ABSTRACT

OCONNOR, TERRENCE JOHN. Network Access Control through Host and Application Analysis. (Under the direction of William Enck).

Despite increasing complexity to networks and applications, the state of practice in network access control has remained largely static for decades. Existing network security solutions provide limited protection against modern and elaborate network compromises where attackers move laterally through the network, because they lack intra-network mediation and distributed information flow control. The omnipresence of Internet-of-Things (IoT) devices with encrypted and propriety protocols obscures network transparency, making it difficult to identify and classify application behaviors. The always-on nature of IoT with perpetual cloud connections complicates policy enforcement because of the heterogeneity of security and privacy designs for offline content caching and device state management. Leveraging host and application behavior analysis provides sufficient context for network access control to address the increasing complexity of networks, opaqueness in applications behaviors, and perpetual cloud connections.

In this dissertation, we demonstrate how specialized and sophisticated network access control schemes address the aforementioned challenges by considering host and application context. We implement two security orchestrator application frameworks to address complex attacks, and provide behavior transparency for smart-home IoT devices. First, we present PivotWall which uses a novel combination of information-flow tracking and Software Defined Networking (SDN) to detect a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. It further enables a variety of attack responses including traffic steering, as well as advanced mechanisms for forensic analysis. Through PivotWall, we extend information flow tracking out from a host and through the entire network. Second, we introduce HomeSnitch, a building block for enhancing smart home transparency and control by classifying IoT device communication by behavior. HomeSnitch uses supervised learning to classify features from connection-oriented application data unit exchanges that represent application-layer dialog between clients and servers. This overcomes the limitations of increasingly encrypted payloads in smart home devices. Having laid the groundwork for access control for smart-home devices with our behavior classification, we then answer critical questions that govern access control policy and mediation for smart-home devices. We present a sufficiently broad study of the application behaviors of smart-home actuator and sensor-based IoT devices and analyze their responses to a loss of the communications channel. We investigate weaknesses in the current heterogeneity of smart-home device security designs and describe implications for offline state and cache decisions. In doing so, we provide the context understanding for effective policy enforcement.

Network Access Control through Host and Application Analysis

by
Terrence John OConnor

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2019

APPROVED BY:

Bradley Reaves	Alexandros Kapravelos
Aydin Aysu	William Enck Chair of Advisory Committee

DEDICATION

For my Ninja Princess and Monkey.

BIOGRAPHY

TJ OConnor received his Bachelor's Degree in Computer Science from the US Military Academy at West Point in 1999. Following graduation, TJ served 20 years in the US Army, retiring at the rank of Lieutenant Colonel. His service included assignments with the 2nd Infantry Division, the 7th Special Forces Group (Airborne), the 10th Special Forces Group (Airborne), the Office of Special Warfare (Airborne), and the 1st Special Forces Command (Airborne). In 2008, TJ earned a Master's Degree in Computer Science from North Carolina State University. Subsequently, he served as an assistant professor in the Electrical Engineering and Computer Science department at the US Military Academy and as the Professor of Military Science for the Florida Institute of Technology. He is the first (but hopefully not last) member of his immediate family to earn a doctorate.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. William Enck, for his unparalleled support over the course of my time in graduate school. I cannot express the gratitude, respect, and debt I owe you. I can only promise to do my best to emulate the positive traits you have taught me. I would also like to thank the members of my doctoral committee including Dr. Bradley Reaves, Dr. Alexandros Kapravelos, and Dr. Aydin Aysu for their valuable insight. I owe a debt of gratitude my teammates on the Wolfpack Security and Privacy Research (WSPR) that have provided me the candid feedback, expertise, and emotional support necessary for this endeavor. Specifically, I'd like to thank Akash Verma, Sanket Gooutam, Micah Bushouse, Benjamin Andow, Luke Deshotels, and Isaac Polinsky. I'd also like to thank Michael Petullo and Sam Huddleston for being my sounding boards on so many ideas. I would finally like to thank my family for their unconditional love, support and belief in me. We did this.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABAC	attribute-based-access-control
ACL	access control list
ADU	application data unit
API	application programming interface
CIA	Central Intelligence Agency
CIDR	classless inter-domain routing
DBSCAN	density-based spatial clustering of applications with noise
DET	data ex-filtration toolkit
DLP	data-loss-prevention
DNS	domain name system
DPI	deep packet inspection
HTTP	hypertext transfer protocol
ICMP	internet control message protocol
IDS	intrusion detection system
IFC	information-flow control
IP	internet protocol
IoT	internet of things
KNN	k-nearest neighbors
LPM	Linux provenance modules
MAC	media access control
ML	machine learning
MLS	multi-level security
NAT	network address translation
NFC	near-field communication
NFV	network functions virtualization
NIFC	network information-flow control

OUI	organizationally unique identifier
OVS	Open vSwitch
PACS	picture archiving and communication system
RAM	random-access memory
RBAC	role-based-access-control
RF	radio frequency
RFC	random forest classifier
RFC	request for comments
SDN	software defined networking
SQL	structured query language
SSL	secure sockets layer
SSP	security service provider
TCB	trusted computing base
TCP	transmission control protocol
TLS	transport layer security
TPR	true positive rate
ToS	type of service
UBMR	unknown behavior miss rate
UUID	universally unique identifier
VLAN	virtual local area network
VPN	virtual private networks
WSN	wireless sensor networks

CHAPTER

1 INTRODUCTION

The state of practice of network access control has not evolved, remaining static with tedious policies to configure and prone to errors [Che18a; Vor17; EG17; Woo04]. While the evolution of network access control is stagnant, complex and elaborate attacks have emerged alongside the growth of a heterogeneity of *always-on*, *always-connected* Internet of Things (IoT) devices. Elaborate attacks commonly move laterally and undetected within an enterprise network towards a target. Existing network security techniques only provide limited protection against such attacks because they lack intra-network mediation and the context of information semantics. Previous approaches to address these sophisticated attacks have relied on passive observation or active watermarking to correlate network flows [HB11; Hou09; Wan05b; Wan05a; WR03; Wan02; ZP00]. However, a myriad of techniques (e.g., timing perturbations, chaff packets, flow splitting or repacketization) can corrupt embedded watermarks and flow correlation.

The widespread adoption and integration of IoT devices is a critical risk to network security. The rapid emergence of IoT has led to a broad and heterogeneous market with flawed security designs [New18; FB17; Fer16; Nay16]. Stark differences exist in vendor provided solutions for access control of these always-on and always-connected devices. Effective access control and policy enforcement for IoT relies on fine-grained analysis of device behaviors. However, the increasing use of encryption to secure network flows complicates classifying the broad array of devices and semantic behaviors. Further, the perpetually-connected nature of IoT presents challenges to access control and policy enforcement. Immature telemetry and caching implementations introduce significant vulnerabilities by *blinding* sensor devices or *confusing* the state of actuator devices. This immaturity and lack of a de-facto approach further complicate access control and policy enforcement.

Ultimately, network-based security systems have not evolved to address modern challenges. Specifically, systems lack the context of information semantics to effectively enforce security policies. Traditional network-based access control can only effectively enforce static rules between individual hosts. It lacks the dynamic approach and context to enforce rules for elaborate attacks spanning across multiple hosts, services, and network flows. Such elaborate attacks operate in network blind spots, hiding from otherwise unaware network security systems. At the network-layer, security systems lack the host-based information context including the transactions and transformations of tainted or classified data or services. Further, network security solutions cannot effectively perform deep packet inspection or protocol analysis against the growing emergence of encrypted and proprietary protocols of smart home IoT devices. Fine-grained control is not achievable without an understanding of application behavior semantics. This is further complicated by the heterogeneity of buffering and state management schemes for IoT devices. To this end, network security solutions are ineffective to modern threats that operate in network blind spots and over encrypted protocols of proprietary applications. Network access control solutions requires additional context to evolve to the modern threat.

Context can offer meaning to otherwise innocuous, benign, or obscured network events. Consider the case of a stepping stone attack [ZP00], where individual flows are permitted but an attacker exploits the aggregate of multiple flows to ex-filtrate sensitive data. Additional host-based context can reduce the semantic gap between network and host controls, enabling transparency for a stealthy attack methodology. Next, examine the challenge of the encrypted network traffic of IoT devices. A stand-alone analysis of encrypted packet payloads offers little insight or intelligence to the behavior of an application or service. However, a network security solution can leverage a machine learning classifier to extract meaning out of the payload size and inter-arrival times to infer application behaviors [LL06; AZH09; AZH11; Zha18]. To this end, context transforms an unintelligent and stagnant network access control system to an empowered solution capable of enforcing policy to mitigate dynamic attack methodologies.

We focus our investigation on adding context to empower fine-grained network access control for elaborate attacks. To this end, we examine the introduction of software designed networking (SDN) and the new opportunities it offers for network defense systems [Fea14; KF13]. SDN offers flexibility, transparency, and distributed control to enable novel security mechanisms not feasible in traditional networks. The logically central location of SDN can be used to both investigate and mediate risks [Bat14]. Centralizing decision making and policy enforcement at the SDN offers a significant evolution of the traditional concept of a reference monitor. Our work examines if network-based access control can evolve to prevent elaborate attacks and application behaviors under varying context (e.g., during specific hours of the day, the presence of encryption, or other the actions of other network devices). To this end, we examine the various design and telemetry decisions for IoT devices as they offer an emerging and complex threat [Gre17b; Gre17a; Dio18].

1.1 Thesis Statement

In this dissertation, we investigate solutions to three weaknesses associated with modern network access control: the lack of practical information flow tracking in a distributed environment, the in-ability to classify semantic application behaviors from IoT encrypted network flows, and the complications of policy enforcement in a heterogeneity of IoT security and privacy designs. We observe that adding additional context to network defenses systems can increase the effectiveness of access control, policy enforcement, and mediation. Therefore, the thesis statement of this work is the following:

Leveraging host and application behavior analysis provides sufficient context for network access control to address the increasing complexity of networks, opaqueness in applications behaviors, and perpetual cloud connection.

The scope of this dissertation focuses on increasing the effectiveness of network access control by leveraging host and application analysis. Within this scope, we provide mediation for complex attacks, offer classification algorithms for proprietary and encrypted flows from IoT devices, and provide effective policy enforcement in the presence of a heterogeneity of IoT cache and state management strategies. We do not consider attestation of our host agents or the integrity of networking devices and consider such attacks outside the scope of this thesis. Further, we only consider IP based protocols and no do consider devices that communicate via other protocols (ZigBee, Bluetooth, NFC.)

This dissertation explores three research questions (RQ) regarding host and application behavior analysis to enhance network access control.

- *RQ1: What attacks can be detected by extending information flow tracking on each host into network layer defenses?* PivotWall demonstrates that moving the information-flow reference monitor to the SDN/NFV increases the context available to defenders during stepping-stone and other elaborate attacks that bypass traditional enterprise defenses.
- *RQ2: What algorithmic approaches can be leveraged to classify encrypted and proprietary application layer traffic of smart-home devices?* HomeSnitch implements a supervised learning technique to classify semantic device behaviors using network traffic, creating a block for transparency and control of smart home IoT device network communication.
- *RQ3: What impact do smart-home device content cache and state management strategies have on effective access control?* Our final work explores the design decisions in smart home IoT devices

that govern cache and state management. We analyze the telemetry and messaging protocols and discuss the impacts of telemetry isolation and buffering on effective access control.

1.2 Thesis Contributions

PivotWall (SoSR 2018): The PivotWall work provides three contributions. First, we extend information-flow control beyond the boundary of the host into the network using software defined networking (SDN). Second, we provide the semantics for a policy language for practically specifying information flow control within an enterprise network. Third, we perform an empirical evaluation, which demonstrates moving the information-flow reference monitor to the SDN/NFV increases the context available to defenders during stepping-stone and other elaborate attacks.

HomeSnitch (WiSec 2019): The HomeSnitch work also makes three contributions. First, we present the HomeSnitch framework which uses machine learning-based classification models for identification of semantic device behaviors. Second, we demonstrate HomeSnitch's utility to classify both encrypted traffic and proprietary application-layer protocols through an evaluation of an independent dataset. Finally, we perform an empirical evaluation in a real home environment that demonstrates how HomeSnitch can help experts identify previously unseen devices and behaviors.

Blinded and Confused (WiSec 2019): This work makes three contributions. First, we performed a sufficiently broad study of how smart home device content implement offline caching and state recovery design decisions. Second, we analyzed the security and privacy implications for a connectivity loss at the semantic behavior layer. We investigate how vendors implement different approaches for these critical security and privacy decisions, potentially creating *blind* spots for sensor devices and state *confusion* for actuator devices. Finally, we demonstrate the impact these decisions have on mediation and policy enforcement mechanisms for effective network access control for smart home devices.

1.3 Thesis Organization

This dissertation aims to address the challenges of practical network access control and policy enforcement in the presence of increasingly complex networks, the omnipresence of IoT devices, and perpetual cloud connections. First, access control in a distributed environment requires effective information flow tracking. However, information flow tracking in a distributed environment presents challenges with precision, practical policy enforcement, label integrity, and effective attribution. Second, IoT devices obscure network transparency, complicating effective control of the network communication. Semantic behaviors of IoT devices present a classification challenge due to encryption and proprietary protocols. Finally, effective mediation and policy enforcement require a broad understanding of how devices implement content caching and state management strategies. However, vendors implements a heterogeneity of these essential design decisions, complicating practical policy enforcement.

Chapter 2: In Chapter 2, we provide an introduction to this thesis's technical areas and related work. First, we present an overview of the capabilities of software defined networking (SDN) with an emphasis on the fine-grained control and flexibility offered in SDN that is not achievable in traditional network access control. Next, we explore the challenges introduced by the growth of the internet of things (IoT), including 1) the threats that accompany off-loading processing and storage to the cloud, 2) the difficulty of enforcing access control, and 3) the lack of transparency in the IoT. Finally, we discuss the trade-offs associated with various supervised and unsupervised machine learning algorithms and explore a framework for evaluating machine learning.

Chapter 3: In Chapter 3, we propose PivotWall, a network security defense that extends information flow tracking on each host into network-level defenses. PivotWall uses a novel combination of information-flow tracking and Software Defined Networking (SDN) to detect a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. It further enables a variety of attack responses including traffic steering, as well as advanced mechanisms for forensic analysis. We show that PivotWall incurs minimal impact on network throughput and latency for untainted traffic with minimal overhead. As such, we demonstrate the utility of information flow tracking as a defense against advanced network-level attacks.

Chapter 4: In Chapter 4, we present HomeSnitch, a building block for enhancing smart home transparency and control by classifying IoT device communication by *behavior*. To identify behaviors, HomeSnitch uses supervised learning to classify features from connection-oriented application data unit exchanges that represent application-layer dialog between clients and servers. We implement HomeSnitch on a commodity wireless router and build upon Software Defined Networking (SDN) primitives to enforce network access controls. We evaluate HomeSnitch against an independent labeled corpus of IoT device network flows. We deploy HomeSnitch in a home environment and empirically evaluate its ability to correctly classify known behaviors as well as discover new behaviors. Through these efforts, we demonstrate the utility of network-level services to classify behaviors of and enforce control on smart home devices.

Chapter 5: In Chapter 5, we present our study of the IoT device design flaws. The always-on, always-connected nature of smart home devices complicates Internet-of-Things (IoT) security and privacy. Unlike traditional hosts, IoT devices constantly send sensor, state, and heartbeat data to cloud-based servers. These data channels require reliable, routine communication, which is often at odds with an IoT device's storage and power constraints. Although recent efforts such as pervasive encryption have addressed protecting data in-transit, there remains little insight into designing mechanisms for protecting integrity and availability for always-connected devices. This chapter seeks to better understand smart home device security by studying the vendor design decisions surrounding IoT telemetry messaging protocols, specifically, the behaviors taken when an IoT device loses connectivity. To understand this, we hypothesize and evaluate sensor blinding and state confusion attacks, measuring their effectiveness against an array of smart home IoT device types. Our analysis uncovers pervasive failure in designing telemetry that reports data to the cloud, and buffering that fails to properly cache undelivered data. We uncover that 22 of 24 studied devices suffer from critical design flaws that (1) enable attacks to transparently disrupt the reporting of device status alerts or (2) prevent the uploading of content integral to the device's core functionality.

Chapter 6: In Chapter 6, we first explore directions for future research. Specifically, we explore the feasibility of probabilistic tainting for resource-constrained devices, extending our behavioral classification model to cyber physical systems, and constructing an unsupervised learning model for on-demand activities. Finally, we present our conclusion by summarizing our contributions to network access control and policy enforcement by leveraging the context from host and application analysis.

CHAPTER

2 BACKGROUND AND RELATED WORK

This chapter introduces this thesis's technical areas and discusses related work. First, we present an overview of the capabilities of software defined networking (SDN). Specifically, we highlight the fine-grained control and flexibility offered in SDN that is not achievable in traditional network access control. Next, we discuss the challenges introduced by the growth of the internet of things (IoT). We explore the threats that accompany off-loading processing and storage to the cloud, the difficulty of enforcing access control, and the lack of transparency in the IoT. Further, we provide examples that explain the motivation for our work. Next, we discuss the ability for machine learning to address problems where static rules and logic cannot solve certain problems. To this end, we present the trade-offs associated with various supervised and unsupervised machine learning algorithms and discuss a framework for evaluation. Finally, we conclude with an overview of the threats examined in Chapters 3- 5.

2.1 Software Defined Networking

The advent of software defined networking (SDN) has enabled dynamic defenses capable of overcoming the limitations of statically-defined traditional networks. The works discussed in this thesis benefit from the paradigm shift of SDN. Specifically, SDN offers fine-grained control of network management by centralizing the decision-making capability at the logically central network controller. By decoupling the data plane from the control plane, SDN offers a level of abstraction that enables flexible and context-aware network access control. To better understand the opportunity SDN affords, we present an overview of

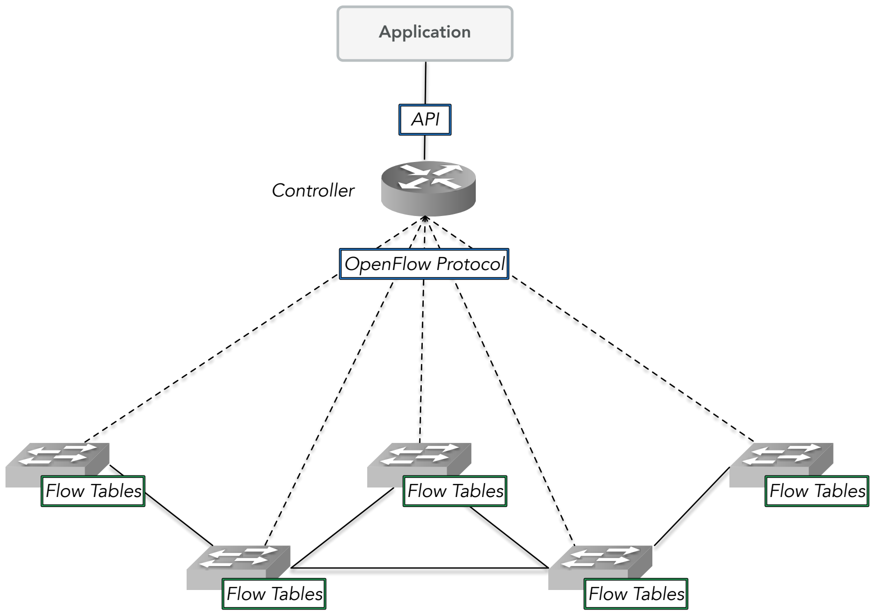


Figure 2.1 Overview of software defined networking

the data, control, and application planes, highlight key works that leverage SDN, and discuss emulation environments for evaluation.

2.1.1 The Data Plane

The versatility of SDN-based defenses derives from decoupling the data plane and the control plane of traditional networks. In an SDN environment, the *data plane* consists of networking devices (e.g., physical or virtual switches) that process and action packets against a set of rules. These rules consist of matching criteria (e.g., packet header fields) with specific instructions (e.g., forwarding a packet to a specific port, rewriting the particular header fields of a packet, dropping the packet, or delivering the packet to the controller for further action). These rules are stored locally in *flow modification* tables on the networking devices. These tables empower switches with the logic of the control plane without the overhead of forwarding the packet for a decision.

2.1.2 The Control Plane

In an SDN environment the control plane moves from the firmware of routers and switches to a software-defined controller. The *control plane* consists of a single controller (or series of controllers) that serve as the central decision making capability in an SDN environment. Communication protocols, including OpenFlow [Ope15] or Netconf [RE11] facilitate communication between the control and data planes. Through this communication, the controller gains a transparent view of the entire network from a logically central location. This central vantage point allows the controller to implement the logic of the application plane. Figure 2.1 illustrates the relationship between the application, control and data planes. The controller continually updates the data plane with flow modifications, enabling dynamic and reactive defense strategies. The control plane is governed by the application plane, which consists of one or a set of applications.

2.1.3 The Application Plane

The SDN *application plane* provides a directly programmable interface to implement agile defenses, such as the security orchestrators we have developed (discussed in Chapters 3 and 4). SDN frameworks bundle controller implementations with well-defined *application programming interfaces* (APIs) that facilitate network management and control. NOX [Gud08], POX [ONL15], Beacon [Eri13], OpenDayLight [Med14], Onos [Ber14a] and RYU [YT19] are examples of component-based SDN frameworks with well-defined APIs. They offer interfaces in C, Java, JavaScript, Python, REST, or Ruby to expose the programmable elements of the controller. By expressing logic as an application instead of a static set of rules, the application plane can dynamically alter the strategy of network defense as it becomes aware of new context. As illustrated in Chapters 3 and 4, SDN applications can enforce customized and dynamic policies that consider context and enforce fine-grained policies at a per-flow, per-device or even per-application behavior layer. To achieve this level of control, the application plane consists of components that process data, extract measurements, and conduct analysis to construct policies. In the following section, we highlight key works that demonstrate the reactive, agile, and context-aware capabilities of network security applications.

2.1.4 Extending SDN to Security

SDN offers a dynamic defense capabilities not achievable in traditional networks. One aspect of this dynamic defense is unique inline traffic rewriting. At the network layer, several works have extended SDN to implement moving target defenses capable of transparently mutating IP addresses that mask the location of hosts [Jaf12; MS15; Kam14]. At the application layer, PSI demonstrated rewriting credentials inline for hosts with hard-coded credentials [Yu17]. Fresco [Shi13a] offered an OpenFlow security application development framework that enabled the ability to rapidly designing modules for detection and mitigation of threats. SNIPs [Heo14] leveraged SDN to offload network intrusion prevention systems (NIPS), which addresses the challenges such as scalability, network load, and latency in traditional load balancing systems. Bates et al. [Bat14] demonstrated the forensic capability of SDN to investigate previously un-observable attacks such as data ex-filtration and collusion between compromised nodes. Next, we discuss emulation in several of these works, along with how our thesis benefit from evaluation in emulated environments.

2.1.5 Emulation Environments

Emulation frameworks enables rapid prototyping and evaluation of SDN applications in virtual environments. Through virtualization, emulation-based systems can replicate large network test-beds as well as simulate network loads. Network simulation and emulation tools include PlanetLab [Pet03], Geni [Ber14b], and Estinet [Wan13]. In the research domain, Mininet [de 14] has emerged as a popular emulation environment. Mininet leverages lightweight virtualization, for processes and network namespaces, to create a realistic virtual network within a single virtual machine (VM). Mininet offers flexible instantiation of different network topologies in software. Mininet provides scalability with the ability to construct an environment with hundreds of switches running inside a single VM. To support reproducibility of experiments, Mininet offers an environment that facilitates ease of sharing between research collaborators. Mininet environments can quickly be distributed among collaborators and replicated. Despite the benefits of Mininet, limitations exist. Mininet does not support WiFi topology and is limited to the resources of a single virtual machine, preventing massive scalability. Adaptations of Mininet overcome these limitations. Mininet-WiFi [FR18] extends Mininet to virtualize wireless networking environments. Mininet-WiFi forks the source of the original Mininet project, extending the source to include WiFi stations and access points. Mininet-HiFi [Han12] extends the original source of Mininet to support the use of Linux containers, achieving host isolation not realizable with standard host instantiation. Mininet-CE [AS13] extends Mininet to emulate a massive network by distributing Mininet nodes and links over a cluster of servers. By extending Mininet to a cluster, it overcomes the resource limitations (e.g., link bandwidth and CPU processing) of a single VM.

2.2 The Internet of Things

Chapters 4 and 5 present solutions to security and privacy challenges introduced by IoT. This section provides an overview of IoT and discusses a subset of these challenges. While the advent of IoT introduces many unique vectors, we focus on the challenges of offloading processing and storage to the cloud, enforcing access control, and the lack of transparency in IoT. We conclude with examples that motivated our work.

2.2.1 Overview of IoT

IoT is the coupling of networked devices, including actuators and sensors, to enable interaction, the exchange of data, and control. IoT offers an always-responsive environment to enable perpetual connectivity to *on-demand services*. With the advent of the smart-home, consumer IoT devices are becoming increasingly common. Examples include digital thermostats, security cameras, connected lighting, digital voice assistants, connected power outlets, and environmental sensors. These devices offer the convenience of monitoring and controlling all aspects of our homes. However, this convenience delivers new risks. Running on resource-constrained embedded single-board computers, these IoT devices process privacy-sensitive data generated in our homes. Several recent high profile compromises have demonstrated the privacy risks of IoT compromises [Gre17b; FB17; Dha15; Har15; MIT17]. Many IoT devices have poor authentication mechanisms, un-patchable software vulnerabilities, and limited access control configurability. Next, we discuss the how the perpetually connected cloud compounds these challenges.

2.2.2 Cloud Services

Cloud services introduce unique threats to the security and privacy of IoT. The cloud offers a new service paradigm that provisions computing-as-a-service [Goo18]. Cloud service enable always-responsive, perpetual connections to IoT on-demand services. This offers globally accessibility, enabling users to connect through the cloud to their IoT devices from anywhere. Further, the cloud offers resources not available on resource constrained IoT devices. Offloading IoT services to the cloud presents an attractive option as it overcomes the processing and storage limitations of IoT devices. However, these same benefits introduce new security challenges. With unlimited and unregulated access to IoT devices, the cloud complicates access control and policy enforcement. Privacy-enhancing encryption is used to secure the connection between IoT devices and the cloud. However, the use of encryption obscures transparency and network-based access control for IoT devices. This issue is complicated by the fact that most IoT devices cannot participate in any host-based access control to govern device access. Ultimately, the cloud removes host-level access control and network-layer intrusion detection and replaces it with blind trust in an off-site platform. Our work in Chapter 4 attempts to overcome this challenge by offering access control and transparency, while Chapter 5 analyzes the difficulties of enforcing access control at the network layer of IoT devices.

2.2.3 Access Control

Modern access control approaches can be divided into either role-based-access-control (RBAC) or attribute-based-access-control (ABAC). RBAC [Fer95] consists of a set of users and roles. Each role is authorized with specific privileges. In contrast, ABAC [YT05] consists of subjects and objects that identify through attributes. The ABAC model offers increased flexibility to enforce fine-grained control for IoT. However, both of these schemes present challenges for IoT devices. Access control for IoT is complicated by the limited access to the operating system and device configurability. Due the resource constraints of most IoT devices, they cannot benefit from host-based access control schemes that enforce RBAC. Further, RBAC is too coarsely defined to accommodate the dynamic heterogeneity of IoT devices.

This limitation makes ABAC attractive approach. The cloud can impose ABAC access control to enforce fine-grained policies. The reliance on the cloud for access control removes transparency and policy enforcement at the user level. However, the ABAC approach shifts the entire onus of trust to cloud-based servers to enforce access control policies since they process, manage, and store the data produced by IoT devices. This presents a conflict as IoT technologies are often motivated by concerns over functionality rather than security. For example, Fernandes et al. [Fer16] discovered security-critical design flaws in the SmartThings model, identifying that 42% of SmartApps were granted capabilities that were not explicitly requested. Access control policies for IoT must be capable of restricting along fine-grained behavior capabilities (e.g., preventing a TV microphone from turning on without notifying the user first). However, to truly enforce policy governing access control, the user requires greater transparency of device capabilities and semantic behaviors. Further, access control requires an understanding how IoT devices buffer content when disconnected from the cloud. In the next section, we discuss the constraints that prevent IoT device users from realizing behavior level transparency.

2.2.4 Device and Behavior Transparency

Device and behavioral fingerprinting is a non-trivial but important problem for IoT to enforce access control. The pervasive use of encryption complications network intrusion detection. Encryption prevents the discerning of semantic behavior attributes from network traffic. Further, the reliance on the cloud obscures intra-device mediation, storage, processing, and control. Identification of device types and semantic behaviors from network-layer traffic is necessary to realize authentication and access control schemes for IoT. To this end, IoTSentinel [Mie17] uses static features such as protocol types, IP addresses, and port addresses to fingerprint device types. However, privacy-enhancing solutions, such as encrypted connections and the cloud, prevent the transparency of devices semantic behaviors. Thus, broad fingerprinting solutions are limited to destination agnostic flow information gained from network and transport layer packet headers. Both IoTSense [Bez18] and Peek-a-Boo [Aca18] leverage machine learning to classify device behaviors. However, both works are limited to fingerprinting known behaviors and do seek to identify new behaviors. Our work in hapter 4 proposes a content agnostic approach to behavior fingerprinting using as machine learning approach that can identify new behaviors. In Chapter 5, we propose an attack technique that leverages discerning between flows that enable the always-responsive nature of IoT against the on-demand services offloaded to the cloud. Next, we analyze three examples that motivate the challenges that accompany the cloud, access control, and behavior transparency.

2.2.5 Motivating Examples

Weeping Angel: Recent actions of the Central Intelligence Agency (CIA) demonstrate the substantial threats to IoT security and privacy introduced by the lack of device transparency. In March 2017, WikiLeaks published a file repository of CIA electronic surveillance activities. The files included information about the *Weeping Angel* implant, a CIA tool that enabled surreptitious recording of Samsung F Series smart televisions [Gre17b; FB17]. Government spies loaded the tool on the television through the *Extending Tool* USB implant, which compromised the operating system of the television. To further deceive the user, this malware disabled the LED lights to falsely indicate that the television was turned off. The repository also contained files describing the *Live Listen Tool* that enabled live audio exfiltration to government servers. The WikiLeaks release also described the agency's pursuit to compromise other IoT devices for the purpose of covert intelligence gathering activities.

Belkin Camera Vulnerability: Next, a recent flaw of the Belkin WeMo Baby Camera demonstrates the flawed access control of many IoT devices. Specifically, the Belkin WeMo Baby Camera allows any attacker to stream remote video if they have previously logged onto the same WiFi router as the camera [Dha15]. Once logged onto the same WiFi network as the device, an attacker can issue a simple service discovery protocol (SSDP) *M-Search* message to discover the location of the camera. Without authentication the attacker is able to query the device for configuration information, including the device serial number. Finally, the attacker issues an HTTP *POST /upnp/control/remoteaccess1* request with the target device's serial number and a self-chosen DeviceID. The serial number and self-chosen DeviceID are the only pieces of information required to authenticate to the device remotely and initiate a session initiation protocol (SIP) connection to the camera. A recent disclosure of a command injection vulnerability targeting the <*smartDevURL*> parameter of the camera escalated the threat from remote viewing without authentication to remote code execution without authentication [Har15]. Several other vulnerabilities (e.g. CVE-2013-6951, CVE-2013-6950) further escalate the threat posed by an attacker.

HomeKit Vulnerability: Finally, a recent vulnerability of the HomeKit platform demonstrates the challenge of securing devices that rely on cloud-based services. IoT devices often rely on cloud-servers for control, processing, or storage. The Apple HomeKit service seamlessly integrates a broad range of IoT device manufactures into the Apple ecosystem. HomeKit allows smart home IoT devices to interact autonomously or through the control of a Siri-enabled device. However, a recent server-side vulnerability allowed unauthorized control of accessories including smart locks and garage door openers [MIT17]. The security researcher that identified the vulnerability leveraged a bug in iOS 11.2 and watchOS 4.2 that leaked the unique identifiers and public and private keys of HomeKit enabled devices. With this pairing identity and private key, the researcher was able impersonate the HomeKit enabled devices. This enabled the researcher to issue commands to the HomeKit daemon to tell devices to perform specific behaviors (e.g. unlock smart locks.) This presents a challenging problem for defense. In terms of the smart home IoT device, it must assume the server-side platform is trusted after mutual authentication. When the server-side platform issues a mutual authenticated command, the device must execute. A server-side compromise to a smart home device could be mitigated by examining context. As an example, the Canary Security Camera, alerts users when remote video is accessed while the user is connected to the home WiFi. However, the heterogeneity and broad range of smart home devices ensures that many devices will fail to consider their context. In the case of the HomeKit compromise, locks should provide some form of notification to a user who has performed a remote unlock.

2.3 Machine Learning

Chapter 4 leverages machine learning (ML) to classify IoT semantic device behaviors. To this end, this section provides an overview of ML. ML is the study of algorithms that use statistical models instead of explicit rules or logic to perform tasks. ML can offer highly accurate threat detection when static rules and logic are unable to represent a problem. As applied to computer security, ML initially proved successful at classifying email spam [Lee10]. ML can overcome pervasive encryption to classify network flows [LL06; AZH09; AZH11]. Recent works have demonstrated precise application behavior classification, including identifying encrypted video stream traffic [Sch17], and fingerprinting NetFlix encrypted movie streams [RK17]. The task of classification consists of two high-level models of learning algorithms: supervised and unsupervised learning.

Supervised learning constructs a mathematical model from a set of data, containing both input samples and desired outputs. Outputs can consist of binary or multi-class labels to represent potential outcomes. The set of inputs are known as features. Selecting appropriate features (i.e., feature engineering) is an important aspect of any ML attempt to mitigate classification errors. Classification models balance two sources of error: *bias* and *variance*. Improperly selected or weighted features can create a bias that misses relationships between input samples and desired outputs. Bias, resulting from missing or under-weighted features, leads to an under-fitted model. In contrast, variance occurs when a model suffers sensitivity to noisy features in training data. Variance, resulting from noisy features, leads to an over-fitted model.

Unsupervised learning constructs a mathematical model from a set data that only contains input samples. Unsupervised learning discovers clusters by determining the distribution of data within a given space. Several factors contribute to the appropriate selection of a clustering algorithm. Particular algorithms are better suited for even sized clusters or a specific number of clusters. Others do better for uneven or an unknown number of clusters. As with supervised learning, feature selection contributes to the accuracy of unsupervised learning. In the following subsections, we highlight the benefits and disadvantages of different machine learning algorithms and discuss their applications for IoT security and privacy.

2.3.1 Supervised Learning Models

The selection of an appropriate supervised learning model requires balancing trade-offs associated with different models. In the following section, we examine a subset of machine learning models and their associated trade-offs. *K-Nearest Neighbors* (KNN) is a naive algorithm that classifies input samples on the similarity distance (i.e., Euclidean distance) to previously classified samples. K refers to the number of nearest neighbors used to measure similarity. A higher value of k can overcome over-fitting by ignoring the noise of mislabeled inputs. However, as k approaches the total number of samples, KNN suffers under-fitting by classifying all samples into one majority class. This makes KNN a poor algorithm for detecting new input sample classes. Further, KNN does not require time to construct a traditional predictive model. As such, it requires more time when classifying an input to compute the similarity distance. *Ensemble models* can reduce the bias and variance by combining several models into a voting classifier. Ensemble models can be furthered divided into models that are created sequentially or in parallel. Parallel (i.e., bagging) models reduce errors by averaging the outcomes of base learning models. Sequential (i.e., boosting) models can empower weak models by weighting previously mis-classified samples. *Random Forests* is an example of a parallel ensemble model [Bre01]. Random Forests consist of several randomized decision trees that are combined for classification. Each decision tree is constructed from a set of randomized features. By randomly selecting features, Random Forests provides a robust algorithm that is resistant to noise. Random Forests, like all ensemble models, uses averaging to control variance that contributes to over-fitting. *Gradient boosting* is an example of a sequential ensemble model [Fri02]. By building the model in stages, gradient boosting can empower weak base learners; however it does require substantial time to construct the model. Unfortunately, boosting-based ensemble classifiers are less resistant to over-fitting than bagging-based classifiers and can introduce variance into the model.

2.3.2 Unsupervised Learning Models

Next, we discuss the trade-offs between different unsupervised machine learning algorithms. First, *K-Means* is an unsupervised learning algorithm that separates samples into even sized clusters. This approach requires the number of clusters to be declared prior to running the algorithm. The algorithm iteratively tries to assign each sample to one of the clusters based on feature proximity to an *exemplar* in the cluster. Due to the approach, K-Means cannot represent uneven sized clusters. In contrast, *Affinity Propagation* offers an approach to uneven sized clusters. Affinity propagation converges on clusters created by iteratively sending messages between sample pairs. As the algorithm converges, it selects *exemplars* to represent each cluster. Affinity propagation succeeds for computer vision and computational biology tasks. However, the $O(N^2T)$ running time of the algorithm presents a challenge for data-sets with a large number of samples. Finally, *DBSCAN* offers a scalable algorithm for uneven clustering. Through density-based clustering, it assigns closely packed samples to clusters. As such, DBSCAN is less vulnerable to noise and robust to outliers. For these reasons, Chapter 4 of our work leverages DBSCAN to create initial clusters for seeding our work.

2.3.3 Scikit-Learn Framework and Evaluation

Our work leverages the *Scikit-Learn* [Ped11], a machine learning framework, for constructing and evaluating machine learning solutions. Scikit-Learn integrates a wide range of algorithms for both supervised and unsupervised problems under an easy-to-use Python interface. Scikit-Learn is a popular framework for ML research and offers seamless integration with other Python scientific modules, including *numpy*, *scipy*, and *cython.* Scikit-learn provides ease-of-use by abstracting machine learning to the Python high-level general purpose language. However, it overcomes the limitations of an interpreted language by leveraging C/C++ wrappers in *cython* [Jov14]. Further, the developers constructed Scikit-Learn with a strong emphasis on API consistency [Bui13].

Scikit-learn offers integrated model evaluation tools to score classification among different multi-class scoring criteria. Our analysis in Chapter 4 evaluates our solution using cross-validation scoring of accuracy, recall, and F1 score. These three scoring criteria provide a balance of different estimates for evaluating the classification precision of a given model. *Cross validation* scoring is a technique that measures prediction precision by partitioning and attempting to classify known samples. *Accuracy* measures the predicted labels for a given class over the sum of accurately predicted labels plus false positives. In contrast, *Recall* is computed by dividing accurately predicted labels by the sum of accurately predicted labels plus false negatives. Finally, *F1 score* offers a weighted average of the precision and recall.

2.4 Threat Model

Sections 3.2.2, 4.2 and 5.2.2 thoroughly examine our threat model and attacker goals for the systems described in Chapters 3-5. In Chapter 3, we consider an attacker that evades intrusion detection systems and bypasses network- and host-based access controls by leveraging stealthy strategies to exploit network blind spots. In this scenario, the attacker may abuse inter-host trust to laterally pivot through an otherwise blocked network route or service. In Chapter 4, we examine the threat of smart home devices executing unapproved behaviors by considering both behaviors from malicious adversaries and benign manufacturers that violate the explicit desires of the end-user. Our HomeSnitch security application protects against attacks via IP connection-oriented protocols. We consider devices that communicate via other protocols (e.g. ZigBee, Zwave, Bluetooth, NFC) as important and orthogonal problems for future work. In Chapter 5, we consider an attacker whose goal is to disable the core functionality of a device by blinding a sensor or confusing the state of an actuator. In all of these scenarios, our trusted computing base (TCB) includes our SDN security applications and the network data plane devices. Further, as discussed in Section 4.8, our work does not address the case of mimicry attacks. We leave attestation of our host agent in Chapter 3 and security applications in Chapters 3 and 4 as deployment tasks.

CHAPTER

3 PIVOTWALL

3.1 Introduction

The state of practice in network access control has remained largely static for decades. The policies that govern access are tedious to configure and prone to errors [Woo04]. As a result, network-security administrators spend a great deal of time writing policy and finely tuning rules. Despite this effort, the result involves low-level semantics, lacks context, and most often is enforced at the dividing perimeter between networks. This leaves controls ill equipped to defend against attacks that effectively originate from within the network. Such attacks arise from bring-your-own-device environments, insecure wireless networks, and compromised web browsers.

Advancements in Software Defined Networking (SDN) have made network access control more dynamic. Ethane [Cas07] advanced the concept of intra-network access control and evolved into the OpenFlow standard. SDN governs at host granularity while maintaining a centralized policy [Jaf12; Shi13a; Kam14; Yu17], and this versatility gives rise to flexible and reactive defenses that consider more than the perimeter. For example, SDN and Network Function Virtualization (NFV) can enable per-host quarantines, provide moving target defenses, and route traffic through stricter enterprise controls. Bates et al. demonstrated an SDN-based forensic system capable of identifying a number of previously unobservable attacks [Bat14]. Hardware switches and other devices from many manufacturers now support the OpenFlow specification; examples include devices from Hewlett-Packard, Cisco, and Pica8.

Operating systems security makes a distinction between access control and information-flow control.

The latter is more context-sensitive because it governs not only who can access information but also what they can do with information once it is accessed. Prior research has considered information flow tracking in a distributed setting [Zel08; Bac14]. However, more practical proposals [PP12; Sun16; Pap13] are limited to a cloud environment where there is tighter control over hosts and their communication. Pedigree [Ram09; Ram08] explored extending taint tags to an enterprise setting, but sacrificed security by probabilistically removing taint information.

We seek to extend information flow tracking out from a host and through the entire network. Similar to Pedigree [Ram09; Ram08], our goal is to reduce the semantic gap between host and network access controls, leading to security policies that better map to the governed activities. Specifically, we seek to extend information flow tracking as a defense against stepping stone attacks within enterprise networks. Such attacks evade traditional network defenses by compromising a series of hosts within a network, repeatedly pivoting laterally towards the final target. Of key importance, tracking flows between and within hosts enhances both real-time defense and post-incident forensics.

In this paper, we propose PivotWall, a novel network security defense that extends information-flow tracking on each host into network-level defenses. PivotWall identifies and defends against previously unobservable attacks through a novel combination of information-flow tracking and SDN's centralized management and intra-network controls. Our evaluation shows that PivotWall can detect a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. Furthermore, we show that PivotWall provides this protection while incurring minimal impact on network throughput and latency for untainted traffic and less than 58% overhead for tainted traffic.

This paper makes the following contributions:

- *We extend information flow control into the network using SDN.* PivotWall extends information flow control beyond the boundary of the host.
- *We propose a policy language for practically specifying information flow control within an enterprise network.* PivotWall's policy language syntax builds on the popular Snort IDS syntax and introduces unique actions that leverage the benefit of SDN technology.
- *We prevent attacks that bypass traditional enterprise defenses.* We demonstrate that moving the information-flow reference monitor to the SDN/NFV increases the context available to defenders during stepping-stone and other elaborate attacks.

The remainder of this paper proceeds as follows: Section 3.2 motivates our work using a recent real-world example. Section 3.3 overviews the PivotWall architecture. Section 3.4 describes its design. Section 3.5 evaluates accuracy and performance overhead. Section 3.6 discusses limitations. Section 3.7 provides an overview of related work. Section 3.8 concludes.

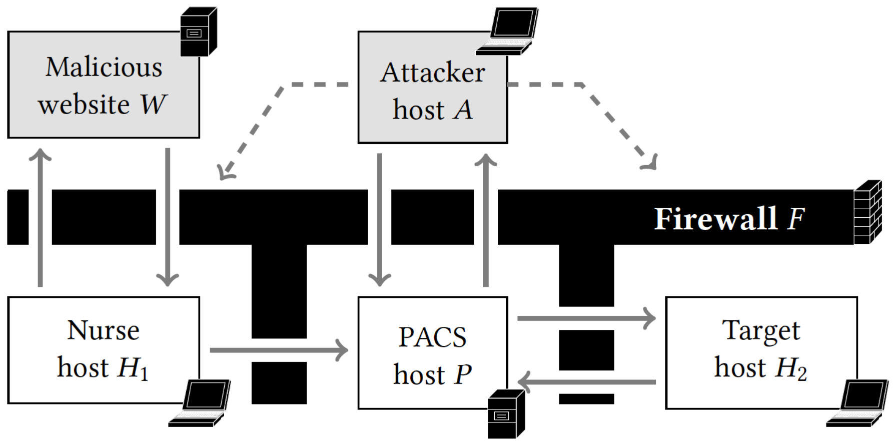


Figure 3.1 Scenario based on 2016 breach of hospital network

3.2 Motivation

PivotWall is motivated by advanced persistent threats (APTs) that use stepping stone attacks within an enterprise network. Such attacks compromise an initial host and then move laterally within the network, evading traditional network defenses. In this section, we provide the necessary background and intuition behind stepping stone attacks through a motivating example. The section concludes with a threat model for the PivotWall design.

3.2.1 Motivating Example

To illustrate the problem of stepping-stone attacks, we present a scenario based on a recent security breach in the healthcare industry. TrapX Research Labs highlighted the breach in a 2016 report [Sto15]. The victim (a hospital) used traditional enterprise defenses including a standard firewall, a heuristic-based intrusion detection system, endpoint security, and anti-virus software. Despite these security measures, attackers stole confidential data.

Figure 3.1 depicts a generalization of the compromised network. The figure includes five hosts and a firewall. The lower hosts are part of the hospital network, and the upper hosts are outside of the network and controlled by the attacker. We depict traffic as directed edges; dashed edges represent traffic blocked by the firewall.

This attack exemplifies a stepping-stone attack. First, the attackers compromised the web browser on a vulnerable workstation H_1 after the user of that workstation visited a malicious website W. Next, the attacker used H_1 to compromise a picture and archive communications system (PACS) P. Ultimately, the attacker compromises workstation H_2 and gains access to confidential data which he exfiltrates via P.

The firewall, F, in the scenario was most likely configured to prevent a direct flow from the Internet to H_1 and H_2. Yet P is less protected, because it facilitates the movement of medical imagery such as

X-rays throughout the hospital and its off-site offices. Thus information can freely flow between A and P

The difficulty arises because F cannot distinguish data that should flow between P and the Internet from data that should not. The context required for this decision is only available by examining a network information-flow control (NIFC) graph that spans both the activity within P and H_2 as well as messages between hosts. Only with this information could F block the confidential traffic before it flows between P and A while permitting benign interaction between P and other hosts. Put another way, information should be able to flow between H_2 and P or between an Internet host and P, but not from H_2 to a host on the Internet by way of P.

In summary, traditional approaches lack the knowledge gained from a NIFC graph containing the flow of confidential access and data throughout the network, whether between or inside hosts. This attack can be stopped by applying data labels and implementing flow control across host boundaries. We capture this conceptual approach in PivotWall, a novel enterprise security architecture that combines the logically central placement of the SDN with the context of host-based information flow tracking.

3.2.2 Threat Model and Assumptions

Our threat model assumes the attacker's goal is to obtain confidential information. To achieve this goal, attackers must evade intrusion detection systems and bypass network- and host-based access controls. To achieve this result, we assume the attacker will use stealthy strategies that abuse trust by laterally pivoting through blind spots in the network [Yu17]. For example, an external attacker might pivot through an employee workstation to launch an internal attack that avoids the enterprise controls at the perimeter of the network. Similarly, an inside attacker might exfiltrate confidential data from a protected service to a globally-accessible server by routing through blind spots. An attacker might create covert or stealthy channels by abusing legitimate protocols including application layer protocols (e.g, Gmail, Slack, Twitter [Ama16]) and network layer protocols (e.g., TCP, ICMP, DNS [ML05; Zan07; Sha16].) An attacker might use tools such as the Data Exfiltration Toolkit [Ama16] or DNSCat2 [Bow16] to create these channels. The goal of PivotWall is to prevent these and other stealthy attacks that abuse blind spots in defenses.

Our trusted computing base (TCB) includes the SDN security application, the network data plane devices, and each host's core operating system (i.e., kernel, core system daemons, and our host-agent). We do not protect against malicious-but-trusted administrators that can detect the host-agent, remove confidentiality labels on data and processes, or disable the host-agent packet labeling implementation. Similarly, we assume networking devices are not compromised to deactivate defense mechanisms. The TCB extends to other SDN applications running on the controller that are not segregated from the PivotWall security application. We consider the attestation of the host-agent, the security application, and enterprise networking devices as important but orthogonal problems. We leave attestation as a deployment task.

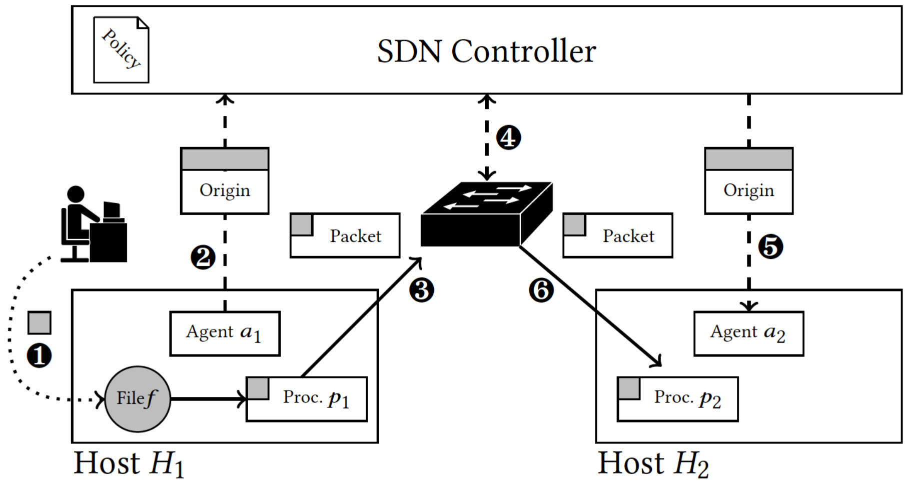


Figure 3.2 Overview: PivotWall Information Flow Control

3.3 Overview

We designed PivotWall to extend information flow tracking across hosts in a distributed environment. PivotWall broadens the enforcement of secrecy by establishing information-flow controls at the SDN controller. This section describes PivotWall's architecture, addresses the identified challenges in centralizing information-flow controls, and discusses the key ideas of PivotWall.

3.3.1 Architecture Overview

The architecture of PivotWall involves three components:

Host agent: Each host on a PivotWall network is governed by a modified SimpleFlow [Joh16] kernel. SimpleFlow tracks the flow of confidential information on a host. SimpleFlow labels packets which emanate from processes that might have read confidential information, and it taints processes that read a packet bearing a confidential label. The PivotWall host agent maintains provenance and sends control messages containing the origin to the controller as described in Section 3.4.2.

Network control plane and SDN controller: A lightweight *Pox* OpenFlow security application creates the necessary network flow modifications that deliver flows bearing confidential packets to the control plane for inspection. At the control plane, the security application implements the policy store, network information flow control (NIFC) graph, and reference monitor. The security application uses these components to inspect flows for a violation, and it implements eight primitive actions for handling confidential flows. These actions include unique methods for redirecting, throttling, or modifying confidential flows as described in Section 3.4.1.

Network data plane: The network hardware implements the OpenFlow flow modifications to deliver labeled flows to the control plane for inspection. The network hardware modifies flows based on the instructions from the SDN controller.

Figure 3.2 summarizes the PivotWall architecture. An administrator has labelled the file f on host H_1 as confidential (❶). When process p_1 reads from this file, SimpleFlow taints process p_1. As process p_1 establishes a network flow from $H_1 \rightarrow H_2$, the PivotWall agent on host H_1 notifies the SDN controller of the unique origin label for the flow (❷). Subsequently, process p_1 writes the confidential information to the network in the form of a labelled packet (❸). Upon receiving this packet, the switch observes that it is labeled and thus queries the SDN controller (❹). The SDN controller acts as a reference monitor and examines the confidentiality, origin, and steering labels against indexed network information flow control (NIFC) graphs. The controller governs the return, mutation, dropping, or redirection of packets based on its configured policy. Here the SDN controller notifies host H_2 of the source of the packet (❺), and it delivers the packet (❻). Finally, SimpleFlow taints process p_2.

3.3.2 Challenges

PivotWall controls the flow of information in a distributed network but is governed by a centralized policy. Practical information flow tracking in a distributed environment requires overcoming the following challenges:

Practical policy enforcement: Precision is a challenge for information flow tracking in a network environment. A lack of accuracy can cause label propagation to fail, violating secrecy. Conversely, coarse precision can cause false positives leading to *taint explosion*.

Label Integrity: While information flow tracking within hosts is well studied, tracking information flows across networks has been limited to statistical measures that break down under even normal operations [Pen05; CM07; Shu11]. Statistical correlation approaches fail when data is compressed, encrypted, or delayed at the host. Furthermore, broadcasting the mandatory protection state of data does not scale.

Attribution: Determining the origin of a policy violation is challenging in a distributed environment. An attacker can take several intermediary steps on the host and the network to conceal their origin. Thus, implementing the *intent* of the policy is challenging without understanding the origin of the data encapsulated in a flow.

3.3.3 Key Ideas in PivotWall

PivotWall addresses the aforementioned challenges through the following key design concepts:

Network Reference Monitor: To achieve dynamic taint analysis at the network layer, PivotWall extends the classical host-centric reference monitor to establish a network access control enforcement mechanism.

Under PivotWall, each network device modifies confidential flows to first pass through the SDN controller application. There PivotWall's reference monitor determines whether or how traffic may flow through the network device. While a typical reference monitor returns a binary response, PivotWall offers a range of responses. The reference monitor only evaluates traffic marked as confidential; non-confidential traffic is not evaluated.

Practical Policy Grammar: A single policy governs the reference monitor's decisions. PivotWall provides the three necessary properties of dynamic taint tracking—namely, *taint sources*, *taint sinks* and *taint propagation rules*—with a policy grammar focused on performance, simplicity, and flexibility. The grammar consists of a series of clear and concise *Snort-like* rules.

PivotWall's policy language expresses *taint sources* as the input source where confidential data entered the network. A taint source can represent a single host, a subnet of hosts, or any hosts in the exclusion of a host or subnet. PivotWall's policy represents *taint sinks* using the same syntax and describes the location where PivotWall applies actions outlined by the rule. Finally, PivotWall defines *taint propagation* rules, describing which action must be applied when data from a particular taint source reaches a taint sink. This key contribution by PivotWall allows a trusted administrator to express practical policies for reactive, fine-grained-modification on a per-flow basis. Rules include typical intrusion detection and prevention systems actions, such as the ability to *pass*, *log*, *drop*, *reject*, or *alert* on network flows. Furthermore, the use of SDN allows the ability to *redirect*, *throttle*, and *rewrite* network flows. Section 3.4.1 expands upon the policy language design and the implementation for reactive actions. Reactive taint propagation rules provides a flexible means of mitigating a wide variety of attack vectors against confidential data and processes.

Persistent Labels: Information flow tracking across distributed hosts can fail when data is transformed intentionally or unintentionally to remove confidential labels. PivotWall overcomes this limitation by establishing distributed persistent labels that seamlessly transfer between the host and the network layer. Through distributed persistent labeling, PivotWall establishes a mandatory protection system complete with labeling, protection, and transition states that cross over the boundary of the host and network. In contrast to traditional MLS models [Den76; BL73], PivotWall adopts an approach an approach similar to taint tracking systems such as TaintDroid [Enc14] where the label indicates if the data contains confidential information of a specific type.

PivotWall tracks information flow within each host on its network using SimpleFlow [Joh16], an information-flow-based access-control system built within the Linux kernel. SimpleFlow permits a trusted administrator to label system objects—such as files, sockets, or pseudo-terminals—as confidential. Processes on a SimpleFlow host that read from confidential objects become tainted, and this taint status follows the writes and reads that result in the flow of confidential data through the system. For example, a process that reads from a pipe shared with a tainted process will itself become tainted. Under SimpleFlow, the Linux kernel will mark any packet written to the network by a tainted process. Processes that read from marked network packets also become tainted. SimpleFlow covers a wide range of system calls [Joh16], and addresses some of the high-bandwidth covert channels found in Unix [Joh16].

PivotWall extends SimpleFlow so that its network filter component further labels traffic with a network taint byte.[1] The network taint byte contains information used by PivotWall's SDN controller to steer the packet through a software-defined network. Combining SimpleFlow with the SDN-based steering label allows labels to persist across in-host and network communication and thus extends information-flow tracking to the network. It also presents the opportunity to create increased expressibility in policy language as we examine in the following section.

Network Information Flow Control (NIFC) Graph: In a simple classical access enforcement mechanism, data is labeled with a single bit which represents notion such as confidentiality or trusted. The reference monitor interface is responsible for querying the policy store to authorize a request. This binary label presents a challenge when implementing a practical policy enforcement mechanism that queries characteristics such as the origin of the confidential data. Practical policy enforcement requires the ability to examine the entire path of a confidentially-labeled object. PivotWall stores the flow of a confidentially-labeled object in a directed graph $G = (V, E)$. The vertices, V, of the NIFC graph represent the union of the set of subjects and objects that have interacted with a single confidentially-labeled object. The set of edges E, represent the flow of information from data in a protection state to a new object. These edges represent network flows to a new host. Representing the confidential data flow as an NIFC graph allows PivotWall to offer heuristics about how particular confidential data violated a policy. Although a simple scenario may result in a single path, complex scenarios may include an attacker attempting multiple data ex-filtration points. Representing the flow of confidential data as an NIFC offers visibility of the extent of the attack and key articulation points that extend bridges to the network border. This insight allows an administer to apply tighter controls on the hosts discovered as articulation points. Section 3.4.1 expands upon the design of the NIFC graph and the heuristics for constructing the complete path of confidential data prior to the policy violation.

3.4 PivotWall

The goal of PivotWall is to protect the secrecy of data or processes by extending information flow tracking to a distributed architecture. We focus on establishing a logically central SDN controller to manage distributed information flow control through a reference monitor, policy store, and a network information flow control (NIFC) graph. A key idea in PivotWall is to maintain a NIFC graph that maintains the provenance propagation of confidential objects. By establishing a NIFC graph, we provide the ability for a system administrator to write rules to prevent attacks that span across multiple connected flows.

[1]The original SimpleFlow design blocked tainted packets at the subnet boundary.

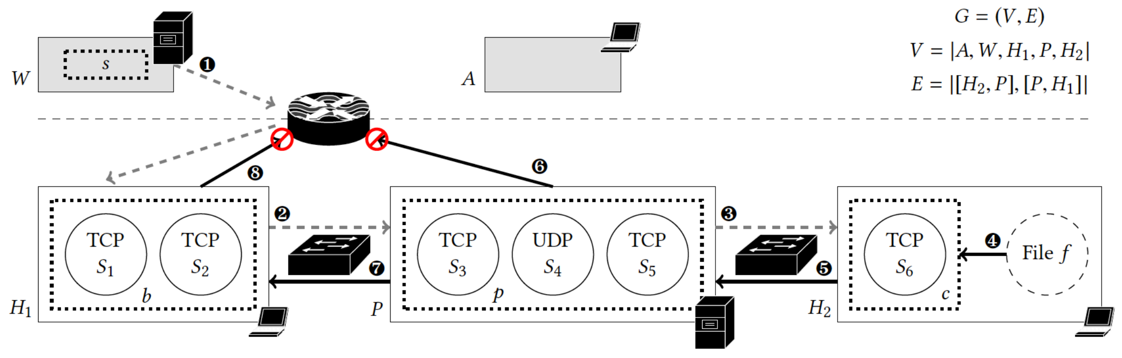


Figure 3.3 Example actions that build an NIFC graph

3.4.1 Network Layer Design

SDN eases network administration by combining a centralized policy with distributed enforcement. OpenFlow-enabled hardware consults a centralized controller to determine how to handle the flows it processes. In the case of PivotWall, this controller application implements a variation of the classical reference monitor to govern network flows. Whereas a classical reference monitor can only permit or deny, PivotWall can leverage SDN technology to throttle, modify, drop, reject, or redirect network flows. An administrator defines these actions by writing a policy, and PivotWall's controller application enforces this policy.

3.4.1.1 Network Information Flow Control Graph

A key insight of PivotWall is its use of a provenance graph to reduce or limit false positives and taint explosion. In contrast, Pedigree [Ram09; Ram08] labeled confidential information with a taint tag, which can lead to taint explosion as the network becomes strongly connected. Pedigree addressed taint explosion by probablastically removing taint bits; however, this design choice sacrifices security.

PivotWall addresses this challenge by maintaining graphs that represent each information flow with a confidential system object (e.g., file, pipe, socket) as its source. This leads to more precise tracking while revealing many sophisticated attacks.

The NIFC is a directed graph $G = (V, E)$ containing the set of vertices $V = \{v1, v2, ...\}$ and set of edges $E = \{e1, e2, ...\}$. The graph represents the flow of a confidential object throughout the network. Each vertex v represents a single host or server that has processed, accessed, or stored a confidential object. Each directed edge e represents the network flows between hosts that have carried or provided access to the confidential object. Algorithm 1 depicts how PivotWall adds vertices and edges to a NIFC. First PivotWall checks the policy to determine if the packet source, destination, and protocol are permitted by the policy (1). The algorithm records the original source and creates a unique graph for a previously unseen confidential object (2−4). The algorithm next checks if the policy permits the path an object has taken by comparing the original source against the packet destination (6). If the policy permits this path, then a directed edge is added from the source and destination in the packet header (7), a per-flow rule is installed (8), and the packet is forwarded to the destination (9). In the case that the policy does not permit the action, the packet is dropped (11) or modified in accordance with the policy. We depict an example graph in the top right hand corner of Figure 3.3.

Figure 3.3 overlays the NIFC graph on the scenario in Figure 3.1, depicting the hosts W, A, H_1, P, and H_2 as rectangles. Overlaid upon this is a graph: processes s, b, p, and c (dotted rectangles) and file f (dashed circle) are the graph's nodes, and the flow of information makes up its directed edges. Also depicted as circles are six sockets S_1–S_6 that facilitate network communication.

An attacker compromises host H_1 after browser process b visits a malicious website (❶). The payload makes a connection to host P and compromises the PACS server process p (❷). Process p then compromises the PACS client process c on host H_2 (❸). The compromised process c reads the confidential file f (❹) and transmits it over the network to host P (❺). Host P then attempts to exfiltrate the file under the cover of a DNS query to host A (❻). Sensing that this was thwarted by PivotWall's SDN controller, the illicit software transmits the confidential information back to host H_1 (❼), but the SDN controller again blocks the exfiltration attempt (❽).

Algorithm 1: HandleTaintedPkt

```
   Input: A tainted packet (p), and the UUID (u)
 1 if $PolicyPermitsPacket(p)$ then
 2     if $u$ not in taintsTable then
 3         $taintsTable[u] \leftarrow [p.src, p.sport]$
 4         $graphTable[u] \leftarrow new\ Directed\ Graph()$
 5     if $PolicyPermitsPath(p, u)$ then
 6         if $edge(p.src, p.dst)$ not in $graphTable[u].edges$ then
 7             $graphTable[u].add\_edge(p.src, p.dst)$
 8         $installFlowRule(p)$
 9         $send(p)$
10         return
11 $drop(p)$
12 return
```

The directed graphs maintained by PivotWall inform the SDN controller as it governs the network. A graph indicates a tainted flow, and it provides context into how the confidential information passed through the network. The graphs exist in a hash table that is indexed by the UUID of the confidential

Listing 3.1 Simple PivotWall rule preventing exfiltration of data from a specific host

```
$HOST1  = 10.1.1.1
$HOME   = 10.1.1.0/24
drop tcp $HOST1 any -> !$HOME any
```

Listing 3.2 Rule For Redirecting Traffic to a HoneyPot

```
$HONEY = 10.1.1.4
redirect tcp $HOST1 any -> !$HOME any (rdst=$HONEY)
```

source object (described in Section 3.4.2). In the example here, the solid black edges represent the graph which PivotWall would correspond with the source f. The dashed gray edges represent incidental communication. Section 3.4.2 describes how PivotWall bridges between host-based information-flow tracking (e.g., tracking a `read` of f) and network-based information-flow tracking.

Storing the provenance history in a directed graph allows for a more expressive rule syntax for defining network policy. The presence of the information-flow graph means that the rules that govern network flows can consider the origin of a flow. For example, the denial at ❻ could realize the confidential source and redirect the DNS request to a sinkhole which would aid in a forensic analysis. The denial at ❽ could reduce the throughput of the exfiltration channel to the point of being useless. Very similar flows which do not contain illicit information could be allowed to pass.

3.4.1.2 Customizable Rules

A key benefit of PivotWall is its ability to consider the provenance propagation of confidential objects when dynamically analyzing flows. To achieve this in a flexible way, PivotWall provides a syntax for specifying the rules that govern flows. Unlike traditional intrusion detection systems, PivotWall examines the entire path of confidential flows when matching a rule pattern. Upon a successful match, PivotWall can respond with the eight base actions depicted in Table 3.1.

The syntax that specifies the most basic rules resembles Snort and references a flow's protocol, direction, and port to determine an action [Roe99]. For example, Listing 3.1 prevents confidential data originating from `HOST1` from egressing outside the local area network by dropping packets. This rule implements data loss prevention with full information flow context to prevent an internal attacker at `HOST1` from ex-filtrating confidential data using stepping stones inside the network.

Similar to Snort, PivotWall interprets option fields inside parenthesis. The rule in Listing 3.2 redirects confidential traffic to a honeypot based on the redirect destination option (`rdst`) specified in the rule. As seen in both examples, PivotWall offers matching a single IP address, host name, or CIDR notation. Variables can be used to easily construct rules. The '!' operator can be used to denote traffic outside of a particular network. Each rule begins with a unique action to handle the violation created by the flow.

Table 3.1 Customizable actions for PivotWall

Action	Description
pass	Permit confidential flow; keep confidential labels intact
untag	Permit confidential flow; remove confidential labels
log	Log packets from confidential flow to pcap
drop	Drop packets from confidential flow and log
reject	Drop packets from flow, log, and send TCP Reset of ICMP unreachable
redirect	Redirect flow using alternate destination or source
slow	Rate throttle packets using TCP congestion window or queuing
modify	Permit flow but rewrite packets based on customizable script

Managing the defense at an SDN controller introduces several unique and different response methods. Since the controller manages the data plane of the network, the controller has the capability to communicate with all enterprise network devices in response to an attack. By sending control messages across the data plane, the SDN controller can near instantaneously communicate and propagate knowledge of a threat to all managed enterprise network devices. PivotWall offers typical basic actions for responding to violations as well as custom-tailored advanced actions.

Basic Actions: PivotWall provides five primitive actions to implement security directives to implement information flow tracking. The *log, drop,* and *reject* actions provide similar functionality to their Snort counterpart actions and are described in Table 3.1. Because the *pass* and *untag* options interact with the labels, they require further discussion. *Pass* permits confidential flows and leaves the network, host, and origin labels intact on the packet headers of the flow. In contrast, *untag* permits the confidential flow, removing the network, host, and origin labels from packets in the flow. Essentially, the pass action propagates labels to the next hop while the untag action removes the confidentiality labels from the flow.

Advanced Actions: To specifically address secrecy and integrity attacks, PivotWall implements three advanced action primitives for redirecting, throttling, isolating, and modifying flows. SDN-enabled traffic redirection has shown promise as a means of mitigating threats by redirecting traffic to a honeypot, sink-hole, or hiding critical resources [Shi13a; Shi13b; Heo14; MS15].

Slow and *modify* are actions that shape traffic and mitigate secrecy and integrity attacks. *Slow* throttles the flow by artificially queuing the flow of IP packets. When possible, the *slow* action throttles the transport layer protocol by rewriting packets to reduce the TCP Congestion Window Size. Reducing the throughput of transport layer protocols degrades the effectiveness of the attack channel thus slowing the propagation of an attack. Essentially, the *Slow* action uses the benefit of SDN technology to implement the concept of a *tarpit* [Yeg04]. By keeping the attacker channel open but unusable, an analyst can investigate an attack in real-time.

The *modify* action reduces the impact of false positives while not decreasing true positives. Consider

Listing 3.3 Modify-Script for Mitigating Covert ICMP

```
rewrite_ping_req(packet):
  ipkt=packet.find("icmp")
  if (ipkt):
    if (ipkt.type == ICMP.TYPE_ECHO_REQUEST):
      ipkt.payload = "A"*len(ipkt.payload)
      packet.payload = ipkt
  return packet
```

the specific cases where an administrator may write a rule that has a high false positive rate (e.g., HTTP, DNS, ICMP traffic). The *modify* action permits the administrator to allow the flow of confidential data but eliminates optional fields that might carry the confidential data. To achieve this, the administrator defines the handling of matched packets with a custom *modify*-script.

To illustrate the utility of the *modify* action, consider a simple use case of ICMP covert traffic. Daemon9 [Dae96] proposed using `ICMP Echo Requests` to carry confidential data in 1996 by embedding data in the an `ICMP Echo Request` payload. Several common data exfiltration tools still use this technique [Inc16]. Listing 3.3 provides a PivotWall *modify*-script that removes covert data from the payload of an `ICMP Echo Request` by rewriting the payload field entirely with the letter *A*.

PivotWall *modify*-scripts provide the customization to address secrecy attacks while limiting the impact on benign traffic on the protocol flows. Simply, PivotWall *modify*-scripts define how packets of a confidential flow are modified to remove covert channels.

3.4.2 Host Design

In order to classify packets that are sent from a host, PivotWall must track flows within that host. Intra-host flows and network flows are inherently different. Flows within a host result primarily from software that loads, generates, transforms, or stores information. In contrast, network flows have more to do with transporting information from one host to another. While information might change in the network, manipulation is generally performed merely to allow information to traverse the network.

Intra-host flows result from user activities and therefore model user intentions and the effect they have on confidential information within the host. Such flows provide links between processes, and they most commonly result from processes invoking system calls such as opens, reads, and writes. One process might concatenate two files to a third, another might read information from the network and write it to a file, and yet another might communicate with a peer process using a pipe. In any case, these flows have the potential of moving confidential information from file to file, from network socket to file, or from process to process through a pipe.

Host Flow Tracking: Whole-system provenance, or the tracking of confidential data throughout the host, is a challenging problem. Several promising solutions rely on the Linux Provenance Module (LPM) interface to gain whole-system provenance and handle side channel attacks [Bat15; Joh16]. PivotWall builds upon SimpleFlow [Joh16] to track the host information flow. As a Linux kernel modification based on the Linux Security Module (LSM) interface, SimpleFlow removes many of the race conditions present in user-space monitoring tools or tools that combine a provenance engine with a separate access-control mechanism.

Labeling: A key component of the PivotWall design is to maintain persistent labels that extend beyond the boundary of the host. When a file is sent over the network or a process is accessed via the network, the label must propagate to the associated network flow. The controller must be able to determine the confidentiality of the traffic and the origin of the confidentiality to match the appropriate policy. SimpleFlow uses a *Netfilter* interface in the kernel to apply a confidential *host label*. This label is a binary representation that depicts if the packet contains classified data. The *host label* is created by borrowing the extra bit in the IP fragment field (identified as the evil bit in RFC 3514 [Bel03]).A packet bearing the evil bit is handled as confidential by any SimpleFlow enabled hosts. However, to extend this label to the network, we modified the SimpleFlow source by adding a network label.

The *network label* provides the steering information which affects how the PivotWall reference monitor should handle packets. The *network label* borrows the IP Type of Service (ToS) header byte. As identified by Fayazbakhsh et al. [Fay13], the IP ToS header provides a full byte that we use to create matching on-demand flow modifications that work with all legacy OpenFlow hardware. Packets bearing the *network label* are brought to the controller on a per-flow basis. However, to apply policy and create flow modifications, PivotWall requires the origin of the confidential packets.

The *origin label* uses a 128-bit universally unique identifier (UUID) to represent a global identifier for each classified object. In our initial design, we padded each packet with the UUID as a Commercial Security Option Type (6) IP Option. However as described in [Fon05], IP Options are rarely well supported. Further, our initial experiment demonstrated that adding 128-bits to every classified packet introduced performance consequences. To ensure compatibility and performance, our design labels network flows by sending the 128-bit UUID in a control message encapsulated in an ICMP packet. While our current implementation has the host agent send this message directly to the controller, it would be straightforward for the OpenFlow switch to capture the control message and forward it to the controller (e.g., in the case of an out-of-band controller). *Pedigree* [Ram09; Ram08] also uses a separate connection to pass provenance data to their network arbitrator.

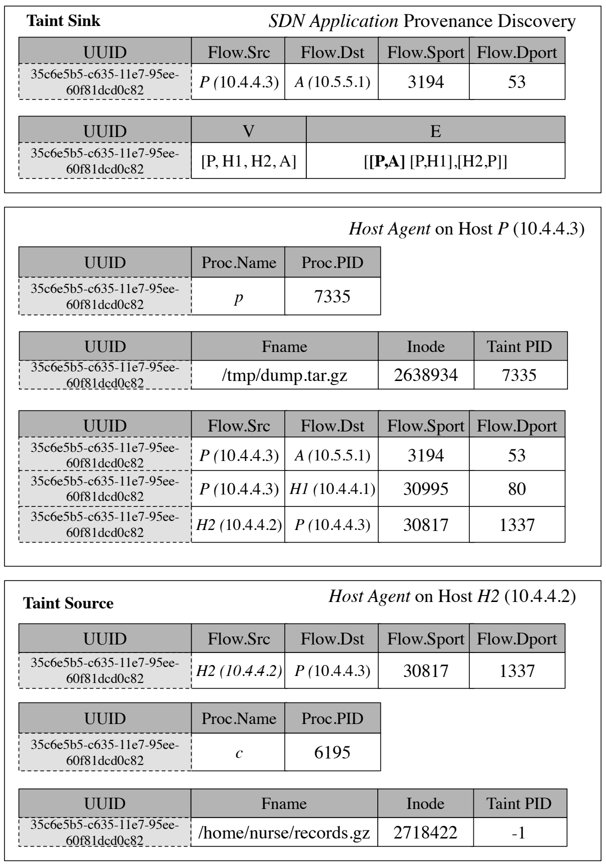
An *origin-label* is generated when an administrator initially labels a file or process as confidential. The host agent propagates the UUID for each interaction with the confidential-labeled file. Consider the example where a process copies data from a confidential-labeled file to a new file. As a result, both the tainted process and the newly-tainted file bear the *origin-label* of the original file. To implement NIFC, these unique identifiers extend beyond the boundary of the host. PivotWall accomplishes this by sending the UUID for each tainted network-flow to the PivotWall application at the controller. The application updates the NIFC graph, indexed by the UUID. Further, PivotWall sends a control message with the UUID to the destination host. The host agent propagates the UUID from the network flow to interactions on the host.

It is possible that a single object might be associated with multiple UUIDs. Consider when two confidential files are merged into a new file. Or a remote access toolkit process may attempt to ex-filtrate multiple confidential files over the same network flow. When multiple confidential objects are merged, the host agent assigns a new UUID to the combination. The host agent sends the updated UUID in a control message to the PivotWall application at the controller. Further, the controller application sends a control message containing the updated UUID to the destination host. The controller creates a new NIFC graph from the merger of the NIFC graphs from each object. Since the new NIFC graph is used to check for policy violations, PivotWall deletes the flow modifications for all edges of the merged graph. This action brings all network flows (represented as edges) back to the controller to re-examine the policy implications of the UUID combination.

3.4.3 Forensic Analysis

The PivotWall taint propagation logs are a valuable resource when performing forensics in response to an attack. Because PivotWall uses taint analysis to detect secrecy attacks, the logs within the SDN application and the host agent can be used to create a provenance graph describing how the attack progressed. PivotWall raises an alarm for the last hop of an attacker's chain. Therefore, a forensics tool can walk backwards to identify all of the hosts and resources that led to the alarm. Both the network and the host maintain provenance data about taints in SQLite databases. Therefore, tainted objects can be easily identified using SQL queries. Producing this set and aggregating the data over the network provides an incident response team and forensic analysts valuable information to contain the attack and remove the attacker from the internal network. The PivotWall heuristics provide visibility of the taint source, the intermediate hosts involved, and the series of intermediate tainted objects (processes, files, and network connections).

Figure 3.4 abstracts the queries necessary to offer insight into the motivating example attack in Section 3.2.1. The first query begins at host P, where a policy violation occurred on the flow $P \rightarrow A$. Examining the NIFC graph, an analyst can determine this violated policy because a path exists from $H_2 \rightarrow A$. Examining the host agent on P, provides information about the specifically tainted processes, files, and network flows. Examining these queries, we determine that the process p propagated the attack and wrote a file named *dump.tar.gz*. Following the graph back to the taint sink (H_2), we determine that process c read the source of the origin taint file *records.gz*. Ultimately, these heuristics and queries provide an administrator the necessary information to determine the anatomy of the attack and the incidental elements of the attack.



Taint Sink — *SDN Application* Provenance Discovery

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*A* (10.5.5.1)	3194	53

UUID	V	E
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	[P, H1, H2, A]	[**[P,A]** [P,H1],[H2,P]]

Host Agent on Host *P* (10.4.4.3)

UUID	Proc.Name	Proc.PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*p*	7335

UUID	Fname	Inode	Taint PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	/tmp/dump.tar.gz	2638934	7335

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*A* (10.5.5.1)	3194	53
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*H1* (10.4.4.1)	30995	80
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*H2* (10.4.4.2)	*P* (10.4.4.3)	30817	1337

Taint Source — *Host Agent* on Host *H2* (10.4.4.2)

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*H2 (10.4.4.2)*	*P* (10.4.4.3)	30817	1337

UUID	Proc.Name	Proc.PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*c*	6195

UUID	Fname	Inode	Taint PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	/home/nurse/records.gz	2718422	-1

Figure 3.4 Provenance discovery enabled by information collected by the controller (NIFC graph) and host (process and file information).

3.5 Performance Evaluation

PivotWall extends information flow tracking to the network by leveraging the logically central placement of the SDN controller. By extending traditional host-based information-flow tracking to the network, PivotWall prevents attacks that bypass existing enterprise defense mechanisms. In this section, we empirically evaluate the design and performance of our prototype by answering the following research questions.

RQ1 (*Stealthy Attack Detection*): Can PivotWall detect attacks that otherwise bypass traditional enterprise defense mechanisms?

RQ2 (*Attack Coverage*): Can PivotWall detect against a broad coverage of communication channels and tools?

RQ3 (*Network Scalability*): What is the scalability of the network controller application?

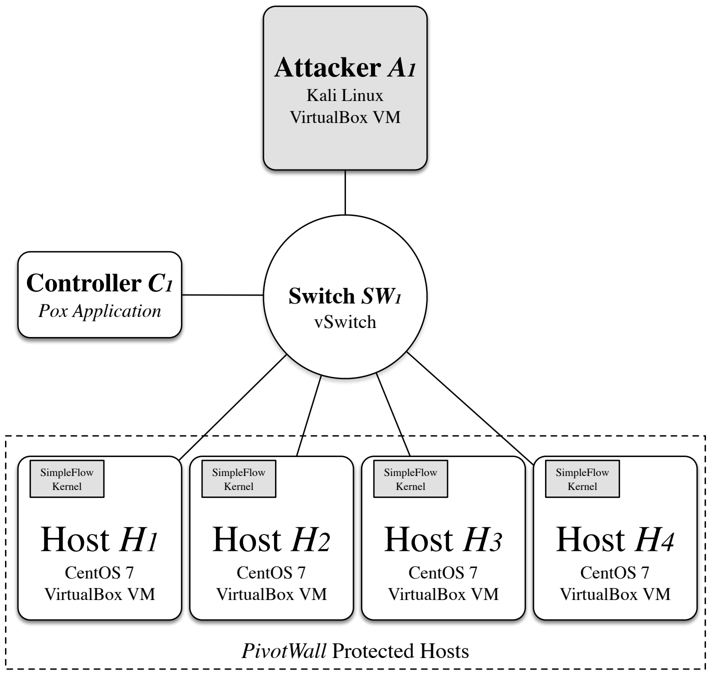


Figure 3.5 Diagram of testbed network

RQ4 (*Response Capability*): Can an administrator extend PivotWall to provide a custom-tailored, flexible response to an attack?

Experimental Setup: Figure 3.5 illustrates the testbed network for our experiments. We created a virtual network infrastructure environment on a 2.8 GHz Intel Core i7 CPU with 12GB of memory running Ubuntu 16.04.2 LTS, an OpenFlow control platform, Open vSwitch 2.7.90 and the Pox 0.2.0 controller running our PivotWall security application. Our hosts protected by PivotWall are VirtualBox instances with 512MB of RAM running CentOS 7, SimpleFlow 0.3.0, and our PivotWall packet relabeling agent. Each host is connected to a virtualized network, via soft SDN switches (Open vSwitch) that support OpenFlow. Our attacker host is a VirtualBox VM with 512MB of RAM running Ubuntu 16.04.2 LTS.

3.5.1 RQ1: Stealthy Attack Detection

Attackers employ stealthy attacks [Yu17] to exploit blind spots in enterprise defenses to circumvent traditional defense mechanisms. We evaluated PivotWall against stealthy attack tools to demonstrate how our prototype defends against attacks that bypass traditional enterprise defenses. Our evaluation considered two specific stealthy attack scenarios. We first examined a data-loss-prevention (DLP) scenario where the attacker originated inside the network. Next, we examined a stepping-stone-attack scenario where the attacker originated external to the network to gain access to confidential data. In both scenarios,

Table 3.2 Stealthy data loss evaluation tests

Toolkit	Encrypted	Model	Snort IDS Detected	Ipfwadm Firewall Detected	PivotWall Detected
Netcat	N	Store-Forward	N	N	Y
Cryptcat	Y	Store-Forward	N	N	Y
Socat Forward	N	Active-Relay	N	N	Y
Meterpeter Port Forward	Y	Active-Relay	N	N	Y

Listing 3.4 PivotWall Detection of Netcat Store-and-Forward

```
PivotWall: Received UUID=b0815bee-b82b-11e7-b120-60f81dcd0c82 for
   10.1.1.2:55848->10.5.5.1:1337
PivotWall: UUID=b0815bee-b82b-11e7-b120-60f81dcd0c82 originates at
   10.1.1.1
PivotWall: Detected New Labeled Flow From 10.1.1.2->10.5.5.1, 55848,
   1337
PivotWall: Rule Matches: drop tcp 10.1.1.1 any -> !10.1.1.0/24 any
PivotWall: Rule Action: drop, sending flow modification for
   10.1.1.2:55848,10.5.5.1:1337
```

we compared PivotWall's ability to detect and mitigate attacks in comparison to traditional defenses.

Stealthy Data Loss Prevention (DLP) Attack: To understand the benefits of PivotWall, we compared it to two traditional enterprise defenses: a network intrusion detection system (*Snort*) and a host-based firewall (*iptables*). We applied the traditional defenses in a conservative manner with the goal of blocking communication from a protected host to an external network. On the IDS, we applied an IDS rule to block any TCP packet from our confidential host (10.10.1.1) to a destination outside our local network (!10.10.1.0/24). At the host firewall, we applied an *iptables* rule to block any TCP packets to a destination outside our local network.

Using the tools listed in Table 3.2, we modeled a stealthy attacker that passed data from the protected host to the intermediary host on the local network. Although these four tools can be used in multiple ways, we tested configurations that maximized coverage between the variables of encryption and relay model. We transferred and stored data on the intermediary host by using netcat and cryptcat before ultimately egressing the data (store-forward). In contrast, we used socat and meterpreter to forward TCP ports, actively relaying the data through the intermediate host to egress the data.

In all cases, the network intrusion detection system and the host-based firewall failed to detect the data loss. Both conservatively applied rules failed to detect the information flow through the intermediary host. To demonstrate PivotWall, we labeled a file as confidential, and applied a simple policy rule that prevented confidential information with an origin of 10.10.1.1 to flow beyond the boundary of the local

Table 3.3 Coverage of DET plugin attacks

Plugin	Description	Transport Protocol	Pivotwall Detected
DNS	DNS Record Request	UDP/TCP	Y
Gmail	B64-encoded e-mail	TCP	Y
Google_Docs	B64-encoded parameterized URL	TCP	Y
HTTP	B64-encoded HTTP Header	TCP	Y
ICMP	ICMP Echo Requests	UDP	Y
Slack	Hex-encoded Slack chat message	TCP	Y
TCP	Hex-encoded Raw TCP socket	TCP	Y
Twitter	B64-encoded Direct Message to self	TCP	Y
UDP	Hex-encoded Raw UDP socket	UDP	Y

area network.

Listing 3.4 displays the logging output on our prototype controller for the netcat case. Upon identifying the origin (taint-source) from the UUID, the reference monitor determined the flow to the destination (taint-sink) matched a rule in the policy and subsequently dropped the packets. As listed in Table 3.2, PivotWall successfully detected and dropped packets containing the confidential file.

Stealthy Stepping Stone Attack: We modeled a stealthy stepping stone attack scenario where an external attacker used an intermediate host to pivot and attack an internal confidential host. This experiment also confirmed that traditional access controls and enterprise defenses lack the context of information flow and fail to detect stealthy attacks.

To illustrate traditional enterprise defenses, we enabled an Apache web server and restricted access to the server to only local area network hosts using an Apache access control list. To bypass this access control list, we established a stepping stone (on the local area network) using the *socat* toolkit. This tool forwarded inbound traffic on TCP Port 4444 on the stepping stone to an external request to TCP Port 80 on the webserver. This bypassed the ACL on the webserver, since the stepping stone IP address was within the permitted ACL. However, this violated the intent of our expressed access policy since our external attacker could access information on the webserver. We repeated the experiment with a meterpreter port forwarder and determined the same result. Traditional access controls cannot determine the information flow and fail against stealthy attacks. To illustrate the information flow tracking aspect of PivotWall, we repeated the experiment but marked the Apache process as confidential. We wrote a PivotWall policy to prevent external communication with the protected webserver.

Table 3.4 iPerf bandwidth results

	Mac-Layer Switch	PivotWall	PivotWall (Confidential)
Transferred (MB)	833.6 ±30.5	836.3 ±22.3	445.2 ±62.7
Bandwidth (Mbits/sec)	696.7 ±25.7	699.0 ±19.0	372.9 ±52.6

3.5.2 RQ2: Attack Coverage

To test the coverage range of PivotWall, we examined its ability to detect a broad range of data exfiltration attacks. The extensible Data Exfiltration Toolkit (DET) [Ama16] provides nine different communication channels for covertly exfiltrating data out of a network, bypassing traditional tools. The tool abuses common protocols including DNS, HTTP and ICMP as well as commercial tools such as Gmail, Slack, and Twitter. To test the coverage, we established an attacker's listening post outside the local area network.

We then modeled an insider attack where an attacker used DET to egress data outside the local area network. We repeated the process for the DET covert channel plugins for DNS, Gmail, Google_Docs, HTTP, Slack, TCP, Twitter and UDP. The results were the same for all covert channels: PivotWall identified the access of the confidential data and applied the PivotWall policy preventing the egress of confidential data outside the local area network. The results are summarized in Table 3.3.

3.5.3 RQ3: Network Scalability

We evaluated the network performance of our prototype solution using the iPerf3 toolkit [Dug16]. To realize the total achievable bandwidth of our solution, we measured the total MBytes transferred during a ten second TCP connection and recorded the impact on the congestion window. We repeated this experiment ten different times to determine the average and standard deviation for bandwidth and bytes transferred during the connection. To understand the impact of the flow modifications of our prototype solutions, we compared PivotWall against an SDN application that performed mac-layer matching. To illustrate the impact of confidential labels, we tainted the iPerf3 process on the Server and repeated the experiment. Our results are illustrated in Table 3.4.

As the results illustrate, both unlabeled PivotWall traffic and mac-layer traffic have similar performance. However, the labeled traffic (e.g. the tainted iPerf3 process) is limited to 53% of the achievable bandwidth. In our experiments, the most notable impact was on the TCP congestion window of labeled traffic. As described in Section 3.4.1, labeled traffic is brought to the controller on a per-flow basis. As the reference monitor matches a rule, the controller sends a flow modification for the remainder of the flow. Prior to the establishment of the flow modification, the TCP congestion window does not grow at

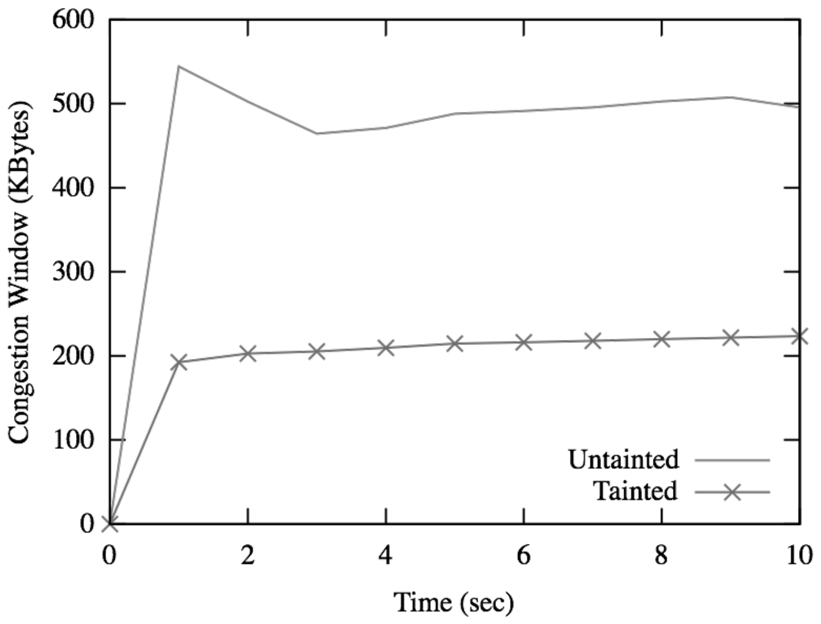


Figure 3.6 PivotWall impact on TCP congestion window

the same rate as mac-layer matched traffic.

Figure 3.6 depicts the average growth of the congestion window for labeled and unlabeled traffic. Labeled traffic suffers for two reasons. Labeled packets pass through a *netfilter* interface (when the label is applied) and are also inspected on a per-flow basis by the reference monitor at the controller. These actions cause the packets to be acknowledged at a slower rate, causing the congestion window to grow at a slower rate compared to packets that bypass the *netfilter* interface and the reference monitor. This host-introduced delay causes the most significant performance impact.

At the SDN layer, PivotWall 's SDN security application performance benefits from the design that only labeled packets are redirected to the controller for inspection. The implementation of PivotWall achieves this by setting the *NW_ToS* flag OFPMatch to to a higher priority over standard flow rules. Further, labeled packets are brought only a per-flow basis. After the first labeled packet is inspected, only packets requiring modification are redirected through the controller. Depending on the demands of packet modification by *Modify-Scripts*, future solutions could examine using NFV to expand the scalability of the PivotWall.

3.5.4 RQ4: Response Capability

We examined the ability of the PivotWall modify-script functionality to respond to a covert attack which uses the DNS protocol as its channel. Detection of covert DNS channels presents an interesting problem, with several proposed machine-learning based approaches for detecting the channel [Kar14; Ich15; Jin15; Sha16; ICH18]. Examples of DNS-based attacker tools include OzymanDNS, dns2tcp, iodine, heyoka,

Listing 3.5 Modify-Script To Log/Redirect DNSCat

```
rewrite_dns_query(packet):
  dns = packet.find('dns')
  ip  = packet.find('ipv4')
  for q in dns.questions:
    log.info("[+] Tainted Query: %s " % q.name)
    log.info("[+] DNS Server: %s " % ip.dstip)
  redir_dst = "10.10.10.10"
  return redir_pkt(packet,redir_dst)
```

element53, DeNiSe, and DNSCat.

We examined how PivotWall prevents an attacker from using DNSCat to covertly embed an attacker channel in valid DNS requests. A significant challenge associated with DNSCat is that it supports forwarding the channel through a legitimate DNS Server to an attacker-controlled authoritative server. Because C2 Tunneling redirects traffic through the victim's default DNS servers, the attacker can use it to bypass egress filtering.

To demonstrate the response capability of PivotWall, we configured DNSCat to exfiltrate data from a victim host to an external domain. In our experiments, DNSCat successfully used DNS Queries to establish a covert channel. However, when the attacker tried to use the established channel to exfiltrate confidential data, the host agent propagated the taint from the file to DNSCat and ultimately to the UDP packets carrying the DNS Requests.

To address this particular attack, we created a PivotWall Modify-Script that rewrote infected queries and logged the domain name used in the attack. Our script, which successfully logged and redirected tainted DNS Queries is listed in Listing 3.5.

3.6 Discussion

Limitations of the Host Agent: We recognize that an attacker can disable the host agent if the attacker fully compromises the host. From the presence of a full compromise, an attacker can un-label confidential data, intercept and modify host-agent control messages, labeled packets, and exhaust the controller resources by a denial of service attack. Attestation of the host agent is necessary but we consider an orthogonal problem for this work. The host agent does not address side channel or covert channels that exfiltrate confidential data outside network connections.

The host agent has difficulty dealing with monolithic applications that do not rely on the operating system to partition information into separate objects. For example, many database engines and web servers themselves implement access controls on information. This is exacerbated by using a single process to manage connections from a number of clients. These factors lead to a semantic gap between monolithic application and the operating system kernel which implements information-flow tracking. The researchers behind SELinux have encountered these same challenges, and they have proposed a series of changes to application software [Mut15; Koh08; Kil03; Mor08]. Finally, the host agent does not address a distributed or cloud environment where processes may move between hosts. We refer to Pappas et. al, who have presented solutions for cloud-based information flow control. [PP12; Pap13]

Blind-Host Assumption Tainting: We do not address hosts that cannot participate in the defense due to resource constraints. Embedded IoT devices are often used in stepping stone attacks because they are programmed with hard-coded credentials in the firmware. While an attacker might be able to construct an SSH pivot through this device, it is highly unlikely that we would be able to modify it to participate in the active defense like a full host. We assume that when a host is unable to participate in the defense, the controller would be able to use *assumption-tainting*. Our initial results offer promising results into *assumption-tainting*. However, we reserve the details for future work.

Integrity Attacks: Not all attacks seek to exfiltrate information. An adversary may seek to, for example, modify a file in a source code repository to insert a back door or similar vulnerability. While the focus of this paper is secrecy attacks, PivotWall provides primitives that are also valuable to defend against integrity attacks. For example, if a Git server is assigned a label, PivotWall network tainting will follow the TCP ACK messages back to to the network flow returning to the attacker on the Internet. However, applying PivotWall in this way may induce significant false positives (e.g., if developers use code snippets from Stack Overflow), and therefore requires further investigation.

3.7 Related Work

PivotWall touches on several areas of prior work. We begin by discussing prior approaches for detecting stepping stone attacks by correlating network traffic. Next we describe how SDN has been leveraged in the past to enhance network configuration and management. Finally, we discuss host-based provenance frameworks.

3.7.1 Detection Using Network Flow Correlation

Previous approaches to detect stepping stone attacks have relied on passive observation [ZP00; Wan02] or actively watermarking packets or flows [WR03; Wan05a; Wan05b] and using the embedded watermark to correlate flows. However, these approaches mostly focus on detecting interactive stepping stone attacks where the attacker maintains a command and control channel and performs the attack steps within a *maximum tolerable delay*. These approaches do not necessarily consider the methods where an attacker evades detection using timing perturbation, introducing random long delays between packets, or adding chaff packets, flow splitting, or repacketization.

Due to concerns that embedded watermarks are observable, RAINBOW [Hou09] proposed an invisible non-blind watermarking technique. RAINBOW records both incoming and outgoing flows and correlates flows using small delays. The technique, however, does not scale to large networks with heavy traffic. Others have proposed techniques that use intervals in different ways to correlate network flows. SWIRL [HB11] changes the locations of packets within selected time slots to encode watermarks. Despite significant effort, there is no robust way of correlating network flows with high accuracy. For an active network flow watermarking technique to work, the watermark must be preserved across stepping stones. However, if the attacker uses store-and-forward or split-relay techniques, the embedded watermark is lost. Store-and-forward attacks were first discussed by Coskun and Memom [CM07]. Along with split-relay based attacks, they necessitate a new approach for stepping stones attack detection. PivotWall aims to address these new challenges while tackling existing ones.

3.7.2 Network Management and Monitoring for SDN

With the increasing popularization of Software-Defined Networks [Fea14; Cas07], many different controller frameworks have been proposed and adopted in practice [Gud08; Eri13; ONL15; Med14; Ber14a]. Kim et al. [KF13] discuss the new possibilities for network management and configuration methods provided by the OpenFlow protocol [McK08; Ope15]. Their studies demonstrate that SDN significantly reduces the complexity of network management in a variety of network settings and for a range of network policies. We leverage the flexibility and the novel properties of SDN to enable security mechanisms that may not be feasible in a traditional network.

There have been a number of proposals that use SDN to enhance network security. Avant-Guard [Shi13b] specifically focuses on addressing the communication bottleneck between the control and data planes by identifying malicious traffic, including network scanning and denial-of-service. Network Iron Curtain [Son14] also focuses on detecting scanning attacks in an SDN environment by implementing a security service that responds to network scanning predefined policies and redirecting attackers to a honeynet. This confuses attackers by providing fake scanning results. Further, Abaid et al. [Aba14] expand the SDN architecture and develop an application to allow deep packet inspection by proposing to elastically partition network traffic on demand using a broad range of detectors. Their research bridges the gap between distributed DPI and SDN-based NIDS. Bates et al. [Bat14] studied the possibility of using SDN for network forensics. They designed an SDN-based forensic system that can be used to investigate previously unobservable attacks such as data exfiltration and collusion between compromised nodes by using SDN to maintain a global view of the network activities.

Apart from leveraging the properties of SDN for building security mechanisms, several recent papers have focused on the security aspects of software defined networks [Shi13a; Por15]. FRESCO [Shi13a] proposes an application development framework for rapidly prototyping security applications for an OpenFlow controller. The authors propose a modular design that allows development through minimal coding. Porras et al. presented SE-Floodlight [Por15], which is a reference implementation of a security-enhanced controller and proposed a role-based hierarchical resolution strategy that could resolves conflicts, along with a Northbound API that could provide authenticated per-message credentials.

In contrast to these prior works, PivotWall enhances the SDN controller with information from host agents. It is also the first solution to use SDN to actively perform network taint analysis to detect stepping stone attacks.

3.7.3 Host Provenance Frameworks

Prior work [Lee13a; Lee13b; Ma16; Bat15] uses audit logging and provenance propagation on hosts to investigate and detect attacks. The central idea of these frameworks is to track the flow of attack provenance from an initial source and so that any future actions can be attributed back to the source.

BEEP [Lee13a] provides efficient, dependence explosion-free logging for binary programs. It partitions a long running process to multiple autonomous units that handle independent input data to perform fine-grained logging. BEEP-generated logs, along with the regular audit logs, enable identifying precise causality between a root cause (i.e. an attack) and its symptoms, avoiding the dependence explosion problem with regular audit logs. In a separate work [Lee13b], the authors of BEEP presented LogGC, which is an audit logging system towards practical computer attack forensics that uses garbage collection to along with partitioning a database file into data units so that dependences can be captured at tuple level. ProTracer [Ma16] is a lightweight provenance tracing system that alternates between system event logging and unit level taint propagation along with event processing. ProTracer leverages a lightweight kernel module and a concurrent userspace daemon. The LPM interface [Bat15] is a framework for the development of provenance-aware systems. It creates trusted provenance-aware execution environments, imposes insignificant performance overhead on normal system operation, and responds to queries about object ancestry in tens of milliseconds.

Most the provenance tracking systems lack the network context while performing provenance propagation. They primarily focus on mitigating host data loss or integrity protection for a single host. Therefore, these systems would have difficulty identifying a stepping stone attack chain. PivotWall complements these prior approaches in that its host agent could be enhanced by incorporating their host-based provenance frameworks.

Pedigree [Ram09; Ram08] is closest to PivotWall in concept. It extends packets with a taint tag derived from host-level information and uses an OpenFlow-based arbiter to make security decisions. To mitigate taint explosion, Pedigree probabalistically removes taint bits, which sacrifices security for usability. PivotWall's NIFC graphs provides an alternate design approach that puts declassification of tainted network packets more directly into the hands of network administrators.

3.8 Conclusion

This paper presented PivotWall, a new network security architecture to implement information flow control in a distributed environment. Our solution combines the logically central placement of the SDN Controller with the context of host-based information flow tracking. We implemented a prototype by implementing a host agent and network controller application. We evaluated our prototype by testing it against a broad coverage of stealthy attack tools, examined the performance impacts, and demonstrated the ability to provide unique response capabilities to real-world attacks. We demonstrate that PivotWall is a feasible approach to enable context-based response and prevention capabilities that would be difficult to realize with existing solutions. With the ability to detect stealthy attacks and implement reactive measures with acceptable impacts to performance, PivotWall is a promising solution for enhancing the security of enterprise networks.

3.9 Acknowledgements

This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS) and NSF grant CNS-1330553. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

CHAPTER

4 HOMESNITCH

4.1 Introduction

The pervasiveness of the Internet-Of-Things (IoT) devices is estimated to reach 50 billion by 2020 [Sch12]. A substantial population of these IoT devices is emerging in the residential environment. These smart home devices offer convenience by monitoring wireless sensors such as temperature, carbon monoxide, and security cameras. Further, these devices enable automating actions including sounding alarms, making coffee, or controlling lighting. As the functionality of these devices mature, they are increasingly interacting autonomously, invisible to the end-user. Turning off a connected alarm can signal a coffee pot to begin brewing and a motion-sensor can automate turning on a light. However, risk is introduced with each newly connected device due to their often-insecure software.

The smart home market is incredibly broad and heterogeneous, lacking sufficient standards and protocols to enforce security and privacy. Yet, these same devices generate, process, and exchange vast amounts of privacy-sensitive and safety-critical information in our homes [Sad15]. Manufacturers can surreptitiously gather data from the end user in violation of their expressed policy desires. As an example, Germany banned the wireless *Cayla* doll, which contained a microphone and uploaded recorded conversations with children [Nay16; Nel17]. Further, an attacker may use a smart home IoT device to gain unauthorized access to other devices in the home. The Central Intelligence Agency's Weeping Angel implant on connected Samsung Televisions targeted home WiFi networks, accessed user-names and passwords, and covertly recorded audio [FB17].

Smart home security is an emerging topic without a clearly defined threat model. A compromised or misbehaving smart home device may attack other network hosts, but it may also eavesdrop on the physical home environment. It is often not necessary to stop the first, or even second, instance of covert audio recording. Rather, it is important to identify misbehaving devices and take some action (e.g., access control, or simply turning off the device). For example, in the case of the Cayla doll, there is limited harm in eavesdropping on a few of a child's conversations, but risk increases the longer the eavesdropping occurs. Similarly, when a device suddenly changes its behavior, as in the case of Weeping Angel recording audio, the owner's goal is to prevent prolonged exposure.

The software on IoT devices is unlikely to improve significantly in the near future. Therefore, consumers would greatly benefit from a network-based solution. However, contemporary tools provide limited help in interpreting communications exchanges taking place in the smart home network, as they require in-depth understanding about network protocols and the specific network set-up. Meaningful monitoring and access control are challenging for experts, let alone for regular smart home users. The task is further complicated by the fact that IoT devices often use encryption, meaning solutions cannot rely on the contents of network traffic.

In this paper, we present HomeSnitch as a building block for improving the transparency and control over the network communications of smart home devices. The key insight of HomeSnitch is to classify and control *semantic behaviors* using features that represent application-layer dialogue between clients and servers. We implement HomeSnitch on a commodity wireless router and build upon Software Defined Networking (SDN) primitives to control network traffic based on end-user preferences. By operating on application behaviors, rather than IP addresses and packet flows, HomeSnitch facilitates enhanced user agents to provide transparency and control over smart home devices. We evaluate HomeSnitch in two ways. First, we evaluate its classification accuracy using a dataset of labeled packet traces from the YourThings [Alr19] project. For this dataset, our analysis demonstrates that HomeSnitch achieves an accuracy of 99.69% in classifying behaviors. Second, we deployed HomeSnitch in a real home environment with 20 devices and evaluated its ability to correctly classify known behaviors, as well as to help discover new behaviors. Such capabilities demonstrate the practical utility and importance of providing end-users with greater transparency and control of smart home devices.

We make the following contributions in this paper.

- *We present the* HomeSnitch *framework for identifying distinct semantic behaviors by smart home devices.* HomeSnitch provides a building block for transparency and control of smart home IoT device network communications by reporting activity on the granularity of semantic device behaviors instead of low-level concepts such as distinct packet flows.

- *We propose a supervised learning technique to classify semantic device behaviors using network traffic.* The technique works on both encrypted traffic and proprietary application-layer protocols and achieves greater than 99% accuracy on an independent data-set. More importantly, HomeSnitch can identify novel behaviors with resistance to under-fitting.

- *We perform an empirical evaluation of* HomeSnitch *in a real home environment.* We demonstrate how HomeSnitch can help identify previously unseen devices and behaviors. We demonstrate our behavior fingerprinting approach enables policy and access control across fine-grained behaviors.

The remainder of this paper proceeds as follows. Section 4.2 provides motivation and articulates the problem. Section 4.3 overviews the HomeSnitch architecture. Section 4.4 describes its design. Section 4.5 describes the implementation. Section 4.6 evaluates accuracy and performance. Section 4.7 presents an empirical study of HomeSnitch in a home area network. Section 4.8 discusses limitations. Section 4.9 overviews related work. Section 4.10 concludes.

4.2 Motivation and Problem

IoT smart home devices present significant security and privacy challenges for consumers. Many smart home devices have poor authentication mechanisms, un-patchable software vulnerabilities, and limited access control configurability. Even when devices are not being exploited, they may present privacy threats due to unexpected behavior implemented by device manufacturers. Providing transparency into device behavior is challenging, as smart home devices frequently use proprietary application-layer protocols. Finally, efforts to secure smart home devices (e.g., HTTPS encryption) further complicate network layer access controls.

Motivating Example (Alexa API Abuse): Checkmarx security researchers recently identified a method for abusing the implicit policies of the Amazon Alexa digital assistant [New18]. In order to covertly record audio from Alexa enabled devices, the researchers created a voice activated application, known as an Alexa *skill*. The application posed as a simple application for solving math problems but instead recorded audio indefinitely. The researchers abused the API by proving a null value to a prompt, silencing the device and deceiving the user into believing it is no longer listening. This abuse of trust motivates our work and presents a challenging problem for defense. In terms of the smart home IoT device, it must assume the server-side platform is trusted after mutual authentication and execute commands. HomeSnitch addresses this by pushing the monitoring and policy enforcement into the network using SDN and a machine learning classification of application behaviors.

Problem Statement: The goal of HomeSnitch is to provide the primitives for transparency and control of behaviors of otherwise resource constrained smart home IoT devices. To achieve this, HomeSnitch seeks to operate on application *behaviors* such as heartbeat, firmware check, reporting, and uploading audio or video recordings. Abstracting network flows and packet data into classifications offers better intuition about the device activity. Thus, HomeSnitch seeks to enable transparency by reporting *which* device performs *what* behaviors, *when* and *how often*, rather than the simple existence and frequency of network traffic. Furthermore, HomeSnitch seeks to enable enhanced systems that enforce network access control policies based on observations of previous undesirable behaviors, thereby allowing end users to use valuable features of smart home devices that otherwise exhibit privacy invasive behaviors.

Threat Model and Assumptions: HomeSnitch is designed for a residential smart home network setup consisting of sensors, actuators, and smart objects, both wired and wirelessly connected to a consumer-grade router. We assume smart home devices contain default credentials, lack sufficient security protocols, enable over-privilege, and contain un-patched vulnerabilities. We assume an adversary can compromise smart home devices either by directly connecting to them either through *NAT hole punching* [Siv16], lateral pivoting from other compromised devices, gaining physical control of the device [Dha15], or abusing cloud-based control. HomeSnitch protects against both malicious adversaries and benign behavior by manufacturers that violate the explicit desires of the end-user.

The adversary's goal is to exploit smart home devices to cause physical effects (e.g., turn off an alarm), ex-filtrate data (e.g., enable remote video surveillance), steal credentials, or compromise other devices in the network. The attacker may launch an attack through malware residing on the device, a remote network attack, or abuse server-side privileges on the specific service-provider for the device. Further, we assume smart home devices lack timely patching and updates to correct vulnerabilities. Defending attacks without the cooperation of the device is a sufficiently hard problem since traditional security approaches involve scanning for vulnerabilities, enabling fine-grained permissions on the host, and patching vulnerabilities. A smart home network of devices cannot employ any of these approaches in the defense. HomeSnitch prevents against attacks via IP connection-oriented protocols. We consider devices that communicate via other protocols (e.g. ZigBee, Bluetooth, NFC) as important and orthogonal problems for future work. As discussed in Section 4.8, our work does not address the case of mimicry attacks.

4.3 Overview

As motivated in the previous section, HomeSnitch seeks to enable practical transparency and control of the network communication of IoT smart home devices. Figure 4.1 depicts our vision of how HomeSnitch would operate in practice. The HomeSnitch software is originally configured with a set of behavior models and default network access control policies. Over time, these models and policies are updated via a feed from a security service provider (e.g., BitDefender [Bit18] or Norton [Nor18]). When IoT smart home devices communicate with servers on the Internet, or with one another on the local LAN, HomeSnitch classifies the network flows as known behaviors for devices. Unknown network flows are matched with a set of the closest device behaviors and optionally reported to the security service provider to enhance training models. End users gain transparency through the HomeSnitch control interface (e.g., a web page or mobile app). This interface reports the network activity of IoT smart home devices based on

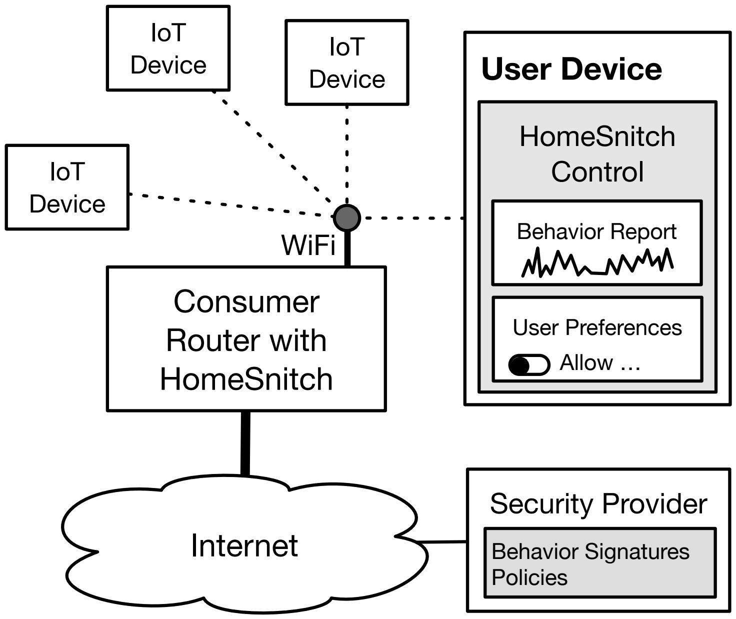


Figure 4.1 HomeSnitch provides a building block for transparency and control of IoT device behaviors.

their high-level behaviors. The interface also allows the end user to specify preferences of what behaviors they would like to allow (e.g., similar to how smartphone platforms enable greater privacy control).

This paper addresses two key research challenges.

- *Behavior Classification:* There is a large semantic gap between the bytes sent in packets and the behaviors that are understandable to users. Classification is further inhibited by encryption and proprietary application-layer protocols.

- *Network Mediation:* Network traffic must be mediated both between devices and the Internet, as well as between devices within the home network. Smart home devices are designed for simple residential networks and functionality often fails when enhanced technologies such as VLANs are introduced. Therefore, the LAN must retain a flat address space.

HomeSnitch overcomes these research challenges through the use of SDN and supervised machine learning. More specifically, it extends the firmware of an operating system for commodity routers (OpenWrt) with OpenFlow capabilities. By using OpenFlow, HomeSnitch is able to mediate all network communication, both between devices and between devices and the internet, without changing the IP address space structure. Using OpenFlow also allows the design to easily scale to environments with multiple access points, which are becoming more common as newer WiFi technologies such as IEEE 802.11ac operate faster over shorter distances. HomeSnitch then uses OpenFlow to send network flow information to an SDN controller application. In our prototype, the SDN controller application runs on

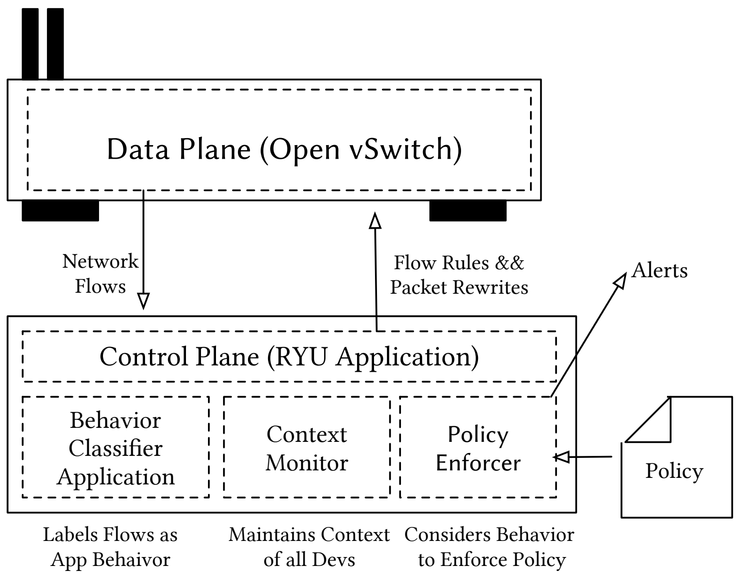

Data Plane (Open vSwitch)
Network Flows
Flow Rules && Packet Rewrites
Alerts
Control Plane (RYU Application)
Behavior Classifier Application
Context Monitor
Policy Enforcer
Policy
Labels Flows as App Behaivor
Maintains Context of all Devs
Considers Behavior to Enforce Policy

Figure 4.2 HomeSnitch leverages a fine-grained behavior classification approach to provide context for effective policy enforcement and access control.

a separate Raspberry Pi controller board connected via Ethernet; however, as consumer devices with greater processing capabilities (e.g., the Turris Omnia [Tur19]) become available, all of the HomeSnitch functionality can be moved to a single device.

The SDN controller application performs behavior classification based on network flow information. Specifically, HomeSnitch extracts application data unit (ADU) exchanges to acquire an abstract representation of the client and server communication. ADU features are then used by a supervised machine learning algorithm to classify the flow as a known behavior triple: (<vendor>, <model>, <activity>), e.g., (Ring, Doorbell, Motion-Upload). Note that human readable names such as "Motion-Upload" must be assigned to a learned behavior. When a network flow has a poor match to any known behavior triple (as defined by a configurable threshold), HomeSnitch provides a set of closest matches. These matches can provide valuable insight into the correct classification (e.g., all closest matches are for the same vendor or the same activity). In practice, we imagine a security service provider that maintains and distributes a trained model of these triples.

4.4 Design

This section describes the design of HomeSnitch, as depicted in Figure 4.2. Our design currently separates the classification logic in the controller application onto an external embedded device; however, as routers continue to become more computationally powerful, it is reasonable to expect the controller to be moved onto the router. We begin the discussion of the HomeSnitch design by describing the behavior classification from network flow information. This classification system represents the core of our technical contribution. We then proceed to describe the context manager, the policy language, and policy enforcement using SDN primitives.

Note that for simplicity, we assume all devices on the network are IoT devices. However, in practice networks will contain desktops, laptops, and smartphones. We assume the existence of techniques to determine if a device is an IoT device, which can be accomplished by looking at the OUI or using more advanced classification systems such as IoT Sentinal [Mie17].

4.4.1 Classification of Application Behaviors

We treat smart home device behavior identification as a multi-class classification problem. More specifically, we seek to map a network flow to the specific application behavior and device that produced it. Examples of device behavior include downloading firmware, receiving a configuration change, and sending video to a remote user. Deep packet inspection, protocol analysis, and certificate inspection provide limited insight into behaviors, as network flows are often encrypted to a cloud-based server such as Google Cloud, Amazon Web Services, or Microsoft Azure.

HomeSnitch defines an application behavior as a triple. We originally sought to define a behavior simply by an activity that transcends vendors and devices. However, in practice, we found vendor and model implementations defined the structure of application data exchanges. This even varies across specific product offerings as we discovered the original Ring Doorbell had a much more simplistic set of behaviors compared against the recent Ring Pro model. Therefore, we decided to tie the activity to a specific vendor and device. We found that this has the benefit of helping identify unknown behaviors. For example, a new behavior may have a relatively close match for another device by the same vendor. The remainder of this section describes our feature selection, classification model, and operational considerations.

Application Data Unit (ADU) Extraction: To overcome the limitation of analyzing encrypted traffic, HomeSnitch constructs application data unit (ADU) exchanges that provide an abstract representation model of the exchange between a client and a server. Several approaches exist to build application-level models from packet headers in an offline manner [HC07; Ols04; ON06]. HomeSnitch uses *adudump* [Ter09], which has the capability for continuous measurement of application-level data from network flows. Additionally, *adudump* only uses TCP/IP packet header traces (i.e., no application layer headers or payloads) to construct a structural model of the application dialog between each server and the client. This overcomes the limitations of increasingly encrypted payloads in smart home devices. As an added benefit, this header information is readily available to our SDN controller applications. In our analysis of several different devices and behaviors, we found that the application data units can be used to accurately distinguish between behaviors.

Feature Selection: HomeSnitch uses supervised machine learning to classify network flows into applica-

Table 4.1 HomeSnitch features describe the application data exchanges. This approach overcomes the limitations of solutions that rely on physical, network, or transport layer headers unique to smart home deployments.

Feature	Category	Importance
average bytes from client per sequence	throughput	0.213104
average bytes from server per sequence	throughput	0.072519
aggregate sever bytes sent over ADU	throughput	0.105775
aggregate client bytes sent over ADU	throughput	0.117552
min bytes from client from a sequence	burstiness	0.038917
min bytes from server from a sequence	burstiness	0.038344
max bytes from server from a sequence	burstiness	0.079063
max bytes from client from a sequence	burstiness	0.135909
stdev of bytes between server sequences	burstiness	0.054491
stdev of bytes between client sequences	burstiness	0.050798
server sequences per ADU	synchronicity	0.013566
client sequences per ADU	synchronicity	0.016211
total time of connection	duration	0.063750

tion behaviors. We selected thirteen features from transport layer headers that describe the client/server dialogues of smart home IoT devices (listed in Table 4.1). Each feature is a descriptive statistic calculated from a single application data unit. ADUs are bidirectional representations of sequential exchanges within a single connection, terminated upon proper connection tear-down or by flow timeout [Ter09]. A key insight that benefits HomeSnitch is that smart home IoT traffic is predominately sequential with the server and client taking turns. This property is due to the limited complexity of devices with most processing done server side. The lack of traffic concurrency allows HomeSnitch to observe the behavior of the application by extracting statistics from each sequential turn.

Our model excludes IP addresses, port numbers, and DNS information as they present several limitations and are increasingly less reliable features [Wil06; Kim08; Dai12]. Instead, we focus on content agnostic features that solely describe the behavior of a smart home application. We recognize a real-world appliance may benefit from additional information (i.e., destination address and port). However, our threat model considers an attacker that is focused on exhibiting a behavior and could use flow information to mask and hide a behavior. We conducted several experiments with real devices to identify features that offer better discriminative capabilities of smart home applications. Specifically, we observed the importance of features describing the burstiness, synchronicity, and throughput of application dialogues. Table 4.1 lists the set of these features, consisting of average, aggregate, minimum and maximum measurements for these dialogues. For purposes of definition, *sequence* indicates a single directional application exchange and *ADU* indicates the entire application data unit dialogue, bounded by the creation and termination of the entire network flow. The notion of burstiness describes the proximity of arrival instances within each other and the variance between each arrival [AF04]. To measure this within an ADU, we examine the variance in terms of both payload size and inter-arrival times. To measure synchronicity, we observed measurements that describe how a client and server take turns sending data within the context of an ADU. Throughput is a routinely used but important measure of the aggregate data sent over the duration of the ADU.

As part of our experiments in Section 4.6, we calculated the feature importance (a measurement of the predictive importance for each feature variable in the random forest) with respect to the YourThings [Alr19] corpus of data (also shown in Table 4.1). Average bytes, max bytes, and aggregate bytes from the IoT device were the most dominant features and accounted for 46.65% of the decision estimate used by our classifier. Since smart home devices are typically designed for specific purposes and communicate in a limited fashion, these features best describe the throughput and burstiness unique to application data units. This insight matches the analysis of the YourThings data-set, where 64.72% of device flows lasted for less than a second. Overall, we carefully selected a balanced set of features to support the multi-class classification problem associated with device behaviors. We discovered burstiness features describe the on-demand actions of motion sensors. In contrast, throughput features better describe data-intensive sensors. Synchronicity and duration offer strong features for data logging and analytic reports.

Classification Model Selection: We considered multiple algorithms (described in Section 4.6) but found Random Forest to be the best for intuitive reasons. Random Forest leverages both ensemble and averaging methods to build several estimators independently and then average their predictions [Ped11]. By leveraging a bagging-based ensemble model, HomeSnitch is resistant to under-fitting with a reduced variance. Our results in Section 4.6.1 demonstrate that Random Forest offers a very accurate model that precisely classifies smart home device behaviors and detects new behaviors.

In Section 4.6, we show that both Random Forest Classifiers (RFC), Gradient Boost, and K-Nearest Neighbor (kNN) algorithms score well in accuracy using k-fold cross validation of a labeled data-set. However, HomeSnitch needs to inform the user when previously unseen behaviors occur. This design constraint further motivates an ensemble or averaging based classifier, as they are more resistant to under-fitting. To detect unknown behaviors, HomeSnitch uses the probability prediction score as a measure to define a threshold of acceptance. Data points below the threshold are predicted as unknown behaviors and a new classification label is created.

We define *Unknown Behavior Miss Rate* (UBMR) as the percentage of data-points in a given data-set that fail to be identified as a new classification. True Positive Rate (TPR) is the total number of true positives divided by the sum of the true positives plus false positives. Ideally, the classification should minimize the UBMR without adverse effect to the TPR. As discussed and empirically evaluated in Section 4.6, we found a threshold of 0.89% as the best balance between UBMR and TPR. The evaluation further demonstrates the utility of Random Forest over kNN.

Training Data: Supervised machine learning approaches require initial training data to determine classes for unclassified data-points. In this paper, we consider two approaches for constructing the initial training data. For the accuracy evaluation in Section 4.6.1, we construct initial labels from a corpus of timestamped behaviors. However, this approach does not scale well. For the empirical study in Section 4.7, we also use clustering algorithms to identify behavior classes in spatial data-sets. Specifically, we use the density sampling algorithm, DBSCAN, as it performs well with small sample sizes and ambiguous sized clusters [Est96; Ped11]. For both methods, we use cross validation scoring to improve the accuracy of training data.

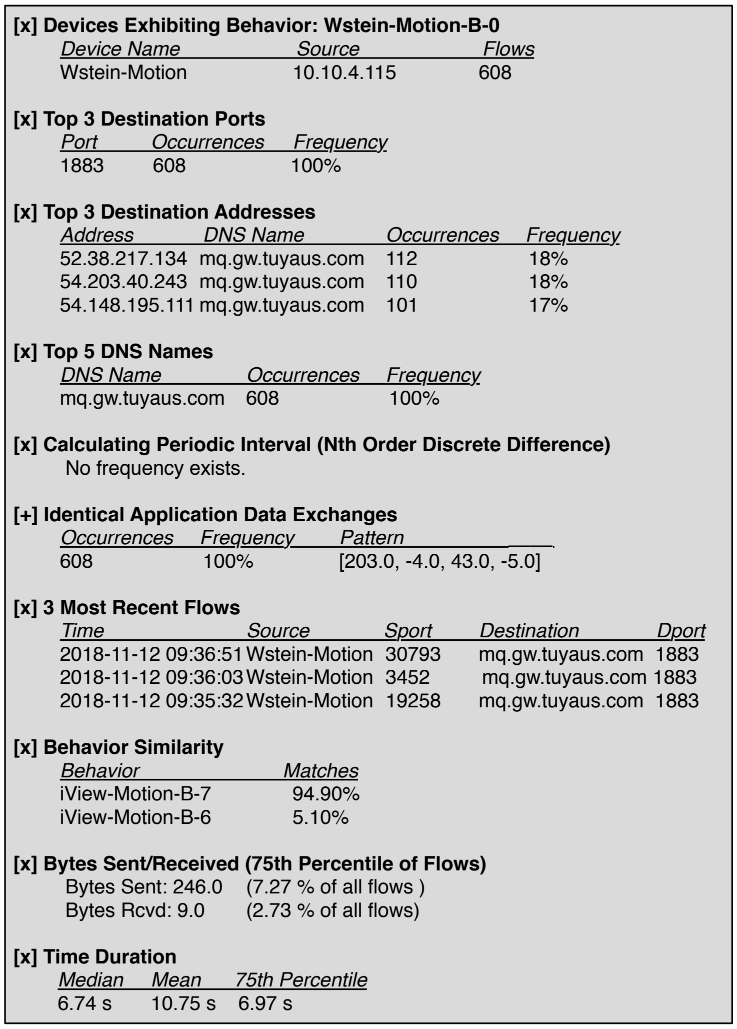
Security Service Provider: In practice, we found that the trained model is often specific to devices by the same manufacturer. To ensure HomeSnitch can detect behaviors for new devices, we propose the use of a security service provider (SSP) to share features and expand the set of supported smart home device behaviors. This approach allows the development of a model that expands as the heterogeneity of devices increases. Both IoT-Sentinel [Mie17] and IoTurva [Haf17] show promise as techniques for an SSP to collect normal and anomalous device activities from consumer deployments. HomeSnitch could use a publish-subscribe interface similar to virus reporting. For example, BitDefender [Bit18] and Norton [Nor18] offer subscription-based services for defending smart home devices.

Approaches that send information out of the home network raise concerns of privacy and data quality. To ensure the privacy of users, HomeSnitch can limit data collection to the features and labels collected in a smart home IoT deployment. When doing so, the labels associated with time-stamps must be appropriately protected (e.g., excluding day, hour, and minute information). Labels must also be associated with devices. Fortunately, the higher order bits of MAC addresses indicate the Organizationally unique identifier (OUI). End users can also be consulted to help identify devices. Finally, such crowd sourcing approaches must address data quality, e.g., via user reputation, voting, or threshold-based acceptance. We leave such considerations for future work.

Behavior Reporting: HomeSnitch offers novel behavior detection and descriptive reporting. Reporting provides the capability to understand the function of newly detected behaviors. Figure 4.3 displays a behavior report that predicts closest matching behaviors while offers behavior-centric details (i.e., frequency, duration, and throughput). This approach allows a user to generalize and discern the purpose of a previously unseen behavior. Figure 4.3 demonstrates our behavior similarity approach by identifying that the behaviors from the iView Motion and Wasserstein Motion devices have a high degree of similarity. In fact, these two devices implement nearly identical lightweight telemetry protocols to Tuyaus cloud-based servers.

4.4.2 Context Monitor

Smart home devices are designed to connect and exchange data with one another. Devices connect directly through explicit communication channels or implicitly through back-end platforms. As illustrated by the Alexa API abuse in Section 4.2, users cannot always trust third party platforms to protect security and privacy. HomeSnitch's context monitor maintains the context and behaviors of all peer smart home devices on the network. It archives historical flow data (source, destination, transport layer ports, time-



```
[x] Devices Exhibiting Behavior: Wstein-Motion-B-0
      Device Name                  Source               Flows
      Wstein-Motion                10.10.4.115          608

[x] Top 3 Destination Ports
      Port       Occurrences    Frequency
      1883       608            100%

[x] Top 3 Destination Addresses
      Address          DNS Name             Occurrences    Frequency
      52.38.217.134  mq.gw.tuyaus.com   112                18%
      54.203.40.243  mq.gw.tuyaus.com   110                18%
      54.148.195.111 mq.gw.tuyaus.com   101                17%

[x] Top 5 DNS Names
      DNS Name              Occurrences    Frequency
      mq.gw.tuyaus.com   608               100%

[x] Calculating Periodic Interval (Nth Order Discrete Difference)
       No frequency exists.

[+] Identical Application Data Exchanges
      Occurrences    Frequency       Pattern
      608               100%            [203.0, -4.0, 43.0, -5.0]

[x] 3 Most Recent Flows
      Time                         Source             Sport        Destination            Dport
      2018-11-12 09:36:51 Wstein-Motion  30793      mq.gw.tuyaus.com  1883
      2018-11-12 09:36:03 Wstein-Motion  3452        mq.gw.tuyaus.com 1883
      2018-11-12 09:35:32 Wstein-Motion  19258      mq.gw.tuyaus.com  1883

[x] Behavior Similarity
      Behavior                       Matches
      iView-Motion-B-7             94.90%
      iView-Motion-B-6             5.10%

[x] Bytes Sent/Received (75th Percentile of Flows)
       Bytes Sent: 246.0    (7.27 % of all flows )
       Bytes Rcvd: 9.0       (2.73 % of all flows)

[x] Time Duration
      Median    Mean      75th Percentile
      6.74 s      10.75 s   6.97 s
```

Figure 4.3 HomeSnitch provides descriptive reports of behavior characteristics and identifies similar behaviors.

stamp, flow duration, and classification labels) in a database at the controller. Our context manager prototype currently provides the time period, the state of being encrypted or not, and the existence of a management device on the network at the time of the flow. In the future, we envision correlating flows to servers to identify the implicit communication occurring between devices or contrarily the lack of such communication. The context is used in three ways. First, it is provided in an activity report to the user in the HomeSnitch control interface. Second, it can be used for access control policies (see Section 4.4.3). Third, we envision a security service provider can collect aggregated historical context to help detect malicious device behaviors. We also envision a system that can identify when less secure devices are leveraged to attack more restricted devices.

$$\langle\text{rule}\rangle ::= \langle\text{action}\rangle\ \langle\text{device}\rangle\ \langle\text{behavior}\rangle\ \langle\text{context}\rangle \tag{4.1}$$
$$\langle\text{action}\rangle ::= \langle\text{alert}\rangle \mid \langle\text{log}\rangle \mid \langle\text{pass}\rangle \mid \langle\text{drop}\rangle \mid \langle\text{reject}\rangle \mid \langle\text{redirect}\rangle \tag{4.2}$$
$$\langle\text{device}\rangle ::= \langle\text{vendor}\rangle\text{“}-\text{”}\langle\text{model}\rangle \tag{4.3}$$
$$\langle\text{vendor}\rangle ::= \langle\text{const}\rangle \tag{4.4}$$
$$\langle\text{model}\rangle ::= \langle\text{const}\rangle \tag{4.5}$$
$$\langle\text{behavior}\rangle ::= \langle\text{vendor}\rangle\ \langle\text{model}\rangle\ \langle\text{activity}\rangle \tag{4.6}$$
$$\langle\text{activity}\rangle ::= \langle\text{const}\rangle \tag{4.7}$$
$$\langle\text{context}\rangle ::= \langle\text{time\_ctx}\rangle \mid \langle\text{enc\_ctx}\rangle \mid \langle\text{mgt\_ctx}\rangle \tag{4.8}$$
$$\langle\text{time\_ctx}\rangle ::= \text{“}\texttt{(T:}\text{”}\ \langle\text{time}\rangle\ \text{“}-\text{”}\ \langle\text{time}\rangle\ \text{“}\texttt{)}\text{”} \tag{4.9}$$
$$\langle\text{enc\_ctx}\rangle ::= \text{“}\texttt{(C:}\text{”}\ \langle\text{encrypted}\rangle \mid \langle\text{unencrypted}\rangle\ \text{“}\texttt{)}\text{”} \tag{4.10}$$
$$\langle\text{mgt\_ctx}\rangle ::= \text{“}\texttt{(M:}\text{”}\ \langle\text{present}\rangle \mid \langle\text{notpresent}\rangle\ \text{“}\texttt{)}\text{”} \tag{4.11}$$
$$\langle\text{time}\rangle ::= \texttt{[0-9] [0-9]}\ \text{“}\texttt{:}\text{”}\ \texttt{[0-9] [0-9]} \tag{4.12}$$
$$\langle\text{const}\rangle ::= \text{“}\prime\text{”}\texttt{[A-Za-z0-9\_.]+}\text{“}\prime\text{”} \tag{4.13}$$

Figure 4.4 HomeSnitch syntax in BNF

4.4.3 Policy

The labels derived in Section 4.4.1 can be used to provide transparency of network traffic as well as access control. In this section, we describe our access control policy language. The typical user will never interface directly with our policy language. However, the expressiveness of the policy language corresponds to the high-level capabilities and may be built upon by enhanced agents.

HomeSnitch Syntax: HomeSnitch enforces policy rules that prevent unwanted device behaviors. HomeSnitch uses a *Device, Behavior, Context → Action* model to express access control policies. The $Device$ predicate is a general description of the device (e.g., Ring-DoorbellCam, WeMo-MotionSensor, Nest-Alarm) that maps to a set of link layer MAC addresses for devices. Next, the $Behavior$ predicate is a behavior triple, as defined in Section 4.4.1: (<vendor>, <model>, <activity>). For example, the triple (WeMo, Motion-Sensor, Upload-Motion) describes the behavior of a WeMo Motion Sensor uploading motion sensor data. Note that the actual string names need to be defined by either the security service provider or the end user. Finally, the $Context$ predicate allows the user to describe complex scenarios based on information in the Context Manager (Section 4.4.2). Our prototype supports connection encryption, time period, and co-located management devices. When the criteria specified in the *Device*, *Behavior*, and *Context* are met, the rule implements the specified $Action$ to enforce fine-grained access control to *alert, log, pass, drop, reject, or redirect*. Figure 4.1 lists the context free grammar for the HomeSnitch policy language.

Policy Examples: Listing 4.1 shows three example policy rules. These examples demonstrate the flexibility to implement access controls for behaviors and context. The first rule prevents disclosing

Listing 4.1 Examples of HomeSnitch's configurable policy.

```
drop Ring-Doorbell, [Ring,Doorbell,Motion-Upload], (T: 0100-0300)
log WeMo-BabyCamera, [WeMo,BabyCamera,Firwmare-Update], (C: unencrypted)
reject LockState-Lock, [Lockstate,Lock,Unlock], (M: notpresent)
```

motion-upload data from a wireless doorbell during the early morning hours of 0100-0300. The second rule logs firmware updates over unencrypted connections. The third rule prevents a smart-lock from unlocking unless the management device (e.g., a smartphone) is attached to the smart home wireless network. Analogous to the popular If-This-Then-That model, we envision a community of shared recipes to address the array of smart home IoT threats [Ur16].

Policy Enforcement: HomeSnitch generates OpenFlow FlowMods representing policy rules to enforce the fine-grained access control on smart home devices and behaviors. To accomplish this, HomeSnitch maintains a historical database of previously seen behaviors and flow-level information (i.e source address, source port, destination address, destination port). To enforce the policy primitive actions, HomeSnitch queries the database for the flow-level information (i.e., a single or set of ports and/or addresses) that can be used to describe the historical flows without conflicting with other behaviors. HomeSnitch uses this query to create flow modifications. Simple actions including *pass* or *drop* can be accomplished with a single flow modification. However, more complex actions (e.g., *reject* or *redirect*) are brought to the controller with distinct action labels (by overwriting the IP ToS as demonstrated in Chapter 3). For the reject action, the controller forges a TCP reset packet. For the redirect action, the controller develops flow modifications using the *OFPActionSetField* primitive to re-write the source and destination MAC and IP Addresses for both the device and destination.

4.5 Implementation

We built HomeSnitch on top of a commodity wireless access point running an open source firmware that supports a virtual switch and the OpenFlow SDN protocol. This section details the key the components of HomeSnitch, including the data and control plane.

Data Plane: We implemented a prototype on a Linksys router, running our custom OpenWrt firmware. We integrated Open vSwitch Support (OVS) with OpenWrt by compiling OVS as a kernel module. Our prototype, running OVS 2.39, supports up to the OpenFlow 1.51 protocol specification. The router is configured to use simultaneous dual band with support for 802.11n (2.4 GHz) and 802.11ac (5GHz). We created a single OVS bridge to bind both WiFi interfaces and all local Ethernet interfaces, allowing OpenFlow to manage wireless connected devices as well as devices connected via Ethernet.

Control Plane: We implemented our SDN Security Orchestrator on a Raspberry Pi 3 Model B single

Table 4.2 HomeSnitch detects known behaviors using a bagging-based ensemble algorithm, and avoids over-fitting.

Model	**Accuracy**	**Recall**	**F1 Score**
k-Nearest Neighbor	99.32 %± .12	87.97 %± 2.16	86.35 %± 2.56
Gradient Boost	64.70 %± 42.10	53.55 %± 33.51	51.72 %± 32.72
Random Forest	**99.69 %± .06**	**94.66 %± 1.21**	**93.93 %± 1.40**

board computer, connected via Ethernet to our router. The Raspberry Pi has a quad-core 64-bit ARM Cortex A53 clocked at 1.2 GHz. We flashed the Ubuntu 16.04 Operating System onto the Raspberry Pi and installed the necessary packages to support the Ryu component-based software defined networking framework. We developed our controller application as a Ryu management application in 6,503 lines of Python. Our Ryu application implements the logic described in Section 4.4 that identifies labeled smart home IoT device behaviors and implements the expressed policies of the end-user. We construct our machine learning models using the *RandomForestClassifier* and *DBSCAN* modules from the *scikit-learn* open source machine learning library [Ped11].

4.6 Evaluation

HomeSnitch identifies device behavior by analyzing communication flows. It trains supervised machine learning models of device/server dialogues, to classify network flows into behavior classifications, regardless of possible encryption and proprietary protocols. In this section, we empirically evaluate the performance of our prototype by answering the following research questions.

RQ1 (*Accuracy*): Is HomeSnitch effective in identifying and classifying smart home IoT device behaviors?

RQ2 (*Performance*): What are the performance impacts on throughput and jitter?

RQ3 (*Required Training*): How many samples are required to train HomeSnitch to classify different behaviors?

4.6.1 RQ1: Behavior Classification Evaluation

Data-set and Experimental Setup: We evaluated the accuracy of our behavior classification model against an independent corpus of IoT device behaviors. The YourThings [Alr19] data-set included network captures from 46 devices, clustered into 148 semantic behaviors, connected to over 2,239 cloud endpoints between 13-19 April 2018. The data-set is representative of the broad smart home IoT market.

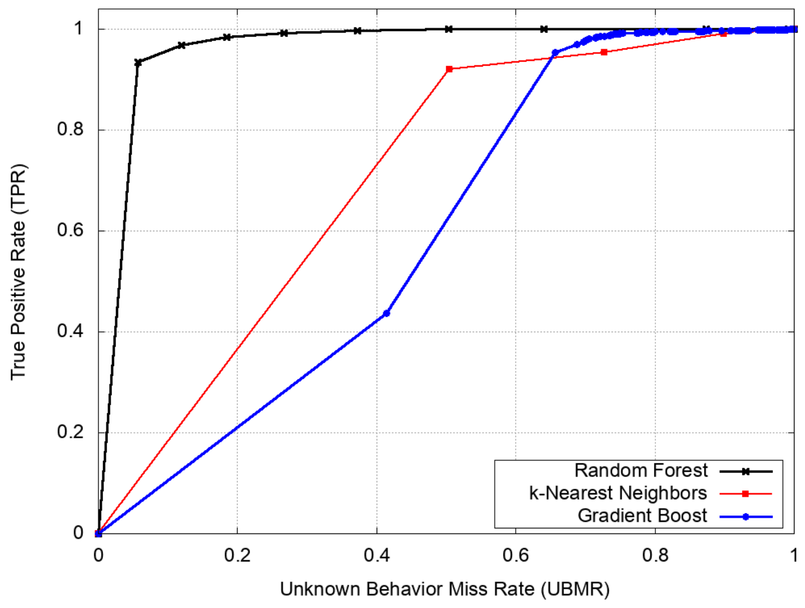


Figure 4.5 Receiver operating characteristics curve for comparison of different supervised learning algorithms abilities' to classify behaviors while detecting new behaviors.

Accuracy: To evaluate accuracy, we compared the performance of Random Forest (our chosen model) against kNN (used in Peek-A-Boo [Aca18]), and Gradient Boost (used in IoTSense [Bez18]) against the data-set. We determined the accuracy through a stratified 10-fold cross-validation procedure against the labeled data-set [Ped11]. Further, we calculated the recall and F1 for all models. Table 4.2 shows the results of our evaluation. As previously described in Section 4.4.1, our approach relies on a Random Forest classification model. Ensemble-based algorithms, including Random Forests and Gradient Boost, typical perform well for numerical features without scaling and perform implicit feature selection. Random Forest yielded the highest score using cross validation scoring. K-Nearest Neighbor (kNN), known for high accuracy and no *a priori* assumption, classified the labeled data-set with high accuracy and moderate recall.

Unknown Behavior Miss Rate: A key feature of HomeSnitch is identifying when new behaviors occur and finding the closest match. To measure the effectiveness of new behavior detection and resistance to under-fitting, we measure the *Unknown Behavior Miss Rate* (UBMR). UMBR is the percentage of data-points in a given data-set belonging to previously unseen classes that fail to be identified as a new class. To calculate UBMR, we used a leave-one-out approach in which we iterated through each of the different behavior classes. For each behavior, we removed all flows corresponding to it from the training data-set. We then trained the classifier using the remaining behaviors and then attempted to label flows belonging to the examined behavior class, determining the confidence score of each classification. We compared results of both kNN, Gradient Boost, and Random Forest classifiers. Since we are attempting to identify unknown behaviors, a low confidence score indicates a better result (more likely to be identified as a new class).

In our evaluation, Random Forests provided better calibrated confidence scores of unknown behavior over kNN and Gradient Boost. In contrast, the other approaches attempted to pull data points belonging to unknown behavior classes into existing classifications. This result supported our initial intuition that an bagging-based ensemble classifier would be more resistant to under-fitting. Figure 4.5 depicts a ROC curve demonstrating the impact of threshold selection on true positive rate (TPR) and UBMR for the YourThings data-set. Our results indicate that a threshold of 0.89 provided the best trade-off between accurately classifying behaviors against identifying unknown behaviors with a 11.96% UBMR and 96.82% TPR. However, we acknowledge that a TPR of 96.82% would certainly raise false alerts for a real-world appliance. Thus, the selection of the threshold must consider the balance of security and usability.

4.6.2 RQ2: Network Performance Evaluation

In this section, we examine the performance associated with monitoring network flows. Setting up a testbed with many physical devices is prohibitively complex. Therefore, we perform experiments in two testbeds: physical and emulated. For the physical testbed, we simulate many IoT devices by having one device increase its connection rate to match that of many devices. For the emulated testbed, we emulate many devices in a *de facto* SDN emulator and confirm the efficacy of the results of the physical testbed.

Experimental Setup: We used the *iPerf* toolkit to actively measure throughput and jitter (i.e., the latency variations among packets). To parameterize *iPerf*, we drew from the flow duration, bandwidth, and protocols of devices of the YourThings data-set discussed in Section 4.6.1. We found an average smart home IoT flow duration of 22.72 seconds. 64.72% of devices connected for less than a second while 17.64% of flows exceeded a minute. The maximum flow in the YourThings data-set lasted 286.32 seconds. Connections occurred both with and without encryption. We compared the results of our HomeSnitch prototype against a SDN controller application performing MAC-layer matching, and a traditional MAC-layer switch. To replicate a traditional MAC-layer controller we connected our virtual switch to the RYU *Simple_Switch.py* application included in the RYU documentation. To measure against a baseline, we configured a separate router with the default firmware.

Physical Testbed: We measured the performance of HomeSnitch over 1,000 iterations using the iPerf toolkit and report average results with a 95% confidence interval. Each flow lasted 10 seconds, and we measured the achieved throughput and jitter at 100ms intervals. Although we cannot completely guarantee outside interference in our experiment, we took conservative actions to prevent frame interference on the wireless standard. The IEEE 802.11n 2.4 GHz wireless standard supports three non-overlapping 20 MHz width channels in the US (1,6, and 11). We connected the workstation to channel 11, as channels 1 and 6

Table 4.3 Performance Impact on a Single Connection

Test	Throughput (Mbps)	Latency (ms)
Physical (Baseline)	15.848 ± 0.087	183.988 ± 2.154
Physical (RYU SimpleSwitch)	15.332 ± 0.096	278.552 ± 3.024
Physical (HomeSnitch)	15.375 ± 0.238	283.452 ± 4.333

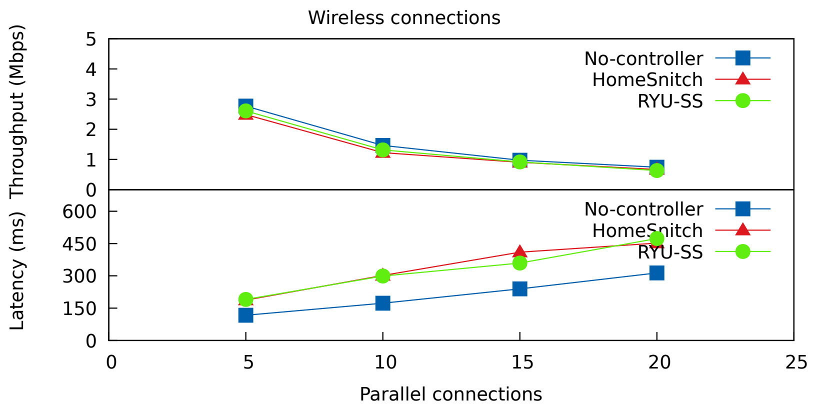


Figure 4.6 Comparison of the impact of multiple wireless connections on latency and throughput for a controller-free wireless router, a simple layer-2 controller application and HomeSnitch.

were used by neighboring networks. We disconnected all other wireless devices from the workstation to avoid introducing collisions. Table 4.3 shows the impact on a single wireless device. Figure 4.6 illustrates the impact of additional devices (simulated by parallel connections) on throughput and latency. As depicted in Table 4.3 and Figure 4.6, there is a 100ms latency overhead incurred for using an SDN controller HomeSnitch implementation to process flows. This is consistent for both the SimpleSwitch and HomeSnitch implementations. However, this has little impact on throughput, retaining 97.01% of the performance.

Emulated Testbed: To understand the performance trade-off associated with the increasing scale of devices, we emulated twenty-five unique wireless stations using the *MiniNet WiFi* framework [FR18]. We hosted this emulation on a workstation physically connected to our router. Stations were created using the 802.11 protocol, placed on the same channel, with randomized distances from the access point. We measured the throughput and jitter for connections to twenty-five unique iPerf server instances. Our results mirrored the results on the physical testbed with a nominal effect on throughput.

4.6.3 RQ3: Required Training

We constructed the cumulative distribution function graph represented in Figure 4.7 by recording the predicted class probability scores for samples from each of the 148 behaviors in our model. Figure 4.7

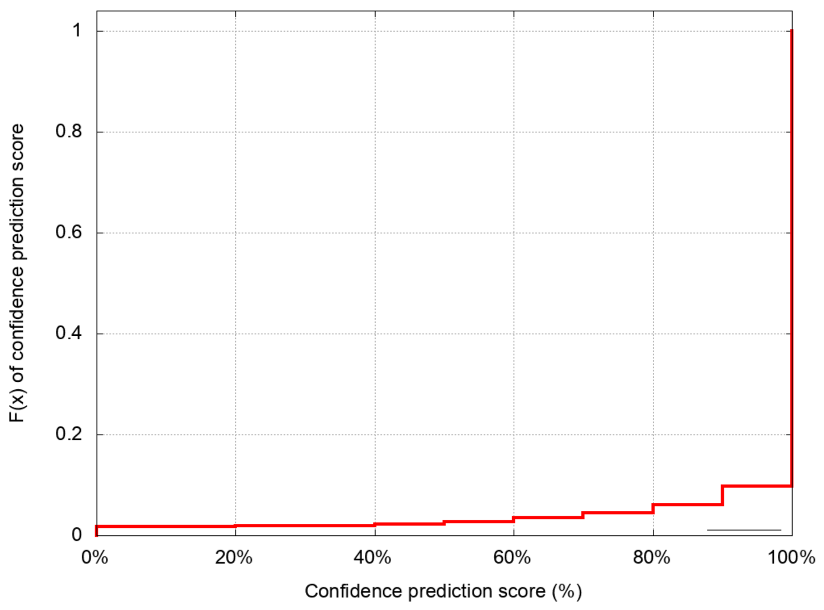


Figure 4.7 CDF for the confidence scores for the 148 predicted behaviors in the 46 device YourThings data-set.

plots our results comparing the confidence prediction score against the cumulative distribution function. The resulting graph demonstrates our model predicts behaviors with a high degree of confidence. In our experiments, we discovered the monolithic nature of smart home devices require few samples to train our model to identify multiple behavior classes. To evaluate the number of required training samples, we used a test-training set approach in which we iterated through 200 samples for each behavior. For each unique class, we removed all the samples from the class. We then incrementally added back each sample as a member of the behavior class. At each increment, we averaged the prediction score of the remaining unlabeled samples from the behavior class. We observed that behaviors required an average of 58.82 and 62.28 samples to achieve 90% and 95% prediction confidence.

4.7 Empirical Study

HomeSnitch uses supervised learning to classify features from connection-oriented application data unit exchanges. The behavior classification allows HomeSnitch to inform the user of device activities and restrict unwanted activities. In this section, we empirically evaluate HomeSnitch's ability to classify device behaviors, identify unknown behavior, and prevent an unwanted behavior by answering the following research questions.

RQ4 (*Behavior Classification*): How effective is HomeSnitch at classifying behaviors in a typical smart home environment?

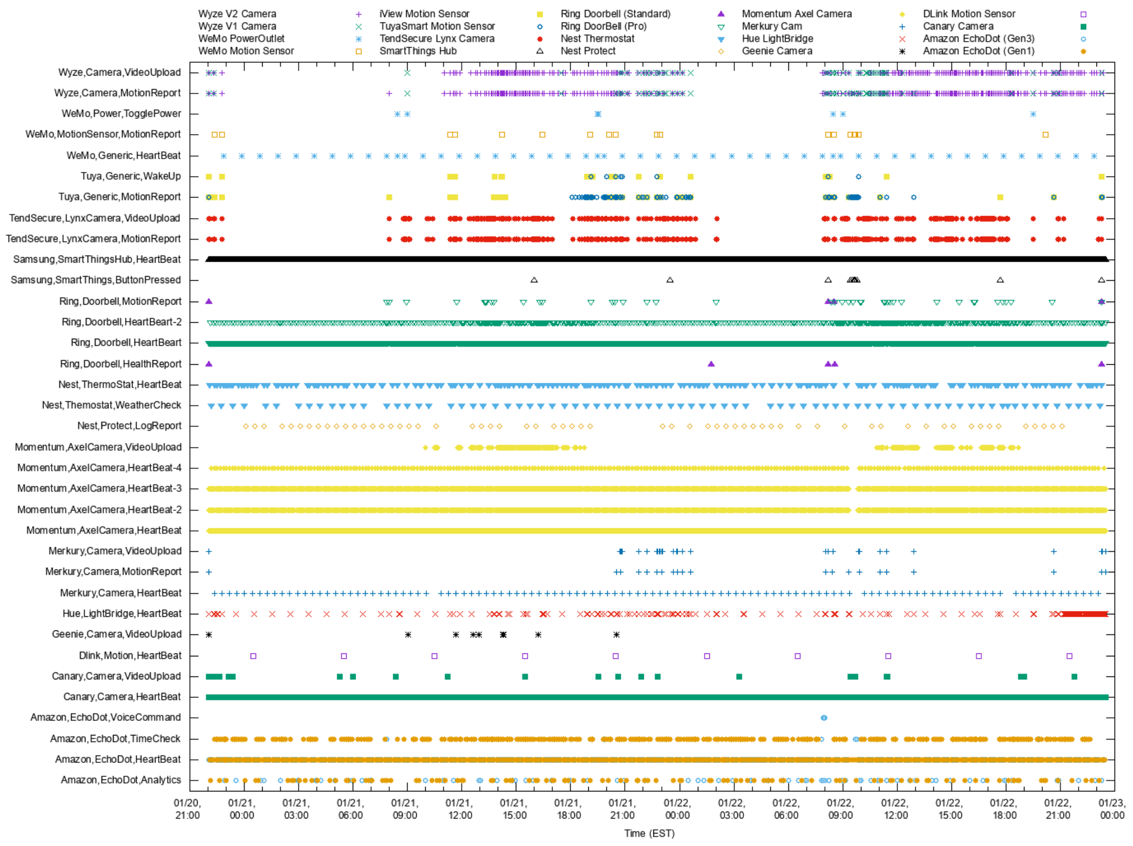


Figure 4.8 Top 35 behaviors over 48 hours sample. HomeSnitch provides semantic behavior transparency and reporting for smart home devices. Note that all clusters were identified by HomeSnitch without training on prior data-sets. The semantic labels were manually created using HomeSnitch's behavior reporting feature to determine and assign descriptive labels.

RQ5 (*Malicious Behavior*): Is it possible for HomeSnitch to identify a malicious attack scenario?

RQ6 (*Policy Expression*): Given a known behavior, can a HomeSnitch policy prevent the behavior?

Experimental Setup: Our experimental setup consisted of 20 devices in active use at the residence of one of the authors of the paper. These devices represent different categories including automation, fitness, appliance, and security camera functions. We configured HomeSnitch to log and classify all smart home IoT device traffic in our environment over two days. We correlated user activities (e.g., turning on/off lights, resetting the coffee pot filter, and triggering motion events) to the observed behaviors classes.

4.7.1 RQ4: Behavior Classification

We observed HomeSnitch's ability to manage security and privacy by classifying behaviors and enforcing policies against 20 devices on the home-network of one of the primary authors.

Table 4.4 HomeSnitch has accuracy across deployments by using features that are transport and destination agnostic.

Device	**ADUs**	**Accuracy**
Amazon Echo Dot	302	90.40%
Black & Decker Crockpot	914	98.47%
Canary Security Camera	15,426	98.88%
Hue Light Bridge	2,362	99.58%

Encryption Observations: HomeSnitch classifies behaviors regardless of the use of encryption or proprietary protocols. In our empirical evaluation, we observed that 76.22% of smart home traffic in our environment was encrypted. These findings support the observations by Sivanathan et al. [Siv17], who identified 65% of IoT device and 70% of all traffic on the Internet is encrypted. This result confirms that packet payload inspection cannot effectively classify IoT application layer protocols and behaviors.

Labeling Behaviors: We applied descriptive labels to the over 34,000 application exchanges recorded over two days. To apply descriptive labels to behavior clusters, we correlated our user-driven actions and leveraged insight from behavior reports. Utilizing our behavior reporting framework, and correlating user-driven activity, we were able to assign descriptive names to behavior triples: *(<vendor>, <model>, <activity>)*. For example, we correlated motion sensing from the Ring Companion application to a specific cluster and labeled the cluster as *(Ring, Doorbell, MotionUpload)*.

Results: Figure 4.8 shows the top 35 behaviors from our empirical evaluation, which accounts for over 90.39% of all application data exchanges over the period. Heartbeat behaviors (i.e., a fixed pattern that occurs periodically) account for over 55.09% of all network flows. Several devices have multiple Heartbeats. For example, the Canary device has two similar Heartbeat behaviors that poll at ten and one minute intervals. Heartbeats enable perpetual connectivity for devices by establishing meet-in-the-middle connections on cloud platforms. Our results demonstrate that user driven actuator control (e.g., turning on lights) and sensory reporting (motion, video uploads) have different signatures from behaviors and can be accurately predicted. Evaluating the labeled data-set, we record a 99.40% cross validation score of the 34,109 labeled behaviors.

Generality of Model: To evaluate how our approach extends across deployments, we classified traffic captures from four devices in the YourThings data-set that overlapped our empirical network. We detected the behaviors from the Echo Dot, Crock-pot, Canary Camera, and Hue Light Bridge with high confidence to samples we collected separately as depicted in Table 4.4. In contrast, we had difficulty classifying a Ring Doorbell. Subsequently, we learned there was a model mismatch between the original and professional versions of the device. The functionality of the newer device, and by extension the behaviors, differed from the original.

4.7.2 RQ5: Malicious Behavior

We evaluated HomeSnitch's ability to classify a malicious behavior by simulating an attack of a smart plug. The TP-Link HS110 is a smart plug that an end-user can control via an app. By interacting with the smart plug, the app can monitor energy consumption and schedule actions. Stroetmann and Esser [SE17] identified that the app to device communication consisted of a protocol encrypted with an XOR cipher. We hosted a malicious copy of the device firmware on our server. Further, we modified Stroetmann's work to build a tool that forced the device to download the malicious firmware. Next, we trained HomeSnitch to recognize a firmware download to the device by observing downloads of benign firmware.

Results: HomeSnitch identified each individual attempt to download the firmware and subsequently logged the activity. This alert offers insight to a user that their device has become compromised by downloading firmware from a malicious server. HomeSnitch provides the ability to train the system and write rules to prevent against such a download (i.e., by white-listing downloads to only the official site). However, we envision a service provider (similar to virus reporting) could provide both behavior training signatures and rules for enforcing behaviors for specific devices.

4.7.3 RQ6: Policy Expression

To illustrate the effectiveness of HomeSnitch's policy enforcement mechanism, we demonstrated the ability for HomeSnitch to create rules to block Ring Doorbell motion detection for a ten-minute period to protect privacy. HomeSnitch provides a web interface to apply policy actions such as disabling motion detection for specific IoT devices. After enabling a rule for our doorbell, we attempted to trigger the motion detection. No motion detection was alerted to the user or logged in the history of the smartphone app. The flows were stopped by the data plane. The flow modifications alerted that 407 packets (31,462 total bytes) were dropped when the doorbell attempted to upload the motion events. When our policy expired, the flow modification was removed and we observed that packets flowed unrestricted to the Ring servers. We generated new motion events and noted that the new events were reported in the app history. However, somewhat surprisingly, we identified that the doorbell did not buffer any of the events that occurred while it was restricted by our policy. Next, we applied a similar rule preventing our D-Link Motion Sensor from uploading motion sensing data. As a result, 26 packets (1,936 total bytes) were dropped. This prevented motion uploads from being displayed on our smartphone's app. However, when the rule expired, the motion sensor uploaded the buffered motion data. This contrast illustrates that the effectiveness of policy enforcement ultimately relies on understanding of the overall buffering and communications model of each IoT devices. Chapter 5 offers further insight on restricting per-device behavior.

4.8 Limitations

Mimicry Attacks: Our supervised machine learning approach leverages an ensemble-method classifier to identify traffic patterns that represent device behaviors. Our work does not address the case of an informed adversary that is able to compromise a device and mimic the traffic patterns of permitted behaviors. An adversary could mask a command and control channel inside traffic. While this attack would be successful in misleading our classification model, an attacker would still need to learn the contents of the policy to overcome its restrictions. Further, the policy could be specified in such a way to permit behaviors to only specific trusted servers. However, we do not entirely rule out the feasibility for an attacker to both mislead the classification and overcome the policy. In this case, we must consider complementary approaches including device identity, anti-tampering, and device attestation [Abe16; Zha16; Mah10]

Partial flow analysis: HomeSnitch requires capturing the full network flow for precise classification. To enable real-time enforcement, we construct flow modification instructions based on historical flow data. However, given our observations of the simplistic nature of smart home device behaviors, we believe it feasible to construct a stochastic model that could real-time predict behaviors with a brief packet queue. We reserve this topic for future work.

4.9 Related Work

The closest related work to HomeSnitch are efforts designed to identify IoT devices based on their network traffic patterns. IoTSentinel [Mie17] use static features such as protocol types, IP addresses, and port addresses to fingerprint devices. However, these features alone cannot differentiate semantic behaviors performed by the fingerprinted devices. In concurrent work, IoTSense [Bez18] expands on IoTSentinel by using device behaviors to fingerprint IoT devices. To some extent, the features used by IotSense are similar to those used by HomeSnitch. However, their notion of "behavior" is different. They do not seek to actually label behaviors. Rather, their goal is strictly fingerprinting devices. This difference in goal causes IoTSense to not consider *new* behaviors when classifying traffic. As we show in Section 4.6, the Gradient Boost algorithm used by IoTSense performs poorly when identifying new behaviors.

Also concurrent to our work, Peek-a-Boo [Aca18] seeks to identify behaviors, such as turning on a light switch, from network traffic. Peek-a-Boo is an attack technique designed to invade the privacy of smart home users. In contrast, HomeSnitch seeks to provide a defense. In doing so, HomeSnitch considers a broader definition of behaviors, including not only user actions (e.g., turning on a switch), but also hidden activities performed by devices (e.g., heartbeats, analytics). Interestingly, Peek-a-Book also chooses kNN over Random Forests to classify traffic; however, it does not consider the detection of previously unclassified (i.e., new) behaviors. HoMonit [Zha18] also seeks to infer behaviors from encrypted wireless traffic. However, their work is narrowly focused on the communication between devices and a SmartThing's hub to implement anomaly detection.

Several solutions have also been proposed for the detection and prevention of attacks in smart home IoT networks. DioT[Ngu18] and IoTPOT [Pa15] focused exclusively on the classification of anomalous traffic in order to detect compromised IoT device bot-net activity. Existing solutions are limited in their ability to enforce policy that considers behavior and context [Yu15]. ContextIoT [Jia17a] provides a vendor-specific solution for the SmartThings platform and does not generalize. IDIoT [Bar17] white-lists devices to the minimal set of device behaviors based on flow information. However, the use of perpetual cloud connections and changing functionality of devices hinders the practicality of exclusively using flow data. In contrast, HomeSnitch offers behavioral transparency for both known and anomalous behaviors. This approach allows us to realize behavior transparency and implement practical context-aware policy.

4.10 Conclusion

This paper presented HomeSnitch, a building block for enhancing transparency and control for end-users by classifying smart home device communication by *behaviors*. HomeSnitch leverages the vantage point of software-defined networking to monitor device/server dialogue exchanges to classify device behaviors and identify new and unknown behaviors. By selecting destination and content-agnostic features, HomeSnitch can classify device behaviors even in the presence of encryption and proprietary protocols. We implemented HomeSnitch on a commodity wireless router and built upon SDN primitives to enforce network access controls. Using a Random Forest Classifier, we achieved over 99% accuracy for classifying behaviors from an independent data-set. Through these efforts, we demonstrated the effectiveness of network-level services to classify behaviors and enforce control on IoT devices.

CHAPTER

5 BLINDED AND CONFUSED

5.1 Introduction

The rapid growth of smart-home IoT devices offers convenience, connecting us to a broad-array of sensors and actuators in our homes. The always-responsive nature of IoT provides on-demand access to seamlessly monitor and control every aspect of our homes. For example, smart-locks allow us to remotely schedule and control access to our homes from a smart phone, and connected doorbells can detect motion and send video push-notifications to our smart phones. The always-on, always-connected nature of smart home IoT devices also offers extensive forensic evidence for criminal investigations and legal proceedings. For example, data from Fitbit, Google Nest, Amazon Echo, and Ring Doorbell devices have aided law enforcement in solving crimes [McH15; HC18]. However, the swift adoption of the IoT smart home market makes these devices, and the information they provide, vulnerable to attacks that compromise privacy and security [Sol18; New18; Apt17; Gre17b].

Most smart home IoT research focuses on protecting the confidentiality of the privacy-sensitive information they generate [Sik17; Bar17; Jia17b; OH16]. While such efforts have improved the confidentiality of IoT devices, there remains a gap in availability and integrity of the information they provide. Recent high-profile examples have illustrated what happens when IoT availability fails, blinding or separating the user from the device. A flawed software update for NEST Thermostats caused a battery drain that deactivated devices, shutting off heat and leaving remote users disconnected [Bil16; Moo16]. While the NEST failure unintentionally disrupted availability, an adversary can craft attacks to degrade availability of smart-home devices. A recent attack against the Amazon Key service demonstrated a crafted 802.11 de-authentication frame could knock a smart-lock offline, forcing the lock to remain unlocked and denying remote access to the user [Hon17].

We hypothesize that the NEST and Amazon Key incidents are not isolated occurrences, but rather indicative of a larger systemic design flaw in many IoT devices. Specifically, designers falsely assume devices are always connected. By violating this assumption, attackers may blind devices and confuse their state by selectively suppressing device telemetry (i.e., data collected and transmitted to the cloud). Telemetry may be classified into channels for each source of data [Goo18]. These channels carry data from two IoT subsystems: the *always-responsive* and the *on-demand* subsystems [Mon18]. To enable an always-connected environment, devices rely on the always-responsive subsystem to present a continually connected view of devices. In contrast, the on-demand subsystem delivers sensor measurements to remote servers and implements actuator functionality. By selectively suppressing one or both of these channels, IoT device functionality can be completely disrupted.

Coarse-grained and fine-grained approaches differ in their ability to suppress device telemetry, and therefore, it is important to characterize the practical implications. *Transparent* attacks provide no indication to the end user. *Permanent* attacks eliminate uploading buffered content after the attack. For example, coarse-grained suppression (i.e., jamming) can blind devices and confuse a device's state; however, it is largely ineffective at controlling user perception as it may trigger device alerts and in-app status changes. In contrast, selectively dropping on-demand packets can blind devices without an impact on user perception.

In this paper, we broadly characterize how smart home IoT devices address the availability and integrity of telemetry reporting. We analyze vulnerabilities in 24 popular consumer smart home devices that span a breadth of device types, including connected motion sensors, security cameras, doorbells, garage door openers, and locks. Our analysis identifies a systemic design flaw that enables sensor blinding and state confusion attacks. In each device we studied, we find instances of device implementation immaturity that allows an IoT device to be trivially disrupted.

This paper makes the following contributions:

- *We propose an attack methodology against telemetry protocols and buffering for smart home IoT devices.* Our attack is capable of blinding sensors and confusing the state of actuators.

- *We evaluate the susceptibility 24 popular consumer smart home devices.* We find that 22 suffer from design flaws that enable attacks including: (1) transparently disrupting the reporting of device status alerts, and (2) preventing the uploading of content integral to the device's core functionality.

Findings: We uncover critical design flaws that inform several broad findings. First, IoT sensor devices commonly isolate the telemetry for on-demand and always-responsive subsystems without co-mingling state and sensor knowledge, allowing an attacker to independently blind or suppress either subsystem.

Second, battery constraints often cause vendors to eliminate the always-responsive subsystem by increasing or eliminating timeouts, allowing an attacker to blind the on-demand subsystem. Third, smart home actuator devices commonly fail to repudiate the delivery of state change messages, exposing the devices to state confusion attacks. Fourth, immature IoT implementations fail to buffer sensor measurements and state changes despite their computational capacity to do so.

Organization: Section 5.2 provides background, motivates our work, and details the adversary threat model. In Section 5.3, we expand the details of sensor blinding and state confusion attacks. In Section 5.4, we uncover the design flaws from a broad array of devices and summarize our findings in a list of recommendation. Section 5.5 offers insight for defense. Section 5.6 discusses related work. Section 5.7 summarizes our conclusions.

5.2 Background & Motivation

The offline actions and message telemetry protocols for smart home devices are fundamentally different. Vendors distinguish themselves by implementing a variety of strategies for telemetry protocols and offline content buffering and state management. For example, the Amazon Echo is designed to exist always-connected to the cloud. The automated speech recognition and natural language processing algorithms for the device are exclusively processed server-side [Anb18]. In contrast, Samsung's SmartThings framework provides a combination of cloud and local storage and processing with the introduction of a local controller [HH16]. The SmartThings controller (i.e., hub) offers optional processing, memory, and storage not available at the limited device level (e.g., a water leak sensor).

Buffering strategies on devices also vary. The CleverLoop Security Camera uses a machine learning algorithm to filter out unimportant movements and record only essential videos, locally storing seven rolling days of video alerts [Cle18]. Once reconnected, the CleverLoop camera uploads up to 8GB of data to cloud-based severs. In contrast, the WyzeCam Camera provides optional SD-Card storage for the device, storing offline video and motion events only locally on the card. Poor design decisions for offline buffering and telemetry introduce two unique attack vectors: *sensor blinding* and *state confusion*. The following section reviews the telemetry of IoT devices to provide necessary context for these attack vectors.

5.2.1 Overview of IoT Telemetry

IoT devices consist of two subsystems: *always-responsive* and *on-demand* [Mon18]. The always-responsive subsystem maintains a perpetual connection to remote servers to report the availability of the device and listen for server-side instructions. In turn, the servers use low-bandwidth messages to monitor connectivity health. We label this message exchange *heartbeats*, since they periodically indicate the connectivity health of a device. When a timeout expires without receiving any heartbeats, servers mark

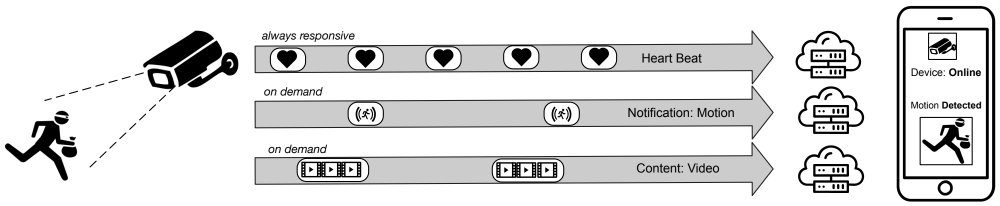


Figure 5.1 The systemic isolation of connectivity (e.g., heart beat), event notification, and content (e.g., video recording) channels for IoT devices present a blinding vulnerability when a channel is independently suppressed.

the device as offline and present the user with a smart phone alert. In our experiments, we measured the timeout period as brief as forty seconds and as long as thirty minutes. Further, some battery constrained devices entirely eliminate the always-responsive subsystem due to the power constraints of periodic messaging.

Conversely, the on-demand subsystem delivers environmental sensor measurements and actuator state changes to remote servers. As an example, a security camera may send an initial motion sensor notification before uploading a video recording. We describe the initial low-bandwidth messages as on-demand *event notifications*, since they are prioritized notifications of a triggered on-demand event. We label the video recording as on-demand *content messages*, since they are often high-bandwidth messages carrying the content of a triggered event. Event notifications and content messages are the primary channels for sensor reporting. Actuators commonly use only event notifications to report state changes.

As depicted in Figure 5.1, devices may establish independent channels for heartbeats, event notifications, and content messages. Channel independence often occurs by network flow division, separating channels among different protocols, transport layer ports, and servers. In contrast, some devices create channel separation through time division by sharing time slots on the same network flow. In this case, a device may interleave event notification and content messages between periodic heartbeats. Attacking *flow division* and *time division* channels require unique approaches, described in Section 5.3.

Sensor Blinding: Sensor blinding attacks the integrity and availability of sensor devices by preventing the delivery of sensory measurements to always-connected IoT servers. Sensor-based IoT devices summarize raw sensory measurements (e.g., motion detection, video recording, auditory, water detection, or other environmental sensors) and report this information. These devices rely on uninterrupted and untampered delivery of messages. However, we discovered in our experiments that devices commonly fail to ensure server-side delivery of sensory measurements. As an example, our evaluation identified the Ring Video Doorbell did not buffer any of the historical events that occurred while the device lacked connectivity. An attacker can exploit this design flaw to blind the device's notification mechanism, which reports motion detection. While our work focuses on smart home devices, contemporary research has shown weaknesses in protection mechanisms for connected sensors by maliciously spoofing sensor reporting, leading to isolated attacks against the US Department of Vehicles [Che18b] and agricultural cyber-physical systems [Ulu14].

State Confusion: State confusion attacks the integrity of actuator devices' state reported to always-connected IoT servers. State confusion is a special subset of sensor blinding attacks, where the sensor blinding impacts actuators. Actuators implement mechanical movement and control, changing the physical state of a device. Disrupting connectivity for actuator devices can pin the device in a fixed state, which can have disastrous consequences for actuator devices (e.g., smart-locks, garage doors, sprinkler systems, or smart-outlets). For example, the Amazon Key service attack demonstrated that a simple 802.11 de-authentication frame could knock a connected smart-lock offline, pinning the device in the current unlocked state [Hon17]. Once disconnected from the wireless network, the device remained indefinitely in the unlock state. Related research has shown the ability to influence a drone's state by GPS spoofing, followed by jamming the control channel, forcing the drone to crash while in the spoofed state [Ker14].

5.2.2 Threat Model

There are many reasons why an attacker may wish to blind sensors or confuse states of smart home IoT devices. Many smart home devices are used to provide physical security. Therefore, criminals may use sensor blinding and state confusion of actuators to gain physical access to a home without creating digital forensic evidence. Criminals may attack homes opportunistically by themselves, or they may be part of organized crime, which may obtain access to tens of thousands of wireless routers. Outside of this traditional threat model, the attacker may also be the perpetrator of domestic abuse. Recent popular media articles [Bow18] have reported instances of domestic abuse that involves stalking using smart home devices. In addition to these attacks, a domestic partner may blind sensors to eliminate forensic evidence or confuse states of actuators to retain physical access to areas.

5.2.2.1 Attacker Goals

We consider an attacker whose goal is to disable the core functionality of a device by blinding a sensor or confusing the state of an actuator. As an illustration, we imagine an attacker that has the goal of circumventing a security camera without triggering an alert or being recorded, or preventing a smart-lock from securing a residence. Ultimately the success of the attacker relies on two variables: the ability to avoid producing alerts (a transparent attack) and the ability to avoid leaving a record (a permanent attack).

5.2.2.2 Attacker Capabilities

An attacker may execute a coarse-grained approach of physical layer suppression (e.g, jamming). Alternatively, the attacker may have or can gain access to the wireless router to implement a fine-grained network layer suppression. We consider the following two attacker capabilities:

Physical Layer Suppression: A physically located attacker can broadly jamming the wireless signal, or exploit flaws in the 802.11 networking scheme including forging un-encrypted de-authentication frames or flooding carrier-sense management frames. Such an adversary can jam traffic at the physical layer but will not achieve their objective of a stealthy attack. Jamming suppresses all device traffic and is a coarse-grained approach that will provide the user with an indication of the attack.

Local Network Layer Suppression: An attacker with control over the wireless router has fine-grained ability to selectively suppress packets. To compromise a wireless router, an attacker may gain access through several different attack vectors [VP18; VP17; VP16; VP15; VP14]. The attack may be mounted remotely from the Internet, or locally from a compromised or malicious device on the LAN. Alternatively, the attack may have a pre-existing relationship with the victim and been given administrative access to the router. This fine-grained access permits selectively suppressing on-demand traffic without adversely affecting user perception of device availability. Once controlling the router, the attacker can control traffic through a variety of techniques: 1) remotely proxying device traffic, 2) local establishment of firewall policies, or 3) connecting the router to a criminally-run remote security orchestrator that selectively suppresses on-demand security camera traffic.

5.2.2.3 Attacker Assumptions

Our approach relies on two key assumptions: 1) the attacker can fingerprint a device and 2) the attacker understands the telemetry model for the target device.

Device Fingerprinting: We assume the attacker can fingerprint a device. This is a reasonable assumption as several recent works have demonstrated highly accurate device fingerprinting with nominal traffic [Mie17; Bez18; Aca18; Apt17]. Further, devices can be identified using IoT reconnaissance tools such as IoTScanner [Sib17] or intercepting medium access control layer headers.

Telemetry Models: We assume the attacker can learn the telemetry model for a specific device. The attack model is generic as it applies across device types, vendors and models. However, the attacker requires knowledge of specific device telemetry and messaging protocols to succeed. As smart home devices are commercial in nature, an attacker can procure and study the telemetry protocol for any device in preparation for an attack.

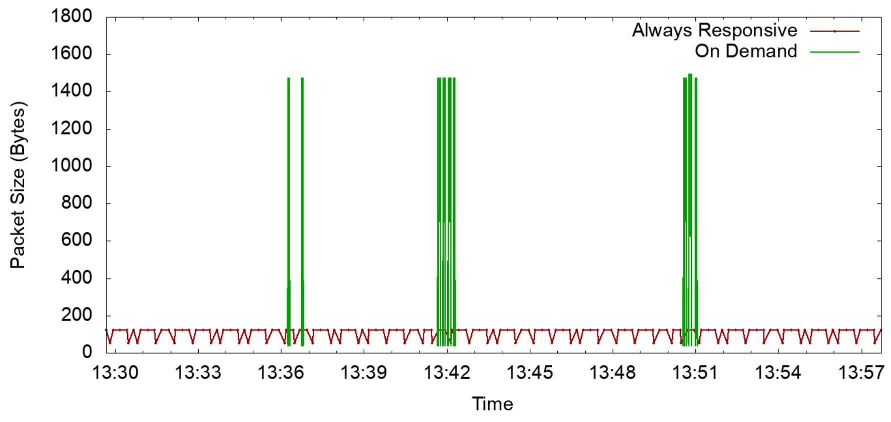


Figure 5.2 The simplistic nature of IoT devices facilitates subsystem classification at the packet layer, enabling an attacker to isolate on-demand events.

5.3 Methodology

In this section, we discuss the different design features that enable attacks against IoT telemetry by considering telemetry division, message timeouts, and buffering decisions. Further, we offer insight for understanding attack severity by discussing attack transparency and permanence.

5.3.1 Attacking Flow Division Telemetry

Flow division telemetry, depicted in Figure 5.1, separates heartbeats, notifications, and content across distinct network flows. To illustrate this isolation, consider the *Tend Secure Lynx Indoor Security Camera*, a connected surveillance camera which supports motion detection and high definition video. In our observations, we determined the camera sent heartbeats to an AWS server using the TinyMessage protocol (TCP port 5104) and separately delivered notification and content messages to AWS servers over SSL. Isolating the always-responsive and on-demand subsection facilitates ideal conditions for sensor blinding and state confusion attacks by enabling the attacker to easily identify and blind the on-demand subsystem. An attack can benefit from the simplistic nature of IoT to classify the data intensive on-demand subsystem and the periodic always-responsive subsystem. In the case of the Lynx Camera, Figure 5.2 depicts three separate on-demand video recordings that peak from the routine low-bandwidth heartbeat messages.

Transparent Blinding: The design flaw of incorrectly separating the privilege of each subsystem enables our attack. In the case of flow division telemetry, devices rarely co-mingle the functionality of the always-responsive and on-demand subsystems. Specifically, a heartbeat is solely responsible for reporting connectivity health and does not advertise on-demand events. Conversely, on-demand messages (notifications, and content) have no impact on connectivity health. By their nature, on-demand subsystem

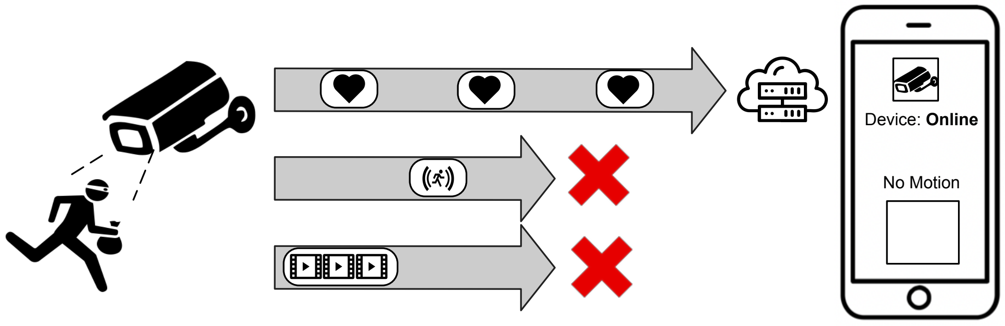


Figure 5.3 On storage constrained IoT devices, the lack of buffering event notifications and content in embedded channels presents a blinding vulnerability when individual packets bearing notification or content are suppressed.

messages are often delivered without prior notification. Thus, an attacker can transparently suppress the on-demand flows, creating the conditions for sensor blinding and state confusion attacks. In the naive case of the Lynx Camera, a firewall rule blocking outbound SSL traffic will blind the camera's on-demand subsystem. With the firewall rule applied, an owner's companion app will report the camera as online; however, the device will silently fail to report any motion detection or video recordings. We label this attack as *transparent blinding*, as the user is unaware the camera system is failing to report on-demand events. Figure 5.3 depicts this transparent attack.

Semi-Transparent Blinding: A partial disruption of the always-responsive subsystem often results in only *semi-transparent blinding*. In this case, the companion app may display intermittent connectivity but avoids overwhelming the user with alerts. In select cases, an adversary is unable to suppress the on-demand subsystem without at least partially disrupting the always-responsive subsystem. This is common in the case of flow-division telemetry, as described in the following subsection. A full disruption of the always-responsive subsystem commonly results in a smart phone alert that informs the user of the disruption. However, a partial disruption often results in only an in-app status message. Thus, the user is only aware of the blinding if they have the companion app open during the attack. In our experiments, we found that several devices were vulnerable to semi-transparent blinding for sufficiently long periods of time. As an example, we found the Iris Hub displayed an intermittent connectivity message inside the companion app after a minute of sensor blinding but waited thirty minutes to deliver a smart phone alert. In our experiments, we also discovered that devices did not aggregate blinded time. To illustrate this flaw, we could blind the Iris hub for 29 minutes, permit 1 minute and then repeat without ever triggering an alert.

Permanent Event Blinding: Event notification and content messages are frequently discarded with on-demand connection failures. The unique storage and processing constraints of IoT enables this problem. In our experiments, we observed all device types rarely buffer on-demand events (even at hubs,

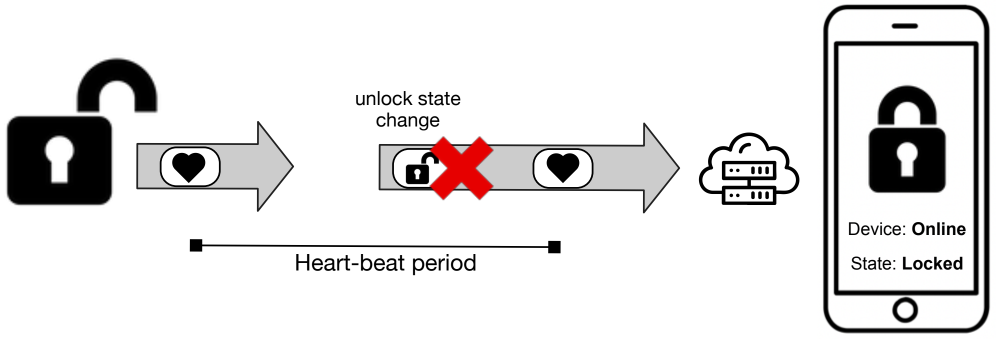


Figure 5.4 On battery constrained IoT devices, lengthy timeouts between heartbeats present an opportunity to suppress unbuffered state changes of actuators.

whose purpose is to provide auxiliary storage). As a running example, the Lynx Camera discarded video recordings while our firewall rule was established. After removing the firewall rule, the camera only uploaded newly detected on-demand events. A combination of factors including the processing power required for data transformation, limited local buffer, and low-power mode/sleep requirements may contribute to this design decision [Goo18]. However, the device is *permanently blinded* when the device fails to buffer and deliver suppressed on-demand events. The design decision of not buffering failed events enables an adversary to selectively suppress the on-demand system for a brief period, resulting in the device permanently failing to ever report the suppressed events. As we discuss in Section 5.4, an attacker can also leverage wireless denial-of-service attacks to permanently blind wireless devices by forging de-authentication frames. While several systems support auxiliary storage (e.g., secure digital memory cards) to buffer events, an attacker can remove or physically alter this storage while the device is blinded.

5.3.2 Attacking Time Division Telemetry

Attacking time division telemetry requires a more fine-grained approach that selectively suppresses individual packets instead of network flows. An attacker must permit always-responsive packets while suppressing on-demand packets to achieve the goal of transparently and permanently blinding devices. This is trivial for connection-less transport layer protocols which fail to maintain state or guarantee delivery. However, most devices rely on SSL, which uses connection-oriented TCP, guaranteeing in-order delivery. To address this challenge, an adversary can benefit from two key design failures: the lack of on-demand event buffering (as previously introduced) and lengthy timeout periods. In the following paragraphs, we address how an attacker can leverage these design failures to attack time division telemetry.

Packet Signatures: Time division telemetry attacks require identifying packets from the on-demand subsystem. The attacker must be able to distinguish between always-responsive packets and on-demand packets. An attacker can benefit from the simplistic nature of IoT to strengthen the classification accuracy. Apthorpe et al. [Apt17] previously demonstrated the ability to infer privacy-sensitive, on-demand activities from analyzing traffic rates of IP traffic. Further, Celosia et al. [CC18] observed that artifacts of Bluetooth L2CAP layer could be used to determine a device state change. In our experiments, we discovered this classification holds true for smart home devices at the packet level.

We observed that we could identify packets based on the consistent nature of the always-responsive subsystem and urgent nature of the on-demand subsystem. To illustrate this, consider the case of the Momentum Axel security camera which supports motion detection, audio detection and video recording over time divided telemetry. The camera's always-on-subsystem sends fixed size 52, 60, or 108-byte heartbeats while the on-demand subsystem sends 617 or 761 byte notification packets (depending on if audio or motion triggered the event), and 1500-byte content packets. Transport layer options can strengthen classification of on-demand activities. We discovered several on-demand packets from several devices (including from the Momentum Camera) set the TCP PUSH flag in order to promptly forward and deliver data. The transport layer uses the TCP PSH Flag to push data out of a TCP socket immediately, rather than waiting for additional data to fill the buffer [ISI81]. Finally, we discovered the always-responsive subsystem sends packets on a consistent and predictive interval. In contrast, the on-demand subsystem interleaves packets between periodic heartbeats.

Buffer Failures: Storage and processing constrained IoT devices often fail to buffer suppressed on-demand content and notifications. In our observations, the lack of acknowledgments for suppressed packets terminates connection-oriented protocols. However, the always-connected nature of IoT immediately establishes a new connection. In design, this new connection should carry the suppressed on-demand events. However, the new connection often began with always-responsive heartbeats instead of the suppressed on-demand content. We observed several cases of motion detectors, security cameras, and even smart hubs simply discarding the suppressed on-demand content and notifications. To amplify the severity of the attack, we often observed a lack of user alerts since the always-responsive subsystem succeeded in delivering a heartbeat within timeout windows. Figure 5.4 illustrates this attack scenario. When the adversary suppresses an on-demand state change, the IoT device simply discards the change and establishes a new connection. The next section expands upon our empirical evaluation and the findings and results for 24 popular smart-home devices.

5.4 Results

In this section, we describe the experimental setup for evaluating the vulnerability of 24 popular smart-home devices to sensor blinding and state confusion attacks. We intentionally chose a simple setup to ease reproducibility and provide all flow modifications to repeat the experiments used in our evaluation. We conclude this section by summarizing a list of our findings that highlight key design failures that contribute to sensor blinding and state confusion attacks against our data-set of devices.

5.4.1 Experiment Setup

We set up a laboratory smart home environment to examine the scope and severity of attacks against a broad array of devices. In our threat model, the adversary's objective is to blind sensors and confuse the state of actuators. The severity of the attack relies on the adversary's transparency and permanence. For each device in our environment, we measured the impact of network layer and physical layer suppression of traffic for a thirty-minute window.

Tested Devices: Our evaluation consisted of a representative set of 24 smart home IoT devices for consumers, available during 2018 from well-known US retailers, including Walmart, Lowe's, Target and Amazon. These devices covered IoT classes related to security cameras, motion sensors, smart home environmental monitoring, connected doorbells, garage door openers and smart-locks. Most devices connected directly through Wi-Fi to our network. In the case of smart-hubs, they were connected to the network via Ethernet and connected to their sensors and actuators via ZigBee or Z-Wave. We did not consider suppressing ZigBee or Z-Wave traffic but instead focused on selectively suppressing traffic that the smart-hub generated to cloud servers. For each device, we downloaded the iPhone companion app to manage the device. As an indication of the popularity of each device, we recorded the number of application downloads for the Android app version on the Google Play Store. (The Apple App Store does not release app download metrics). Note, the Geenie and Merkury devices use the same companion app. Further, the Iris and SmartThings sensors are managed by their respective singular apps. All other apps are single purpose apps. An overview of the tested devices is shown in Table 5.1.

Labeling Results: Our evaluation considered the ability to blind a device's sensors and confuse the state (if the device maintained state). We used full (●), half-filled (◐) and empty (○) circles to label the severity of the transparency or permanence of the attack. Under the *transparent* column, a full circle represents the attack succeeded without an in-app status change or smart phone alert. A half-filled circle indicates semi-transparent blinding (e.g., the companion app displayed an offline status message but failed to provide a smart phone alert). Under the *permanent* column, a full circle indicates that the sensor behavior or state change is never reported after the attack. A half-filled circle indicates that the attack buffered a period but not all of the attack (e.g., a camera system buffered the most recent event) or that the state confusion lasted only a brief period (e.g., the companion app for a lock remained confused for five minutes after the attack).

Network Layer Suppression: To suppress traffic with the fine-grained approach described in Section 5.3, we configured a consumer grade wireless router as a software defined switch and connected it to a local software defined controller. Our controller logic implemented OpenFlow flow modifications that matched the appropriate header fields that correlated to on-demand events for each device. For reproducibility, we

Table 5.1 To demonstrate the broad scope of the problem, we have profiled 24* popular smart home IoT devices and measured the effectiveness of sensor blinding and state confusion attacks by analyzing their transparency and permanence for these devices. Our results indicate systemic design flaws contributed to attacks against all devices classes.

Device	App Downloads	Firmware Version	Attack Type	Network Layer Suppression		Physical Layer Suppression	
				Transparent Attack	Permanent Attack	Transparent Attack	Permanent Attack
D-Link DCH-S150 Motion Sensor	100,000+	1.23	Sensor Blinding	●	○	◑[2]	○
Belkin F7C028 Motion Sensor	500,000+	2.00.11057	Sensor Blinding	◑[2]	●	◑[2]	●
Wasserstein Motion Sensor	5,000+	1.10	Sensor Blinding	●	●	●	●
iView S200 Motion Sensor	5,000+	1.10	Sensor Blinding	●	●	●	●
TuyaSmart M01 Motion Sensor	5,000+	1.10	Sensor Blinding	●	●	●	●
D-Link DCS-8010LH Camera	100,000+	1.02.02	Sensor Blinding	○	○	○	●
Momentum MOCAM-720-01 Camera	100,000+	5.1.8	Sensor Blinding	◑[3]	○	◑[3]	◑[4]
Merkury MI-CW007-199W Camera	10,000+	2.0.9	Sensor Blinding	●	●	◑[2]	●
Geenie CN-CW003 Camera	100,000+	1.10.16	Sensor Blinding	●	●	◑[2]	●
Tend Secure Lynx Indoor 2 Camera	50,000+	00.15.003	Sensor Blinding	●	●	◑	●
Wyze V1 Camera	100,000+	3.9.3.72	Sensor Blinding	●	◑[4]	◑[8]	◑[4]
Wyze V2 Camera	100,000+	4.9.3.64	Sensor Blinding	●	◑[4]	◑[8]	◑[4]
Canary 1 Security Camera	100,000+	4.0.0	Sensor Blinding	◑[5]	○	◑[5]	●
Iris Door Contact Sensor	100,000+	2.2.0.009	Sensor Blinding	◑[2]	●	–	–
Iris Motion Sensor	100,000+	2.2.0.009	Sensor Blinding	◑[2]	●	–	–
Iris Water Sensor	100,000+	2.2.0.009	Sensor Blinding	○[6]	○	–	–
SmartThings Contact Sensor	100,000,000+	000.024.00022	Sensor Blinding	●	●	–	–
SmartThings Motion Sensor	100,000,000+	000.024.00022	Sensor Blinding	●	●	–	–
SmartThings Water Sensor	100,000,000+	000.024.00022	Sensor Blinding	●	●	–	–
SmartThings Button	100,000,000+	000.024.00022	Sensor Blinding	●	●	–	–
Ring Pro Doorbell (Wired)	1,000,000+	Up to Date[1]	Sensor Blinding	◑[2]	●	◑[2]	●
Ring Doorbell (Battery)	1,000,000+	Up to Date[1]	Sensor Blinding	●	●	●	●
MyQ Garage Door Opener	500,000+	Up to Date[1]	State Confusion	◑[2]	◑[7]	◑[2]	◑[7]
Schlage Deadbolt (w/ Iris Hub)	100,000+	2.2.0.009	State Confusion	◑[2]	●	–	–
Schlage Deadbolt (w/ SmartThings Hub)	100,000,000+	000.024.00022	State Confusion	●	○	–	–
Total				15/25	16/25	4/15	11/15

● Attack succeeded.
◑ Attack partially succeeded (i.e., semi-transparent blinding and/or partial event buffering).
○ Attack failed (i.e., alerted smart phone and/or buffered all offline events).

* Our data-set consists of 24 total devices and 25 configurations since the Schlage Deadbolt is configured with both hubs.

[1] Device only reports *firmware up to date*; does not report firmware version number.
[2] Smart phone app displayed offline status; smart phone did not alert user.
[3] Smart phone app displayed online; smart phone alerted user.
[4] Device buffered a single motion detection event.
[5] Smart phone only alerts offline when app opened.
[6] Hub alerts water leak via audible alarm; smart phone did not alert user.
[7] Smart phone app is only confused for 5 minutes.
[8] Clicking a device in the smart phone app results in an error message.

have included the matching fields in Table 5.2. During the thirty-minute attack window, we stimulated on-demand activities for each device.

Physical Layer Suppression: To determine the effect of physical layer suppressed, we implemented a coarse grained denial-of-service attack by exploiting a vulnerability of the data-link layer. This approach suppressed both the always-responsive and on-demand subsystems. This attack replicated a local criminal

Table 5.2 Packet Signatures for Eliminating the On-Demand Subsystem of Devices

Device	On-Demand Packet Signature
D-Link Motion Sensor	PROTOCOL == SSL AND PAYLOAD LENGTH > 475 BYTES
Belkin Motion Sensor	PROTOCOL == TCP AND (DST PORT == 8443 OR DST PORT == 3478)
Wasserstein Motion Sensor	PROTOCOL == MQTT AND PAYLOAD CONTAINS "out"
iView Motion Sensor	PROTOCOL == MQTT AND PAYLOAD CONTAINS "out"
TuyaSmart Motion Sensor	PROTOCOL == MQTT AND PAYLOAD CONTAINS "out"
Canary Security Camera	PROTOCOL == SSL AND PACKET LENGTH == 1500 BYTES
D-Link Camera	PROTOCOL == SSL AND PSH/ACK SET and PAYLOAD LENGTH > 600 BYTES
Momentum Camera	PROTOCOL == SSL AND PSH/ACK SET and PAYLOAD LENGTH > 600 BYTES
Merkury Camera	PROTOCOL == MQTT AND PAYLOAD CONTAINS "smart/device/out"
Wyze Camera	PROTOCOL == TCP AND DPORT == 8443
Tend Secure Camera	PROTOCOL == SSL
Geenie Camera	PROTOCOL == MQTT AND PAYLOAD CONTAINS "smart/device/out"
Iris Hub	PROTOCOL == SSL AND PSH/ACK FLAGS AND PAYLOAD LENGTH > 250 BYTES
SmartThings Hub	PROTOCOL == SSL AND PAYLOAD LENGTH > 359 BYTES
Ring Pro DoorBell	PROTOCOL == SSL OR TCP DST PORT == 15063 OR 9999
Ring Doorbell	PROTOCOL == TCP AND DPORT == 80
MyQ Garage Door Opener	PROTOCOL == TCP AND PAYLOAD LENGTH == 125 BYTES

from our threat model in Section 5.2. For each device, we forged 802.11 de-authentication frames using the aireplay-ng packet injection tool [d'O19]. These forged frames disrupted the wireless connectivity of the devices, eliminating both subsystems. We recognize there are more subtle approaches to reactive jamming, that could selectively suppress the on-demand subsystem, but leave these approaches for future work [Wil11].

5.4.2 Evaluation Results

Table 5.1 summarizes the results of our evaluation. Our results demonstrate that 22 of 24 devices suffer from critical design flaws that enable attacks to transparently disrupt the reporting of device status alerts or prevent the uploading of content integral to the device's core functionality. These results confirm our hypothesis that the attacks against NEST and Amazon Key are not isolated instances. IoT developers falsely assume devices are always connected and are unprepared when that assumption is violated. In our data-set, 15 devices fail to provide the user any indication they are under an attack and 16 devices fail to buffer any content for the duration of the attack. An additional two devices fail to buffer any content when the physical channel is eliminated by jamming.

We find that our proposed attack vector, selectively suppressing network layer traffic, offers stealth over physical layer suppression. To this end, we demonstrate a reliable methodology for blinding security cameras and motion sensors that relies on attacking flow division telemetry. Our attack methodology, purposely simple and easily replicated, has significant ramifications about the forensic value of the data these always-connected devices produce. Ultimately, we uncover immature IoT implementations that fail to implement telemetry protocols and buffer sensor measurements and state changes despite their computational capacity to do so.

In the following subsection, we summarize these findings by examining the problematic designs of telemetry, battery conservation, controller storage, controller prioritization, and buffering during traffic suppression.

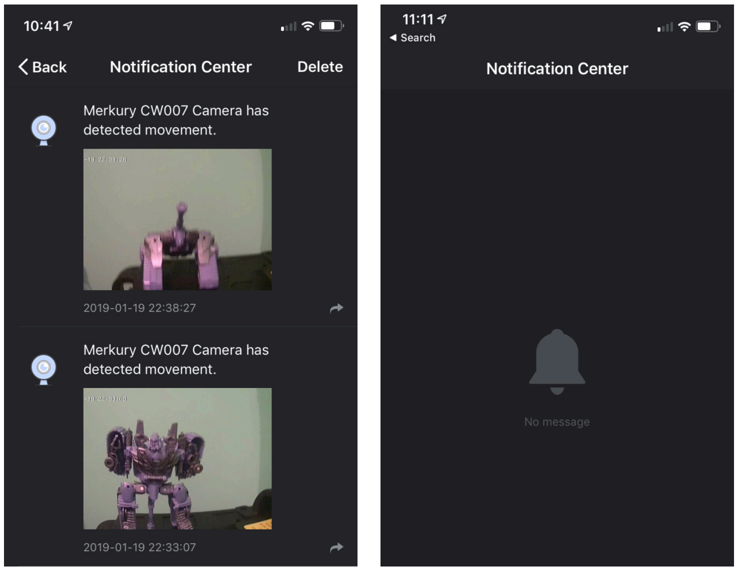
5.4.3 Evaluation Findings

Finding 1: ***Flow division telemetry commonly isolates the on-demand and always-responsive subsystems.*** The design decision to isolate telemetry channels into separate flows, without co-mingling state and sensor knowledge, facilitates sensor blinding vulnerabilities. We successfully blinded the availability of the on-demand subsystem in several devices in our evaluation data-set: including the Ring Doorbell, the Merkury, Wyze, Lynx, and Geenie security cameras and several motion sensor devices.

Figure 5.5 illustrates the results of sensor blinding the Merkury security camera. In the case of this example, we stimulated the on-demand motion sensor by placing an object in front of the camera system. We passively observed that the camera system sent the motion notification over a plain-text MQTT connection and sent heartbeats and video content over SSL. We identified the traffic from each subsystems by correlating the packet timings to the history in the companion app. We then repeated the experiment suppressing only the on-demand MQTT flow. When triggered, the camera attempted to initiate the MQTT connection but our flow modifications suppressed the corresponding packets. Despite the device uploading a video over the SSL connection, the companion app displayed a blank event history. Although the attack eliminated the on-demand channel from delivering motion notifications, the companion app failed to alert the user. After eliminating the flow modification, the camera system returned to full functionality but failed to subsequently upload the buffered motion notifications.

Finding 2: ***Battery constraints often eliminate the always responsive subsystem***. Battery constraints present a key design decision in the integrity of the always-responsive subsystem. In our experiments, we determined vendors often eliminate the always-responsive subsystem by increasing or eliminating timeouts. This trade-off conserves battery power at the cost of device responsiveness. Table 5.3 enumerates the heartbeat intervals and timeouts for all devices to provide an understanding the trade-off made by different devices. To specifically illustrate the battery paradigm, we compared the evaluation of the professional and standard version of the Ring Doorbell. The professional version is connected to low voltage power while the standard version is battery-powered. The doorbells offer similar functionality. However, the differing implementations of the always-responsive and on-demand subsystems facilitate transparent blinding against the standard version.

Figure 5.6 depicts the differing channels for the flow-division telemetry. In our evaluation, both devices transmitted a *health report* (carrying metrics about device power and signal strength) over SSL. Also, both delivered on-demand content and notifications to remote servers. The standard version used HTTP-over-TLS for content and notification, the professional version delivered notifications over TCP



Merkury CW007 Camera has detected movement.

2019-01-19 22:38:27

Merkury CW007 Camera has detected movement.

2019-01-19 22:33:07

No message

(A) Device functioning correctly (B) Device sensor blinded

Figure 5.5 The design decision to isolate telemetry channels into separate flows that do not co-mingle state and sensor knowledge leads to blinding and confusion vulnerabilities. In the figure above, the Merkury camera is functioning properly in (a) and blinded in (b).

Port 15063 and content separately. The key distinction is the always responsive subsystem, which enables streaming video. The professional version, which supports streaming video by default, transmitted periodic heartbeats over SSL. In contrast, the standard version did not send any periodic heartbeats and lacked streaming video in the default configuration. Instead of the heart-beats, the standard version uses the health reports to determine device health and connectivity. This problematic decision is amplified by the fact that the health reports are only delivered on-demand with motion sense notifications. Thus, the server does not anticipate them on any interval and they cannot provide ground truth about the device.

To blind the motion sensors of both devices, we enabled flow modifications that dropped notification and content packets. During the period of our flow modifications, our attack blinded both devices by preventing the delivery of on-demand notifications and content for motion-triggered events. Further, nether device buffered the events. However, we ran into a challenge transparently blinding the professional version. On the professional version, we could not distinguish between packets originating from the always responsive subsystem (heartbeats) and the on demand subsystem (content) since they were encapsulated in the same network flow. However, this presented no issue for the standard version since it lacked an always responsive subsystem. Thus, our flow transparently blinded the standard version. During the period of the attack, the companion app reported the standard Ring doorbell as online and failed to sense or report any motion detection events. We also found similar vulnerabilities in the Wasserstein, iView, and Tuya battery-powered systems that eliminated the always-responsive subsystem.

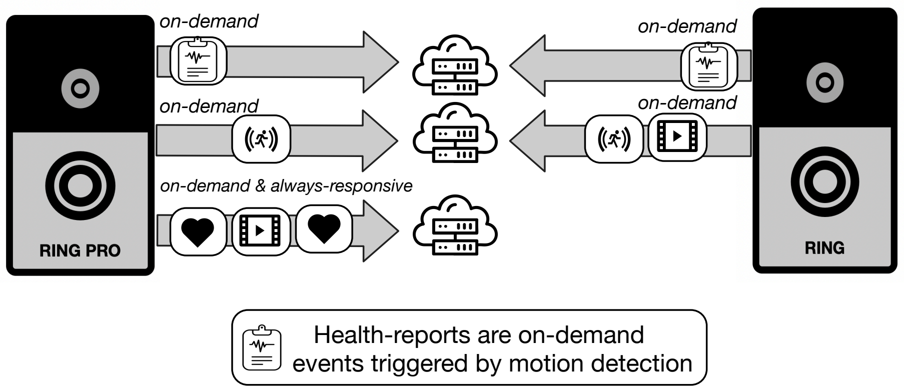

on-demand
on-demand
on-demand & always-responsive
RING PRO
on-demand
on-demand
RING
Health-reports are on-demand events triggered by motion detection

Figure 5.6 The standard Ring uses on-demand *health reports*, triggered via motion, to report the health of the device. The design decision to conserve battery by eliminating the always-responsive subsystem facilitates transparently blinding the standard Ring over the Ring Pro.

Finding 3: ***Flawed controller designs introduce device layer vulnerabilities.*** To illustrate state confusion attacks, we analyzed the different methods the Iris and SmartThings controllers handled suppressed telemetry for the Schlage Deadbolt. We included the Schlage touchscreen deadbolt in our data-set as it offers Z-Wave connectivity and supports different controllers including SmartThings, Iris, Alexa, and Wink. By pairing the deadbolt with a controller, a user can remotely control the state of the deadbolt using the controller's companion app. In our analysis, we discovered the design of the Iris controller permitted suppressing state reporting to the companion app.

Further, we identified the two controllers distinctly handled suppressed telemetry, leading to a significant vulnerability in the state reporting of the Iris controller's companion app. To understand the impact of controller design decisions, we suppressed the on-demand telemetry that reported the unlock action. Both controllers reported on-demand and always-responsive telemetry over SSL, complicating the attack. However, we determined that packets carrying the on-demand telemetry were quite larger than the always responsive heartbeat messages. Thus, our flow modifications dropped packets larger than 250 and 359 bytes for the Iris and SmartThings controllers, respectively.

While suppressing the on-demand channel, we physically unlocked the deadbolt. We verified the deadbolt did correctly report the state change over Z-wave to the controller. Subsequently, our flow modifications dropped the on-demand state change messages from both controllers to their remote servers. After the flow modifications expired, we examined the status of the companion app. Initially both controllers reported the deadbolt as locked as we had suppressed the initial state change. However, after a maximum period of 100 seconds, the SmartThings controller updated the state as unlocked. In contrast, the Iris controller failed to subsequently update the state of the lock in the companion app. Figure 5.7 depicts the companion app reporting the incorrect state of the lock thirty minutes after the flow modifications expired.

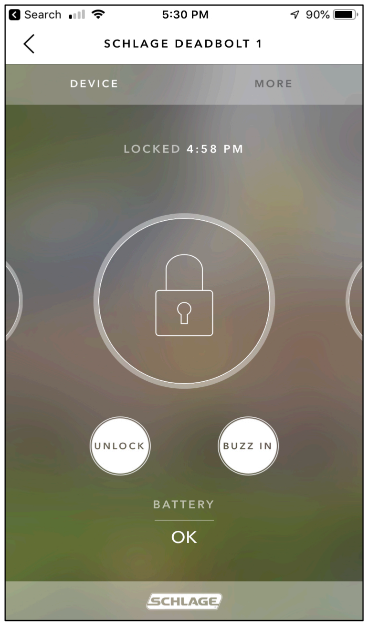
Table 5.3 Lengthy heart-beat time periods facilitate traffic suppression and blinding attacks without smart phone alerts.

Device	Telemetry	Protocols	HeartBeat (s)
D-Link Motion Sensor	Time Division	SSL	5
Belkin Motion Sensor	Flow Division	STUN, SSL, MQTT	270
Wasserstein Motion Sensor	Flow Division	MQTT, HTTP	—
iView Motion Sensor	Flow Division	MQTT, HTTP	—
TuyaSmart Motion Sensor	Flow Division	MQTT, HTTP	—
Canary Security Camera	Time Division	SSL	30
DLink Camera	Time Division	SSL	55
Momentum Camera	Time Division	SSL	30
Merkury Camera	Flow Division	SSL, MQTT, UDP	120
Wyze V1 Camera	Flow Division	SSL, MQTT-S, UDP(1001)	5
Wyze V2 Camera	Flow Division	SSL, MQTT-S, UDP(1001)	5
Tend Secure Camera	Flow Division	SSL,TinyMsg	15
Geenie Camera	Flow Division	MQTT, UDP	100
Iris Hub	Time Division	SSL	5
SmartThings Hub	Time Division	SSL	30
Ring Pro DoorBell	Flow Division	H264, SSL, RTP, TCP(9999)	30
Ring Doorbell	Flow Division	H264, SSL, RTP	—
MyQ Garage Door Opener	Time Division	MQTT-S	10

As we analyzed the telemetry of both devices, we discovered the systemic design flaw that lead to the attack. The SmartThings controller sent a periodic state update every 100 seconds. We passively observed this by locking and unlocking the deadbolt and then correlating the subsequent network traffic and smart phone app status. Even without a change to the state of the device, the SmartThings controller periodically transmitted the state of the device every 100 seconds. However, the Iris controller treated a state change as a single fixed event and only reported the state change when the action physically occurred. Thus, suppressing the single state change permanently confused the Iris smart phone app.

Finding 4: ***Controllers can prioritize buffering event notification based on severity.*** Comparing the distinct responses for a *water leak* between the Iris and SmartThings controller offers insight about buffering priorities. We evaluated this distinction by suppressing both controller's on-demand channels for thirty minutes. During the attack, we stimulated an on-demand event by placing the water leak sensors for both controllers under a running water faucet. During traffic suppression, the smart phone application for the Iris controller displayed an in-app offline status but failed to provide a smart phone alert.

However, the Iris controller distinguished itself by uniquely responding with an audible water leak alarm, lasting until the end of the attack. After ending the telemetry suppression, the Iris controller immediately uploaded the buffered water leak. In the default configuration, the back-end Iris services sent an email and phone call about the water leak. In contrast, the SmartThings app displayed no history of a previous water leak. We further examined the SmartThings device history using the SmartThings *Groovy IDE* and learned the hub never buffered or uploaded suppressed telemetry (i.e., the water leak notification). This behavior correlated to the sensor behavior we saw for SmartThings contact, motion,



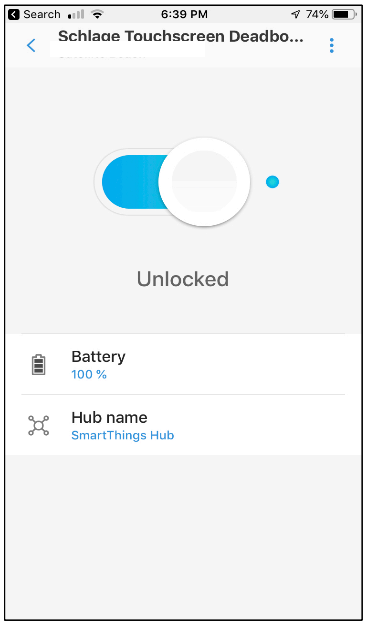


Figure 5.7 The Iris controller fails to buffer and re-transmit suppressed telemetry of the Schlage Deadbolt, facilitating the problematic case where the Iris app falsely reports the deadbolt as locked when an unlock action occurs during telemetry suppression.

and button sensors that contained blank event histories after telemetry suppression. The SmartThings hub has 512MB DDR3 RAM, and 4GB of Flash Memory. After operating system demands, memory and storage exists to buffer priority events. In our observations, the SmartThings hub encapsulated a water leak detection message in a single packet of 418 bytes that consisted of 20 fields of a record. For high priority events (e.g., water leaks, carbon monoxide or fire alarms), the hub should buffer the events locally and deliver when the telemetry suppression ends.

Finding 5: *Devices are less likely to buffer when both the on-demand and always-responsive subsystems are suppressed.* Attacking the physical channel is a coarse-grained approach that reduces the transparency of the attack as it suppresses both the on-demand and always-responsive subsystems. In our experiments, physical layer suppression frequently resulted in smart phone alerts and in-app status changes, indicating connectivity losses at a higher degree than fine-grained network layer suppression. While physical layer suppression lacks transparency, we discovered that devices were less likely to buffer content when the physical channel was suppressed. In particular, the Canary and D-Link cameras both buffered content under network layer suppression but failed to buffer the same content under physical layer suppression. Without a connection to the wireless network, the devices failed to buffer any triggered motion alerts until they regained connectivity. Thus an attacker who valued attack permanence over transparency would benefit from physical layer suppression.

5.5 Potential Countermeasures

This section discusses defenses against sensor blinding and state confusing attacks introduced in the previous sections and reserves future work. Several countermeasures exist to defend against our proposed attack vector. However, a full evaluation of countermeasures deserves a separate work with an emphasis on user studies to understand the transparency of the attack vector and the impact on the usability of devices.

Traffic Shaping: Traffic shaping solutions offer the ability to obscure on-demand activities from an attacker by manipulating traffic. Without the ability to isolate and identify the on-demand system, the adversary's attack is marginalized if not entirely defeated. Traffic shaping has been widely studied as a countermeasure to securing HTTPS from leaking privacy-sensitive information [LL06; Wri09]. Traffic shaping presents a solution as encryption is not enough to protect user privacy for IoT devices. The packet size and frequency of encrypted network traffic between an IoT device and the cloud is enough to reveal device-level activity [Aca18; Apt17]. Our proposed attack vector relies on the ability to infer packets containing on-demand notifications and content. Contemporary works have proposed methods for padding IoT device packets. Apthorpe et al. [Apt18] developed stochastic traffic padding to obfuscate user level activities. Further, Malekzadeh et al. [Mal18] proposed Replacement AutoEncoder, a novel algorithm that transforms and masks the features of sensitive data. However, these methods do not address the discriminative nature of packet timings that reveal the always-responsive subsystem. Previous traffic shaping works have proposed random timing delays to protect privacy-sensitive disclosure of a channel [SW06]. However, this solution requires further study, as the always-responsive subsystem relies on the predictable arrival of periodic heartbeat messages.

Per-IoT Virtual Private Networks: Virtual Private Networks (VPNs) offer the ability to defend against an attacker that manipulates traffic at the border of our residential network. However, they do not protect against the model of a locally compromised wireless router. In this case, the IoT device itself would need to establish the VPN to benefit from the security and privacy it provides. This approach is similar to the per-app VPN offered for iPhones since iOS 9 [App18]. Per-IoT virtual private networks would provide partial protection from an attacker inferring traffic activities and selectively suppressing the on-demand subsystem of IoT. However, VPN encapsulation would still need to incorporate traffic shaping to prevent discovery on subsystems based on size or timing. While applying per-IoT VPNs can guarantee security by placing a layer of abstraction over the on-demand and always-responsive subsystems, that guarantee comes at a performance cost. Further study can examine the impact on the quality of the on-demand system content and the user perception of device responsiveness.

Secure IoT Design: An IoT device's battery constraints, limited storage, and small processing capacity present challenges for secure software design development. However, as we discussed in Finding 4, they are often falsely used to motivate the challenges of IoT. As we analyzed in our findings, design flaws contributed to the severity of the attack by enabling transparency and permanence of our attacks. IoT firmware often isolates the on-demand and always responsive subsystems, segregating their functionality on different applications that establish distinct connections. Applications that implement network heartbeats must unify this segregation and gain awareness of the on-demand subsystem state. Further, the on-demand subsystem must enforce repudiation ensuring that content and notifications are acknowledged by the remote back-end application. When feasible, the on-demand system must buffer notifications and content. As identified in Finding 4, IoT devices require priority buffer scheme to preserve preferential content and notifications. Periodic re-synchronization of sensor state is a low bandwidth solution that can mitigate the effects of sensor blinding and state confusion. However, periodic re-synchronization does not address the case of battery-constrained devices that are unable to benefit this approach. Finally, devices must select short-term timeouts that balance overwhelming the user while providing a transparent view of connectivity. All of these decisions impact the usability of the device and require further user study.

WSN and Embedded Security Approaches: The field of Wireless Sensor Networks (WSNs) offers insight for securing on-demand sensor reporting [Bas13]. As WSN nodes have little provisions for security, they are vulnerable to attacks that compromise the integrity and availability of their reports. Roy et al. [Roy14] proposed an attack-resilient computation algorithm that minimizes communication while verifying node integrity. Based on this approach, we consider introducing message counters into the always-responsive subsystem that maintain the on-demand subsystem state. Sciancalepore et al. [Sci18] proposed a distributed protocol suitable for memory constrained devices, that guaranteed message delivery under jamming by using decoy messages. Based on their work, we hypothesize that on-demand decoy messages can determine the presence of blinding or confusing attacks. Further, the field of *trusted scheduling* for embedded systems offers another approach. Masti et al. [Mas12] proposed trusted scheduling to ensure execution of safety-critical applications for embedded devices. From this insight, we consider introducing a subsystem-dependent telemetry model. This approach would delay delivery of always-responsive messages until on-demand messages were acknowledged and repudiated. All of these proposed approaches require further user study.

5.6 Related Work

Protecting IoT devices and the privacy-sensitive information they produce is an emerging field of computer security. To this end, there have been several surveys that holistically examined the security and privacy of IoT [Sik18; Yas18; Zha17]. In the closest related work, Obermaier and Hutle [OH16] examined the authentication and encryption schemes of cloud-based video system surveillance. However, their work narrowly focused on the implementation for four specific camera models. Wu and Lagasse [WL19] demonstrated an isolated case of the predictive nature of IoT by training a neural network to detect a single hidden wireless camera and classify video recording traffic. Wood et al. [Woo17] narrowly focused on the implementation immaturity of four medical devices, including one device that leaked sensitive health information in clear-text traffic. While these works demonstrate the implementation immaturity and predictive nature of IoT, they offer little insight against a broader nature as we have demonstrated in our work.

Concurrent work has provided threat modeling for IoT device sensors. Chen et al. [Che18b] examined spoofing attacks that confused sensors and caused traffic congestion to the U.S. Department of Transportation traffic control system. Uluagac et al. [Ulu14] analyzed sensory channel threats for cyber physical systems. Their work focuses on protecting the integrity of the sensory channel (e.g., light, temperature, infrared) to prevent adversarial use as a component of an attack. Lakshminarayana et al. [Lak18] studied the impact of signal jamming against communication based train control systems and developed counter measures to limit the attacks' impact. These works provide insight into specific IoT deployments. However, we investigate problems in device telemetry, battery conservation, and implementation maturity that span several smart home IoT classes.

5.7 Conclusion

In this work, we hypothesized that the NEST and Amazon Key incidents are not isolated occurrences, but rather indicatives of a larger systemic design flaw in many IoT devices. This paper explored the design flaws in smart home IoT devices that facilitate sensor blinding and state confusion attacks. We have shown the telemetry and messaging protocols of smart home IoT devices commonly isolate the always-responsive and on-demand subsystems, enabling these attacks. To demonstrate the broad scope of the problem, we have profiled 24 popular smart home IoT devices and measured the severity of these attacks by analyzing their transparency and permanence. We uncover that 22 of 24 studied devices suffer from critical design flaws that (1) enable attacks to transparently disrupt the reporting of device status alerts or (2) prevent the uploading of content integral to the device's core functionality. Further, we examined the impact and feasibility of several countermeasures including traffic shaping, per-IoT VPNs, and secure design approaches.

CHAPTER

6 FUTURE WORK AND CONCLUSION

6.1 Future Work

This thesis set out to investigate the ability to leverage the host and application behavior context to enable fine-grained access control and policy enforcement. In the following section, we explore directions for future research.

Blind-Host Assumption Tainting: Our work in Chapter 3 did not address the case of hosts that cannot participate in defense due to resource constraints. As discussed in Chapter 4, IoT and other embedded devices can be leveraged as part of a stepping-stone attack. We assume that when a host is unable to participate in the defense, the controller would be able to use probabilistic tainting. Our initial results offer promising results into probabilistic tainting. We would like to explore the feasibility of probabilistic tainting to construct our network information flow control graph. Further, we believe we can use a supervised machine-learning (ML) classifier to generate prediction scores for the probabilistic tainting model. Such an approach would extend the work of Chapter 3 to defends against cases where an attacker pivots through a resource-constrained device.

Extending Classification To Cyber Physical Systems: Our analysis in Chapters 4 and 5 was limited to the smart home market of IoT devices since they are readily available. However, Wurm et al. [Wur16] demonstrated that smart home and industrial IoT devices commonly share vulnerabilities. Future research may expand the study of our attack methodology demonstrated in Chapter 5 against other always-on IoT devices, such as medical or Supervisory Control and Data Acquisition (SCADA). Further, we would like to examine the feasibility of traffic classification for Cyber Physical Systems, leveraging the techniques demonstrated in Chapter 4.

Unsupervised Classification of On-Demand Activities: In Chapter 5, we discussed the systemic design decisions and implementation immaturity that allow an adversary to blind sensors and confuse the state of smart home devices. Our attack methodology relied on a network-snooping attacker that manipulated traffic. While our approach relied on a specific understanding of device telemetry, future work should empirically investigate an unsupervised ML classification of the on-demand and always-responsive subsystems. This work can benefit from existing ML approaches for IoT traffic classification [Mie17; Bez18; Aca18]. Future work could also investigate the feasibility of suppressing telemetry through selective wireless jamming. This approach can leverage existing ML models for traffic classification in the presence of wireless encryption [RK16; Sch17].

6.2 Conclusion

In this thesis, we hypothesized that access control could leverage host and application behavior analysis to provide the context to address the challenges of an increasing complexity of networks, opaqueness in applications behaviors, and perpetual cloud connection. We proved this hypothesis by (1) extending information flow tracking from hosts through the network, (2) classifying encrypted and proprietary application layer traffic of IoT devices, and (3) analyzing the impact of access control given the heterogeneity of buffering and state management approaches for IoT devices.

In the first work, we presented PivotWall, a new network security architecture that implements information flow control in a distributed environment. Our solution leveraged the logically central placement of the SDN Controller to extend host-based information flow tracking into the network. We evaluated our prototype by testing against a broad coverage of stealthy attack tools, examined the performance impacts, and demonstrated the ability to provide unique response capabilities to real-world attacks. Our evaluation demonstrated that PivotWall detects a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. Further, we demonstrated that PivotWall incurs minimal impact on network throughput and latency for benign (i.e, untainted) traffic. We demonstrated that PivotWall presents a feasible approach to enable context-based response and prevention capabilities that would be difficult to realize with existing solutions. With the ability to detect stealthy attacks and implement reactive measures with acceptable impacts to performance, PivotWall offers a promising solution for enhancing the security of enterprise networks.

In the second work, we presented HomeSnitch, a building block for enhancing smart home transparency and control by classifying IoT device communication by behavior. HomeSnitch uses supervised learning to classify features from connection-oriented application data unit exchanges that represent application-layer dialog between clients and servers. We implemented HomeSnitch on a commodity wireless router and built upon SDN primitives to enforce network access controls. We demonstrated that by selecting destination and content-agnostic features, HomeSnitch can classify device behaviors even in the presence of encryption and proprietary protocols. Using a Random Forest Classifier, we achieved over 99% accuracy for classifying behaviors from an independent dataset. Further, we deployed HomeSnitch in a realistic home environment and empirically evaluated its ability to accurately classify known behaviors as well as discover new behaviors. Through these efforts, we demonstrated the effectiveness of network-level services to classify behaviors and enforce control on resource-constrained IoT devices.

In the third work, we presented our study of the vendor design decisions surrounding IoT telemetry messaging protocols, specifically, the behaviors taken when an IoT device loses connectivity. We hypothesized and evaluated sensor blinding and state confusion attacks, measuring their effectiveness against an array of smart home IoT device types. We demonstrated that the telemetry and messaging protocols of smart home IoT devices commonly isolate the always-responsive and on-demand subsystems, enabling these attacks. We demonstrated the broad scope of the problem. Our work profiled 24 popular smart home IoT devices and measured the severity of these attacks by analyzing their transparency and permanence. We revleaded that 22 of 24 studied devices suffered from critical design flaws that (1) enabled attacks to transparently disrupt the reporting of device status alerts or (2) prevented the uploading of content integral to the device's core functionality. Through this work, we answered critical questions that govern access control policy and mediation for smarthome devices.

In conclusion, our work provides a new approach for network access control and policy enforcement by leveraging context from the host and analysis of application behaviors. Our work motivates future examination and analysis in this area of context-aware access control, and offers insight about the direction of network defense solutions.

BIBLIOGRAPHY

[Aba14] Abaid, Z. et al. “MalwareMonitor: An SDN-based Framework for Securing Large Networks”. *International Conference on emerging Networking EXperiments (CoNEXT)*. ACM, 2014, pp. 40–42.

[Abe16] Abera, T. et al. “Invited - Things, Trouble, Trust: On Building Trust in IoT Systems”. *Annual Design Automation Conference (DAC)*. Austin, Texas: ACM, 2016, 121:1–121:6.

[Aca18] Acar, A. et al. *Peek-a-Boo: I see your smart home activities, even encrypted!* https://arxiv.org/pdf/1808.02741.pdf. 2018.

[Alr19] Alrawi, O. et al. “SoK: Security Evaluation of Home-Based IoT Deployments”. *IEEE Security and Privacy*. San Francisco, CA: IEEE, 2019.

[AZH09] Alshammari, R. & Zincir-Heywood, A. N. “Machine Learning Based Encrypted Traffic Classification: Identifying SSH and Skype”. *International Conference on Computational Intelligence for Security and Defense Applications (CISDA)*. Ottawa, Ontario, Canada: IEEE, 2009, pp. 289–296.

[AZH11] Alshammari, R. & Zincir-Heywood, A. N. “Can encrypted traffic be identified without port numbers, IP addresses and payload inspection?” *Computer networks*. Vol. 55. 6. Elsevier, 2011, pp. 1326–1350.

[Ama16] Amar, P. *Data Exfiltration Toolkit (DET)*. https://github.com/sensepost/DET. 2016.

[Anb18] Anbazhagan, V. S. et al. *Service for developing dialog-driven applications*. US Patent App. 15/360,814. 2018.

[AS13] Antonenko, V. & Smelyanskiy, R. “Global network modelling based on mininet approach.” *SIGCOMM workshop on Hot topics in software defined networking*. ACM. 2013, pp. 145–146.

[App18] Apple Computers, Inc. *iOS 12 Security Guide iOS*. https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf. Version 12. 2018.

[Apt17] Apthorpe, N. et al. *Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic*. http://arxiv.org/abs/1708.05044. 2017.

[Apt18] Apthorpe, N. et al. *Keeping the Smart Home Private with Smart (er) IoT Traffic Shaping*. https://arxiv.org/pdf/1812.00955.pdf. 2018.

[AF04] Ayyorgun, S. & Feng, W. C. “A Deterministic Definition of Burstiness For Network Traffic Characterization”. *International Conference on Computer Communications and Networks*. 2004.

[Bac14] Bacon, J. et al. “Information flow control for secure cloud computing”. *Transactions on Network and Service Management*. Vol. 11. 1. IEEE, 2014, pp. 76–89.

[Bar17] Barrera, D. et al. *IDIoT: Securing the Internet of Things like it’s 1994*. https://arxiv.org/abs/1712.03623. 2017.

[Bas13] Basagni, S. et al. *Mobile ad hoc networking: Cutting edge directions*. Vol. 35. Hoboken, NJ: John Wiley & Sons, 2013.

[Bat14] Bates, A. et al. “Let SDN be your eyes: Secure forensics in data center networks”. *NDSS workshop on security of emerging network technologies (SENT)*. 2014.

[Bat15] Bates, A. et al. “Trustworthy whole-system provenance for the Linux kernel”. *USENIX Security Symposium*. 2015, pp. 319–334.

[BL73] Bell, D. E. & LaPadula, L. J. *Secure computer systems: Mathematical foundations*. Tech. rep. Bedford, MA: MITRE Corp, 1973.

[Bel03] Bellovin, S. *RFC 3514: The Security Flag in the IPv4 Header*. https://rfc-editor.org/rfc/rfc3514.txt. 2003.

[Ber14a] Berde, P. et al. “ONOS: towards an open, distributed SDN OS”. *Workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[Ber14b] Berman, M. et al. “GENI: A federated testbed for innovative network experiments”. *Computer Networks*. Vol. 61. Elsevier, 2014, pp. 5–23.

[Bez18] Bezawada, B. et al. “Behavioral Fingerprinting of IoT Devices”. *Workshop on Attacks and Solutions in Hardware Security (ASHES)*. Toronto, Canada: ACM, 2018, pp. 41–50.

[Bil16] Bilton, N. *Nest Thermostat Glitch Leaves Users in the Cold*. https://nyti.ms/1Or0eH0. 2016.

[Bit18] Bitdefender. *BOX 2: The new way to secure your connected home*. https://www.bitdefender.com/box/. 2018.

[Bow16] Bowes, R. *DNSCat2: A DNS tunnel*. https://github.com/iagox86/dnscat2. 2016.

[Bow18] Bowles, N. *Thermostats, Locks and Lights: Digital Tools of Domestic Abuse*. https://nyti.ms/2KdGgVC. 2018.

[Bre01] Breiman, L. “Random forests”. *Machine learning*. Vol. 45. 1. Springer, 2001, pp. 5–32.

[Bui13] Buitinck, L. et al. “API design for machine learning software: experiences from the scikit-learn project”. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[Cas07] Casado, M. et al. “Ethane: Taking control of the enterprise”. *SIGCOMM Computer Communication Review*. Vol. 37. 4. ACM. 2007, pp. 1–12.

[CC18] Celosia, G. & Cunche, M. “Detecting Smartphone State Changes Through a Bluetooth Based Timing Attack”. *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. Stockholm, Sweden: ACM, 2018, pp. 154–159.

[Che18a] Chen, H. et al. “Quantifying the security effectiveness of firewalls and DMZs”. *Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. ACM. 2018, p. 9.

[Che18b] Chen, Q. A. et al. “Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control”. *Network and Distributed Systems Security Symposium (NDSS)*. San Diego, CA: Internet Society, 2018, pp. 1–15.

[Cle18] CleverLoop. *Smart Home Security System: FAQs Before Purchase*. https://www.cleverloop.com/faqs.html. 2018.

[CM07] Coskun, B. & Memon, N. “Efficient detection of delay-constrained relay nodes”. *Annual Computer Security Applications Conference (ACSAC)*. AMC, 2007, pp. 353–362.

[Dae96] Daemon9. *Project Loki: ICMP Tunneling*. http://phrack.org/issues/49/1.html. 1996.

[Dai12] Dainotti, A. et al. “Issues and future directions in traffic classification”. *IEEE Network*. Vol. 26. 1. IEEE, 2012.

[de 14] de Oliveira, R. L. S. et al. “Using Mininet for emulation and prototyping Software-Defined Networks”. *Colombian Conference on Communications and Computing (COLCOM)*. IEEE, 2014, pp. 1–6.

[Den76] Denning, D. E. “A lattice model of secure information flow”. *Communications of the ACM*. Vol. 19. 5. ACM, 1976, pp. 236–243.

[Dha15] Dhanjani, N. *Abusing the internet of things: blackouts, freakouts, and stakeouts*. O’Reilly Media, Inc., 2015.

[Dio18] Diop, A. et al. “Questioning the security and efficiency of the ESIoT approach”. *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. ACM. 2018, pp. 202–207.

[d’O19] d’Otreppe, T. *Aircrack-ng*. https://www.aircrack-ng.org. 2019.

[Dug16] Dugan, J. et al. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr. 2016.

[EG17] Elmallah, E. S. & Gouda, M. G. “Hardness of firewall analysis”. *Transactions on Dependable and Secure Computing*. Vol. 14. 3. IEEE, 2017, pp. 339–349.

[Enc14] Enck, W. et al. “TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones”. *Transactions on Computer Systems (TOCS).* Vol. 32. 2. ACM, 2014, p. 5.

[Eri13] Erickson, D. “The beacon openflow controller”. *SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 13–18.

[Est96] Ester, M. et al. “A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. *International Conference on Knowledge Discovery and Data Mining (KDD).* Vol. 96. 34. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[Fay13] Fayazbakhsh, S. K. et al. “Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions”. *SIGCOMM workshop on Hot topics in software defined networking.* ACM. 2013, pp. 19–24.

[Fea14] Feamster, N. et al. “The road to SDN: an intellectual history of programmable networks”. *SIGCOMM Computer Communication Review.* Vol. 44(2). ACM, 2014, pp. 87–98.

[Fer16] Fernandes, E. et al. “Security Analysis of Emerging Smart Home Applications”. *Symposium on Security and Privacy (S&P).* IEEE. 2016, pp. 636–654.

[Fer95] Ferraiolo, D. et al. “Role-based access control (RBAC): Features and motivations”. *Annual computer security application conference (ACSAC).* 1995, pp. 241–48.

[Fon05] Fonseca, R. et al. *IP Options are not an option.* http://www2.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-24.html. 2005.

[FR18] Fontes, R. & Rothenberg, C. *Mininet-WiFi: Emulator for Software-Defined Wireless Networks.* https://github.com/intrig-unicamp/mininet-wifi. 2018.

[FB17] Fox-Brewster, T. *Here’s How The CIA Allegedly Hacked Samsung Smart TVs – And How To Protect Yourself.* https://www.forbes.com/sites/thomasbrewster/2017/03/07/cia-wikileaks-samsung-smart-tv-hack-security. 2017.

[Fri02] Friedman, J. H. “Stochastic gradient boosting”. Vol. 38. 4. Elsevier, 2002, pp. 367–378.

[Goo18] Google. *Overview of Internet of Things: Data Storage.* https://cloud.google.com/solutions/iot-overview. 2018.

[Gre17a] Greenberg, A. *The Reaper Botnet Has Already Infected a Million Networks.* https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/. 2017.

[Gre17b] Greis, F. *Vault 7: CIA hacking tools revealed.* https://www.wikileaks.org/ciav7p1/. 2017.

[Gud08] Gude, N. et al. “NOX: towards an operating system for networks”. *SIGCOMM Computer Communication Review.* Vol. 38. ACM, 2008, pp. 105–110.

[Haf17] Hafeez, I. et al. “IOTURVA: Securing Device-to-Device (D2D) Communication in IoT Networks”. *Workshop on Challenged Networks (CHANTS)*. Snowbird, Utah: ACM, 2017, pp. 1–6.

[HH16] Hagins, J. & Hawkinson, A. *Distributed control scheme for remote control and monitoring of devices through a data network*. US Patent 9,462,041. 2016.

[Han12] Handigol, N. et al. “Reproducible network experiments using container-based emulation”. *International Conference on Emerging Networking Experiments and Technologies*. ACM. 2012, pp. 253–264.

[HC18] Hanna, J. & Chan, S. *The murder suspect denies it. The victim’s Fitbit tells another story, police says*. https://www.cnn.com/2018/10/04/us/california-fitbit-killing/. 2018.

[Har15] Hart, B. *My SecTor Story: Root Shell on the Belkin WeMo Switch*. https://www.tripwire.com/state-of-security/featured/my-sector-story-root-shell-on-the-belkin-wemo-switch/. 2015.

[Heo14] Heorhiadi, V. et al. “SNIPS: A Software-Defined Approach for Scaling Intrusion Prevention Systems via Offloading”. *Information Systems Security (ISS)*. 2014, pp. 9–29.

[HC07] Hernández-Campos, F. et al. *Modeling and generating TCP application workloads*. Tech. rep. 2007, pp. 280–289.

[Hon17] Honorof, M. *Amazon Cloud Cam, Key Flaws Could Let in Burglars*. https://www.tomsguide.com/us/amazon-key-cloud-cam-hack,news-26132.html. 2017.

[HB11] Houmansadr, A. & Borisov, N. “SWIRL: A Scalable Watermark to Detect Correlated Network Flows.” *Network and Distributed System Security Symposium (NDSS)*. 2011.

[Hou09] Houmansadr, A. et al. “RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows.” *Network and Distributed System Security Symposium (NDSS)*. 2009.

[Ich15] Ichise, H. et al. “Detection method of DNS-based botnet communication using obtained NS record history”. *Computer Software and Applications Conference (COMPSAC)*. Vol. 3. IEEE. 2015, pp. 676–677.

[ICH18] ICHISE, H. et al. “Analysis of DNS TXT Record Usage and Consideration of Botnet Communication Detection”. *IEICE Transactions on Communications*. The Institute of Electronics, Information and Communication Engineers, 2018.

[Inc16] Ince, M. *Data Exfiltration Attacks against Corporate Network*. https://pentest.blog/data-exfiltration-tunneling-attacks-against-corporate-network/. 2016.

[ISI81] ISI. *IETF RFC 793: Transmission Control Protocol (TCP)*. https://www.ietf.org/rfc/rfc793.txt. 1981.

[Jaf12] Jafarian, J. H. et al. “Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking”. *Workshop on Hot Topics in Software Defined Networks (HotSDN)*. Helsinki, Finland: ACM, 2012, pp. 127–132.

[Jia17a] Jia, Y. J. et al. “ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms”. *Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, 2017.

[Jia17b] Jia, Y. J. et al. “ContexloT: Towards Providing Contextual Integrity to Appified IoT Platforms.” *Network and Distributed System Security Symposium (NDSS)*. 2017.

[Jin15] Jin, Y. et al. “Design of detecting botnet communication by monitoring direct outbound DNS queries”. *Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2015, pp. 37–41.

[Joh16] Johnson, R. et al. “Studying Naïve Users and the Insider Threat with SimpleFlow”. *CCS International Workshop on Managing Insider Security Threats (MIST)*. Vienna, Austria: ACM, 2016, pp. 35–46.

[Jov14] Jovic, A. et al. “An overview of free software tools for general data mining”. *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1112–1117.

[Kam14] Kampanakis, P. et al. “SDN-based solutions for Moving Target Defense network protection”. *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2014, pp. 1–6.

[Kar14] Kara, A. M. et al. “Detection of malicious payload distribution channels in DNS”. *International Conference on Communications (ICC)*. IEEE. 2014, pp. 853–858.

[Ker14] Kerns, A. J. et al. “Unmanned aircraft capture and control via GPS spoofing”. *Field Robotics*. Vol. 31. 4. Wiley Online Library, 2014, pp. 617–636.

[Kil03] Kilpatrick, D. et al. *Securing The X Window System With SELinux*. Tech. rep. 2003, pp. 1–33.

[KF13] Kim, H. & Feamster, N. “Improving network management with software defined networking”. *Communications Magazine*. Vol. 51. 2. IEEE, 2013, pp. 114–119.

[Kim08] Kim, H. et al. “Internet traffic classification demystified: myths, caveats, and the best practices”. *Conference on emerging Networking EXperiments (CoNEXT)*. ACM. 2008, p. 11.

[Koh08] Kohei, K. “Security-Enhanced PostgreSQL: System-Wide Consistency in Access Control”. Presentation at the 2008 PostgreSQL Conference. Ottawa, Canada, 2008.

[Lak18] Lakshminarayana, S. et al. “Signal Jamming Attacks Against Communication-Based Train Control: Attack Impact and Countermeasure”. *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. Stockholm, Sweden: ACM, 2018, pp. 160–171.

[Lee13a] Lee, K. H. et al. “High Accuracy Attack Provenance via Binary-based Execution Partition.” *Network and Distributed System Security Symposium (NDSS)*. 2013.

[Lee13b] Lee, K. H. et al. “Loggc: Garbage collecting audit log”. *SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1005–1016.

[Lee10] Lee, K. et al. “Uncovering social spammers: social honeypots+ machine learning”. *SIGIR conference on Research and development in information retrieval*. ACM. 2010, pp. 435–442.

[LL06] Liberatore, M. & Levine, B. N. “Inferring the Source of Encrypted HTTP Connections”. *Conference on Computer and Communications Security (CCS)*. Alexandria, Virginia: ACM, 2006, pp. 255–263.

[Ma16] Ma, S. et al. “ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting”. *Network and Distributed System Security Symposium (NDSS)*. 2016.

[MS15] MacFarland, D. C. & Shue, C. A. “The SDN Shuffle: Creating a Moving-Target Defense Using Host-based Software-Defined Networking”. *Workshop on Moving Target Defense (MTD)*. Denver, Colorado: ACM, 2015, pp. 37–41.

[Mah10] Mahalle, P. et al. “Identity Management Framework towards Internet of Things (IoT): Roadmap and Key Challenges”. *Recent Trends in Network Security and Applications*. Ed. by Meghanathan, N. et al. Springer Berlin Heidelberg, 2010, pp. 430–439.

[Mal18] Malekzadeh, M. et al. “Replacement AutoEncoder: A Privacy-Preserving Algorithm for Sensory Data Analysis”. *Conference on Internet-of-Things Design and Implementation (IoTDI)*. Orlando, FL: IEEE, 2018, pp. 165–176.

[Mas12] Masti, R. J. et al. “Enabling Trusted Scheduling in Embedded Systems”. *Annual Computer Security Applications Conference (ACSAC)*. Orlando, Florida: ACM, 2012, pp. 61–70.

[McH15] McHugh, M. *How NestDoor and Nest Cams are helping cops solve crimes*. https://www.wired.com/2015/12/nextdoor-crime-nest-cams/. 2015.

[McK08] McKeown, N. et al. “OpenFlow: enabling innovation in campus networks”. *SIGCOMM Computer Communication Review*. Vol. 38. 2. ACM, 2008, pp. 69–74.

[Med14] Medved, J. et al. “Opendaylight: Towards a model-driven sdn controller architecture”. *International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 2014.

[Mie17] Miettinen, M. et al. “IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT”. *International Conference on Distributed Computing Systems (ICDCS)*. IEEE. Atlanta, GA: IEEE, 2017, pp. 2177–2184.

[MIT17] MITRE. *CVE-2017-13903: HomeKit Vulnerability*. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13903. 2017.

[Mon18] Montoya, M. et al. “SWARD: A Secure WAke-up RaDio Against Denial-of-Service on IoT Devices”. *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. ACM. Stockholm, Sweden: ACM, 2018, pp. 190–195.

[Moo16] Moon, M. *Software bug forced Nest thermostats offline*. http://engt.co/1Oskqsf. 2016.

[Mor08] Morris, J. “Have You Driven an SELinux Lately?” *Linux Symposium (OLS)*. Vol. 2. Ottawa, Ontario, Canada, 2008, pp. 101–114.

[ML05] Murdoch, S. J. & Lewis, S. “Embedding Covert Channels into TCP/IP”. *International Conference on Information Hiding (IH)*. Barcelona, Spain: Springer-Verlag, 2005, pp. 247–261.

[Mut15] Mutti, S. et al. “SeSQLite: Security Enhanced SQLite: Mandatory Access Control for Android Databases”. *Annual Computer Security Applications Conference (ACSAC)*. Los Angeles, CA: ACM, 2015, pp. 411–420.

[Nay16] Naylor, B. *This Doll May Be Recording What Children Say, Privacy Groups Charge*. https://www.npr.org/sections/alltechconsidered/2016/12/20/506208146/this-doll-may-be-recording-what-children-say-privacy-groups-charge. 2016.

[Nel17] Nelson, S. S. *Germany Bans My Friend CaylaDoll Over Spying Concerns*. https://www.npr.org/2017/02/20/516292295/germany-bans-my-friend-cayla-doll-over-spying-concerns. 2017.

[New18] Newman, L. H. *Turning an Echo Into a Spy Device Only Took Some Clever Coding*. https://www.wired.com/story/amazon-echo-alexa-skill-spying. 2018.

[Ngu18] Nguyen, T. D. et al. *DÏoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices*. https://arxiv.org/pdf/1804.07474.pdf. 2018.

[Nor18] Norton. *Core Secure WiFi Router*. https://us.norton.com/core. 2018.

[OH16] Obermaier, J. & Hutle, M. “Analyzing the Security and Privacy of Cloud-based Video Surveillance Systems”. *International Workshop on IoT Privacy, Trust, and Security (IoTPTS)*. Xi’an, China: ACM, 2016, pp. 22–28.

[ON06] Olshefski, D. & Nieh, J. “Understanding the Management of Client Perceived Response Time”. *Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. Saint Malo, France: ACM, 2006, pp. 240–251.

[Ols04] Olshefski, D. P. et al. “ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams”. *Symposium on Operating System Design and Implementation (OSDI)*. San Francisco, California: USENIX Association, 2004, pp. 333–346.

[ONL15] ONL. *Pox Controller Wiki*. https://openflow.stanfofrd.edu/display/ONL/POX+Wiki. 2015.

[Ope15] Open Networking Foundation. *OpenFlow Switch Specification, Version 1.5.1.* https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf. 2015.

[Pa15] Pa, Y. M. P. et al. “IoTPOT: Analysing the Rise of IoT Compromises”. *Conference on Offensive Technologies (WOOT).* Washington, D.C.: USENIX Association, 2015, pp. 9–9.

[PP12] Papagiannis, I. & Pietzuch, P. “CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud”. *Workshop on Cloud Computing Security Workshop (CCSW).* Raleigh, North Carolina: ACM, 2012, pp. 97–102.

[Pap13] Pappas, V. et al. “CloudFence: Data flow tracking as a cloud service”. *International Workshop on Recent Advances in Intrusion Detection.* Springer. 2013, pp. 411–431.

[Ped11] Pedregosa, F. et al. “Scikit-learn: Machine Learning in Python”. *J. Mach. Learn. Res.* Vol. 12. JMLR.org, 2011, pp. 2825–2830.

[Pen05] Peng, P. et al. “Active timing-based correlation of perturbed traffic flows with chaff packets”. *International Conference on Distributed Computing Systems Workshops.* IEEE, 2005, pp. 107–113.

[Pet03] Peterson, L. et al. “A blueprint for introducing disruptive technology into the Internet”. *SIGCOMM Computer Communication Review.* Vol. 33. 1. ACM, 2003, pp. 59–64.

[Por15] Porras, P. A. et al. “Securing the Software Defined Network Control Layer.” *Network and Distributed System Security Symposium (NDSS).* 2015.

[RE11] R. Enns M. Bjorklund, J. S.J. S. *RFC6241: Network Configuration Protocol (NETCONF).* https://tools.ietf.org/html/rfc6241. 2011.

[Ram08] Ramachandran, A. et al. *Packets with provenance.* Tech. rep. Georgia Institute of Technology, 2008.

[Ram09] Ramachandran, A. et al. “Securing enterprise networks using traffic tainting”. *Georgia Inst. Technol., Tech. Rep. GTCS-09-15.* Atlanta, GA, 2009.

[RK16] Reed, A. & Klimkowski, B. *Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections.* 2016.

[RK17] Reed, A. & Kranch, M. “Identifying HTTPS-Protected Netflix Videos in Real-Time”. *Conference on Data and Application Security and Privacy (CODASPY).* Scottsdale, Arizona: ACM, 2017, pp. 361–368.

[Roe99] Roesch, M. et al. “Snort: Lightweight intrusion detection for networks.” *Large Installation System Administration Conference (LISA).* Vol. 99. 1. 1999, pp. 229–238.

[Roy14] Roy, S. et al. “Secure data aggregation in wireless sensor networks: Filtering out the attacker’s impact”. *Transactions on Information Forensics and Security*. Vol. 9. 4. IEEE, 2014, pp. 681–694.

[Sad15] Sadeghi, A. R. et al. “Security and privacy challenges in industrial Internet of Things”. *Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6.

[Sch12] Schindler, H. R. et al. *Examining Europe’s Policy Options to Foster Development of the ’Internet of Things’*. http://www.rand.org/pubs/research_reports/RR356.html. 2012.

[Sch17] Schuster, R. et al. “Beauty and the burst: Remote identification of encrypted video streams”. *USENIX Security*. 2017.

[Sci18] Sciancalepore, S. et al. “Strength of Crowd (SOC) Defeating a Reactive Jammer in IoT with Decoy Messages”. *Sensors*. Vol. 18. 10. Multidisciplinary Digital Publishing Institute, 2018, p. 3492.

[Sha16] Sharma, P. et al. “BotMAD: Botnet malicious activity detector based on DNS traffic analysis”. *International Conference on Next Generation Computing Technologies (NGCT)*. IEEE. 2016, pp. 824–830.

[Shi13a] Shin, S. et al. “FRESCO: Modular Composable Security Services for Software-Defined Networks”. *Network and Distributed System Security Symposium (NDSS)*. 2013.

[Shi13b] Shin, S. et al. “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks”. *SIGSAC Conference on Computer & Communications Security (CCS)*. Berlin, Germany: ACM, 2013, pp. 413–424.

[SW06] Shmatikov, V. & Wang, M.-H. “Timing analysis in low-latency mix networks: Attacks and defenses”. *European Symposium on Research in Computer Security*. Springer. 2006, pp. 18–33.

[Shu11] Shullich, R. et al. “A survey of research in stepping-stone detection”. *Electronic Commerce Studies*. 2011.

[Sib17] Siby, S. et al. “IoTScanner: Detecting Privacy Threats in IoT Neighborhoods”. *International Workshop on IoT Privacy, Trust, and Security (IoTPTS)*. Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 23–30.

[Sik17] Sikder, A. K. et al. “6thsense: A context-aware sensor-based attack detector for smart devices”. *USENIX Security*. Vancouver, BC, Canada, 2017.

[Sik18] Sikder, A. K. et al. *A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications*. https://arxiv.org/pdf/1802.02041.pdf. 2018.

[Siv17] Sivanathan, A. et al. “Characterizing and classifying IoT traffic in smart cities and campuses”. *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2017, pp. 559–564.

[Siv16] Sivaraman, V. et al. “Smart-Phones Attacking Smart-Homes”. *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. Darmstadt, Germany: ACM, 2016, pp. 195–200.

[Sol18] Soltan, S. et al. “BlackIoT: IoT Botnet of high wattage devices can disrupt the power grid”. *Proc. USENIX Security*. Vol. 18. Baltimore, MD: USENIX, 2018, pp. 15–32.

[Son14] Song, Y. et al. “Network Iron Curtain: Hide Enterprise Networks with OpenFlow”. *Information Security Applications (ISA)*. 2014.

[Sto15] Storm, D. “MEDJACK: Hackers hijacking medical devices to create backdoors in hospital networks”. *Computerworld*. Vol. 8. 2015.

[SE17] Stroetmann, u. & Esser, T. *Reverse Engineering the TP-Link HS110*. https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/. 2017.

[Sun16] Sun, Y. et al. “Pileus: Protecting User Resources from Vulnerable Cloud Services”. *Annual Conference on Computer Security Applications (ACSAC)*. Los Angeles, California: ACM, 2016, pp. 52–64.

[Ter09] Terrell, J. et al. “Passive, Streaming Inference of TCP Connection Structure for Network Server Management”. *Traffic Monitoring and Analysis*. Ed. by Papadopouli, M. et al. Springer Berlin Heidelberg, 2009, pp. 42–53.

[Tur19] Turris. *Omnia: More than just a router*. https://omnia.turris.cz/en/. 2019.

[Ulu14] Uluagac, A. S. et al. “Sensory channel threats to Cyber Physical Systems: A wake-up call”. *Conference on Communications and Network Security*. San Francisco, CA: IEEE, 2014, pp. 301–309.

[Ur16] Ur, B. et al. “Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes”. *Conference on Human Factors in Computing Systems (CHI)*. San Jose, California: ACM, 2016, pp. 3227–3231.

[VP14] Vanhoef, M. & Piessens, F. “Advanced Wi-Fi Attacks Using Commodity Hardware”. *Annual Computer Security Applications Conference (ACSAC)*. New Orleans, Louisiana: ACM, 2014, pp. 256–265.

[VP15] Vanhoef, M. & Piessens, F. “All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS.” *USENIX Security Symposium*. 2015, pp. 97–112.

[VP16] Vanhoef, M. & Piessens, F. “Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys.” *USENIX Security Symposium*. 2016, pp. 673–688.

[VP17] Vanhoef, M. & Piessens, F. “Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2”. *SIGSAC Conference on Computer and Communications Security (CCS)*. Dallas, Texas: ACM, 2017, pp. 1313–1328.

[VP18] Vanhoef, M. & Piessens, F. “Release the Kraken: New KRACKs in the 802.11 Standard”. *SIGSAC Conference on Computer and Communications Security (CCS)*. Toronto, Canada: ACM, 2018, pp. 299–314.

[Vor17] Voronkov, A. et al. “Systematic Literature Review on Usability of Firewall Configuration”. *Computing Surveys (CSUR)*. Vol. 50. 6. ACM, 2017, p. 87.

[Wan13] Wang, S. et al. “EstiNet openflow network simulator and emulator”. *IEEE Communications Magazine*. Vol. 51. 9. 2013, pp. 110–117.

[WR03] Wang, X. & Reeves, D. S. “Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays”. *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2003, pp. 20–29.

[Wan02] Wang, X. et al. “Inter-packet delay based correlation for tracing encrypted connections through stepping stones”. *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2002, pp. 244–263.

[Wan05a] Wang, X. et al. “Robust network-based attack attribution through probabilistic watermarking of packet flows”. TR-2005-10. 2005.

[Wan05b] Wang, X. et al. “Tracking anonymous peer-to-peer VoIP calls on the internet”. *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2005, pp. 81–91.

[Wil11] Wilhelm, M. et al. “Short Paper: Reactive Jamming in Wireless Networks: How Realistic is the Threat?” *Proceedings of the Fourth ACM Conference on Wireless Network Security*. WiSec ’11. Hamburg, Germany: ACM, 2011, pp. 47–52.

[Wil06] Williams, N. et al. “A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification”. *SIGCOMM Computer Communication Review*. Vol. 36. 5. ACM, 2006, pp. 5–16.

[Woo17] Wood, D. et al. “Cleartext Data Transmissions in Consumer IoT Medical Devices”. *Workshop on Internet of Things Security and Privacy (IoT S&P)*. Dallas, Texas: ACM, 2017, pp. 7–12.

[Woo04] Wool, A. “A Quantitative Study of Firewall Configuration Errors”. *Computer*. Vol. 37. 6. IEEE Computer Society Press, 2004, pp. 62–67.

[Wri09] Wright, C. V. et al. “Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis.” *Network and Distributed System Security Symposium (NDSS)*. Vol. 9. San Diego, CA: Internet Society, 2009.

[WL19] Wu, K. & Lagesse, B. “Do You See What I See? Detecting Hidden Streaming Cameras Through Similarity of Simultaneous Observation”. *Pervasive Computing and Communications (Percom)*. Kyoto, Japan: IEEE, 2019.

[Wur16] Wurm, J. et al. “Security analysis on consumer and industrial IoT devices”. *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016, pp. 519–524.

[YT19] Yamamoto Takashi Fujita Tomonori, I. Y. *RYU SDN: Component-based software defined networking framework*. http://osrg.github.com/ryu/. 2019.

[Yas18] Yasuura, H. et al. *Smart sensors at the IoT frontier*. Springer, 2018.

[Yeg04] Yegneswaran, V. et al. “Global Intrusion Detection in the DOMINO Overlay System”. *Network and Distributed System Security Symposium (NDSS)*. 2004.

[Yu15] Yu, T. et al. “Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things”. *Workshop on Hot Topics in Networks (HotNets)*. Philadelphia, PA: ACM, 2015, 5:1–5:7.

[Yu17] Yu, T. et al. “PSI: Precise Security Instrumentation for Enterprise Networks”. *Network and Distributed System Security Symposium (NDSS)*. 2017.

[YT05] Yuan, E. & Tong, J. “Attributed based access control (ABAC) for web services”. *International Conference on Web Services (ICWS)*. IEEE. 2005.

[Zan07] Zander, S. et al. “A survey of covert channels and countermeasures in computer network protocols”. *Communications Surveys & Tutorials*. Vol. 9. 3. IEEE, 2007, pp. 44–57.

[Zel08] Zeldovich, N. et al. “Securing Distributed Systems with Information Flow Control”. *Symposium on Networked Systems Design and Implementation (NSDI)*. San Francisco, California: USENIX Association, 2008, pp. 293–308.

[Zha17] Zhang, N. et al. *Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be*. http://arxiv.org/abs/1703.09809. 2017.

[Zha18] Zhang, W. et al. “HoMonit: Monitoring Smart Home Apps from Encrypted Traffic”. *SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 1074–1088.

[Zha16] Zhang, X. et al. “Deploying IoT devices to make buildings smart: Performance evaluation and deployment experience”. *World Forum on Internet of Things (WF-IoT)*. IEEE. 2016, pp. 530–535.

[ZP00] Zhang, Y. & Paxson, V. “Detecting Stepping Stones.” *USENIX Security Symposium*. Vol. 171. 2000, p. 184.

APPENDICES

APPENDIX

A ADDITIONAL HOMESNITCH EXPERIMENTS

A.1 Digital Voice Assistant Classification

A key contribution of HomeSnitch is the ability to accurately identify unknown and new behaviors. In this section, we demonstrate HomeSnitch's ability to identify a new device behavior of an Alexa-enabled digital voice assistant. This study also highlights the relatively brief period required to train the model.

We observed the behavior of an Echo Dot voice assistant for nine hours in our HomeSnitch environment. We clustered behaviors using the unsupervised clustering described in Chapter 4. Of note, we used destination-agnostic feature selection from application data units to shape the clusters. Echo Dot behaviors consisted of 103 unique flows, 10% (103/1,021) of the overall flows of the network and 15% (4/26) of the behavior clusters identified by our unsupervised clustering approach. Figure A.1 depicts the classification of behaviors during the entire experiment. The most frequent Echo Dot behavior during this time was an internet connectivity check via HTTP to *spectrum.s3.amazonaws.com*. The next most frequent was an encrypted analytic session to *device-metrics-us.amazon.com*. The least frequent activity was a plaintext time synchronization to *kindle-time.amazon.com.* After unsupervised clustering, we conducted a cross-validation scoring using our supervised learning model to check for overfitting and

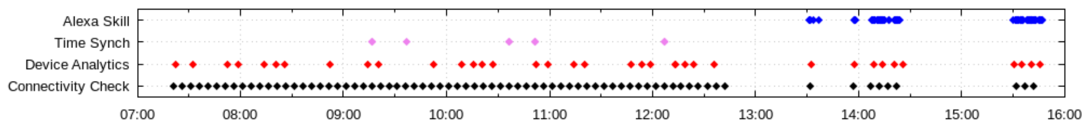


Figure A.1 Echo Dot behavior classification results

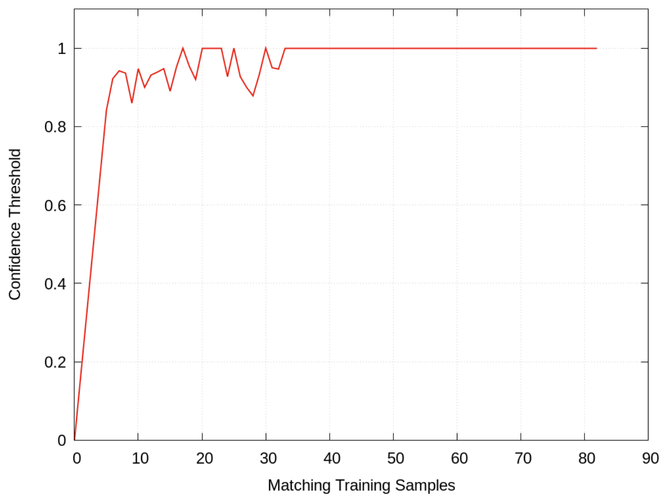
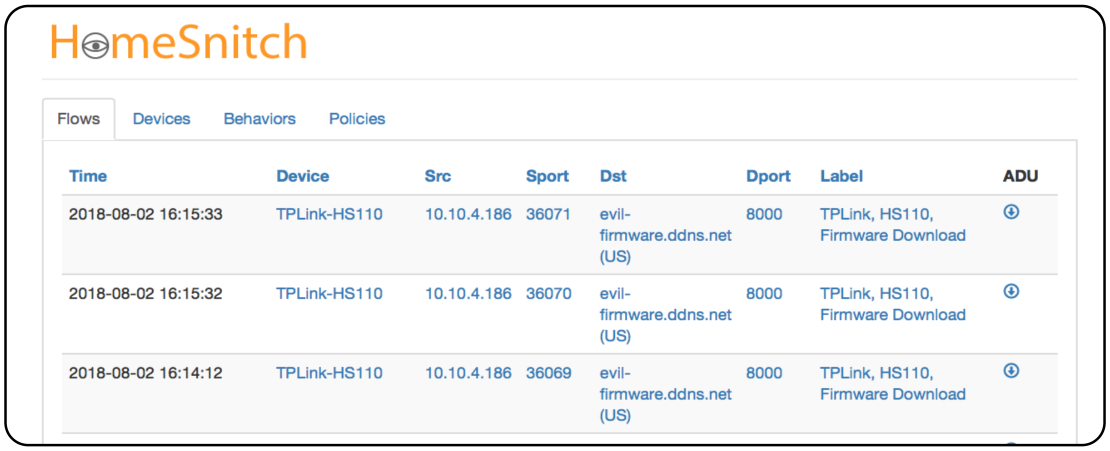


Figure A.2 ADU samples vs. threshold for Alexa skill

determined the cross validation score for all 1,021 flows as 97.979 ± .004 %.

To represent a new behavior, we installed an Alexa skill via the Amazon Web-based platform for the Echo Dot. Alexa *skills* extend the usability of the Alexa-enabled voice assistants by exposing the API to developers. We selected the top-rated skill, *Peaceful Meditation*, which allows a user to ask Alexa to plays peaceful music. We then recorded over 80 interactions asking Alexa to launch the skill. We sampled an initial five flows associated with Alexa skills and added them directly to the model. After the initial sampling, we predicted each subsequent interaction and recorded the prediction confidence associated with each identified flow. Figure A.2 demonstrates the results of this experiment, where the prediction confidence matures after 30 samples are placed into the model. Figure A.2 further strengthens our argument for HomeSnitch's key contribution to classify behaviors and identify new behaviors. With the ability to correctly classify behaviors, we next examine the functionality for HomeSnitch to implement policies via SDN primitives.



Time	Device	Src	Sport	Dst	Dport	Label	ADU
2018-08-02 16:15:33	TPLink-HS110	10.10.4.186	36071	evil-firmware.ddns.net (US)	8000	TPLink, HS110, Firmware Download	⊕
2018-08-02 16:15:32	TPLink-HS110	10.10.4.186	36070	evil-firmware.ddns.net (US)	8000	TPLink, HS110, Firmware Download	⊕
2018-08-02 16:14:12	TPLink-HS110	10.10.4.186	36069	evil-firmware.ddns.net (US)	8000	TPLink, HS110, Firmware Download	⊕

Figure A.3 Example classification of malicious firmware

A.2 Firmware Classification

Next, we evaluated HomeSnitch's ability to classify a malicious behavior or attack by simulating an attack of a vulnerable smart plug. The TP-Link HS110 is a cloud-enabled smart plug that an end user can control via an app. By interacting with the smart plug, the app can monitor energy consumption and remotely schedule actions. Stroetmann and Esser revelaed that the app to device communication consists of a human-readable JSON protocol encrypted only with an XOR cipher [SE17]. We hosted a malicious copy of the device firmware on a malicious server (i.e., *evil-firmware.ddns.net*). Further, we modified Stroetmann and Esser's work to build a tool that forced the device to repeatedly download the malicious firmware. Next, we trained HomeSnitch to recognize a firmware download to the device by observing repeated downloads of benign firmware.

Figure A.3 depicts the corresponding results on the HomeSnitch web application. HomeSnitch identified each individual attempts to download the firmware and subsequently logged the activity. This alert offers insight to users that their device has become compromised by downloading firmware from a malicious server. HomeSnitch provides the ability to train the system and write rules to prevent against such a download (i.e., by white-listing downloads to only the official *tp-link.com* site). However, we envision that a service provider (similar to virus or threat reporting) could provide both behavior training signatures and rules for enforcing behaviors for specific devices.

APPENDIX

B
OVERVIEW OF BLINDING AND STATE CONFUSION ATTACKS

B.1 Taxonomy of Attacks

Chapter 5 evaluates the security impact of suppressing smart home device telemetry. In our work, we evaluate the attacker's ability to control the user perception of device functionality. Figure B.1 provides an illustration of key terms we use to evaluate the severity of the attack. Transparent attacks provide no indication to the end user. Permanent attacks eliminate the uploading of buffered content after the attack. Sensor blinding attacks the integrity and availability of sensor devices by preventing the delivery of sensory measurements to always-connected IoT servers. State confusion, a special subset of sensor blinding attack, violates the integrity of actuator devices' state reported to always-connected IoT servers.

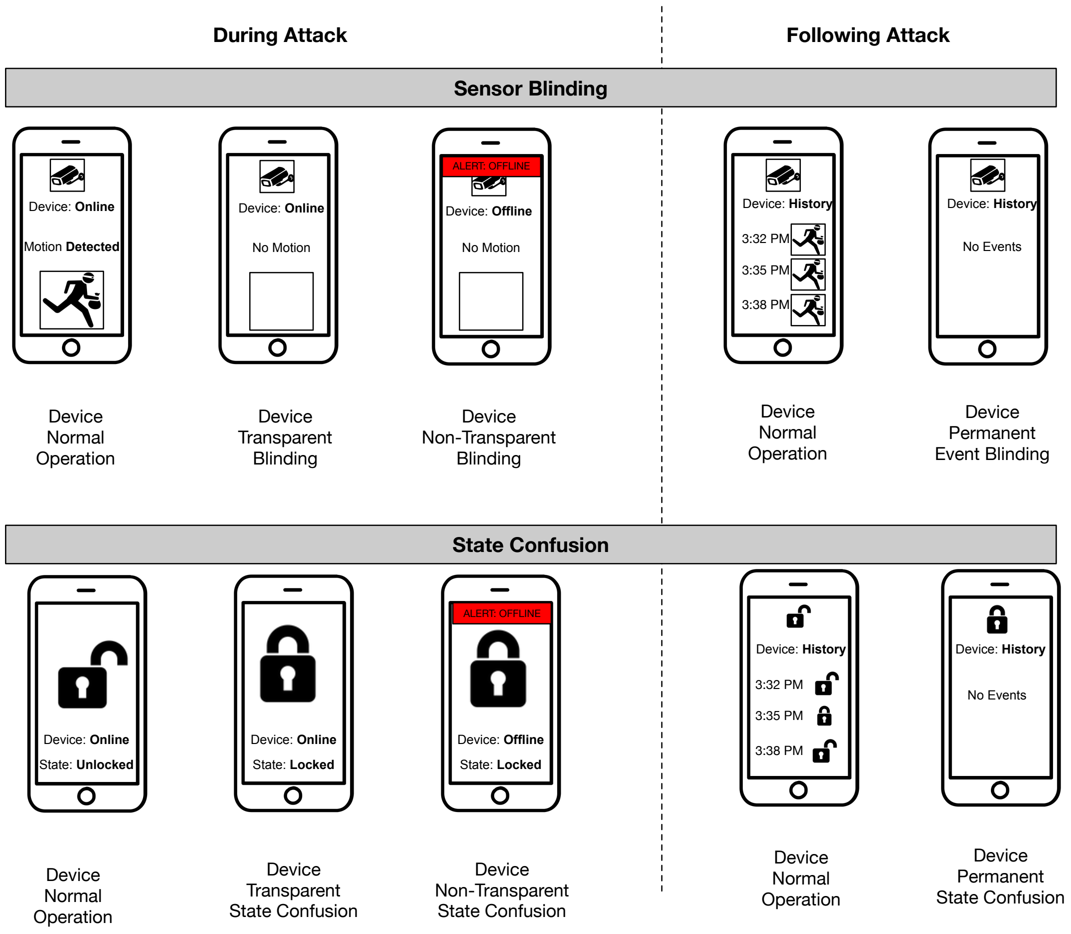


Figure B.1 Taxonomy of device blinding and state confusion attacks.