

Securing Smart Homes with OpenFlow

Mitchell Frank and Majid Ghaderi

Department of Computer Science, University of Calgary

Abstract—The Internet of Things (IoT) trend is introducing additional devices to home networks. Home networks face the same threats as every other network. Recently, IoT devices have been compromised by attackers and used as staging points for further attacks. Home users may not have the technical capability or funding to run advanced security devices designed to protect enterprises. Solutions to this problem exist, but in some cases, they rely on third party cloud services or require custom protocols to be deployed within the home network. Reliance on third party services comes with privacy implications, as well as the increased risk for a third party to be responsible for securing a network they may not directly control. Custom network protocols can effectively reduce the attack surface of home networks, but these are not easily compatible with devices in operation today. In this paper, we propose a new model for protecting home networks utilizing OpenFlow enabled Access Points (APs). The solution automatically builds least-permissive policies for each device and subsequently enforces the policies without requiring customized protocols. This allows the system to protect any connected wireless device. The design allows for a flexible deployment model and is capable of running on low cost hardware as an all-in-one unit. We perform a complete implementation and evaluation of the solution. The system can effectively limit the ability for compromised IoT devices to attack internal and external networks at a low cost to initial connection times.

I. INTRODUCTION

A. Security and the Internet of Things

The Internet of Things (IoT) is the growing trend to connect every device to the Internet. It is estimated that 500 billion devices will be connected to the Internet by the year 2030 [1]. The IoT trend is connecting many devices including: fitness devices, televisions, refrigerators, washing machines, and alarm systems. IoT devices target both home and corporate networks.

Increased connectivity results in additional security challenges for networks. Networks face a number of diverse threats, including: ransomware, malware, and denial of service attacks. Cyber attacks have sparked a new industry, the Cyber Attack Business [2]. Enterprise and home networks are targets for malicious actors.

The cyber security industry is growing rapidly to counter malicious actors. The cyber security market is projected to have an estimated value of \$248 billion by the year 2023 [3]. There are many products available in the market for advanced users and enterprises. These solutions include technologies such as Intrusion Detection Systems (IDSs) and Next-Generation Firewalls (NGFWs) [4]. These advanced solutions can be costly to acquire or require specialized hardware and knowledge for proper configuration. This presents difficulties when applying the same concepts to a home network as assumptions cannot be made about the home user in terms of

network administration capabilities. Devices marketed towards home users have a track record of being inadequately hardened which has led to home users being subject to attacks. Recent examples include the VPNFilter campaign [5] which targets routers and is used as a staging point for future attacks. Search engines targeting insecure network devices enable quick discovery of improperly configured or insecure devices [6].

The National Institute of Standards and Technology (NIST) groups the challenges faced by IoT devices into three categories [7]: device security, data security, and privacy. Device security is essential in preventing the device from being used as a pivot point for further attacks. Data security refers to protecting the Confidentiality, Availability and Integrity (CIA) of data being generated or stored on the device. Privacy encompasses the protection of an individuals' privacy through the use of an IoT device. Following the CIA model, there is a range of security issues that impact IoT devices such as data leakage, denial of service, and impersonation (see [8] for a detailed discussion).

Transferring responsibility for home network protection to Internet Service Providers (ISPs) is proposed as a solution for addressing IoT security problems [9]. Such an approach could effectively address insecure IoT devices being connected to the internet. Two variations of this solution can be envisioned: thick and thin. In the thick security model, the home user is ultimately responsible for all network security and the Internet Service Provider (ISP) provides no support to the home user. In the thin security model, the ISP takes full responsibility for the home network and is responsible for ensuring it remains secured. However, this approach may have legal implications for the ISP as discussed in [9]. IoT devices also encounter availability challenges as they often rely on Cloud Service Providers (CSPs). If a home relies on a cloud service with no local fallbacks, it may become unresponsive or unreliable if the upstream provider becomes unavailable. A solution is to place an offline device in a home network to synchronize states between local and cloud services [10]. We explore related work and our solution to address the aforementioned challenges in the remainder of this paper.

B. Our Work

In this work, we present the design and evaluation of a solution to secure home networks which we call the Flow Policy Enforcer (FPE). The FPE takes a least-permissive policy based approach to network security. This enables IoT devices to operate as intended but restricts their network functionality in the event they are compromised, thereby reducing the ability for an attacker to use an IoT device to stage further attacks.

Our approach differs from other works which require custom protocols or customization to the network infrastructure. Some of these other works are explored in the next section of this paper. While these solutions are able to provide security, they would not be compatible with current networking standards out of the box. As a result, our solution is designed with the following goals in mind:

- Vendor agnostic: It must only require a compatible access point and not require hardware from a specific manufacturer.
- Compatible: There can be no modification to the end devices or customization of network protocols.
- Automatic: User interaction is not required. Additional functionality can be utilized by the user if they choose to do so.
- Secure: IoT devices are limited to least-functionality to restrict their network access in the event they are compromised.

The FPE controls data flow through an OpenFlow enabled Access Point (AP). Such APs are commercially available.¹ As devices are connected, the FPE automatically learns least-permissive policies for each device. After a period of time, the learned policy is enforced on the device. This prevents a compromised device from being used to attack other network resources. Additionally, the FPE does not require any cloud services for data processing and all data resides in the privacy of the home network.

Our contributions in this paper are:

- 1) We present the Flow Policy Enforcer, our solution to address the security challenges associated with IoT devices.
- 2) We implement the FPE and deploy it on low-cost hardware in a Local Area Network (LAN).
- 3) We present measurement of the system and demonstrate the effectiveness of our solution.

C. Paper Organization

Background Information and Related Work is reviewed in section II. The solution is proposed in section III and implementation is discussed in section IV. System evaluation is explored in section V. The work is concluded and future work is discussed in section VI.

II. BACKGROUND AND RELATED WORK

A. Background Information

1) *Software Defined Networking*: Traditional network devices such as routers and switches integrate the control and data planes into a single device. The control plane makes decisions on how network traffic should be forwarded within the device and the data plane is responsible for performing the forwarding actions. Software Defined Networking (SDN) [11] is a model for separating the control plane from the data plane. The control plane logic is brought into a centralized controller which can make decisions based on the entire network layout. The network devices receive forwarding instructions from the

centralized controller and perform forwarding actions based on received rules.

2) *OpenFlow*: OpenFlow is an SDN protocol which allows for detailed control of the forwarding plane in network devices [12]. For example, in a conventional switch, the control plane and the forwarding plane are combined in a single unit. OpenFlow allows the control plane to be configurable. By extension, this configurability allows the forwarding plane to perform custom actions as defined by the OpenFlow controller. The processing pipeline of OpenFlow is defined in flow tables and flow entries.

Flow tables are read in sequential order and allow for sets of flow entries to be grouped together in a single table. When a matching flow entry is found, the actions configured in the flow entry are executed.

Flow entries are comprised of the following fields: a cookie, priority, match fields, counters, and actions. The flow entry can be configured to perform actions including modification of the frame and sending it out of a specified port, group of ports, and sending the frame itself to the network controller. A low priority flow entry can be used as a default action in the device to handle unknown frames.

A controller is capable of configuring the flow rules on OpenFlow devices. Controllers can add, update, and remove flow entries and flow tables from a central point in the network. This allows for detailed control of network devices which extends beyond functionality provided by standard network protocols.

B. IoT Device Identification

Successful identification of IoT devices can be used to provide the network with additional information. This information could be used to restrict access or ensure that traffic originates from a trusted IoT device. Aman et al. proposed a data provenance protocol for identifying devices [13]. Their work builds on the use of Physical Unclonable Functions (PUFs) of a device. A PUF is a feature of a physical object which can be treated as unique and unclonable. When the IoT is initially deployed, a Challenge-Response Pair (CRP) from the device is registered with the server. The PUFs are used to generate and register a set of fingerprints with the server. Future communications between the IoT device and server require the use of the appropriate fingerprint and can be used to mutually authenticate each other.

Meidan et al. proposed the use of machine learning classifiers to identify IoT devices based on network traffic in their solution called ProfilloT [14]. Given a device and a series of sessions, a classifier is constructed for each session of the device. These classifiers output a probability for any input session that the input was generated by the device. The optimal classification threshold for each of these single-session identifiers is calculated. Finally, the optimal count of individual single-session identifiers is calculated to provide no false positives or false negatives for a given device. Once the classifiers are constructed, session flows can be run through the classifiers to identify which device is generating traffic on the network.

¹For example, the AT-TQ4600-OF13 by Allied Telesis

In [15], Danev et al. explore the identification of devices based on physical layer attributes. These attributes can be the result of manufacturing imperfections or other in-specification or out-of-specification operation of wireless equipment. In their survey, the authors discuss the feasibility of identifying 802.11 wireless devices based on physical attributes. Further research is required to determine feasibility, cost and other factors of this approach.

Our solution varies from the approaches in [13] and [14] as we automatically build policies for any device connected to the network. We do not require specialized hardware to identify connection characteristics or perform machine learning based on sets of well-known devices.

C. OpenFlow in WiFi and Security

There are some works on using OpenFlow in WiFi networks. For example, Vestin et al. explore the use of OpenFlow in Wireless Local Area Networks (WLANs) in [16]. In this work, the CloudMAC platform is composed of the following pieces: Wireless Termination Points (WTPs), Virtual Access Points (VAPs), an OpenFlow controller, and an OpenFlow switch. The VAPs are virtual machines which run on a central server and perform actions as if they were a traditional AP by processing frames. WTPs are antennas which are used solely for the purpose of receiving and transmitting packets. The OpenFlow switch is controlled by the OpenFlow controller and the controller sets up flow rules forwarding packets between VAPs and WTPs. Performance testing done by the authors shows that the setup has low enough latency to support traditional wireless clients.

Porras et al. proposed a method for securing OpenFlow flow rule changes called FortNOX in [17]. FortNOX is an extension for the NOX OpenFlow controller and can be run as an extension of the controller. FortNOX provides a mechanism for a network administrator to define a policy for the network. FortNOX ensures that new flow entries do not conflict with the policies defined for the network. In the case of a conflict, FortNOX performs a series of comparisons, including the priority of the source requesting the rule to be added. An administrator is notified in the case where a conflict cannot be resolved.

Our solution varies from [16] as we do not virtualize the AP functionality to an external server. Similar to [17], we take a least-permissive policy based approach. Our solution differs as we automatically learn and enforce policies on network devices instead of focusing on conflicting OpenFlow rules.

D. Cloud Based Firewalls

Cloud based firewalls are another approach for protecting networks. These firewalls utilize a cloud component for analysis and can scale in the cloud to meet customer demand. Taylor et al. show this is a viable approach and explore the potential impact of latency and data center locations in [18].

Shirali-Shahreza et al. introduce a cloud based solution for protecting home networks in [19]. In this work, network protection is broken down into two main components: the enforcer and the mastermind. The enforcer is a device which sits at the

network edge. This allows the enforcer to either allow or block flows at the edge of the network. The mastermind is a cloud service that communicates with and configures the enforcer. The enforcer samples flows and sends sampled packets to the mastermind. As the mastermind receives information from the enforcer, it can update flow rules on the enforcer to block malicious flows, as well as adjust the sampling rate on the enforcer to account for networking impact of the sampling process.

Nobakht et al. proposed a method for protecting home networks called IoT Intrusion Detection and Mitigation (IoT-IDM) in [20]. This approach proposes that a home user and a Software as a Service (SaaS) provider coordinate on network security. The devices to be monitored must be manually input by an administrator into the IoT-IDM system. Once configured, a virtual sensor is created over the network flow which sends traffic to a feature extractor. The feature extractor analyzes fields of interest to the flow and a detection unit builds machine learning models based on the features. These models can then be placed in a mitigation module which runs the machine learning modules and can block flows that are considered to be malicious. The SaaS provider is responsible for configuration of the feature extractor and detection units to identify which fields are of interest and which models should be built off of acquired data.

Our approach differs from [19] and [20] as our solution is deployed within a LAN. Avoiding reliance on third parties allows us to avoid privacy and availability concerns associated to cloud based solutions.

III. FLOW POLICY ENFORCER

A. System Architecture

Figure 1 is a diagram of the system architecture. The Flow Policy Enforcer (FPE) is comprised of four components: a web application, an SQL database, an OpenFlow controller, and the Redis database [21].

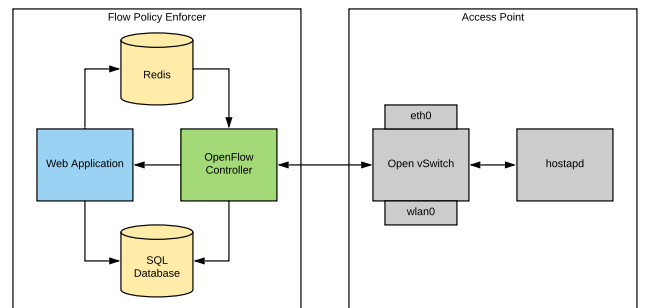


Fig. 1. Architecture of the Flow Policy Enforcer.

B. Web Application

The web application is responsible for maintaining system state and integrity. It utilizes an SQL database for all recorded data. The web application publishes messages to the controller using Redis based on user interaction. For example, a message is sent to the controller via Redis if the user decides to block

all network access for a given device. The web application also provides a user interface which allows users to manage the system, modify rules, view usage statistics, and transition device states. We will explore rules and device states in a later section.

The web application also hosts a REST API which exposes functionality to the OpenFlow controller. The API has two primary endpoints:

- The *access points* endpoint allows the controller to request an initial set of rules and a list of managed devices from the web application. This endpoint is also used by the controller to register a new station to a given AP. If a station has already been registered, its registration is updated to point to the newly associated AP. The registration process is used to reduce the number of flow rules created for a given station in the event that the system is managing multiple APs.
- The *packets* endpoint allows the controller to submit DNS lookups and statistics to the web application. These data types are submitted to the web application for processing as it is not critical to process the data quickly.

C. OpenFlow Controller

The OpenFlow controller interfaces between OpenFlow enabled access points, the web application, and the SQL database. It communicates to the web application using the REST API, receives messages from the web application via Redis queue subscriptions, and communicates with access points using the OpenFlow protocol.

The controller maintains a persistent connection with each connected AP. When a connection to an AP is created, the controller performs two main actions. First, it creates a set of default rules on the AP. Secondly, it utilizes the *access points* API endpoint to request an initial set of rules for each station which belongs to the AP. This persistent connection is also used when the controller receives instructions to create or modify flow rules from the web application.

The controller maintains a list of device states in memory. These device states are used to determine which actions the controller should take when an unknown packet reaches the controller through one of the default rules on the AP. If applicable, the controller will process the packet, create a rule to permit the connection in the access point, and inject the permitted packet through the data plane of the access point. This rule will be saved in the SQL database as described in the next section.

D. Client States

Any device associated to the AP can be in one of four states: Full Access, IoT Learning, IoT Frozen, or Blocked. The states can be represented as a complete graph, that is, a device can be transitioned between any two given states by the user. The system will automatically transition clients between the IoT Learning and IoT Frozen states once the learning time has elapsed. In addition to the transitions, the initial state is configurable by the user, but, by default, it sets all devices which connect for the first time as IoT Learning.

We assume that IoT devices initially connected to the network are not compromised. Additionally, we assume that they will not be compromised for a period of time. By default, we specify the learning time as 48 hours. These assumptions are made to provide learning time for construction of per-device policies. The automatic transition between the IoT Learning and IoT Frozen states satisfies the automatic goal of the system.

Full Access allows the device to have unrestricted access to local and external networks. This is provided as an option in case the user connects a laptop or other device for which they do not want to restrict access. The Blocked state creates a drop rule for the specified device to prevent it from sending or receiving data on the network.

E. Device Policies

Each device being monitored is assigned a unique policy. A policy is a collection of rule sets. Each rule set is a collection of rules. By default, every policy has a default learning policy where automatically learned rules are recorded. Optionally, users can create custom rule sets and apply them to multiple policies.

A rule is the collection of parameters which define allowed communication paths for the device. Each rule is constructed of the following elements: source MAC address, destination MAC address, source IP address, destination IP address, port, protocol (TCP or UDP), and direction (inbound or outbound). The direction is used to configure the port on the correct OpenFlow rule due to source port randomization. On outgoing connections from the IoT device, the port is compared to the destination port of the external server. Conversely, on incoming connections to an IoT device, the port is compared to the destination port to the IoT device. The technical details of each rule are abstracted from the user in the web interface. The devices can be assigned a custom name and icon. If the user goes into the rule modification view, they can see the full details of each rule.

While a device is in IoT Learning state, the FPE module constructs policies by creating rules based on traffic generated by the device. The goal of this approach is to generate a minimal rule set which allows the device to function as intended. As the device transitions into the IoT Frozen state, it can only communicate with resources allowed by the policy.

F. AP Default Rules

When an access point first connects to the controller, it receives a set of default rules from the controller to support basic functionality. Table I lists the default rules.

TABLE I
DEFAULT RULES CONFIGURED ON ACCESS POINTS

Number	Priority	Input	Output	Rule
1	100	wlan0	local	type=0x888e
2	100	local	wlan0	type=0x888e
3	50	eth0	wlan0	dest=ff:ff:ff:ff:ff:ff
4	0	any	controller	match any


Name	MAC Address	Access Point	Access Level	Quick Action
 Test Device	00:01:02:03:04:05 Sample Manufacturer	Demo AP	<div>IoT - Learning</div> <div>Learning time remaining: 2 days</div>	<div>Full</div> <div>IoT - Learning</div> <div>IoT - Frozen</div> <div>Block</div>

Fig. 2. A screenshot of the stations page within the user interface.

Rules 1 and 2 support EAPOL packet forwarding for authentication using WPA2 to the *hostapd* service running on the access point. Rule 3 configures the forwarding of broadcast packets from the local area network to the IoT devices; this enables support for protocols such as DHCP. Rule 4 is a default rule that sends any packet which does not match a higher priority rule to the controller. Rule 4 is used by the controller to enrol any device seen for the first time with the FPE. It also enables the controller to handle packets which do not match other rules.

G. Station Default Rules

TABLE II
DEFAULT RULES PER IOT DEVICE

Number	Priority	Input	Output	Rule
5	24	wlan0	controller	type=tcp source=IoT MAC
6	24	eth0	controller	type=tcp dest=IoT MAC
7	24	wlan0	controller	type=udp source=IoT MAC
8	24	eth0	controller	type=udp dest=IoT MAC
9	23	wlan0	eth0	source=IoT MAC
10	23	eth0	wlan0	dest=IoT MAC

The controller requests a list of IoT devices from the web application. For each IoT device identified, the controller creates a set of rules. Rules 5-8 serve to redirect any TCP and UDP traffic to the controller. When this occurs, the controller extracts the information from the packet necessary to construct a rule. The controller then queries the SQL database to see if a matching rule has been recorded. If a matching rule is not found and the device is in IoT Learning mode, the controller creates the rule in the AP with a priority of 25. The priorities were chosen such that IoT Device rules would be processed in between Rules 3 and 4 from Table I, as rules are processed from highest to lowest priority. The learned rule is subsequently written to the database. In the event the device is in IoT Frozen mode and a matching rule is found, the same process is followed with the exception that a new rule is not written to the database. An idle timeout of 30 minutes is configured on IoT rules to reduce the set of rules maintained in the access point. The rule is removed from the flow table if a packet matching the rule has not been processed within the idle timeout.

Rules 9-10 allow other types of traffic between the station and the network such as ARP and ICMP. It would be possible

to expand on Rules 9-10 to make them more restrictive in the future.

H. DNS Support

Restricting rules to IP addresses may be problematic when devices use DNS, as the destination IP address may change on future connections. In order to address this challenge, the system dynamically tracks DNS responses and associates domain names with rules.

When the FPE creates an allow rule for UDP on port 53 on the inbound direction to an IoT device, it creates a flow rule with a set of actions. First, the packet is sent to the IoT device. Next, a VLAN tag is added to the packet and the packet is sent to the controller. When the controller receives a packet and parses the VLAN tag, it sends the domain names contained within the response to the web application. The web application records the IP addresses associated to the domain name. This allows the web application to compare new connection requests from IoT devices to both the initially-seen IP addresses and the IP addresses associated with the domain name in the datastore.

IV. IMPLEMENTATION

A. User Interface

We performed a full implementation of the FPE for evaluation purposes. The web application allows the user to view APs, stations, rules, and settings. Figure 2 is a screenshot from the stations page within the user interface. This page shows information such as the device, MAC address, manufacturer, and device state. The *Quick Action* buttons allow the user to transition the device between any of the states.

The user may select any station and view, modify, and delete the rules which restrict the device to given flows once in the IoT Frozen state. Usage information for each rule is gathered from the AP and displayed to the user. Figure 3 is a screenshot from the user interface and shows the bandwidth usage of one iPerf3 [22] session between a station and a server running on the LAN. The UI displays *Home Network* as the connection type as the IP address belongs in the RFC1918 [23] range. Otherwise, it would display *Internet*.

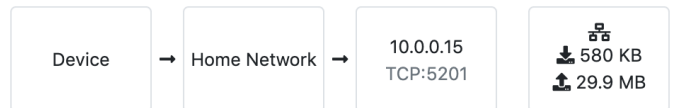


Fig. 3. Screenshot of rule representation and usage information.

Additionally, the user may modify the default learning time for IoT devices and the default state for new devices connecting to the access point. The system defaults to IoT Learning with 48 hours of learning time in order to provide secure default configurations.

B. Implementation Details

1) *Web Application*: The web application was written in Ruby on Rails, running on Ruby version 2.6.3 with the Rails framework version 6.0.0. This framework was chosen for cross compatibility across many platforms. The FontAwesome [24] library was used for displaying intuitive icons in the user interface.

2) *OpenFlow Controller*: The controller is a custom controller written in the Go programming language. It uses the *gopacket* [25] and *gofc* [26] libraries for packet serialization and OpenFlow communication. Go compiles to a native executable and allowed us to achieve a small memory footprint in comparison to other commonly used OpenFlow controllers based off of the Java programming language. Go, in combination with the aforementioned libraries, facilitates working with packet data and TCP connections, such as the OpenFlow datapath. The ability to customize all OpenFlow actions enables the controller to perform actions tailored to the FPE design.

C. Data Flow

Figure 4 is a sequence diagram representing the communication flow for packets being processed by the FPE for the purpose of learning or enforcing a policy. In this case, there are no rules for forwarding the packet installed in the Open vSwitch instance on the AP and in this example the packet matches Rule 4 from the previous section.

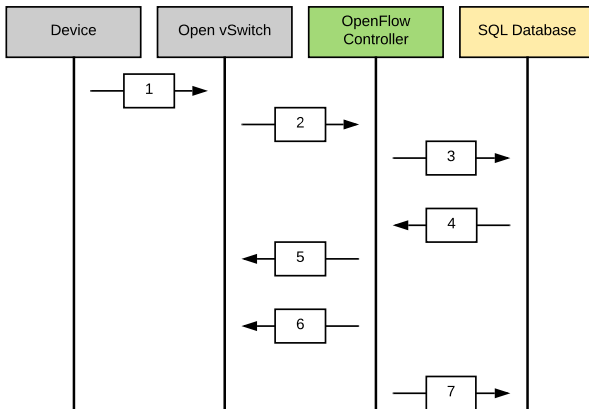


Fig. 4. Sequence diagram for learned packets.

The steps in the data flow are as follows:

- 1) The wireless device sends a packet to the AP.
- 2) The packet is sent from the AP to the controller via an OpenFlow PacketIn message.

- 3) The controller compares the MAC addresses contained within the packet to the list of IoT devices being managed by the controller. If a match is found, the controller queries the SQL database for a recorded rule matching parameters of the packet.
- 4) The SQL database responds to the query.
- 5) The controller constructs two OpenFlow Flow Modify messages. These two messages allow for bidirectional communication through the AP for matching packets. The Open vSwitch interface receives the Flow Modify messages and adds the flows to the flow table.
- 6) The OpenFlow Controller constructs an OpenFlow PacketOut message with the packet contained within the PacketIn message. This message is sent to the AP and the packet is sent from the data plane in the device.
- 7) The OpenFlow Controller records the rule in the SQL database.

Steps 1-6 are followed for devices in the IoT Learning or IoT Frozen states. Step 7 would not occur for a device in the IoT Frozen state. Additionally, steps 3-7 would not occur if a matching rule was not found in the event the device is in the IoT Frozen state.

D. Challenges

A number of challenges were faced during the implementation of the system.

1) *Packet Loss*: In an early implementation, the packet was passed from the OpenFlow Controller to the Web Application for processing and the PacketOut message was not utilized. This led to the first packet of each flow being lost and required the source to resend the packet. This led to significant delays as it relied on the source operating system to retry the connection after the flow was installed in the Open vSwitch. Persisting the packet in the OpenFlow Controller until a decision was made and then injecting it into the data plane of the Open vSwitch alleviated this issue.

2) *Bidirectional Data Flow*: Each rule in the policy results in two flows being installed in the Open vSwitch instance. This also requires the flow to be constructed from the perspective of the ingress and egress port. For example, assume a device is connecting through the AP to a server. When the ingress port is the wireless interface, the source MAC is the device and the destination MAC is the server. When the ingress port is the ethernet interface, the source MAC is the server and the destination MAC is the device. Application logic was written to create the least-permissive flow rule with respect to the ingress and egress port.

3) *Code Efficiency*: Packet processing must be efficient in order to minimize impact to devices. Significant effort went into the codebase and methods chosen to minimize the processing time within the OpenFlow Controller and the Web Application.

E. Deployment

The FPE supports two deployment models. It can be run on a server and used with any OpenFlow enabled AP in the network. It can also be deployed on a device configured to

run both the FPE and an AP, such as a Raspberry Pi. This flexible design allows the FPE to protect any IoT device which operates as a client to 802.11 wireless networks. Monitoring communications at the AP enables the system to prevent compromised IoT devices from attacking internal and external resources.

The ability to use FPE with any available hardware without customizing any protocols satisfies the goal of having a vendor agnostic solution. The user can elect to use FPE with the vendor of their choice or run it on low-cost hardware. Additionally, the goal for compatibility is achieved through the use of OpenFlow and well-defined protocols.

V. EVALUATION

We performed three different evaluations of the FPE. First, we conducted performance testing of the system. Next, we performed an analysis of IoT devices using the FPE. Finally, we discuss the merits of the system in terms of security and usability.

A. Performance Testing

1) *Connection Testing*: Tests were performed using a Raspberry Pi 4 as an AP. Each test was performed using the same AP in different configurations.

- *Topology One* uses the Raspberry Pi as an AP running *hostapd* in bridge mode, without Open vSwitch.
- *Topology Two* uses the Raspberry Pi with Open vSwitch forwarding rules configured. This represents a system with the rules populated in the flow table.
- *Topology Three* runs the FPE in learning mode and Open vSwitch on the Raspberry Pi.
- *Topology Four* runs the FPE in learning mode on a server separate from the Raspberry Pi AP running Open vSwitch.

In order to reset the learning mode for *Topologies Three* and *Four*, the learned rule was erased from the database and the flow table cleared from the Open vSwitch instance. Each measurement was performed 10 times and the average result was taken. The Raspberry Pi is configured using the 2.4GHz 802.11G band. The Unix time utility was used in combination with *iPerf3* sending a 500 byte message to represent the delay a user would encounter while using the FPE to initiate a new connection. Table III lists the results.

TABLE III
AVERAGE CONNECTION ESTABLISHMENT TIME PER TOPOLOGY

Topology	Time (milliseconds)
One	106
Two	104
Three	145
Four	114

Based on the results, we see that there is a cost to using the FPE module depending on the deployment method used. *Hostapd* and Open vSwitch with flow rules installed are close with 106 and 104 milliseconds on average, respectively. Running the FPE on the Raspberry Pi caused the initial connection

to take 145 milliseconds on average, which is an increase of 41 milliseconds over the best case. When running the FPE on a separate server, the initial connection time was on average 8 milliseconds slower than *hostapd* and 10 milliseconds slower than OpenFlow with rules already configured. It should be noted this is for new connections that are not already in the flow table. The FPE installs flow rules with an idle timeout of 30 minutes to enable quick reuse of existing rules.

The connection delay is less significant when put in context of an ongoing transmission. While there is a slight delay in setting up the flows for the first packet, subsequent packets would follow the timing of *Topology Two* as the flows would already be established.

2) *Scalability Testing*: We performed scalability testing of the FPE focusing on *Topologies Three* and *Four* described in the previous experiment. This was accomplished by writing a custom testing framework in Go utilizing *goroutines*. A *goroutine* is a concurrency feature in Go. For the purpose of scalability testing, the framework was configured to generate packets which were guaranteed to result in new rules being learned by the system.

The physical ethernet and wireless interfaces were replaced with virtual interfaces in the Open vSwitch instance. The testing framework creates two primary *goroutines*. The producer injects the desired number of packets on one virtual interface and records the time of packet creation in a shared memory location. When the consumer thread receives a packet, it creates a new *goroutine*, records the current time and goes into a waiting state. Once all of the packets have been received, the consumers read from the shared memory location and record the duration of time between packet injection and packet reception.

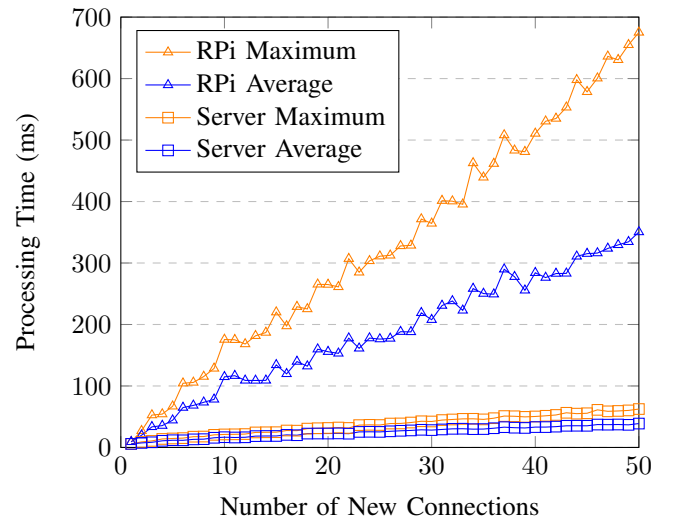


Fig. 5. Individual packet processing time during scalability testing.

Figure 5 displays the experiment results. Each experiment was repeated 10 times and the average result was recorded. For the maximum cases, the average of the maximums was recorded.

The FPE results in higher delays per packet as the number of new concurrent connections scales in the Raspberry Pi. It

should be noted that this is the worst-case scenario in which the FPE receives a large number of new connections with no existing flow rules existing in the Open vSwitch instance. The results of this experiment show the FPE can handle a moderate number of new concurrent connections at a given time when running on the Raspberry Pi and can scale to a significant number of new connections when running on a separate server.

3) *Blocked Connection Testing*: We also performed an evaluation of system performance while blocking some connections based on *Topologies Three* and *Four*. The performance of the system was evaluated while measuring the packet processing time as an increasing number of connections were blocked.

In each test, 50 new connections were created using the same scalability testing framework as previously described. The device was configured to be in the IoT Frozen state. Rules were pre-created in the database but were not created as flows in the Open vSwitch instance. This allowed us to evaluate the performance of the system in the case where rules have been learned by the FPE.

The number of blocked connections were increased from no blocked connections to 25 blocked connections. As blocked connections were introduced, they were evenly distributed in the permitted connections. This allows a better representation of overall performance of the system by preventing the clustering of blocked or permitted connections being processed by the FPE. For example, every second connection was blocked when testing 50 new connections with 25 connections being blocked.

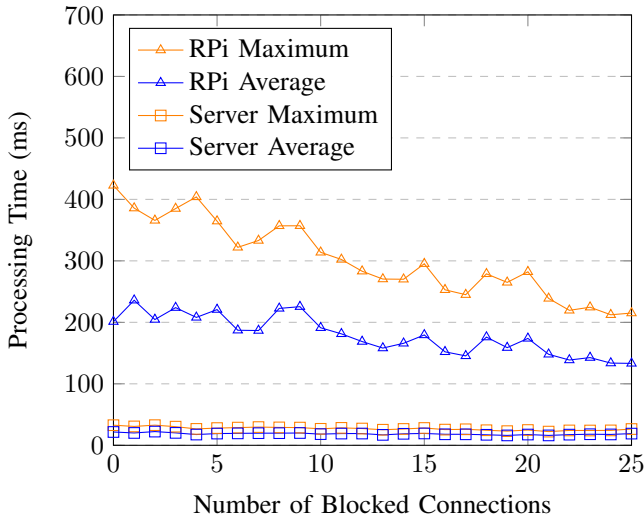


Fig. 6. Individual packet processing time during blocked connection testing.

Figure 6 shows the results of this experiment. As we can see, the system can efficiently block unknown connections while permitting learned rules to be installed in the Open vSwitch instance and injecting the packet into the data plane. As the number of blocked connections increases, the average and maximum packet processing time of permitted packets decreases. This is a result of the system performing fewer actions to process the blocked packets. Once the FPE determines a packet is not to be permitted nor learned, as the device is in

the IoT Frozen state, the processing pipeline can immediately stop processing. This allows for more resources to be spent on processing the next packet in the FPE queue. This performance improvement is noticeable when the FPE is running on the server than when it is running on the Raspberry Pi.

B. IoT Device Analysis

1) *IoT Scale*: We connected an IoT scale to the FPE for analysis. The scale uploads the user's weight and other information for display in a mobile phone application. Four flow rules were recorded for the device. The four flows were: DHCP, two DNS lookups, and one connection to an upstream server.

The scale performed DNS lookups against the DHCP-provided DNS server and the 8.8.8.8 DNS server. The device failed to operate when blocked from communicating with 8.8.8.8. The scale remained functional when blocking the device from the DHCP-provided DNS server and allowing it to access the 8.8.8.8 DNS server. This result is interesting as the device does not respect attributes provided by the DHCP server and requires access to what appears to be a hardcoded IP address. This device would lose functionality in the event of upstream connection issues to 8.8.8.8, regardless of the availability of alternative DNS servers. Despite the unexpected DNS behaviour, the device did not appear to create many connections to the internet beyond the intended purpose of the device.

2) *Smart TV*: We connected a Smart TV from a well known manufacturer to the FPE. Smart TVs have been subject to criticism and privacy concerns [27]. The goal of this analysis was to allow the TV stream content from a popular streaming service and block all other connections. We obtained the list of domain names from the streaming service website. We created a rule set for the streaming service domains and attached the rule set to the policy for the TV. Since the domains provided by streaming service may use subdomains, we configured the rule set to allow for any subdomain of the whitelisted domain names to be used.

We noted that the Smart TV attempts many connections to services beyond those necessary for streaming content. Based on the DNS requests, these connections include connections to social media services, log aggregation services, advertising services, and other vendor-managed servers. However, we were able to successfully block these connections and create a least-permissive policy for the device. In the end, the policy had nine rules: DHCP, DNS, and seven domain names for the streaming service.

This demonstrates the value of configurability within the FPE to address privacy concerns while still enabling an IoT device to perform desired functionality.

C. System Discussion

1) *Security*: As we have seen, there is a cost to running the FPE in terms of connection setup. The cost is reduced as flows are reused and the initial connection time is taken within context of a larger transfer of data.

Suppose an attacker compromised an IoT device in each of the topologies, and, in the case of the FPE topologies, the device is in the IoT Frozen state. In the first two topologies, the attacker would have full access to the local network as there is no policy enforcement on the network traffic. This device could be utilized to pivot and attack other internal and external network resources.

The attacker would face a significant reduction in the ability to pivot within the network in the cases where the FPE is running. The compromised IoT device would be restricted to communication to ports and services that are defined in the policy for the IoT device. For example, if the IoT device had never communicated via SMBv1, it would not be possible for an attacker to utilize TCP ports 139 and 445 to execute the exploit known as *ETERNALBLUE* [28] if vulnerable devices were located in the same LAN.

2) *Usability*: As we have previously discussed, the goal of the FPE is to provide a vendor agnostic solution to securing smart homes. In order to effectively protect home networks, the solution must be easy to setup and maintain.

The user must perform tasks such as configuring an SSID and passphrase. The default FPE configuration automatically protects and limits IoT devices from accessing network resources beyond their initial connection requirements. Advanced users can further restrict devices to communication to services of their choice, as we saw in the Smart TV discussion.

The solution is capable of running on a low cost Raspberry Pi. The FPE workload can be shifted to another computer on the network if higher performance is desired.

VI. CONCLUSION

In this paper, we have proposed a new system for securing IoT devices within home networks. The FPE module is able to protect the network by restricting IoT devices to least permissive rule sets. This prevents compromised devices from accessing internal and external network resources and reduces the capability of an attacker to utilize a compromised IoT device. Additionally, the Flow Policy Enforcer can be deployed on low-cost hardware, does not rely on cloud services, and does not rely on custom protocols. This enables the FPE to be a plug-and-play solution that can be easily be applied to home networks.

We focused solely on preventing a compromised device from accessing TCP or UDP resources within a network. The FPE could be expanded to be even more restrictive and allow devices to only use the protocols they request during the learning period. Furthermore, integrating a protocol such as RFC 8520 [29] would allow for FPE to pre-populate policies and place them in the IoT Frozen state without any learning time required.

REFERENCES

- [1] Cisco, "Internet of things at a glance," June 2016. [Online]. Available: <https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf>
- [2] K. Huang *et al.*, "Systematically understanding the cyber attack business: A survey," *ACM CSUR*, vol. 51, no. 4, pp. 70:1–70:36, Jul. 2018. [Online]. Available: <http://doi.acm.org/10.1145/3199674>
- [3] Markets and Markets, "Cybersecurity market by solution, service, security type, deployment mode, organization size, industry vertical, and region - global forecast to 2023," September 2018. [Online]. Available: <https://www.marketsandmarkets.com/Market-Reports/cyber-security-market-505.html>
- [4] K. Neupane, R. Haddad, and L. Chen, "Next generation firewall for network security: A survey," in *Proc. IEEE SoutheastCon 2018*, April 2018.
- [5] Talos, "New vpnfilter malware targets at least 500k networking devices worldwide," May 2018. [Online]. Available: <https://blog.talosintelligence.com/2018/05/VPNFilter.html>
- [6] J. Matherly, "Complete guide to shodan," *Shodan, LLC (2016-02-25)*, 2015.
- [7] K. Boeckl *et al.*, "Considerations for managing internet of things (iot) cybersecurity and privacy risks," National Institute of Standards and Technology, Tech. Rep., 2018.
- [8] M. Frustaci *et al.*, "Evaluating critical security issues of the iot world: Present and future challenges," *IEEE IoT-J*, vol. 5, no. 4, pp. 2483–2495, Aug 2018.
- [9] E. Kritzinger and S. von Solms, "Home user security- from thick security-oriented home users to thin security- oriented home users," in *Proc. SAI*, Oct 2013, pp. 340–345.
- [10] T. T. Doan *et al.*, "Towards a resilient smart home," in *Proc. ACM IoT S&P*. New York, NY, USA: ACM, 2018, pp. 15–21. [Online]. Available: <http://doi.acm.org/10.1145/3229565.3229570>
- [11] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [12] The Open Networking Foundation, "OpenFlow Switch Specification," September 2012.
- [13] M. N. Aman, K. C. Chua, and B. Sikdar, "Secure data provenance for the internet of things," in *Proc. ACM IoTPTS*. New York, NY, USA: ACM, 2017, pp. 11–14. [Online]. Available: <http://doi.acm.org/10.1145/3055245.3055255>
- [14] Y. Meidan *et al.*, "Profilot: A machine learning approach for iot device identification based on network traffic analysis," in *Proc. ACM SAC*. New York, NY, USA: ACM, 2017, pp. 506–509. [Online]. Available: <http://doi.acm.org/10.1145/3019612.3019878>
- [15] B. Danev, D. Zanetti, and S. Capkun, "On physical-layer identification of wireless devices," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 6:1–6:29, Dec. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2379776.2379782>
- [16] J. Vestin *et al.*, "Cloudmac: Towards software defined wlans," *ACM SIGMOBILE MC2R*, vol. 16, no. 4, pp. 42–45, Feb. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2436196.2436217>
- [17] P. Porras *et al.*, "A security enforcement kernel for openflow networks," in *Proc. ACM HotSDN*. New York, NY, USA: ACM, 2012, pp. 121–126. [Online]. Available: <http://doi.acm.org/10.1145/2342441.2342466>
- [18] C. R. Taylor *et al.*, "On the feasibility of cloud-based sdn controllers for residential networks," in *Proc. IEEE NFV-SDN*, Nov 2017, pp. 1–6.
- [19] S. Shirali-Shahreza and Y. Ganjali, "Protecting home user devices with an sdn-based firewall," *IEEE T-CE*, vol. 64, no. 1, pp. 92–100, Feb 2018.
- [20] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home iot using openflow," in *Proc. ARES*, Aug 2016, pp. 147–156.
- [21] "Redis," Accessed 2019. [Online]. Available: <https://redis.io/>
- [22] "iperf - the ultimate speed test tool for tcp, udp and sctp," Accessed 2019. [Online]. Available: <https://iperf.fr/>
- [23] Y. Rekhter *et al.*, "Address allocation for private internets," Tech. Rep., 1996.
- [24] "Fontawesome," Accessed 2019. [Online]. Available: <https://fontawesome.com>
- [25] "Gopacket," Accessed 2019. [Online]. Available: <https://github.com/google/gopacket>
- [26] "gofc," Accessed 2019. [Online]. Available: <https://github.com/Kmotiko/gofc>
- [27] J. Brodtkin, "Smart tvs are invading privacy and should be investigated, senators say," July 2018. [Online]. Available: <https://arstechnica.com/tech-policy/2018/07/smart-tvs-are-invading-privacy-and-should-be-investigated-senators-say/>
- [28] "Smb exploited: Wannacry use of eternalblue," Accessed 2019. [Online]. Available: <https://www.fireeye.com/blog/threat-research/2017/05/smb-exploited-wannacry-use-of-eternalblue.html>
- [29] E. Lear, R. Droms, and D. Romascanu, "Manufacturer usage description specification," Tech. Rep., 2019.