Flow Table Modification Using Behavioral-Based Fingerprinting Technique to Facilitate Zero Trust Identity Management and Access Control

Vinton Morris
Cybersecurity Assurance and Policy Center
Morgan State University
Baltimore, MD 21251
Email: vinton.morris@morgan.edu

Kevin Kornegay
Department of Electrical and Computer Engineering
Morgan State University
Baltimore, MD 21251
Email: Kevin.kornegay@morgan.edu

2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) | 979-8-3315-0769-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/CCWC62904.2025.10903861

***Abstract*—Traditional security mechanisms and architectures are falling short, impacting both organizations and end users alike. The rapid growth and variety of Internet of Things (IoT) devices have significantly expanded the attack surface and increased network vulnerabilities. In recent years, Zero Trust (ZT) has gained traction in its attempt to solve the inherent flaws of the traditional perimeter-centric security architecture. While there have been great strides in supporting users, applications, and general-purpose computing devices (GPD), current ZT architectures lack the necessary capabilities to support IoT devices. In this paper, we present a method of authentication that profiles the network traffic and builds unique fingerprints for IoT devices. The fingerprints are a representation of the behavior of the IoT devices. We then compare the on-demand generated fingerprints of a device attempting to communicate to those in the database and update the flow tables of a software-defined networking (SDN) switch, accordingly ensuring that only devices with matching profiles can communicate on the network. We argue that the predictive nature of IoT device traffic and their limited protocol set make them a candidate for this method of authentication and access control that can satisfies the Zero Trust Architecture's identity management component.**

1. Introduction

Traditional security mechanisms and architectures are unable to keep up with today's dynamic environments. The frequency and severity of security breaches [1], [2], [3], [4], [5] have significantly increased in recent years, causing many organizations to rethink their security operations. Zero Trust Architecture (ZTA), as illustrated in Figure 1, promises a solution to the various security challenges of the perimeter-based security model. Zero Trust (ZT) is a cybersecurity strategy that secures an organization by eliminating implicit trust and ensuring continuous verification at every stage of an access request [6], [7], [8]. Based on the principle of "never trust, always verify," [9], [10], [11], [12], [13], ZT protects modern environments by using robust authentication methods, leveraging network segmentation, preventing lateral movement, and simplifying granular "least access" policies [14], [15]. The creation of ZT was in direct response to traditional security models, which operated on the flawed assumption that all devices and users on the internal network were trustworthy. This implicit trust often gives network users, including threat actors and malicious insiders, the ability to move laterally and access or ex-filtrate sensitive data due to a lack of granular security controls. This has become a serious concern for many organizations. These devices have become staples of our everyday personal and professional lives because they provide automation, help to streamline processes, increase productivity, better serve customers, and reduce costs. With the rapid proliferation and diversity of Internet of Things (IoT) devices, networks have become extremely vulnerable due to the following reasons:

- **Resource Constrained:** IoT devices are often resource-constrained with little to no security built in, which makes them more enticing for attackers. These devices have limited storage space, processing power, and battery life. They are often optimized to conserve power and, as a result, place little emphasis on security. Implementing security processes in these devices is highly complex due to the limited capabilities available on the devices.
- **Bypass Traditional Zero Trust Architecture solutions:** IoT and its class of devices tend to bypass traditional ZT solutions designed to validate authentication and access requests. A user in a ZTA environment can present MFA tokens or can be presented with methods such as CAPTHAs, while a general-purpose computing device can present certificates or other hardware-based solutions. Many IoT devices are simple sensors and lack the necessary hardware components to support such solutions.
- **Lack the capability for agents:** IoT devices and its class of devices often lack the capability for agents or the ability to attest to the device identity. A general-purpose computing device (GPD) often presents the ability to install an agent or other applications that can attest to the device's identity. IoT

devices offer no such capabilities.

In current software-defined networking (SDN) architectures, flows are often installed proactively. This means that flows are installed to handle all possible traffic prior to the arrival of such network traffic. In this architecture, network traffic will simply be allowed without any means of authentication.

This research presents a method of authentication that profiles network traffic and builds unique fingerprints for IoT devices. The fingerprints are a representation of the behavior of the IoT devices. Through extensive packet capture, feature extraction, and machine learning, unique profiles are created for each device. The unique profile is a representation of a fingerprint. Just like human beings have multiple fingerprints, an IoT device can also have various fingerprints; however, a fingerprint is unique to a single device. We then compare the on-demand generated fingerprints of a device attempting to communicate to those in the database and update the flow tables of a SDN switch, accordingly ensuring that only devices with matching profiles can communicate on the network. Due to their limited functionality, IoT devices behave in a specific pattern and are often highly predictive. The predictive nature of IoT device traffic and their limited protocol set make them a candidate for this method of authentication and access control, satisfying the ZTA's identity management and access control component. To utilize this method of authentication, we created an Opendaylight (ODL) Bundle that intercepts packets sent to the controller and sends them to a Flask Web server. The ODL Bundle is written in Java. The Flask Web server calls on several Python scripts to perform packet parsing, feature extraction, fingerprint generation, database query, and device classification. While Flask uses Python as the main programming language, a system integrator could use any language they choose, as this is done external to the controller. The remainder of this paper is organized as follows: the following section reviews related work. Section 3 presents an overview of our proposed architecture and explains its key components. Section 4 explains the environmental environment used in this paper. Section 5 presents our experimental scenarios and measurements. Sections 6 and 7 discuss the research problem and conclude this paper, respectively.

2. Related Work

Comer [18] presents an architecture that disaggregates controller functionality and externalizes packet processing designed to decentralize micro-services at the control plane level. They utilized a Kafka event distribution application that listens for incoming packets from network devices and forwards them to a Kafka cluster for processing. This work is the closest to our work; however their work focuses on measuring response time. Our work focuses on the externalization of packet processing for the purpose of identity management and access control.

Miettinen et al. [19] was one of the first works that delved into this space. It uses SDN to implement network isolation and traffic filtering. They sought to identify the types of devices being connected to the network and enable the enforcement of rules in an effort to constrain the communication of vulnerable devices and minimize damage resulting from their compromise. They used static rule set and only examined device status at setup time. This method is static and does not perform continuous authentication as required by ZTA. Our work provides dynamic authentication and access control by continuously validating connection requests from each device.

Eidle et al. [20] presented results from an experimental cybersecurity test bed, which implements aspects of a Zero Trust data communication network using autonomic OODA loops (observe, orient, decide, act). They present test results of trials in which identity management with automated threat response and packet-based authentication were combined with dynamic management of eight distinct network trust levels. The log parsing and orchestration software works alongside open-source log management tools to coordinate and integrate threat response from firewalls, authentication gateways, and other network devices. Their work focused on detecting and blocking Distributed Denial of Service (DDoS) attacks and not authentication. Our work focuses on blocking traffic from devices that have failed authentication by dropping the communication request.

3. Overview of the Proposed Architecture

Figure 1 presents a basic ZTA. It is divided into two distinct areas: a data or forwarding plane and a control plane. The current SDN models separates the functionality into three broad pieces: a data plane, a control plane, and an application plane [22]. An SDN controller implements the control plane. Most SDN controllers employ two types of application programming interfaces (APIs) used to communicate with outside entities: a Northbound (NB) interface that defines communication between an external application and control plane software running in the controller and a Southbound (SB) interface that defines communication between the control plane software running in the controller and underlying network devices [23]. The data or forwarding plane simply forwards traffic based on predefined flows, while the control plane ultimately makes the decision about what should be forwarded. Figure 2 depicts our proposed ZTA, while Figure 3 illustrates the proposed ZTA that supports device authentication using device behavior. The key components are:

- **Opendaylight Bundle:** The first step to provide authentication is to add a mechanism to the SDN controller that can intercept packets sent to it by the OpenFlow switches and extract relevant information. Our solution is designed around a Java Bundle that resides on the SDN controller. The Java[1] Bundle integrates with the SDN controller and intercepts OpenFlow packets that are sent to the controller and

1. https://javadoc.io/doc/org.opendaylight.mdsal/mdsal-docs/13.0.1/index.html

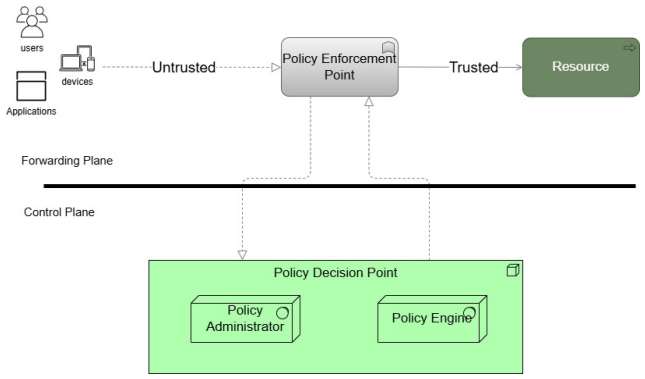

Figure 1. Zero Trust Architecture.

extracts the packet payload and forward them as a byte string to the Flask Web Server. The ODL Java Bundle also has the responsibility of writing new bidirectional flows to the OpenFlow switch(s) and sending a packet-out message for the packets that are forwarded to the SDN controller.

- **Flask Web Server:** The Flask Web Server[2] receives the packet payload from the SDN controller and performs the necessary operations on the packet. As Figure 4 demonstrates, it parses the packet and generates a fingerprint, which is used to query the database. If a matching fingerprint is found in the database, the packet is deemed to be from a legitimate device. Based on the returned values, it sends certain parsed values back to the SDN controller. These values include source and destination MAC address, source and destination IP address, source and destination port, the table number, the protocol value (TCP, UDP, ICMP, etc.), the Ethernet type as well as the flow ID, which is a random 6-digit hexadecimal value.
- **Python Scripts:** The Python scripts are used to perform the necessary parsing of the packet, fingerprint generation and additions to the database, database query, and device classification. In addition, the Python scripts serve to perform the machine learning portion of the application to ultimately classify new devices being introduced into the network.
- **MySQL Database:** The MySQL[3] database is used to store the fingerprints. The database stores the fingerprints, which is a combination of the the device MAC address and 16 binary features, as well as the device ID, which is simply represented as the device MAC address.

The following subsections explain how the proposed solution can be used to provide device authentication on a legitimate device, block a device that is not legitimate from communication, and identify a new device from communicating on the network.

2. https://flask.palletsprojects.com/en/3.0.x/
3. https://www.mysql.com/

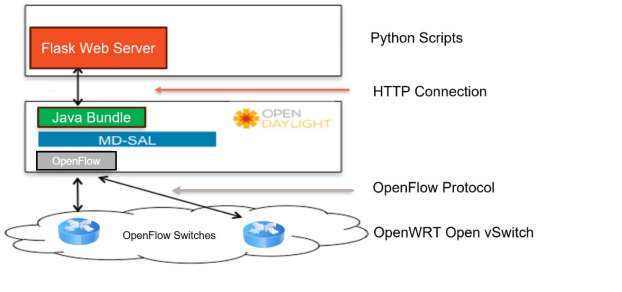

Figure 2. The proposed Zero Trust Architecture components.

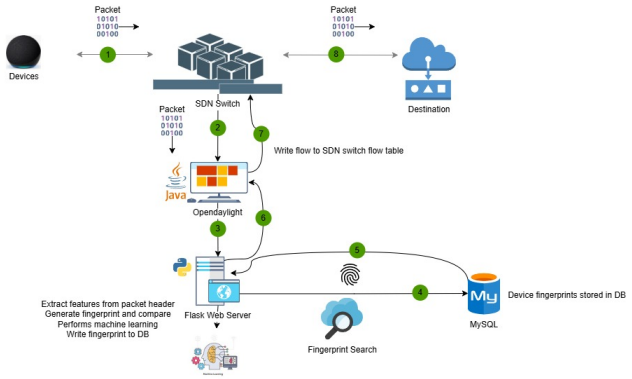

Figure 3. Proposed Zero Trust Architecture for Identity Management and Access Control.

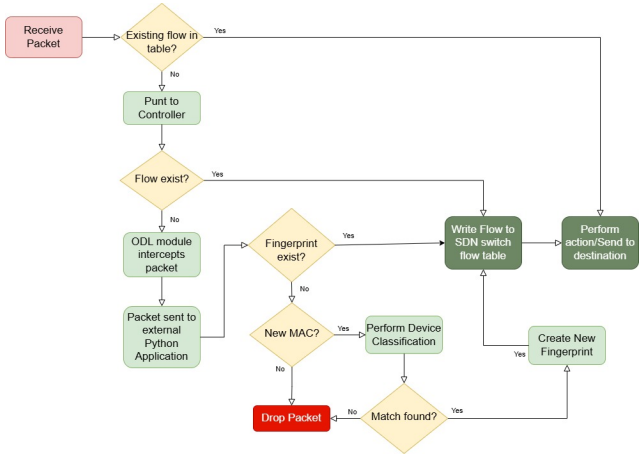

Figure 4. Packet processing and flow rule installation sequence diagram.

3.1. Example of a legitimate device

This section uses our reactive application to provide device authentication.

The application acts in response to packets for which no forwarding rule exists in the flow table and installs a new rule for the flow. To implement the application, it must receive an incoming packet, extract the packet payload, generate a fingerprint from the packet header data, compare the generated fingerprint to the fingerprints in the database,

generate the flow rule(s), and install the flow rule(s) on the appropriate network device. For the flow rule to be installed on the network device, a matching fingerprint must reside in the database. To facilitate communication, a variety of static flows are written in the switch(s). For example: static flows are written to allow Address Resolution Protocol (ARP). Additionally, static flows are written to allow the switch(s) to communicate with the controller. As illustrated in Figure 4 and Figure 9, when a packet arrives at the controller, and no forwarding rules exist, the packet is sent to the SDN controller for further processing. Key information is then extracted from the packet header and compared to information pre-installed in the database. If it finds matching information, the SDN controller will install flow rule(s) to allow the traffic.

3.2. Prevent Illegitimate Devices

This section uses our reactive application to prevent illegitimate devices from communicating on the network.

Our application can prevent illegitimate devices from communicating on the network. Our application uses a reactive flow model. A fingerprint will be generated when a device attempts to communicate using the packet header information. This will be done in response to the absence of a matching flow in the flow table. That fingerprint will then be compared to pre-installed fingerprints located in the database. If the fingerprint does not match, a flow rule will not be installed, and traffic will be dropped for the illegitimate device attempting to communicate.

3.3. Commissioning New Device

This section uses our reactive application to commission a new device by performing device classification.

From time to time, it may be necessary to add new devices to the network. Our application can identify new devices that match an existing device type being introduced into the network. Additionally, fingerprints will be added to the database for a new device that matches an existing device. As illustrated in Figure 9, the application will extract the packet header features and perform device type classification using the Random Forest algorithm ”predict_proba” function. If the prediction matches an existing device type, it will extract the fingerprint of the new device as well as its device type identification and add it to the database for future use. In the event that a new device does not match any existing device, the device will be classified as ”Other” and will require manual intervention by the network programmer to generate fingerprints and add them to the database for future use. The application will only match a device where the packet header statistical values are close enough to match an existing device type.

4. Experimental Setup

4.1. Implementation Details

We implemented our solution using the Opendaylight[4] SDN controller. The OpenDaylight Project is a collaborative open-source project hosted by the Linux Foundation. The project serves as a platform for software-defined networking (SDN) for customizing, automating, and monitoring computer networks of any size and scale [24]. To further demonstrate our solution, we utilized a Flask Web Server to serve as our external application that interacts with the various Python[5] scripts. Flask is a micro web framework written in Python that does not require any particular tools or libraries. Additionally, we utilized MySQL as the database repository to store the various fingerprints. We developed an Opendaylight Java bundle using Maven. Maven[6] allows us to identify all the necessary dependencies and bundle them together. This allows the bundle to download all the necessary dependencies for its operation. Additionally, because this is a bundle, it allows for easy portability among SDN controllers.

4.2. Experimental Testbed

We conducted our experiments on our SDN testbed, which is a Linksys WRT 3200ACM router. Multiple studies [25], [26], [27], [28], [29], [30] have been completed utilizing Mininet[7] as a virtual SDN switch. We opted to use a physical device because the testbed contains real-world devices that require a connection point. We flashed the router with OpenWRT[8] version 23.02.02 and installed the Open vSwitch (OVS) package. Open vSwitch[9] is a production-grade, multilayer virtual switch licensed under the open-source Apache 2.0 license. It is built to enable large-scale network automation through programmatic extensions while maintaining support for standard management interfaces and protocols. We added a total of 25 IoT devices, as well as a few Raspberry Pis, to the testbed, as illustrated in Figure 5. We added one (1) Amazon Alexa, one (1) Amazon Fire TV Stick, one (1) Blink Doorbell Camera, one (1) Blink Mini Camera, one (1) VR520 IR Camera, one (1) Govee Leak Detector and two (2) sensors, three (3) Heiman Door Sensors, two (2) LWITT Light Bulbs, five (5) Sengled light bulbs (for of them were the exact same make and model), one (1) Sonof WiFi Smart Switch, one (1) TP-Link Light Bulb, four (4) TP-Link Smart Plugs, and one (1) Web Power Switch. The two Govee Leak Detector devices utilize radio frequency (RF) and are connected to the Govee Leak Detector Gateway; therefore, we performed classification on the gateway device and not the individual leak detectors.

4. https://docs.opendaylight.org/en/stable-calcium/
5. https://www.python.org/
6. https://mvnrepository.com/
7. https://mininet.org/
8. https://openwrt.org/
9. https://www.openvswitch.org/

Figure 5. Testbed.

TABLE 1. IoT DEVICES IN THE TESTBED

Device Name	Device Type	Quantity	Protocol
Amazon Alexa	Home Assistant	1	WiFi
Amazon Fire TV	Media	1	WiFi
Blink Doorbell Camera	Camera	1	WiFi
Blink Mini Camera	Camera	1	WiFi
Camera VR520 IR	Camera	1	Ethernet
Govee Gateway	Gateway	1	WiFi
Govee Leak Detector	Sensor	2	RF
Heiman Door Sensor	Sensor	3	WiFi
LVWIT Light Bulb	Light Bulb	2	WiFi
Sengled Light Bulb	Light Bulb	5	WiFi
Sonof WiFi Smart Switch	Smart Switch	1	WiFi
TP-Link Light Bulb	Light Bulb	1	WiFi
TP-Ling Plug	Smart Plug	4	WiFi
Web Power Switch	Smart Switch	1	WiFi

Table 1 presents the devices added as well as the device types, the quantity of each, and the protocol that is used to connect to the network. We added an OFP_NORMAL flow to OVS and captured packets using Wireshark[10] (4.2.3) and TCPdump for approximately 20 days. The 20 days were not 20 continuous days. The OFP_NORMAL flow was added to ensure that the SDN switch operated as a layer 2 device and forwarded all traffic that it received. We utilized a Samsung Notebook Model NP940X5J Intel Core i7-4500U CPU @ 1.80 GHZ, 8 GB RAM. We installed VirtualBox 7.0.14 r161095 and created two (2) Ubuntu 22.04.4 virtual machines. We installed Opendaylight Calcium 2024.03 with three (3) GB RAM and one (1) CPU on the first virtual machine. We installed MySQL 8 on the 2nd virtual machine with two (2) GB RAM and one (1) CPU.

5. Experimental Phase Results

- **Learning Phase:** The primary objective of the learning phase is to collect data. In this phase, we determined how much data should be collected and determined the optimal features to use. We determined what features would be useful in identifying the device as obtained from the packet header, as well as other features not included in the packet header that would be useful.
- **Analysis Phase:** The primary objective of the analysis phase is to identify the devices that are communicating on the network with high accuracy. To utilize our solution, we need to be able to identify the devices with high certainty. To achieve this, we utilized research conducted by Kostas et al. [21], where they presented a method called aggregation of like devices; however, in our work, we aggregated devices that are identical in nature. The idea behind this is that devices that are identical will behave in a similar manner. Additionally, we modified one feature related to the way that they presented the IP address count. While this feature allow the model to perform well utilizing the training and test data, it did not perform well with a validation set. Validation sets often contain unseen data. Due to the way that they utilize the IP Address count feature, it always reset at zero. In a large dataset, this value can become substantial. This feature is extremely important in the classification process. To overcome this limitation, we proposed creating a dictionary that maintains the count that is available throughout the entire training and validation process. We were able to achieve accuracy, precision, and recall above 99% using Random Forest, as illustrated in Figure 6. Kostas et al. [21] showed in their work that Random Forest as an ensemble machine learning model works well for multi-class classification problems. Fingerprints are proactively added to the database by taking all of the packets captured during the learning phase, removing certain packets such as ARP –due to static flow, removing all duplicates, and extracting the needed features used in the fingerprint from the dataset.
- **Implementation Phase:** In the implementation phase, the OpenFlow switch, which operates based on reactive flows, sends the received packet to the SDN controller as an OFP_PACKET_IN message. This is the original packet encapsulated in an OpenFlow packet, as illustrated in Figure 7. The SDN controller utilizing our Packet-Forwarder bundle extracts the payload (the original packet) from the OpenFlow packet as a byte string, ensuring that it is not a null packet. It then sends the extracted OpenFlow packet payload to the Flask Web Server as an HTTP Post request. The Flask Web Server receives the payload string and parses the string to obtain the packet header information. It then performs the sequence of activities as illustrated in Figure 9. The main activity performed is to generate a fingerprint from the parsed packet header data. We defined the fingerprint generated as follows: Let $f1, f2, f3, \ldots, f16$ represent 16 binary features obtained from the packet header. Let MAC be the Ethernet source address obtained from the packet header as a binary string. We denote a

10. https://www.wireshark.org/

	precision	recall	f1-score	support
AmazonAlexa	0.99	0.99	0.99	178307
AmazonFireTV	0.99	0.99	0.99	141626
BlinkDoorbellCamera	0.99	0.98	0.98	62530
BlinkminiCamera	0.96	0.97	0.96	32701
CameraV520IR	1.00	1.00	1.00	6483
GP_Device	0.99	0.98	0.98	2886
GoveeLeakDetector	1.00	1.00	1.00	15714
HeimanDoorSensor	1.00	0.99	0.99	1183
LVWIT	1.00	1.00	1.00	9251
MerossSmartGarage	0.99	0.99	0.99	2880
SengledBulb	1.00	1.00	1.00	64424
SengledBulb5	1.00	1.00	1.00	2731
SonofWiFiSmartSwitch	1.00	0.99	1.00	1239
TP-LinkLightBulb	0.96	0.95	0.95	2891
TP-LinkPlug	0.98	0.98	0.98	8974
WebPowerSwitch	1.00	1.00	1.00	1911
accuracy			0.99	535731
macro avg	0.99	0.99	0.99	535731
weighted avg	0.99	0.99	0.99	535731

Figure 6. Device classification report.

TABLE 2. FEATURES USED IN FINGERPRINT

Protocols Layers	Protocols
Layer 2 Protocols	ARP, LLC
Layer 3 Protocols	IP, IPV6, ICMP, ICMP6
Layer 4 Protocols	TCP, UDP
Layer 7 protocols	HTTP, HTTPS, DHCP, BOOTP, SSDP, DNS, MDNS, NTP

Protocol	Length	Info
OpenFlow	231	Type: OFPT_PACKET_IN

Figure 7. OpenFlow Packet-In message.

fingerprint Pi as the combination of $MACi$ and $f1, f2, f3, \ldots, f16$ concatenated together:

$$Pi = MACi \frown (fi1, fi2, fi3, \ldots, fi16)$$

We extract the fingerprint from each packet that is sent to the SDN controller as a 64-bit binary string. Figure 8 presents fingerprints generated for one device. There are 3 unique fingerprints corresponding to a single device. These fingerprint are representative of the "Blink Mini Camera". A device is not limited to the number of fingerprints that it can have. The 16 features used are listed in table 2.

Once Flask completes the necessary operations, it returns certain parsed values to the SDN controller. These values include source and destination Ethernet address, source and destination IP address, source and destination ports, protocol, the flow ID, and the Table ID. The flow ID is a randomly generated 6-digit hexadecimal string, while the reverse flow adds

Fingerprints

1001000000111001010111110000111111011100000101010010000100000100
1001000000111001010111110000111111011100000101010010000100110000
1001000000111001010111110000111111011100000101010010001001000000

Figure 8. 64-bit fingerprints in binary format for 1 device.

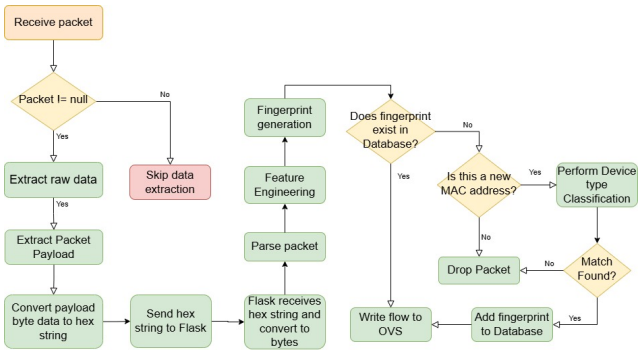

Figure 9. Zero Trust application process flow

an "r" to the flow ID. The Packet-Forwarder bundle uses the parsed values to update the flow table with bi-directional flows. It adds a flow based on the original packet and a flow with the parsed values in reverse. Additionally, the Packet-Forwarder bundle adds two timeout values to the flow rule: a hard timeout and a soft timeout. The soft timeout removes the flow if no packet is received at the network device with matching criteria within a specified interval, while the hard timeout removes the flow rule after a specified amount of time set by the network programmer. This ensures that the device will be re-authenticated periodically. The bi-directional flows are written with the following parameters:
cookie, table ID, idle_timeout, hard_timeout, send_flow_rem priority, Protocol, dl_vlan, dl_src, dl_dst, nw_src, nw_dst, tp_src, tp_dst and the actions to take for the specified packet.

The initial packets sent by a device provide authentication, while the flow(s) written to the SDN Switch provide access control for the specific device.

Additionally, we observed the additional latency introduced while using our Packet-Forwarder application to implement authentication. Figure 10 illustrates the additional latency introduced. This latency is incurred exclusively by the initial packet, as subsequent packets are not redirected to the SDN controller until the designated timeout has elapsed. It is important to note that this latency is in addition to the standard network operational latency. We elected to present the additional latency rather than the overall latency for two primary reasons: (1) each network environment is unique, and the latency observed in one setting may not accurately represent

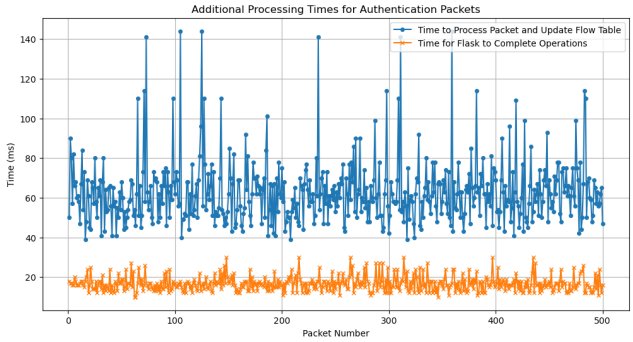

Figure 10. Additional latency to implement ZTA authentication.

that in another, and (2) the additional latency specifically characterizes the delay introduced by the first transmitted packet, rather than applying uniformly to every packet sent by a device.

6. Discussion and Future Work

The proposed architecture introduces new research questions that need to be investigated:

- As the environment increases the number of flow rules will also increase. We will need to determine the optimal amount of time to keep flow rules in the flow table(s) to prevent flow table overflow.
- Many IoT devices utilize long poling or persistent web sockets to keep connections open. This is done since repeated requests to a server often waste resources. Additionally, by holding the connection open it allows users, other devices, or applications to interact with the device via their control app. For this reason, the flow rules should remain sufficiently long enough to not impact the ability of users to interact with the device near real-time while ensuring it is short enough to encourage re-authentication.
- A comparative analysis with other authentication methods used in IoT device authentication. Common authentication methods used to authenticate a device on a network include but are not limited to password authentication, certificate-based authentication, Extensible Authentication Protocol (EAP), Kerberos, Single Sign-On (SSO), Remote Authentication Dial-In User Service (RADIUS), and mutual authentication. Many IoT devices lack the capability to support these types of authentication.

7. Conclusion

In this paper, we present an architecture for Zero Trust Identity Management and Access Control that utilizes an Opendaylight Java Bundle, Flask Web Server, and Python to provide IoT device authentication and access control. This is a critical step in providing a mechanism that can provide authentication to IoT devices that lack the necessary resources to support traditional Zero Trust Architecture technologies. We showed that utilizing an external application introduces some overhead into the system for the initial packet of a communication attempt, however, this overhead is tolerable because of the benefits that it provides. This overhead could be reduced by utilizing more efficient hardware or optimized Python code.

References

[1] A. H. Seh, M. Zarour, M. Alenezi, A. K. Sarkar, A. Agrawal, R. Kumar, and R. A. Khan, "Healthcare Data Breaches: Insights and Implications," Healthcare (Basel, Switzerland), vol. 8, no. 2, p. 133, 2020. doi: 10.3390/healthcare8020133.

[2] C. X. Ou, X. Zhang, S. Angelopoulos, R. M. Davison, and N. Janse, "Security breaches and organization response strategy: Exploring consumers' threat and coping appraisals," International Journal of Information Management, vol. 65, p. 102498, 2022. doi: 10.1016/j.ijinfomgt.2022.102498.

[3] J. Li, W. Xiao, and C. Zhang, "Data security crisis in universities: Identification of key factors affecting data breach incidents," Humanities and Social Sciences Communications, vol. 10, no. 1, 2023. doi: 10.1057/s41599-023-01757-0.

[4] M. Hill, D. Swinhoe, and J. Leyden, "The 18 biggest data breaches of the 21st century," CSO Online, Sep. 12, 2024. [Online]. Available: https://www.csoonline.com/article/534628/the-biggest-data-breaches-of-the-21st-century.html

[5] D. Hylender, P. Langlois, A. Pinto, and S. Widup, 2024 Data Breach Investigations Report, 100th ed. Verizon, 2024. [Online]. Available: https://www.verizon.com/business/resources/Tad3/reports/2024-dbir-data-breach-investigations-report.pdf

[6] O. C. Edo, T. Tenebe, E. Etu, A. Ayuwu, J. Emakhu, and S. Adebiyi, "Zero Trust Architecture: Trend and Impact on Information Security," Int. J. Emerg. Technol. Adv. Eng., vol. 12, no. 7, pp. 140–147, 2022. doi: 10.46338/ijetae0722_15.

[7] O. Borchert and A. Tan, "Implementing a Zero Trust," National Institute of Standards and Technology (NIST), Mar. 2020. [Online]. Available: https://www.nccoe.nist.gov/sites/default/files/legacy-files/zt-arch-project-description-draft.

[8] M. Shore, S. Zeadally and A. Keshariya, "Zero Trust: The What, How, Why, and When," in Computer, vol. 54, no. 11, pp. 26-35, Nov. 2021, doi: 10.1109/MC.2021.3090018.

[9] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero Trust Architecture (ZTA): A Comprehensive Survey," IEEE Access, vol. 10, pp. 57143–57179, 2022. doi: 10.1109/ACCESS.2022.3174679.

[10] D. Puthal, S. P. Mohanty, P. Nanda, and U. Choppali, "Building Security Perimeters to Protect Network Systems Against Cyber Threats [Future Directions]," IEEE Consum. Electron. Mag., vol. 6, no. 4, pp. 24–27, 2017. doi: 10.1109/MCE.2017.2714744.

[11] Deloitte, "Zero Trust 2021 A Revolutionary Approach to Cyber or Just Another Buzz Word?" Cybersecurity, 2021. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/de/Documents/risk/deloitte-cyber-zero-trust.pdf

[12] C. Buck, C. Olenberger, A. Schweizer, F. Völter, and T. Eymann, "Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust," Comput. Secur., vol. 110, p. 102436, 2021. doi: 10.1016/j.cose.2021.102436.

[13] E. S. Hosney, I. T. A. Halim, and A. H. Yousef, "An Artificial Intelligence Approach for Deploying Zero Trust Architecture (ZTA)," in 5th Int. Conf. Comput. Informatics, ICCI, 2022, pp. 343–350. doi: 10.1109/ICCI54321.2022.9756117.

[14] M. Bush and A. Mashatan, "From Zero to 100," Commun. ACM, vol. 66, no. 2, pp. 48–55, 2023. doi: 10.1145/3573127.

[15] M. Campbell, "Beyond Zero Trust: Trust Is a Vulnerability," Computer, vol. 53, no. 10, pp. 110–113, 2020. doi: 10.1109/MC.2020.3011081.

[16] U. Mattsson, "Zero Trust Architecture," in Controlling Privacy and the Use of Data Assets, pp. 127–134, 2022. doi: 10.1201/9781003189664-11.

[17] N. Sheikh, M. Pawar, and V. Lawrence, "Zero trust using network micro-segmentation," in IEEE INFOCOM 2021 - IEEE Conf. Comput. Commun. Workshops, 2021. doi: 10.1109/INFOCOMWKSHPS51825.2021.9484645.

[18] D. Comer and A. Rastegarnia, "Externalization of Packet Processing in Software Defined Networking," IEEE Netw. Lett., vol. 1, no. 3, pp. 124–127, 2019. doi: 10.1109/LNET.2019.2918155.

[19] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," in Proc. Int. Conf. Distrib. Comput. Syst., 2017, pp. 2177–2184. doi: 10.1109/ICDCS.2017.283.

[20] D. Eidle, S. Y. Ni, C. Decusatis, and A. Sager, "Autonomic security for zero trust networks," in IEEE 8th Annu. Ubiquitous Comput. Electron. Mobile Commun. Conf., 2017, pp. 288–293. doi: 10.1109/UEMCON.2017.8249053.

[21] K. Kostas, M. Just, and M. A. Lones, "IoTDevID: A Behavior-Based Device Identification Method for the IoT," 2021. doi: 10.1109/JIOT.2022.3191951.

[22] C. Trois, M. D. D. Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," IEEE Commun. Surveys Tuts., vol. 18, no. 4, pp. 2687–2712, 4th Quart., 2016

[23] J. H. Cox et al., "Advancing software-defined networks: A survey," IEEE Access, vol. 5, pp. 25487–25526, 2017

[24] "OpenDaylight project," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/OpenDayligh_Project. Accessed: Nov. 2, 2024.

[25] S. Bera, S. Misra and A. Jamalipour, "FlowStat: Adaptive Flow-Rule Placement for Per-Flow Statistics in SDN," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 3, pp. 530-539, March 2019, doi: 10.1109/JSAC.2019.2894239.

[26] K. P. Kumar and P. Sivanesan, "Flow rule-based routing protocol management system in software-defined IoT sensor network for IoT applications," International Journal of Communication Systems, vol. 35, no. 11, pp. 1–17, 2022. doi: 10.1002/dac.5182.

[27] P. K. Sharma, J. H. Park, Y. S. Jeong, and J. H. Park, "SHSec: SDN based Secure Smart Home Network Architecture for Internet of Things," Mobile Networks and Applications, vol. 24, no. 3, pp. 913–924, 2019. doi: 10.1007/s11036-018-1147-3.

[28] P. Krishnan, K. Jain, K. Achuthan, and R. Buyya, "Software-Defined Security-by-Contract for Blockchain-enabled MUD-aware Industrial IoT Edge Networks," IEEE Transactions on Industrial Informatics, 2021. doi: 10.1109/TII.2021.3084341.

[29] K. S. Sahoo and D. Puthal, "SDN-Assisted DDoS Defense Framework for the Internet of Multimedia Things," ACM Transactions on Multimedia Computing, Communications and Applications, vol. 16, no. 3s, 2021. doi: 10.1145/3394956.

[30] M. Sisov, "Building a Software-Defined Networking System with OpenDaylight Controller," 2016.