Automating Microservices for Smart Intrusion Detection

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Computer Engineering

By
Tejas Chumbalkar, Brandon Lee Gaerlan, Karthik Thirugnaman Jagadeesan,
Yash Pamnani

May 2019

APPROVED

Dr. Younghee Park, Project Advisor

ABSTRACT

Automating Microservices for Smart Intrusion Detection

By Tejas Chumbalkar, Brandon Lee Gaerlan, Karthik Thirugnaman Jagadeesan, Yash Pamnani

The Internet of Things (IoT) is a concept in which a huge collection of devices are interconnected and remotely accessible through the internet. With an exponential increase IoT devices, there is a considerable increase in the number of cyber-attacks present in the area of IoT. Some of these attacks include botnets and Distributed Denial of Service (DDoS) attacks. Exposed services, low security, default credentials and low cost are some of the reason attackers target IoT devices. Due to the inherent vulnerabilities in these IoT devices, popular botnet attacks such as Mirai and Bashlite take advantage of the insecure IoT devices and can bring down an entire network infrastructure. Software-Defined Networks (SDN) in such cases can help in mitigating the attacks due to their ability to separate the data plane from the control plane. This plane separation will allow for better fine-grained decisions in relation to traffic routing, load balancing, and even decisions related to security. SDN in conjunction with Network-Function Virtualization (NFV) can be used to manage network resources by deploying Virtualized Network Functions (VNFs) to reduce the overhead of traditional networks while increasing network capacity and defense capabilities in today's modern networks. While most modern Intrusion Detection Systems (IDS) are sufficient for today's network security, they are inefficient and have poor resource utilization with regards to processing the network traffic. We propose to utilize the SDN controller by implementing lightweight NFs with existing network protocols along with IoT protocols

in the data plane using protocol-specific VNFs. Machine learning and Deep Learning algorithms will then be implemented in the control plane to analyze IoT related network traffic in real-time to defend against external attacks and respond accordingly.

Acknowledgments

The authors are deeply indebted to Professor Dr. Younghee Park for her invaluable comments and assistance in the preparation of this study.

1

Automating Microservices for Smart Intrusion Detection

Brandon Lee Gaerlan, Tejas Chumbalkar, Karthik Thirugnanam Jagadeesan, Yash Pamnani

***Abstract*—The Internet of Things (IoT) is a concept in which a huge collection of devices are interconnected and remotely accessible through the internet. With an exponential increase IoT devices, there is a considerable increase in the number of cyber-attacks present in the area of IoT. Some of these attacks include botnets and Distributed Denial of Service (DDoS) attacks. Exposed services, low security, default credentials and low cost are some of the reason attackers target IoT devices. Due to the inherent vulnerabilities in these IoT devices, popular botnet attacks such as Mirai and Bashlite take advantage of the insecure IoT devices and can bring down an entire network infrastructure. Software-Defined Networks (SDN) in such cases can help in mitigating the attacks due to their ability to separate the data plane from the control plane. This plane separation will allow for better fine-grained decisions in relation to traffic routing, load balancing, and even decisions related to security. SDN in conjunction with Network-Function Virtualization (NFV) can be used to manage network resources by deploying Virtualized Network Functions (VNFs) to reduce the overhead of traditional networks while increasing network capacity and defense capabilities in today's modern networks. While most modern Intrusion Detection Systems (IDS) are sufficient for today's network security, they are inefficient and have poor resource utilization with regards to processing the network traffic. We propose to utilize the SDN controller by implementing lightweight NFs with existing network protocols along with IoT protocols in the data plane using protocol-specific VNFs. Machine Learning and Deep Learning algorithms will then be implemented in the control plane to analyze IoT related network traffic in real-time to defend against external attacks and respond accordingly.**

***Index Terms*—IoT, SDN, Intrusion Detection, NFV**

I. Introduction

Internet-of-Things (IoT) is a system of devices with embedded software intelligence and sensors connected over the internet to collect, share and analyze data. Kevin Ashton, co-founder of the Auto-ID Center at MIT, first to mention the term internet of things in a presentation to Procter & Gamble (P&G) in 1999[14]. After the introduction of IoT, this idea of interconnecting devices and sharing has allowed us to perform many different tasks such as monitor health, keep people connected, etc. However, due to its rapid growth, there have been several vulnerabilities and exploits which this network-of-devices susceptible to attacks. As a result, we need to enhance the existing framework to provide security in such situations.

SDN has been proven the most successful framework to be deployed in a distributed network environment to provide robust security. SDN is a networking concept which essentially allows network administrators to separate the control-plane and the data-plane of a switch or router allowing the switch faster packet-forwarding capabilities without the overhead of making control-plane decisions using high level languages and Application Programming Interfaces (APIs). An SDN-controller is deployed which acts a centralized point-of-control for the traffic flowing through the network. This traffic can then be filtered and controlled by using various flow rules based on network requirements. Scalability, centralized control, rapid deployment are some of the major advantages of SDN. Since a majority of security policies are implemented in the SDN controller, it is natural that development should be targeted towards new applications built on top of the controller.

Coupled with SDN, NFV provides the flexibility of being able to deploy a middlebox functionality with ease by virtualizing NFs and Containers as a mode of deployment. To build an intelligent IDS, we need to ensure that the VNFs can detect network intrusions with minimal intervention from the network administrators/operators while also adjusting to the fluctuating demands to run on multiple hardware platforms for portability and interoperability.

Deep Learning is a branch of machine learning which teaches machines to learn based off unsupervised data. This makes Deep Learning ideal for a number of scenarios such as being able to make efficient routing decisions to provide fast and efficient end-to-end delivery[1], the ability to perform sophisticated network intrusion without any labeled data. By implementing such models, we relieve the burden of the security administrators having to manually declare new signatures for unknown traffic.

In this paper we propose an IDS that isolates and operates on network protocol specific traffic. Implementing our IDS in this manner allows for our IDS systems to be lightweight thus reducing the significant processing overhead that existing IDS currently suffer from. In addition, to the VNF containers, we will also be utilizing the SDN controller to embed deep learning functions such as anomaly detection, feature set reduction, etc. By implementing our IDS in this manner, we can make our system robust by scaling our systems accordingly to demand while ensuring highly accurate detections for real-world network traffic detection.

II. Problem Statement / Project Architecture

In this section, we describe the existing architectures that have been developed in regard to existing solutions for intrusion detection along with some of their aforementioned negativities. The architecture for our project is related to a

1

smarter and lighter defense mechanism utilizing various different technologies. Two main technologies utilized throughout the course of this project will be SDN and NFV. Since the SDN controller has a global view of the entire network, routing decisions can be made on the fly through one controller while NFV allows for Network Functions (NFs) to be embedded as software running on Commercial off the shelf (COTS) machines. This kind of flexibility provides numerous benefits such as decreased Operational Expenditures, Ease of scaling, etc.

By utilizing SDN and NFV, we enable smarter network functions that are capable of processing large loads of data while being scalable, lightweight, and accurate. The following describes each of the modules/components of our system in more detail in the following subsections.

A. *Orchestration and Deployment*

NFV solves the issues that traditional networks are plagued with: inability to scale properly, harder to deploy, difficult to configure, etc. For a better intelligent intrusion detection, we need to ensure that the components of our system are lightweight, deployable, can be orchestrated, and are able to address demands by scaling the clusters up or down. For our research, the SDN Controller will cooperate with the Management and Orchestrator (MANO) (in this case, Kubernetes) to instantiate additional Network Functions based on a variety of factors including CPU Metrics, Packet Rate, Benign/Malicious traffic detection rate, etc. For load balancing, CPU Metrics will be utilized to instantiate additional switches and/or Machine Learning Models in order to ensure that traffic and prediction queries are handled appropriately based on traffic and system utilization. The containers instantiated to handle the traffic will contain OpenVSwitch, along these containers to have their traffic flow orchestrated by the SDN Controller (likely to sit in the same space as the MANO). In addition to the load balancing and routing capabilities provided, specific NFs can be instantiated based on the results of the Security NF to drop malicious traffic, forward the malicious traffic to a honeypot for further analysis, or even instantiate another IDS for additional alerts.

B. *Machine Learning*

Network Intrusion Detection Systems (NIDS) have been commonly used to ensure that Network and Security Administrators are alerted first hand upon any intrusions that are being conducted in the networks of companies, businesses, etc. However, while existing solutions like Snort, Bro and Suricata[12][13] are sufficient for the normal use case of intrusions, complex and well-crafted intrusions are much harder to detection and require much more intensive rules and signatures to be embedded in the NIDS making them harder to manage. A popular approach to a more robust NIDS is to create one enabled with a machine learning model [24]. In most instances, traffic needs to be gathered from a honey pot to ensure that a mixture of benign and malicious traffic can be observed. After collecting the data, the data needs to be preprocessed for the machine learning model to be able to accurately decipher whether the packet that has arrived will be benign or malicious. Previously observed works related to this approach of machine learning for intrusion detection can be found in [5][7][11]. Other works involve using Deep Learning for even more robust intrusion detection [18][20][22][23]. There have also been other approaches in the machine-learning space for smarter intrusion detection. One approach is to use an online machine-learning algorithm to train and learn from single fed data on the fly. In [25], Mirsky et al, have used an online approach where they use auto encoders to detect malicious traffic on the fly. The advantage to this approach, if done correctly, will allow for shorter initial training time, accurate and consistent predictions, and lightweight system utilization. However robust the online machine-learning approach maybe, it still suffers from having to ensure that the machine learning model is constantly online. If there is a decaying factor attached to the machine learning model, then the accuracy of the model may decline overtime due to not having received any data due to the model being unavailable.

III. TECHNOLOGY DESCRIPTION

A. *Docker*

Docker is a program that allows for operating system virtualization by managing the operating system kernel allowing for isolated container-based applications to run any application in any environment that supports Docker. Docker packages a service into a standard unit known as a Docker image. The full package of an image includes the code, runtime and system libraries. A Docker container is a light weight machine and has an instantaneous boot up to run any package and provides a level of isolation from other containers and processes that are running on the system host. Utilizing Docker allows for various workloads to scale up/down quickly and promptly. These containers also provide consistent environments ensuring proper workflow in various development environments which is ideal for fast and rapid developments. Unlike Virtual Machines (VMs), containers don't come with their own Operating system and this is the difference between megabytes and gigabytes. Docker containers are single process and stateless which allows for safe deployments of applications while working efficiently to share resources allowing for simple scalability.

B. *Kubernetes*

Docker is a great tool for running container applications providing for consistent environments, fast and frequent developments, etc. However, Docker is not built for orchestration to manage a cluster of Docker containers. Kubernetes is an orchestration system used for automation of the deployment, operation, and scaling of Docker containers, check for health status and resource availability. Scheduling and on the fly decision making are important features of the container orchestration engine. Container as a service can be done using Kubernetes. Dynamic provisioning and horizontal scalability and fault tolerance are key for micro service architecture.

C. Prometheus:

Prometheus collects metrics at scale using the HTTP another client of microservices. Prometheus can collect thousands of targets, millions of time series and no dependencies, very easy to scale. Prometheus provides multi-dimensional data model and acts as a powerful querying model and can be integrated with hundreds of tools. It can be used for monitoring services via exporters and it can be also used to monitor Kubernetes clusters.

D. Grafana

Grafana is connected with Prometheus in order to get the performance data of the Kubernetes cluster. We can import a dashboard specific to Kubernetes and the dashboard will be used to analyze the performance of the Kubernetes cluster.

E. In-House Cloud

Cloud infrastructure provides a independent platform to host high process applications. It boosts the productivity as the run time of the application hosted on cloud is far greater than the one hosted on a local machine. Cloud platforms provide high availability and scalability for the applications. There is also no physical overhead of managing the server and related resources. We have used an in-house cloud instance hosted by the university. This instance forms the central component of the infrastructure. We host the SDN controller and the Kubernetes Master on the cloud instance. For security, we have used Nginx and Ngrok proxy servers to host applications on HTTP (80) and HTTPS (443) rather than exposing it to a local IP address and a random port number.

IV. PROJECT DESIGN

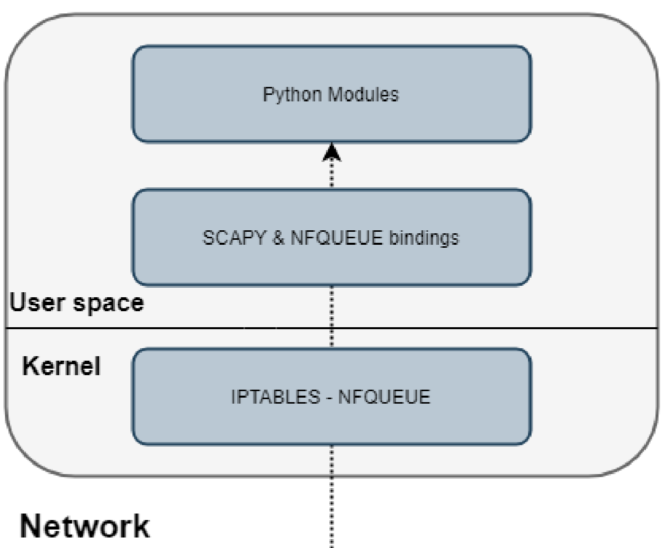


Fig 1: Scapy Architecture

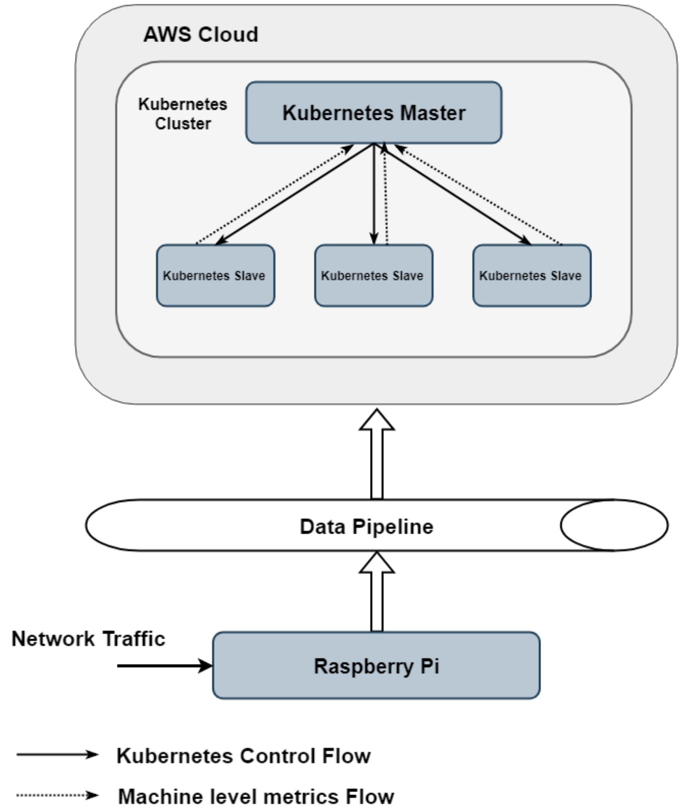


Fig 2: Project Architecture

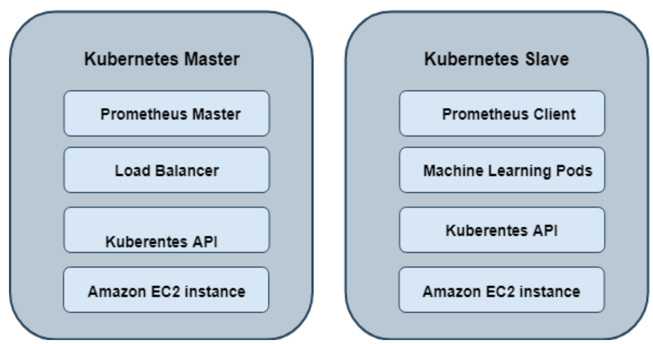


Fig 3: Kubernetes Components

V. PROJECT IMPLEMENTATION

We have designed a microservice hybrid network infrastructure that includes an IoT gateway device and our in-house cloud server. The gateway device is an OpenVSwitch (OVS) that acts as a gateway for external network traffic. This element of the infrastructure is responsible for sending the packets to the SDN Controller based off of packet-in messages. Currently we examine packets related to OSI layers along with Wi-Fi related traffic information. According to the Kyoto dataset, we extract 14 features from the packet. These features are used to train and test the Kyoto dataset [19].

Below are the 14 secondary features:

1. Duration of the connection.
2. Service type of the connection.

3. Source bytes.
4. Destination bytes.
5. Count: Number of connections where the source and destination IP address are same as that of the current connection in the 2 second timeframe.
6. Same_Service_Rate: Percentage of connections with the same service type with respect to the Count feature.
7. Serror_Rate: Percentage of the connection with the SYN errors as of Count feature.
8. Service_Serror_Rate: Percentage of the connection with the SYN as of Serror_Rate.
9. Destination_Host_Count: Number of connections from past 100 connection that have same source and destination IP address as that of the current connection.
10. Destination_Host_Service_Count: Number of connections from past 100 connection that have same destination IP address and service type as that of the current connection.
11. Destination_Host_Same_Source_Port_Rate: Percentage of the connection with the same source port as of the current connection in Destination_Host_Count feature.
12. Destination_Host_Serror_Rate: Percentage of the connection with the SYN errors as of Destination_Host_Count feature.
13. Destination_Host_Service_Serror_Rate: Percentage of the connection with the SYN errors as of Destination_Host_Service_Count feature.
14. Transport layer Protocol of the packet.

These secondary feature information is extracted from the packet-in messages to the controller. The information is saved in a CSV file format and is made available on an REST API endpoint. The Raspberry PIs hosting the machine learning model can host the API endpoint which will be used by the SDN Controller to fetch the data and feed it into the containerized machine learning model for training and testing purposes.

The codebase for the feature extraction model is done in Python and written as a module for the Ryu SDN Controller. Below is the output of the feature extraction model.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Duration	Service	Source bytes	Destination bytes	Count	Same srv rate	Serror rate	Srv serror rate	Dst host count	Dst host srv count	Dst host same src	Dst host serror rate	Dst host srv serror rate	Protocol
0	https	93	0	0	0	0	0	1	1	1	0	0	TCP
0.00436	https	171	0	0	0	0	0	1	1	1	0	0	TCP
0.00687	https	225	0	0	0	0	0	1	1	1	0	0	TCP
0.00906	https	225	60	0	0	0	1	1	1	0	1	1	TCP
0.00019	https	225	120	0	0	0	1	1	1	0	1	1	TCP
0.00556	https	225	180	0	0	0	1	1	1	0	1	1	TCP
0.00562	https	279	180	0	0	0	1	1	1	1	1	1	TCP
0	dns	85	0	0	0	0	0	1	1	1	0	0	UDP
0	dns	85	0	0	0	0	0	1	1	1	0	0	UDP
0	dns	85	0	0	0	0	0	1	1	1	0	0	UDP
0.01239	dns	170	0	0	0	0	0	1	1	1	0	0	UDP
0.00438	dns	170	194	0	0	0	0	1	2	0	0	0	UDP
-0.0045	dns	85	194	0	0	0	0	1	2	0	0	0	UDP
0.00471	dns	170	435	0	0	0	0	1	2	0	0	0	UDP
0	https	74	0	0	0	0	0	1	1	1	0	0	TCP
0.03603	https	74	60	0	0	0	0	1	1	0	0	0	TCP

Fig 4: Secondary Features

We leverage the high availability and scalability features offered by the in-house cloud infrastructure. Kubernetes is used to manage our network infrastructure. Kubernetes operates on a master-slave configuration. In our project, we form a cluster of a single master node and three slave nodes. The master node being the cloud instance and Raspberry Pi's as the slave nodes.

In order to detect intrusions in our network, we need to ensure that we have a robust machine-learning model that provides accurate results based on previously trained data, in this case the Kyoto Dataset[19]. The machine learning utilized for the attack detection is logistic regression and random forest classification. Logistic regression is a model that works on predictive analysis by utilizing the sigmoid function for classifying data into different classes based on probability.

Docker is used to create a microservice infrastructure. Each service model in the network infrastructure is hosted on a docker container. These docker containers are created as a Deployment object and deployed into the Kubernetes cluster by the master node. Also, a Service object is created specific to each Deployment object to have a constant reachable endpoint to the container (even if the container fails/restarts).

The Kubernetes Master node deploys a Machine Learning model as a Deployment object on the slave nodes. A load balancer provided by traffic is used as a load balancer/ingress service. This deployment object is used to load balance the traffic onto the slave nodes hosting the ML model. The real time secondary feature generated by Raspberry pi is the intended traffic for the traffic hosted load balancer.

```
root@master:/home/seed# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
example-app-6bb64c978b-x8szr      1/1     Running   1          3d7h
prometheus-prometheus-0           3/3     Running   4          3d7h
```

Fig 5: Kubernetes Pods

Along with the application specific pods, Kubernetes manages the health and network management of the nodes inside the cluster. Kubernetes taints a node and blocks further deployment of docker containers if the node is found unhealthy. The parameters of the health check is determined by CPU, memory, Disk I/O usage.

```
root@master:/home/seed# kubectl get svc -n monitoring
NAME                                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
alertmanager-operated                   ClusterIP   None             <none>        9093/TCP,6783/TCP   4d7h
monitoring-grafana                      ClusterIP   10.99.232.148    <none>        80/TCP              4d7h
monitoring-kube-state-metrics           ClusterIP   10.99.184.67     <none>        8080/TCP            4d7h
monitoring-prometheus-node-exporter     ClusterIP   10.100.192.50    <none>        9100/TCP            4d7h
monitoring-prometheus-oper-alertmanager ClusterIP   10.108.179.48    <none>        9093/TCP            4d7h
monitoring-prometheus-oper-operator     ClusterIP   10.111.110.162   <none>        8080/TCP            4d7h
monitoring-prometheus-oper-prometheus   ClusterIP   10.107.36.232    <none>        9090/TCP            4d7h
prometheus-operated                     ClusterIP   None             <none>        9090/TCP            4d7h
```

Fig 6: Kubernetes Service I

```
root@master:/home/seed/Downloads# kubectl get svc -n monitoring
NAME                                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
alertmanager-operated                   ClusterIP   None             <none>        9093/TCP,6783/TCP   4d6h
monitoring-grafana                      ClusterIP   10.99.232.148    <none>        80/TCP              4d6h
monitoring-kube-state-metrics           ClusterIP   10.99.184.67     <none>        8080/TCP            4d6h
monitoring-prometheus-node-exporter     ClusterIP   10.100.192.50    <none>        9100/TCP            4d6h
monitoring-prometheus-oper-alertmanager ClusterIP   10.108.179.48    <none>        9093/TCP            4d6h
monitoring-prometheus-oper-operator     ClusterIP   10.111.110.162   <none>        8080/TCP            4d6h
monitoring-prometheus-oper-prometheus   ClusterIP   10.107.36.232    <none>        9090/TCP            4d6h
prometheus-operated                     ClusterIP   None             <none>        9090/TCP            4d6h
```

Fig 7: Kubernetes Service II

Prometheus, a metrics and alerting tool is used in integration with Kubernetes to scrape metric information at machine as well as application level. It is a data model operating on time series data. The main Prometheus server has a pull mechanism to collect the data over HTTP protocol.

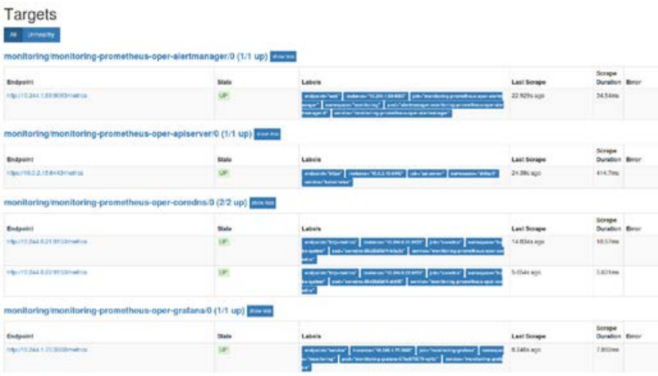
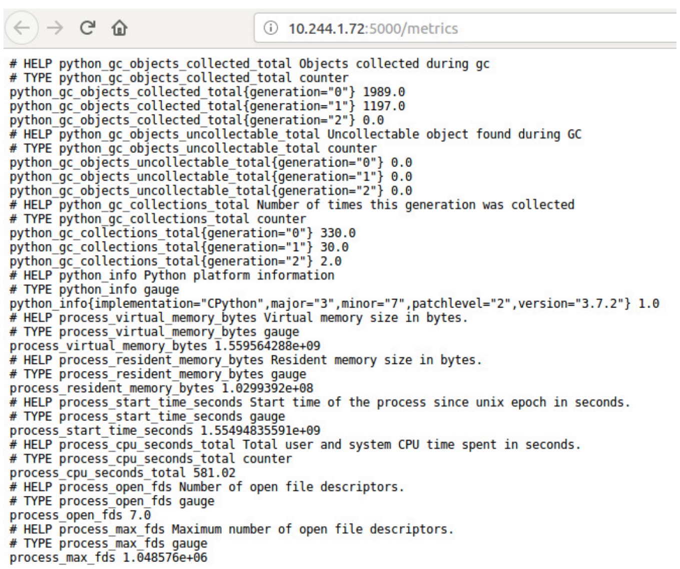


Fig 8: Prometheus Target I

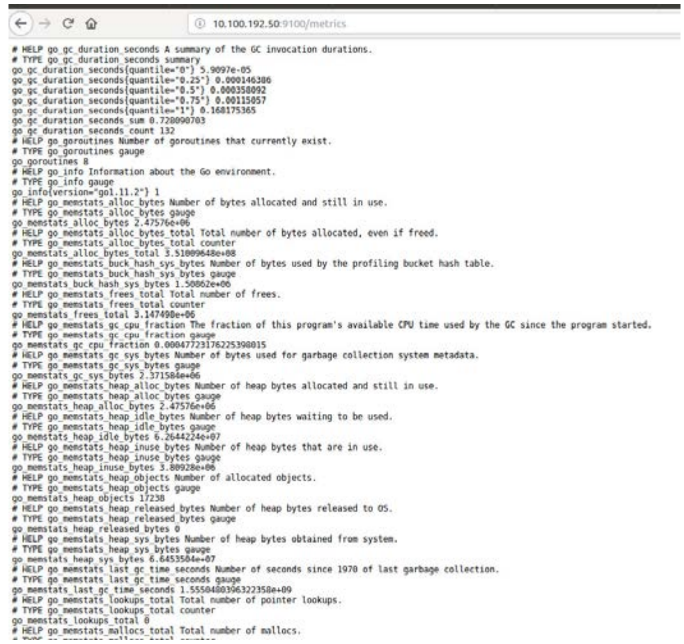
Fig 9: Prometheus Target II

A Node exporter is used to collect machine level metrics over a valid HTTP API endpoint.



```
10.244.1.72:5000/metrics
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 1989.0
python_gc_objects_collected_total{generation="1"} 1197.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 330.0
python_gc_collections_total{generation="1"} 30.0
python_gc_collections_total{generation="2"} 2.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="7",patchlevel="2",version="3.7.2"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.559564288e+09
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.0299392e+08
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.55494835591e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 581.02
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 7.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
```

Fig 10: Node Metrics I



```
10.100.192.50:9100/metrics
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 5.9097e-05
go_gc_duration_seconds{quantile="0.25"} 0.000146386
go_gc_duration_seconds{quantile="0.5"} 0.000358092
go_gc_duration_seconds{quantile="0.75"} 0.00115057
go_gc_duration_seconds{quantile="1"} 0.168175365
go_gc_duration_seconds_sum 0.728090703
go_gc_duration_seconds_count 132
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.11.2"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.47576e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 3.51009648e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.50862e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 3.147498e+06
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0.00047723176225398015
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 2.371584e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.47576e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 6.2644224e+07
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.80928e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 17238
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 0
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 6.6453504e+07
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 1.5550480396322358e+09
# HELP go_memstats_lookups_total Total number of pointer lookups.
# TYPE go_memstats_lookups_total counter
go_memstats_lookups_total 0
# HELP go_memstats_mallocs_total Total number of mallocs.
```

Fig 11: Node Metrics II

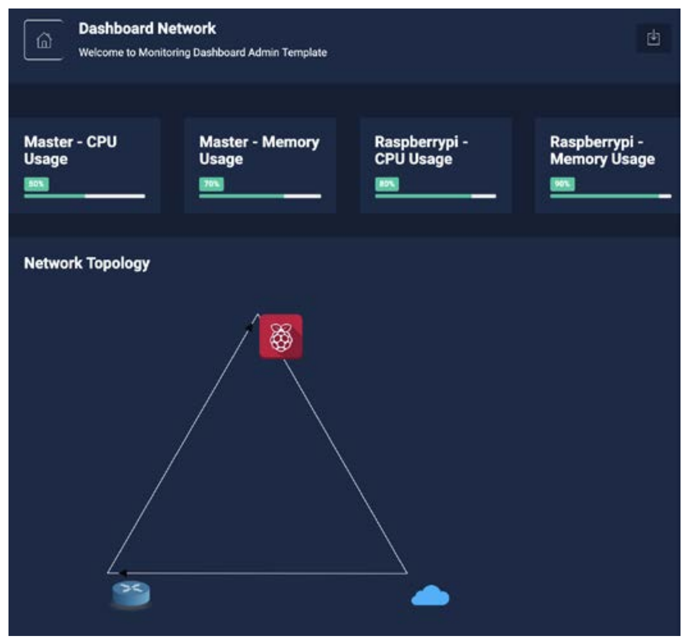


Fig 12: Web UI

The Master server has parameters like CPU metrics monitoring and Health status of the nodes whether the traffic is live or not. Further information regarding the Kubernetes clusters will also be displayed on the dashboard. On the other hand, slave nodes display the live output and the CPU metrics. The dashboard is a web application with an endpoint connected with the in-house cloud instance and the dashboard is updated dynamically through JSON data. Further drop-down option is given for the users to sort by services. Each node displays the parameters associated with them which are nothing but the feature extraction parameters of the Machine learning model.

VI. TESTING AND VERIFICATION

To test if our Kubernetes cluster is working properly we can check the Kubernetes Dashboard. The Kubernetes dashboard with the cluster details are shown:

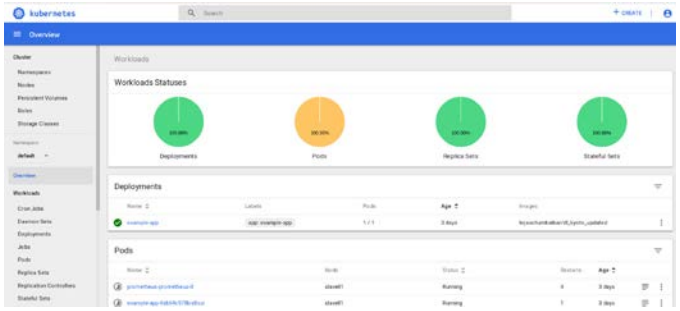


Fig 13: Kubernetes Dashboard I

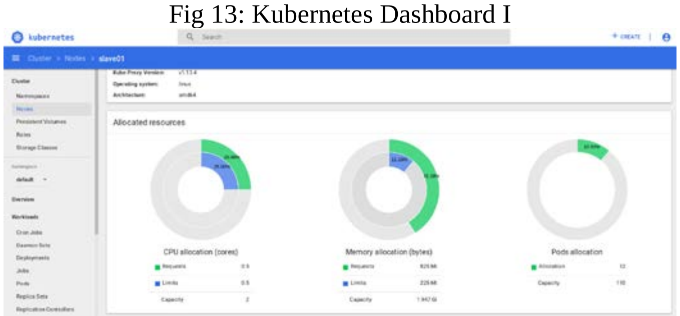


Fig 14: Kubernetes Dashboard II

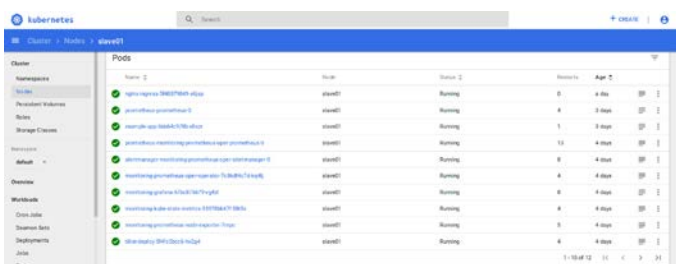


Fig 15: Kubernetes Dashboard III

Fig 16: Prometheus and Grafana Dashboard showing the monitoring details of the Kubernetes cluster.

Fig 17: Grafana I

Fig 18: Grafana II


Fig 19: Grafana III

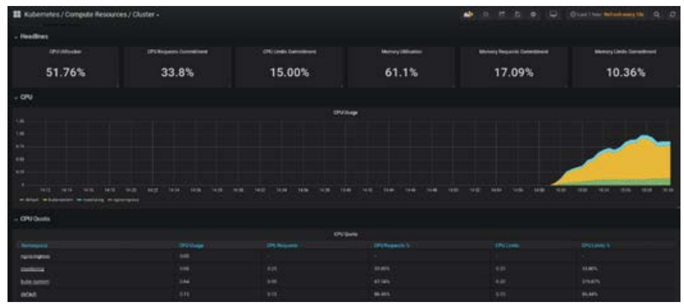


Fig 20: Grafana IV

The Machine Learning model is used to detect whether the network traffic is malicious or not and the output of the Machine learning model is shown here:

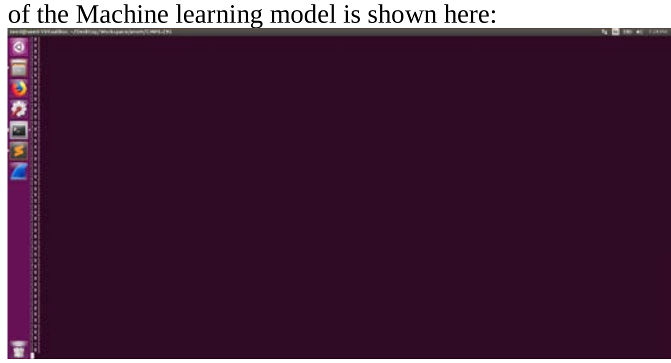


Fig 21: Machine Learning Output

The Network topology is visualized and the output is shown as a dashboard which can be seen as follows:

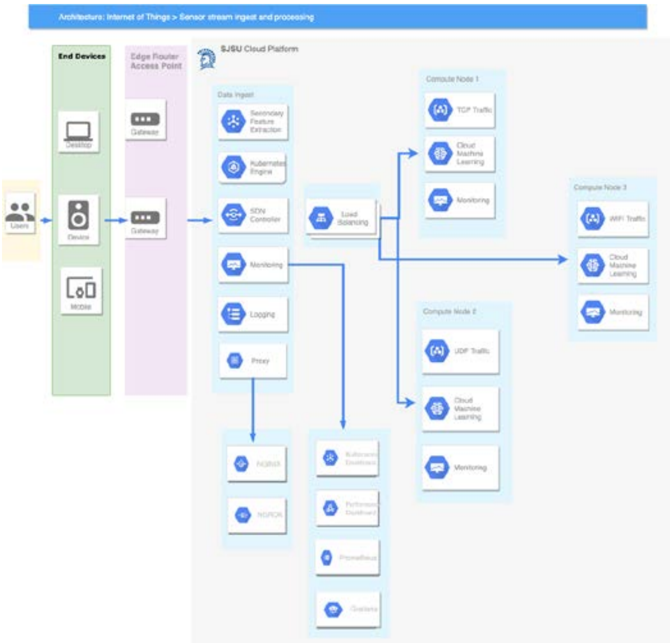


Fig 22: Network Topology

VII. SUMMARY

In this project, we are making a lightweight and resource-optimized IDS. To achieve this, we use Kubernetes to implement a master-slave network topology which greatly helps in managing a dynamic and scalable network system. Kubernetes also helps in monitoring metrics like CPU usage, memory usage, etc. of all the nodes in the network. To reduce the size of our intrusion detection methods, we make them protocol-specific and convert them into VNFs by using Docker containers to run the functions on the respective machines. Kubernetes and Docker work well together which helps in deploying the protocol-specific to any node in the network with great speed and ease. In the end, we use Grafana and Prometheus for data-visualization to create a user-friendly dashboard.

VIII. RELATED WORKS

Thanks to the advents of SDN and NFV, networks are virtualized, scalable, automatable, and manageable based on the former while the later allows for robust network infrastructures to be rapidly configured and deployed while ensuring dynamic adjustments to ever changing business requirements. However, even with the advancement in this area of network infrastructures, security threats and vulnerabilities are still existent especially in the new paradigm of networking.

ClickOS is high-performance softwarized middlebox platform that allows for very lightweight NFs to be instantiated thus ensuring that a minimum amount of resources are utilized while ensuring the maximum performance possible[10]. Though ClickOS presents us with a variety of positive features, the development and management of these NFs are much more complex than our existing solution thus rendering one of the traits of SDN and NFV almost ineffective. A better approach would be to utilize Docker to instantiate NFs due to Docker's ease of use for deploying container applications. One such study that already utilizes this approach is called Deep NFV. Their approach utilizes container-based applications to run Deep Learning models on edge nodes. By utilizing this approach, the burden on centralized servers can be relieved by having specific computations done on the edge node rather than a centralized server perform all of the processing which can have some performance implications in terms of processing power, system utilization, etc.

IX. CONCLUSIONS

Most of the available commercial Intrusion Detection Systems offer better security in exchange for system resources. Deploying Docker containers with compact, specific virtualized network functions use less resources than they sound. This makes our IDS resource-effective without compromising the level of security. Also using Machine Learning and Deep Learning algorithms for intrusion detection makes our IDS more robust and effective. This project is an attempt to combine concepts of SDN, NFV & IDS with the goal to provide best, resource-effective, scalable and dynamic security to any network

ACKNOWLEDGMENT

The authors are deeply indebted to Dr. Younghee Park for her invaluable ideas and guidance in the implementation of this project.

REFERENCES

[1] Pei, J., Hong, P., & Li, D. (2018, May). Virtual Network Function Selection and Chaining Based on Deep Learning in SDN and NFV-Enabled Networks. In 2018 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 1-6). IEEE.

[2] Ahmed, M. E., Kim, H., & Park, M. (2017, October). Mitigating dns query-based ddos attacks with machine learning on software-defined networking. In Military Communications Conference (MILCOM), MILCOM 2017-2017 IEEE (pp. 11-16). IEEE.

[3] Vilalta, R., Ciungu, R., Mayoral, A., Casellas, R., Martinez, R., Pubill, D., ... & Verikoukis, C. (2016, December). Improving security in internet of things with software defined networking. In Global Communications Conference (GLOBECOM), 2016 IEEE(pp. 1-6). IEEE.

[4] Xu, T., Gao, D., Dong, P., Zhang, H., Foh, C. H., & Chao, H. C. (2017). Defending against new-flow attack in sdn-based internet of things. IEEE Access, 5, 3431-3443.

[5] Park, Y., Chandaliya, P., Muralidharan, A., Kumar, N., & Hu, H. (2017, March). Dynamic Defense Provision via Network Functions Virtualization. In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (pp. 43-46). ACM.

[6] Gonzalez, C., Charfadine, S. M., Flauzac, O., & Nolot, F. (2016, July). SDN-based security framework for the IoT in distributed grid. In Computer and Energy Science (SpliTech), International Multidisciplinary Conference on (pp. 1-5). IEEE.

[7] Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016, October). Deep learning approach for network intrusion detection in software defined networking. In Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on (pp. 258-263). IEEE.

[8] Mosenia, A., & Jha, N. K. (2017). A comprehensive study of security of internet-of-things. IEEE Transactions on Emerging Topics in Computing, 5(4), 586-602.

[9] Li, L., Ota, K., & Dong, M. (2018). DeepNFV: A Light-weight Framework for Intelligent Edge Network Functions Virtualization. IEEE Network, (99), 1-6.

[10] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., & Huici, F. (2014, April). ClickOS and the art of network function virtualization. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation(pp. 459-473). USENIX Association.

[11] Doshi, R., Apthorpe, N., & Feamster, N. (2018). Machine Learning DDoS Detection for Consumer Internet of Things Devices. arXiv preprint arXiv:1804.04159.

[12] *Suricata-vs-snort*. (2016, July 12). Retrieved from Aldeid: https://www.aldeid.com/wiki/Suricata-vs-snort#Environment_.26_methodology

[13] Thongkanchorn, K., Ngamsuriyaroj, S., & Visoottiviseth, V. (2013). *Evaluation Studies of Three Intrusion Detection Systems under Various Attacks and Rule Sets.* Bangkok: IEEE.

[14] Rouse, M. (n.d.). *Internet Of Things (IoT)*. Retrieved from TechTarget: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[15] Wallen, J. (2017, June 01). *Five nightmarish attacks that show the risks of IoT security*. Retrieved from ZDNet: https://www.zdnet.com/article/5-nightmarish-attacks-that-show-the-risks-of-iot-security/

[16] *The 5 Worst Examples of IoT Hacking and Vulnerabilities in Recorded History*. (2017, May 10). Retrieved from IoT for all: https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/

[17] Deogirikar, J., & Vidhate, A. (2017). Security Attacks in IoT: A Survey. *International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)* (pp. 32-37). IEEE.

[18] A Deep Learning Approach for Network Intrusion Detection System Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam College Of Engineering The University of Toledo Toledo, OH-43606, USA {quamar.niyaz, weiqing.sun, ahmad.javaid, mansoor.alam2}@utoledo.edu

[19] Protic, Danijela. (2018). Review of KDD Cup '99, NSL-KDD and Kyoto 2006+ datasets. Vojnotehnicki glasnik. 66. 580-596. 10.5937/vojtehg66-16670.

[20] N. Shone, T. N. Ngoc, V. D. Phai and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, Feb. 2018. doi: 10.1109/TETCI.2017.2772792

[21] Machine-learning based Threat-aware System in Software Defined Networks Chungsik Song, Younghee Park, Keyur Golani , Youngsoo Kim , Kalgi Bhatt , Kunal Goswami , Computer Engineering Department , Electronic Engineering Department, San Jose State University, San Jose, CA USA

[22] Semi-Supervised Statistical Approach for Network Anomaly Detection Naila Belhadj Aissaa , Mohamed Guerroumia a University of Science and Technology Houari Boumediene, Algiers, Algeria The 6th International Symposium on Frontiers in Ambient and Mobile Systems (FAMS 2016)

[23] Deep Learning Approach for Network Intrusion Detection in Software Defined Networking Tuan A Tang , Lotfi Mhamdi , Des McLernon, Syed Ali Raza Zaidi and Mounir Ghogho ∗School of Electronic and Electrical Engineering, The University of Leeds, Leeds, UK. International University of Rabat, Morocco

[24] Harale, N. and Meshram, D. (2016). Data Mining Techniques for Network Intrusion Detection and Prevention Systems. *International Journal of Innovative Research in Computer Science & Technology*, pp.175-180.

[25] Y. Mirsky, T. Doitshman, Y. Elovici and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection", *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. Available: 10.14722/ndss.2018.23204 [Accessed 3 May 2019].