Authenticating Endpoints and Vetting Connections in Residential Networks

Yu Liu[1], Curtis R. Taylor[1,2] and Craig A. Shue[1]
[1]Worcester Polytechnic Institute
{yuliu, crtaylor, cshue}@cs.wpi.edu
[2]Oak Ridge National Laboratory
taylorcr@ornl.gov

***Abstract*—The security of residential networks can vary greatly. These networks are often administrated by end-users who may lack security expertise or the resources to adequately defend their networks. Insecure residential networks provide attackers with opportunities to infiltrate systems and create a platform for launching powerful attacks. To address these issues, we introduce a new approach that uses software-defined networking (SDN) to allow home users to outsource their security maintenance to a cloud-based service provider. Using this architecture, we show how a novel *network-based* two-factor authentication approach can be used to protect Internet of Things devices. Our approach works without requiring modifications to end-devices. We further show how security modules can enforce protocol messages to limit the attack surface in vulnerable devices. Our analysis shows that the system is effective and adds less than 50 milliseconds of delay to the start of a connection with less than 100 microseconds of delay for subsequent packets.**

***Keywords*-software-defined networking; residential networks; two-factor authentication**

I. Introduction

Most residential networks are created and managed by end-users that may lack computer security expertise. These users may employ weak security practices, such as not changing default usernames and passwords on devices, which allow attackers to easily compromise systems on the network. Further, the end-devices themselves may introduce weaknesses, such as failing to encrypt traffic or to patch vulnerabilities. This is a particular challenge in Internet-enabled embedded devices, commonly referred to as "Internet of Things" (IoT) devices, in which only the device manufacturer can deploy updates. As a result, residential networks may also be home to compromised devices that enable attackers to persist, pivot to other devices, and control the home's environment.

One way to combat the insecurity of residential networks is to outsource the security management to experts. We envision a service provider that hosts systems outside the network, potentially in a cloud data center, to manage the devices inside the residential network. With recent developments in software-defined networking (SDN), it is now possible for an off-site controller to remotely manage a network's infrastructure [12].

To capitalize on the benefits of outsourced network management, we need new approaches to distinguish legitimate traffic from potentially malicious activity. In this work, we examine whether 1) the initiator is an authorized party 2) the exchanged messages conform to expectations (i.e., they follow protocol and historical interactions). When properly implemented, these requirements could greatly reduce a network's attack surface.

Our work revolves around two research questions: *What mechanisms can effectively authenticate client communication without requiring changes to server or client applications? What are the performance overheads associated with implementing this in consumer-grade hardware?* We explore these questions by installing OpenFlow software on an existing consumer-grade router and install custom modules on an OpenFlow controller to implement device-agnostic, network-level multi-factor authentication and strict enforcement of IoT device communication using a historically-derived rules.

We make the following contributions in this work:

1. Enable Device-Agnostic Authentication: We link the Google Authenticator verification system with an OpenFlow controller module so that clients that successfully authenticate are granted temporary network access to IoT devices. Unauthenticated devices lack network access to reach the destination resources, thus preventing attacks that, for example, could subvert built-in authentication checks. For devices with their own authentication systems, our approach essentially adds a second factor of authentication (the possession of a shared key). The approach is effective at blocking attackers and, for roughly 90% of new connections, the approach adds less than 50 ms of delay.

2. Enforce IoT Device Protocols: IoT devices are typically purpose-built and provide a small set of services via relatively simple protocols. As a result, we can exhaustively enumerate the packets used by these IoT devices during standard operation. We build a state machine model of each protected device and only allow packets to the device in which the payload follows known transitions within that state machine. This prevents attacks such as buffer overflows or other speciallycrafted messages that could exploit a vulnerability on the device. The approach can detect and discard packets that violate the protocol, preventing attacks from reaching the device, usually with less than 200 ms of delay.

II. Related work

The security challenges of IoT devices have been explored from many angles. Babar *et al.* [1] explored the security model of IoT devices and the threats that they face. Zhang *et al.* [18] explore the challenges for IoT device research and the privacy challenges that some devices introduce. Kolias *et al.* [6] investigated the role that vulnerable IoT devices can play in large-scale attacks, such as those enabled by the Mirai botnet. While the topic of IoT device security is being studied by the community, little work focuses on residential networks.

In a position paper, Feamster [3] proposed outsourcing residential network security to cloud-based service providers. He recognized the challenges of educating all residential users on how to properly secure their networks and proposed using OpenFlow to secure these devices. Later work explored the potential for such functionality, but focused on scenarios that require Internet Service Provider support [5], [11]. Unfortunately, few ISPs have offered such support for customers and the reliance on their support may limit deployability. We use existing consumer routers and cloud-hosted OpenFlow controllers to eliminate a reliance on ISP support.

Other work has focused on enterprise-centric solutions for outsourcing network management. Sherry *et al.* [10] introduced APLOMB, an approach that used a specialized network router to redirect traffic to cloud-hosted middleboxes. APLOMB's need for specialized hardware, the retransmission of each network packet received to a middlebox, and the need for DNS modifications make the approach infeasible in many residential networks. With residential networks, such proxying is not needed.

The most closely related work is our own prior work in which we introduced a mechanism for residential users to communicate through a cloud-based server to avoid exposing their IP addresses while using voice-over-IP software, like Skype [15]. That approach helped to protect users from crippling denial-of-service attacks while engaged in other activity, such as competitive online gaming. In our TLSDeputy project [14], we built upon that infrastructure and focused on how to protect communication when using the TLS protocol. That system examined server TLS certificates and checked for revocations, a step omitted by some browsers at the time. In this paper, we focus on protecting IoT devices.

III. Threat Model

Residential IoT devices may have unpatched vulnerabilities, default passwords, or simply weak passwords. These devices may have built-in authentication mechanisms; however, we consider these built-in mechanisms to be second-factor authentication which may offer only limited security benefits in practice. Further, within the LAN, some devices may implicitly trust all other devices and communicate without any authentication mechanism at all. This approach is common for IoT devices and video players, in which physical presence and/or knowledge of wireless network keys is considered sufficient evidence of authorization.

We assume our router is directly connected to each end-point device and there are no routes to these devices which bypass our router. We consider our authentication server, OpenFlow controller, and modified router to be part of the trusted computing base (TCB). All other devices on the network are considered untrusted and potentially compromised.

IV. Background

Multi-factor authentication requires two or more separate forms of authentication information as proof of identity. Common forms of authentication include passwords, possession of random values from previous interactions (e.g., web browser cookies), possession of cryptographic keys, possession of specific hardware security tokens, or information transmitted out-of-band, such as messages to another computing device. In our approach, the second factor is similar to the possession of a cryptographic key since authentication requests require appropriate keying data to be present on the initiating client.

Our approach uses software-defined networking (SDN), in which routing decisions can be made in software programmatically rather than using a pre-defined look-up table. In our scenario, each device is connected to a consumer-grade router running Open vSwitch (OVS) [7], a popular OpenFlow switch implementation. The OVS router locally maintains a flow table consisting of a five-tuple (IP$_{src}$, IP$_{dst}$, Port$_{src}$, Port$_{dst}$, and transport protocol) along with the associated action (e.g., drop or forward to a given interface port). If a packet arrives with fields that do not match an existing entry, the OVS router elevates new connection requests to a cloud-hosted OpenFlow controller. The controller runs software modules that can be used to make the appropriate decision and return the instructions to the OVS router. The controller is typically involved only in the initial packets in a given flow. However, in some circumstances, the controller may direct the OVS router to tunnel packets associated with a given flow through a *middlebox*, which is a hardware or software system that can perform arbitrary inspection and manipulation of a packet while en route to the destination. For convenience, we co-locate the middleboxes and controller in our experiments.

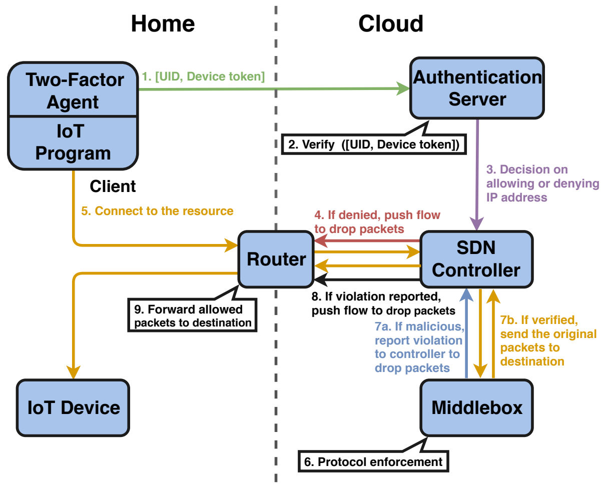
Home
Cloud
Two-Factor Agent
IoT Program
Client
1. [UID, Device token]
Authentication Server
2. Verify ([UID, Device token])
3. Decision on allowing or denying IP address
4. If denied, push flow to drop packets
5. Connect to the resource
Router
SDN Controller
8. If violation reported, push flow to drop packets
9. Forward allowed packets to destination
7a. If malicious, report violation to controller to drop packets
7b. If verified, send the original packets to destination
IoT Device
Middlebox
6. Protocol enforcement

Fig. 1. The OVS router enforces two-factor authentication. A client must send an authentication request to the authentication server before interacting with an IoT device. The authentication server sends the result to the SDN controller, which adds flows to the OVS router to allow or deny the subsequent request. The traffic may traverse a middlebox that can ask for additional flow changes.

V. Approach: Vetting New Flows

Our approach of vetting new flows can enhance security in multiple ways. We provide an example of second-factor authentication for IoT devices and protocol enforcement of IoT device communication. These approaches do not require modifications to the IoT devices, which may be impractical in some settings.

A client must send a request and resource-specific authentication token, potentially via a separate application, before connecting to a given IoT device, as shown in Figure 1. If the authentication server is able to verify the client's token, it notifies the OpenFlow controller of the result. When the client attempts to access the IoT device, the OVS router elevates the request to the controller. If the controller has received an authentication notification from the server, it orders the OVS router to cache a rule allowing that client IP address to temporarily reach the IoT device. Otherwise, it orders the OVS router to drop the flow's packets.

Some IoT devices have a more restricted API than generic computing devices. This restricted API allows us to enhance our middleboxes to implement communication models for each IoT device type and use these models to restrict the type of messages allowed when communicating with each IoT device. We build regular expressions for matching the packet payload that constrain the messages to the small set that we observed during a training phase. These expressions allow the variability needed for session-specific values. In Figure 2, we provide an example of this approach for the payload between a smartphone and an IoT device. The request orders the device to activate or deactivate.

```
Content-Type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1
#SetBinaryState"
Content-Length: 383
Host: 192.168.3.157:49153
User-Agent: CyberGarage-HTTP/1.0
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.
org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/">
 <s:Body>
  <u:SetBinaryState xmlns:u="urn:Belkin:
service:basicevent:1">
   <BinaryState>1</BinaryState>
   <Duration></Duration>
   <EndAction></EndAction>
   <UDN></UDN>
  </u:SetBinaryState>
 </s:Body>
</s:Envelope>
```

Fig. 2. Example payload from the tested Smart Switch (some line-wraps were added to fit within a column). The interaction is via a SOAP request using XML encoding allowing the payload and structure to easily be enforced programmatically.

Our regular expressions allow these two states and allows variation in the content length associated with the packet. The expressions require other fields to remain static. This approach enables the middlebox to allow only authorized messages and to easily detect unauthorized messages. In the instance an unauthorized message is detected, the middlebox can discard the packet and alert the OpenFlow controller. The controller can then alter the flow entry at the OVS router to discard subsequent packets in the flow.

VI. Implementation

We implement our system using a TP-LINK Archer C7 consumer-grade router to directly connect our protected end-points. We install OpenWrt [16] with Open vSwitch (OVS) [8] on the router. The router communicates via the OpenFlow protocol to a local controller running the Floodlight [9] Java-based OpenFlow controller software. The controller runs on Mac laptop that is connected to one of the OVS router's LAN ports. The controller laptop has four 2.6GHz cores and 16 GB RAM. We add a custom module to the Floodlight controller that subscribes to all packets processed at the controller.

When hosts communicate within the same subnet, the OVS router normally connects the two using its hardware switch, bypassing the need for the packet to be examined in software. However, our approach requires that each flow be examined in order to provide access control, so we needed to avoid this behavior. We configured the router to place each of its physical interface ports on a separate virtual LAN (VLAN) and allowed the router's main processor to route packets across VLANs. We likewise use wireless isolation to VLAN radio communication [17].

Our two-factor authentication server uses the Google Authenticator library [4] in a C program that runs on an Ubuntu 14.04 laptop that is connected via Ethernet to the OVS router. That laptop has four 2GHz cores and 8GB of RAM. Using a one-way hash function based on the current time and a pre-shared secret, the server dynamically generates one-time use secret tokens for each registered device. Each registered device obtains a copy of the pre-shared secret. With this value, the client can use the current time and pre-shared secret to compute its own version of the hash output and send it to the server, which can verify its authenticity by comparing it with its own locally computed value. Upon successful verification, the authentication server communicates with the controller over a network socket to send the client's IP address and authorized destination. The controller stores a record for the authentication result to authorize subsequent flows to that destination for a short period.

A. Protecting IoT Devices

In our experiments on IoT devices, we focus on the Belkin WeMo Smart Switch [2]. The Smart Switch is a electrical power outlet adapter that plugs into a standard electrical outlet and exposes another outlet that devices, such as a lamp, can use. The Smart Switch can be controlled through a smartphone application in both the iOS and Android operating systems. Through the smartphone application, a user can activate or deactivate the power flow, gaining the ability to control the power to the connected device remotely.

In testing, we use the WeMo smartphone application and manually toggle the switch setting 1,000 times. For the two-factor authentication system, we automated our experiments using a Python script using Google Authenticator on a wirelessly-connected Mac Mini, which had two 2.6 GHz cores and 8 GB of RAM.

To model the types of communication between the smartphone and IoT device, we consider a state machine. To move from the initial state, the smartphone must send one of a small set of valid messages. Upon receiving such a message, we advance the state of the device and consider the new messages available. By continually repeating this process, we are able to block any messages that are not valid for the given state and prevent any malicious messages from arriving. We build our state machine over a series of runs with an uncompromised device and then use the state machine in a middlebox to enforce its actions. Each path through the state machine can be considered a packet sequence and each transition is determined by matching the payload of a given packet with a known regular expression for that packet.

In Figure 2, we provide an example of an initial request with HTTP payload. It begins with the HTTP header, followed by the *envelope* structure. The *content-length*, *HOST*, and *BinaryState* fields may vary across devices and actions. We build regular expressions that match these dynamic parts while requiring exact matches for the static components. When a packet matches a sequence, we push the packet information plus a time stamp into a packet list. The timestamp allows us to reset our state machine after a timeout occurs.

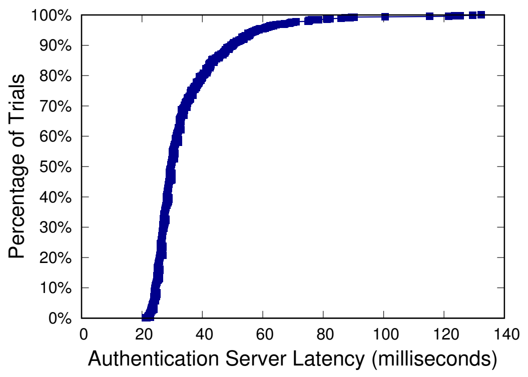

Fig. 3. Two-factor authentication request delay (1,000 trials).

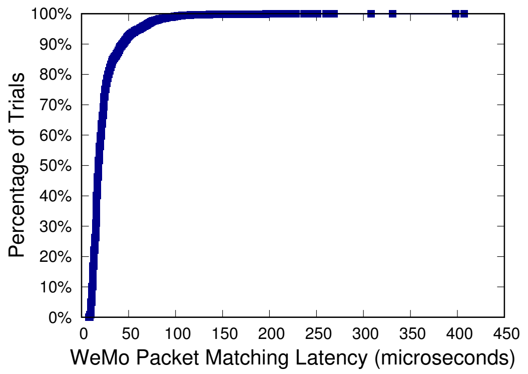

Fig. 4. Delay from the protocol enforcement module for the IoT device traffic. These results are from 67,335 packets produced during 1,000 trials. We omitted 30 outliers (0.04% of data) for readability.

VII. Security and Performance Evaluation

To evaluate the security of our approach, we made connections to our Smart Switch IoT device. To create legitimate behavior, we followed the protocol of providing two-factor authentication connections and then connected to the device. Likewise, the interactions with the WeMo device conformed to the protocol expected for that device. In the two-factor authentication approach, we create malicious connections simply by initiating a connection without completing the two-factor authentication steps. In the protocol enforcement experiment, we send packets that contained random bytes as payload to

TABLE I
Security evaluation results for each approach

Approach	Request	Trials	Allowed	Blocked
Two-Factor Authentication	legitimate	20	20	0
	malicious	20	0	20
IoT Protocol Enforcement	legitimate	20	20	0
	malicious	20	0	20

create malicious packets. Each approach was tested in isolation, with only one module active.

In Table I, we show the results of these experiments. In each case, the approach allowed the authorized connections and denied the connections that were malicious. These results show that each of these approaches can offer significant security advantages. While the IoT protocol enforcement system must be customized to each type of IoT device, the two-factor authentication are more generally applicable.

For our performance evaluation, we consider the performance of the two-factor authentication system and the IoT protocol enforcement module. We consider the performance of each individually. Our results are for a controller and middlebox in the LAN. Additional propagation delays would be incurred for remote deployments.

To determine the timing overheads of the two-factor authentication system, we used a script on the client that recorded the results of the `time.time()` function in Python, which returns the current Unix epoch time with microsecond resolution. We measured the time before we issued the request to the authentication server and after we received the server's response. By subtracting these values, we could determine the amount of time elapsed. In Figure 3, we see that the majority of requests are satisfied within approximately 30 milliseconds and 90% are satisfied within about 50 milliseconds. These delays are unlikely to be noticeable to an end user.

For all the modules running on the controller, including the IoT protocol enforcement, we used the `System.nanotime()` function in Java to record the timestamp when the module started and when it ended and calculated the overhead as the difference.

For the IoT protocol enforcement, we show our results in Figure 4. We see the system processed over 99% of packets in less than 100 microseconds. This process is particularly fast since there are significant static elements to the packets that can be used to distinguish legitimacy.

VIII. Discussion

Our work uses a controller and middlebox located in the LAN. When these components are further away, latency could become a concern. However, our prior work has explored residential connectivity with public cloud data centers and found that over 90% of residential networks in the United States were within 50 milliseconds of a public data center [12]. That study found such latency would have only a small impact on the user experience, even in applications like web browsing.

The two-factor authentication system could be automated using a resource like the `netfilter_queue` library which could intercept a packet, send the authentication request, and requeue the packet. Prior work has used a similar technique [13]. This would essentially create a device-level authorization system.

Our IoT device state machine was based off of a Smart Switch with a limited API. Other IoT devices may have more functionality and require more involved state machines to replicate their protocols. However, other modern IoT devices may have constrained behavior.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1651540.

References

[1] Sachin Babar, Parikshit Mahalle, Antonietta Stango, Neeli Prasad, and Ramjee Prasad. Proposed security model and threat taxonomy for the Internet of Things (IoT). In *International Conference on Network Security and Applications*, pages 420–429. Springer, 2010.

[2] Belkin International, Inc. Wemo switch smart plug. http://www.belkin.com/us/p/P-F7C027, 2018.

[3] Nick Feamster. Outsourcing home network security. In *ACM SIGCOMM Workshop on Home Networks*, pages 37–42, 2010.

[4] Thomas Habets. Google Authenticator PAM module. https://github.com/google/google-authenticator-libpam, February 2018.

[5] Kamran Riaz Khan, Zaafar Ahmed, Shabbir Ahmed, Affan Syed, and Syed Ali Khayam. Rapid and scalable ISP service delivery through a programmable middlebox. *SIGCOMM Computer Communication Review*, 44(3):31–37, July 2014.

[6] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[7] Linux Foundation. Open vSwitch. https://www.openvswitch.org.

[8] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of Open vSwitch. In *NSDI*, pages 117–130, 2015.

[9] Project Floodlight. Project Floodlight. http://www.projectfloodlight.org/floodlight/, 2018.

[10] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 2012.

[11] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani. Network-level security and privacy control for smart-home IoT devices. In *Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 163–167, 2015.

[12] Curtis R. Taylor, Tian Guo, Craig A. Shue, and Mohamed E. Najd. On the feasibility of cloud-based SDN controllers for residential networks. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017.

[13] Curtis R Taylor, Douglas C MacFarland, Doran R Smestad, and Craig A Shue. Contextual, flow-based access control with scalable host-based SDN techniques. In *IEEE INFOCOM*, 2016.

[14] Curtis R Taylor and Craig A Shue. Validating security protocols with cloud-based middleboxes. In *IEEE Conference on Communications and Network Security (CNS)*, pages 261–269, 2016.

[15] Curtis R. Taylor, Craig A. Shue, and Mohamed E. Najd. Whole home proxies: Bringing enterprise-grade security to residential networks. In *IEEE International Conf. on Communications*, 2016.

[16] The LEDE Project. Welcome to the OpenWrt project. https://openwrt.org, February 2018.

[17] Pierre Trudeau and Stephane Laroche. Configurable quality-of-service support per virtual access point (vap) in a wireless lan (wlan) access device, November 11 2014. US Patent 8,885,539.

[18] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. IoT security: ongoing challenges and research opportunities. In *IEEE Conference on Service-Oriented Computing and Applications*, 2014.