored in the agflow and connection tables. Furthermore, by not dropping packets as in D-WARD, D-WARD+ instead informs devices to transmit more slowly. Doing so avoids retransmissions of packets from benign devices, thus lowering network overhead and power consumption.

5.6 Evaluation

We evaluate TWINKLE's two-mode design by showing how D-WARD+ outperforms D-WARD in source-end DDoS defense. The metrics we focused on are retransmissions, connection duration, energy consumption, and memory



(a) Retransmissions



(b) Connection duration

Figure 20. Comparison of number of retransmissions and connection duration under D-WARD and D-WARD+.

consumption. We additionally compared the effects of D-WARD+ and D-WARD on a naive TCP flooding attack versus a smart TCP flooding attack, and analyze the differences in how the two systems detect and mitigate such attacks.

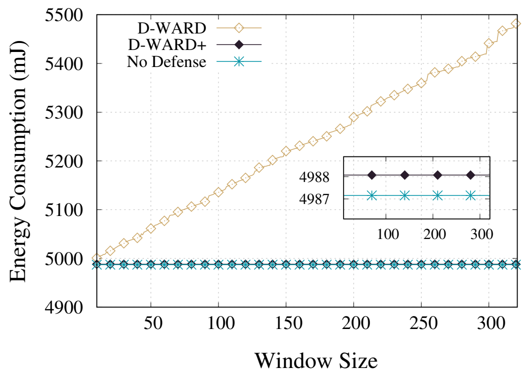
The evaluation results are a mixture of real-world testing, simulation, and formulation, where small-scale testing was performed on a real-world IoT testbed and large-scale results were obtained through simulation and formulation. For small-scale results, we constructed two IoT testbeds, one in which the devices communicated over 802.11b/g/n Wi-Fi, and the other in which the devices communicated over Bluetooth LE. Both testbeds consisted of several low-powered Raspberry Pi Zero W devices (1 GHz single-core CPU, and 512 MB of RAM), and a 2015 Dell XPS (2.2 GHz dual-core CPU, and 8 GB of RAM) as the border router. For large-scale results, we implemented the TWINKLE framework, D-WARD+, and D-WARD in Java on a 2015 Dell XPS with the same specifications as mentioned previously. When comparing D-WARD and D-WARD+ through formulation, TCP Reno is utilized for congestion control.

The main difference between D-WARD+ and D-WARD is that D-WARD+ utilizes the fast retransmit mechanism instead of dropping packets from suspicious 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 subsection, we analyzed the attainability of these goals in a smart-home network that utilizes D-WARD+. Specifically, we analyzed the following:

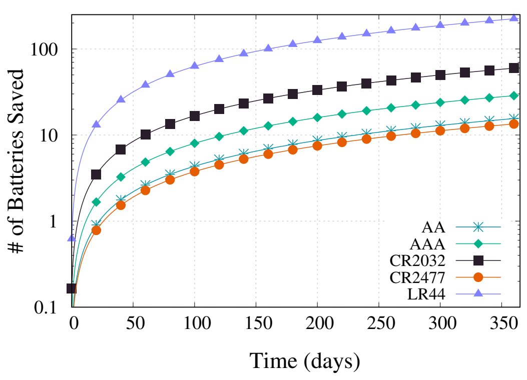
1. the ratio of retransmissions D-WARD requires of a benign suspicious connection over the amount required by D-WARD+;
2. the difference in connection duration of a benign suspicious connection under D-WARD compared to that of D-WARD+;
3. the energy consumed by a benign device under D-WARD and D-WARD+;
4. the amount of memory consumed by the border router running D-WARD and D-WARD+;
5. the behavior of a naive attacker under D-WARD and D-WARD+;
6. the behavior of a smart attacker under D-WARD and D-WARD+.

5.6.1 Retransmissions.

In order to compare the amount of retransmissions required of a benign suspicious connection by D-WARD and D-WARD+, we examined, through formulation, the number of retransmissions required of a benign suspicious 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+.



(a) Energy consumption



(b) Number of batteries saved

Figure 21. Energy consumption under D-WARD and D-WARD+, along with the number of batteries D-WARD+ saves over a one year period.

Figure 20a presents the ratio of retransmissions of a benign suspicious connection under D-WARD over D-WARD+. We call this ratio "magnitude of improvement" because it signifies the magnitude at which D-WARD+ prevents unnecessary retransmissions, as compared to its counterpart. We measure the magnitude of improvement with respect to two main parameters: the sender's congestion window size, W, at the time D-WARD or D-WARD+ detects a potentially malicious 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. Mirkovic et al. set f_{dec} to 1/2 by default [93]. 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 suspicious devices follow TCP congestion control, D-WARD drops $W - W * f_{dec}$ segments every 2 RTTs. Thus, as W increases, D-WARD drops more segments which causes more retransmissions. With a large window size and depending on the f_{dec}, D-WARD may require more than 500 times the number of retransmissions than D-WARD+. Even when the window size is less than 10 and given any pre-set fraction of allowed traffic, D-

WARD still requires more than 10 times the number of retransmissions than D-WARD+.

5.6.2 Connection Duration. We further compared how long a benign suspicious 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 suspicious connection that attempts to send 2.5 MB of data outside of the policed network, again at a maximum bandwidth of 250 Kb/s, under both D-WARD and D-WARD+.

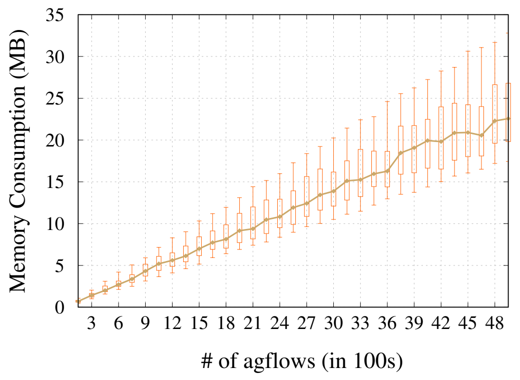
Figure 20b shows the magnitude of improvement in connection duration of D-WARD+ over D-WARD (ratio of average connection duration under D-WARD, over average connection duration under D-WARD+) with respect to the two main parameters W and f_{dec}. When f_{dec} is set low, D-WARD may punish a benign suspicious connection too heavily which leads to long connection durations. However, in cases where f_{dec} is set high (0.5 or above) and W is small, a benign suspicious connection's duration under D-WARD+ is only slightly faster (at most 3 seconds) than if it were under D-WARD (i.e., a small to no magnitude of improvement).

5.6.3 Energy Consumption. Based on the analysis presented by Feeney et al. [49], who estimate the microwatt seconds consumed by a wireless device with respect to the amount of data transmitted, we estimate, through formulation, the energy consumption D-WARD and D-WARD+ requires from a benign IoT device. The results are shown in Figure 21a. Under D-WARD (with an f_{dec} of 1/2), energy consumption increases linearly with respect to the benign device's congestion window size. However, under D-WARD+, energy consumption is static across varying window sizes. Furthermore, D-WARD+ contributes to less than 1 mJ of extra energy consumption for a benign device. This is again due to the fact that D-WARD+ does not throttle traffic and, as a result, does not require a large amount of retransmissions.

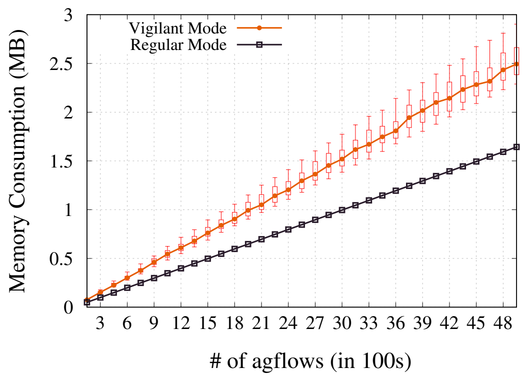
We further extend the energy consumption results to analyze D-WARD's cost of increased energy consumption by showing how much battery life D-WARD+ can save over D-WARD. We examine the battery life of a benign IoT device under D-WARD and D-WARD+ across five popular IoT batteries (alkaline AA, alkaline AAA, CR2032, CR2477, and LR44) and present how many more batteries an IoT device consumes under D-WARD throughout a year of deployment. These results are shown in Figure 21b. On average, across the batteries tested, a benign IoT device consumes 15.55 less batteries every year under D-WARD+.

Note that energy consumption should still be a concern for devices that are plugged into an external power source. Energy efficiency is a critical factor for the rise in smart home environments and this is especially true for large-scale environments, such as smart cities [137]. Therefore, a defense system that minimizes energy consumption is preferable in IoT environments, no matter if the devices are plugged-in or battery powered.

5.6.4 Memory Consumption. Next, we analyzed the memory consumption at the border router under D-WARD and D-WARD+. D-WARD maintains information about agflows and connections, while D-WARD+ in regular mode maintains information about agflows, and in vigilant mode, maintains information about suspicious and bad connections. Therefore, the number of agflows and connections per agflow affects the memory consumption of both systems.



(a) D-WARD

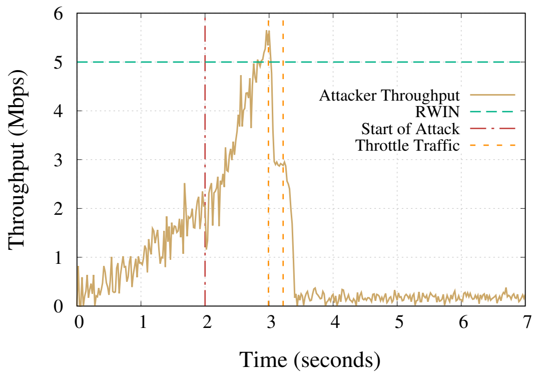


(b) D-WARD+

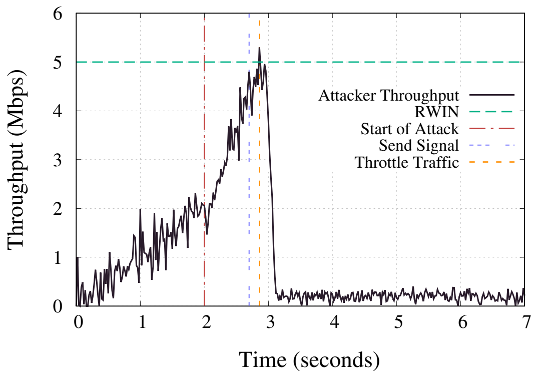
Figure 22. Memory consumption for D-WARD and D-WARD+ (in regular mode and vigilant mode). In each graph, 80% of all agflows are good and 80% of all connections are good.

Figure 22 shows the large-scale memory consumption results of both systems through simulation (a total of 100 runs). We assume that 80% of all agflows are good and 80% of all connections are good. The number of connections per agflow follows a power law distribution, where a few agflows have many connections (over 100), while most agflows have only a few connections each (less than 5).

It is clear that D-WARD (Figure 22a) incurs more memory consumption than D-WARD+ (Figure 22b), especially as the number of agflows increase. This is due to the fact that D-WARD+ does not keep track of connection-level information in regular mode, and only keeps track of suspicious and bad connections in vigilant mode, while D-WARD keeps track of agflow and connection information continuously. Furthermore, the added memory consumption in vigilant mode for keeping track of the RWIN for each suspicious connection is insignificant. The memory consumption D-WARD incurs in a traditional network is more than acceptable (and probably acceptable in a smart home environment), but not in constrained IoT networks where the border router, like the devices in the network, is memory-constrained.



(a) D-WARD



(b) D-WARD+

Figure 23. Behavior of a naive attacker under D-WARD and D-WARD+.

5.6.5 Naive TCP Flooding Attack. A *naive* TCP flooding attack is one in which the attacker ignores TCP congestion and flow control. In this subsection, we analyze how D-WARD and D-WARD+ handle a naive attacker.

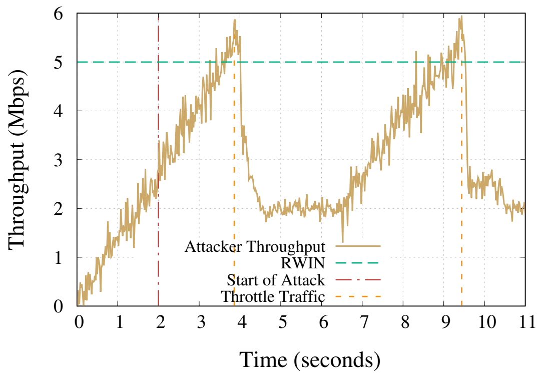
D-WARD classifies connections based on the ratio of the number of packets sent and received, where a ratio that surpasses the maximum threshold indicates an attack. Depending on how the maximum threshold is set, D-WARD may either allow some DDoS traffic to leave the policed network or incorrectly classify benign connections as bad.

In most cases (i.e., not taking into account when significant packet loss is present), when the sender is not overwhelming the receiver, the ratio of the number of packets sent and received stays relatively constant (which we call the normal ratio), regardless of the sending rate. However, as soon as the sending rate begins to overwhelm the receiver (i.e., RWIN is surpassed), the ratio of the number of packets sent and received increases with respect to the sending rate. If the maximum threshold is set below the normal ratio, D-WARD has a devastating impact on benign connections. Namely, whenever a benign connection surpasses the threshold, it essentially throttles continuously causing major collateral damage.

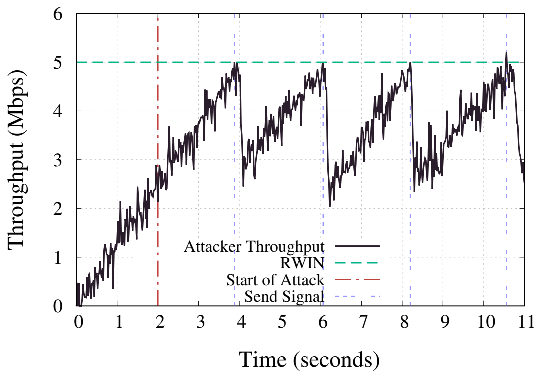
Therefore, D-WARD aims to learn the correct normal ratio so it can set the maximum threshold to be higher. However, the larger the difference between the maximum threshold and the normal threshold, the more time it takes D-WARD to detect and throttle DDoS traffic. Mirkovic et al. set the maximum threshold to 3 by default [93], and we therefore use this value in our evaluation.

Figure 23a shows the behavior of a naive attacker under D-WARD in our 802.11b/g/n Wi-Fi testbed. The results of the Bluetooth LE testbed is relatively similar, and therefore, we only present the results of the 802.11b/g/n network. We start the DDoS attack at 2 seconds. At around 3 seconds, the maximum threshold is surpassed, causing D-WARD to throttle the connection and cut the throughput in half (we set f_{dec} to 1/2). Note, in Figures 23 and 24, the green dotted line represents the maximum throughput the receiver can handle before RWIN is surpassed, which we label as "RWIN" for simplicity. While D-WARD throttles the naive attacker, it still allows a small amount of DDoS traffic to leave the network (DDoS traffic here refers to the traffic that surpasses the victim's RWIN). Specifically, since the maximum threshold is met after RWIN is surpassed, the receiver is under DDoS attack for about 300 milliseconds before the connection is throttled. The attacker continues to send at its maximum sending rate even after it's connection is throttled because it ignores TCP congestion control. At around 3.2 seconds, D-WARD again throttles the connection (this time continuously) and classifies the connection as bad.

Unlike D-WARD, D-WARD+ keeps track of RWIN, thereby avoiding the hassle of choosing a correct maximum threshold for the ratio of number of packets sent and received. Figure 23b shows the behavior of a naive attacker under D-WARD+ (again, in our 802.11b/g/n Wi-Fi testbed). At around 2.7 seconds, D-



(a) D-WARD



(b) D-WARD+

Figure 24. Behavior of a smart attacker under D-WARD and D-WARD+.

WARD+ sends a signal to the attacker (three duplicate ACKs), which the attacker ignores (because it is ignoring TCP congestion control). After a few milliseconds (the time it takes a packet from the attacker to reach the border router), D-WARD+ notices that the attacker is not complying to the signal, labels the connection as bad, and begins throttling. Note that in these few milliseconds, the attacker's throughput can surpass RWIN, but it is immediately throttled.

In summary, both D-WARD and D-WARD+ handle naive attackers similarly. However, D-WARD+ does a slightly better job of preventing the victim from being overwhelmed with traffic that surpasses its RWIN. The differences between D-WARD and D-WARD+ is more noticeable when handling a smart attacker.

5.6.6 Smart TCP Flooding Attack. A *smart* TCP flooding attack is one in which the attacker follows TCP congestion control, but not flow control. In this subsection, we analyze how D-WARD and D-WARD+ handle a smart attacker.

Figure 24a shows the behavior of a smart attacker under D-WARD in our 802.11b/g/n Wi-Fi testbed. The DDoS attack begins at 2 seconds. At around 4 seconds, the maximum threshold for the ratio of the number of packets sent and received is surpassed, causing D-WARD to throttle the connection and cut the throughput in half. Similar to the case of the naive attacker under D-WARD, the receiver is under DDoS attack for about 500 milliseconds before the connection is throttled. However, unlike in the naive attacker scenario, the smart attacker follows TCP congestion control and cuts its window size in half. For the next 2 seconds, D-WARD checks to see if the attacker continues to comply with the rate limit, which, in this case, the attacker does. Once the observation period is over, D-WARD linearly increases the connection's rate limit. At around 9.5 seconds, the connection again surpasses the maximum threshold and is throttled. But again, D-WARD allows DDoS traffic to leave the policed network for about 500 milliseconds. The attacker then cuts its sending rate again in half and complies with the rate limit. This trend continues, allowing the smart attacker to launch a successful periodic DDoS attack on the victim. Also note that if the maximum threshold were higher, the amount and length of the DDoS attack would increase.

Figure 24b shows the behavior of a smart attacker under D-WARD+ (again, in our 802.11b/g/n Wi-Fi testbed). At around 4 seconds, D-WARD+ sends a signal to the attacker, which the smart attacker responds to by cutting its sending rate in half (unlike the naive attacker). However, in this case, unlike D-WARD, no DDoS traffic surpassing RWIN leaves the policed network. The attacker follows TCP congestion control and linearly increases its sending rate (congestion avoidance phase). D-WARD+ again sends the attacker a signal at around 6 seconds. This trend continues. At each peak, D-WARD+ allows the attacker no more than 1/4 RTT (or travel time between attacker and border router) amount of time to send DDoS traffic surpassing RWIN (which, in this case, can be seen on the 4th signal

Table 5. Summary of key findings for the DDoS attack case study.

Metrics	**Key Findings**
Retransmissions	With a large W, D-WARD may require more than 500 times the number of retransmissions than D-WARD+, and with a small W, D-WARD may require more than 10 times the number of retransmissions than D-WARD+.
Connection Duration	When fdec is low and W is high, D-WARD can cause up to 7 times longer connection duration than D-WARD+, and when fdec is high and W is low, connection duration under D-WARD+ is only at most 3 seconds faster than under D-WARD.
Energy Consumption	Under D-WARD, energy consumption increases linearly with respect to window size, while under D-WARD+ energy consumption is static across varying window sizes and contributes to less than 1 mJ of extra energy consumption for a benign device. This leads 15.55 less batteries consumed every year under D-WARD+.
Memory Consumption	On average, D-WARD consumes 10 times more memory consumption than D-WARD+.

– at around 10.5 seconds). This is significantly less than the amount of DDoS traffic that D-WARD allows to leave the network and makes an attack essentially unfeasible. However, to prevent even a minuscule amount of DDoS traffic from leaving the network, D-WARD+ could drop any traffic surpassing RWIN or preemptively send a signal before traffic surpasses RWIN, but such actions could cause a (relatively small) negative impact on benign devices that behave similarly to smart attackers.

5.6.7 Evaluation Summary. The various metrics measured and analyzed for the DDoS attack case study clearly show that the two-mode design of D-WARD+ helps make it more suitable in an IoT environment as compared to its counterpart, D-WARD. Table 5 shows a summary of the key findings for the DDoS attack case study. For D-WARD+, not throttling any connections in regular mode, the signal mechanism in vigilant mode, and only throttling when a connection is labeled bad, allows for the reduction in retransmissions and connection duration, which has the positive side effect of reducing overall energy consumption in the network. Furthermore, only storing limited information in regular mode, allows D-WARD+ to significantly reduce its overall memory consumption, as compared to D-WARD. In conclusion, with D-WARD+ as proof, the TWINKLE framework can transform a security application for the smart home environment, into one that achieves equal to better defense efficacy than classical security applications, while consuming significantly less resources.

5.7 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 classical security applications become ineffective or inapplicable in an IoT environment. Using the smart home as the battleground, this paper proposes and implements a security framework called TWINKLE that endeavors to address a fundamental dilemma facing any security solution for IoT: the solution must 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. As demonstrated by our detailed studies and evaluations in applying TWINKLE to DDoS attacks, we can successfully convert prior solutions to effective but more resource-efficient versions.

CHAPTER VI
DEFENSE INSIDE OF AN IOT NETWORK: MITIGATING ATTACKS CLOSER TO THE DEVICES ORIGINATING DDOS TRAFFIC

In the previous chapter, we make the assumption that a network operator has access to a gateway at which they can deploy a security solution. However, this assumption may not hold true in the real-world for some IoT networks. For example, networks with cellular IoT devices connect to cell sites or base stations that are most likely inaccessible to the owners of the devices. Furthermore, some IoT networks may have multiple gateways that connect them to the Internet, and deploying security solutions on every possible gateway may be infeasible.

Therefore, in this chapter, we focus on defense close to the devices in an IoT network originating DDoS traffic, propose a mobile firewall system, and evaluate our system by analyzing how well it mitigates DDoS traffic generated by infected devices in an IoT network.

The chapter is derived in small part from the following unpublished work: An Open Firewall Ecosystem with On-Demand Security for Internet of Things by Strobel, D.; Mergendahl, S.; Hu, Z.; Supan, M.; Sisodia, D.; Li, J. The content of this chapter was written entirely by me, and I was responsible for conducting all of the presented analyses.

6.1 Introduction

We use the term **open IoT network** to refer to an IoT network that has an inaccessible gateway (or gateways) connecting the network to the Internet. Because we cannot assume access to or rely on a central gateway in an open IoT network, we need to be able to detect and mitigate attacks as close to the devices as possible. However, as opposed to traditional networks, there are several requirements that a security solution must satisfy when deployed (1) in any IoT network, (2) closer to devices, and (3) specifically, in an open IoT network.

Traditional networks are set up as star networks where all nodes within the network are directly connected to a central node, such as a gateway router. However, IoT networks leverage mesh networking, where each node in the network can act as routers and forward traffic to nodes within communication range. Thus, mesh networking introduces blind spots within a network for security solutions that are deployed on a central node, such as gateway-based firewalls. Although, security solutions deployed in IoT networks with a single, accessible gateway may be able to detect and mitigate a majority of network attacks, but as mentioned before, this does not apply to open IoT networks. Therefore, a security solution deployed within an open IoT network must handle more blind spots than traditional networks.

Traditional networks are also fairly static in terms of mobility and churn, making statically distributed security solutions effective. On the other hand, IoT networks are far more dynamic. IoT devices can be mobile, and continuously enter and leave networks. Furthermore, it is common for low-powered IoT devices to duty-cycle to reduce energy consumption, further increasing churn in the network. To handle the mobility and churn of IoT devices, security solutions must also be dynamically distributed, mobile, and provide security on-demand. In other words, a security solution deployed within an IoT network, whether open or not, must handle more dynamic network topologies than traditional networks.

Furthermore, installing security solutions directly on machines that make up traditional networks is far easier than on IoT devices. As mentioned in Chapter V, this is in-part due to the fact that traditional networks tend not to be concerned with resource consumption (e.g., CPU and memory). However, another critical issue is that installing security solutions directly on IoT devices is infeasible in many cases. For example, in some cases, installing new software might require a shutdown of the device, which may not be possible in networks that provide critical, always-on, services. Also, some IoT devices deployed in certain environments may be difficult to physically access or intermittently powered, making new software installation extremely challenging. Therefore, a security solution that needs to be deployed close to the IoT devices must handle security threats without modification of the devices themselves.

Lastly, as an IoT network evolves, new types of devices may join the network. As new devices join a network, new threats and attacks are also introduced into the network. Over time, the number of security solutions to run in order to handle all possible attacks will explode, making it infeasible to rely on a network operator for constantly deciding which security solutions to run given the current state of the network. Therefore, a security solution deployed within any IoT network must handle the dynamic security needs of the network as new threats emerge.

In this chapter, we focus on defense close to the devices in an open IoT network originating DDoS traffic, and propose a security system that satisfies each of the aforementioned requirements. Specifically, we introduce a mobile firewall system with on-demand security, which leverages the software defined networking (SDN) and network function virtualization (NFV) paradigms to publish and subscribe security functionality in an IoT network. This system introduces mobile security nodes that can monitor different parts of the network on-demand, and a security controller that decides which security functions to deploy on the mobile security nodes to verify and mitigate attacks. To show the efficacy of such a system, we analyze how well it can mitigate DDoS traffic generated by infected devices in an open IoT network.

The rest of this chapter is organized as follows. We first survey related work in Section 6.2, then describe the design of our mobile firewall system in Section 6.3. Next, we conduct a preliminary evaluation of our system in Section 6.4, and conclude the chapter in Section 6.5.

6.2 Background and Related Work

We organize the related work into three sections: IoT intrusion detection and mitigation systems, SDN and NFV for Security, and DDoS mitigation at source-end IoT networks.

6.2.1 IoT Intrusion Detection and Mitigation Systems. One of the main purposes of the mobile firewall system is to detect and mitigate intrusions. We therefore begin by taking a look at papers that focus on intrusion detection and mitigation for IoT networks.

In Chapter III we analyzed recent IDSes that tackle several attacks that IoT devices and networks face, such as side-channel attacks, execution of malicious processes, routing attacks, spoofing attacks, voice-command injection attacks, attacks via encrypted traffic analysis, and hidden inter-application interactions. These IDSes can be used as security functions for the mobile firewall system in order to detect and mitigate various attacks.

There have been several surveys on IDSes for IoT throughout the last few decades, and we list a few recent surveys below. Zarpelao et al. [162], Mosenia et al. [96], and Gendreau et al. [55] all surveyed intrusion detection in IoT. Many IDS-based security papers for IoT not only focus on attack detection and mitigation, but also placement of security resources. The careful placement of IDSes is more critical in IoT networks than in traditional networks due to - limited computational/memory/battery capacity at IoT devices, and - device-to-device communication or mesh networking. A recent survey by Chaabouni et al. [27] focused on machine learning-based network IDSes for IoT — a research area still in its infancy. All four surveys conclude that detection and prevention are difficult in the realm of IoT due to the limited computational power of IoT devices.

6.2.2 SDN and NFV for Security. We leverage the SDN and NFV paradigms of centralized decision making and dynamic installation of functions to design the mobile firewall system. We therefore survey the areas of SDN and NFV, and analyze how others in the research community have used these technologies towards enhancing network security [47].

SDN provides various features to a network, and one such feature is traffic isolation. Through the decoupling of the control plane and data plane, SDN allows for controllers to separate and isolate traffic by enforcing routing policies at network switches. For example, Yiakoumis et al. [159] applied this aspect of SDN to smart home networks to facilitate home network management by virtualizing the smart home infrastructure. Specifically, the authors sliced the common physical infrastructure between multiple service providers and users to more easily share the common infrastructure, which also supported many policies and business models for cost sharing. For security purposes, traffic isolation can be used to dynamically separate suspicious or malicious traffic flows from legitimate traffic, and to this end, SDN can offer different levels of network abstractions to appropriately separate traffic.

Another feature SDN provides is centralized visibility which can be used for network monitoring. An SDN controller has wide visibility of the data plane of the switches it controls, and can query those switches for various statistics and status information. The controller can provide these statistics to applications running on the control plane to get a continuously updated view of the underlying network infrastructure. Applications for detecting network-wide anomalies and attacks rely on such information. For example, there are many SDN-based applications for timely DDoS detection [155]. In fact, Mehdi et al. [90] showed that detection algorithms that do not perform well at identifying anomalies at the ISP level, instead perform well at the network edge when leveraging SDN.

Lastly, an important feature of SDN is dynamic flow control, or the capability of a controller to dynamically install and update forwarding rules in switches to manage traffic flows. This feature of SDN allows for a controller to forward traffic to security nodes for closer inspection. For example, Shin et al [123] presented a framework called CloudWatcher, which utilized SDN to dynamically forwards traffic flows to pre-installed network security devices so that all packets flowing through the network were inspected. Shin et al. [124] introduced NetSecVisor, which is an SDN-based system that determines optimal routing paths based on policy requirements to maximize the utilization of pre-installed and fixed-location security devices. Furthermore, Yoon et al. [160] presented various SDN-based security functions, such as firewalls and IDSes/IPSes, that are possible due to the increased network manageability provided by dynamic flow control. Lallo et al. [43] proposed an SDN-based architecture that allowed an Industrial Control System (ICS) operator to replicate and forward sensitive traffic flows to strategically placed IDSes to maximize the number of analyzed flows. This architecture enabled the use of spare bandwidth in the network to forward the replicated traffic while avoiding packet loss of production traffic. Note, due to dynamic flow control and the increased ease of deploying traffic filtering rules, SDN has led to a plethora of DDoS filtering solutions [116, 145, 48, 87, 108, 167, 44].

In addition to the aforementioned features, SDN provides a platform for NFV. NFV enables the use of commodity servers for deploying virtual network functions (VNFs), including security functions. Due to the various constraints and heterogeneity of IoT devices and networks, offloading security functionality to virtualized middleboxes can be extremely useful to IoT networks. For example, Yu et al. [161] presented IoTSec, a security architecture for IoT networks that allowed rapid instantiation of customized micro-middleboxes running various security functions. In fact, Boudi et al. [18] analyzed the feasibility of container-based security solutions on resource-constrained edge nodes, concluded that container-based security functions have extremely low overhead compared to native execution of security functions, and therefore supported the provisioning of VNFs even in constrained IoT environments.

A powerful feature enabled by SDN and NFV is VNF chaining. Through VNF chaining, traffic can be processed through chains of virtualized middleboxes, which can drastically reduce operational costs and improve resource utilization. An example of leveraging VNF chaining to secure IoT networks is Securebox [59]. Securebox provides a combination of on-demand security and management services, through traffic analysis and device state configuration. Such a system allows for flexible onloading and offloading of security functions to IoT devices, and easier deployment of new security functions as new security threats emerge.

6.2.3 DDoS Mitigation at Source-End IoT Networks.

A main focus of this chapter is to leverage the mobile firewall system to mitigate DDoS traffic originating from IoT devices within an open IoT network. In Chapter V we detail D-WARD, a source-end DDoS defense solution for traditional end-hosts on the Internet, but show that several drawbacks may arise when deploying it in an IoT environment. We therefore analyze several papers related to source-end DDoS mitigation and prevention specifically for IoT networks.

After the high-profile DDoS attacks caused by the Mirai IoT botnet in 2016, the National Institute of Standards and Technology (NIST) introduced the following objectives: (1) reduce the vulnerability of IoT devices from being turned into bots and (2) limit the utility of compromised IoT devices to malicious actors, and presented primary technical elements to achieve this objective [105]. Some of the technical elements the NIST proposed were network gateways/routers supporting wired and wireless network access, Manufacturer Usage Description (MUD) Specification controllers and file servers (i.e., a standard defined by the IETF, or Internet Engineering Task Force, for embedded software that allows IoT device manufacturers to advertise device specifications and intended communication patterns when devices are connected to a network), Dynamic Host Configuration Protocol (DHCP) and update servers, and threat signaling servers, among other elements. While such elements would not completely eliminate IoT devices from being compromised, but the NIST argued that they would significantly increase the effort required by malicious actors to compromise devices in home or small-business IoT networks.

In the last decade, researchers have turned to relatively newer technologies such as SDN and blockchain to address the problem of IoT-based DDoS attacks.

In fact, there have been many papers that leverage SDN to mitigate DDoS attacks at source-end IoT networks [41]. One such solution, presented by Yang et al. [157], utilizes a distributed network of SDN-based IoT gateways running OpenFlow with detection and mitigation functions for distributed DDoS detection and mitigation in real-time. Javaid et al. [72] integrated IoT networks with blockchain by using smart contracts to replace the traditional centralized IoT infrastructure with a decentralized one. Within this decentralized infrastructure, IoT devices are required to access the network using smart contracts, which prevents compromised devices from launching DDoS attacks by using static resource allocation for devices in the network.

The main drawback to each of these solutions is that they make large assumptions about the infrastructure of the IoT networks that they are trying to protect. Most existing IoT networks cannot utilize SDN and blockchain technologies as is, and therefore, these solutions will most likely not be widely-adopted in the real-world. While the mobile firewall system does leverage the SDN and NFV paradigms of centralized decision making and dynamic installation of functions, it does not require any major modifications of the network infrastructure, unlike the aforementioned solutions.

6.3 Mobile Firewall System Design

In this section, we detail the components and modules that make up the mobile firewall system, explain how we leverage the SDN and NFV design paradigms, and discuss various extensions to the mobile firewall system.

6.3.1 System Components.

The two main components are the **mobile security node** and the **security controller**. At a high level, the mobile security node is responsible for 1) monitoring the network by collecting network

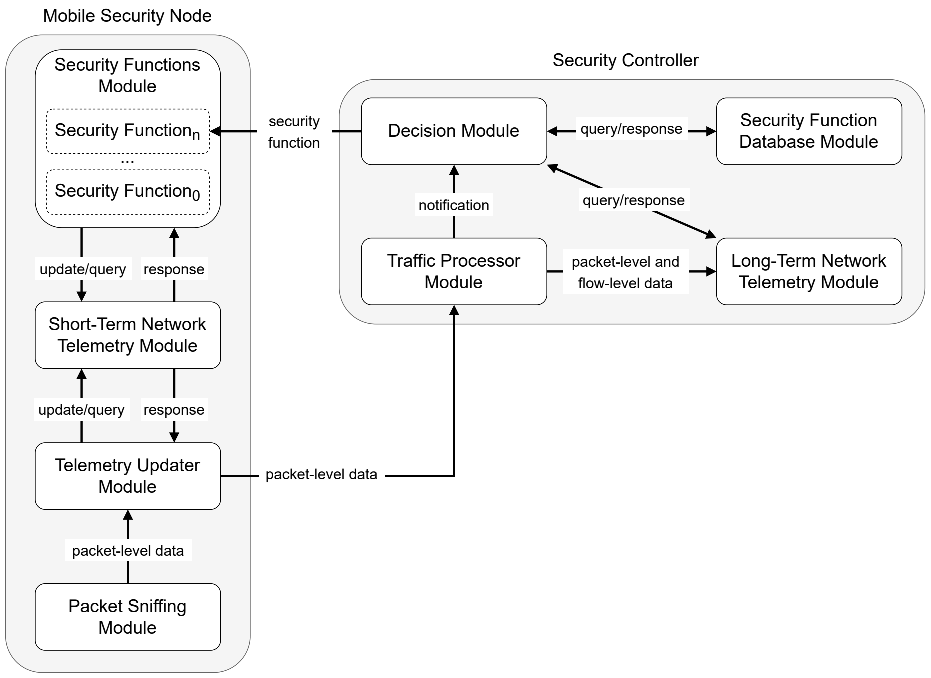


Figure 25. Diagram of the mobile firewall system.

telemetry information, and 2) running security functions to verify and mitigate attacks. Based on the network telemetry information provided by the mobile security node, the main responsibility of the security controller is to decide which security functions to deploy on the mobile security node. Note, as we explain later, the mobile firewall system can include multiple mobile security nodes and even multiple security controllers, but for simplicity, we describe how the system functions with a single mobile security node and security controller. Figure 25 depicts a diagram of the mobile firewall system, including each component and module.

6.3.1.1 Mobile Security Node. We begin by detailing each module in the mobile security node and the interactions between modules.

Packet Sniffer Module: In order to monitor the network without needing to install any software on individual IoT devices in the network, the mobile security node employs packet sniffing via the packet sniffing module. Packet sniffing allows the mobile security node to collect and log packets that are received on a specific wireless channel, regardless of how the packets are addressed. In order to do so, the mobile security node must have a wireless network interface controller (WNIC) that supports RFMON (Radio Frequency MONitor) mode, also known as monitor mode for short [1]. Due to the heterogeneity of IoT networks as it pertains to physical layer protocols and the fact that most sniffers can only monitor one channel at a time, a mobile security node may need to use multiple WNICs to monitor all traffic within an IoT network.

The packet sniffer module collects *sampled* packet-level information (i.e., as PCAP files) from the network. Captured packets may include management, control, and data frames, all of which may be important in detecting suspicious behavior in the network. The packet sniffer module sends the packet-level information it collects to the telemetry updater module.

Telemetry Updater Module: The telemetry updater module serves two main purposes. First, it receives information from the packet sniffer module and updates the information stored in the short-term network telemetry module. Second, when the mobile security node relays telemetry information to the security controller, the telemetry updater module queries the short-term network telemetry module for the latest telemetry information and sends that information to the security controller's traffic processor module.

Short-Term Network Telemetry Module: The short-term network telemetry module stores all of the sampled packet-level information collected by the packet sniffer module. Because the mobile security node itself may be a resource-constrained device, it can only store a limited amount of information. Therefore, once the telemetry updater module transfers telemetry information to the security controller, it deletes the information stored in the short-term network telemetry module. As a result, the short-term network telemetry module only stores a limited, but up-to-date snapshot of the network at any given time.

Security Functions Module: Finally, the security functions module stores and executes security functions that the mobile security node needs to run. After analyzing the network telemetry information, the security controller may publish security functions to the mobile security node to verify and mitigate attacks. Because security functions need access to up-to-date network telemetry information, the security functions module can query the short-term network telemetry module whenever necessary. Note, multiple security functions can be executed as separate threads within the security functions module.

6.3.1.2 Security Controller. Now, we detail each module and their interactions in the security controller.

Traffic Processor Module: The main function of the traffic processing module is to process the packet-level information received from the mobile security node's telemetry updater module into flow-level information. It can also convert the PCAP files it receives into more easily parsable formats, such as sFlow. It does so by running a utility called sflowtool on the PCAP files it receives, which converts the PCAP files into sFlow and NetFlow files. The traffic processor module stores both the packet-level information and flow-level information into the long-term network telemetry module. Note, it is important to keep both levels of granularity of information because, depending on the attacks in question, both may be needed in order to detect suspicious behavior.

Long-Term Network Telemetry Module: Like the mobile security node's short-term network telemetry module, the long-term network telemetry module provides a view of the network by storing packet-level and flow-level information. Note, because the information is derived from sampled data, the view of the network provided by the long-term network telemetry module is not a complete one, but should still be sufficient for detecting attacks. Unlike the mobile security node, we assume that the security controller is not resource-constrained, and therefore is able to keep a more long-term view of the network.

Security Function Database Module: Security functions are stored in the security function database module. As explained in Chapter V, security functions can be represented as files containing parameters, source code, and compilation instructions. When the *decision module* decides that a security function needs to be deployed on the mobile security node, it will query the security function database module for the given function.

Decision Module: The decision module is responsible for deciding which security function(s) to run, given the network telemetry information, and publishing those functions onto the mobile security node. Figure 26 shows a detailed diagram of the decision module. The decision module contains a detection initiation thread, attack detection thread(s), and a security function selection thread. After the traffic processor module processes new telemetry information from the mobile security node, it notifies the decision module's detection initiation thread, which in turn queries the long-term network telemetry module. The detection initiation thread initiates each attack detection thread and provides them with network telemetry information. Each attack detection thread attempts to detect a particular attack by taking in network telemetry information as input,

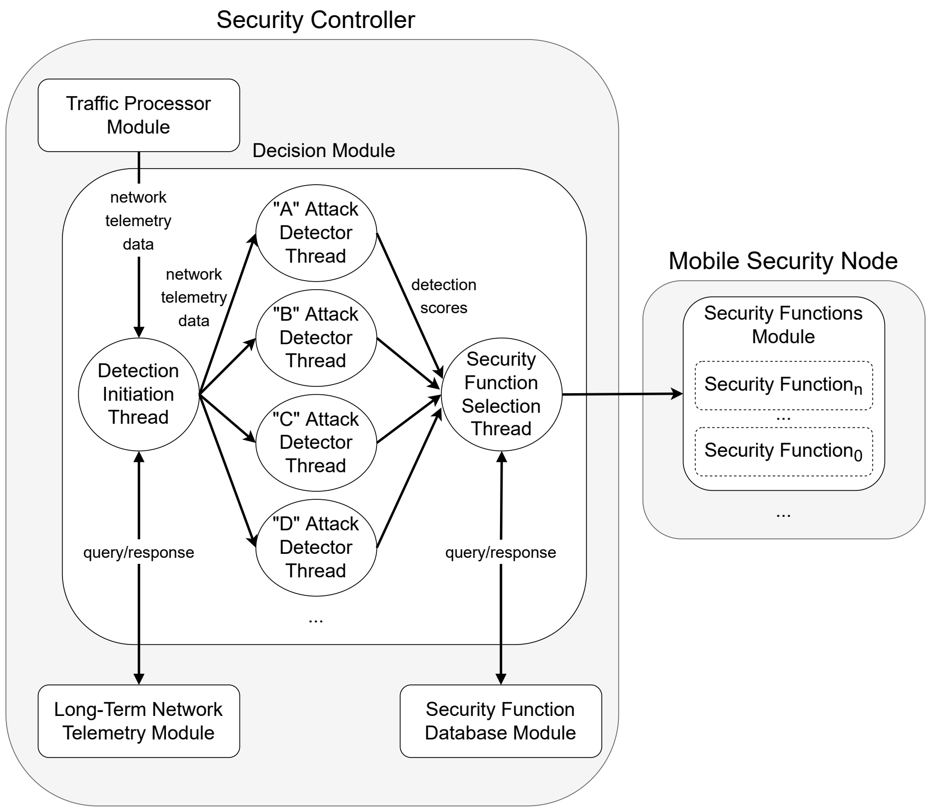

Security Controller
Traffic Processor Module
Decision Module
network telemetry data
network telemetry data
"A" Attack Detector Thread
"B" Attack Detector Thread
"C" Attack Detector Thread
"D" Attack Detector Thread
...
detection scores
Detection Initiation Thread
Security Function Selection Thread
query/response
query/response
Long-Term Network Telemetry Module
Security Function Database Module
Mobile Security Node
Security Functions Module
Security Function$_n$
...
Security Function$_0$
...

Figure 26. Diagram of the security controller's decision module.

and outputting a *detection score*, which represents how confident the detector is that the particular attack is occurring. The detection scores are fed to the security function selection thread, which, given the detection scores, decides which security functions should run in the network, queries the security function database module for those functions, and publishes those functions to the mobile security node's security functions module.

There are several key details worth mentioning about the decision module. First, the decision module can add and remove security functions to and from the mobile security node on-demand. Specifically, once an attack is no longer present in the network (i.e., the detection score for a particular attack is low), the security function selection thread removes all security functions pertaining to that attack from the mobile security node's security functions module. Second, a network operator can deploy on-demand a detector into the decision module and security functions into the security function database module for each attack that they want to defend their network against. This is similar to how SBDRs and SBHRs can be added to the TWINKLE framework on-demand, as explained in Chapter V. Third, a network operator can manually create a mapping within the security function selection thread between the detection scores and the security functions to deploy, and adjust this mapping based on the needs of the network. However, this mapping and decision process can also be learned over time using unsupervised learning methods, thereby requiring little to no human intervention.

6.3.2 Leveraging the SDN and NFV Design Paradigms. Due to the limited resources and heterogeneity of IoT networks, the enforcement of appropriate security and privacy measures is challenging. For the most part, traditional security solutions either cannot contend with the constrained resources on devices, or cannot cope with the increasing attack vectors against IoT networks. However, by leveraging the SDN and NFV design paradigms, security solutions can more easily be applied to IoT networks. Specifically, the SDN paradigm of offloading processing and decision making required by gateway nodes or IoT devices to a central controller solves the problem of having to deal with limited resources, and ensures scalability to support more traffic, devices, and functionalities. The NFV paradigm of dynamic allocation of functions, security can be scaled up or down according to the needs of the network, thereby allowing a network operator to more easily cope with the ever-increasing security threats facing IoT networks.

To show how the mobile firewall system leverages the SDN and NFV design paradigms, we compare its design to the design of the TWINKLE framework. First, in order to detect attacks close to the devices in an open IoT network, TWINKLE must rely on devices (either regular devices already in the network or dedicated devices) running the watchdog component to detect suspicious behaviors. However, in the mobile firewall system, we move the detection process to the security controller, which has far more resources than the watchdog nodes. This allows us to scale up the number of attacks the system can detect since it is not constrained by the resources of individual IoT devices. Furthermore, a system can achieve more accurate detection when detection occurs at a central location, where it has access to a more holistic view of the network. Second, in order to mitigate attacks close to the devices in an open IoT network, TWINKLE must rely on devices running the security engine component, in the same way the mobile firewall system must rely on the mobile security node to run security functions. However, in TWINKLE, the installation of SBHRs on security engine nodes is not done on-demand. Whenever a network operator wants to handle a new attack that may occur sometime in the future, they install an SBHR for that attack on each security engine node, which will only be used when that attack is detected. The SBHR will remain on the security engine node even after the attack is mitigated. This design unfortunately does not scale. While an IoT network may face a plethora of different types of attacks over its lifetime, it is safe to assume that it will most likely face only a handful at most at at any given time. Therefore, in the mobile firewall system, the decision module installs security functions on the mobile security node on-demand, better utilizing the resources on the mobile security node and allowing the system to handle far more types of attacks in the long run.

6.3.3 Running Security Applications. Similar to TWINKLE, security applications should be able to be plugged into the mobile security system with relative ease. Let us take SWORD and D-WARD+ as example security applications, and show how they may run on the mobile security system.

Assuming a network operator wants to prevent worm traffic from leaving their network, SWORD can be used to detect outgoing worm traffic. Note, SWORD requires flow-level information in order to detect worm traffic. By default, the mobile security node will collect packet-level network telemetry data, which will eventually be converted into flow-level data by the security controller's traffic processor module. The decision module will then take the flow-level data as input and run SWORD to detect worm traffic. If worm traffic is detected, it will publish a security function to drop the traffic from infected devices.

D-WARD+ can be used to detect, verify, and mitigate DDoS traffic leaving an open IoT network using the mobile firewall system, which it would not be able to do comprehensively if deployed on a single gateway router in the open IoT network. D-WARD+ also requires flow-level information to detect DDoS traffic. Therefore, the decision module will take the flow-level data as input and run D-WARD+ to detect suspicious and attack connections. It will then publish a security function which leverages the fast retransmit mechanism in TCP congestion control to verify that the suspicious connections are indeed attack connections. Furthermore, any device initiating attack connections will have its traffic dropped.

Note, each security function must be lightweight enough to run on the potentially resource-constrained mobile security node. The resources under consideration not only include the CPU and memory consumption of the functions, but the power consumption required to transmit and receive packets. Ideally, the mobile security node can remain in sleep mode most of the time, and wake when needed. However, it is difficult to know when such transitions should happen, given the fact that an attack can occur at any given moment.

6.3.4 Extensions.

6.3.4.1 Security controller in the cloud. In order to further increase the system's scalability, the security controller can be deployed in the cloud, where it has access to far more resources. This would allow a security controller to control several different IoT networks simultaneously, and make sharing of resources and information between security controllers easier. However, an issue to consider with this approach is the latency between the mobile security node and the security controller.

6.3.4.2 Multiple mobile security nodes. The mobile security system can be extended to support multiple mobile security nodes. This extension would include a mobile node status module, which stores information, such as IP address, listening port, architecture, board, operating system information, and memory capacity, about each mobile security node, and the security functions currently running on each node. Whenever a mobile security node joins the system, it subscribes to the security controller by sharing its information with the security controller. The security function selection thread will update the mobile node status module whenever it publishes a security function to a mobile security nodes. This allows the system to balance utilization of the cumulative resources available on the mobile security nodes.

Furthermore, having multiple mobile security nodes allows for the chaining of security functions. For example, if multiple security functions need to be executed to verify and mitigate a particular attack, instead of deploying all of the functions on a single mobile security node, the system can deploy each function on a separate node. The functions can share information as the output of one function may be required as input to another, thereby creating a chain between the mobile security nodes.

6.3.4.3 Crowdsourcing. Throughout this chapter we discussed the use of mobile IoT devices, as mobile security nodes, dedicated solely for security purposes. However, the software that makes up the mobile security node can be installed on most types of hardware, such as a small Raspberry Pi Zero W, and a network operator can attach this hardware to devices in the network, such as a automated vacuum cleaner. Users of the network can even volunteer to use their own everyday objects, such as a coffee cup, as mobile security nodes. This way the system can leverage the power of crowdsourcing to collectively create a mobile firewall ecosystem. In this approach, the system loses complete control over the mobile security nodes, but gains larger coverage, potentially leading to a more secure network.

6.4 Evaluation

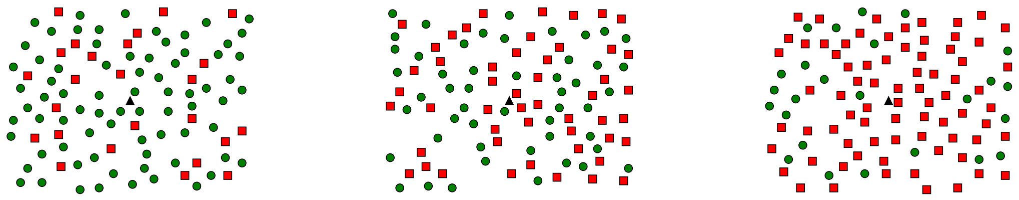
In this section, we evaluate the efficacy of the mobile firewall system at mitigating DDoS traffic generated in an open IoT network. We begin by providing a brief overview of the procedure, explain the key metric and parameters, and detail the evaluation environment. We then present our results.

6.4.1 Procedure Overview. The goal of this evaluation is to provide *initial* results on the efficacy of the mobile firewall system in a specific case study: mitigating DDoS traffic as close to the devices as possible in an open IoT network generating DDoS traffic towards a victim on the Internet. To do so, we simulate an open IoT network, where a set of nodes are infected and sending DDoS traffic out of the network. We also simulate a single mobile security node inside of the network which is running a security function to drop the DDoS traffic. Since we are only focused on mitigating the attack, we assume that the mobile security node already knows which devices are infected. As mentioned in Chapter VII, we plan on conducting a more comprehensive study on the entire mobile firewall system in the future.

6.4.2 Key metric and parameters. The key metric we focus on in this evaluation is **mitigation efficacy**. We show mitigation efficacy in terms of the amount of DDoS traffic the system drops over the evaluation period. We also study the amount of DDoS traffic the system allows to leave the network over time.

The three main parameters in this evaluation are the *percentage of infected devices*, *distance traveled* by the mobile security node, and *time*. We also vary the *speed of the mobile security node* as an additional parameter. As we show later, all aforementioned parameters have a direct impact on the system's mitigation efficacy.

6.4.3 Evaluation Environment. In this evaluation, we simulate a 6LoWPAN open IoT network. 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 [151]. 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



(a) Infection rate of 25%   (b) Infection rate of 50%   (c) Infection rate of 75%

Figure 27. Three example networks with different infection rates.

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 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, creates its set of parents, chooses the preferred parent, and calculates its own rank (which is greater than the rank of each of its parents).

Our IoT network consists of 100 nodes evenly distributed over a 100m x 100m area. The network uses IEEE 802.15.4 at the physical layer. Depending on the infection rate, we select a set of nodes to be infected uniformaly and at random. These nodes launch a direct UDP flooding attack towards a single victim outside of the IoT network. Figure 27a, figure 27b, and figure 27c show three different example networks, each with varying infection rates (25%, 50%, and 75%, respectively), where the black triangle represents the mobile security node, the green circles represent benign nodes, and the red squares represent malicious nodes. The entire simulation is implemented in Python, and we utilize Scapy's 6LoWPAN library [16] to implement 6LoWPAN. We also utilize SimpleRPL, which is a Linux-based implementation of RPL as defined in RFC 6550 [151] from the

National Institute of Standards and Technology (NIST) to implement RPL in simulations [30].

It is important to understand how the security function running on the mobile security node mitigates the DDoS attack. First, it uses Dijkstra's algorithm to find the shortest path between malicious nodes. Once it reaches within transmission range of a malicious node, in order to drop the packets generated by that node, it must first get the malicious node to route its traffic to the mobile security node. It does so by sending a DIO message to the malicious node, advertising the lowest rank possible, which will in turn cause the malicious node to treat the mobile security node as its parent, thereby routing its traffic through the mobile security node. The mobile security node can then simply drop all of the traffic that the malicious node sends it.

Note, in Chapter V, we argue that benign devices could suffer significantly from the loss caused by a security solution dropping its traffic, and therefore instead leverage the fast retransmit mechanism (for TCP-based attacks) in D-WARD+. However, since we are operating under the assumption that the system has already verified the devices that are malicious, we are not concerned with dropping traffic from legitimate devices. Furthermore, using the fast retransmit mechanism to reduce the sending rate of malicious devices allows more DDoS traffic to leave the network, as opposed to dropping the traffic outright.

6.4.4 Mitigation Efficacy Results. We begin by analyzing Figure 28, which shows the volume of DDoS traffic allowed (in terms of Mb) with respect to the percentage of devices infected, or infection rate. For these results, the speed of the mobile security node is set to 10m/s, which is approximately the top speed for most commercial drones. Clearly, as the percentage of infected devices

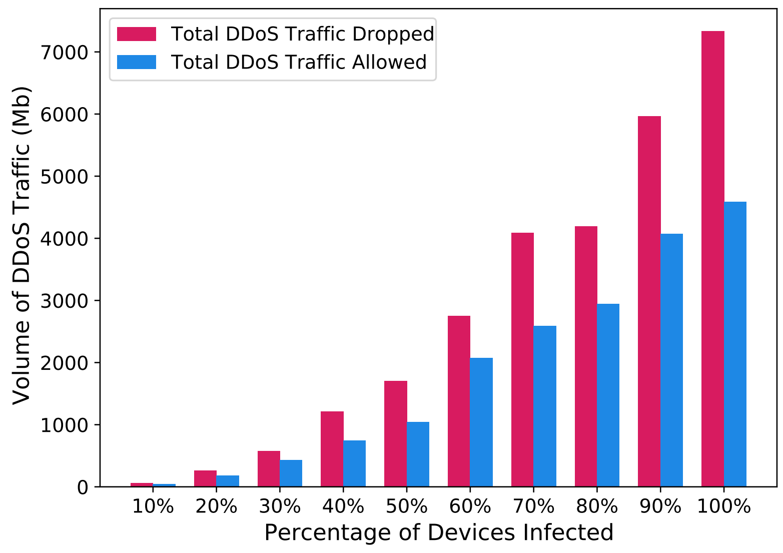
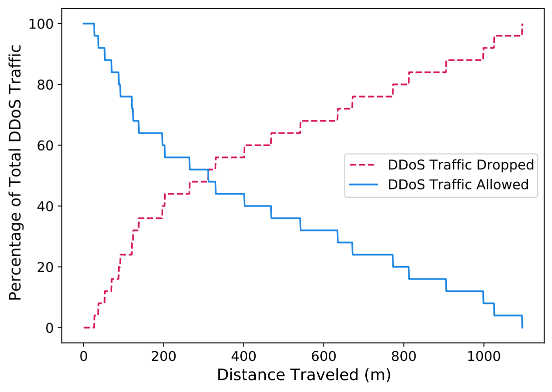


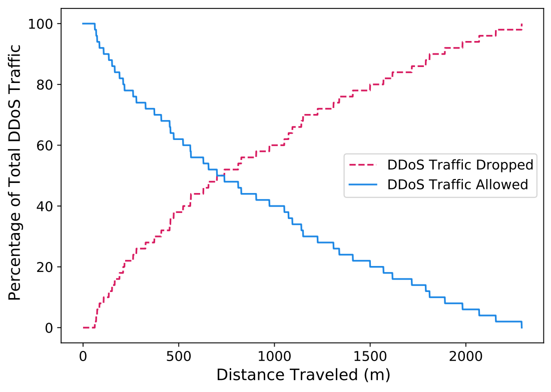
Figure 28. Volume of DDoS traffic allowed with respect to the percentage of devices infected.

increases, so does the amount of DDoS traffic dropped and the amount of DDoS traffic allowed. However, in all circumstances, the amount of dropped DDoS traffic is more than the amount of DDoS traffic allowed. In fact, as the infection rate increases, so does the ratio of dropped to allowed DDoS traffic. The reason why the system allows some DDoS traffic to leave the network is because the mobile security node must travel within transmission radius of each malicious node before it can drop its traffic, which takes time, depending on how fast it is traveling.

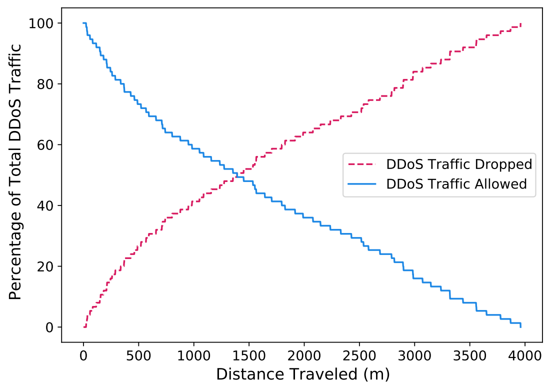
To take a closer look at how the distance traveled by a mobile security node affects mitigation efficacy, we analyze Figure 29 next, which shows the percentage of total DDoS traffic generated by the network (i.e., both dropped and allowed) with respect to the cumulative distance traveled by the mobile security node, when traveling at a speed of 10m/s. From Figure 29a, when 25% of the network is infected, we can observe that the mobile security node must travel a total of approximately 1100m in order to completely mitigate the attack. From Figure 29b,



(a) Infection rate of 25%



(b) Infection rate of 50%

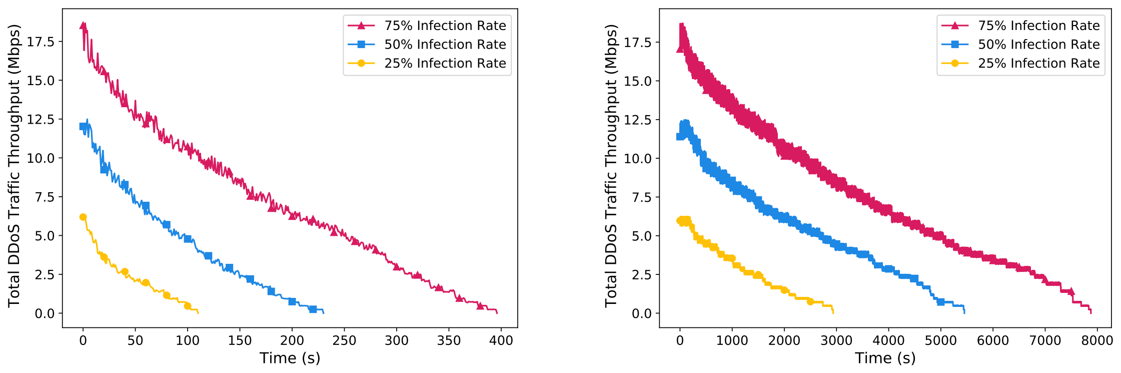


(c) Infection rate of 75%

Figure 29. Percentage of total DDoS traffic generated by the network with respect to the cumulative distance traveled by the mobile security node.

when 50% of the network is infected, and from Figure 29c, when 75% of the network is infected, the mobile security node must travel approximately 2400m and 4000m to mitigate the attack, respectively. From all three graphs, we see a logarithmic growth in the percentage of DDoS traffic dropped and an exponential decay in the percentage of DDoS traffic allowed.

Lastly, we analyze the time it takes to mitigate an attack, by presenting Figure 30, which shows the total DDoS traffic throughput (in terms of Mbps) with respect to time (in seconds) for different mobile security node speeds. Figure 30a shows how the total DDoS traffic throughput changes over time when the mobile



(a) Mobile security node with a speed of 10m/s

(b) Mobile security node with a speed of 0.4m/s

Figure 30. Total DDoS traffic throughput (in terms of Mbps) with respect to time.

security node is traveling at a speed of 10m/s. We can clearly see that the infection rate has a direct effect on the amount of time it takes the mobile security node to mitigate the attack. On the other hand, Figure 30b shows the total DDoS traffic throughput over time when the mobile security node is traveling at a speed of 0.4m/s, which is approximately the top speed for most automated vacuum cleaners. As expected, at a significantly slower speed, the single mobile security node takes a significantly longer amount of time to mitigate the attack. These results show that either the speed of a device should be considered when choosing an appropriate mobile security node, or that enough mobile security nodes should be present in the network, in order reduce mitigation latency.

6.5 Conclusion

In open IoT networks, or IoT networks that have inaccessible gateways connecting them to the Internet, security solutions cannot rely on a central location, where all the network traffic passes through, to be deployed at. Instead, such networks require security solutions to defend against attacks as close to the devices as possible. However, there are several challenges a security solution faces when handling attacks close to the devices, such as dealing with device heterogeneity, limited resources on devices, and the infeasibility of modifying devices, while trying to tackle the ever-growing and changing IoT attack landscape.

In this chapter, we present the mobile firewall system, which leverages the SDN/NFV design paradigm to publish and subscribe security functionality in an open IoT network. This system consists of mobile security nodes that monitor the network on-demand, and a security controller that decides which security functions to deploy on the mobile security nodes. In order to detect and mitigate DDoS traffic generated by an IoT network, the mobile firewall does not need to modify devices in the network, and therefore has no negative impact on device resources. Furthermore, security functions can be installed on-demand to handle new security threats as they emerge.

Through simulation, we show the efficacy of the mobile firewall system at mitigating DDoS traffic originating in an open IoT network. Specifically, we measure the amount of DDoS traffic dropped and the amount of DDoS traffic allowed, while varying several parameters, such as the network's infection rate. Overall, the mobile firewall system is able to successfully mitigate the DDoS attack, regardless of the network's infection rate.

The mobile firewall system is an ongoing project that we will continue to work on in the future. We plan on eventually implementing a fully functioning prototype in a real IoT network.

CHAPTER VII

FUTURE WORK

In this chapter, we outline several possible future directions for research presented in this dissertation.

7.1 Future Work Related to SWORD

There are several open issues related to SWORD that warrant future work. First, as described in Chapter 4.3, a distributed version of SWORD can be studied, where multiple SWORD instances are distributed throughout the network to collectively monitor traffic. Further, in evaluating SWORD against inbound worms, there are several Mirai worm variants [11], and SWORD can be evaluated against these variants. Finally, SWORD could be deployed and evaluated in a real-world network on the Internet to verify the findings in this dissertation.

7.2 Future Work Related to TWINKLE

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 security manager and security engine 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.

Another issue is the feasibility of running event watchdog code on devices in the smart home. We assume that an event watchdog device can run various lightweight processes. However, some devices, such as legacy and extremely resource-constrained devices, may not have the ability to install even lightweight processes. Therefore, in some cases, additional devices need to be added to the network to act as event watchdogs. Furthermore, event watchdogs are required to have enough resources to run the lightweight SBDRs. Again, the feasibility of running SBDRs is dependent on the security application.

The placement of event watchdog nodes is another important issue that a network administrator must consider. Event watchdog nodes may be placed in multiple locations in the network in scenarios where suspicious behaviors cannot be detected at a central location, such as the border router. In such scenarios, selecting an effective placement strategy for the event watchdog nodes is paramount to effectively detect and mitigate a potential attack. However, selecting an effective placement strategy is not trivial. This event watchdog placement problem can be represented as a vertex-cover problem, where the constraint is the number of devices that can run the event watchdog code plus the number of event watchdog specific nodes that the network administrator can add into the network, and the objective is to maximize the number of nodes that are within transmission range of an event watchdog node.

In future work, we plan on studying, and eventually addressing, the aforementioned issues.

7.3 Future Work Related to The Mobile Firewall System

As we mentioned in Chapter 6.5, the mobile firewall system is still in its early stage, and is an ongoing project that we will continue to work on in the future. In the near future, we plan on implementing a fully functioning prototype, and test it in a real IoT network.

Additionally, in Chapter 6.3, we detailed several extensions that could improve the scalability, coverage, and ultimately, the efficacy of the mobile firewall system. Specifically, these extensions include leveraging cloud computing by deploying security controllers in the cloud, utilizing multiple mobile security nodes, and embracing the power of crowdsourcing to collectively create a mobile firewall ecosystem. We plan on thoroughly investigating each extension, and potentially apply these extensions to the prototype in future work.

CHAPTER VIII

CONCLUSIONS

As the number of internet-connected devices increases over time, so too does the number and scale of IoT-enabled DDoS attacks. Unfortunately, the lack of attention paid to the ever-growing problem of IoT-enabled DDoS by the IoT security community has lead to a lack of network-based solutions targeted directly for IoT networks to address IoT-enabled DDoS.

In this dissertation, we tackled the problem of IoT-enabled DDoS attacks. We presented three complimentary and inherently connected network-based prevention, detection, and mitigation approaches against IoT-enabled DDoS at three critical vantage points on the Internet. Furthermore, we showed each approach's effectiveness through extensive evaluation. With this dissertation, we thus provide a more holistic, defense in depth approach to handling the ever-growing threat of IoT-enabled DDoS.

Specifically, in Chapter IV, we presented a worm detector called SWORD, which we showed helps prevent IoT devices from being turned into DDoS bots by quickly detecting the presence of worm traffic towards an IoT network at an upstream ISP/IXP. Then, in Chapter V, we presented a security framework called TWINKLE, which efficiently detected DDoS traffic leaving an IoT network by inspecting traffic at its gateway. Lastly, in Chapter VI, we presented the mobile firewall system, which has the ability to publish security functionalities in an IoT network on-demand without needing to modify the devices in the network, and showed that it can effectively mitigate DDoS attacks as close to the devices in an IoT network originating DDoS traffic.

Ultimately, we hope that this dissertation can provide new insights into the area of IoT security, and serve as a basis to future work hoping to comprehensively tackle the problem of IoT-enabled DDoS.

BIBLIOGRAPHY

[1] RFMON: Radio Frequency Monitoring, Monitor Mode. `https://wlanbook.wordpress.com/2008/01/03/rfmon-monitor-mode/`.

[2] FRGP Continuous Flow Dataset, IMPACT ID: USC-LANDER/FRGPContinuousFlowData-20090729/rev3998. USC/LANDER. `http://www.isi.edu/ant/lander`, 2016.

[3] FRGP Continuous Flow Dataset, IMPACT ID: USC-LANDER/Mirai-FRGP-scanning-20160908/rev10326. USC/LANDER. `http://www.isi.edu/ant/lander`, 2016.

[4] Satori IoT Botnet Variant. `https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/satori-iot-botnet/`, 2018.

[5] Snort - Network Intrusion Detection & Prevention System. `https://www.snort.org/`, 2019.

[6] Suricata - Open Source IDS / IPS / NSM engine. `https://suricata-ids.org/`, 2019.

[7] Abduvaliyev, A., Pathan, A.-S. K., Zhou, J., Roman, R., and Wong, W.-C. On the vital areas of intrusion detection systems in wireless sensor networks. vol. 15, IEEE, pp. 1223–1237.

[8] Abie, H., and Balasingham, I. Risk-based adaptive security for smart iot in ehealth. In *Proceedings of the 7th International Conference on Body Area Networks* (2012), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 269–275.

[9] Alrawi, O., Lever, C., Antonakakis, M., and Monrose, F. SoK: Security Evaluation of Home-Based IoT Deployments. *IEEE Symposium on Security and Privacy (S&P)* (2019).

[10] Alrawi, O., Lever, C., et al. SOK: Security Evaluation of Home-Based IoT Deployments. In *Symposium on Security & Privacy* (2019).

[11] Alrawi, O., Lever, C., Valakuzhy, K., et al. The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle. In *USENIX Security Symposium* (2021).

[12] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Arbor, A., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Arbor, A., Invernizzi, L., Kallitsis, M., Network, M., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., Zhou, Y., Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the Mirai Botnet. *USENIX Security Symposium* (2017).

[13] Apthorpe, N., Reisman, D., Sundaresan, S., Narayanan, A., and Feamster, N. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. *arXiv preprint arXiv:1708.05044* (2017).

[14] Bastys, I., Balliu, M., and Sabelfeld, A. If This Then What? Controlling Flows in IoT Apps. *ACM Conference on Computer and Communications Security (CCS)* (2018).

[15] Bernabe, J. B., Hernandez, J. L., Moreno, M. V., and Gomez, A. F. S. Privacy-preserving security framework for a social-aware internet of things. In *International Conference on Ubiquitous Computing and Ambient Intelligence* (2014), Springer, pp. 408–415.

[16] Bernardini, C. A., and Potter, G. 6lowpan protocol stack. `https://github.com/secdev/scapy/blob/master/scapy/layers/sixlowpan.py`, 2020.

[17] Bilge, L., Balzarotti, D., et al. Disclosure: Detecting Botnet Command and Control Servers through Large-scale Netflow Analysis. In *Annual Computer Security Applications Conference* (2012).

[18] Boudi, A., Farris, I., Bagaa, M., and Taleb, T. Assessing Lightweight Virtualization for Security-as-a-Service at the Network Edge. *IEICE Transactions on Communications* (2019).

[19] Breitenbacher, D., Homoliak, I., Aung, Y. L., Tippenhauer, N. O., and Elovici, Y. HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2019).

[20] Cashdollar, L. LATEST ECHOBOT: 26 INFECTION VECTORS. `https://blogs.akamai.com/sitr/2019/06/latest-echobot-26-infection-vectors.html`, 2019.

[21] Celik, Z. B., Babun, L., Sikder, A. K., Aksu, H., Sikder, A. K., Tan, G., Mcdaniel, P., and Uluagac, A. S. Sensitive Information Tracking in Commodity IoT. *USENIX Security Symposium* (2018).

[22] Celik, Z. B., Fernandes, E., Pauley, E., Tan, G., and McDaniel, P. Program analysis of commodity iot applications for security and privacy: Challenges and opportunities.

[23] Celik, Z. B., Tan, G., and McDaniel, P. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. *Network and Distributed System Security Symposium (NDSS)* (2019).

[24] Celik, Z. B., Tan, G., and McDaniel, P. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *Network and Distributed System Security Symposium (NDSS)* (2019).

[25] Cervantes, C., Poplade, D., Nogueira, M., and 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* (2015), IEEE, pp. 606–611.

[26] Cetin, O., Ganan, C., Altena, L., Kasama, T., Inoue, D., Tamiya, K., Tie, Y., Yoshioka, K., and van Eeten, M. Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai. *Network and Distributed System Security Symposium (NDSS)* (2019).

[27] Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., and Faruki, P. Network Intrusion Detection for IoT Security Based on Learning Techniques. *IEEE Communications Surveys and Tutorials* (2019).

[28] Chaman, A., Wang, J., Sun, J., Hassanieh, H., and Choudhury, R. R. Ghostbuster: Detecting the Presence of Hidden Eavesdroppers. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2018).

[29] Chen, W., Chen, L., Huang, Y., Zhang, X., Wang, L., Ruby, R., and Wu, K. Taprint: Secure Text Input for Commodity Smart Wristbands. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2019).

[30] Cheneau, T. Simplerpl. `https://github.com/tcheneau/simpleRPL`, 2013.

[31] Chiariotti, F., Pielli, C., Laurenti, N., and Zanella, A. A Game-Theoretic Analysis of Energy-Depleting Jamming Attacks with a Learning Counterstrategy. *ACM Transactions on Sensor Networks* (2019).

[32] Cimpanu, C. BrickerBot Author Retires Claiming to Have Bricked over 10 Million IoT Devices, 2017.

[33] Cimpanu, C. A decade of hacking. `https://www.zdnet.com/article/a-decade-of-hacking-the-most-notable-cyber-security-events-of-the-2010s/`, 2019.

[34] Cimpanu, C. A decade of malware. `https://www.zdnet.com/article/a-decade-of-malware-top-botnets-of-the-2010s/`, 2019.

[35] Collins, M. P., and Reiter, M. K. Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. *Symposium on Recent Advances in Intrusion Detection (RAID)* (2007).

[36] Copos, B., Levitt, K., Bishop, M., and Rowe, J. Is Anybody Home? Inferring Activity from Smart Home Network Traffic. *IEEE Symposium on Security and Privacy Workshops (SPW)* (2016).

[37] Costin, A., Zaddach, J., and Francillon, A. A Large Scale Analysis of the Security of Embedded Firmwares. *USENIX Security Symposium* (2014).

[38] Crandall, J., Su, Z., et al. On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic & Metamorphic Worm Exploits. *ACM Conference on Computer and Communications Security (CCS)* (2005).

[39] Cui, A., and Stolfo, S. J. A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide-Area Scan. *Annual Computer Security Applications Conference (ACSAC)* (2010).

[40] Czyz, J., Luckie, M., Allman, M., and Bailey, M. Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy. *Network and Distributed System Security Symposium (NDSS)* (2017).

[41] Dantas Silva, F. S., Silva, E., Neto, E. P., Lemos, M., Venancio Neto, A. J., and Esposito, F. A Taxonomy of DDoS Attack Mitigation Approaches Featured by SDN Technologies in IoT Scenarios. *Sensors* (2020).

[42] Denning, T., Kohno, T., and Levy, H. M. Computer security and the modern home. vol. 56, ACM, pp. 94–103.

[43] di Lallo, R., Griscioli, F., Lospoto, G., Mostafaei, H., Pizzonia, M., and Rimondini, M. Leveraging SDN to Monitor Critical Infrastructure Networks in a Smarter Way. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (2017).

[44] Dietzel, C., Wichtlhuber, M., Smaragdakis, G., and Feldmann, A. Stellar: Network attack mitigation using advanced blackholing. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (2018), CoNEXT.

[45] Ding, W., and Hu, H. On the Safety of IoT Device Physical Interaction Control. *ACM Conference on Computer and Communications Security (CCS)* (2018).

[46] Doshi, R., Apthorpe, N., and Feamster, N. Machine Learning DDoS Detection for Consumer IoT Devices. In *Security and Privacy Workshops* (2018).

[47] Farris, I., Taleb, T., Khettab, Y., and Song, J. A Survey on Emerging SDN and NFV Security Mechanisms for IoT Systems. *IEEE Communications Surveys & Tutorials* (2018).

[48] Fayaz, S. K., Tobioka, Y., Sekar, V., and Bailey, M. Bohatei: Flexible and elastic DDoS defense. In *USENIX Security Symposium* (2015).

[49] Feeney, L. M., and Nilsson, M. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)* (2001), vol. 3, IEEE, pp. 1548–1557.

[50] Feng, H., Fawaz, K., and Shin, K. G. Continuous Authentication for Voice Assistants. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2017).

[51] Feng, X., Li, Q., Wang, H., and Sun, L. Acquisitional Rule-based Engine for Discovering Internet-of-Things Devices. *USENIX Security Symposium* (2018).

[52] Fernandes, E., Jung, J., and Prakash, A. Security Analysis of Emerging Smart Home Applications. *IEEE Symposium on Security and Privacy (S&P)* (2016).

[53] Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., and Prakash, A. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. *USENIX Security Symposium* (2016).

[54] Fernandes, E., Rahmati, A., Jung, J., and Prakash, A. Decentralized Action Integrity for Trigger-Action IoT Platforms. *Network and Distributed System Security Symposium (NDSS)* (2018).

[55] Gendreau, A. A., and Moorman, M. Survey of Intrusion Detection Systems Towards an End to End Secure Internet of Things. *International Conference on Future Internet of Things and Cloud (FiCloud)* (2016).

[56] Granjal, J., Monteiro, E., and Sa Silva, J. Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *IEEE Communications Surveys and Tutorials* (2015).

[57] Gu, G., Perdisci, R., Zhang, J., et al. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX Security Symposium* (2008).

[58] Gu, G., Sharif, M., Qin, X., et al. Worm Detection, Early Warning and Response Based on Local Victim Information. *Annual Computer Security Applications Conference (ACSAC)* (2004).

[59] Hafeez, I., Ding, A. Y., Suomalainen, L., Kirichenko, A., and Tarkoma, S. Securebox: Toward Safer and Smarter IoT Network. *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking* (2016).

[60] He, W., Padhi, R., Ofek, J., Golla, M., Dürmuth, M., Fernandes, E., and Ur, B. Rethinking Access Control and Authentication for the Home Internet of Things (IoT). *USENIX Security Symposium* (2018).

[61] He, Y., Bian, J., Tong, X., Qian, Z., Zhu, W., Tian, X., and Wang, X. Canceling Inaudible Voice Commands Against Voice Control Systems. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2019).

[62] Herwig, S., Harvey, K., Hughey, G., Roberts, R., and Levin, D. Measurement and Analysis of Hajime, A Peer-to-Peer IoT Botnet. *Network and Distributed System Security Symposium (NDSS)* (2019).

[63] Hilton, S. Dyn analysis summary of friday october 21 attack. `https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack`, 2016.

[64] Ho, G., Leung, D., Mishra, P., Hosseini, A., Song, D., and Wagner, D. Smart Locks: Lessons for Securing Commodity Internet of Things Devices. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2016).

[65] Hong, J., Levy, A., Riliskis, L., and Levis, P. Don't Talk Unless I Say So! Securing the Internet of Things with Default-Off Networking. *ACM/IEEE International Conference on Internet of Things Design and Implementation (IoTDI)* (2018).

[66] Hummel, R., Hildebrand, C., Modi, H., et al. NETSCOUT Threat Intelligence Report: DDoS in a Time of Pandemic. https://www.netscout.com/threatreport/, 2021.

[67] Hutchins, B., Zhou, M., Reddy, A., Li, M., Jin, W., and Yang, L. Beat-PIN: A User Authentication Mechanism for Wearable Devices Through Secret Beats. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2018).

[68] Ikeda, S. IoT-Based DDoS Attacks Are Growing and Making Use of Common Vulnerabilities. https://www.cpomagazine.com/cyber-security/iot-based-ddos-attacks-are-growing-and-making-use-of-common-vulnerabilities/, 2020.

[69] Ilascu, I. IoT Botnets Responsible for More Powerful DDoS Attacks. https://www.bitdefender.com/box/blog/iot-news/iot-botnets-responsible-powerful-ddos-attacks/, 2018.

[70] Ilascu, I. New Silex Malware Trashes IoT Devices Using Default Passwords. https://www.bleepingcomputer.com/news/security/new-silex-malware-trashes-iot-devices-using-default-passwords/, 2019.

[71] Janita. DDoS Attack Halts Heating in Finland Amidst Winter, 2016.

[72] Javaid, U., Siang, A. K., Aman, M. N., and Sikdar, B. Mitigating loT Device based DDoS Attacks using Blockchain. *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems* (2018).

[73] Jung, J., Milito, R., and Paxson, V. On the Adaptive Real-Time Detection of Fast-Propagating Network Worms. *Conference on Detection of Intrusions & Malware & Vulnerability Assessment (DIMVA)* (2007).

[74] Kalofonos, D. N., and 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* (2007), IEEE, pp. 1–5.

[75] Kang, W. M., Moon, S. Y., and Park, J. H. An enhanced security framework for home appliances in smart home. Springer Berlin Heidelberg, pp. 1–6.

[76] Kent, S., and Atkinson, R. RFC 2401: Security Architecture for the Internet Protocol. https://www.rfc-editor.org/rfc/rfc2401.html, 2015.

[77] Kim, H.-A., and Karp, B. Autograph: Toward Automated, Distributed Worm Signature Detection. *USENIX Security Symposium* (2004).

[78] Kumar, D., Paccagnella, R., Murley, P., Hennenfent, E., Mason, J., Bates, A., and Bailey, M. Skill Squatting Attacks on Amazon Alexa. *USENIX Security Symposium* (2018).

[79] Kumar, P., Braeken, A., Gurtov, A., Iinatti, J., and Ha, P. Anonymous secure framework in connected smart home environments. IEEE, pp. 968–979.

[80] Li, C., Ji, X., Zhou, X., Zhang, J., Tian, J., Zhang, Y., and Xu, W. HlcAuth: Key-free and Secure Communications via Home-Limited Channel. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2018).

[81] Li, J., and Stafford, S. Detecting Smart, Self-Propagating Internet Worms. *IEEE Conference on Communications and Network Security (CNS)* (2014).

[82] Li, P., Salour, M., and Su, X. A Survey of Internet Worm Detection & Containment. In *IEEE Communications Surveys & Tutorials* (2008).

[83] Li, X., Yan, F., Zuo, F., Zeng, Q., and Luo, L. Touch Well Before Use: Intuitive and Secure Authentication for IoT Devices. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2019).

[84] Li, Z., Wang, L., Chen, Y., and Fu, Z. Network-based and Attack-resilient Length Signature Generation for Zero-day Polymorphic Worms. *International Conference on Network Protocols (ICNP)* (2007).

[85] Liang, Z., and Sekar, R. Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers. *ACM Conference on Computer and Communications Security (CCS)* (2005).

[86] Lin, F., Song, C., Zhuang, Y., Xu, W., Li, C., and Ren, K. Cardiac Scan: A Non-Contact and Continuous Heart-Based User Authentication System. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2017).

[87] Liu, Z., Jin, H., Hu, Y.-C., and Bailey, M. Middlepolice: Toward enforcing destination-defined policies in the middle of the Internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS.

[88] Mare, S., Girvin, L., Roesner, F., and Kohno, T. Consumer smart homes: Where we are and where we need to go. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications* (2019), ACM, pp. 117–122.

[89] Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M. H., Cunha, I., Guedes, D., and Meira, W. The Evolution of Bashlite and Mirai IoT Botnets. *IEEE Symposium on Computers and Communications (ISCC)* (2018).

[90] Mehdi, S. A., Khalid, J., and Khayam, S. A. Revisiting Traffic Anomaly Detection Using Software Defined Networking. *Symposium on Recent Advances in Intrusion Detection (RAID)* (2011).

[91] Melara, M. S., Liu, D. H., and Freedman, M. J. Pyronia: Redesigning Least Privilege and Isolation for the Age of IoT. *arXiv preprint arXiv:1903.01950* (2019).

[92] Minn, Y., Pa, P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. IoTPOT: Analysing the Rise of IoT Compromises. *USENIX Workshop on Offensive Technologies (WOOT)* (2015).

[93] Mirkovic, J., and Reiher, P. D-ward: A source-end defense against flooding denial-of-service attacks. vol. 2, IEEE, pp. 216–232.

[94] Mitev, R., Miettinen, M., and Sadeghi, A. R. Alexa Lied to Me: Skill-Based Man-in-the-Middle Attacks on Virtual Assistants. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2019).

[95] Morgner, P., Mattejat, S., and Benenson, Z. All Your Bulbs are Belong to Us: Investigating the Current State of Security in Connected Lighting Systems. *arXiv preprint arXiv:1608.03732* (2016).

[96] Mosenia, A., and Niraj, K. A Comprehensive Study of Internet of Things. *IEEE Transactions on Emerging Topics in Computing* (2017).

[97] Moskowitz, R., and Nikander, P. RFC 4423: Host Identity Protocol Architecture. `https://datatracker.ietf.org/doc/html/draft-ietf-hip-arch`, 2015.

[98] Neisse, R., Steri, G., and Baldini, G. Enforcement of security policy rules for the internet of things. In *IEEE 10th International Conference on Wireless and Mobile Computing* (2014), IEEE, pp. 165–172.

[99] Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., and Ghani, N. Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. *IEEE Communications Surveys and Tutorials* (2019).

[100] Newsome, J., and Song, D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. *Network and Distributed System Security Symposium (NDSS)* (2005).

[101] Nguyen, V., Ibrahim, M., Truong, H., Nguyen, P., Gruteser, M., Howard, R., and Vu, T. Body-Guided Communications: A Low-Power, Highly-Confined Primitive to Track and Secure Every Touch. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2018).

[102] Notra, S., Siddiqi, M., Gharakheili, H. H., Sivaraman, V., and Boreli, R. An experimental study of security and privacy risks with emerging household appliances. In *IEEE Conference on Communications and Network Security* (2014), IEEE, pp. 79–84.

[103] Paxson, V. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks 31*, 23-24 (1999), 2435–2463.

[104] Perkins, C., Belding-Royer, E., and Daas, S. RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing. *Internet Engineering Task Force* (2003).

[105] Polk, T., Souppaya, M., Haag, B., and Barker, W. C. Mitigating IoT-Based Distributed Denial of Service (DDoS). *US Department of Commerce, National Institute of Standards and Technology* (2017).

[106] Procopiou, A., Komninos, N., and Douligeris, C. ForChaos: Real Time Application DDoS Detection using Forecasting and Chaos Theory in Smart Home IoT Network. *Wireless Communications and Mobile Computing* (2019).

[107] Rahmati, A., Fernandes, E., Eykholt, K., and Prakash, A. Tyche: A Risk-Based Permission Model for Smart Homes. *IEEE Cybersecurity Development Conference (SecDev)* (2018).

[108] Ramanathan, S., Mirkovic, J., Yu, M., and Zhang, Y. SENSS against volumetric DDoS attacks. In *Proceedings of the 34th Annual Computer Security Applications Conference* (2018), ACSAC.

[109] Raza, S., Duquennoy, S., Chung, T., Yazar, D., Voigt, T., and Roedig, U. Securing Communication in 6LoWPAN with Compressed IPsec. *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)* (2011).

[110] Raza, S., Shafagh, H., Hewage, K., Hummen, R., and Voigt, T. Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal* (2013).

[111] Raza, S., Wallgren, L., and Voigt, T. SVELTE: Real-Time Intrusion Detection in The Internet of Things. *Ad Hoc networks* (2013).

[112] Roman, R., Zhou, J., and Lopez, J. Applying intrusion detection systems to wireless sensor networks. In *IEEE Consumer Communications & Networking Conference* (2006), IEEE, pp. 640–644.

[113] Ronen, E., and Shamir, A. Extended Functionality Attacks on IoT Devices: The Case of Smart Lights. *IEEE European Symposium on Security and Privacy (Euro S&P)* (2016).

[114] Ronen, E., Shamir, A., Weingarten, A. O., and Oflynn, C. IoT Goes Nuclear: Creating a Zigbee Chain Reaction. *IEEE Symposium on Security and Privacy (S&P)* (2018).

[115] Ryan, M. Bluetooth Smart: The Good, The Bad, The Ugly ... and The Fix. *Black Hat USA* (2013).

[116] Sahay, R., Blanc, G., Zhang, Z., and Debar, H. Towards Autonomic DDoS Mitigation using Software Defined Networking. *Workshop on Security of Emerging Networking Technologies* (2015).

[117] Schechter, S. E., Jung, J., and Berger, A. W. Fast Detection of Scanning Worm Infections. *Symposium on Recent Advances in Intrusion Detection (RAID)* (2004).

[118] Sehgal, A., Perelman, V., Kuryla, S., and Schonwalder, J. Management of resource constrained devices in the internet of things. *IEEE Communications Magazine 50*, 12 (2012).

[119] Sekar, V., Xie, Y., Reiter, M. K., et al. A Multi-Resolution Approach for Worm Detection and Containment. *International Conference on Dependable Systems and Networks (DSN)* (2006).

[120] Shafagh, H., Hithnawi, A., Burkhalter, L., Fischli, P., Shafagh, H., Hithnawi, A., Burkhalter, L., Fischli, P., Shar, S. D. S., Shafagh, H., Burkhalter, L., Fischli, P., and Duquennoy, S. Secure Sharing of Partially Homomorphic Encrypted IoT Data. *ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2017).

[121] Shafagh, H., Hithnawi, A., Dröscher, A., Duquennoy, S., and Hu, W. Talos: Encrypted Query Processing for the Internet of Things. *ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2015).

[122] Shelby, Z., Hartke, K., Bormann, C., and Frank, B. RFC 7252: The constrained application protocol (CoAP). *Internet Engineering Task Force* (2014).

[123] Shin, S., and Gu, G. CloudWatcher: Network Security Monitoring using OpenFlow in Dynamic Cloud Networks (or: How to provide security monitoring as a service in clouds? *International Conference on Network Protocols (ICNP)* (2012).

[124] Shin, S., Wang, H., and Gu, G. A First Step Toward Network Security Virtualization: From Concept To Prototype. *IEEE Transactions on Information Forensics and Security* (2015).

[125] Simpson, A. K., Roesner, F., and Kohno, T. Securing Vulnerable Home IoT Devices with an In-Hub Security Manager. *International Workshop on Pervasive Smart Living Spaces (PerLS)* (2017).

[126] Simpson, A. K., Roesner, F., and Kohno, T. Securing vulnerable home iot devices with an in-hub security manager. In *IEEE International Conference on Pervasive Computing and Communications Workshops* (2017), IEEE, pp. 551–556.

[127] Singh, A., Chawla, N., Ko, J. H., Kar, M., and Mukhopadhyay, S. Energy Efficient and Side-Channel Secure Cryptographic Hardware for IoT-Edge Nodes. *IEEE Internet of Things Journal* (2019).

[128] Singh, S., Estan, C., et al. Automated Worm Fingerprinting. *Symposium on Operating System Design and Implementation (OSDI)* (2004).

[129] Sinha, S. State of IoT 2021. `https://iot-analytics.com/number-connected-iot-devices/`, 2021.

[130] Sinha, S. State of IoT 2021. `https://iot-analytics.com/number-connected-iot-devices/`, 2021.

[131] Sisodia, D. On the State of Internet of Things Security: Vulnerabilities, Attacks, and Recent Countermeasures. University of Oregon. Tech. rep., AREA-202001-Sisodia, 2020.

[132] Sisodia, D., Mergendahl, S., Li, J., and Cam, H. Securing the Smart Home via a Two-Mode Security Framework. *International Conference on Security and Privacy in Communication Systems (SecureComm)* (2018).

[133] Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. Classifying IoT Devices in Smart Environments using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* (2018).

[134] Sivaraman, V., Gharakheili, H. H., Fernandes, C., Clark, N., and Karliychuk, T. Smart iot devices in the home: Security and privacy implications. vol. 37, IEEE, pp. 71–79.

[135] Sohi, S. M., Seifert, J.-P., and Ganji, F. RNNIDS: Enhancing Network Intrusion Detection Systems through Deep Learning. *Computers & Security* (2021).

[136] Soltan, S., Mittal, P., Vincent, H., and Poor, H. V. BlackIoT: IoT Botnet of High Wattage Devices Can Disrupt the Power Grid. *USENIX Security Symposium* (2018).

[137] Song, J. The realities of smart city development. `https://www.forbes.com/sites/forbestechcouncil/2019/05/14/the-realities-of-smart-city-development`, 2019.

[138] Stafford, S., and Li, J. Behavior-based Worm Detectors Compared. In *Symposium on Recent Advances in Intrusion Detection* (2010).

[139] Staniford, S., Paxson, V., and Weaver, N. How to 0wn the Internet in Your Spare Time. In *USENIX Security Symposium* (2002).

[140] Team, O. P. Ossec: Open source hids security. `https://ossec.github.io/index.html`, 2019.

[141] Vasserman, E. Y., and Hopper, N. Vampire Attacks: Draining Life from Wireless Ad Hoc Sensor Networks. *IEEE Transactions on Mobile Computing* (2013).

[142] Vicente, M., Galera, B., and Remillano, A. Bashlite IoT Malware Updated with Mining and Backdoor Commands, Targets WeMo Devices. `https://blog.trendmicro.com/trendlabs-security-intelligence/bashlite-iot-malware-updated-with-mining-and-backdoor-commands-targets-wemo-devices/`, 2019.

[143] Vidgren, N., Haataja, K., Patiño-Andres, J. L., Ramírez-Sanchis, J. J., and Toivanen, P. Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned. *Hawaii International Conference on System Sciences (HICSS)* (2013).

[144] Wallgren, L., Raza, S., and Voigt, T. Routing Attacks and Countermeasures in the RPL-Based Internet of Things. *International Journal of Distributed Sensor Networks* (2013).

[145] Wang, B., Zheng, Y., Lou, W., and Hou, Y. T. DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking. *Computer Networks* (2015).

[146] Wang, C., Guo, X., Wang, Y., Chen, Y., and Liu, B. Friend or Foe? Your Wearable Devices Reveal your Personal PIN. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (2016).

[147] Wang, H., Lai, T. T. T., and Choudhury, R. R. MoLe: Motion Leaks Through Smartwatch Sensors. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2015).

[148] Wang, Q., Datta, P., Yang, W., Liu, S., Bates, A., and Gunter, C. A. Charting the Attack Surface of Trigger-Action IoT Platforms. *ACM Conference on Computer and Communications Security (CCS)* (2019).

[149] Wang, Q., Hassan, W. U., Bates, A., and Gunter, C. Fear and Logging in the Internet of Things. *Network and Distributed System Security Symposium (NDSS)* (2018).

[150] Ward, M. Why some computer viruses refuse to die. `https://www.bbc.com/news/technology-44564709`, 2018.

[151] Winter, T., Thubert, P., Brandt, A., Hui, J., and Kelsey, R. RFC 6550: RPL: IPv6 Routing Protocol for Low-power and Lossy Networks. `https://tools.ietf.org/html/rfc6550`, 2012.

[152] Winter, T., Thubert, P., Brandt, A., Hui, J., and Kelsey, R. RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. *Internet Engineering Task Force* (2014).

[153] Wurm, J., Hoang, K., Arias, O., Sadeghi, A. R., and Jin, Y. Security Analysis on Consumer and Industrial IoT Devices. *Asia and South Pacific Design Automation Conference (ASP-DAC)* (2016).

[154] Xu, M., Huber, M., Sun, Z., England, P., Peinado, M., Lee, S., Marochko, A., Mattoon, D., Spiger, R., and Thom, S. Dominance as a New Trusted Computing Primitive for the Internet of Things. *IEEE Symposium on Security and Privacy (S&P)* (2019).

[155] Yan, Q., Yu, F. R., Gong, Q., and Li, J. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges. *IEEE Communications Surveys & Tutorials* (2015).

[156] Yan, Z., Song, Q., Tan, R., Li, Y., and Kong, A. W. K. Towards Touch-to-Access Device Authentication Using Induced Body Electric Potentials. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2019).

[157] Yang, Y., Wang, J., Zhai, B., and Liu, J. IoT-Based DDoS Attack Detection and Mitigation Using the Edge of SDN. *International Symposium on Cyberspace Safety and Security (CSS)* (2019).

[158] Yang, Z., Huang, Q., and Zhang, Q. NICScater: Backscater as a Covert Channel in Mobile Devices. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2017).

[159] Yiakoumis, Y., Yap, K.-K., Katti, S., Parulkar, G., and McKeown, N. Slicing Home Networks. *Proceedings of the 2nd ACM SIGCOMM Workshop on Home Networks* (2011).

[160] Yoon, C., Park, T., Lee, S., Kang, H., Shin, S., and Zhang, Z. Enabling Security Functions with SDN: A Feasibility Study. *Computer Networks* (2015).

[161] Yu, T., Sekar, V., Seshan, S., Agarwal, Y., and Xu, C. Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (2015).

[162] Zarpelao, B. B., Miani, R. S., Kawakani, C. T., and de Alvarenga, S. C. A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications* (2017).

[163] Zeng, E., and Roesner, F. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study. *USENIX Security Symposium* (2019).

[164] Zhang, G., Yan, C., Ji, X., Zhang, T., Zhang, T., and Xu, W. DolphinAttack: Inaudible voice commands. *ACM Conference on Computer and Communications Security (CCS)* (2017).

[165] Zhang, N., Mi, X., Feng, X., Wang, X., Tian, Y., and Qian, F. Dangerous Skills: Understanding and Mitigating Security Risks of Voice-Controlled Third-Party Functions on Virtual Personal Assistant Systems. *IEEE Symposium on Security and Privacy (S&P)* (2019).

[166] Zhang, W., Meng, Y., Liu, Y., Zhang, X., Zhang, Y., and Zhu, H. Homonit: Monitoring Smart Home Apps from Encrypted Traffic. *ACM Conference on Computer and Communications Security (CCS)* (2018).

[167] Zheng, J., Li, Q., Gu, G., Cao, J., Yau, D. K., and Wu, J. Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis. *IEEE Transactions on Information Forensics and Security* (2018).

[168] Zhou, H., Hu, Y., Yang, X., Pan, H., Guo, W., and Zou, C. C. A Worm Detection System Based on Deep Learning. *IEEE Access* (2020).

[169] Zillner, T., and Strobl, S. ZigBee Exploited: The Good, The Bad and The Ugly. *Black Hat USA* (2015).