Technical Report TR3: ISIF ASIA Funded Project

30 April 2019

Security Architecture for IoT

Prof. Vijay Varadharajan, Dr Uday Tupakula, Kallol Karmakar

Advanced Cyber Security Engineering Research Centre (ACSRC)

Faculty of Engineering and Built Environment

The University of Newcastle

Executive Summary

The report presents project summary on the “Software Defined Networks based Security Architecture for IoT Infrastructures” project funded by the ISIF group. There are three milestones for the project with specific deliverables for each milestone. As part of first milestone, we have conducted detailed survey of attacks related to IoT and proposed the design and development of feature distributed malware attacks for IoT. We then developed some security requirements that need to be considered for designing and developing security architecture for IoT Applications. Please refer to Milestone 1 report for more details on the outcomes. As part of second milestone, we developed a lightweight mutual authentication protocol based on a novel public key encryption scheme for smart city applications. The proposed protocol takes a balance between the efficiency and communication cost without sacrificing the security. Please refer to Milestone 2 report for more details on the outcomes. In this report we present a summary of the outcome of third milestone ”Security Architecture for IoT” which consists of the following tasks: i) Report on the Design of Security Architecture and Attack Detection in IoT Infrastructures and ii) Proof of Concept of Security Architecture for IoT.

As part of third milestone, we have deigned and developed a security architecture for IoT networks by leveraging the underlying features supported by Software Defined Networks (SDN). Our security architecture restricts network access to authenticated IoT devices. We use fine granular policies to secure the flows in the IoT network infrastructure and provide a lightweight protocol to authenticate IoT devices. The security policies are based on parameters such as user, device, switch/gateway, location, routing information, services accessed as well as trust labels associated with the switches and controllers in different domains. We have also considered on demand security of user data by encrypting the traffic at the edge device and decrypting the traffic at a device closer to the destination (which is intended to receive the traffic. A novel feature of the proposed architecture is its ability to specify path based security policies, which is a distinct advantage in SDN enabled IoT infrastructures. For instance, certain IoT communications need to go through a path of switches with certain security attributes. Such path based policies are critical for secure applications but also useful for normal applications with different requirements (e.g. bandwidth requirements for different traffic such as audio versus video traf-

fic). At another level, our policies will be able to specific rules reflecting social aspects such as devices belonging to certain groups of users such as ethnicity or gender as well as constraints related to certain cultures such as different cultural organizations and their attributes. Such features can address various characteristics associated with social norms and ethics associated with the data collected from these devices as well as expected interactions between various devices in different domains. These can be achieved due to fine granular nature of the policies and its ability to reflect the context and their associated characteristics.
An overall feature of the security architecture is its ability to make use of the dynamic visibility of the network connectivity and the flows in the network in achieving dynamic updates to security policies for achieving secure communications. This is particularly important with emerging new security threats and the need to dynamically update security policies based on the distributed network state and detection of new anomaly events. Such an integrated security approach involving authentication of IoT devices and enabling authorized flows can help to protect IoT networks from malicious IoT devices and attacks such as IoT botnet based DDoS attacks. We have implemented and validated our architecture using ONOS SDN Controller and Raspbian Virtual Machines, and demonstrated how the proposed security mechanisms can counteract malware packet injection, DDoS attacks using Mirai, spoofing/masquerading and Man-in-The-Middle attacks.

The authors would like to thank ISIF ASIA for their financial contribution to the Project.

CONTENTS

1. INTRODUCTION

The report presents project summary on the “Software Defined Networks based Security Architecture for IoT Infrastructures” project funded by the ISIF group. There are three milestones for the project with specific deliverables for each milestone.

As part of first milestone, we presented an overview of the IoT Architecture with discussion on the different layers of the architecture and protocols that are used at different layers of the IoT Architecture. Then we presented threat model for the IoT Architecture with discussion on the several attacks that are possible at different layers of the IoT Architecture. We also proposed the design and development of feature distributed malware attacks for Internet of Things in the context of smart homes. Smart homes consist of various smart home devices, providing convenient services to the tenants. These devices are usually connected to the Internet, and the tenants can integrate individual devices and make smart home workflows using Internet services such as IFTTT. The focus of our work was on how the attackers can generate various cyber-physical and advanced cyber-attacks by exploiting this integration aspect in smart home. In particular, we have designed and developed an advanced feature distributed malware that can be used to perform various malicious activities such as stealing the victim’s IFTTT cookies, using smart devices as information sources of the malware and distributing malware functionalities to other devices, performing malicious without being noticed, causing financial damages to the victim, and evasively exfiltrating data. Our experiments show that the proposed attacks enable the attackers to control any smart home device integrated with IFTTT, without requiring compromise of individual smart home devices. We believe our approach of using the integrating Internet service as an attack vector and distributing malware features to the smart home devices will be helpful to the development of more secure smart home environments and different IoT applications. Finally, we have analysed some of the previously proposed techniques to deal with the attacks in IoT applications and identified few security requirements that need to be considered for designing and for developing security architecture for IoT applications.

As part of second milestone, we developed a lightweight mutual authentication protocol based on a novel public key encryption scheme for smart city applications. The proposed protocol takes a balance between the efficiency and communication cost without sacrificing the security. We evaluated the performance of our protocol in software and hardware environments. On the same security level, our protocol performance is significantly better than existing RSA and ECC based protocols. We also provided security analysis of the proposed encryption scheme and the mutual authentication protocol. The proposed protocol is an n-pass lightweight mutual authentication protocol. The value of n is related to the desired security level of the protocol and the system parameters of the encryption scheme. The lightweight mutual authentication protocol applies the proposed encryption scheme as a building block. The security of the proposed n-pass mutual authentication is guaranteed by the security of the Needham-Schroeder protocol. We have shown that our protocol can resists attacks such as man-in-the-middle attack and impersonation attack. We evaluated the protocol on Contiki OS and CC2538 evaluation modules. The experimental evaluations show that our protocol is respectively 88 and 7 times faster than RSA and ECC on the security level of 112 bits. The mutual authentication time can be further reduced if online/offline technique is enabled.

In this report we present a summary of the outcome of third milestone ”Security Architecture for IoT” which consists of the following tasks: i) Report on the Design of Security Architecture and Attack Detection in IoT Infrastructures and ii) Proof of Concept of Security Architecture for IoT.

During this period, we have designed and developed security architecture which makes use of Software Defined Networks (SDN) to securely manage the IoT infrastructures. SDN decouples the control plane from the data plane, and provides a centralized authority (SDN Controller) to manage the resources (such as network and IoT devices) within the domain under its control. As the SDN Controller has visibility over its network domain, the applications running in the Controller have the ability to manage the security of the underlying IoT network infrastructure. The use of SDN to manage the IoT devices is not in itself new; other works [1, 2] have used such an approach to detect malicious devices and detect attacks. In [3] authors have presented a comprehensive survey of how SDN can be used to secure the IoT devices and the IoT network infrastructure. However, in our architecture, SDN Controller acts as a security policy decision authority, and the network switches and IoT gateways enforce the security policies in the IoT network infrastructure. Such a policy driven approach provides the capability to achieve secure management of network flows in an IoT infrastructure in a dynamic manner, and deal with security attacks in a proactive manner.

The specific contributions for third milestone can be summarised as follows:

- A SDN based security architecture that uses a policy based approach to secure IoT network infrastructure and detect malicious IoT devices and attacks in intra domains.
- Authentication of IoT devices using a light-weight protocol, which is in turn used in the provisioning of network services to authenticated IoT devices.
- Secure access to network services by authenticated devices using OAuth protocol.
- Extension of SDN based security architecture to secure IoT network infrastructure and detect malicious IoT devices and attacks in inter domains.
- Demonstration of the proposed security architecture and protocols using a realistic IoT scenario, showing how it can protect IoT infrastructure from attacks such as Malware Injection, DDoS, Spoofing/Masquerading and Man-in-The-Middle (MiTM) attack.
- Performance analysis of the proposed solution for securing IoT infrastructure.

The report is structured as follows. Chapter 2 describes the SDN based security architecture for IoT infrastructure. Chapter 3 demonstrates the application of the proposed security architecture using different IoT scenarios. In particular, we discuss IoT botnet attack scenarios and how they are successfully counteracted using our security architecture. Relevant related works are discussed in Chapter 4. Finally, Chapter 5 concludes the report.

2. SDN BASED SECURITY ARCHITECTURE FOR IOT NETWORK INFRASTRUCTURE

We use Software Defined Networks (SDN) to develop our security architecture for IoT network infrastructure. We first describe the rationale behind the choice of SDN as the underlying technology to secure IoT network infrastructure. Then we describe our security architecture for IoT network.

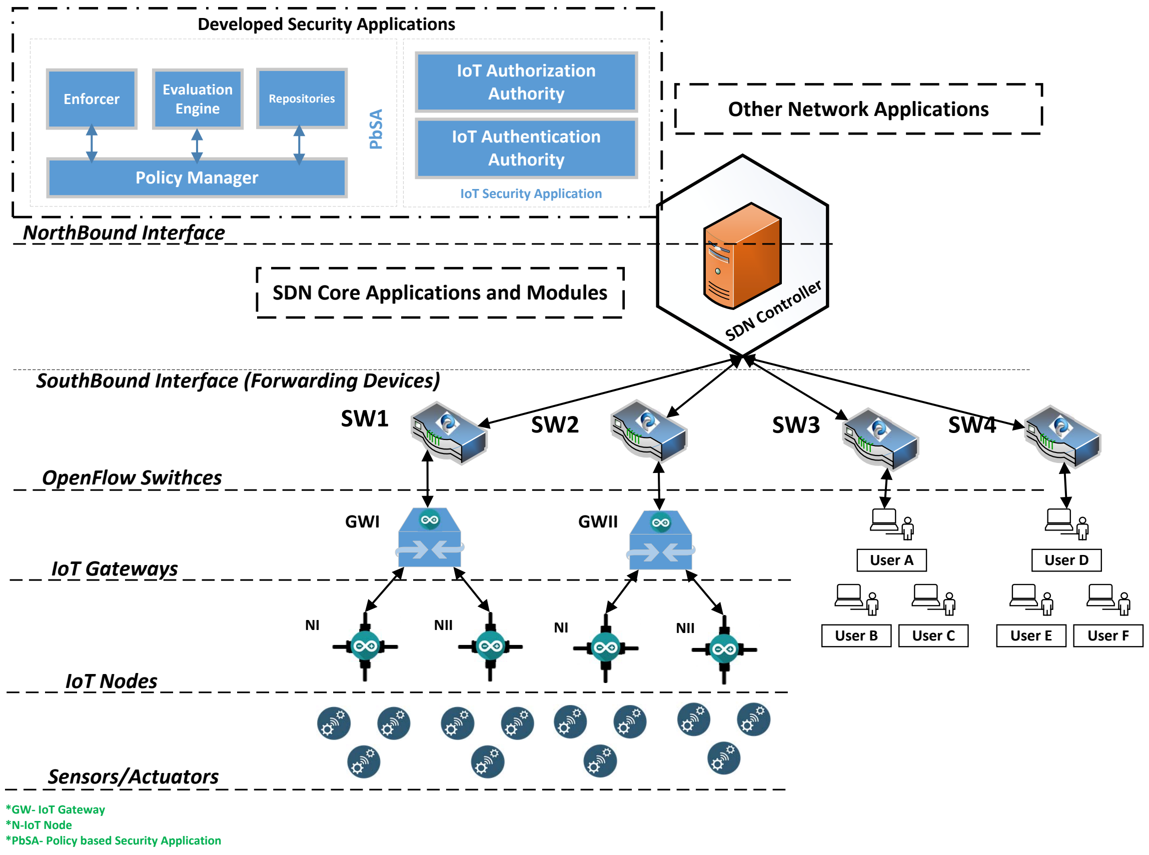


Figure 2.1: Security Architecture for IoT Network Infrastructure

2.1 Why SDN for IoT Security?

The features of SDN technology which make it a suitable platform for securing IoT infrastructure are as follows:

Separation of Control Plane from Data Plane: This is useful for designing our policy based security architecture at the SDN Controller in the control plane and enforcing the security policies in the network switches and IoT devices in the data plane. The SDN Controller communicates with the switches in the data plane using open and standardised interfaces and protocols (OpenFlow), which is useful for securing the communications between the policy decision authority in the Controller and the enforcement mechanisms in the IoT devices and switches.

Network Domain View: The SDN Controller has visibility over the whole network domain under its jurisdiction. This can be used by our architecture to achieve secure management of IoT devices and flows in the network infrastructure. The Controller maintains a topological information database which logs information about all the forwarding devices connected to the Controller. This will be useful in the specification of path based security policies in our architecture.

SDN Northbound Applications: The SDN provides flexible northbound API which enables us to develop secure applications or use third-party applications to securely monitor and control the behaviour of IoT devices and network nodes in the SDN network domain. Our security architecture has a secure application running on top of the Controller (developed using its northbound API) that provides security services in the IoT infrastructure.

2.2 Security Architecture for IoT Network Infrastructure

Our proposed SDN based security architecture uses policies to control and manage IoT devices, services and network entities (switches, nodes and gateways). Figure 2.1 shows the proposed SDN based security architecture. The heart of the security architecture is the SDN Controller where security policies reside and are evaluated. The IoT actuators and sensors are the end devices and connect to the IoT Nodes. These IoT Nodes are connected to the IoT Gateways via either via wired or wireless networks. The IoT Gateways are connected to the SDN Controller. In some cases, OpenFlow switches can act as IoT Gateways/Nodes. Our implementation considers OpenFlow switches as IoT Gateways. Users are connected to the OpenFlow switches.

The IoT devices are of a heterogeneous nature. The IoT devices can use different network protocols, authentication mechanism, can have different operation and application platforms. Furthermore, there can be a large number of IoT devices. Hence, a scalable solution is needed, recognizing the individual capabilities of the connected IoT devices. In our security architecture, we have created a device provisioning mechanism for each IoT device groups using a template based approach. Each template consists of variables explaining device capabilities like protocol, manufacturer, authentication mechanisms and other features. This device provisioning functionality is integrated with the core application of the Controller, enabling it to provision the IoT devices at runtime.

In our implementation, we have developed two secure applications that control and manage the behaviour of IoT devices in the network. These applications runs over an SDN Controller. The first one is the *Policy based Security Application (PbSA)*, and the second one is called *IoT Security Applications (ISA)*. *ISA* has two sub-modules, namely *IoT Authentication Authority* and *IoT Authorization Authority*. Before describing in detail the functions of each of these applications and modules, we first provide a high-level overview of the security interactions with the IoT devices in the network infrastructure.

The operation of our security architecture can be viewed as involving two phases. In the first phase, we have interactions that are associated with authentication. New IoT devices that are to be connected to the network need to be authenticated. IoT Gateways forward IoT device information (such as device ID) using *Packet_in* messages to the SDN Controller. The *IoT Authentication Authority* module in the ISA application carries out the authentication process using a lightweight elliptic curve cryptography (ECC) based authentication protocol. This process of IoT authentication will in turn make the network domain and services visible to authenticated IoT Devices.

In the second phase, the PbSA checks the network service request from authenticated IoT devices against the specified security policies. The PbSA has fine grained policies that control the behaviour of the IoT devices; for instance, a user "A" accessing via OpenFlow Switch 3 (*SW3*) is allowed to read data from Sensor 2 connected through *Gateway I (SW1 & Node NII)*. We describe the security policies and their implementation in detail in Section 2.3. If the service request is permissible according to *PbSA* security policies, then our security architecture uses the OAuth protocol to provide a token to the IoT device. The IoT device uses this token for further communication in the network. This process is jointly performed by *PbSA* and *IoT Authorization Authority*.

2.3 Policy Based Security Architecture for Intra Domain

Our IoT network infrastructure consists of OpenFlow switches/IoT Gateways and end hosts (IoT sensors/actuators). We have used a single SDN Controller to manage the IoT network infrastructure. The network devices (OF devices) forward the packets generated by the IoT devices/users, which are then subjected to the policies specified in the SDN Controller for transfer in the network. Figure 2.1 shows the Policy-based Security Architecture (PbSA) for securing the IoT Infrastructure using SDN. As *PbSA* is designed to be modular, the components of *PbSA* can be implemented on a single host or can be distributed over multiple hosts. Here, we provide a detailed description of different modules of *PbSA*.

PbSA consists of four major modules, namely a) Policy Manager, b) Evaluation Engine, c) Repositories & d) Policy Enforcer.

- **Policy Manager:** Is the core component of the security architecture, as it manages every operation such as extracts IoT device flow attributes, updates the Topology Repository, and instructs the Policy Enforcer to enforce the policies at the OpenFlow IoT Gateways. It also communicates with the *ISA* application for the transfer of OAuth token after checking the network service request from the IoT devices.
- **Evaluation Engine:** Evaluates the service request against the relevant policies stored in the Policy Repository.
- **Repositories:** Our architecture has two repositories: a) Topology Repository and b) Policy Repository. The Topology Repository contains the network topology of IoT devices and end

hosts/users. SDN controller has its own device Topology Repository. We are using the same Topology Repository for this purpose. The Policy Repository contains the Policy Expressions (PE) associated with the various IoT devices and the associated flow attributes. The attributes in PEs also include security parameters such as security labels associated with the OpenFlow IoT Gateways.

- **Policy Enforcer:** Fetches the required information from the OpenFlow IoT Gateways and enforces the flow rules obtained from the Policy Manager.

Security Policy Specifications

The security policy specifications are expressed as Policy Expressions (PE), which specify whether packets and flows from IoT devices and end hosts follow a particular path or paths in the network, and the conditions under which the packets and flows follow these paths. The PE specification syntax uses an enhanced version of RFC1102 [4]. They are fine-grained and specifies a range of policies using various attributes of IoT devices and flows; for instance, these attributes include different types of devices, source and destination attributes, flow attributes and constraints, requested services, security services and security labels. The attributes are: **(a) Flow Attributes:** Flow ID, sequence of packets associated with the Flow, type of packets, Security Profile indicating the set of security services that are to be associated with the packets in the Flow; **(b) Device Attributes:** ID specific to IoT sensor/actuator; **(c) Switch Attributes:** Identities of the Switches and Security Label of the Switches (In our case, these are OpenFlow IoT Gateways); **(d) Host Attributes:** Identities of Hosts such as Source/Destination Host ID; **(e) Fow and Domain Constraints:** Constraints such as Flow Constraints (FlowCons) and Domain Constraints (DomCons) associated with a specific device Flow; **(f) Services:** For which the PE applies (e.g. FTP storage access); **(g) Time Validity:** The period for which the PE remains valid; and **(h) Path:** Indicates a specific sequence of switches any particular flow from specific IoT devices/users should traverse.

The Constraints (Flow and Domain) are conditions that apply to specific flows from any IoT devices. For instance, a constraint might specify the flow from a specific type of sensor, should only go through a set of switches that can provide a guaranteed bandwidth. From a security point of view, a constraint could be that a flow should only go through OpenFlow switches that

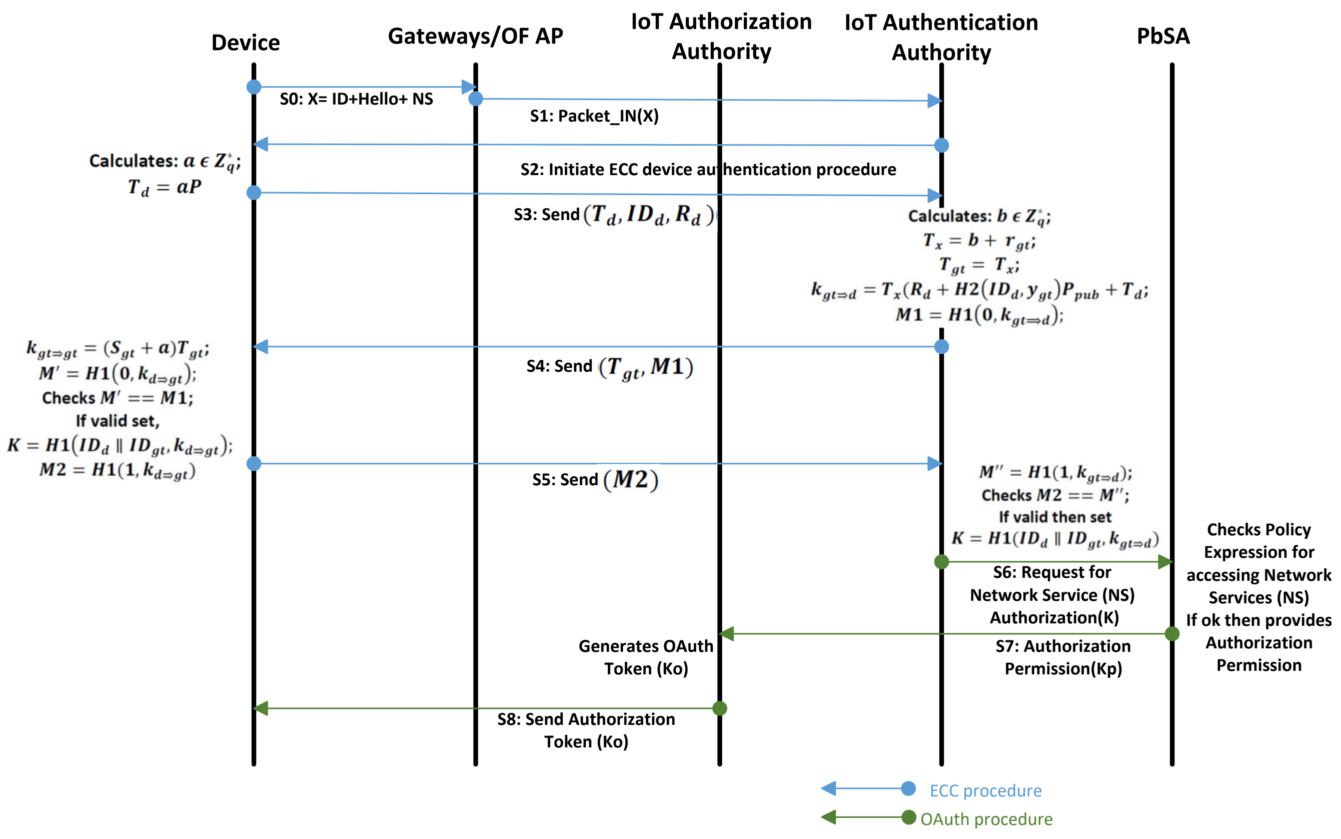

Device
Gateways/OF AP
IoT Authorization Authority
IoT Authentication Authority
PbSA
S0: X= ID+Hello+ NS
S1: Packet_IN(X)
S2: Initiate ECC device authentication procedure
Calculates: $a \in Z_q^*$; $T_d = aP$
S3: Send (T_d, ID_d, R_d)
Calculates: $b \in Z_q^*$; $T_x = b + r_{gt}$; $T_{gt} = T_x$; $k_{gt\Rightarrow d} = T_x(R_d + H2(ID_d, y_{gt})P_{pub} + T_d$; $M1 = H1(0, k_{gt\Rightarrow d})$;
S4: Send $(T_{gt}, M1)$
$k_{gt\Rightarrow gt} = (S_{gt} + a)T_{gt}$; $M' = H1(0, k_{d\Rightarrow gt})$; Checks $M' == M1$; If valid set, $K = H1(ID_d \parallel ID_{gt}, k_{d\Rightarrow gt})$; $M2 = H1(1, k_{d\Rightarrow gt})$
S5: Send $(M2)$
$M'' = H1(1, k_{gt\Rightarrow d})$; Checks $M2 == M''$; If valid then set $K = H1(ID_d \parallel ID_{gt}, k_{gt\Rightarrow d})$
S6: Request for Network Service (NS) Authorization(K)
Checks Policy Expression for accessing Network Services (NS) If ok then provides Authorization Permission
S7: Authorization Permission(Kp)
Generates OAuth Token (Ko)
S8: Send Authorization Token (Ko)
ECC procedure
OAuth procedure

Figure 2.2: Security Protocol process diagram

have a particular Security Label. The PEs support wildcards for attributes, enabling it to set policies for a group of IoTs/services. A simplified Policy Expression template is as follows:

PE_i= $< FlowID, IoTDeviceID, SourceAS, DestAS, SourceHostIP, DestHostIP,$

$SourceMAC, DestMAC, User, FlowCons, DomCons, Services, Sec-Profile,$

$Seq-Path >:< Actions >$

where i is the Policy Expression number.

2.3.1 IoT Security Application (ISA)

In this section, we briefly explain the security protocols used in our security architecture. These are implemented by the *IoT Security Application (ISA)*.

In our architecture, each IoT device has a unique ID and connects to IoT Gateways using wireless networks. The IoT Gateways are basically OpenFlow Switches/Access Points (OF-AP) which support OpenFlow protocol. As mentioned earlier, our security protocol has two phases, namely the authentication phase followed by the policy application and the OAuth protocol phase.

IoT devices are authenticated in the first phase of the security protocol. We have used a lightweight ECC protocol [5] to authenticate the IoT devices. An IoT device sends a “Hello” message with its ID (Message X in Figure 2.2) requesting for a Network Service (NS). The OF-APs sends the ID and Network Service (NS) request via $Packet\_in$ messages to the *IoT Authentication Authority*. *IoT Authentication Authority* performs ECC authentication of the IoT devices. At the end of the authentication phase, *IoT Authentication Authority* authenticates the IoT device and establishes a common secret key with the IoT device that can be used for further secure communications.

In the authorization phase, the *PbSA* checks the Network Service (NS) request from the IoT device using the security policy specifications described above. The *IoT Authentication Authority* uses the key established by the ECC protocol in the authorization request to *PbSA*. If the device and flow attributes for the Network Service (NS) satisfy the Policy Expressions, then the PbSA requests the *IoT Authorization Authority* to generate the OAuth Token for the particular IoT device for the specific Network Service(s) requested. The token is then used by the authenticated IoT device to access the required Network Service(s). This process is illustrated in Figure 2.2.

Lightweight ECC based Authentication for IoT Devices

The Elliptic Curve Cryptography (ECC) based public key system uses the algebraic structure of elliptic curves as their finite points. It is computationally faster than other public key cryptosystems such as RSA. Hence it is more suitable for computationally constrained IoT devices. It can be used to achieve key agreement as well as encryption and digital signature. In our security architecture, the IoT devices are authenticated using a lightweight ECC based authentication protocol proposed in [5]. Then we use the key established using this protocol to encrypt the data from the authenticated IoT devices. This lightweight security protocol works in conjunction with the OAuth protocol.

The ECC based authentication protocol used in our architecture has 3 stages: Setup, Installation and Key Agreement. Algorithm 1 below gives an outline of the 3 stages. The Key Agreement (Stage 3) is illustrated in Figure 2.3. The performance of the proposed protocol (in terms of both computation and communication costs) is higher than the previously proposed protocols.

Device		IoT Authentication Authority
Calculates: $a \in Z_q^*$;		
$T_d = aP$	$(T_d, ID_d, R_d)\rightarrow$	Calculates: $b \in Z_q^*$;
		$T_x = b + r_{gt}$;
		$T_{gt} = T_x$;
$k_{gt\Rightarrow gt} = (S_{gt} + a)T_{gt}$;	$\leftarrow(T_{gt}, M1)$	$k_{gt\Rightarrow d} = T_x(R_d + H2(ID_d, y_{gt})P_{pub} + T_d$;
$M' = H1(0, k_{d\Rightarrow gt})$;		$M1 = H1(0, k_{gt\Rightarrow d})$;
Checks $M' == M1$;		
If valid set,		
$K = H1(ID_d \parallel ID_{gt}, k_{d\Rightarrow gt})$;		
$M2 = H1(1, k_{d\Rightarrow gt})$	$(M2)\rightarrow$	$M'' = H1(1, k_{gt\Rightarrow d})$;
		Checks $M2 == M''$;
		If valid then set
		$K = H1(ID_d \parallel ID_{gt}, k_{gt\Rightarrow d})$

Figure 2.3: Key agreements between Devices and *IoT Authentication Authority*

Hence it is particularly suitable for the IoT environment as it shifts the processing load from the resource-limited devices to the more powerful servers in the Controller.

Algorithm 1: ECC Protocol

```
1 Step1 (Setup): a) The IoT Authentication Authority chooses a prime number $q$,
    computes $F_q, E/F_q, G_q, P$;
  b) The IoT Authentication Authority chooses a master key $x$ and calculates public key
    $P_{pub} = xP \in E/F_q$;
  c) It then computes two hashes $H1$ and $H2$;
  d) $F_q, E/F_q, G_q, P, P_{pub}, H1, H2$ are made public.;
2 Step2 (Installation): IoT Authentication Authority and devices separately generate and
    validate their private and public keys.;
  $R_{gt} = r_{gt}P$, where $r_{gt}$ is a random number $r_{gt} \in Z_q^*$.
  $y_{gt} = H_1(ID_{gt}, R_{gt}).x$
  $R_d = r_dP$, where $r$ is a random number $r_i \in Z_q^*$.
  $y_d = H_2(ID_d, y_{gt}).x$
  $S_d = r_d + y_d$
  $S_d$ and $R_d$ are the private and public values of $d$.
  At the end of this stage each device has its own $S_d, R_d, y_d, r_d$ and IoT Authentication
    Authority has $(y_{gt}, r_{gt})$;
3 Step3 (Key Agreement): Between IoT Authentication Authority and IoT Devices;
```

In [5], the authors analyze the security of the ECC authentication scheme and show that it is secure against active attackers who are capable of eavesdropping, modifying and injecting messages in the protocol.

Authorization for Network Services using OAuth Protocol

The authorization service using the OAuth protocol [6, 7] works as follows: It consists of four actors: i) Client, ii) Resource Owner, iii) Resource Server and iv) Authorization Server. The client contacts the Resource Owner of the resource. The Resource Owner grants the access to the client by sending an authorization code. The client delivers the received authorization code to the Authorization Server. The Authorization Server verifies the authorization code and releases a token containing the details of the consent provided to the client (time limit, scope, and so on). The client forwards the token to the Resource Server. The Resource Server checks the validity of the received token, and in the affirmative case provides access to the protected resource.

In our SDN-IoT architecture, network services are the resources. Figure 2.2 shows the OAuth Token handover process (marked with a continuous green line). First, the *IoT Authentication Authority* authenticates the IoT devices using ECC protocol mentioned above. After the authentication phase, it requests access to network services on behalf of the authenticated IoT devices. The PbSA checks the policy repository and if the network service request is valid, then it issues an *Authorization Permission (Kp)*, which is forwarded to the *IoT Authorization Authority*. The *IoT Authorization Authority* generates the *OAuth Token (Ko)*. We have used JSON Web Token (JWT) for this purpose. The *IoT Authorization Authority* forwards the OAuth Token to the IoT devices. The *OAuth Token (Ko)* contains the user ID, IoT device ID, device type (e.g. sensor or actuator) and expiry time (network service access time). The IoT device integrates this OAuth Token in their network packet while accessing the network services.

In our current implementation, we have signed the OAuth Token using the private key of *IoT Authentication Authority* S_gt, which was used in the ECC authentication phase. The signature is verified by the network service using the public key of *IoT Authentication Authority* R_gt before service provision. (In a more general distributed architecture where *IoT Authorization Authority* and *IoT Authentication Authority* are not co-located, the ECC authentication protocol will be used to generate public and private values for *IoT Authorization Authority* using a similar process as for an IoT device and this private key will be used to sign the OAuth Token).

2.4 Policy Based Security Architecture for Inter Domains

In this Section, we will describe how the Policy Based Security Architecture can be extended to support secure end to end communication between IoT devices in multiple AS domains.

We assume that each AS domain is controlled by a SDN Controller. Each Controller runs a separate instance of the PbSA and ISA application. We have added a new module to handle the AS domain communication. It is known as Packet Handle Creator. A Packet Handle Creator module creates the necessary handles from the visited Controller which is piggy backed with the payload from the Policy Manager. These handles are used to check the authenticity of the packet and the enforcement of policies at the switches.

With intra-domain communications, the traffic from source to destination passes through devices within a single SDN domain and the requested services are provided by the servers and devices in the same domain. In this case, the traffic and service requests are subjected to security policies in the PbSA in the SDN Controller of that domain. The routing process begins from the host that is generating the packets and the request, which is the source of the communication. This source host could be any client, such as a mobile device. The initial packet header from the source host is sent by the switch (to which this host is connected) to the SDN Controller in the AS domain. The header contains all the usual network and service parameters such as the source address, the packet type. The PbSA application in the Controller extracts the relevant parameters from the incoming packets and uses the Policy Repository and the Policy Manager to determine whether the relevant Policy Expressions are satisfied. If the Policy Expressions are valid for the incoming packets, then PbSA will enforce the specified actions as flow rules in the appropriate data plane devices such as switches to transfer the packets.

Inter-domain communications for IoT devices involving multiple domains require cooperation between SDN Controllers, as the communications are subject to security policies of multiple Controllers. Hence inter-domain routing of traffic requires an SDN Controller in one AS domain to have knowledge of other Controllers in other AS domains. To create a topological map of a distributed SDN environment, we have used the traceroute mechanism. Note that although there are other alternatives [8] for topology discovery such as Border Gateway Protocol (BGP) and Internet Routing Registries, each one of them has its own issues. For the purpose of our prototype, traceroute was found to be sufficient and the easier one to use. It is not a critical part of our design and it is mainly an implementation issue. From the architecture point of view, each Controller has a Topology Repository to store the mapping of the topology information.

SDN Controllers in each AS keep this Repository updated by running traceroute at different times. With traceroute, we have included an additional security attribute, a Security Label, in the ICMP response message from each AS SDN Controller. The intention of the Security Label is to reflect the level of security associated with that particular Controller. In our current architecture, this Security Label is hardwired (static) and is specified at the time of installation of the Controller based on the reputation of the manufacturer of the Controller. In the next stage of the development of the architecture, we will develop a meta-level security protocol that will enable secure and dynamic updating of the Security Label depending on the behaviour of the Controller over time. This will be done as part of a trust model which we are in the process of developing for distributed SDN environment. We have modified the ICMP response messages to attach the Security Labels. Hence each AS domain SDN Controller now has the ability to discover the topological information as well as the levels of security associated with all the neighboring AS domain Controllers in this Repository.

Consider the distributed SDN environment shown in Figure 2.4 and the associated Topology Repository tables are shown in the figure too. Each hexagon in Figure 2.4 represents an AS domain. We have represented the AS Gateways using Gateway OpenFlow Switches. Each Table in Figure 2.4 shows the Topological Repository for the respective domains SDN controller. *AS_ID* is the identity of a particular AS, *Sec_Label* is the security label of the AS domain and Hops is the distance from source to destination AS. The edge OpenFlow Switches are represented using the notation ([source AS ID]SW[destination AS ID]), e. g. switch connecting AS1 to AS2 is represented as $1SW2$.

In the inter-domain setting, our architecture introduces two additional mechanisms which have conceptual significance.

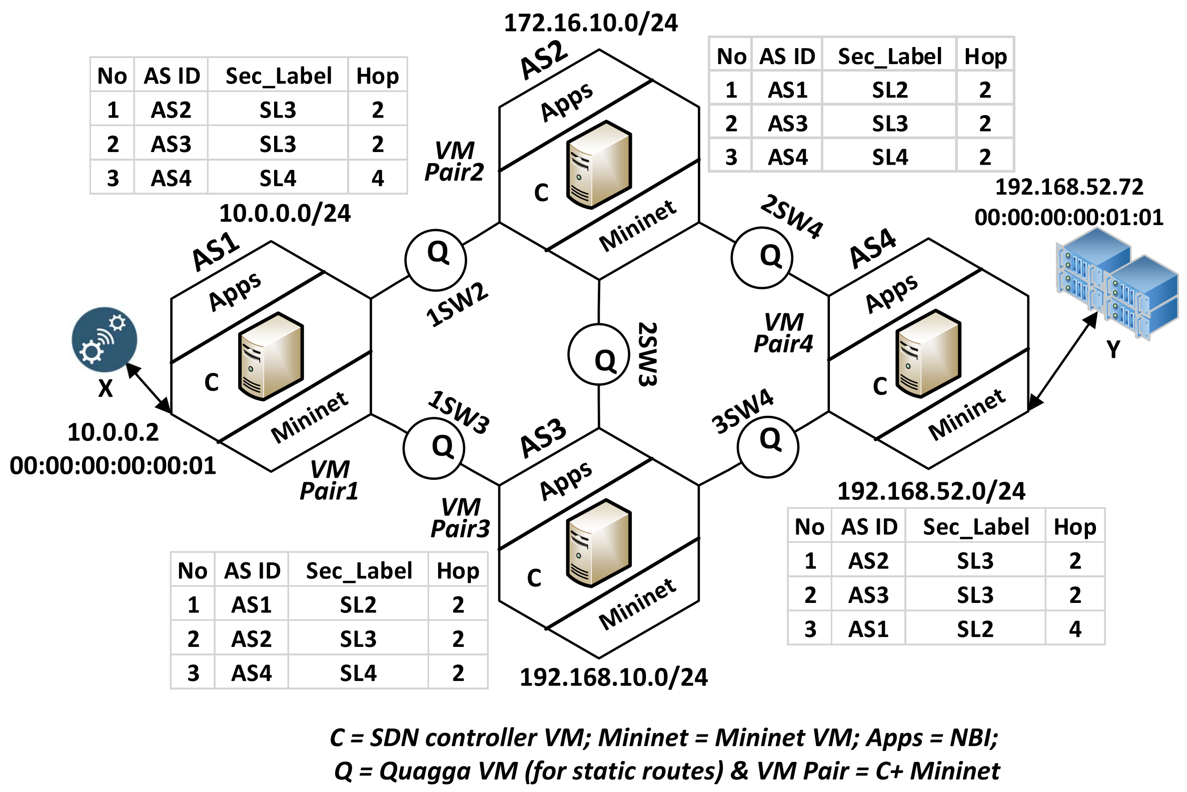
Handle: The first mechanism is a *Handle*. PbSA creates a *Handle* and tags to each IoT device flow request. The Handle consists of a list of visited AS domain IDs. The *Packet* + *Handle* is then transferred to the next AS Domain Controller. A similar process is repeated as the packet goes through all the transit AS domain SDN Controllers until the packet reaches its destination. This Handle will be protected for integrity and it will be used in the validation of flows across multiple domains.

Policy Transfer Token The second mechanism is a *Policy Transfer Token*, which comprises policy constraints that are transferred from one AS domain to the subsequent transit AS domains and which need to be satisfied by the flow, as the packets are transferred. These constraints need to be taken into account in addition to the policy constraints of the transit domains. For instance, if there is a constraint that an IoT devices traffic should only pass through AS domains with security label greater than a certain threshold, then this constraint needs to be satisfied by subsequent transit domains. Suppose an AS domain SDN Controller (with AS ID = 10) has a constraint that packets should only be forwarded through a path of AS domain SDN Controllers that have a security label greater than $SL3$. In distributed systems, in general, it is not possible for one domain Controller to know about policies of other domains. Hence there is a need to transfer the policy constraints, which are communicated via *Policy Transfer Tokens*. The significance of the transfer token is that the policy constraints that are transferred are only those policies that are specific to flows and packets in that flow. Such a mechanism is useful as it enables partial delegation of policies that are flow dependent. The next section describes in detail the specification of security policies. In this project, we assume that the AS domain Controllers are secure and trusted, and that if and only if the policies can be satisfied in the domain, the receiving Controller will accept the packets.

In terms of the notation, we denote the Handle as $H_i^{AS_k}$ which is tagged to the packet (flow request), where H_i is the Handle for a particular communication *i*, and *k* is the ID of the AS domain SDN Controller which created the Handle. Similarly, the *Policy Transfer Token* is denoted as $PTT_i^{AS_k}$, where again i denotes the specific communication and k denotes the ID of the AS domain. Hence an AS domain SDN Controller creates the augmented Packet using the original Packet as well as the Handle and the *Policy Transfer Token*.

2.4.1 Security Architecture Walk-through

Let us now give a brief walk-through of the operation of the proposed security architecture described above. We will use the inter-domain scenario given in Figure 2.4 to illustrate the various steps involved. The Tables beside each of the AS domains in Figure 2.4 represent the Topology Repository. Table 2.1 shows the policies in each of these AS domains stored in the Policy Repositories of PbSA. Here, X is a IoT device and its logging information in a database



No	AS ID	Sec_Label	Hop
1	AS2	SL3	2
2	AS3	SL3	2
3	AS4	SL4	4

No	AS ID	Sec_Label	Hop
1	AS1	SL2	2
2	AS3	SL3	2
3	AS4	SL4	2

No	AS ID	Sec_Label	Hop
1	AS1	SL2	2
2	AS2	SL3	2
3	AS4	SL4	2

No	AS ID	Sec_Label	Hop
1	AS2	SL3	2
2	AS3	SL3	2
3	AS1	SL2	4

Figure 2.4: Implemented Network with Routing Information.

server in Y.

Y is a trusted database server sitting in AS4. However, the IoT device X is a new devices. Hence, it needs to go through the authentication phase first. AS1 local gateways connecting the IoT devices passes the necessary information to the AS1 SDN Controller via the *packet˙IN* message. ISA application using a light weight ECC protocol authenticates the new IoT devices. Next, PbSA checks policy repository for a valid policy expression. AS1 contains a valid policy expression for devices in the IP range 10.0.0.0/24. Thus, the security architecture provides an OAuth token to the IoT device. The IoT device uses this for future communication. Now, we will describe how, the devices upload the sensor data to the database server in a different domain.

The initial packet header from the source device is sent by the switch (to which this host is connected) to the SDN Controller in that AS domain. The PbSA application in the Controller extracts the relevant parameters from the incoming packets and uses the Policy Repository and the Policy Manager to determine whether the relevant Policy Expressions are satisfied. If the Policy Expressions are valid for the incoming packets, then PbSA will enforce the specified actions as flow rules in the appropriate data plane devices such as switches to transfer the

Table 2.1: Stored Policy Terms

AS ID	Policy Expression
AS1	$PE_1^{AS1} =< *, (10.0.0.0/24, EDU, SL2), (192.168.52.0/24), 10.0.0.2, *, *, *, *, *, SL2+ =, (80, 443), conf, * >:< (1SW2, Allow) >$
AS2	$PE_4^{AS2} =< *, (10.0.0.0/24, EDU, SL2), (192.168.52.0/24), 10.0.0.2, *, *, *, *, *, SL2+ =, (80, 443), conf, (AS1) >:< Allow >$
AS3	$PE_2^{AS3} =< *, (10.0.0.0/25, EDU, SL2), (192.168.52.0/24), *, *, *, *, *, *, SL2+ =, (80, 443), conf, (AS1, AS2) >:< Allow >$
AS4	$PE_2^{AS4} =< *, (10.0.0.0/25, EDU, SL2), (192.168.52.0/24), 10.0.0.2, 192.168.52.72, *, *, *, *, *, (80, 443), conf, (AS1, AS2, AS3) >:< Allow >$

packets.

In this scenario, the IoT device X (with IP address $10.0.0.2$) wishes to upload the sensor data to the database server Y (with IP address $192.168.52.72$). As X and Y reside in two different AS domains, communication between them occurs via transit AS domains. After the IoT device authentication and network authorization phase, packets from X go to the SDN Controller of AS1. As the Policy Expression PE_1^{AS1} in AS1 matches with this particular network traffic, HTTP & HTTPS traffic originating from $10.0.0.2$ are allowed to go to Y in the subnet $192.168.52.0/24$. However let us assume that there is a flow constraint which specifies communications between X and Y must occur only through domains which have security levels greater than or equal to $SL2$. This will result in the traffic routed through the AS2 domain via the OpenFlow Switch *1SW2* (Connected to AS1). A similar process occurs in AS2 and the traffic is sent to Y in AS4.

We also have constraints such as a flow or flows between two entities in a domain A and B should only go through paths whose security labels are greater than or equal to a specific security label L. Satisfying this rule requires the PbSA to determine the labels of the various paths between the devices, and then check whether the flow constraint is satisfied. The policy rules can have Boolean expressions such as the security label of the switch S_i should be less than or equal to the security label of the neighbouring switch S_{i+1}. If the neighbouring switch meets this constraint, then the flow will be directed through it; if not, than another neighbouring switch will be selected. In general, flow and domain constraints can require some form of evaluation which is more than just policy matching.

2.5 User Attributes

Our Security Architecture can readily support enforce policies with specific rules reflecting social aspects such as devices belonging to certain groups of users such as ethnicity or gender as well as constraints related to certain cultures such as different cultural organizations and their attributes. For instance the security architecture can be used to store all the devices belonging to specific users and this information can be used to enforce different types of policies based on the user gender and culture. For instance, the security architecture can be used to enable secure communication between the same gender or people with specific health issue (that are using similar body sensors) or advertise cultural/religious events based on the location/religion. Such features can address various characteristics associated with social norms and ethics associated with the data collected from these devices as well as expected interactions between various devices in different domains. These can be achieved due to fine granular nature of the policies and its ability to reflect the context and their associated characteristics.

3. IMPLEMENTATION

We have created a simulation network with ONOS (SDN Controller), Oracle VM Box, GNS3 and mininet-wifi [9] using a workstation (Core i7 - 7700K @ 4.20 GHz CPU; 64 GB of RAM). Our network configuration is shown in Figure 3.1. The Rasbian VMs are used to simulate IoT devices. We have developed two ONOS applications, namely *Policy based Security Application (PbSA)* and *IoT Security Application (ISA)*.We have used JSON database to create our Policy Repository.

3.1 Threat Mitigation

We have provided a detailed analysis of the IoT attacks as part of first milestone report. In this reports, we will focus on specific attacks that can be mitigated with our security architecture. First we will explain the attack in brief and then explain how our security architecture mitigates the attack.

3.1.1 Threats:

Threat 1 (Malware injection) : The IoT devices firmware often have low strength security features burned in their hardware by the device manufacturers. Hence they are vulnerable to different types of network compromises and packet injection attacks. For instance, an IoT device with open Telnet port allowing a malicious adversary to inject malware packets and compromise the device for further coordinated attacks. The execution of malware often uses the software/API in the Application Layer.

Threat 2 (DDoS) : In the IoT network, an adversary can use IoT devices to launch DDoS attacks to clog network services and specific IoT device services. Due to many IoT devices not

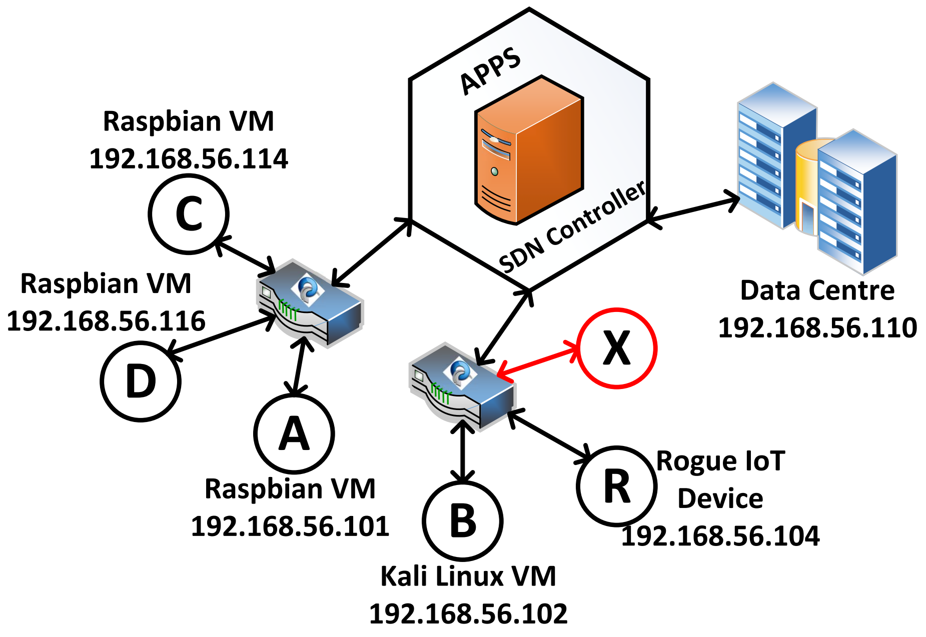


Figure 3.1: Network configuration

having any security functionality and due to their resource-constrained nature, they are prone to attacks leading to energy depletion. Then these compromised devices launch coordinated flooding attacks against the network services and specific users/devices. These attacks target the Network & Perception layer.

Mirai is a perfect example of Threat 1 & 2. The first phase of Mirai uses malware packet injection attack to compromise the IoT devices and create zombies/bots. In the second phase, zombies (compromised devices) use a coordinated packet flooding to clog the network services. We have provided a detailed discussion on Mirai in subsection 3.1.2.

Threat 3 (Spoofing/masquerading) : In the IoT infrastructure, a malicious adversary can try to impersonate another user/device. This attack can then be used to capture/modify the sensitive information from/in the devices as well as send malicious instructions to the actuators.

Threat 4 (MiTM attack) : In a MiTM attack, an adversary intercepts the communication between the two parties. The IoT infrastructure is also vulnerable to this type of attack. An adversary or a malicious IoT device can change the IoT device/user ARP caches of the two communicating parties to initiate a MiTM attack in the IoT infrastructure.

3.1.2 Mitigation:

We now present how our security architecture mitigates the above mentioned threats. **Threats 1 & 2:**

We will use Mirai as an example of **Threats 1 & 2**. In Mirai, an attacker first injects the malware into IoT devices and then launches a co-ordinated DDoS attack using these infected devices. In this section, first, we describe a Mirai based DDoS attack specific to IoT environment and then show how our security architecture helps to prevent this attack.

Mirai:

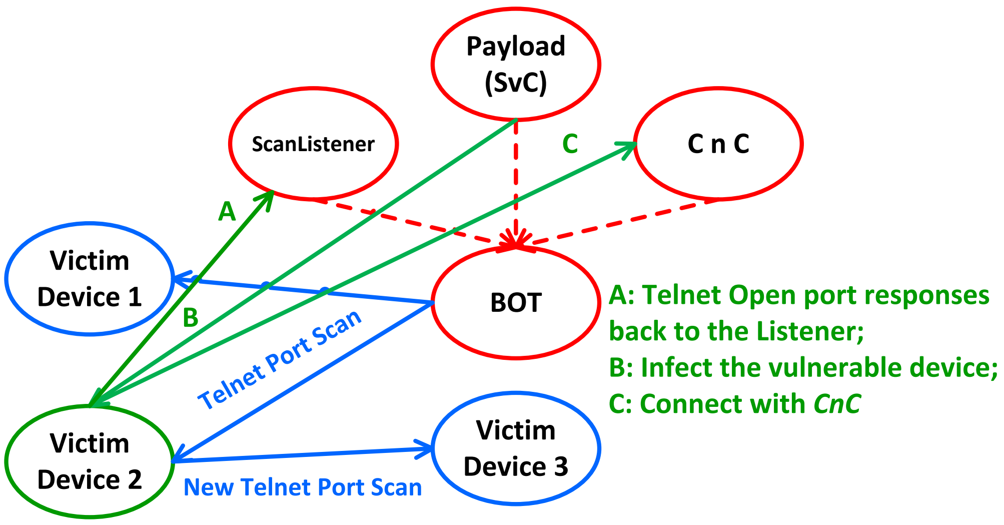
Each Mirai infected bot runs two parallel processes. One is a Command and Control (CnC) center and the other one is ScanListener server as shown in Figure 3.2a.

Once a device is compromised by Mirai code, it starts to scan other vulnerable hosts in the network. It scans for the open *telnet* ports with an active *telnet* server. This phase is the scanning phase. Once it finds a host, it carries out brute force attack on the *telnet* server to find passwords and corresponding user names. Mirai code has 62 default user name and password combinations (shown by green solid line in Figure 3.2a). The bot uses these pairs of user names and passwords to log on to the *telnet* server. After a successful login, it blocks the ports 22, 23 and 80. Thus, the users of this device will not be able to access it. Then the malicious payload is uploaded on to the captured device to turn it into a bot. If the vulnerable device is successfully converted to a bot, it connects to the CnC center of the bot. Then it can generate different types of DDoS attacks. The DDoS attacks that Mirai can generate includes UDP, SYN, ACK, TCP Stomp, DNS, HTTP, GRE IP and GRE Ethernet Flood.

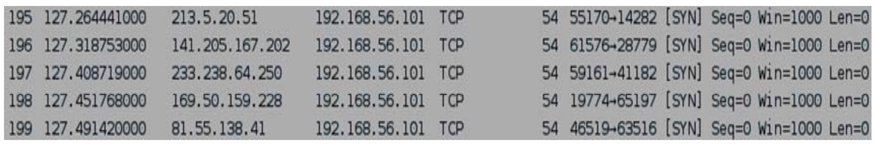
Mitigation:

We have experimented with all the above flooding attacks and found that our security architecture is able to counteract them successfully. In this report, we demonstrate the scenario for Mirai SYN flooding attack.

Consider a simple network shown in Figure 3.1, where $192.168.56.101$ is a Raspbian VM and $192.168.56.102$ is a Kali linux 2.0 VM, and $192.168.56.104$ is a rogue IoT device with Raspbian VM. We also have two other Raspbian VMs with IPs $192.168.56.114$ and $192.168.56.116$ that

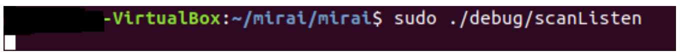


(a)



```
195  127.264441000   213.5.20.51       192.168.56.101  TCP        54  55170→14282 [SYN] Seq=0 Win=1000 Len=0
196  127.318753000   141.205.167.202   192.168.56.101  TCP        54  61576→28779 [SYN] Seq=0 Win=1000 Len=0
197  127.408719000   233.238.64.250    192.168.56.101  TCP        54  59161→41182 [SYN] Seq=0 Win=1000 Len=0
198  127.451768000   169.50.159.228    192.168.56.101  TCP        54  19774→65197 [SYN] Seq=0 Win=1000 Len=0
199  127.491420000   81.55.138.41      192.168.56.101  TCP        54  46519→63516 [SYN] Seq=0 Win=1000 Len=0
```

(b)



```
[redacted]-VirtualBox:~/mirai/mirai$ sudo ./debug/scanListen
```

(c)

Figure 3.2: a) Mirai Bot working diagram, b) Mirai successful SYN Flood attack, c) Mirai attack loader process after dropping the attack

are used in other attack scenarios.

To infect the environment, we have disabled our security applications running over the SDN Controller and compiled the Mirai source code in Kali Linux VM. All the Raspbian VMs are setup with a default username/password as ”admin/admin”. When the Mirai code is executed in one of the VM, the malware is injected into the network and they are infected. Figure 3.2b shows the SYN Flood traffic to the IoT virtual machine with IP $192.168.56.101$ (originating from $192.168.56.102$).

Then we activated our security applications (*PbSA* and *ISA*), We have policies specific to *http* service (PE_{18}) and *telnet* access (PE_{16}). PE_{18} states that *http* communication from IoT sources A and B to destination data center will be allowed. PE_{16} states that any *telnet* request to IoT devices from any source host will be dropped. So the PbSA denies *telnet* access to the attacker and prevents infection of the IoT devices. So that they cannot be turned into zombies.

PE_{16}= $< *, *, *, *, *, *, *, *, *, *, *, 23, *, * >:< Deny >$

PE_{18}= $< *, (A, B), *, *, *, 192.168.56.110, *, *, *, *, *, (80, 443, 8080), *, * >:< Allow >$

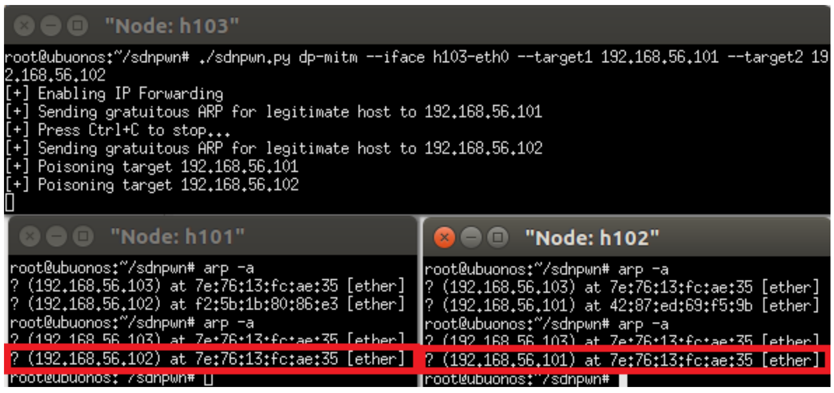
PE_{20}= $< *, *, *, *, *, 192.168.56.110, *, *, *, *, *, 21, enc, * >:< Allow >$

Now consider that a new zombie IoT device tries to access the network. Authentication will fail due to ISA application and IoT Authentication Authority.

Now, let us consider that a device has been physically tampered with and made into a zombie and was able to successfully authenticate itself. In this case, the PbSA application comes into operation and will prevent the zombie from launching flooding attacks. This is achieved using $PE18$ and PE_{20} which only allow a zombie device to send HTTP traffic and encrypted FTP traffic to the Data Centre ($192.168.56.110$). However, the zombie is not able to carry-out flooding attacks (such as UDP flooding, SYN flooding and HTTP flood). Furthermore, bot scanning is not possible due to PE_{16}. Hence, the bot can not create new bots as well as flood the network.

Every vulnerable device responds to the ScanListener in Mirai. Mirai has another process listener called Loader that lists the responses from the vulnerable hosts. Figure 3.2c shows the list of active vulnerable hosts. In our case, this is zero. Hence this shows how device specific policies in our security architecture can help to defend IoT devices.

Threats 3 & 4: Threats 3 & 4 are interlinked with each other. Here, in order to launch a MiTM attack (Threat 4), we have used ARP cache poisoning of host user/devices. The ARP cache poisoning uses ARP spoofing (Threat 3).

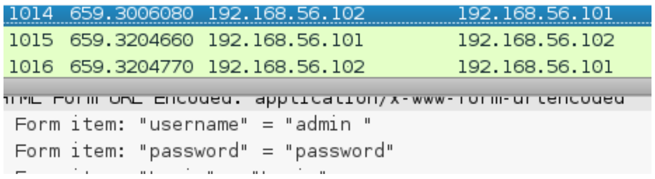


```
"Node: h103"
root@ubuonos:~/sdnpwn# ./sdnpwn.py dp-mitm --iface h103-eth0 --target1 192.168.56.101 --target2 19
2.168.56.102
[+] Enabling IP Forwarding
[+] Sending gratuitous ARP for legitimate host to 192.168.56.101
[+] Press Ctrl+C to stop...
[+] Sending gratuitous ARP for legitimate host to 192.168.56.102
[+] Poisoning target 192.168.56.101
[+] Poisoning target 192.168.56.102
```

```
"Node: h101"
root@ubuonos:~/sdnpwn# arp -a
? (192.168.56.103) at 7e:76:13:fc:ae:35 [ether]
? (192.168.56.102) at f2:5b:1b:80:86:e3 [ether]
root@ubuonos:~/sdnpwn# arp -a
? (192.168.56.103) at 7e:76:13:fc:ae:35 [ether]
? (192.168.56.102) at 7e:76:13:fc:ae:35 [ether]
root@ubuonos:~/sdnpwn#
```

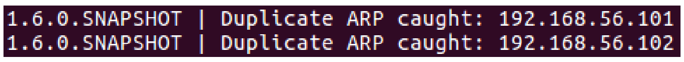
```
"Node: h102"
root@ubuonos:~/sdnpwn# arp -a
? (192.168.56.103) at 7e:76:13:fc:ae:35 [ether]
? (192.168.56.101) at 42:87:ed:69:f5:9b [ether]
root@ubuonos:~/sdnpwn# arp -a
? (192.168.56.103) at 7e:76:13:fc:ae:35 [ether]
? (192.168.56.101) at 7e:76:13:fc:ae:35 [ether]
root@ubuonos:~/sdnpwn#
```

(a)



```
1014  659.3006080  192.168.56.102          192.168.56.101
1015  659.3204660  192.168.56.101          192.168.56.102
1016  659.3204770  192.168.56.102          192.168.56.101
HTML Form URL Encoded: application/x-www-form-urlencoded
 Form item: "username" = "admin "
 Form item: "password" = "password"
```

(b)



```
1.6.0.SNAPSHOT | Duplicate ARP caught: 192.168.56.101
1.6.0.SNAPSHOT | Duplicate ARP caught: 192.168.56.102
```

(c)

Figure 3.3: (a) Successful MITM attack, (b) Capturing Admin/Password (c) Detected Attack

These two attacks are typical of an insider attack in any IoT infrastructure, where an employee collects sensor data for malicious purposes [10]. First, we will present how the attacks work and then we will show how our security architecture helps to protect the network.

We have used one of the Raspberry VMs as an IoT device, and have used a web-server (IP: 192.168.56.101) to log the IoT data. The web-server has a dedicated user/password for the admin user and devices. In this case, it is: “admin/password ”. The IP address of the Kali Linux VM is 192.168.56.102. The adversary uses SDNPWN Toolkit to poison the device/user ARP caches using *arpspoof* attack (Threat 3) [11, 12]. To poison the ARP cache, the adversary first sends a legitimate flow request to the Controller to install a flow rule, which establishes a route to the victim IoT device. Then, the adversary forges a gratuitous ARP request to poison the

APR cache of the IoT device. The adversary also poisons the ARP cache of another user. The IoT device packets are re-routed through the malicious adversary. This essentially allows the adversary to view, copy, modify any instruction/data to or from the IoT device. The adversary is also able to launch a MiTM attack (Threat 4). Moreover, it introduces the possibility of additional attacks such as SSL stripping and session hijacking. Figure 3.3a shows a successful attack. Figure 3.3b shows the administrator log-in request containing the login ("admin") and password ("password") that have been captured.

With our security architecture, each of the devices, as well as the user machines, are authenticated before they are connected to the IoT infrastructure using ECC. An adversar, who is not an insider, will not be able to bypass the authentication phase. On the other hand, if the adversary is an insider and is able to pass the ECC authentication phase, then the adversary will be restricted by the security policies in the authorization service. Also, at the end of the ECC authentication phase, each IoT device has established a secret key, which is used to encrypt its flows. This creates a secure channel which the IoT device can use to send the credentials to the web-server. Hence, the adversary is unable to steal the credentials. Figure 3.3c shows the attempted attack detected by our architecture.

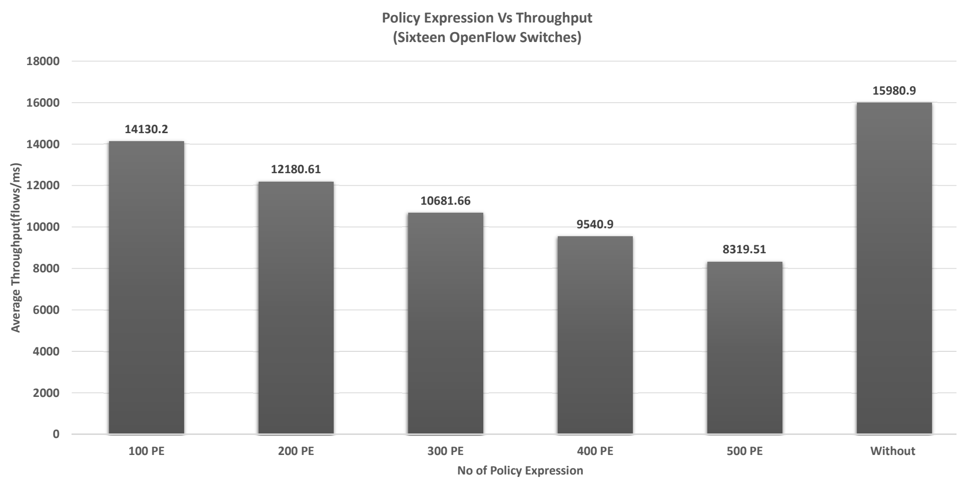
3.2 Performance

In this section, we present the performance of our security application and IoT devices connected to the SDN network.

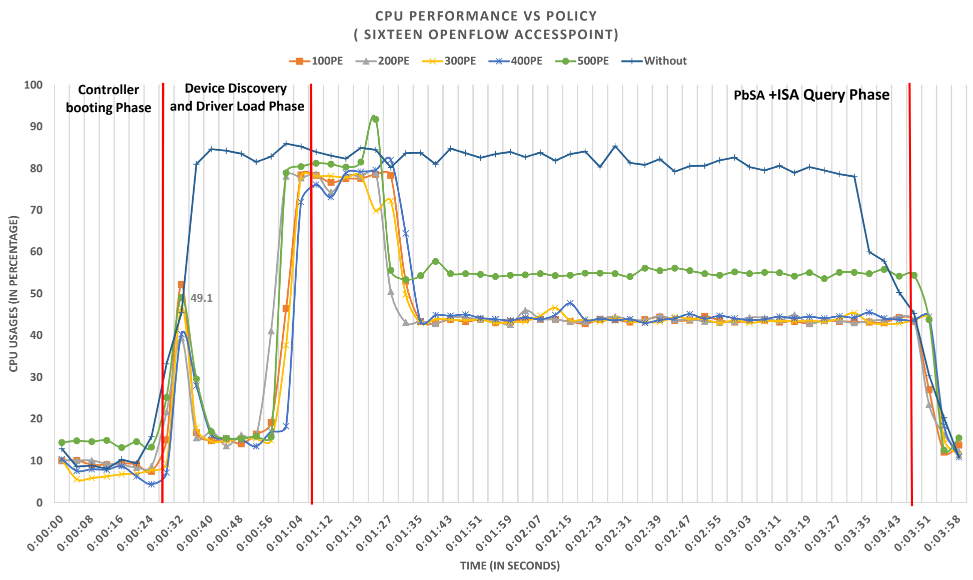
Application Performance

First, we present the performance of our applications running over ONOS. We measure the throughput, CPU usages, and heap memory usage with and without our security architecture using CBench and JConsole. Our topology uses 16 OpenFlow switches each connecting to 500 hosts.In this report, we focus mostly on the security issues of an IoT network infrastructure rather than the performance issues.

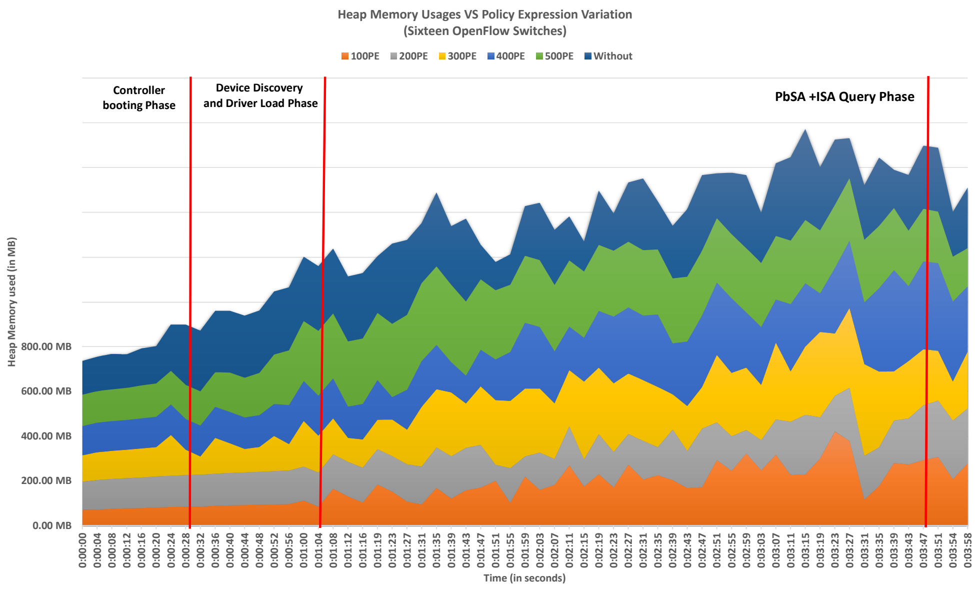
Figure 3.4a shows a comparison between throughput of ONOS running with and without our security applications. We varied the PEs and measured the throughput. The bar chart illustrates



(a)



(b)



(c)

Figure 3.4: a) Throughput, b) CPU Usages, and c) Heap Memory Usages

a decrease in throughput with increasing PEs. With 500 installed PEs, throughput is 8319.51 flow/ms. Without PbSA, ISA and with default applications running over ONOS, the throughput is 15980.9 flow/ms.

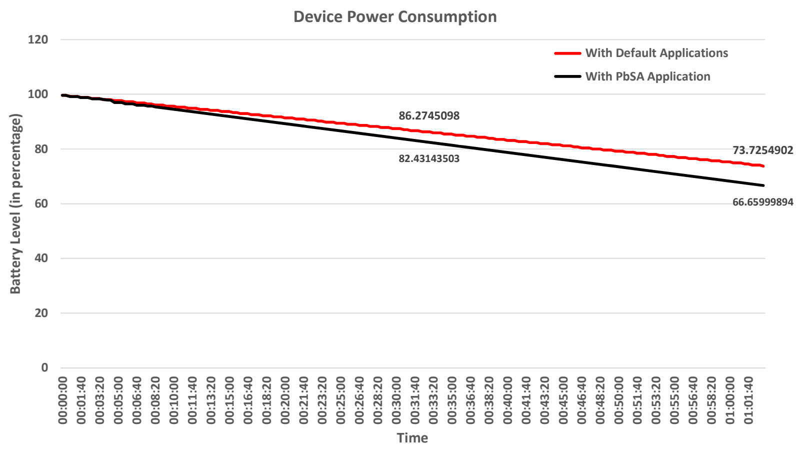
Figures 3.4b and 3.4c shows CPU and heap memory usage with and without our applications; we have varied the PEs from 100 to 500. With default applications, ONOS installs a number of flows in the OpenFlow switches, and this incurs CPU load as well as heap memory usage. On the other hand, applications associated with our security architecture only installs the permitted flows thereby incurring less CPU and heap memory usage. We have provided a comparison between both cases by changing the number of installed PEs. We have marked three distinct zones in the curves based on ONOS tail log messages. They are i) Controller booting phase, ii) Device discovery and driver loading phase, and iii) PbSA+ISA query phase. The phases are marked using a red line in the graphs (Fig 3.4b, 3.4c).

During Controller booting phase, the various core Controller modules/ NBI applications gets activated such as DHCP, Forwarding application etc, incurring a huge consumption in CPU and Heap memory resources. In case of device discovery it also increases. As mentioned previously with cbench, in case of normal SDN operation, each end host pings each other incurring huge CPU and Heap memory usages. But with our security application some of the end host requests are dropped incurring less CPU and Heap memory usages.

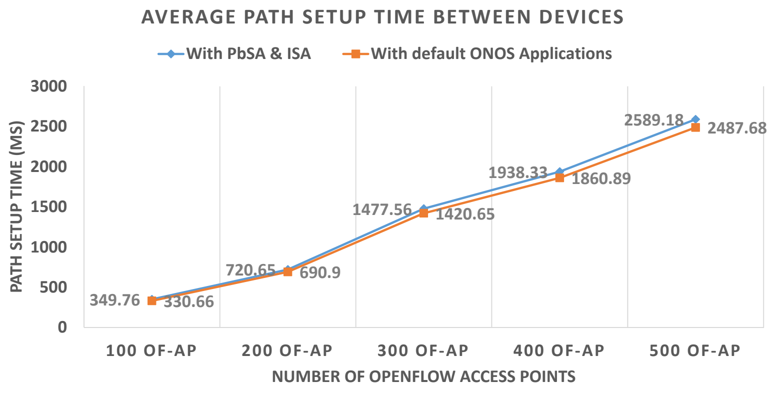
In Figure 3.4b, without our security architecture associated application, the CPU uses are steady at around 10% when ONOS boots up. During the device discovery process, CPU usage increases to 84%. Finally, while the flows are installed and communication process starts between the IoT devices, the CPU usage remains consistent between 78-85%. With 500 PE, the PbSA and ISA running over ONOS Controller use 14% of the CPU. During the device discovery phase, CPU usage increases to 80%. Due to non-permissible security policies, communication requests are dropped by the Controller and the CPU usage decreases steadily. Finally, it becomes constant around 55-60%.

Heap memory usage follows a similar pattern to the CPU usage. Without PbSA and ISA, it remains at 150-160 MB during the ONOS booting phase. It increases up to 280MB during the device discovery phase. Finally, in the communication establishment phase, it steadily increases to 370 MB. With 500 PEs in PbSA and ISA application running, heap memory usage becomes 140-150 MB during the Controller booting phase. During the device discovery phase, it steadily increases up to 290MB. Due to non-permissible request, the heap memory usage remains between 280-350MB.

Device Performance:



(a)



(b)

Figure 3.5: a) Device Power Consumption and b) Average Path Setup Time between Devices

We present the device end-to-end average path setup time (with varying OF-AP) and battery consumption of an active IoT device. In Figure 3.5a with default applications running in the IoT infrastructure, the IoT device's battery decreases steadily to 73% within an hour of communication. With PbSA and ISA, the IoT device's battery consumption decreases steadily to 67% within an hour. The security mechanisms used in the IoT devices drains the battery power, which is visible in 3.5a.

The average path setup time between IoT devices increases in both cases with the varying number of OF-AP (Figure 3.5b). With ONOS default applications running, the average path setup time for 100 OF-AP is $330.66ms$ and it steadily increases to $2487.68ms$ with 500 OF-APs. On the other hand, with PbSA and ISA running over the controller causes some delay in path setup time. In this case, with 100 OF-AP the average path setup time is $349.76ms$ and it steadily increases to $2589.18ms$ with 500 OF-AP.

In this Section, we will describe how the intra domain security arechitecture can be extended to multiple domains.

4. RELATED WORKS

There are several related works that are relevant to the work presented in this report. We have divided them into three categories for comparison with our work.

4.1 Policies and SDN

In RFC 1102, Clark introduced policy-based routing for autonomous domains [4], which proposed a simple policy syntax for interdomain communications. We have refined and extended the policy syntax to develop fine-grained security policy specifications targeted for IoT devices and SDN network characteristics. Das *et al.* in [13] present a context-sensitive policy framework for IoT devices, to control and protect information sharing between them. Their policies capture the diverse nature of IoT devices and their interaction with network users using attribute-based access control policy. Their work mostly focuses on the privacy of the user data. In our case, we have focused on securing the IoT network infrastructure using fine granular access policies. Beetle [14] is an access control policy framework for operating systems (Linux, Android) to control application interaction with peripheral device resources, and provides transparent access to network devices. Later, Hong used the Beetle framework in home network gateways to control IoT communications [15]. This work does not address authentication of IoT devices, or users, which our architecture does consider. Other work on access control policies for IoT devices can be found in [16, 17, 18]. However, none of these works have used SDN to manage and enforce the policies. Furthermore, they are mainly concerned with user security rather than IoT security.

Some relevant related works in the context of network policy control in SDN include [19, 20, 21, 22, 23]. However these works did not address security aspects and in particular, did not provide fine grained security policy based network management for IoT services, which has been the main focus in our work. In 2016, we have proposed a policy based security architecture for intra domain SDN communications in [24]. In this current work, we have extended this policy language and architecture for IoT network infrastructure and added new features to secure IoT devices from malicious devices and attacks.

4.2 IoT Security and Attacks

Pongle *et al.* in [25], Lyu *et al.* in [26] and Mendoza *et al.* in [27] consider various attacks to compromise IoT networks. These works focus on analysing the vulnerabilities in IoT network infrastructure, whereas our work is mainly concerned with the design of authentication and authorization policy based security architecture for IoT network infrastructure. Pa *et al.* in [28] provide an analysis of Telnet-based attacks on IoT devices. They propose an IoTPot (a honeypot) and IoTBox (a sandbox), which help to attract Telnet based attacks against various IoT devices running on different CPU architectures. Their work mostly focuses on analysing the IoT malware threats. In this report, we have provided a detailed analysis of Mirai attack. Moreover our security architecture helps to block telnet based attacks and prevent their spreading. Capellupo *et al.* in [29] present an analysis of present home automation devices such as Amazon Echo and TpLink smart plug and discuss how they can pose major security threats to home networks and to user privacy. Their work focuses mainly on addressing different threat vectors for the home IoT network infrastructure. Our approach helps to rectify some of the problems mentioned in their work; for instance, our architecture can help to prevent unauthorised users/devices gaining access to network services and IoT device traffic.

4.3 Use of SDN for IoT

A network activity based IoT device blacklisting approach has been proposed in [30]. The authors have used SDN features to dynamically block/ quarantine the malicious IoT devices using access control policies. They have shown how a Philips Hue light-bulb can be protected from user impersonation attack. This work only focuses on a particular type of IoT devices.

Our approach provides a security architecture for different types of IoT devices. Moreover, the devices can be authenticated before they use the network services, which makes the IoT network infrastructure more secure.

Use of SDN to control the IoT infrastructure is not new. SDN-WISE is the closest approach to our architecture [2]. The investigators have used the SDN Controller to limit the amount of information exchange between wireless sensor nodes. They introduced an API interface over the ONOS Controller to program the IoT nodes. In our case, we have introduced authentication and authorization services for securing the flows from IoT devices. Hesham *et al.* in [31] propose a simple Network Access Control (NAC) mechanism for machine to machine (M2M) devices using SDN. Their NAC architecture for M2M devices captures minimal network features (such as source and destination IP and bit rates). It is not as fine grained as our policies and does not consider the IoT device specific parameters in access control. Qin proposed MINA (Multinetwork INformation Architecture) [32], which uses SDN to manage IoT network infrastructure. The primary focus of this work is in the management of network services and scheduling flows. The work in [33] is concerned with the quality of service and considers the use of IoT infrastructure for managing large scale data generated from IoT sensors.

Xu *et al.* in [34] used SDN to prevent flow attacks in IoT infrastructure. They utilised dynamic access control mechanism and real-time monitoring of the flow packets to defend the network attacks. Black SDN [35] used SDN features to encrypt the header and payload to prevent some attacks in the IoT infrastructure. Miettinen *et al.* in [1] developed an IoT device identification system known as IoT Sentinel. IoT Sentinel used SDN to extract IoT device specific attributes. In their model, legitimate IoT devices were identified using predictive modeling. In their work, Gonzalez *et al.* in [36] considered management of a large number of IoT devices and their traffic using SDN. However, this is not a security policy based approach.

In our approach, we have developed a security architecture that consists of lightweight authentication and fine grained security policies, which have been enforced using SDN to control the IoT network infrastructure. None of the above SDN-IoT approaches provide both lightweight authentication and policy based authorization integrated with SDN architecture for IoT networks. This differentiates the work presented in this report from prior work in this area.

5. CONCLUSION

The report presents project summary on the “Software Defined Networks based Security Architecture for IoT Infrastructures” project funded by the ISIF group. There are three milestones for the project with specific deliverables for each milestone. In this report we presented a summary of the outcome of third milestone ”Security Architecture for IoT” which consists of the following tasks: i) Report on the Design of Security Architecture and Attack Detection in IoT Infrastructures and ii) Proof of Concept of Security Architecture for IoT. The specific contributions for third milestone can be summarised as follows:

- A SDN based security architecture that uses a policy based approach to secure IoT network infrastructure and detect malicious IoT devices and attacks in intra domains.
- Authentication of IoT devices using a light-weight protocol, which is in turn used in the provisioning of network services to authenticated IoT devices.
- Secure access to network services by authenticated devices using OAuth protocol.
- Extension of SDN based security architecture to secure IoT network infrastructure and detect malicious IoT devices and attacks in inter domains.
- Demonstration of the proposed security architecture and protocols using a realistic IoT scenario, showing how it can protect IoT infrastructure from attacks such as as the Mirai based DDoS attacks, and detect malicious IoT devices and flows.
- Performance analysis of the proposed solution for securing IoT infrastructure.

The authors would like to thank ISIF ASIA for their financial contribution to the Project.

BIBLIOGRAPHY

[1] M. Miettinen et al., "Iot sentinel: Automated device-type identification for security enforcement in iot," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 2177–2184.

[2] L. Galluccio et al., "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 513–521.

[3] K. Kalkan and S. Zeadally, "Securing internet of things (iot) with software defined networking (sdn)," *IEEE Communications Magazine*, no. 99, pp. 1–7, 2017.

[4] D. Clark, "Policy routing in internet protocols. request for comment rfc-1102," *Network Information Center*, 1989.

[5] A. Mohammadali et al., "A novel identity-based key establishment method for advanced metering infrastructure in smart grid," *IEEE Trans. on Smart Grid*, 2016.

[6] D. Hardt, "The oauth 2.0 authorization framework," 2012.

[7] S. Sciancalepore et al., "Oauth-iot: An access control framework for the internet of things based on open standards," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 676–681.

[8] X. T. Phan et al., "A collaborative model for routing in multi-domains openflow networks," in *Computing, Management and Telecommunications, International Conference on*. IEEE, 2013, pp. 278–283.

[9] R. d. R. Fontes et al., “Mininet-wifi: A platform for hybrid physical-virtual software-defined wireless networking research,” in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 607–608.

[10] B. Visan et al., “Vulnerabilities in hub architecture iot devices,” in *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual*. IEEE, 2017, pp. 83–88.

[11] Z. Yongchi et al., “Analysis of arp spoof attack,” *Programmable Controller & Factory Automation*, vol. 2, p. 017, 2005.

[12] D. Smyth et al., “Exploiting pitfalls in software-defined networking implementation,” in *Cyber Security And Protection Of Digital Services, 2016 International Conference On*. IEEE, 2016, pp. 1–8.

[13] P. K. Das et al., “Context-sensitive policy based security in internet of things,” in *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[14] A. A. Levy et al., “Beetle: Flexible communication for bluetooth low energy,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys ’16. New York, NY, USA: ACM, 2016, pp. 111–122.

[15] J. Hong et al., “Demo: Building comprehensible access control for the internet of things using beetle,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*, ser. MobiSys ’16 Companion. ACM, 2016, pp. 102–102.

[16] A. L. M. Neto et al., “Aot: Authentication and access control for the entire iot device life-cycle,” in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 1–15.

[17] C.-J. M. Liang et al., “Sift: building an internet of safe things,” in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. ACM, 2015, pp. 298–309.

[18] A. A. Yavuz, “Eta: efficient and tiny and authentication for heterogeneous wireless systems,” in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*. ACM, 2013, pp. 67–72.

[19] N. Foster et al., “Frenetic: A network programming language,” in *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[20] T. L. Hinrichs et al., “Practical declarative network management,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 1–10.

[21] J. Reich et al., “Modular sdn programming with pyretic,” *Technical Reprot of USENIX*, 2013.

[22] A. Voellmy et al., “Nettle: Taking the sting out of programming network routers,” in *Practical Aspects of Declarative Languages*. Springer, 2011, pp. 235–249.

[23] ——, “Maple: simplifying sdn programming using algorithmic policies,” in *ACM SIGCOMM Computer Comm. Review*, vol. 43, no. 4. ACM, 2013, pp. 87–98.

[24] K. K. Karmakar et al., “Policy based security architecture for software defined networks,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 658–663.

[25] P. Pongle et al., “A survey: Attacks on rpl and 6lowpan in iot,” in *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[26] M. Lyu et al., “Quantifying the reflective ddos attack capability of household iot devices,” in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 46–51.

[27] C. V. Mendoza et al., “Defense for selective attacks in the iot with a distributed trust management scheme,” in *Consumer Electronics (ISCE), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 53–54.

[28] Y. M. P. Pa et al., “Iotpot: analysing the rise of iot compromises,” *EMU*, vol. 9, p. 1, 2015.

[29] M. Capellupo et al., “Security and attack vector analysis of iot devices,” in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2017, pp. 593–606.

[30] V. Sivaraman et al., “Network-level security and privacy control for smart-home iot devices,” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*. IEEE, 2015, pp. 163–167.

[31] A. Hesham et al., “A simplified network access control design and implementation for m2m communication using sdn,” in *Wireless Communications and Networking Conference Workshops (WCNCW), 2017 IEEE*. IEEE, 2017, pp. 1–5.

[32] Z. Qin et al., “A software defined networking architecture for the internet-of-things,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[33] Y. Lu et al., “Sdtcp: Towards datacenter tcp congestion control with sdn for iot applications,” *Sensors*, vol. 17, no. 1, p. 109, 2017.

[34] T. Xu et al., “Defending against new-flow attack in sdn-based internet of things,” *IEEE Access*, 2017.

[35] S. Chakrabarty et al., “Black sdn for the internet of things,” in *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE, 2015, pp. 190–198.

[36] C. Gonzalez et al., “Sdn-based security framework for the iot in distributed grid,” in *Computer and Energy Science (SpliTech), International Multidisciplinary Conference on*. IEEE, 2016, pp. 1–5.