Trust *and* Verify: A Complexity-Based IoT Behavioral Enforcement Method. *

Kyle Haefner[1][0000−0001−8884−0159] and Indrakshi Ray[1][0000−0002−0714−7676]

Colorado State University, Fort Collins CO 80523, USA

Abstract. In an Internet of Things (IoT) environment, devices may become compromised by cyber or physical attacks causing security and privacy breaches. When a device is compromised, its network behavior changes. In an IoT environment where there is insufficient attack data available and the data is unlabeled, novelty detection algorithms may be used to detect outliers. A novelty threshold determines whether the network flow is an outlier. In an IoT environment, we have different types of devices, some more complex than others. Simple devices have more predictable network behavior than complex ones. This work introduces a novel access control method for IoT devices by tuning novelty detection algorithm hyper-parameters based on a device's network complexity. This method relies only on network flow characteristics and is accomplished in an autonomous fashion requiring no labeled data. By analyzing connection based parameters and variance of each device's network traffic, we develop a formalized measurement of complexity for each device. We show that this complexity measure is positively correlated to how accurately a device can be modeled by a novelty detection algorithm. We then use this complexity metric to tune the hyper-parameters of the algorithm specific to each device. We propose an enforcement architecture based on Software Defined Networking (SDN) that uses the complexity of the device to define the precision of the decision boundary of the novelty algorithm.

Keywords: IoT · Security · Unsupervised Machine Learning · Access Control.

1 Introduction

The billions of devices that bridge the cyber and physical worlds have already altered how we interact with our physical surroundings. Smart speakers respond to spoken requests for information, provide reminders or simply turn on and off the lights. Embedded cameras can detect who we are and respond to our gestures to do mundane tasks such as turning up the volume, dim the lights, etc. For all the convenience and function that these devices, they also bring a long history of poorly implemented security, unpatched vulnerabilities, and privacy

* This work was supported by NSF under Award Number CNS 1822118, Cyber Risk Research, NIST, Statnett, AMI, and Cable Television Laboratories.

violations. Poorly implemented security on these devices has lead to distributed denial of service (DDoS) attacks specifically originating from these devices [8].

Security baselines [4, 5] and strong endpoint security in international standards [24] are steps in the right direction, but there will always be insecure devices; either because they were manufactured that way or did not receive software patches. This is highlighted in the large corpus of research [1, 25, 25, 23, 27, 26] that documents how and why vulnerable IoT devices are prone to security attacks. We will never be able to depend on all of our devices being completely secure, therefore we must instead depend on the network to help us to monitor and secure the devices for us. To scale to the networks of tomorrow and to be of practical use to the average consumer, network based IoT security must be done in a largely autonomous manner.

Unlike networks of the past, made up of a small number of general purpose machines, Internet of Things (IoT) networks will increasingly be made up of a large number of specialized devices designed to do a single task. The single purpose and often constrained nature of these devices makes them harder to intrinsically secure, but easier to extrinsically analyze. A single temperature sensor, for example, will not be able to run an anti-malware application, but does have a simple and predictable network traffic footprint.

This work exploits this single purpose nature and the correspondingly predictable network behavior of IoT devices to autonomously derive several measures of complexity based entirely on their network traffic. This allows not only for the classification and evaluation of IoT devices based on their complexity, but also enables each IoT device's historic network behavior to be more accurately modeled using an anomaly detection algorithm that is tuned to this complexity.

For enforcement we employ a software defined network that can proactively take several actions on a flow such as counting, logging, rate-limiting, delaying and blocking. Previous enforcement architectures were built on a binary model of trust and enforcement, i.e. block or allow traffic for a particular port or for a particular flow. Instead of binary enforcement; block or allow, our model allows the enforcement function to dynamically adjust for the complexity of the device (a direct measure of how well it can be modeled) and the abnormality of the flow (the measure of its separation from inliers). Highly abnormal flows from very simple devices can be automatically blocked, while such flows from more complex devices can be rate-limited or logged. Effectively this places more trust in devices that can be accurately modeled and less trust in devices that cannot be accurately modeled. We believe this is a key contribution and we have developed a ground truth methodology to test our model and a network reference architecture to enforce it.

This behavior-based flow routing model can be used as the first line of defense; to slow or prevent botnets and DDoS attacks at their source by detecting anomalous traffic at the granularity of individual flows from specific devices. Proper implementation would allow the network to selectively isolate and block malicious flows, leaving devices continuing to perform their primary function.

Research Contributions

- We formalize measurements of device complexity and establish a definition of device behavior based on anomaly/novelty detection formulated from IP header traffic all using unsupervised techniques that require no labeled data.
- We hypothesize that devices with smaller complexity values will show less of an aberration in its behavior compared with those of higher complexity values. Our results justify this. Thus, we tune the outlier threshold for the anomaly detection algorithms in accordance with device complexity.
- We propose a test architecture that uses the complexity tuned behavior to autonomously monitor and enforce learned behavior from devices.

This work is organized as follows; in Section 2 we review related research on how to analyze behavior and secure IoT devices. In Section 3 we describe the lab setup and the data collected. In Section 4 we develop methods for measuring complexity and how devices are classified into discrete groups. Section 5 describes how we develop a method for modeling learned behavior of IoT devices and describe the enforcement architecture in section 6. In section 7 we describe how the tuning of the hyper-parameters affects precision, recall and false positives of the model. Finally in section 8 we summarize the work and propose possible next steps in this research.

2 Related Work

2.1 Device Identity Detection

Loepz-Martin et al. [10] build a network traffic classifier (NTC) using a recurrent neural network (RNN) and apply it to labeled IoT traffic. The goal of this is to identify the types of traffic and services exhibited by an IoT device as a step toward identifying the device.

Miettinen et al. [13] have developed a method, called IoT Sentinel, that uses machine learning to designate a device type on the network, referred to by the authors as a device fingerprint. Using the random forest algorithm and 23 network features they were able to identify device types on the network based on the device's traffic. The 23 features are based on layer two, three and four of the OSI networking stack. Expecting that the body of the packet will be encrypted, all the features the authors employed are based on unencrypted parts of the traffic like IP headers information.

Bezawada et al. [2] build on the work done in [13] by using a machine learning approach to broadly identify the device and place it in a predefined category, such as a light bulb. According to the authors, even devices from different manufacturers can be placed into general categories such as two separate light bulbs can be identified and placed into a lighting category.

All supervised solutions of fingerprinting devices suffer from a similar problem in that they require labeled data for each device. Not only this but a supervised classifier must be trained on every device, and potentially retrained on devices after a firmware update.

2.2 IoT Behavior And Autonomous Techniques

The following works use various means of autonomous and unsupervised machine learning approaches to identifying devices and device behavior. This has advantages over statically defined access control lists and firewall rules.

IoT-Keeper [7] is an edge based IoT anomaly based access control system that uses correlation-based feature selection to determine which features do not contribute to the anomaly detection. AuDI [12] implemented an autonomous device-type identification that uses the periodicity of device communications resulting in abstract device categories that could be used to enforce access control policies. DioT [14] extends the AuDI classification model to create a federated approach by aggregating device anomaly detection profiles.

Ren et al. use a privacy focused approach to enumerating and analyzing IoT behavior [19]. Ortiz et al. set up a probabilistic framework to monitor device behavior using an LSTM (Long Term Short Term Memory) neural network, to learn from inherent sequencing of TCP flows to automatically learn features from device traffic with the intent of categorizing devices and distinguishing between IoT devices and Non-IoT devices [18]. The authors are able to identify previously known devices after only 18 TCP-flow samples and categorize devices into two classes IoT and Non-IoT.

2.3 Complexity and Predictability

Formalized measurement of complexity as applied in a computer science context is probably most often associated with the works of Andrey Nikolaevich Kolmogorov, who defined the complexity of an object as the shortest computer program to produce the object as an output [9]. This simple notion arises again in the work of Jorma Rissanen whose work on the minimum description length principal that establishes that the best model for a set of data is one that leads to the best compression of the data [21].

In the paper Predictability, Complexity, and Learning authors Bialek et al. establish a formal result that predictive information provides a general measure of complexity [3]. In this work we propose that the relationship between predictive information and complexity is commutative, i.e. not only does predictive information lead to a measure of complexity, but that complexity provides a general measure of predictive information.

In machine learning this relationship leads to the logical notion that the less complex the model the more accurately it can be modeled, or to put this in the context of IoT, the less complex the device the more accurately we can define its behavior. Specifically, this work builds an anomaly based behavioral model, where the device's complexity directly affects the decision boundary that differentiates between inliers and outliers.

Our model can be used to determine a representative set of flows, along with a learned decision boundary, that define the behavior of a device and these flows can be directly loaded into flow tables of Openflow enabled switches. We believe

that this will scale to the broad spectrum of devices and adapt to any new configurations of devices in the future.

Take, for example, a refrigerator that is also an Android tablet, the methodologies previously mentioned in the related works, would struggle to characterize such a device. Our method does not try to recognize this device as either a refrigerator or a tablet, it does not try to guess at the service or characterize the device's application layer data. Our model does not rely on learning specific human interactions with the refrigerator, nor determining if those interactions are anomalous. Our model only relies on how complex the refrigerator appears on the network and how much it stays within our learned boundary of behavior.

This work extends the work done in [6] by using a novelty detection algorithm and formalizing a ground truth testing methodology, to show the efficacy of the model at recognizing new and anomalous traffic.

3 Data Format and Collection

Data was collected from a real residential network with approximately 25 devices (Table 1) over the course of 37 days. These devices range from general computing devices like laptops and smartphones, IoT hubs with several IoT Devices using Zigbee or Zwave behind them, to single-purpose devices such as light bulbs and temperature sensors. Data was collected by a central MicroTik router (Figure 1) and sent to nprobe [15] running on a Raspberry Pi. Flows were stored in a MariaDB relational database. Table 2 shows the features of the data collected.

Table 1. List of Devices

Home Devices		
• Amcrest Camera	•Plex Server	•Raspberry PI 3
•Google Home	•Samsung Note 8	•Smart Things Hub
•J. Chromebook (Asus)	•Xbox One (2)	•Appple Macbook Pro
•Philips Hue Hub	•Chromecast	•Echo Dot
•Eufy Doorbell	•Motorola Android	•HP Stream Laptop (2)
•Eufy light bulb	•TP Link Switch	•Roku Express
•B. Chromebook (HP)	•Brother Printer	•Roku Stick
•Amazon Alexa 1st gen.	•Fire Tablet (3)	

Flows were aggregated with a maximum of 30 minutes per flow. Inactive flow timeout was set to 15 seconds. If the devices have not exchanged traffic in 15 seconds the flow is completed and recorded. For training and test data sets the data is filtered by an individual IP address. The test environment is configured such that the devices always receive the same IPv4 address.

Definition 1. *Network Flow: A sequence of packets where all the packets in the flow have the same tuple: source address, destination address, source port, destination port and protocol.*

Table 2. Data Features

Feature	Description
IPV4_SRC_ADDR	IPv4 Source Address
IPV4_DST_ADDR	IPv4 Destination Address
IN_PKTS	Incoming flow packets (Device->Destination)
IN_BYTES	Incoming flow bytes (Device->Remote)
OUT_PKTS	Outgoing flow packets (Remote->Device)
OUT_BYTES	Outgoing flow bytes (Remote->Device)
L4_SRC_PORT	IPv4 Source Port
L4-DST_PORT	IPv4 Destination Port
PROTOCOL	IP Protocol Identifier

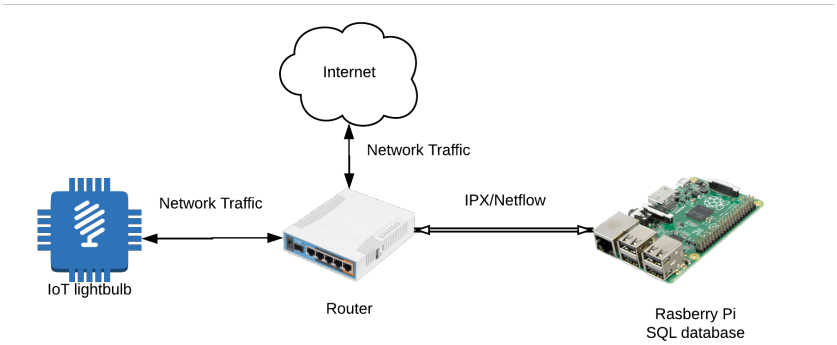

Fig. 1. Data Collection Architecture

4 Device Complexity Classification

Device complexity is an aggregate measurement of a device's IP connections, d_{ip}, and how much its traffic varies over time d_v. Devices that are single purpose should have simple network behavior and general purpose devices will have more complex network behavior. Figure 2 shows where devices should fall along a spectrum of simple and complex.

Fig. 2. Spectrum of Network Complexity

4.1 Device IP Complexity

This research examines how devices form connections. Simply counting the number of unique IP addresses that a device connects with fails to take into account the inherent service-oriented hierarchical structure of the IPv4 address space, where companies and services are often part of similar subnets. To account for this we propose a complexity measure based on a simple ratio of *IP spread* to *IP depth*. IP spread is the number of unique source and destination IP addresses that interact with the device. IP depth is the number of IP addresses that belong to the same higher level octets.

Definition 2. ***IP Tree, IP Branch, IP Leaf, IP Spread*** *An* IP tree *is a unique first order octet which comprises the root of the tree. An* IP Branch *is a second or third order octet that has one or more fourth order octets under it. An* IP leaf *is a unique fourth order octet.* IP Spread *is the sum of total unique IP addresses that interact with a device.*

Device IP Spread (IP_{Spread})

$$IP_{Spread} = \sum IP_{trees} \tag{1}$$

Device IP Depth (IP_{Depth})

$$IP_{Depth} = \frac{\sum IP_{leaf}}{\sum IP_{branch}} \tag{2}$$

Device IP Complexity (d_{ip})

$$d_{ip} = \frac{IP_{Spread}}{IP_{Depth}} \tag{3}$$

To calculate IP spread and depth we build unordered trees of each IP address where the first order octet is the root and lower octets are children. Then we can calculate how many trees, branches and leaf nodes each IoT device contacts. A large number of IP trees with few branches indicates a large IP spread. A small number of IP trees that have many branches and leaves indicates a large IP depth. IP spread/depth is used as one measure of a device's complexity. The intuition here is that this complexity measure mirrors how services are organized based on common IP subnets. As devices form connections out to the Internet their complexity goes up. As the number of connections that have common first, second and third octets increases this has a corresponding reduction in the complexity measurement of the device. Devices belonging to a single ecosystem such as Google Home have a small number of broad trees (low IP Spread and high IP Depth) as they connect to mostly Google's networks dedicated to these types of devices. Other devices such as laptops and smart phones make connections to many unique destinations thus leading to a large number of thin trees each having fewer branches and leaves. Figure 3 shows the total IP complexity of each device. Devices that are more general purpose have higher complexity and are are grouped together on the right of the figure. Single purpose and lower complexity devices are grouped on the left of the figure.

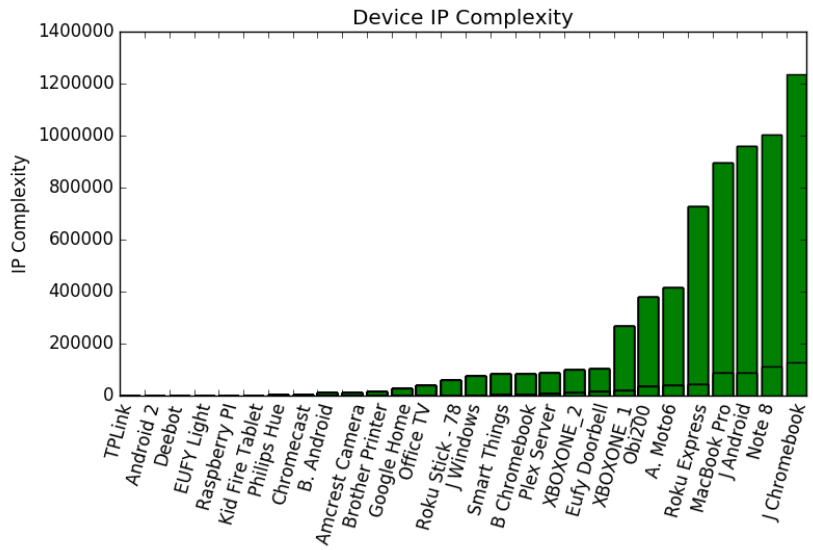

Fig. 3. IP Device Complexity

4.2 Device Variance

The variance metric comes from the simple notion that devices on a network present different variances based on what they do on the network. Device variance is calculated by taking the sum of the standard deviation of n device features d_f in the training set as shown in Equation 4. Each device's variance is graphed in Figure 4. Here again, devices that tend to be more general purpose have higher complexity and are grouped on the right of the figure. Single purpose and lower complexity devices are grouped on the left of the figure.

Device Variance (d_v)

$$d_v = \sum_{f=1}^{n} \sigma_{d_f} \tag{4}$$

4.3 Aggregate Complexity

Overall device complexity is the sum of the average device variance and the average device IP complexity as calculated in Equation 5 and shown in Figure 5. Devices in the figure again show that general purpose and higher complexity devices tend toward the right side of the graph and more single purpose lower complex devices tend to be grouped on the left side of the graph.

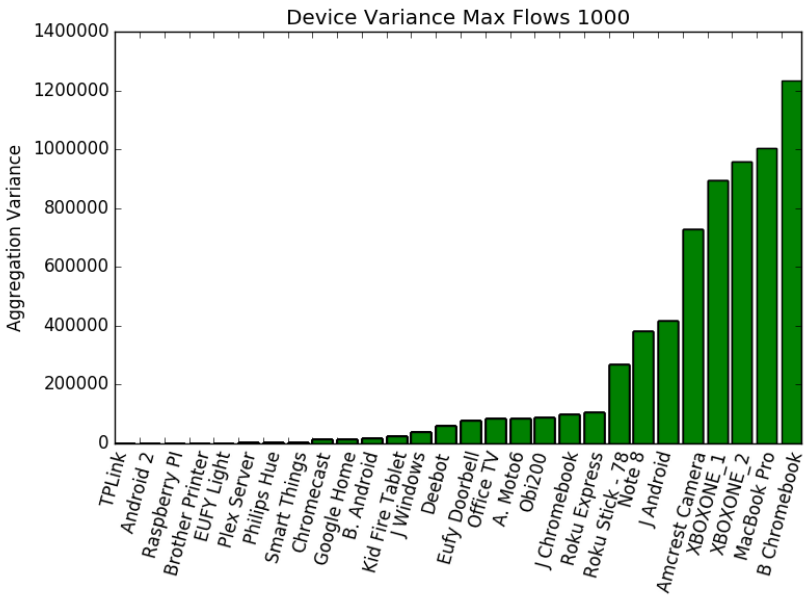

Fig. 4. Average Device Network Variance

Aggregate Device Complexity (A_{DC})

$$A_{DC} = d_{ip} + d_v \tag{5}$$

Discrete Complexity Organizing the devices based on logarithmic magnitudes of complexity allows us to easily examine device characteristics within discrete groups as shown in in Figure 6.

Discrete Device Complexity (D_{DC})

$$D_{DC} = \lceil log_{10} A_{DC} \rceil \tag{6}$$

5 Behavior

Given the complex interactions that IoT devices have with the physical world, behavior represents the dynamic and changing network footprint exhibited by these devices. The sensing and actuating response of IoT devices that bridges the cyber and physical world requires new methods of defining what is normal and what is abnormal. IoT devices, even the same make and model from the same manufacturer will exhibit slightly different behavior based on how they interact

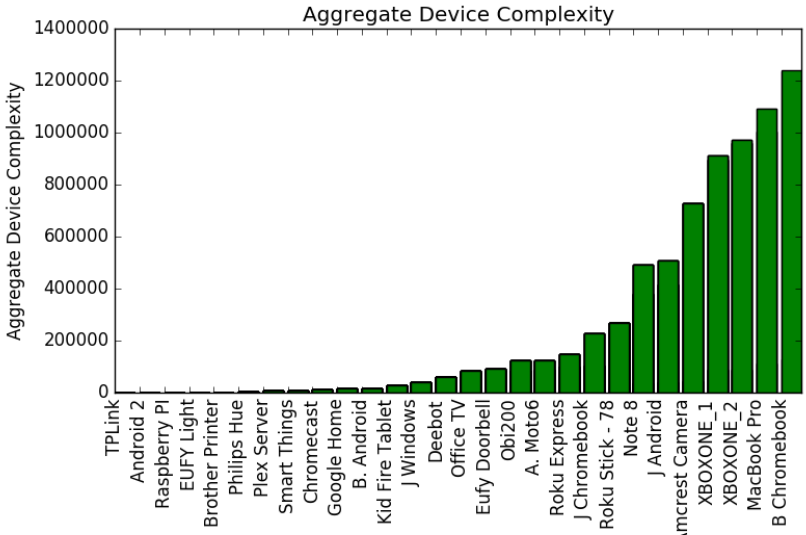

Fig. 5. Aggregate Complexity

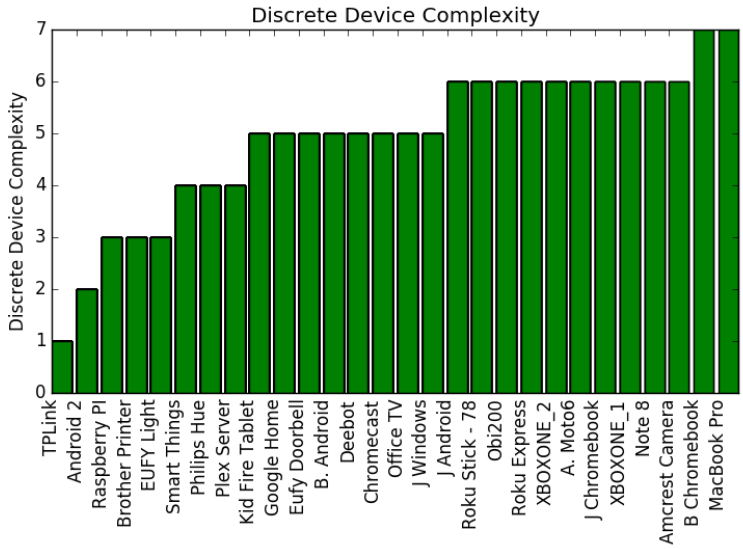

Fig. 6. Discrete Complexity

with the human inhabitants, each other and the environment. Two very similar devices, that have different apps installed may act very differently. This variance in behavior requires that the model is tailored to these specific and individual devices.

We begin by defining IoT device behavior based on the past history of network interactions of the device, bounded by the most extreme of these interactions in the training set. To model the degree of normality and extremity of behavior we turn to classic outlier detection algorithms, adding what we believe to be a key contribution of this research, we tune the hyper-parameter of the outlier detection algorithm to the specific device based on the measure of complexity as defined in the previous section.

This method has the direct affect of making the decision boundary of the trained model a more precise fit for simple devices and more generalized for complex devices. This allows the detection algorithm to be more strict in identifying outliers for simple devices and more lax for complex devices. This enhances the model, enabling it to adaptively prioritize new extreme behavior and reduce false positives for simple devices.

5.1 Novelty Detection

Novelty detection is a form of outlier detection where the training set is considered untainted by outliers i.e. only positive samples. New observations are classified and determined to fall within the decision boundary are inliers and observations outside the decision boundary are outliers. To derive a behavior for a device we employ the One Class Support Vector Machine (OCSVM) algorithm using the Radial Basis Function (RBF) kernel [11]. Outlier flows detected during the training phase are recorded, and form the set of flows we call significant flows.

Definition 3. Significant Flow: *A* significant flow *is one that is marked as an outlier by the OCSVM during training. This set of flows plus the decision boundary forms the behavior boundary of the device.*

Definition 4. *Device Behavioral Boundary:* Device behavioral boundary *is the set of all unique significant flows and the decision boundary found during training.*

To establish that the complexity measurement is a statistically relevant one we take the linear regression of the number of outliers found by the OCSVM using the default values of $\nu = 0.5$ and gamma as calculated in equation 7. The correlation of outliers to anomalies can be seen in Figure 7.

$$\gamma = \frac{1}{n_{features} * x.var()} \tag{7}$$

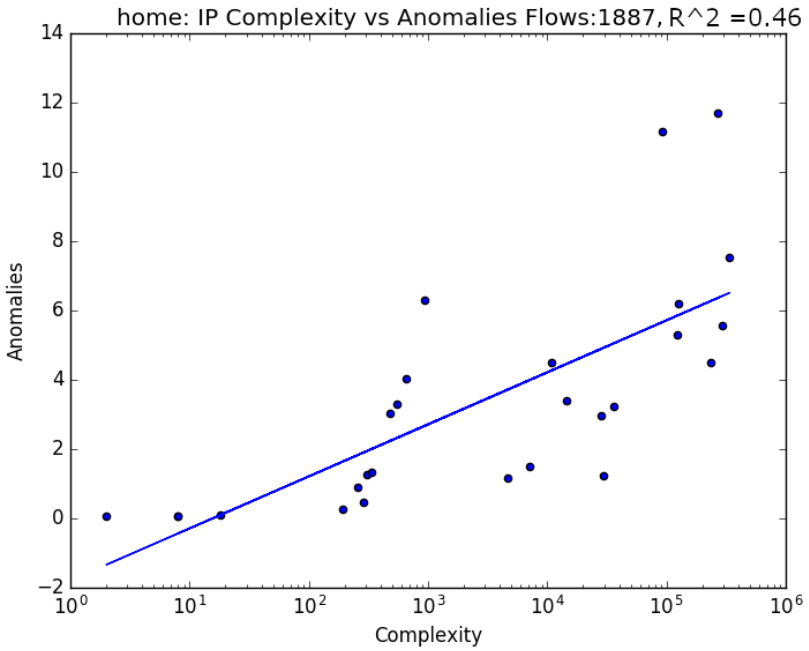

Fig. 7. Complexity Vs Anomalies

5.2 Novelty Detection Tuning Using Device Complexity

To the best of our knowledge, labeled anomaly data for each of the devices in Table 1 does not exist. To test the efficacy of our model we developed the following testing ground-truth methodology:

Training and Testing Set: We took up to 1000 historical flows from each device, used 80% for training and 20% for testing. We assume that all of the testing set consists of inliers (i.e. no a priori outliers). We then developed a set of outliers by randomly generating each IPv4 destination octet such that each IP address generated conforms to a non-reserved IP address [20]. Destination ports were randomly generated in the range 1-65535 and protocols were randomly picked from the set (1,6,17) which were the protocols found in the training data.

The OCSVM using the RBF kernel is governed by the two hyper-parameters, ν (nu) and γ (gamma). Gamma sets the radius of the RBF kernel by determining the influence of each example of the decision boundary and ν sets the upper bound on fraction of errors during training and the lower bound on the fraction of support vectors used. For the purposes of this work we set this using the 'scale' option of Sci-kit learn which uses the following equation to determine gamma.

This research examines three methods to establish ν for devices; static ν set uniformly across all devices; a dynamic ν set per device, and a ν tuned to the complexity of the device.

5.3 Static Hyper-Parameter ν

For the static method an average ν is found and applied uniformly across all the device models. To find the average ν, each device was modeled using OCSVM with ν varied over the range of 0.00001 to 0.5. The ν for each device that had the best F1 score was saved and the mean ν value was calculated across all the devices. This average ν was then used to train the model for each device and test for anomalies. This gives a baseline model where there is a balance between precision and recall and where the hyper-parameter ν is set to the same value for each device.

5.4 Dynamic Hyper-Parameter ν

The dynamic method finds the best ν for each device and that ν is applied individually to each model. To find the ν value for individual devices, each device was modeled using OCSVM with ν varied over the range of 0.00001 to 0.5. The ν for each device that had the highest F1 score was then used to train the model for that device and test for anomalies. This gives results that balance precision and recall and a model that is tuned per device.

5.5 Complexity-Tuned Dynamic ν

To tune the model based on complexity a value for ν is found that minimizes false positives for low complex devices. To find a ν value that is tuned to the complexity of the device, each device was modeled using OCSVM with ν varied over the range of 0.00001 to 0.5. The ν for each device that had the highest F_β scores where $\beta = \hat{A}_{DC}$, where $\hat{A}_{DC}$ is the normalized value (between 0 and 1) of the aggregate device complexity as defined in section 4. This search prioritizes minimizing false negatives on low complexity devices as seen in equation 8.

$$F_\beta = (1 + \beta^2)\frac{precision * recall}{(\beta^2 * precision) + recall} \tag{8}$$

6 Enforcement Architecture

The enforcement architecture shown in Figure 8 is based on a centralized model where there is a single device that acts as a router, gateway and access point. To implement the enforcement architecture we use a Raspberry Pi 4. The Raspberry Pi 4 is a single-board computer based on an ARM architecture. We chose this as it is a reasonable representation of the embedded architectures used in today's more powerful home routers, and analyze if it is capable of handling both training the novelty detection model and switching and routing done by the SDN controller and SDN switch.

SDN architectures decouple the control and the data plane in routers and switches. This opens the network to new services, features, and a level of dynamism that was previously not possible. This work leverages the programmability of the network to dynamically allow, block, rate-limit, log and route traffic

based on if the flow is novel, the degree of novelty, and the complexity of the device.

The reference enforcement architecture developed for this work uses the OpenFlow [16] reference soft switch called OpenVSwitch[17]. OpenVSwitch supports OpenFlow versions 1.0-1.5.

RYU is a software defined network controller that implements OpenFlow. In this prototype we use RYU to setup and control OpenVswitch[22]

The flow collector consists of a Raspberry Pi running a netflow collection software called nprobe. Nprobe stores the flows into a MariaDB database.

6.1 Enforcement

Enforcement of the currently proposed test environment examines only values known at connection time. Aggregate flow metrics will be examined in a future work. The connection enforcement stage is only run once at flow connection setup. The connection features include IP header attributes such as IP source, IP destination, port, and protocol. If the model detects an outlier based on the connection features it will use the current device confidence scores and the outlier degree to the flow to calculate the flow trust score.

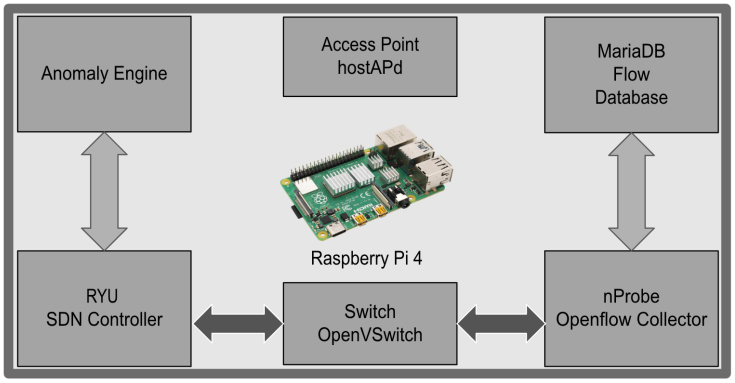

Fig. 8. Enforcement Architecture

In Figure 8 the anomaly detection engine loads previously trained device models stored as serialized python objects and calculates device complexity, behavioral boundary, and flow scores. For flows that do not exist in the current flow rules table of OpenVSwitch, RYU queries the anomaly engine to determine if the flow is an outlier, inlier or significant flow (a flow that was an outlier during the training stage). If a flow is determined to be an outlier and the policy for that device is to drop outliers then the flow is simply not added to the flow table matched rules and is dropped.

The architecture in Figure 8 allows the network to make extremely granular flow decisions on every flow in the network, including inbound/outbound traffic to the Internet and intra-network device traffic. Based on the behavioral boundary there is no need to isolate an entire device, just the flows that are found to be abnormal.

7 Results

7.1 Static Hyper-Parameter ν

Figure 9 shows the OCSVM classifier trained on each device with a static ν and applied uniformly to all devices based on the search that optimizes the F1 score as defined in 5.2. This model has an average false positive rate (FPR) of 0.082 averaged across all devices.

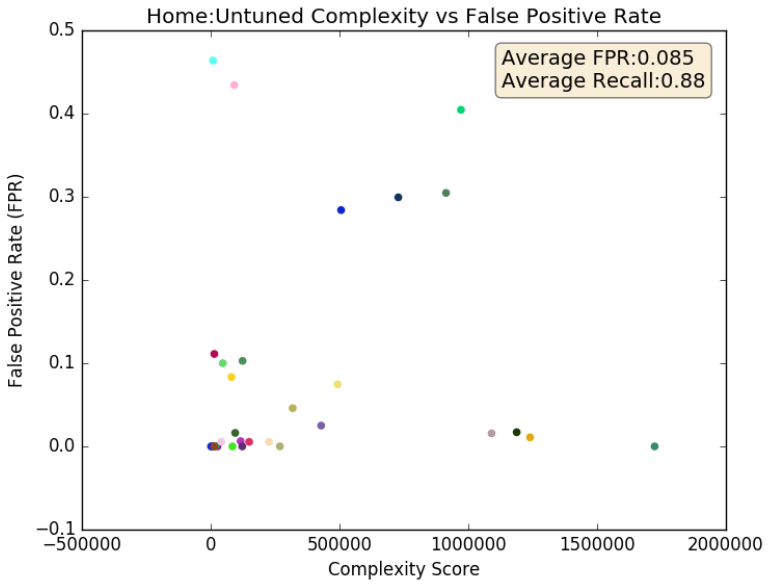

Fig. 9. Un-Tuned Model

Figure 10 shows the OCSVM where each device model is tuned with the ν that optimizes the F1 score. This model has a average false positive rate (FPR) of 0.062.

Figure 11 shows the OCSVM where each device is tuned to the ν that uses the complexity of the device to influence the precision of the F_β score. There is a noticeable drop in the number of false positives overall, however more importantly there is a greater drop in the false positive rate for the lower complexity

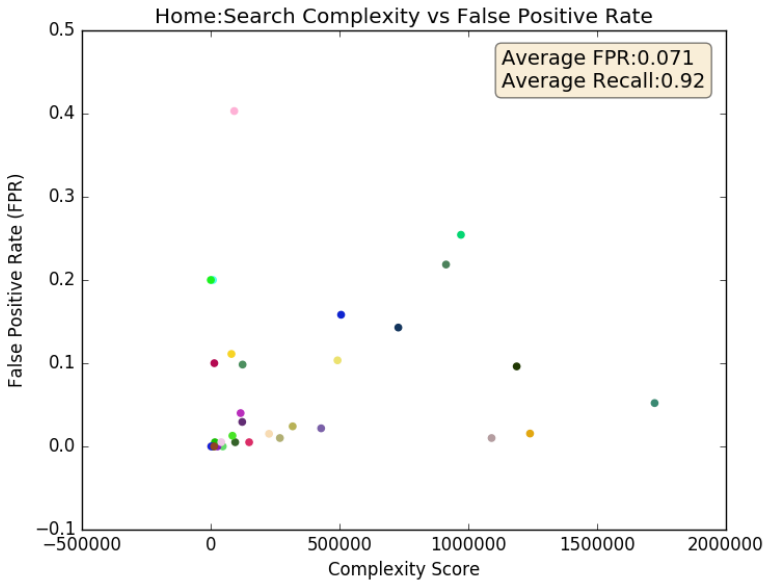

Fig. 10. Search Model

devices. This can be seen in the fact that more of the devices on the left (the low complexity devices) have markedly smaller false positive rates. This is the expected result as we are tuning the F_β score weighted toward precision on these devices.

Table 3. Low Complexity Model Characteristics

Model Type	D_{DC} <5	D_{DC} <4	A_{DC} <Mean	A_{DC} <1 Std Dev
Static	P=0.984 R=0.549 FPR=0.107	P=0.98 R=0.582 FPR=0.114	P=0.911 R=0.893 FPR=0.061	P=0.914 R=0.911 FPR=0.06
Dynamic	P=0.927 R=0.893 FPR=0.072	P=0.916 R=0.581 FPR=0.113	P=0.946 R=0.936 FPR=0.053	P=0.959 R=0.946 FPR=0.044
Tuned	P=0.951 R=0.915 FPR=0.048	P=0.96 R=0.955 FPR=0.033	P=0.975 R=0.878 FPR=0.029	P=0.976 R=0.887 FPR=0.026

Table 3 shows how the three models perform on several subsets of the device space where devices have low complexity measures. The first column shows the model on devices that have a discrete device complexity DD_C of less than 5 (as calculated in section 4.3. The second column shows devices with a D_{DC} of

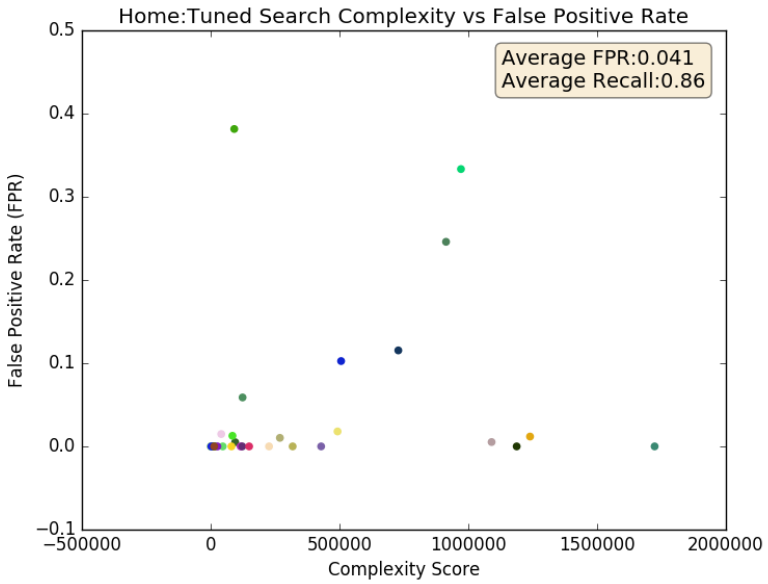

Fig. 11. Tuned Search Model

less than 4. The third column shows devices that have an aggregate complexity A_{DC} less than mean complexity of the set of all devices. Finally, the last column shows the devices that have A_{DC} of less then one standard deviation.

The tuned model outperforms both the dynamic and the static models in terms of precision and false positive rate. This is expected as the tuned model has a higher weight for minimizing false positives than the other two with a small trade-off of lower recall. It is also notable, that the tuned model performs better on the higher complex devices as shown in column 1,2 with the tuned model having better precision *and* recall than the static and dynamic models.

8 Conclusions and Future Work

In this work we established an autonomous and unsupervised method to formally measure the complexity of a network device based solely on the network flows from that device. We show that this complexity measure has a positive correlation to the number of outliers found in an un-tuned anomaly detection engine. We then used this measure of device complexity to develop a behavioral model for each device based on a tuned novelty detection engine. We show that this behavioral model has a lower FPR for all devices and performs better than both the static and dynamic modeling methods. Finally, we propose a network architecture based on SDN to dynamically enforce our model.

In future work, we will look at additional methods of establishing network complexity such as incorporating a second enforcement stage based on aggregate flow features such as bytes per second and packets per second. This will allow the model to account for connections that are normal in the connection attributes, but may be anomalous based on bandwidth. We also apply our complexity model to other domains, such as industrial IoT networks and SCADA based networks. Finally we will look at how we can use a supervised learning approach to finger print devices and use this to bootstrap a behavior model across know common devices and device ecosystems.

References

1. Apthorpe, N., Reisman, D., Feamster, N.: A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. arXiv preprint arXiv:1705.06805 (2017)
2. Bezawada, B., Bachani, M., Peterson, J., Shirazi, H., Ray, I., Ray, I.: Behavioral fingerprinting of iot devices. In: Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security. pp. 41–50. ACM (2018)
3. Bialek, W., Nemenman, I., Tishby, N.: Predictability, complexity, and learning. Neural computation **13**(11), 2409–2463 (2001)
4. Fagan, M., Megas, K., Scarfone, K., M, S.: Core cybersecurity feature baseline for securable iot devices. Tech. rep., National Institute of Standards and Technology (2019)
5. Group, C.C.W.: The c2 consensus on iot device security baseline capabilities. Tech. rep., Consumer Technology Association (2019)
6. Haefner, K., Ray, I.: Complexiot: Behavior-based trust for iot networks. In: 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). pp. 56–65. IEEE (2019)
7. Hafeez, I., Antikainen, M., Ding, A.Y., Tarkoma, S.: Iot-keeper: Securing iot communications in edge networks. arXiv preprint arXiv:1810.08415 (2018)
8. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: Ddos in the iot Mirai and other botnets. Computer **50**(7), 80–84 (2017)
9. Kolmogorov, A.N.: On tables of random numbers. Sankhyā: The Indian Journal of Statistics, Series A pp. 369–376 (1963)
10. Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J.: Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access **5**, 18042–18050 (2017)
11. Manevitz, L.M., Yousef, M.: One-class svms for document classification. Journal of machine Learning research **2**(Dec), 139–154 (2001)
12. Marchal, S., Miettinen, M., Nguyen, T.D., Sadeghi, A.R., Asokan, N.: Audi: Toward autonomous iot device-type identification using periodic communication. IEEE Journal on Selected Areas in Communications **37**(6), 1402–1412 (2019)
13. Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A.R., Tarkoma, S.: Iot sentinel: Automated device-type identification for security enforcement in iot. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 2177–2184. IEEE (2017)
14. Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., Sadeghi, A.R.: Dïot: A federated self-learning anomaly detection system for iot. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 756–767. IEEE (2019)

15. An extensible netflow v5/v9/ipfix probe for ipv4/v6 (2020), https://www.ntop.org/products/netflow/nprobe/
16. Openflow switch erata, open networking foundation, onf ts-001 (2012), https://www.opennetworking.org/wp-content/uploads/2013/07/openflow-spec-v1.0.1.pdf
17. Production quality, multilayer open virtual switch (2019), https://www.openvswitch.org
18. Ortiz, J., Crawford, C., Le, F.: Devicemien: network device behavior modeling for identifying unknown iot devices. In: Proceedings of the International Conference on Internet of Things Design and Implementation. pp. 106–117. ACM (2019)
19. Ren, J., Dubois, D.J., Choffnes, D., Mandalari, A.M., Kolcun, R., Haddadi, H.: Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In: Proceedings of the Internet Measurement Conference. pp. 267–279. ACM (2019)
20. Requirements for internet hosts – communication layers (1989), https://tools.ietf.org/html/rfc1122
21. Rissanen, J.: Stochastic complexity in statistical inquiry. World Scientific (1989)
22. Ryu sdn framework (2019), https://osrg.github.io/ryu/
23. Sachidananda, V., Siboni, S., Shabtai, A., Toh, J., Bhairav, S., Elovici, Y.: Let the cat out of the bag: A holistic approach towards security analysis of the internet of things. In: Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security. pp. 3–10. ACM (2017)
24. Various: Open connectivity foundation (ocf) specification — part 2: Security specification. Standard, Open Connectivity Foundation (2019)
25. Wilson, C., Hargreaves, T., Hauxwell-Baldwin, R.: Benefits and risks of smart home technologies. Energy Policy **103**, 72–83 (2017)
26. Yang, Y., Wu, L., Yin, G., Li, L., Zhao, H.: A survey on security and privacy issues in internet-of-things. IEEE Internet of Things Journal **4**(5), 1250–1258 (2017)
27. Zhao, K., Ge, L.: A survey on the internet of things security. In: 2013 Ninth international conference on computational intelligence and security. pp. 663–667. IEEE (2013)