RADAR: A Robust Behavioral Anomaly Detection for IoT Devices in Enterprise Networks

Tianlong Yu, Yuqiong Sun, Susanta Nanda, Vyas Sekar, Srinivasan Seshan

May 21, 2019

CMU-CyLab-19-003

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

RADAR: A Robust Behavioral Anomaly Detection for IoT Devices in Enterprise Networks

Tianlong Yu[†], Yuqiong Sun[♭], Susanta Nanda[♭], Vyas Sekar[†], Srinivasan Seshan[†]

[†]Carnegie Mellon University, [♭]Symantec

ABSTRACT

IoT devices deployed inside enterprise networks (e.g., routers, storage appliances, cameras) are emerging security threats for enterprises. It is impractical for security administrators to address IoT threats with existing enterprise or smart home security techniques, e.g., host-based or mobile-based detection are not applicable, network firewall rules are too coarse-grained, signature-based detection fails with zero-day attacks, and existing anomaly detection mechanisms are ineffective for IoT devices (e.g., cannot detect IoT backdoor access) as they are proposed for computer activities (e.g., email spear phishing). Fortunately, we observe that unlike general-purpose computing devices, the normal behavior of an IoT device is limited (e.g., a camera has zooming-in, video streaming and audio recording behaviors). Based on this insight, we revisit *behavioral anomaly detection* at the network layer. Designing such a system is challenging on two fronts. First, we need a behavior model to abstract the key characteristics of IoT-specific behaviors (e.g., commands or arguments used) from network traffic. Second, in practical enterprise environment, the network traces for learning normal behavior models are unlabeled and potentially polluted. We address these challenges in designing RADAR, a practical and robust behavioral anomaly detection system for enterprise IoT devices. We design a novel learning mechanism that can build benign behavior models (finite-state-machines) for IoT devices, from unlabeled and potentially polluted network traces. We show that our approach achieves high detection accuracy (F-Score improved by 5X comparing with other approaches) and is robust to polluted behavior samples (F-Score>0.9 when 15% of the network traffic of IoT devices is polluted).

KEYWORDS

IoT security; behavioral anomaly detection; robust learning

1 INTRODUCTION

As Internet-of-Things (IoT) devices proliferate [7], a significant and growing majority of these devices (78.8% [9]) are deployed inside enterprise networks. These enterprise IoT devices include both infrastructure devices (such as routers, switches and network attached storages) and consumer devices (such as cameras, DVRs, smart lights, fire alarms, sensor gateways and smart speakers). Unfortunately, enterprise IoT devices often have numerous security flaws and have become severe security threats to enterprise networks [13].

These enterprise IoT devices pose several unique challenges for existing security solutions that include host-based defenses[16, 18], mobile-application-based defenses[40, 57], signature-based defenses[51, 53], network access control, and existing anomaly detection mechanisms[29, 31, 43–45, 52, 56]. The scale and diversity of enterprise IoT deployments make it hard for security administrators to correctly configure ACLs (access control lists), e.g., blocking the access to a NAS would impact legitimate users. The management APIs of some enterprise IoT devices (e.g., snmp client or smb client) are different from common management APIs of smart home IoT devices (e.g. mobile apps), so mobile-application-based defenses [40, 57] are not applicable here. In an enterprise environment, attackers are intended to adopt more stealthy techniques; consequently, common attacks against enterprise IoTs, such as code injection and authentication bypass, are designed to easily evade flow-statistics-based anomaly detection [43] as well as anomaly detection proposed for specialized computer activities (e.g., email spear phishing or malware C&C) [29, 31, 44, 45, 52, 56]. Finally, host-based defenses[16, 18] or signature-based network defenses [51, 53] are not applicable due to the fact that IoT devices have limited resources and that zero-day attacks are common for these devices.

We argue that, network-based behavioral anomaly detection tailored for IoT devices would offer a pragmatic path for security administrators. Our key insight is that an IoT device serves single-purpose functionality in the enterprise setting, and is likely to exhibit a limited set of high-level behaviors under normal operation. When an attacker tricks the device into performing a malicious operation – for example, using a printer to exfiltrate a file as opposed to printing it – this action would appear as an anomaly.

Building on this insight, we develop a *behavioral anomaly detection* system to detect IoT-related malicious activities in enterprise networks. This is challenging on two key fronts:

Modeling IoT behaviors from network traffic: Even though an IoT device exhibits a limited set of high-level behaviors (e.g., camera's video recording behavior), there is a gap between a high-level behavior and the network traffic structures associated with it (e.g., the sequence of http URIs used for the camera's video recording behavior). It is non-trivial to bridge the gap and define a network behavior model to abstract key characteristics of a device's high-level behaviors (e.g., commands or arguments used, sequential semantics and directionality) from network traffic.

Availability of labeled behavior samples: Labeled behavior samples (network traces) of the IoT devices are required for accurately learning the behavior model. However, such labeled data is often difficult to obtain in a practical enterprise setup [55]. Attack samples that cover all possible attacks are difficult to obtain because zero-day attacks are common for enterprise IoTs. Benign samples are also hard to obtain, as generating all possible benign

network traces for every IoT device deployed is too costly for administrators if not impossible. A common practice is to directly install the devices in production environment and then passively monitor their behaviors. The network traces collected in this case is prone to pollution (i.e., the network traces may already contain attack samples). Therefore, a robust learning approach that can handle the unlabeled, potentially polluted samples is required in operational enterprise environments.

In this paper, we present RADAR (Robust behavioral Anomaly Detection for enterprise IoT devices), which tackles these challenges and detects behavior anomalies based on benign behavior models learned from unlabeled, potentially polluted network traffic. RADAR contains two components – a passive *learning* component and an online *detection* component. The learning component passively monitors the network traffic of the enterprise IoT devices, and build FSM (finite-state machine) models to *precisely* abstract their behaviors (Section 4). To learn the behavior model from unlabeled network traces, RADAR provides a FSM inference mechanism based on the characteristics of IoT traffic (Section 5). Then, to address the challenge of pollution, RADAR provides an outlier detection mechanism to generate multiple candidate models, and leverages several IoT heuristics to filter out polluted models and selects the best benign model from the candidate models (Section 5). After the benign behavior model is obtained, the detection component checks if the network traffic of a device is consistent with the benign FSM model and raises alerts when anomalies are detected.

To evaluate RADAR, we setup a sandbox IoT environment with 9 classes of IoTs in an organization. We conducted an IRB-approved study, and obtained more than 10000 benign behavior samples from 10 human users. We also built an enterprise IoT penetration test bundle and obtained more than 1000 attack samples. Benign behavior samples and attack behavior samples are randomly mixed to generate the unlabeled samples for evaluation. We show that our system provides accurate detection and lower FP and FN (F-Score>0.9 and F-Score improved by 5X) when comparing with the best detection results by previous anomaly detections adapted to enterprise IoTs, including two clustering-based anomaly detection (Antonakakis *et al.*[28] and Perdisci *et al.*[52]) and the two protocol FSM-based anomaly detection (Prospex[34] and Suricata[26]). We show that the FSM model effectively detects common IoT attacks including code injection, authentication bypass and unrestricted outbound access. We also demonstrate that our approach is robust to polluted behavior samples and yields low FP and FN (F-Score>0.9) when the fraction of polluted samples ranges from 1% to 15%.

2 BACKGROUND AND MOTIVATION

In this section, we investigate the prevalence of IoT devices deployed in enterprise networks, their security issues and why existing security solutions fail to address them.

Prevalence of enterprise IoT devices: IoT devices deployed in enterprise networks are *special-purpose* computing systems. Typical enterprise IoT devices include both infrastructure devices (such as routers, switches and network attached storage (NAS)) and consumer devices (such as cameras, DVRs, smart lights, fire alarms, sensors & sensor gateways, smart speakers, smart TVs, smart thermostats, smart plugs, and smart refrigerators). To investigate the common types of such devices, we leverage Censys[3] and

Device Class	Total Number
Router	162931
Network Switch	9503
NAS	328503
Camera	12890
DVR	12645
Smart Light	28083
Fire Alarm	104693
Sensor Gateway	20502
Smart Speaker	58
Smart TV	1209
Thermostat	316
Smart Plug	283
Refrigerator	195

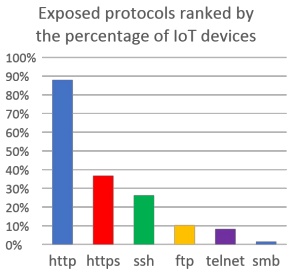


Figure 1: Different classes of IoT devices in enterprises and their network protocols exposed.

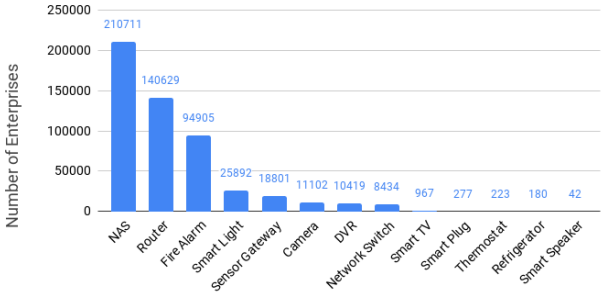


Figure 2: Common IoT devices in enterprises.

SHODAN[25] to search for these IoT devices deployed inside enterprise networks. Specifically, we use keywords such as "tags:nas" to find device of each class, and keywords such as "inc", "llc", "company" and "corporation" on the WHOIS records to search for devices inside enterprise. To avoid including IoT devices in smart homes, we filter out common ISPs providing residential internet connectivity[15].

As shown in Figure 1, the number of the IoT devices deployed inside the enterprises is large[1]. In addition, Figure 1 shows that these devices expose multiple sensitive protocols that can be accessed and exploited by attackers, including http, https, ssh, ftp, telnet and smb. Surprisingly, unencrypted protocols including telnet, http and ftp are widely used by enterprise IoT devices. For example, 88% of the enterprise IoT devices use unencrypted http protocol, as shown in Figure 1. Therefore, monitoring the unencrypted traffic alone is important for IoT devices in enterprise networks.

Then, to investigate what IoT devices are commonly used in enterprise, in Figure 2, we rank these IoT devices by the number of enterprises that have them deployed. We use the organization field in the WHOIS records to distinguish enterprises. From Figure 2, we can see that 8 classes of IoT devices, namely, NASes, routers, fire alarms, smart lights, sensor gateways, cameras, DVRs and network switches, all have been deployed in more than 1K enterprises. In this paper, we also consider the smart speakers (Amazon Alexa and Google Mini) due to their growing deployment in enterprises/organizations (especially in conference rooms and hotels[1]). Therefore, we consider these 9 classes of IoT devices as the commonly used devices in this paper.

Security issues for IoT devices in enterprise: As shown by previous work on firmware analysis [32, 35, 36, 54] and by reports

[1] Noting that the result here only includes IoT devices that are exposed to the internet, and the actual number of all enterprise IoT devices is likely to be even higher.

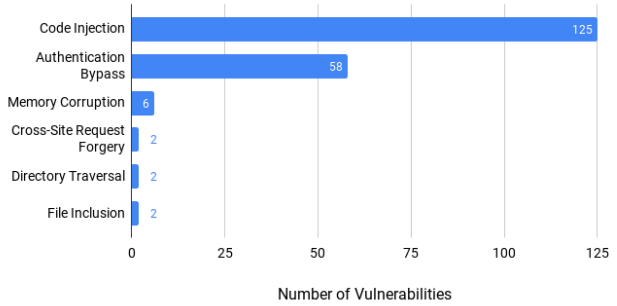


Figure 3: Different vulnerabilities for commonly used enterprise IoT devices from 1999 to 2018 from NVD[19].

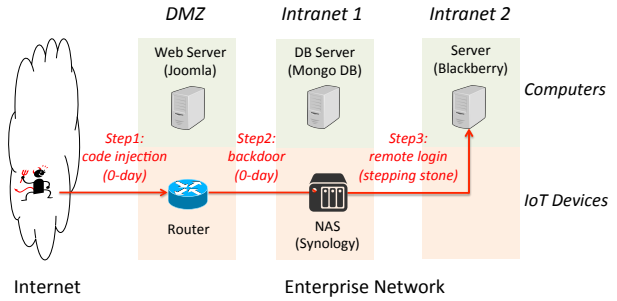


Figure 4: The famous HackingTeam incident to show how attackers can intrude the enterprise via IoT devices.

of security incidents [4, 5, 10], existing enterprise IoT devices have several security issues:

- IoT devices often have flaws when processing commands/arguments, which allows the attacker to inject code via the flawed commands or arguments [4, 5, 32, 35, 36].
- Many IoT devices contain flawed authentication routines (backdoors) [54], which allow the attacker to bypass authentication.
- Since the IoT devices are deployed inside the enterprise networks, they often have unrestricted access to other IoT devices or computers, which allows the attacker to use the compromised IoT devices as stepping stones [10].

To identify common vulnerabilities for enterprise IoT deployments, we select a few devices to represent each IoT device class and analyzed their vulnerabilities. Figure 3 shows the number of different types of vulnerabilities reported between 1999 to 2018 in the National Vulnerability Database[19] for Cisco IOS router/switch, FreeNAS, D-Link camera, HiSilicion DVR, Philips Hue smart light, OpenWRT fire alarm, LoRA sensor gateway, Alexa Echo Dot2 and Google Mini smart speakers. Code injection and authentication bypass are the primary source of vulnerabilities. We also consider the unrestricted outbound access, which are commonly seen in incidents after the IoT devices are compromised [10].

A case study of an actual attack: We describe a realistic attack scenario to highlight the above security issues and why existing security mechanisms, including host-based detection [16, 18], signature-based detection [51, 53], ACLs and network-based anomaly detection [29, 31, 43–45, 52, 56], are impractical for administrators.

Figure 4 depicts a real-world security incident (a IoT-based intrusion against the HackingTeam - a famous hacker organization[10]). In step 1, the attacker found a zero-day flaw and compromised the router using code injection. In step 2, after compromising the router, the attacker obtained access to the NAS via a backdoor (authentication bypass). In step 3, he used the NAS to remotely login to the Blackberry Server and exfiltrated valuable data.

Now we discuss why current security solutions fail to address above threats:

- Host-based detection [16, 18] is impractical because IoT devices have limited resources to support any host-based detection. Therefore, in step 1, the attacker chose to attack the router rather than the webserver [10].
- Signature-based network detection [51, 53] is ineffective because zero-day attacks are common (because of bad development practice). For example, in step 1, attacker easily found a zero-day code injection vulnerability within two weeks and evaded the signature-based detection between the Internet and the DMZ zone.
- Network-layer access control (ACLs) fails because of the scale and diversity of the enterprise IoT devices. In step 2, the administrators cannot simply block the access from the DMZ to the NAS because the NAS was used to backup some servers in DMZ.
- Current network anomaly detection is easy to evade because they either focus on low-level traffic statistics [43] or only model a specific type of attack behaviors for generic computers (e.g., malware C&C, port scanning or email spear phishing) [29, 31, 44, 45, 52, 56]. In step 3, the attacker performs remote login at a low frequency to avoid causing flow-statistics anomalies. Also, the remote login behavior is a legitimate computer behavior (although it is abnormal for a NAS), so the anomaly detection specialized for computer activities[29, 31, 44, 45, 52, 56] will not raise any alert.

3 SYSTEM OVERVIEW

In this section, we present an overview of RADAR. Our design focuses on detecting the *IoT-related malicious activities* including code injection, authentication bypass, and the unrestricted outbound access, based on the security issues of enterprise IoTs described in Section 2.

Threat Model: The goal of the attacker is to attack the enterprise network by compromising devices, exfiltrating data, or disrupting services offered by IoT devices.

Specifically, we make the following assumptions:

- The attacker can inject code via the flawed commands/arguments or bypass the authentication (e.g., using backdoor) to compromise the enterprise IoT devices, exfiltrate data, or disrupt services of enterprise IoT devices.
- The attacker can leverage the unrestricted outbound access of the IoT devices to attack other IoT devices or computers in the enterprise network.
- The attacker cannot bypass (e.g., via covert channels) or compromise the detection component.
- The attacker cannot bypass the interception techniques used by the enterprise admins that enforces inspection on encrypted traffic (e.g., Google and other companies use various TLS proxy solutions, such as Google Beyondcorp Framework[8], to inspect encrypted traffic).

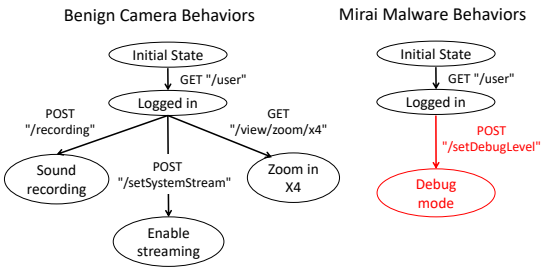


Figure 5: The limited benign behaviors of a camera and the attack behaviors (Mirai Malware[27]) against it.

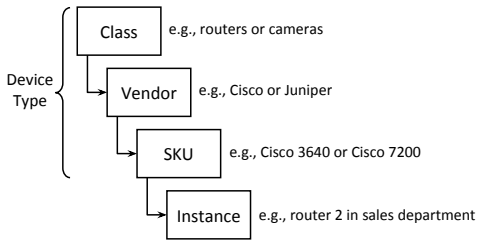


Figure 6: Different granularities to categorize IoTs.

Intuition behind RADAR: Our key insight is that each enterprise IoT device is typically used for a limited number of specialized use cases. Therefore, each device has a limited set of behaviors even if the network protocols it employs can support a wide range of behaviors. To perform attacks, attackers are likely to deviate from the specialized use cases, so attacks will be logically different from the limited device behaviors. For example, Figure 5 shows the difference between the limited benign behaviors of a camera and the Mirai malware[27] against it. The benign behaviors include logging in, recording sound, enable streaming and zoom in. The attack behavior, however, includes sending a HTTP POST message with URI "/setDebugLevel" to force the camera to debug mode (so the attacker can change the camera's firmware).

Thus, we posit that detecting *behavioral anomalies* can help us identify IoT-related malicious activities.

Problem Formulation: At a high-level, the problem is to learn a network behavior model for IoT devices from historical network traffic, and then use the model to detect IoT attacks in real-time.

The first thing we need to define is the *granularity* of the model, i.e., the aggregated set of IoT devices sharing the same model. As shown in Figure 6, IoTs can be categorized at different granularities, including the device class (class of functionality, e.g., router or camera), the vendor of the device (e.g., Cisco or Juniper), SKUs (Stock Keeping Unit, a unique identifier for each distinct product) and instances (the physical device deployed in a certain environment). We define the *device type* as the combination of device class, vendor and SKU (e.g. router of Cisco 3640). Formally, let $device_{t,i}$ denote the device instance number i of type t (e.g., Cisco 3640 router with instance ID 2) in the enterprise. Now we introduce a *granularity* notion g, and the aggregated set of IoTs sharing the same model can be denoted as $\{device_{t,i}\}_g$. The administrator can specify the granularity g to be a device instance, a device SKU or a device type.

Next, we explain the structure of the IoT network traffic that we are modeling. Suppose IoT devices of granularity g, denoted as $\{device_{t,i}\}_g$, support a number of possible protocols $\{p\}$ (e.g., http, ftp, telnet) to communicate with other IoT devices or computers. For each protocol p, we can logically consider a notion of a application protocol *session*, defined as a group of TCP (or UDP) connections executing a specific task. A *session* consists of a *series* of Application-layer Data Unit (ADUs). An ADU is defined as all the application-layer data in a network flow. A network flow is all the packets in one direction in a tcp/udp connection. So an ADU is a consecutive chunk of application data, which spans one or more packets in one direction in a connection. The connections can be grouped to sessions by previous approaches [46, 51] using timing information or protocol specification. In this paper, we rely on Bro IDS [51] to identify session structure. We assume that the network protocol formats can be obtained from RFCs or by existing protocol reverse engineering mechanisms [37]. We assume that the traffic is either unencrypted inside the enterprise (e.g., http, telnet or DNS traffic) or other enterprise security mechanisms (e.g., TLS proxy [2]) are employed to observe encrypted traffic (e.g., Google BeyondCorp [8]).

In this paper, the problem we address is: Given a *historical trace* $Trace_{g,p}$ consists of multiple sessions $\{SN_{g,p,l}\}_l$ (l is session index) for a set of aggregated device instances $\{device_{t,i}\}_g$ and protocol p, we need to learn a behavioral model to identify if a future session $SN_{g,p,l}$ is anomalous. We focus on homogeneous sessions that consist of consecutive connections within the same protocol and leave anomalies that span multiple protocols for future work.

System Overview: Figure 7 shows the overview of the RADAR system with two stages:

Offline Learning: For a set of aggregated IoT devices $\{device_{t,i}\}_g$ with granularity g and protocol p, we split $Trace_{g,p}$ into application sessions $\{SN_l\}_l$ and further subdivide each session SN_l into its constituent ADU series $\{A_l\}$. Then, given the unlabeled $\{A_l\}$, RADAR provides a robust learning mechanism to build a FSM model $FSM_{g,p}^{benign}$ to abstract the benign behaviors of the aggregated IoT devices. We will explain why RADAR uses a FSM model in Section 4.

Online Detection: In the online stage, we take each completed session $SN_{g,p,l}$ in real-time and again parse it into ADU series $A_{g,p,l}$ and then compute a real-time FSM model $FSM_{g,p}^{mix}$ from ADU series $A_{g,p,l}$ and historical FSM model $FSM_{g,p}^{benign}$. Then we calculate the similarity (defined in Section 5) between $FSM_{g,p}^{mix}$ and $FSM_{g,p}^{benign}$. If the similarity is less than a similarity threshold dt, we raise an alert on session $SN_{g,p,l}$ and $A_{g,p,l}$. We will show that our approach is robust to the configuration of threshold dt in Section 7 and Figure 19.

In the above workflow, RADAR is designed to build an aggregated behavior model $FSM_{g,p}^{benign}$ with a granularity notion g. This design provides a good coverage of possible benign behaviors and makes it easy for RADAR to scale in the enterprise setting. To find out why, let's consider a fine-grained behavior model at per device instance per protocol granularity (e.g., a FSM behavior model for the HTTP traffic of camera instance 1). A single device instance (camera instance 1) may not have sufficient historical traces to cover possible benign behaviors (so the FP will be high), and is hard to scale as the number of device instances is large. By building an aggregated behavior model, RADAR can use more historical traces to better cover possible benign behaviors. To better scale in the

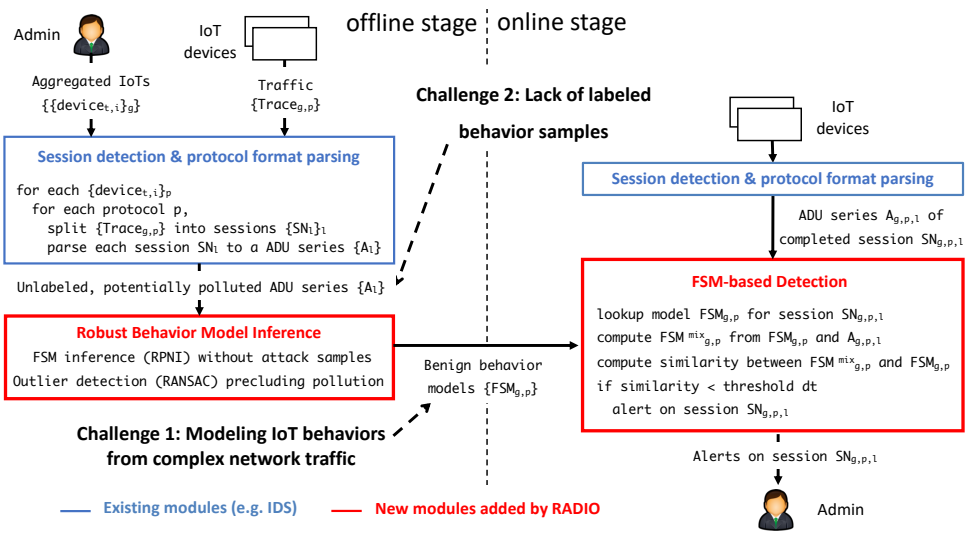


Figure 7: The system architecture of RADAR.

operational setting of an enterprise, RADAR only needs to process the historical traces of a few device instances (e.g., of the same device type) to build an aggregated behavior model in the offline stage, and apply the aggregated model to other device instances (e.g., of the same type) in the online stage. In Section 7, we will show that, by building aggregated behavior model from only a few device instances, RADAR can provide good behavior coverage and scalability.

Challenges: To build RADAR, there are two key challenges.

Challenge 1. Modeling IoT behaviors from network traffic: From our explanation of the structure of IoT network traffic, it is obvious that the limited set of high-level IoT behaviors are concealed in complex network traffic patterns (e.g., the ADU series). It is hard to decide how to abstract the key characteristics of the complex network traffic patterns (e.g., the sequential semantics of ADU series) to represent the high-level IoT behaviors. Previous network anomaly detection approaches [26, 29, 31, 34, 43, 43, 45, 52, 56] focusing on flow statistics or malicious computer behaviors (e.g., email spear phishing) are not applicable for this task.

Challenge 2. Lack of labeled behavior samples: As stated in Section 1, it is difficult to obtain clean and labeled samples in practical enterprise setting [55]. Attack behavior samples are hard to obtain as zero-day attacks are common for enterprise IoT devices. Benign behavior samples are hard to obtain as the management cost to generate all possible benign samples are high considering the scale and diversity of enterprise IoT devices. Passive monitoring also does not solve the problem as it is prone to pollution. Therefore, RADAR has to provide a robust learning mechanism that can learn benign behavior models from unlabeled, potentially polluted behavior samples (unlabeled ADU series $\{A_I\}$ in Figure 7).

We address challenge 1 by defining a behavior model that can *precisely* abstract IoT behaviors in Section 4. We address challenge 2 by designing a robust learning mechanism for enterprise IoT devices in Section 5.

4 NETWORK BEHAVIOR MODEL FOR IOTS

Given the *historical network traces*, we need a network behavior model to abstract the key characteristics of IoT-specific behaviors.

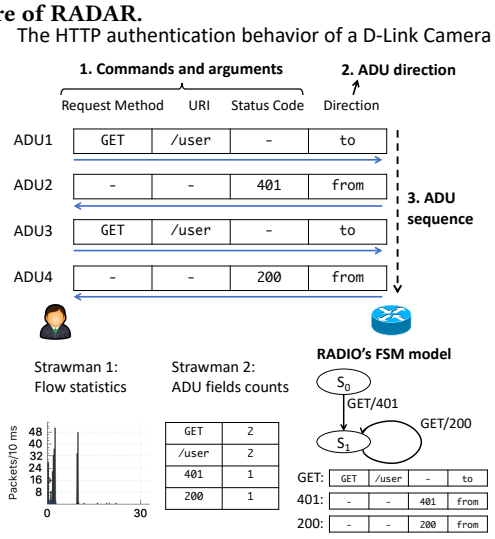


Figure 8: The HTTP authentication behavior of a D-Link camera and two modeling strawmans.

Formally, let $Trace_{g,p}$ denote the *historical trace* of protocol p from a set of aggregated devices $\{device_{t,i}\}_g$, our goal is to define a behavior model to abstract the behaviors of the aggregated devices. In Figure 8, we illustrate the modeling process with a D-Link camera's HTTP authentication behavior.

The HTTP authentication behavior includes 4 ADUs. In ADU1, the user is trying to access the camera with the request method `GET` and URI `/user`. Then in ADU2, the camera notifies the user to authenticate by using status code `401`. Then user authenticates using ADU3. The camera notifies the user that the authentication is successful using the status code `200` in ADU4. Based on this example, there are three key aspects of the behavior: 1) the commands and arguments included in each ADU (e.g., the the request method, URI and status code); 2) the direction of each ADU, i.e., whether the ADU is sent `to` or `from` the IoT device; 3) the sequence of the ADUs, e.g., the sequence of ADU1 to ADU4 indicates the authentication routine of the camera.

Now, we can think of some seemingly natural strawman solutions for modeling the IoT behaviors. One natural starting point (Figure 8, Strawman 1) may be to use the coarse-grained flow statistics such as packet counts [43]. However, the flow statistics do not capture IoT's commands and arguments at application-level (e.g., URI). Alternatively, we can try to build a bag-of-words model (Figure 8, Strawman 2) that counts the appearance of different ADUs field values. However, the bag-of-words model cannot capture the directionality or sequential semantics (e.g., normal authentication for this camera have status code `401` before status code `200`, otherwise it is backdoor access).

To overcome the limitations of the strawman solutions, we suggest the following network behavior model. Instead of flow-level statistics, we choose to abstract application-level ADU fields [2] that contain the device-function-specific commands and arguments for enterprise IoTs [3]. To capture the directional semantics, we explicitly include the direction of the ADU (from device or to device) to model the interaction between the user and the device (e.g., HTTP GET from admin to router). Finally, to capture the sequential semantics of a session, we define a FSM model (Finite State Machine) to capture the sequential pattern in the ADU series.

To capture both ADU fields and ADU direction, we define an *abstracted ADU* m as the combination of a set of ADU fields $\{f_1, ..., f_v\}$ and an ADU direction d_m. The ADU direction d_m captures whether the ADU is sent from the device or to the device. For example, the *abstracted ADU* for ADU 1 in Figure 8 is $\{GET, /user, -, to\}$.

To represent the sequence of ADUs in a compact way, we abstract the sequence of ADUs using a FSM model. We define the FSM model as a Mealy Machine [4]. A Mealy Machine FSM is a six tuple $< S, s_0, \Sigma_I, \Sigma_O, \delta, \lambda >$, where S is a finite non-empty set of states, $s_0 \in S$ is the initial state, Σ_I is a finite set of input symbols,Σ_O is a finite set of output symbols, $\delta : S \times \Sigma_I \rightarrow S$ is the transition relation, $\lambda : S \times \Sigma_I \rightarrow \Sigma_O$ is the output relation. For example, for the camera's HTTP authentication behaviors in Figure 8, the transition relation at S_0 is $\{S_0\} \times \{GET\} \rightarrow \{S_1\}$, and the output relation at S_0 is $\{S_0\} \times \{GET\} \rightarrow \{401\}$. In RADAR, the Mealy Machines are deterministic, which means at any given state (e.g. S_0), one input (e.g. *GET*) only yields one output (e.g. 401).

To define the transition relation δ and output relation λ, we leverage the *causality* between a request ADU and a response ADU, i.e., at a given state of the FSM (e.g. S_0), a specific request ADU (e.g. *GET*) will result in a specific response ADU (e.g. 401). Therefore, we define the transition relation as $\delta : S \times \{m_{request}\} \rightarrow S$ and the output relation $\lambda : S \times \{m_{request}\} \rightarrow \{m_{response}\}$, where m is the *abstracted ADU*.

However, the above definition cannot handle general protocols that may not have request and response ADU pairs (e.g., Telnet). This is because the above definition relies on the *causality* between request and response (i.e., a certain request result in a certain response). To abstract protocols without request and response ADU pairs, our idea is to also leverage the *causality* between the

[2]An ADU field is defined as a sequence of lines of characters with special syntax defined by the protocol format.

[3]We can identify these ADU fields by existing protocol reverse engineering mechanisms[37].

[4]Another alternative is Moore Machine. We did not choose Moore Machine as it has "rejecting" states that relies on attack samples (as counter examples) to generate.

Abstracted CMP Messages

ID	Subset $\{m_a\}$	Subset $\{m_b\}$
	Direction+Command	Subcommand
1	from + Will	Echo
2	from + Do	Suppress Go Ahead
3	to + Suboption	Terminal Type
4	to + Do	Negotiate About
5	to + Will	Environment Option
6	from + Suboption	Remote FlowControl
7		Linemode
8		New Environment Option
9		Status
10		End

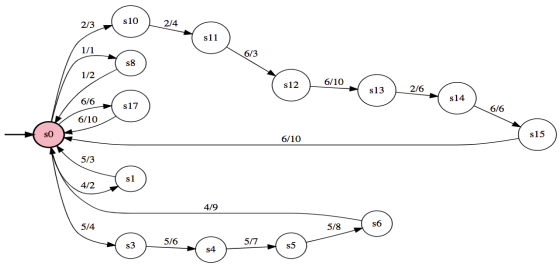


Figure 9: RADAR's FSM model that represents a Cisco router's behaviors under Cluster Management Protocol.

ADU*fields* inside each ADU, i.e., the occurrence of some ADU fields result in the occurrence of some other ADU fields in an ADU. For example, a typical protocol without request and response ADU pairs, called Cisco Cluster Management Protocol (CMP) used by Cisco routers and switches, is demonstrated in Figure 9. As shown in the table, the occurrence of some ADU fields (such as `Suboption` command) will result in the occurrence of some other ADU fields (such as `Terminal Type` and `Remote FlowControl`). Based on this idea, an abstracted ADU m can be split into two subsets of ADU fields $m_a = \{f_a\}$ and $m_b = \{f_b\}$, e.g. $\{f_a\}$ is the command and $\{f_b\}$ are the combination of its arguments. At a given state of the FSM, a subset of ADU fields $\{f_a\}$ will result in a specific subset of ADU fields $\{f_b\}$. Therefore, for such protocols, we define the transition relation as $\delta : S \times \{m_a\} \rightarrow S$ and the output relation $\lambda : S \times \{m_a\} \rightarrow \{m_b\}$.

Figure 9 illustrates an end-to-end example of how FSM model captures the behavior of Cisco router under Cluster Management Protocol (without request and response ADU pairs). At the top of the figure, the table shows the alphabet for the transitions. The FSM at the bottom captures common router management behaviors such as login, traceroute, and clear router status. For example, a common router management behavior is captured by the transition from s_0 to s_8 in Figure 9. Specifically, the transition relation and the output relation here mean that, at s_0, if a m_{a1} ADU field (a `Will` command) is seen, another m_{b1} ADU field (an `Echo` subcommand) will also be seen, and the state will transit to s_8.

5 LEARNING IOT BEHAVIOR MODEL

Given the *historical traces* from a set of aggregated IoT devices, RADAR will learn a FSM model as defined in Section 4 to abstract the benign behaviors of the aggregated IoT devices. There are two key requirements for RADAR's learning mechanism. First, the learning mechanism should be able to learn the FSM model with unlabeled samples. Second, the learning mechanism should be able to learn the FSM model from polluted *historical traces*. In this section, we describe how we modify the state-of-art FSM inference algorithm to learn a behavior model with unlabeled samples, and how we modify the state-of-art outlier detection algorithm to address the pollution issue.

High level idea: A natural starting point for learning an FSM from the *historical trace* is using the state-of-the-art FSM inference algorithms such as RPNI (Regular Positive and Negative Inference) [50]. Such FSM inference algorithms require both labeled benign samples and labeled attack samples to learn the FSM (Gold et al. [42]), which are hard to obtain in enterprise setting. Therefore, the FSM inference algorithms cannot be applied here. To address this issue, our idea is to remove the requirement for attack samples by using the *causality* in the IoT traffic (e.g., one ADU result in another ADU) to determine the FSM states for unlabeled samples. We modified the state-of-the-art FSM inference algorithm called RPNI[5], and leverage two IoT causality heuristics to learn the behavior model for unlabeled samples.

Note that this does not address the issue of pollution. If the historical trace is unlabeled and contains malicious behaviors, then the FSM model generated will also contain malicious behaviors, thus the FNs will be high. To address this issue, our idea is to novelly intergrate the state-of-the-art outlier detection algorithms (e.g., RANSAC[41]) with FSM inference algorithm RPNI[50] to generate multiple candidate FSM models, and design a novel loss function for RANSAC to select the best benign model from the candidate models. In particular, two popular state-of-the-art outlier detection algorithms are RANSAC[41] and Hough Transformation[39]. The latter is designed for image structures (e.g., image pixels) and is not applicable for IoT FSM models.

Next, we will first walk through the basic RPNI algorithm and show our modifications. Then, we will introduce the basic RANSAC algorithm and discuss how we modifies the RANSAC to address the pollution issue.

Basic RPNI: The basic RPNI algorithm, shown in Figure 10, is designed as follows. In Step 1, benign or "positive" samples are used to build a tree where each sample is a path from root to leaf. In Step 2, basic RPNI merges nodes to compact the tree and uses labeled attack or "negative" samples to prevent merging of nodes that would otherwise lead to false negatives. For example, in the bottom left figure of Figure 10, in Step 2, if S_0 and S_2 are merged, then the counterexample $GET, 200$ (indicating an authentication bypass) will be included. When there are no attack samples, every merge is allowed, and it results in a single state accepting all inputs, and the false negatives will be high.

RADAR's modification for RPNI: We extend this algorithm to generate an FSM model without relying on the labeled samples. We achieve this by replacing the traditional merge with two causality heuristics: (1) at any state, same request ADU yields same response ADU, and (2) at any state, same ADU fields subset m_a yields same message fields subset m_b. We adopt the above heuristics based on the observation that the protocols used by the enterprise IoTs either have strong correlations between their request and response ADUs (e.g. the request method and the status code of HTTP) or have strong correlations between the commands and arguments inside an ADU (e.g. the command and subcommand of Telnet). Based on the protocols we have considered in this paper, we posit that

[5]The state-of-the art FSM inference algorithms include RPNI, Biermann & Feldman's algorithm[30], and DeLeTe2[38]. Biermann & Feldman's algorithm and DeLeTe2 generate non-deterministic FSM, while RPNI generates deterministic FSM. Nondeterministic means it can transition to, and be in, multiple states at once, which is contradicting the deterministic behaviors of enterprise IoT devices.

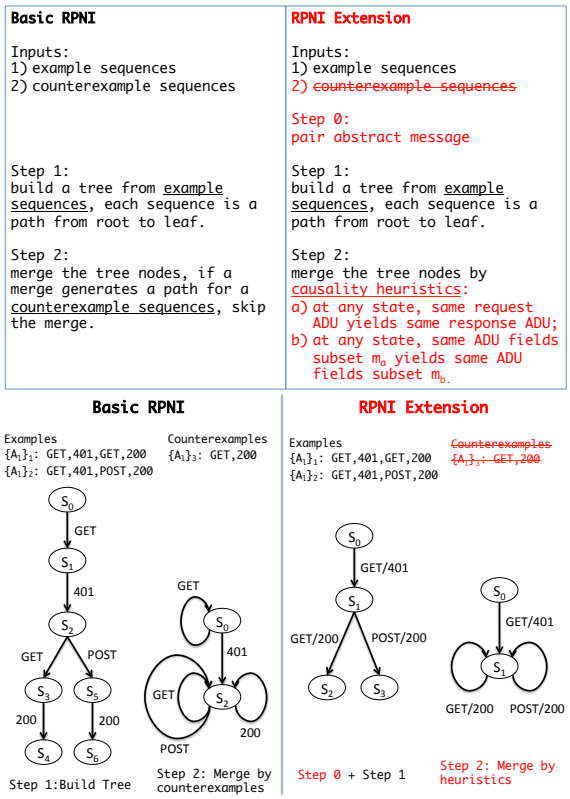


Figure 10: Basic RPNI algorithm and our extension.

the two heuristics are generalizable. Consider a session for any protocol, the session is composed by a sequence of ADUs. If the protocol is composed of request and response pairs (e.g., HTTP, SMB and SNMP), the occurrence of a response highly depends on the previous request. If the protocol is not composed of request and response pairs (e.g. Telnet, DNS and MQTT), then the occurrence of some ADU fields highly depends on some other ADU fields. For example, for Telnet, the choice of `subcommand` field in an ADU highly depends the `command` field.

To extend RPNI, we first need to adapt basic RPNI to Mealy Machine, so we add a Step 0 to pair abstract message and build a Mealy Machine tree in Figure 10. Then we modify Step 2 to use the causality heuristics, e.g., in the bottom right figure in Figure 10, in Step 2, our causality heuristics will prevent S_0 and S_1 from been merged, otherwise a GET input in S_0 will result in two outputs - 200 and 401, which contradicts the determinism of Mealy machine.

Basic RANSAC: We first provide some background on the RANSAC algorithm in Figure 11. RANSAC stands for "Random Sampling and Consensus". Intuitively, if the amount of pollution is not a significant majority, then we can learn a robust model even with polluted data if the model is "self-consistent" in some way. The basic idea here is to randomly sample data from a possibly polluted training set, build a model using just this random sample, and then evaluate how well this model "fits" the remaining dataset. We can simply repeat this process for a few iterations and pick the "best" model over these iterations.

For a set of data items, or simply *items*, the algorithm first selects a random subset of size r and builds a model from the subset. For example, in Figure 11, build a line from a subset of points. Then

```
Input: D - a set of data points
for iterations ← 1 to k
    D_iter ← SampleRand(D, r)
    ▷ build a base model from r samples
    Model_iter ← Learn(D_iter)
    D_add ← {}
    D_test ← D − D_iter
    for each d ∈ D_test
        ▷ check if the rest of the data fits the model.
        if DataFit(d, Model_iter) < dt
            D_add ← D_add ⋃ d
    ▷ check if a good model is found
    if D_add > n
        ▷ Update the model with all "inliers"
        Model_iter ← Learn(D_iter ⋃ D_add)
        ▷ evaluate the loss on "inliers"
        loss = ModelLoss(Model_iter, D_add)
return the Model_iter with the smallest loss in k iterations
```

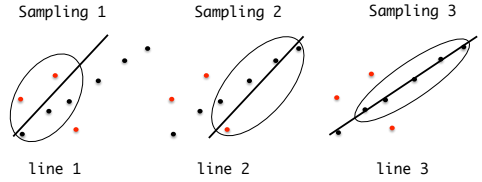
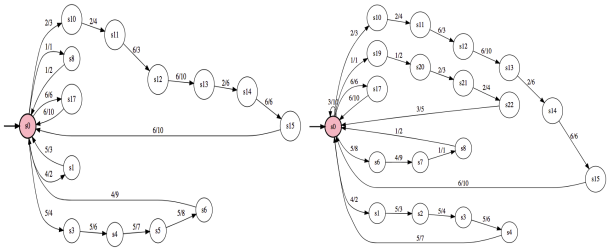


Figure 11: Basic RANSAC and three iterations generating different models (line 1 - 3).

the algorithm tests all other items against the model based on a *DataFit* function and if the mismatch is less than a threshold dt, then the item is said to fit the model well, and it adds the item to the subset. For example, in Figure 11, the *DataFit* function can be easily defined as the Euclidean distance from point to line. Eventually, if the subset size is greater than a size n, the model is considered to be reasonably good. It then evaluates the model with a *ModelLoss* function and computes the loss. For example, in Figure 11, the *ModelLoss* function can be easily defined as the sum of Euclidean distance from point to line. It repeats this process for k iterations and selects the model with the minimum loss as the best model.

RADAR extension for RANSAC: However, RANSAC is not directly applicable for the FSM model of IoT-specific behaviors. This is because there are two modules in RANSAC that are hard to define for the FSM model of IoT behaviors. First, we need to define a *DataFit* function to find whether an ADU series A fits an FSM model. More importantly, we need a *ModelLoss* function to evaluate how likely a model is a good benign IoT behavior model.

First, we show how we define the *DataFit* function by taking advantage of the structural differences of the FSM models. The input of a *DataFit* function is a ADU series A and an FSM model, and the output is whether this ADU series A fits the *FSM*. First, we convert each ADU series A to a sequence of transitions $\delta_1, ..., \delta_A$, and match $\delta_1, ..., \delta_A$ with *FSM* starting from its initial state s_0. We can define the output of the *DataFit* function as the number of unmatched transitions in $\delta_1, ..., \delta_A$. However, the problem with this definition is that the transitions $\delta_1, ..., \delta_A$ generated by some attack behaviors may deviate only slightly from the benign FSM model, resulting in a very small mismatch. We amplify the mismatch caused by the attack behavior A', by computing the Levenshtein distance between two FSM models – *FSM* and *FSM*′. Here *FSM* corresponds to the original subset of benign ADU series set, say $\{A_l\}_{origin}$, and *FSM*′ corresponds to $\{A_l\}_{origin} \cup A'$. For example, Figure 12 shows the *FSM* generated from all benign behaviors, and the *FSM*′ generated adding one more malicious behavior (code injection for



FSM generated from all benign behaviors

FSM′ generated by adding one more malicious behavior (code injection)

Figure 12: A benign behavior model and a malicious behavior model.

```
for iterations ← 1 to k
    {A_l}_iter ← SampleRand({A_l}, r)
    ▷ build a base model from r samples
    FSM_iter ← RPNIExtension({A_l}_iter)
    {A_l}_add ← {}
    {A_l}_test ← {A_l} − {A_l}_iter
    for each A ∈ {A_l}_test
        ▷ check if the rest of the ADU fits the FSM.
        if DataFit(A, FSM_iter) < dt
            {A_l}_add ← {A_l}_add ⋃ A
    ▷ check if a good model is found
    if {A_l}_add > n
        ▷ Update the model with all "inliers"
        FSM_iter ← RPNIExtension({A_l}_iter ⋃ {A_l}_add)
        ▷ evaluate the loss on "inliers"
        loss = ModelLoss(FSM_iter, {A_l}_add)
return the FSM_iter with the smallest loss in k iterations
```

Figure 13: RADAR's learning mechanism.

Cisco routers) to the subset. Our design takes advantage of the structural differences of the FSM models to magnify the difference between attack behaviors and benign behaviors. We can observe that the small difference (2 transitions differences) in Figure 9 is amplified to a huge structural difference between *FSM* and *FSM*′. Based on this idea, we define the data fit function *DataFit* as follows:

$$DataFit(FSM, A) = lev(FSM, FSM'), \tag{1}$$

where $FSM' = RPNIExtension(FSM, A')$ and lev is the Levenshtein distance between two FSMs, defined as:

$$\begin{aligned} lev(FSM_1, FSM_2) = (|\lambda_1| + |\lambda_2| - 2 * |\lambda_1 \cap \lambda_2|) \\ +(|\delta_1| + |\delta_2| - 2 * |\delta_1 \cap \delta_2|), \end{aligned} \tag{2}$$

Next, we show how to define the *ModelLoss* function. The *ModelLoss* function is used to evaluate how likely an FSM model is a good benign behavior model. We posit that an attacker will need to use extra states and/or state transitions beyond what the benign model has for the limited set of IoT behaviors to launch an attack. Based on this idea, we define the *ModelLoss* function as follows:

$$ModelLoss(FSM, \{A_l\}) = \omega_\lambda * |\lambda| + \omega_\delta * |\delta|, \tag{3}$$

where $\omega_\lambda * |\lambda|$ represents the loss caused by using extra transitions and $\omega_\delta * |\delta|$ represents the loss caused by using extra states. Together they capture how much the attacker's behavior deviates from the limited enterprise IoT behaviors.

Putting it together: Now we put the RPNI extension and RANSAC extension together. The pseudo code is given in Figure 13. Comparing with basic RANSAC, the input is changed to a set of ADU series. We replaced the model learning with our RPNI extension and applied RADAR's *DataFit* and *ModelLoss* function.

6 IMPLEMENTATION

We implemented RADAR by extending a number of open source tools and libraries; e.g., we extend `Bro` to parse different protocols and incorporate RADAR's *passive learning* and *online detection* module written in Java (3000 LoC).

- *Protocol Parser:* We use `Bro` to parse seven different protocols http, https, CMP, snmp, smb, dns and telnet. We write `Bro Scripts` for each protocol to build RADAR's abstract ADUs. The abstracted ADUs are exported to `Bro` logs that are fed into the offline learning module as well as the online detection module.
- *FSM Model:* We implement the FSM model as a Java class for a Mealy Machine, and implemented operations such as calculating the Levenshtein distance between two FSMs.
- *Learning Module:* We implement our custom extensions of RPNI and RANSAC in Java. The input to the learning module are the set of `Bro` ADU logs and the output is an FSM model to abstract benign behaviors.
- *Online Detection Module:* We implement the online detection module in Java. This module takes as input the benign or reference FSM model from the *passive learning* module and a ADU series for a new session. Then, we check if the ADU series of the completed session logically fits the benign FSM model. If not, RADAR outputs an alert on this completed session to an alert log file. To enable further diagnosis, we also output the abstracted ADUs, the FSM model and the mismatches in the FSM model to help administrators identify the root cause of the alert.

7 EVALUATION

In this section, we evaluate RADAR and show that

- RADAR achieves low FPs and FNs, improving the F-Score by 5X comparing with adapting prior anomaly detection approaches [26, 28, 34, 52] for IoTs.
- RADAR is robust against pollution (F-Score>0.9 when the percentage of polluted traffic ranges from 1% to 15%),
- RADAR's behavior model can be aggregated across device instances for behavior coverage and scalability,
- RADAR's performance is robust across a range of configuration settings (F-Score>0.9) and does not require fine-grained tuning to achieve satisfying performance, and
- RADAR's representation output can provide useful insight for enterprise admin to further investigate the incidence.

Setup: We evaluate RADAR by considering the common enterprise IoT devices as discussed in Section 2, i.e., routers, switches, NAS, cameras, DVRs, smart lights, smart plugs, fire alarms, sensor gateways and smart speakers (Amazon Alexa and Google Mini)[6]. We setup a sandbox IoT environment in an organization with the above 9 classes of IoT devices deployed, as shown in Figure 18 in Appendix A. The IoT devices are either physical IoT devices or emulated IoT devices running their official firmware via QEMU[23]. To handle IoT devices with encrypted traffic, we can use TLS proxy to inspect the encrypted traffic (e.g., for Amazon Alexa, we setup a TLS proxy with certification and private key from Alexa Skills), or only monitoring the unencrypted part of traffic (e.g., for Google Mini, we monitor the unencrypted HTTP and DNS traffic).

One challenge for evaluating RADAR is that there are few benign or attack traces available for IoTs in enterprise setting. We surveyed a number of public repositories (e.g., [17, 21]) and found few relevant traces for enterprise IoTs "in the wild". To address this challenge, we conducted an IRB-approved study to build an network traffic dataset for 10 types of IoT devices (Figure 14, Alexa and Google Mini considered as 2 device types). The dataset contains 371.6 MB of pcaps with >700K packets, including >10,000 benign traces and >1000 attack traces. The traces are recorded from our sandbox IoT environment. Each trace contains all the packets of a benign or malicious session. To collect benign traces, we invite 10 human users to operate on the IoT devices and record the network traffic as benign traces. We provide the official user interfaces of the IoT devices (webpage, terminal or mobile apps) to the users. The users are free to use any operation provided by the user interfaces. For example, D-Link camera is controlled by a webpage with audio on/off button, streaming on/off button and zoom-in X2, X4, X8 buttons. The users are free to click these buttons. To collect attack traces, we build an IoT attack bundle, including several recent IoT exploits, such as the Mirai malware[27] (against Camera), DNS rebinding[14] (against smart speakers, smart lights, routers and switches), Alexa eaves-dropping attack[6] (against smart speakers), SambaCry[24] (against NAS) and CMP/SNMP exploits[12] (CIA Vault 7 Exploits against routers and switches). Each attack trace includes a unique sequence of real-world exploits (e.g., different code injections) and after-exploit operations (e.g., exfiltrate video/voice or modify credentials) that are commonly seen in enterprises. In total, we used 24 exploits (2-3 exploits per device type) and 92 after-exploit operations (8-10 per device type) to build the attack traces. Each attack trace is different from each other (different sequence of operations). Then to build an unlabeled trace set for evaluation, we randomly mix the attack traces and benign user traces by percentage (e.g., mix 100 attack traces and 900 benign traces for a trace set with 10% of polluted traces.).

Accuracy: Figure 14 compares RADAR with several candidate solutions on canonical accuracy metrics such as the false positive rate, false negative rate, overall accuracy, and F-score. The Accuracy describes the correctness of detection, defined as $Accuracy = \frac{TP+TN}{Pos+Neg}$. The FPR (False Positive Rate) describes the occurrence of False Positives, defined as $FPR = \frac{FP}{FP+TN}$. The FNR (False Negative Rate) describes the occurrence of False Negatives, defined as $FNR = \frac{FN}{FN+TP}$. The F-Score reflects the detection performance combing FP and FN, defined as $FScore = \frac{2*Precision*Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. These measurements are taken over the unlabeled traces sets for 9 classes of enterprise IoT devices. Each unlabeled traces set contains 1000 traces and 10% of them are attack traces.

We compare RADAR with other approaches that can be applied to unlabeled sample sets: two clustering-based anomaly detection (Ngram + Xmeans by Antonakakis *et al.*[28] and Editing Distance + Single Hierachy by Perdisci *et al.*[52]) and two FSM-based approaches (Prospex[34] and the commercial Suricata IDS[26]). To

[6] Smart speakers (Amazon Alexa and Google Mini) are included in our evaluation for their growing deployment (especially in conference rooms and hotels[1]), their complex functionality (e.g. play music, check news or online shopping) and their encrypted communication, altogether created a challenging use case for RADAR.

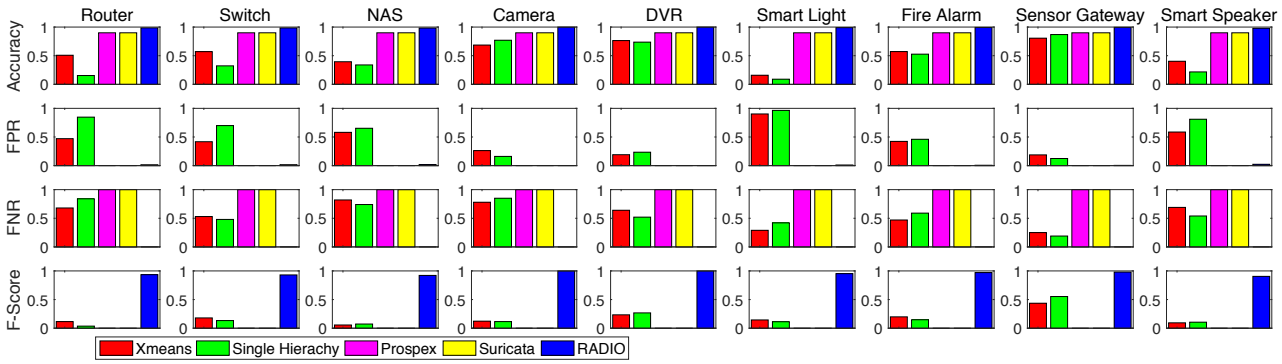


Figure 14: We measure the FP and FN of five different approaches over routers, switches, NASes, cameras, DVRs, smart lights, fire alarms, sensor gateways and smart speakers. 10% of the traces for each device classes are attack samples.

apply the approach by Antonakakis *et al.*[28], we extract Ngrams from the ADU series of each session, and use Xmeans to cluster the sessions. To apply the approach by Perdisci *et al.*[52], we calculate the Editing Distance among the ADU series of each session, and use the Editing Distance vector of each session to cluster the sessions. To apply the approach by Prospex[34], we implemented the FSM inference mechanism in Prospex[34] to infer the FSM model. We evaluate the performance of other approaches by selecting the best result yield by different parameter combinations[7] while using a fixed set of parameters for RADAR (sampling size r = 4% of all samples, model assertion threshold n = 20% of all samples and iteration k = 10000 times).

The first row shows that RADAR achieves the highest accuracy close to 1. The second and third row shows that RADAR achieves both low False Positive Rate and low False Negative Rate. The clustering approaches often have a high False Positive Rate because benign traces are often grouped into different clusters and only one cluster is considered as normal by their mechanism. The FSM-based approaches often have a high False Negative Rate, because their mechanism cannot filter out the polluted data in the historical traces, so the behavior models they generate contain malicious behaviors. The last row shows that RADAR's detection performance combining FP and FN outperforms the comparison approaches and achieves a F-Score close to 1.

In summary, even with a fixed (default) parameter, RADAR outperforms the best results of other approaches under different parameters by 5X.

Robustness against pollution: Next, we show that RADAR is robust to unlabeled, potentially polluted sample sets from diverse enterprise IoTs. There are four factors that may impact the detection accuracy of RADAR: the fraction of attack samples; the diversity of attack types in the attack samples; the time (number of iterations) used by RADAR to build the model; and the diverse classes of IoT devices. We evaluate the robustness of RADAR against the impact of these four factors (by default, we set the fraction of attack samples to 15%; the attack types to be evenly distributed; the time used to 200s; and the device class to be Cisco routers).

First, we change the fraction of attack samples (traces) in the sample sets and measure the accuracy of RADAR, as shown in Figure 15(a). We generate 8 polluted trace sets (each with 1000 traces) for Cisco router with the fraction of attack traces ranging from 1% to 50%[8]. We measure the RADAR's average F-Score (with min-max error) in 100 runs. We can see that, when the fraction of attack traces is less than 15%, RADAR is accurate (F-Score>0.9). Note that if the fraction of attack traces is higher than 15%, then the attack is "noisy" and should be easy for the security administrators to identify (e.g., by flow statistics).

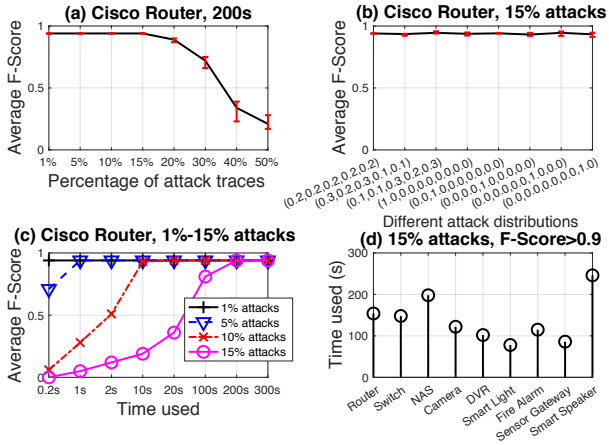


Figure 15: RADAR's robustness against pollution.

Next, we evaluate the impact of the distribution of different types of attacks in the sample sets, as shown in Figure 15(b). We use 5 different types of attacks (including four code injections and the DNS rebinding attack) to build the attack traces set (of 150 attack traces) for Cisco router. We change the distribution of the 5 types of attacks and measure RADAR's average F-Scores (with min-max error bar) in 100 runs. For example, in Figure 15(b), the distribution "(0.2, 0.2, 0.2, 0.2, 0.2)" in the X axis means that each type of attack occupies 20% of the attack traces. We can see that the variation of the F-Score is small under different distributions. This is because RADAR magnifies the structural changes to the FSM behavior model caused by attacks, thus is robust against the distribution changes of different types of attacks.

After that, we change the time (number of iterations) used by RADAR to build the model to see if RADAR can generate a precise FSM model (F-Score close to 1) in a reasonable amount of time. In Figure 15(c), we change the iteration number k in the RANSAC

[7]For Xmeans, we adjust the number of seeds from 2 to 20 and maximum number of clusters from 2 to 20. For Single Hierarchy Clustering, we adjust the number of clusters from 2 to 20 and try three different distance functions Euclidean Distance, Manhattan Distance and Chebyshev Distance. For FSM-based approaches, there are no parameters to adjust.

[8]By theory, the RANSAC module will not work when the fraction surpasses 50%[41].

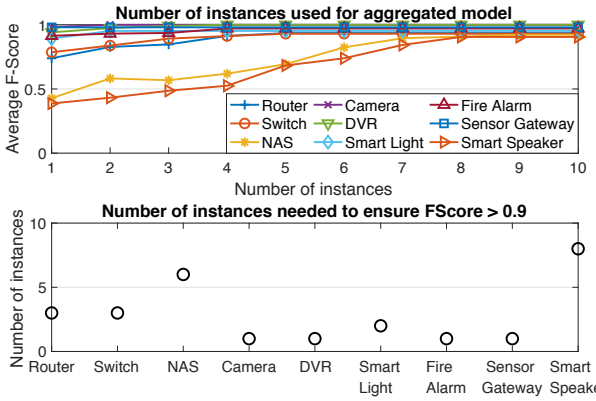


Figure 16: Number of instances needed to learn the aggregated models.

component to control the time used to generate the FSM model and measure the F-Score for traces sets with fraction of attack traces ranging from 1% to 15% (for Cisco routers). We can see that, when the fraction of attack traces changes from 1% to 15%, RADAR can always generate a precise FSM model within 200 seconds (F-Score is 0.935).

We also show that RADAR is able to generate a precise FSM model for different classes of IoT devices. Figure 15(d) shows the time used by RADAR to generate a precise FSM model (F-Score>0.9) for 9 different classes of enterprise IoTs. We can see that, even when 15% of the traces are polluted, RADAR can generate a precise FSM model for different IoT devices within 248 seconds (the smart speaker case). Since building the model is one-time effort in the offline stage, the time cost of 248*s* is acceptable.

Aggregated behavior model for coverage and scalability: In this part, we show that, by building aggregated behavior model from only a few device instances, RADAR can provide good behavior coverage and scalability. We focus on aggregated behavior model across IoT device instances of the same device type (e.g., all Cisco 3640 routers), or of the same class & vendor (e.g., all Cisco routers) in an enterprise. We leave the aggregated model for IoT device instances of different vendors (e.g. all Cisco and Juniper routers) or deployed in different enterprises for future work.

In the first experiment, we evaluate how many device instances are needed to generate a good model to cover the benign behaviors for one type of devices. We collect the historical traces for 10 device instances (each with a different user to maximize the behavior difference) for each device types shown in Figure 16. We randomly select a subset of instances from the 10 instances, and build an aggregated behavior model form the unlabeled traces (10% polluted) of subset of instances. Then we evaluate the aggregated behavior model using all the unlabeled traces (10% polluted) from all the 10 instances and measure the average F-Score by running the experiment for 10 times.

The top figure in Figure 16 shows the F-Score when increasing the number of instances used to generate the unlabeled sample sets for learning from 1 to 10. The X axis is the number of instances used to build the aggregated model, and the Y axis is the average F-Score. From the figure we can see that for some devices, such as NAS or smart speaker, the F-Score is low for 1 user-instance case, meaning that the traces from a single instance are not sufficient to

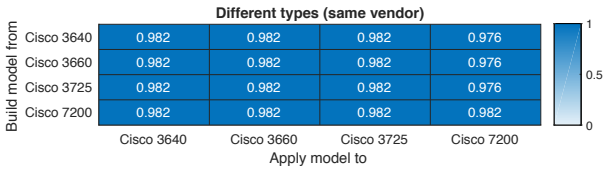


Figure 17: Aggregated behavior model across different device types (same class & vendor).

provide a good coverage of the benign behaviors. Then the bottom figure in Figure 16 shows the number of device instances needed to ensure that the F-Score is greater than 0.9 for each device type (same as device class here). We can see that all the device type need at most 8 instances to provide a good coverage (F-Score>0.9). And while Camera, DVR, Smart light, Fire Alarm and Sensor Gateway only need one instance to ensure F-Score is greater than 0.9, routers, switches, NASes and smart speakers need more instances. This is because the behavior options for the routers, switches, NASes and smart speakers are more diverse than the cameras, DVR, smart lights, fire alarms and sensor gateways. For example, the NASes provides 36 operations while the smart lights only provide two operations (on and off).

Next, we explore the aggregated behavior model across device instances of the same class & vendor (different device type, e.g., Cisco 3640 routers and Cisco 7200 routers). In Figure 17, we generate the behavior model from a few router instances (5 instances) of one type (Cisco 3640) in the offline stage, then apply the model to a few other instances (5 instances) of a different type (e.g. Cisco 7200) for detection. Then we measure the F-Score to see if the model is suitable for sharing. Figure 17 shows the heatmap of the F-Scores We can see that the F-Score varies little across the instances of the same class & vendor (Cisco routers), which means it is possible to share the behavior model across device instances of the same class & vendor.

Sensitivity to parameters: There are three parameters for RADAR: the threshold *dt* of the *DataFit* function that judge if a sample fits a FSM model; the sampling size *r*; and the assertion size *n* that describes the number of samples to assert a FSM is good. We measure the average F-Score of RADAR under different parameters (Figure 19 in Appendix B) and observed that RADAR is not sensitive to the configuration of parameters and does not require parameter tuning to achieve satisfying detection performance.

Diagnostic Utility: Finally, we show that the FSM behavior model of RADAR is useful for enterprise security administrators to diagnose the attacks against enterprise IoT devices (e.g. which command/argument is flawed and used by attacker). As shown in Figure 20 in Appendix C, RADAR can automatically identify the mismatches between the benign behavior model and the ADU series of the attack, and will raise an alert containing the mismatched states and transitions. Using the alerts we obtained from the detection of RADAR, we were able to obtain the information about several flaws the IoT attack exploits, which are not available from NVD[19]. The results are shown in Table 2 in Appendix C. This shows that the FSM behavior model of RADAR can help the admin quickly identify the flaws of the IoT devices and better respond to the security issues of the enterprise IoT devices.

Approach	Class	Method	Input	Target	Precise Model	Pollution
Bartos *et al.*[29]	Classification	SVM	Proxy logs	Malware	✗	✗
Webwitness[49]	Classification	Statistic classifier	Proxy logs	Malware downloads	✗	✗
Soska *et al.*[56]	Classification	Ensemble of decision trees	Webpage	Infected website	✗	✗
Antonakakis *et al.*[28]	Clustering	Ngram + Xmeans	DNS messages	Dga malware	✗	✓
Nazca[45]	Clustering	Graph clustering	packets	Malware distribution	✗	✓
Ho *et al.*[44]	Outlier detection	Anomaly scoring	Emails	Spearphishing attacks	✗	✓
Botminer[43]	Clustering	X-means	Flow statistics	Botnet	✗	✓
Perdisci *et al.*[52]	Clustering & Outlier detection	Editing distance + Single Hierarchy Clustering	HTTP packets	HTTP-based Malware	✗	✓
Cho *et al.*[33]	FSM-based	L Star	Packets	Botnet	✓	✗
Mace[34]	FSM-based	L Star	Packets	Protocol anomalies	✓	✗
Prospex[34]	FSM-based	Prefix tree	Packets	Protocol anomalies	✓	✗
Suricata[26]	FSM-based	Handcrafted protocol FSM	Packets	Protocol anomalies	✓	✗
RADAR	FSM Inference + Outlier detection	RPNI + RANSAC	ADU series	IoT anomalies	✓	✓

Table 1: Overview of the state-of-the-art approaches focusing on network anomaly detection.

8 RELATED WORK

Now we discuss related work in the IoT security and anomaly detection space.

Firmware based analysis: Firmware analysis approaches[32, 35, 36, 54] expose the vulnerabilities inside the device's firmware. However, firmware may not always be available to enterprises for analysis. Other attestation approaches[47] detect unauthorized modification of IoT firmware, but they are limited to detecting firmware modification attacks.

Anomaly detection approaches: The closest work is a rich body of previous anomaly detection approaches that use statistical or behavioral anomalies to alert on attacks. As shown in Table 1, previous network anomaly detection approaches include (1) classification[29, 56], (2) clustering & outlier detection[43, 45, 52], and (3) FSM-based detection [33, 34, 46]. To the best of our knowledge, none of the existing approaches focus on detecting IoT-specific behavioral anomalies. So, instead of directly applying these approaches to IoT devices, we discuss if the methods they use can be adapted to detect IoT-specific behavioral anomalies.

The classification-based approaches [29, 49, 56] need labeled attack and benign samples to train the classifier and cannot apply to unlabeled samples. The clustering-based approaches can apply to unlabeled samples, but none of the previous work provide a precise model to capture the application-level behaviors of enterprise IoT devices. Botminer[43] uses flow statistics as model, which might be useful to detect data exfiltration (by flow data bytes) or botnets (by connection statistics), but cannot detect code injection or authentication bypass attacks. Nazca[45] focuses on graphic representation to detect malware distribution, and Ho *et al.*[44] focuses on email representation to detect spearphishing. The model they use cannot be applied to enterprise IoT devices. Of all the clustering approaches we mentioned, we found two of them that can be adapted to detect IoT attacks. We can use Ngram method, as described by Antonakakis *et al.*[28], to calculate similarity between message series followed by the Xmeans method to cluster them. Likewise, we can use editing distance method, as described in Perdisci *et al.*[52], to calculate similarity between message series and then use single hierarchical clustering to cluster them. Such approaches, however, yield high FP and FN rates because it is hard to exhaustively cover all the benign message series.

Among all the FSM-based approaches studied, we found only two passive monitoring approaches, Prospex[34] and Suricata[26], which can be adapted to detect IoT-specific anomalies. However, they cannot differentiate attack samples from benign ones in an unlabeled sample set in the FSM inference process and may generate states and transitions representing attack behaviors. As a result, they tend to generate high FNs.

Other IoT security work: Several other related works [40, 48] also studied the IoT security problem. FlowFence[40] provides data protection for IoT applications. IoT Sentinel[48] tries to identify the device types in a smart home setting. Both of these complement our work by focusing on other aspects of IoT security.

9 DISCUSSION

Attack evasion: Any behavioral anomaly detection is subject to evasion attacks that follow legitimate behaviors and RADAR is no different. However, many of the core attack behaviors that grant the attackers higher privilege or sensitive information intrinsically violates the legitimate patterns and RADAR would be effective.

Prevention vs. detection: As a starting point, our work focuses on a detection system. That said, our approach can also potentially be extended to implement a real time prevention system (e.g., block or modify the anomalous messages) or be augmented with other kinds of defenses (e.g., dynamic quarantine or deep inspection) when our detection raises alerts.

Non-enterprise IoT: We hypothesize that the insight on behavior being restricted is more broadly true for IoT deployments beyond enterprises. Some of the enterprise IoT devices (e.g. NAS, camera or smart speaker) also widely exist in smart home environment. An interesting direction for future work is to deploy RADAR in smart home environment and evaluate its effectiveness.

10 CONCLUSION

Our work explored a somewhat atypical use case for IoT inside enterprise networks. These devices are an increasing threat to enterprise security. We show that it is possible to revisit and practically realize a behavioral anomaly detection system for such settings. In designing RADAR, we tackled two key challenges: a compact model for the behaviors and learning the model from unlabeled and possibly polluted data. Our results are promising and show that RADAR can detect many IoT attacks in reality and outperforms several state-of-art detection systems. Looking forward, RADAR serves as an interesting proof-of-concept for revisiting behavioral anomaly detection for IoT deployments more broadly.

REFERENCES

[1] 2018. Amazon Wants More Hotels Using Alexa Voice-Powered Services. https://skift.com/2017/11/30/amazon-wants-more-hotels-using-alexa-voice-powered-services/. (2018).

[2] 2018. Blue Coat. https://www.symantec.com/products/bc-data-loss-prevention-dlp. (2018).

[3] 2018. Censys. https://censys.io/. (2018).

[4] 2018. CVE-2017-3881 Detail. https://nvd.nist.gov/vuln/detail/CVE-2017-3881. (2018).

[5] 2018. CVE-2017-6744 Detail. https://nvd.nist.gov/vuln/detail/CVE-2017-6744. (2018).

[6] 2018. Eavesdropping with Amazon Alexa. https://www.checkmarx.com/2018/04/25/eavesdropping-with-amazon-alexa/. (2018).

[7] 2018. ForeScout IoT Enterprise Risk Report. https://www.forescout.com/wp-content/uploads/2016/10/ForeScout-IoT-Enterprise-Risk-Report.pdf. (2018).

[8] 2018. Google BeyondCorp. https://beyondcorp.com/. (2018).

[9] 2018. A Guide to the Internet of Things Infographic. https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html. (2018).

[10] 2018. Hacking Team hack. https://gist.github.com/Sjord/ac8dfff3a3ac3180c065f370f24b30a8. (2018).

[11] 2018. HiSilicon DVR hack. https://github.com/tothi/pwn-hisilicon-dvr. (2018).

[12] 2018. How to Mitigate CIA Vault 7 Exploits on Your Cisco Switches. http://info.stack8.com/blog/how-to-mitigate-cia-vault-7-exploits-cve-2017-3881-on-your-cisco-switches. (2018).

[13] 2018. IDC Global Technology Prediction. https://www.idc.com/getdoc.jsp?containerId=252872. (2018).

[14] 2018. IoT Security Flaw Leaves 496 Million Devices Vulnerable At Businesses: Report. https://crn.com/news/internet-of-things/300106806/iot-security-flaw-leaves-496-million-devices-vulnerable-at-businesses-report.html. (2018).

[15] 2018. List of broadband providers in the United States. https://en.wikipedia.org/wiki/List_of_broadband_providers_in_the_United_States. (2018).

[16] 2018. McAfee. https://www.mcafee.com/us/index.html. (2018).

[17] 2018. NETRESEC. http://www.netresec.com/?page=PcapFiles. (2018).

[18] 2018. Norton By Symantec. https://us.norton.com/. (2018).

[19] 2018. NVD. https://nvd.nist.gov/. (2018).

[20] 2018. OpenWrt 10.03 - Multiple Cross-Site Scripting Vulnerabilities. https://www.exploit-db.com/exploits/34994/. (2018).

[21] 2018. PacketTotal. https://packettotal.com/. (2018).

[22] 2018. Philips Hue susceptible to hack, vulnerable to blackouts. https://www.engadget.com/2013/08/14/philips-hue-smart-light-security-issues/. (2018).

[23] 2018. QEMU. https://www.qemu.org/. (2018).

[24] 2018. SAMBACRY, THE SEVEN YEAR OLD SAMBA VULNERABILITY, IS THE NEXT BIG THREAT. https://www.guardicore.com/2017/05/samba/. (2018).

[25] 2018. SHODAN. https://www.shodan.io/. (2018).

[26] 2018. Suricata IDS/IPS. https://suricata-ids.org/. (2018).

[27] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *USENIX Security Symposium*.

[28] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware.. In *USENIX Security Symposium*.

[29] Karel Bartos, Michal Sofka, and Vojtech Franc. 2016. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants.. In *USENIX Security Symposium*. 807–822.

[30] Alan W Biermann and Jerome A Feldman. 1972. On the synthesis of finite-state machines from samples of their behavior. *IEEE transactions on Computers* 100, 6 (1972), 592–597.

[31] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Maarten H Everts, Pieter Hartel, Rick Hofstede, Willem Jonker, and Andreas Peter. 2017. DECANTeR: DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting. In *Proceedings of Computer Security Applications Conference*.

[32] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware.. In *NDSS*.

[33] Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. 2010. Inference and analysis of formal models of botnet command and control protocols. In *CCS*.

[34] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. 2009. Prospex: Protocol specification extraction. In *IEEE Symposium on Security and Privacy*.

[35] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. 2016. Automated dynamic firmware analysis at scale: a case study on embedded web interfaces. In *Proceedings of the 11th ACM on Asia CCS*. ACM.

[36] Ang Cui, Michael Costello, and Salvatore J Stolfo. 2013. When Firmware Modifications Attack: A Case Study of Embedded Exploitation.. In *NDSS*.

[37] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. 2007. Discoverer: Automatic Protocol Reverse Engineering from Network Traces.. In *USENIX Security Symposium*. 1–14.

[38] François Denis, Aurélien Lemay, and Alain Terlutte. 2004. Learning regular languages using RFSAs. *Theoretical computer science* 313, 2 (2004), 267–294.

[39] Richard O Duda and Peter E Hart. 1972. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM* 15, 1 (1972), 11–15.

[40] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks.. In *USENIX Security Symposium*.

[41] Martin A Fischler and Robert C Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

[42] E Mark Gold. 1967. Language identification in the limit. *Information and control* 10, 5 (1967), 447–474.

[43] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection.. In *USENIX security symposium*. 139–154.

[44] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. 2017. Detecting Credential Spearphishing in Enterprise Settings. In *USENIX Security Symposium*.

[45] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks.. In *NDSS*, Vol. 14. 23–26.

[46] Jayanthkumar Kannan, Jaeyeon Jung, Vern Paxson, and Can Emre Koksal. 2006. Semi-automated discovery of application session structure. In *ACM IMC*.

[47] Yanlin Li, Jonathan M McCune, and Adrian Perrig. 2011. VIPER: verifying the integrity of PERipherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 3–16.

[48] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, Tommaso Frassetto, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. IoT Sentinel Demo: Automated device-type identification for security enforcement in IoT. In *ICDCS*.

[49] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. 2015. WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths.. In *USENIX Security Symposium*. 1025–1040.

[50] José Oncina and Pedro Garcia. 1992. Inferring regular languages in polynomial update time. (1992).

[51] Vern Paxson. 1999. Bro: A System for Detecting Network Intruders in Real-Time. In *Computer Networks*. 2435–2463.

[52] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces.. In *NSDI*, Vol. 10. 14.

[53] Martin Roesch et al. 1999. Snort: Lightweight Intrusion Detection for Networks.. In *LISA*, Vol. 99. 229–238.

[54] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware.. In *NDSS*.

[55] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*. 305–316.

[56] Kyle Soska and Nicolas Christin. 2014. Automatically Detecting Vulnerable Websites Before They Turn Malicious.. In *USENIX Security Symposium*. 625–640.

[57] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. 2017. Smartauth: User-centered authorization for the internet of things. In *USENIX Security Symposium*.

A EVALUATION SETUP

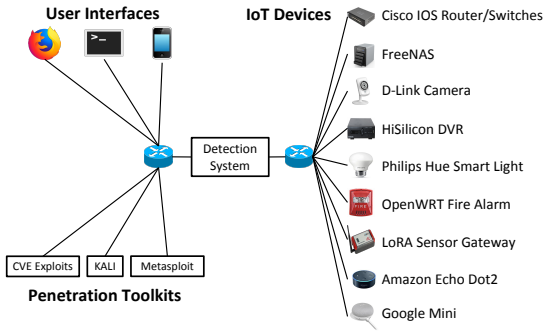


Figure 18: The sandbox IoT environment in a medium-sized organization.

B SENSITIVITY TO PARAMETERS

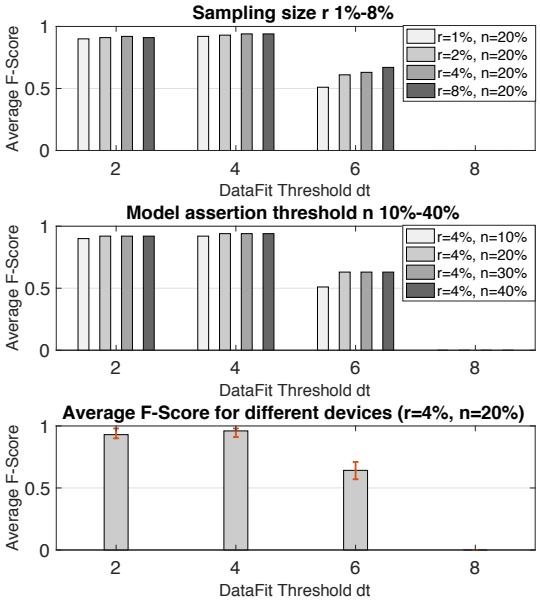
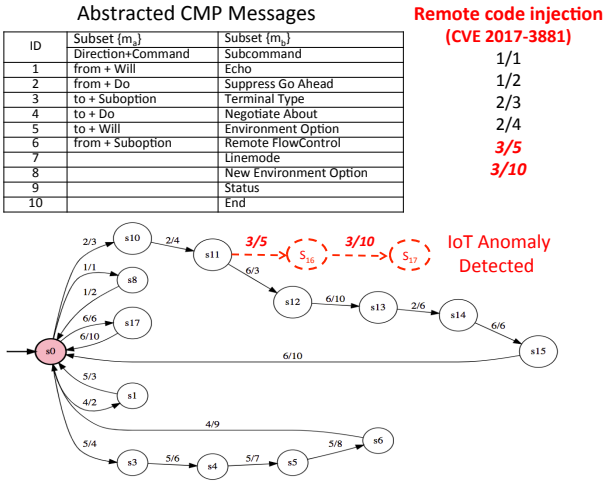


Figure 19: The average F-Score for different combination of parameters. The top and middle figure are measured on Cisco router and the bottom figure is measured on different devices. The above measurements suggest that RADAR is not sensitive to the configuration of parameters. However, it is worth noting that the threshold *dt* of the *DataFit* function should not be too large to sensitively distinguish FSMs.

C DIAGNOSTIC UTILITY



ID	Subset (m_2) Direction+Command	Subset (m_3) Subcommand
1	from + Will	Echo
2	from + Do	Suppress Go Ahead
3	to + Suboption	Terminal Type
4	to + Do	Negotiate About
5	to + Will	Environment Option
6	from + Suboption	Remote FlowControl
7		Linemode
8		New Environment Option
9		Status
10		End

Figure 20: An example of how RADAR's benign behavior model can help to identify an code injection attack (CVE 2017-3881).

Device	Attack	Flaws exploited
Cisco IOS router	CMP Remote Code Exec[12]	The attacker uses the Subcommand called *Environment Option* of the CMP protocol for code injection.
Cisco IOS router	SNMP Remote Code Exec	The attacker uses the *oid* argument of the SNMP protocol for code injection.
Cisco IOS switch	HTTP Authentication Bypass	The first GET/POST request of the attacker is replied with a 200 OK response because the attacker bypassed the authentication.
FreeNAS	SambaCry[24]	The attacker first uploads a malicious file, and then uses *Create Request File* command on this file to trigger the execution of the malicious payload.
Philips Hue smart light	Smart light blackout[22]	The attacker first accesses the smart light bridge to obtain the token used to issue commands, then use the token to turn off all the light to perform a blackout.
HiSilicon DVR	HiSilicon DVR hack[11]	The attacker try to access the root path to escalate the privilege and perform code injection.
OpenWRT fire alarm	Cross-site scripting[20]	The attacker can access multiple URIs under "cgi-bin" to inject arbitrary code.
D-Link Camera	Mirai malware[27]	The botnet exploits a backdoor access provided by the debug module of the D-Link Camera.
Amazon Echo Dot2	Alexa eaves dropping attack[6]	Malicious Alexa skills applications are sending back empty prompts to keep the session alive to eaves dropping on users.
Google Mini	DNS rebinding attack[14]	The code snippet from the attacker directly accesses the http APIs of the Google Mini smart speaker at "/payload/google-home" to probe sensitive information.

Table 2: The alerts and FSM model from RADAR can provide detailed informations about the IoT flaws that are been exploited.