Using OpenFlow for Fine-Grained Network Access Control

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Masters of Cyber Security

at

The University of Waikato

by

Michael Washer

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2020

Abstract

This paper investigates the feasibility of a network access control solution which provides fine-grained access control in an OpenFlow software defined networking environment. Using the Faucet SDN controller a solution is developed which provides the dynamic allocation of both VLANs and ACLs to produce an access control solution for deploying user-specific access restrictions, allowing for consistent network access as clients move around the network. The EAPoL protocol is used for authentication, providing all major EAP methods for authentication and mac authentication bypass functionality for devices unable to connect to tradition secure environments.

Acknowledgements

I would like to thank my parents, Michelle and Alan for their continual support throughout my academic journey. I would also like to thank my brother and sister, Jonathan and Sarah-Michelle. I have always been able to look up to you and find quality role-models.

I would also like to thank the WAND team at The University of Waikato, especially Brad Cowie and Dr. Richard Nelson for assisting me throughout my research and providing me the resources, information and inspiration to write this paper.

Contents

Chapter 1

Introduction

1.1 The Problem

Computer networks are used to facilitate the spread of malware, leading to wide-spread damage felt throughout the business world. [1]

With advancements in exploitation techniques[2] the traditional 'perimeter' approach to network security is obsolete. Due to the increase in B.Y.O.D schemes and the proliferation of wireless technologies, it is no longer feasible to control the devices that attach to our networks. Firewalls and de-militarized zones (DMZ) are not enough to maintain network safety when threats arise from within the network.

Once a single device inside the perimeter is compromised, this device can both probe the local network, mapping and searching for vulnerable devices, and initiate reverse-shell communication bypassing most firewall systems.

The scope of this threat is increased with the broad adoption of Internet of Things (IoT) devices. A growing number of security vulnerabilities are being discovered in IoT devices and there is growing concern that they are becoming a common attack vectors for malicious actors.[3][4]

Studies have shown that many common IoT devices are being sold running out-of-date software with known vulnerabilities, many including remote code execution vulnerabilities.[5] These devices are always-on, network enabled and

if compromised can be used to monitor a network looking for vulnerabilities. A term has been coined for the development of IoT devices as insecure-by-design. [6][7]

When designing an enterprise network, complete client isolation is not feasible as this will effect the deployment of in-house services, such as NFS and Samba shares.

There are traditional solutions that when deployed provide the ability for limiting clients access to network resources, such as providing multiple VLANs and designating ports that shall be on each VLAN. However this solution does not accommodate clients that move over the network, such as B.Y.O.D devices and can be resource intensive due to port-management overhead.

Through the use of authentication we are able to block unauthorised access to a network, dropping traffic until a client verifies their identity. This is known as network access control (NAC).

By providing individual clients with different credentials we are able to differentiate between clients as they connect and even provide personalised resource limits through the use of fine-grained packet filtering techniques.

These fine-grained access rules can be defined as role-based access control (RBAC) policies and associated with users to provide fine-grained network access control.

As clients move around the network they will be forced to authenticate to access resources, on successful authentication their appropriate RBAC polices can be deployed on their connecting port regulating their access to network resources. This is known as fine-grained port-base network access control.

Port-based NAC is powerful, but switches and ports must be configured manually to support NAC which can lead to a large organisational overhead and increase the difficulty of making changes to the network.[8]

A way to mitigate the overhead of protocol-management on network devices is through the use of software defined networking (SDN).

SDN attempts to address the decentralised style of traditional network

management by moving all network control to central points, called controllers. These controllers manage the flow of traffic over the network, providing forwarding devices under their control with actions to perform on incoming messages. This movies all of the protocol logic away from the network into centralised locations, removing a large amount of the overhead experienced when performing protocol configuration on a traditional network.

I have separated the concerns above into a concise set of issues for reference:

1. Networks are scaling at a rate that is making them difficult to manage through traditional means.

2. Clients should be limited in their access to network resources as they can become compromised.

3. Deployment of network resource restrictions should be seamless and not unreasonably increase management overhead.

In this masters project I propose a system that will combine the centralised model of SDN and the security of NAC systems, minimising the overhead of traditional NAC systems while providing a secure network. I also propose that any solution that combines these design principles must also be light-weight and not use any vendor specific technologies as inter-vendor compatibility is crucial for widespread adoption.

This system must:

- use open and standard authentication protocols to allow interoperability between vendors.

- have the ability apply role-based access control policies based on authentication details.

- be light-weight and not overly hinder normal management of the network.

1.2 Contribution

Throughout this project, I work towards providing an NAC mechanism for the Faucet SDN controller. The NAC implementation provides an 802.1x implementation, providing EAP authentication for network access. EAP is an open-standard defining a network authentication mechanisms to provide credential-based authentication. The EAP-based protocol used throughout the project is EAPoL, providing EAP authentication over an 802.3 Ethernet networks. Mac authentication bypass (MAB) is also provided as an option for device that are unable to perform the EAP authentication.

Chewie, the developed project, is provided to the network through the use of an NFV port. Using SDN control mechanisms all EAPoL (Ethertype: 0x888e[9]) traffic from controlled ports is routed to the NFV port for processing and all other traffic is denied.

Once authentication is complete, credential-based session restrictions are put in place to manage a clients access to networked resources. This is achieved through the controller setting a combination of port-based ACLs and VLANs based on the provided RBAC information from Chewie.

Chewie supports the use of both EAPoL and MAB for authentication and RBAC restrictions are applicable to both authentication mechanisms.

1.3 Overview

This chapter, Chapter 1 presents an overview of the problem, defining key terms used throughout the rest of the report and specifies an overview of personal contributions made to the field.

Chapter 2 continues by discussing further background into network access control, the related protocols and the industry expectations of a NAC system.

Chapter 3 discusses the current state of software defined networking, contrasting two common open-source controllers and discussing the benefits of SDN.

Chapter 4 discusses the current state of literature for NAC in an SDN environment, including a review of related studies and how they have affected the direction of this paper.

Chapter 5 focuses on scope definition, providing an in-depth problem analysis; listing project completion requirements for reflection at the completion of the project.

Chapter 6 discusses detailed implementation detail for the project. Discussing the overall architecture of the chosen solution and how that has been implemented into source-code. This chapter focuses heavily on the inner-workings and implementation aspect of the system.

Chapter 7 defined personal contributions to the project. As the developed system was a join effort between multiple universities, this chapter defined my personal research process and developed features towards the end project.

Chapter 8 reflects on the difficulties experienced during the development phase, discussing the design iterations and listing some pitfalls to help future developers.

Chapter 9 is an evaluation of the project, assessing whether the defined completion requirements have been achieved and to what degree.

Chapter 10 discusses potential areas for future research and extension to the current system.

Chapter 11 is a conclusion chapter; summarising the findings and providing an overview of the research outcomes.

Chapter 2

Background: Network Access Control

Network Access Control (NAC) is the use of authentication protocols in combination with control policies to limit unauthorised access to network resources. [10]

A common use of NAC is Wi-Fi Protected Access (WPA), the security standard used to safeguard household wireless networks from unauthorised access. The personal version of WPA is achieved through the distribution of pre-shared keys (PSK). A PSK is a key that is distributed to all authorised users and is used during the authentication process to prove authorisation. It is also used to derive the symmetric encryption keys used for communication between the client and the wireless AP.

The PSK model is not a valid option for enterprise[11], due to overhead on the event of staff turnover. If staff turnover occurs a new PSK must be generated and distributed to all remaining staff to fully remove the ex-employee's access.

The PSK model also provides no accountability. As all clients share the same credentials there is no way to associate performed actions and traffic to specific authorised users on the network. Access logs can be creating using MAC addresses of devices but this is not viable solution as there is no clear

association between devices and authorised users. Also authorised users could use disposable devices or spoof MAC addresses to perform malicious activities without revealing their identity.

To address these issues, the the Internet Engineering Task Force (IETF)[12] proposed the Extensible Authentication Protocol (EAP)[13]. EAP is a framework by which authentication protocols can be written and provides common functions such as negotiation of EAP-methods.

An EAP-method is an individual authentication method. It defines the specifics of how authentication is to be performed between a user and server, but is not a line-protocol and does not provide a mechanism for transporting the messages across a network. There are over 30 EAP-methods defined with EAP-MD5, EAP-TLS, EAP-TTLS and EAP-PEAP being the most common.

All EAP-methods support individual authentication credentials for clients providing a solution for accountability, but does not define a mechanism for logging or authorisation management.

As EAP-methods do not define a line-protocol, an encapsulation layer must be used to transport EAP messages over a network. This encapsulation layer can be an Authentication, Authorization and Accounting (AAA)[14] protocol allowing for additional attributes to be transported with EAP messages. These attributes can be used by network components to enforce authorisation policies and the supporting AAA servers can be used for storage of access request logs and user credentials.

The 802.1x protocol defines a port-based network access control solution for IEEE 802 networks. By creating a separation of authentication and access control, EAP-methods are used to perform authentication while 802.1x provides mechanisms for blocking ports. As EAP messages must be encapsulated, Ethernet headers are used to transport them over the LAN network. Before successful authentication is achieved all traffic, except required EAP messages, is dropped on a port. On successful completion of the EAP authentication process the port is opened and traffic is not dropped from the

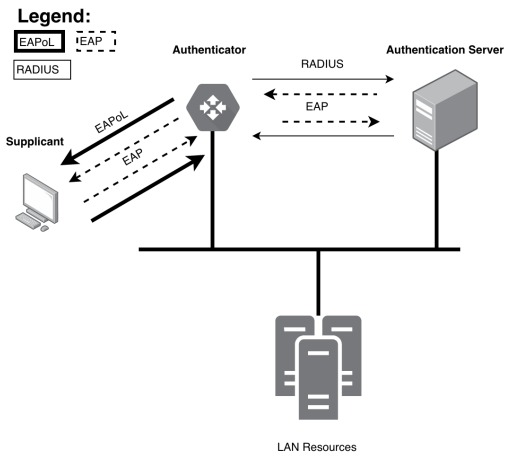

Figure 2.1: 802.1x Protocol

authenticated client.

2.1 Extensible Authentication Protocol (EAP)

EAP defines a framework for the design and development of network authentication protocols. It provides common functions and negotiation for authentication methods, these methods are known as EAP-methods.

EAP-methods define the method used to provide identity verification throughout the authentication process. An example of an EAP-method is EAP-TLS.

EAP is not a line-protocol and required an encapsulation protocol layer to send requests over a network. Examples of encapsulating protocols are EAPoL or RADIUS, which are discussed in further detail below.

The EAP framework splits the responsibilities of authentication into three main components.

- The Supplicant
- The Authenticator
- The Authentication Server

These components are expected in all EAP-methods and are required by protocols using EAP as the authentication method. The components are described

in more detail below.

2.1.1 Supplicant

The Supplicant, also known as the peer in RFC 3784[13], represents a client that wishes to join the controlled network. It communicates directly with the Authenticator, negotiating a supported EAP-method and provides credentials or authentication tokens to perform authentication.

2.1.2 Authenticator

The Authenticator receives EAP authentication requests from one, or many, Supplicants and forwards them to the Authentication Server. Is is responsible separating the Supplicant from the Authentication Server and provides a distributed network of authentication points.

The Authenticator is used as the 'gate-keeper' of the network and is capable of performing authentication without the use of the Authentication Server.

2.1.3 Authentication Server

The Authentication Server is the primary store of client identities. It receives requests from one, or many Authenticator nodes and is generally used for performing EAP authentication with the Supplicant. [Refer to figure 2.2]

The Authenticator is also notified of completed authentication attempts by the Authentication Server. If the Authentication Sever is communicating using an AAA protocol at the encapsulation layer, it may provide access control policies to the Authenticator to be added as restrictions on the session that is created.

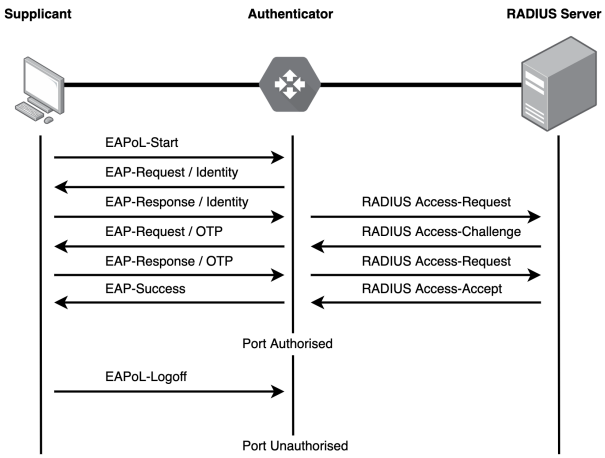

Figure 2.2: 802.1x Packet Diagram

2.2 Remote Authentication Dial-In User Service (RADIUS)

Remote Authentication Dial-In User Service (RADIUS) is an AAA protocol. It can be used as an encapsulation protocol for EAP-methods and provides mechanisms for sharing AAA information that may not be available with other EAP encapsulation methods, such as EAPoL. In the 802.1x design, RADIUS is used for communicating EAP messages between the Authenticator and Authentication Server and is used to provide authorisation information to the Authenticator. RADIUS servers also perform as Authentication Servers, providing additional functionality such as storing a centralised storage of authentication credentials, authorisation rules, accounting logs and other AAA information that is associated with user credentials in its credential-store. [15] RADIUS also provides support for authentication without EAP by receiving user credentials through RADIUS-Attributes and perform authentication using the provided information.

There are 4 primary RADIUS packet-types used during RAIDUS authentication; Access-Request, Access-Challenge, Access-Accept and Access-Reject.

- Access-Request: Initiates the authentication process with the server. It is used to provide credential information to the server and respond to

Access-Challenge requests.

- Access-Challenge: Requests further information about the client and their credentials. This can include performing encryption / decryption functions using their private key to prove identity and is used for EAP authentication.

- Access-Accept: The user has successfully authenticated with the network as is to be granted access to network resources. RADIUS attributes can be attached to this message to provide session restriction requirements to the Authenticator.

- Access-Reject: The user has been unsuccessful with the authentication attempt and must not be granted access to network resources.

Special RADIUS Attributes can be attached to RADIUS Access-Accept messages used to notify an Authenticator about restrictions that must be placed on an authenticated session. There are a large number of RADIUS attributes used for access restriction policies supplied as open-standards and vendor-specific attributes.

During this project I will only be looking at open standard attributes [16] provided by the IETF and IANA in an attempt to provide a multi-vendor support. The IETF have also provided a set of usage guidelines available for best practices that I will attempt to comply with.[17]
I have defined the attributes that are of note for this project below:

Attribute 64 — Tunnel-Type:

This attribute indicates the type of tunnel that will be used for the Supplicant's session. This can be used in conjunction with to Tunnel-Private-Group-Id to specify a VLAN to be allocated for a session.

Attribute 81 — Tunnel-Private-Group-Id:

This attribute indicates the name or ID of a tunnel in which the Supplicant's

session will be placed. When used in conjunction with RADIUS 64 attribute, it can be used to define a VLAN ID.

Attribute 11 — Filter-Id:

This attribute indicates the name of a filter that will be applied to the Supplicant's session.[18]

Attribute 92 — NAS-Filter-Rule:

This attribute indicates an individual filter rule that can be applied to the Supplicant's session. The format of the rule is not clearly defined in the specification however the two main approaches implemented by main vendors.

Cisco implements dACLs using a vendor-specific alternative that contains iOS terminal commands for setting up an ACL.[19].

HPE uses the standard attribute [91] and has attempted to standardise the format of a rule, defining both an allow and deny rule syntax. [20]

Note, for most vendor implementations the Filter-ID and Tunnel-Private-Group-Id attributes should be used to identify existing resources and does not guarantee the creation of VLANs, ACLs or other filters.

2.3 Extensible Authentication Protocol over LAN (EAPoL)

Extensible Authentication Protocol over LAN (EAPoL) is a line-protocol for EAP messages. It is used to encapsulate and transport EAP messages over an IEEE 802 network.

EAPoL encapsulates EAP messages in Ethernet frames. This provides a generic solution for port-based authentication as Ethernet is supported protocol of traditional networking equipment. EAPoL frames can be identified by their ether-type of 0x888e and support transport of all EAP-methods as encapsulation of the whole EAP message is achieved. [Refer to Figure 2.3]

2.3.1 The Authentication Process

When a Supplicant connects to a controlled port, it sends an EAPoL-Start message to the multi-cast address 01:80:c2:00:00:03. Authenticators controlling the network boundaries receive these requests and initiate the authentication process. [Refer to Figure 2.2]

The authentication process consists of the 5 steps discussed below:

1. On receiving the EAPoL-Start message, the Authenticator sends an Identity-Request in response to the Supplicant requesting further credential information.

2. The required credential information is packaged into an EAP Identity Response and forwarded through the Authenticator to the Authentication Server.

3. On the Authentication Server receiving the Identity-Response a challenge is generated and provided back to the Supplicant.

4. The Supplicant completes the requested challenge using the EAP-Method that is relevant, and an EAP-Request is sent back to the Authenticator for processing by the Authentication Server.

5. On successful authentication, the Authenticator will receive the EAP-Success message and open the port. On an unsuccessful authentication attempt, the Authenticator will receive the EAP failure message and the port will remain closed.

2.4 IEEE 802.1x Protocol

The 802.1x protocol defines a port-based network access control solution by combining the use of EAPoL and RADIUS.

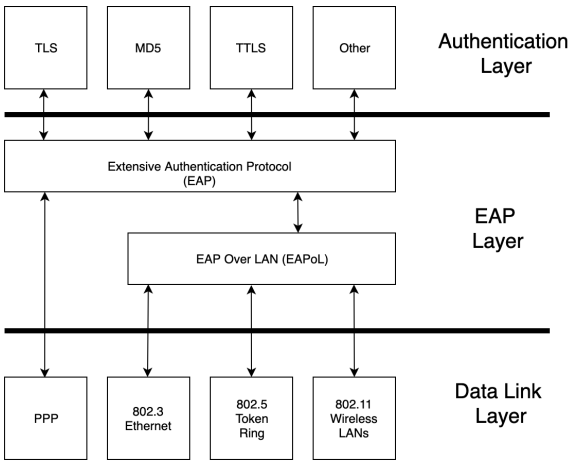

Figure 2.3: EAP Layers

A separation of authentication and access control is created by using EAP-methods to perform authentication and RADIUS messages to transport AAA-related information.

Communication between the Supplicant and Authenticator is achieved using EAPoL to transport EAP messages over the Ethernet network. Communication between the Authenticator and the Authentication Server; referred to as the RADIUS server; is achieved using RADIUS to transport EAP messages.

All traffic, excluding EAPoL traffic, is dropped by the Authenticator; Blocking unauthenticated clients from accessing networking resources and requiring a successful authentication attempt before the port is opened for network access.

On completion of a successful authentication attempt, the Authentication Server will provide the Authenticator with any access restrictions defined in its credential-store as RADIUS attributes. These attributes can be used by capable Authenticators to apply VLANs or ACLs on client sessions before opening controlled ports. [Refer to Figure: 2.1]

Capabilities of filtering on a controlled port include; limiting the type of traffic that users can send, limiting access to resources and placing users on separate VLANs. A practical example of this could involve actions such as filtering all RDP traffic for all non-technical staff or placing all members of a

work-team in the same VLAN.

As 802.1x provides a NAC solution for IEEE 802 networks it can be paired with WPA to provide an enterprise alternative to the PSK implementation. When clients connect, their authentication requests use individual credentials and all other traffic is blocked until successful authentication is achieved. This removes the overhead of WPA-PSK on the event of staff turnover and provides the ability for logging individual network access requests.

2.5 MAC Authentication Bypass

MAC Authentication Bypass (MAB) is a method of providing access to an 802.1x controlled network for devices that are unable to perform EAP authentication. MAB performs a MAC-based authentication attempt on behalf of a connecting client. This authentication then occurs without the knowledge of the client, and access is granted on a successful authentication attempt.

A MAB-enabled Authenticator is notified of a connecting client through Dynamic Host Configuration Protocol (DHCP) broadcast requests. The DHCP broadcast messages are sent by new clients requesting configuration information of the local network. This allows clients to configure themselves in accordance with the network policies.

From these requests the Authenticator can extract the source MAC address, build an AAA Access-Request containing the MAC information and send this to the Authentication Server.

As the authentication is performed between the Authenticator and Authentication Server, AAA authentication methods can be used and EAP-methods are not required. The MAC address information can be provided to the Authentication server using AAA attributes and the authentication can be performed without a challenge being performed by the Supplicant.

As MAB uses MAC addresses for authentication[21] it is a less secure alternative to EAP and should only be applied on an individual port basis. NIC

drivers can be used to obfuscate their original MAC with a user-defined MAC. This is known as MAC address spoofing and can be used to authenticate to a MAC-based authentication system. However, this is more secure than leaving open ports for devices that are unable to perform EAP-based authentication.

Chapter 3

Background: Software Defined Networking

As networks grow in size, traditional network management techniques are becoming more time-consuming. Routine network maintenance and minor changes can require every affected network device be manually reconfigured to comply with the new configuration requirements.

Software defined networks (SDN) attempt to resolve this problem by extracting the control mechanisms out of individual networking devices, placing them into centralised locations, called SDN controllers.[22]

These central SDN controllers produce a set of forwarding rules by performing decision logic about forwarding traditionally found in the networking devices. They perform protocols equivalent to traditional route-decision protocols, such as BGP and OSPF, and distribute the produced forwarding rules to controlled networking devices. These received forwarding rules are used by networking devices as references on ingress message events and are used orchestrate the flow of traffic over the controlled network.

The produced forwarding rules are distributed to relevant subscribing networking devices, also known as forwarding devices, and are referenced on ingress message events for actions to be performed on the packet / frame.

As the message traverses a forwarding device, a match is made according

to the available forwarding rules and the associated actions with the matched rule are on the message. As the action-list is executed message values may be changed and the message is appropriately duplicated, discarded or forwarded according to the stored forwarding rule.

With the introduction of centralised controllers there is added latency when no matching rule is present on the forwarding device. As all networking mapping protocols are performed on the controller, the controller must be informed of the ingress message and a new forwarding rule produced for the forwarding device to process the unmatched message.

Efficient controllers are configured to populate forwarding devices with common forwarding rules to avoid unnecessary overhead produced by contacting the controller. Performance of the data plane is not hindered if a matching forwarding rule is present on the forwarding device; no contact with the controller is required and the associated forwarding actions are performed locally. [23]

SDN Controllers are generally run in software; This allows for rapid deployment of software patches and network configuration changes that are propagated through the network. These changes are only required to be performed on the central controllers and forwarding rules reflecting these changes will be distributed to all relevant controlled switches. This includes patches to network mapping protocols and traffic engineering logic.

As network mapping logic is performed centrally, forwarding devices are limited in their knowledge of the OSI layer requirements and forwarding devices are able to perform operations of both a layer-2 and layer-3 networking device. If the SDN controller provides forwarding rules that include removing VLAN tags or replacing MAC address values for IP routing then these can be performed on forwarding switches.

SDN Controllers also maintain the current state of network configuration, providing for a centralised store of configuration. This creates an easier mechanism for configuration back-up and roll-back.

3.1 Networking Devices

Network devices functionality can be segmented into two abstract logical components.

1. The control plane, which is responsible for performing decisions about traffic flow.

2. The data plane, which performs all operations on messages and is responsible for forwarding packets over the network.

3.1.1 The Control Plane

The control plane is responsible for maintaining a network map of the current network topology. This is traditionally achieved through the use of distributed networking protocols, such as OSPF and BGP. A map of currently available routes is built and simple forwarding rules are produced for consumption by the data plane. The control plane makes all decisions about traffic flow and is where traffic engineering occurs. Latency of operations is less importance in the control plane, with efficiency of forwarding rules and building shortest path routes taking precedent. [24]

3.1.2 The Data Plane

The data plane, also known as the forwarding plane, is responsible for performing all packet manipulation and forwarding actions. As ingress messages arrive, actions are performed on these messages before they are forwarded or discarded in accordance with the matching forwarding rules. The primary goal of the forwarding plane is to perform forwarding and manipulation actions as quickly as possible. Actions performed on messages include moving messages from an ingress port to an egress port, or discard.

3.2 Traditional Networking Devices

Traditional networking devices perform the actions of both the control plane and forwarding plane individually. The control segments of each networking device must use standard networking protocols, such as OSPF, to build and maintain their own map of the current network. This requires constant communication between distributed networking devices and the use of special protocols to avoid developing loops in the network, such as RSTP.

3.3 SDN Networking Devices

SDN networking devices do not perform control plane operations. Centralised SDN Controllers build forwarding rules that are used by controlled networking devices to perform all operations of the data plane. These devices are also known as switching devices. As forwarding instructions can include operations at both OSI Layer 2 and Layer 3, switching devices are able to perform operations of traditional routers such as inter-VLAN routing. All mapping protocols are performed by the controller, simplifying the development and maintenance of the network map and providing a central place to perform traffic engineering. If an unknown packet arrives at a switching device, it consults the controller and a new forwarding rule is established on the switching device.

3.4 OpenFlow

OpenFlow (OF) is a standard that defines the communication between SDN controllers and SDN switching devices on a network. [25] OpenFlow controllers produce 'flows', which consist of matching criteria and associated actions, and distribute them to controlled devices using OpenFlow FlowMod messages.

- Matching criteria is used to define which packets should be affected by a flow. Options for specifying matches on values such as destination address of packets and ether-type of frames.

- Action fields are used to define operations that will be performed on a packet as it traverses a flow, such as appending a VLAN tag or outputting the packet out a port.

Received flows are organised on controlled devices through the use of flow-tables, which are chained together to provide a complex packet-processing pipeline allowing multi-step actions.

When a controlled switch encounters a packet that does not match any flows in the first table, a packet-in message is generated and sent to the controller. The switch may send the ingress packet's metadata, or the whole packet back to the controller, based on configuration. On receiving the packet-in message, the Controller will generate and respond with an appropriate flow rule that applies to the packet. This flow rule is added to the origin switch for use with future packets, slowly populating the flow-tables. It is common for OpenFlow SDN Controllers[26] to pre-load flow rules into their forwarding devices to avoid control network congesting and improve latency. [27]

When a match is made on a packet, the actions associated with the flow are performed on the packet and it may be forwarded to another flow-table, to an outgoing port or discarded in accordance with the flow rule.

3.5 OpenFlow SDN Controllers

A number of open-source OF SDN controllers are freely available on the market. I will discuss 3 available controllers and contrast them against the requirements for this project.

3.5.1 Faucet

Written in Python 3, Faucet is a simplest implementation of an OpenFlow SDN controller discussed. It is written using the Ryu SDN framework, following a light-weight design philosophy providing a minimalist approach to controller design. The upper-bridge of Faucet does not provide a general-purpose API

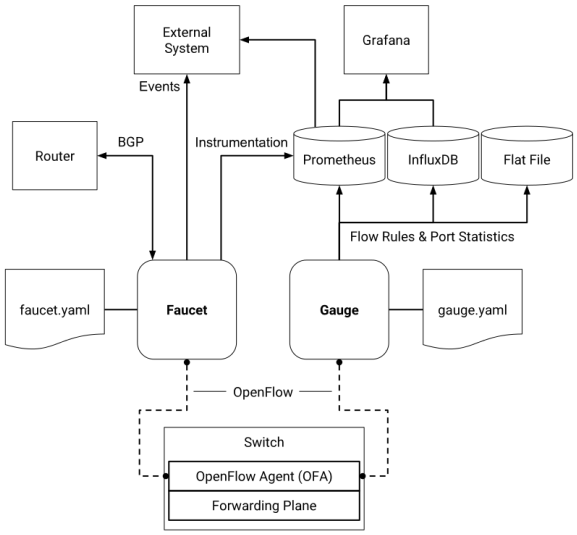

Figure 3.1: Faucet Architecture Diagram (From the Official Faucet Documentation [28])

but allows for extension through the development of custom APIs that are used to attach plugins.[Refer to Figure 3.1]

These plugins are able to hook into Faucet's internal events and interact with the flow-control engine. An example of a plugin is the BGP plugin, Beka[29], which is used to provide BGP functionality to a Faucet-controlled network.

Faucet only supports OpenFlow on the south-bridge, which dramatically simplifies the design but may be limiting as other SDN controllers support a number of other protocols.

Currently there is no support for NAC functionality in a Faucet driven network.

3.5.2 Floodlight

Written in Java, Floodlight is another simple SDN controller providing the required functionality for managing an OpenFlow network. Modules are available for adding stateful firewall functionality to the controller and a Web-UI for ease of management of the controller. Deploying Floodlight on a virtualised environment was achieved fairly easily and the controller is intuitive to

use however a combination of the explicitness of Java and the customisability of modules makes the Floodlight code-base far larger than Faucet's and will require more initial overhead for development towards a proof-of-concept fine-grained NAC solution. There is an example of a course-grained NAC solution using Floodlight[30] discussed further in the next chapter.

3.5.3 OpenDaylight

OpenDaylight (ODL) is a complex SDN controller, supporting a number of south-bridge protocols including OpenFlow and NetConf. It is currently the industry standard for large-scale deployment of SDN and is heavily customisable. The design philosophy is an 'all-in-one' controller, making it difficult to work deploy but provides the most functionality of all three developers.

ODL will not be investigated further due to the complexity of deployment and fragmented documentation due to the multiple available versions provided by different vendors.

3.6 Conclusion

SDN provides a solution to the maintainability issues experienced as networks grow in size. OpenFlow is the leading open standard for communication between SDN controllers and their controlled switching devices. Comparing the controllers discussed in this chapter, both Floodlight and Faucet appear to be valid options for developing a proof-of-concept NAC solution providing fine-grained access control. ODL is too complex for the purposes of this project and would better suit a large-scale development project. It requires an unreasonable overhead for a smaller project like this one. Unlike Floodlight, there are no NAC examples available for Faucet. However, I have been unable to locate any code for the NAC solution referenced in the 'SDN-Driven Authentication and Access Control System'[30] paper and it is my opinion that Python 3 is a better programming language than Java for rapid development of smaller

systems. Also, the plugin system provided by Faucet will provide adequate event-hooks for an 802.1x module to attach to to the flow-engine of Faucet, in a similar fashion to the Beka module. For these reasons I have decided that the Faucet controller will be a better underlying system for prototyping the NAC solution required for this project and will be used throughout the project for development purposes.

Chapter 4

Background - Related Studies

A number of studies have influenced decisions made throughout this project. The most influential have been listed in this chapter, noting their relevance and the decisions they have impacted. Using these papers as reference, I was able to refine the scope of the project and define remaining gaps in the current literature that I attempt to fill with this thesis.

4.1 Vilnius University - Course-Grain SDN NAC

Two papers produced by the Vilnius University of Lithuania have been used as the foundation of my research. Involving the development and deployment of an SDN-based NAC solution using Floodlight, an open-source SDN controller, and open-standard RADIUS technology.

The first paper investigates the difficulty of developing such a system and the latter discusses the deployment of the system at a university campus.

4.1.1 SDN-Driven Authentication and Access Control System

'SDN-Driven Authentication and Access Control System'[30] investigates the feasibility of implementing an 802.1x network access control solution in an SDN environment.

This paper looks specifically at the limiting access to network resources through the use of segregated virtual networks using VLANs. As users connect to a controlled port, all non-essential traffic is blocked while redirecting the client to a web-based authentication interface hosted on at the Floodlight controller. From this interface users are able to provide authentication credentials which are challenged using a RADIUS server.

On successful completion of the authentication process, the RADIUS server provides the SDN controller with group-based access policies in the form of VLAN references to be applied to the Port. These policies are sent using the standard IETF RADIUS attributes set out in 4.1.1.1.

On receipt of the RADIUS attributes, flow rules are built and sent to the controlled switches, placing the authenticated port in the appropriate VLAN.

This paper provides useful insight into passing RBAC policies from the RADIUS server to the controller, but is very limited in the policies able to be sent. As users are placed into groups and all access-control is achieved at the group level, this does not provide granular control over individual users.

Also once a user is placed on a VLAN, there are no mechanisms in place to restrict the type of traffic that a user can send, as would be when using ACLs.

4.1.1.1 RADIUS Attributes

IETF 64 - Tunnel Type

Defining the tunnel type that will be used by the port. In this case it is value 13 - VLAN

IETF 65 - Tunnel Medium Type

Attribute is used to define the transport medium that should be used. As VLANs are provided at layer 2, the value used will be 6 - 802 frames.

IETF 81 - Tunnel Private Group ID

Attribute is used to indicate the group id for the tunnelled session. In this case the VLAN ID that will be applied to the session on successful authentication.

4.1.2 SDN Enhanced Campus Network Authentication and Access Control System

This paper extends the previous investigation[30] by deploying the developed SDN NAC solution over a university campus and discussing the network requirements and design parameters in detail.

As with the previous paper, access control is provided through the use of VLANs exclusively, which limits the granularity of restrictions that can be put in place on the users session. However this solution experiences no issues with scaling during the deployment at the university and is proven to be a valid solution for medium to large group-based networks, such as universities. This solution provides a separation of staff and student traffic over the network.[31]

The paper also discusses how authentication requests are received by system. Using the OpenFlow packet-in mechanism, all authentication requests are passed to the Controller before they are forwarded to the RADIUS server; This is a security vulnerability as all unauthenticated users on controlled ports are able to send authentication requests. This could result in congestion on the control network and stop FlowMod messages from arriving at their destination switches.

4.2 FlowIdentity: Software Defined Network Access Control

[32] 'FlowIdentity: Software Defined Network Access Control' provide a combination of a custom SDN controller and NAC solution built using the Trema OpenFlow framework. FlowIdentity uses the HostAPd daemon for 802.1x functionality. This solution does not provide a web-based interface, instead requiring a Supplicant application be present on the connecting clients machine.

On connection a client sends an EAPoL packet to the multicast address

01:80:c2:00:00:03 to initiate the authentication process. As with the solution in 4.1.1 all authentication requests are sent to the Controller using the OpenFlow packet-in method.

Access control is achieved through the use of a stateful role-based firewall (rbFW) with which administrators are able to define RBAC policy that can be dynamically enforced on successful user authentication. This role-information is sent from the RADIUS server to the Controller using the RADIUS IETF attribute 18 Reply-Message.

IETF 18 - Reply-Message

Attribute is used to indicate text that might be displayed to the user using the RADIUS server. In this case, it is used to define RBAC policy for placing restrictions on authenticated sessions.

On receiving the role information, it is appended to a global identity table that is used by the rbFW for authorization and flow evaluation. This evaluation occurs as authenticated clients attempt to send traffic over the network.

The switch forwards all traffic without pre-defined flow rules to the Controller using the packet-in mechanism, provided to the controller with all unmatched packets. These packets can be evaluated against the rbFW rules for their associated client and a flow can be introduced to the relevant switches to allow the traffic.

If the traffic is denied, a deny rule is pushed down to the switch to avoid overwhelming the controller with denied flow requests.

This solution provides the ability for highly granular access control over the network but also requires a large amount of traffic to be sent to the controller. I believe that investigation needs to be performed assessing the risk of overwhelming the controller with flow requests as the network scales. Also mechanism used for passing information between the controller and the RADIUS server (IETF 18) is not compliant with intended purpose of the attribute.

4.3 Conclusion

Reviewing examples of NAC solutions has provided valuable insight to define sections in the current literature requiring further investigation. These have been listed below and will be analysed further in 5.

- An SDN NAC solution that does not use the OpenFlow packet-in messages to perform authentication
- Ability to filter traffic using fine-grained restriction methods, such as ACLs
- Ability to provide different restriction requirements for each user on authentication
- Support for multiple clients, both authenticated and unauthenticated, on a single managed port without creating security loopholes

Chapter 5

Problem Analysis

This chapter analyses the requirements of the project, contrasting the current state of literature to provide a clear list of project objectives and build success criteria for reference on conclusion of the project.

5.1 Requirements

In addition to the research requirements defined for this project, there are also system functionality requirements imposed through expectations of NAC solutions available in industry. Without these additional requirements the system would not be a viable NAC solution.

5.1.1 Industry Requirements

- Supporting a large number of EAP-methods, including EAP-TLS, PEAP, EAP-TTLS and EAP-MD5
- Allow authentication credentials to be stored centrally in an LDAP server
- Allow client access restrictions to be stored in LDAP
- Allow IoT devices and other devices incapable of EAP authentication to connect securely to a network
- Deny access to the network when a client is not authenticated

- Allow both controlled and open ports on the same switch
- Be light-weight, not hindering normal functionality of the network
- Use exclusively open standards to avoid vendor lock-in

5.2 Solution Design

For this project, I propose to provide an 802.1x Authenticator for the Faucet SDN Controller to provide network access control to a Faucet-driven network.

5.2.1 Architecture

The 802.1x Authenticator application will communicate with the Faucet SDN Controller through the plugin API and must receive EAP requests directly from the network, without them traversing the Faucet Controller.

This can be achieved by exposing the Authenticator to the data network, providing 802.1x as a virtualised network function (VNF). The port connecting the VNF to the data network is known as a network function virtualisation (NFV) port. Deployment of NFV ports have a number of benefits over using packet-in messages including better scalability independent of the Controller. This approach also removes the ability for unauthenticated users to send traffic directly to the Controller.

All EAPoL traffic from controlled ports must be directed to the closest NFV port using dedicated flow-rules while dropping all other traffic from these ports. Creating these flow-rules will require Faucet to know of NFV ports on startup to ensure traffic is successfully dropped from the required ports. [Refer to Figure 5.1]

On successful completion of an authentication attempt, the Authenticator must notify the Controller with the switch, port number and MAC address of the successfully authenticated client. The Authenticator may also provide a VLAN number and/or restriction policies to be applied on the session, if

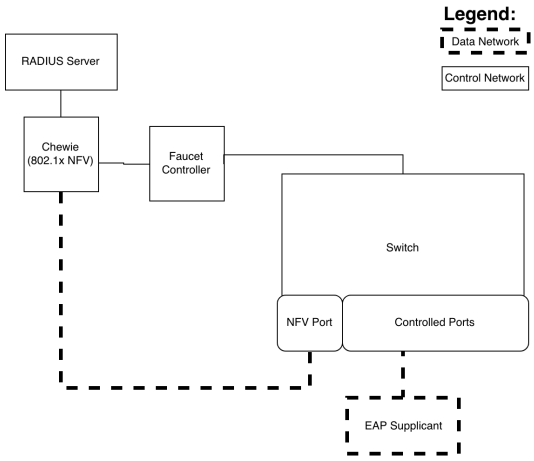

Figure 5.1: Chewie Setup Diagram

provided by the RADIUS server. The Controller must produce flow-rules in accordance with these requirements and deploy them to the relevant switches.

5.2.2 The Authentication Process

On a switch receiving an authentication request, the frame must be marked with a unique switch and port identification number before it is forwarded to the closest NFV port. This allows the Authenticator to know the origin port of the authentication request.

The Authenticator will perform packet-translation when it receives EAPoL frames. The Ethernet layer must be removed from the EAPoL request exposing the EAP message, which must populate the contents of a RADIUS request before it is forwarded to the RADIUS server. This process is performed in reverse for messages originating from the RADIUS server.

By forwarding EAP requests to the RADIUS server, the Authenticator is not required to perform authentication locally and is implicitly able to perform all EAP-methods supported by the RADIUS server. This allows the Authenticator to remain light-weight while providing all required functionality of a typical NAC system.

The ability to store all credential data in an LDAP service can also be leveraged from the RADIUS server as most common RADIUS servers, such as

FreeRadius[33], provide integration with LDAP services.

Performing EAP authentication on the RADIUS server requires the Authenticator maintain state of the RADIUS communication as it RADIUS challenge-responses require knowledge of previous RADIUS message-authenticator used in the communication. However, this overhead is far outweighed by the leveraged functionality provided by the RADIUS server and can be modelled using a state machine.

5.2.3 Authentication Without EAP

A large number of devices, such as printers and IoT devices, are unable to perform 802.1x authentication due to the EAP requirements. Most IoT devices are unable to perform supplicant duties and this hinders their ability to join secure networks.

The Authenticator for this project must provide a connection mechanism by which these devices are able to securely join a network without removing authentication requirements from the connection port.

Due to this requirement, the Authenticator will provide an alternative to EAP known as mac authentication bypass (MAB).

MAB provides secure authentication on behalf of a connecting client using their MAC address. This is achieved by sniffing MAC addresses from ingress DHCP requests from controlled ports. These MAC addresses can be used by the Authenticator to perform a RADIUS authentication process on behalf of the new client. [34]

As there is no standardised protocol for MAB leaving the format of the MAC credentials in the RADIUS server ambiguous. To perform successful authentication the format of MAC address in RADIUS must be known by the Authenticator. For this project I will be expecting MAB credentials will consist of MAC addresses stored as both user-name and password, without separating colons. This format may differ depending on the Authenticator and the vendor providing the devices.[35] [36]

On successful authentication, the Authenticator will receive all session restriction policies and VLAN information where applicable. These requirements will be provided to the Controller where the appropriate flows must be built and distributed to the relevant switches.

If the authentication process is not successful, all traffic from the client will be dropped.

As MAB authentication is performed by the Authenticator, the connecting client is not aware the authentication process has occurred. Also, the Authenticator requires the connecting device use DHCP. If a local static IP address is configured on the connecting devices the authentication process will not trigger and the port will remain closed.

MAB is less secure than an EAP-based authentication method as devices can be configured to spoof user-defined MAC addresses, but as there are many devices unable to perform EAP this is a secure alternative to providing an unprotected port for these devices.

As MAB is less secure that EAP, it will required to be explicitly applied to a controlled port.

Chapter 6

Implementation Details

This chapter discusses the implementation details of the developed project referencing technologies used throughout the development process, the code architecture and other implementation specific information related to the project.

6.1 Chewie

During the course of the project, I have worked towards developing Chewie; an 802.1x Authenticator written in Python3.
Chewie is written to function as a stand-alone application for processing 802.1x requests, but is primarily intended for use with the Faucet SDN controller.
Chewie's design philosophy is an extension to Faucet's; aiming for easily-legible code using small and reusable code segments in an attempt to provide a light-weight solution for network authentication. The architecture focuses primarily on leveraging functionality present in 802.1x applications, acting exclusively as an Authenticator; translating EAPoL to RADIUS and forwarding messages.

As ingress packets arrive through the supplicant-facing interface, they are translated into their respective RADIUS equivalents and forwarded through a RADIUS-facing interface for processing. This allows the RADIUS server to perform all identity and authentication checks whilst allowing Chewie to be aware of all changes in authentication state.

State is maintained in Chewie through the use of state machines, allow-

ing for detection of any deviation from the expected packet-flow, such as attempting to bypass the authentication by starting communications with an authentication successful response.

On a successful authentication event, Chewie notifies Faucet of the authentication status change through a custom plugin API. Client information and session restrictions are sent with notification messages allowing Faucet to add restrictions on client sessions. Likewise, Chewie is notified of any changes of state for managed ports through the same interface.

This custom Chewie/Faucet API required small amounts of additional code to be added to the main Faucet code-base, which has now been merged into the main open-source project[37]. The current version of Faucet (Version 1.5.9) has full Chewie support.

The Chewie code is provided as a separate repository[38] which is also open-source.

6.2 Functionality Overview

Chewie's functionality can be split into 4 major components listed below and explained in further detail in the following sections.

1. RADIUS communication

   All communication between Chewie and the RADIUS server is achieved through the use of a datagram socket. Using a network connection separate to the data network, Chewie and RADIUS are able to send messages using UDP, as per the RADIUS specification. [39]

2. Faucet communication

   Communication between Faucet and Chewie is achieved through a custom API written into Faucet. This API uses callback functions to notify Faucet of authentication events as they occur. Chewie is also notified on port-up and port-down events through function calls. [40]

3. Supplicant communication

   Communication between Chewie and the connecting Supplicant is achieved through the use of an NFV port connected to the data network. Faucet forwards relevant traffic originating from managed ports to the closest NFV port where it is consumed by one of two listening sockets. One socket is dedicated to receiving EAPoL requests and the other dedicated to receiving DHCP requests.

4. State Management Authentication state in Chewie is managed through the use of state machines. When a request arrives from a new client, a new state-machine is generated and attached to the authenticating client. The state machine is then used to manage the authentication process for that client. There are two types of state machine in Chewie, an EAP state machine used for managing the EAP authentication process and a MAB state machine used for performing MAB authentication.

6.3 Faucet Communication

6.3.1 Authentication Events

In Python3, functions are handled as first-class objects. This allows them to be stored and passed around like variables. Communication between Faucet and Chewie leverages this functionality, passing callback functions to Chewie on instantiation. These callback functions are used to notify Faucet of authentication events as they occur.
Definitions for the callback functions have been provided below, explaining expected parameter values and expected Faucet behaviour.

- **auth_handler**

  This callback function is used to notify Faucet of a successful authentication. The function requires information about the client, including the MAC address, port location and any restrictions to be placed on the

session such as a VLAN identifier and/or ACLs to be applied to the session. It is expected that this function will apply all relevant restrictions and open the provided port for communication with the secure network.

- **logoff_handler**

  This callback function is used to notify Faucet of a successful logoff event. The function requires information about the client, including the MAC address and port location. It is expected that this function remove the authenticated session for the given MAC address on the provided port. The client will no longer have the ability to communicate with the secure network.

- **failure_handler**

  This callback function is used to notify Faucet of an unsuccessful authentication attempt. The function requires information about the client, including the MAC address and port location. It is expected that this function remove an authenticated session for the given MAC address on the provided port, if present. The client will not have the ability to communicate with the secure network after the completion of this handler.

6.3.2 Port Status Events

On instantiation, Faucet retains a copy of the Chewie object and uses it to notify Chewie of any state changes of managed ports. This is done through calling methods provided by the Chewie public interface. The relevant methods are defined below, defining the expected parameter values and expected Chewie behaviour.

- **port_up**

  Triggered in the event of a port-up event, Faucet builds and deploys flows to direct all EAPoL traffic to the closest NFV port. If MAB is active on the port then DHCP traffic is also forwarded. Chewie is notified of the

change in status and must start sending pre-emptive identity requests looking for clients to authenticate.

- **port_down**

  Triggered in the event of a port-down event. Faucet destroys all flows for the port, removing any VLANs and ACLs associated with the port. Chewie is notified of the change in status, must stop any outstanding jobs and destroy all objects associated with the port.

In future development it is proposed that the callback system be removed and replaced with Google's Remote Procedure Call functionality, gRPC[41]. This will allow for better separation of concerns, creating a clear barrier between Chewie and Faucet and allow Chewie to scale without the need for Faucet. It will also allow Chewie to connect to SDN controllers that are not written in Python. [For further reading; Refer to Chapter 10]

6.4 RADIUS Communication

RADIUS communication is achieved through a RADIUS-facing interface separate from the data network. It has been separated to stop any clients attempting to communicate directly with the RADIUS server. Chewie provides options for all standard RADIUS packet-types for authentication. These include Access-Request, Access-Challenge, Access-Accept and Access-Reject packets which are defined in Section 2.2.

RADIUS Accounting [42] is not provided by mainline Chewie but is available in an experimental branch. [43]

6.5 Supplicant Communication

Communication with the connecting client is achieved through a 2-step process.

6.5.1 EAPoL

First, as the client connects to a managed port, Faucet deploys flows to route all EAPoL traffic to the closest NFV port.
Secondly, as frames are received by the switch they are marked with their origin. As no extra information can be attached to an EAPoL frame without introducing another protocol layer, the origin port and switch of the frame is marked in-place of the destination MAC address (DST MAC) and the frame is forwarded to the NFV port using flow actions. The origin marking consists of a 2-byte number representing the switch index in the local network and the port number within the switch. These numbers are padded with 0's to meet the required 48-bits for the DST MAC field.

As the DST MAC field is not that of Chewie's supplicant-facing NIC, the supplicant-facing socket must be placed into promiscuous mode for Chewie to receive the expected traffic.

Once the socket receives a frame, the origin is extracted from the DST MAC and recorded against the frame for future reference and the EAPoL frame is sent to the EAP state machines for processing.

6.5.2 MAB

If MAB is active on the port, DHCP traffic is also forwarded to the NFV port with the EAPoL traffic, the ETH DST value is overwritten and the DHCP packets are forwarded to the supplicant-facing interface using flow rules.

Once the MAB socket receives a DHCP packet, the origin is extracted from the DST MAC and recorded against the frame for future reference and the DHCP packet is sent to the MAB state machines for processing.

6.6 Managing Authentication State

The current authentication status of clients is maintained through a list of state machines (SM). As a new client attempts to authenticate, a state machine

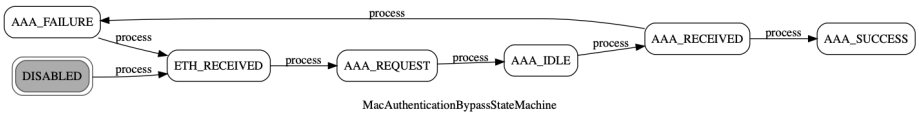

Figure 6.1: MAB State Machine

unique to their origin port and MAC address is created. This state machine follows the authentication process and ensure that there is no deviation from the expected packet-types. The state machines also regulate which requests may be forwarded through to the RADIUS server.

There are two types of state machines that are used to process user authentication; The MAB state machine (MAB SM), which is used to perform all MAB requests and the EAP state machine, which is used to process all EAPoL authentication requests.

6.6.1 MAB State Machine

As a DHCP request is received on the NFV port, a MAB state machine is created and passed the DHCP request. The MAB state machine uses the DHCP request to generate a RADIUS Access-Request containing the MAC address of the client as the following attributes:

- **IETF 1 - User-Name**

  Stored in plain text with no separators

- **IETF 2 - User-Password**

  Stored as ciphertext. All separators are removed and the MAC address is encrypted using the MD5-based algorithm defined in the RADIUS RFC[44]

- **IETF 31 - Calling-Station-ID**

  Stored in plain text, separating digit-pairs using a hyphen ('-')

On receiving the request, the state machine transitions from the 'DISABLED' state into the 'ETH_RECEIVED' state for processing. The MAC

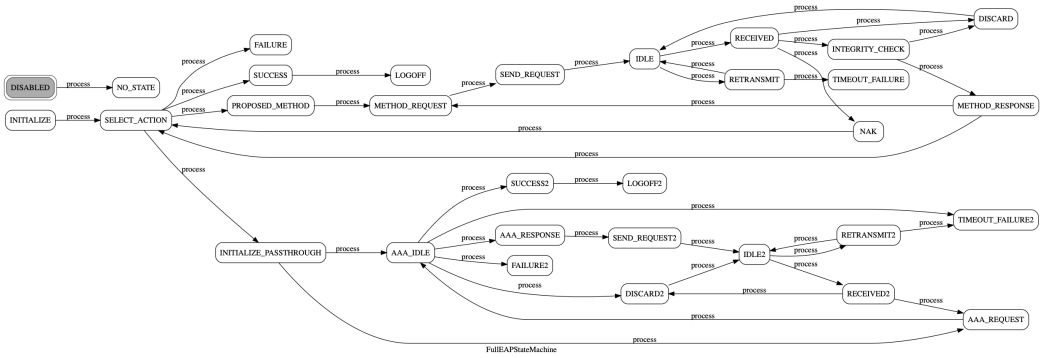

Figure 6.2: EAP State Machine

address is extracted from the packet and store in the SM. A RADIUS request is generated according to the above attributes and send to the RADIUS server, transitioning the SM through the 'AAA_REQUEST' state to the 'AAA_IDLE' state, where it waits for a response from the RADIUS server.

On receiving a response from the RADIUS server, the state transitions to 'AAA_RECEIVED'.

If the response is successful the SM transitions to 'AAA_SUCCESS' and the 'auth_handler' callback function is triggered opening the session.

Likewise, if the response is unsuccessful the SM transitions to 'AAA_FAILURE' and the 'failure_handler' is triggered, forcing the session closed.

For more information about state transitions, refer to Figure 6.1.

6.6.2 EAP State Machine

The state machine defined in RFC 4137[45] is used as the basis of the EAP state machine (EAP SM) implementation. Initially deviating from the defined state machine through the introduction of the 'LOGOFF' and 'LOGOFF2' states, we have extended upon the original state machine design for increased code clarity and reuse. Our EAP SM implementation can be split into 3 functional segments listed below:

- **Method Negotiation**

  Method negotiation is performed to ensure that both parties communi-

cate using the same EAP method.[13]. Chewie notifies the Supplicant that EAP pass-through is required and all following authentication requirements are performed by the authentication server.

- **Local Authentication**

  Local authentication is provided by the EAP state machine in accordance with RFC 4137[45] but is not implemented nor supported.

- **RADIUS Authentication**

  RADIUS authentication is the intended verification technique and involves the Supplicant performing an authentication process with the RADIUS server. All EAP-based RADIUS authentication methods supported by Chewie require a challenge to be performed by the Supplicant. The challenge is performed in accordance with the EAP method chosen and provided back to the RADIUS server to prove identity.

As a request is received by the EAP state machine, the SM transitions to the 'SELECT_ACTION' state, where the 'Method Negotiation' segment is initiated. The available EAP authentication methods are proposed by Chewie to the supplicant and an EAP method is agreed upon. On successful completion of 'Method Negotiation', the SM is returned to the 'SELECT_ACTION' state to initiate the second phase of authentication.

If the outcome is 'Local Authentication', the SM is transitioned into either the 'SUCCESS' or 'FAILURE' state depending on the result of a local look-up for the provided credentials. If the credentials provided successful authentication the SM transitions to the 'SUCCESS' state and 'auth_handler' is called, if the credentials fail to provide a success authentication the SM transitions to the 'FAILURE' state and the 'failure_handler‘ is called.

If the outcome is RADIUS authentication, the SM is transitions into the 'INITIALIZE_PASSTHROUGH' state and the RADIUS authentication process begins.

RADIUS authentication is discussed further in the following section as it pertains to both the EAP State Machine and the MAB State Machine. For further information about the EAP SM state transitions, refer to Figure: 6.2.

6.7 RADIUS Authentication

Successful RADIUS authentication can be achieved through providing user-credentials in the initiating RADIUS Access-Request packet, using RADIUS attributes. This method is used by Chewie's MAB implementation.

EAP authentication is achieved through the result of a generated challenge provided to the Supplicant. The result of the challenge is provided back to the RADIUS server where it is used to complete the authentication process.

Both authentication methods are initiated using a RADIUS Access-Request sent from Chewie to the RADIUS server.

6.7.1 AAA Authentication

The MAB state machine performs RADIUS authentication providing required information as RADIUS attributes. On receiving the first RADIUS Access-Request, the RADIUS server is provided with enough information to verify a clients identity and provide an authentication response. For MAB this is achieved through the use of the User-Name and User-Password attributes and is the authentication-type used for MAB implementation in Chewie. For further information on the packet flow required for challenge-free authentication refer to 6.3

6.7.2 EAP Authentication

Challenged authentication is required to perform an EAP-based authentication. Once the EAP-method negotiation has completed, an EAP Identity Response is sent from the Supplicant. The Authenticator (Chewie) encapsulates the EAP Response in a RADIUS Access-Request packet and forwards

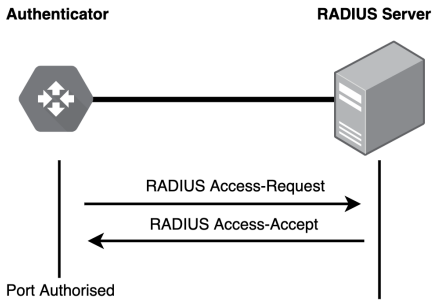

Figure 6.3: Challenge-Free RADIUS Packet Flow

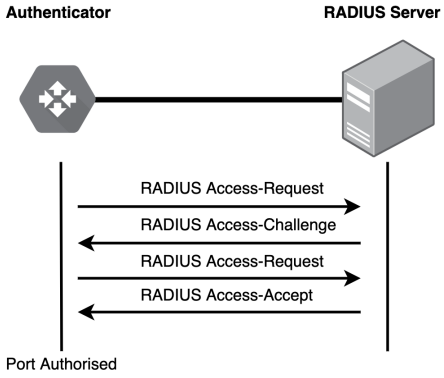

Figure 6.4: EAP Challenge RADIUS Packet Flow

the request to the RADIUS server for processing.

On receiving the RADIUS encapsulated EAP packet, an EAP-challenge is generated and sent back to the Supplicant. The challenge is performed in accordance with the selected EAP method and the result is sent to the RADIUS server completing the EAP authentication process. A RADIUS Access-Accept or RADIUS Access-Reject is then provided, if the authentication is successful or unsuccessful respectively. For further information on the packet flow for challenged authentication, refer to 6.4

6.8 Configuration

Faucet is configured through the use of a YAML file. Configuration is read by Faucet and passed to Chewie on start-up. For a port to be controlled by Chewie, it must be marked as a 802.1x port in the Faucet configura-

tion file. Along with controlled ports, the configuration file contains values for connecting with the RADIUS server. These values include authentication credentials, the RADIUS server IP address, RADIUS-facing interface, the Supplicant-facing interface and NFV port locations. All related preferences are provided to Chewie as arguments during initialisation.

Once a port is marked as a controlled port, flow rules are deployed for forwarding EAPoL traffic to the NFV port.

NFV ports are defined in the Faucet configuration file, defining the port number and switch id that Chewie is connected to on the data-network. All authentication requests from managed ports are routed to the NFV port for processing. [40]

6.9 Applying Session Restrictions (RBAC)

On successful authentication, the RADIUS server notifies Chewie using a RADIUS Access-Accept packet, attaching any VLAN and RBAC session requirements as RADIUS attributes. If present, Chewie passes these requirements to Faucet with an authentication status change event. Faucet builds all flow rules in accordance with the provided restrictions and distributes flows to relevant switches.

VLAN requirements are passed to Chewie using the Tunnel-Type (IETF 64), Tunnel-Medium-Type (IETF 65) and Tunnel-Private-Group-ID (IETF 81) RADIUS attributes.

RBAC requirements are passed to Chewie by defining the ID of an ACL to be applied on the authenticated port. The attribute used for transporting the ACL ID is Filter-Id (IETF 11).[40]

All used ACLs must be known by Faucet on startup and defined in the Faucet configuration file. This is required as Faucet allocates matching resources on switches at start-up and must be restarted to defined new matching fields.

Chapter 7

Personal Contribution

When starting development with Chewie, the project was in an early proof-of-concept state. EAP-MD5 was the only authentication method available and the code was written with all values publicly accessible. There was also no clear separation between logical components with RADIUS logic tightly coupled with EAP logic.

This created issues when adding functionality as components were tightly coupled code-reuse was minimal. The EAP state machine contained most of the EAP and RADIUS logic but its internal state variables were changed throughout the rest of the project making it difficult to add MAB logic as the whole project was dependant upon the EAP pipeline.

Chewie did successfully block traffic on unauthenticated ports, notifying Faucet of changes in authentication state but did not provide any further restriction or VLAN information. It also provided a number of useful abstractions for producing RADIUS packets.

7.1 Code Refactor and Documentation

During the first section of development, I spent most of the time refactoring code, splitting object into maintainable segments and attempting to remove as much tight-coupling as possible. There were a number of unit-tests already in place for Chewie, making this process easier, providing a quick way to ensure

functionality was not lost during development.

I also developed scripts for Sphinx[46], a documentation building application, to build diagrams and provide documentation for the project. This made it easier to familiarise myself with the code-base and hopefully help with on-boarding future developers.

On successful completion of the TravisCI[47] continuous integration testing suite the documentation is published to ReadTheDocs [48], an online documentation hosting platform.

To assist with development, I also developed a Docker-environment for integration testing. As Chewie's functionality expects a multi-device environment for there developed a need for an easily created virtual testing environments that could be used for integration tests. I developed this using Docker[49] and OpenVSwitch[50].

Along with the Docker environment I wrote a number of python-driven integration tests used for automatically testing the integration of Chewie with Faucet.

The Docker environment built and setup a virtual switch, configuring ports for the Faucet, Chewie, a RADIUS server and a connecting Supplicant on a controlled port.

1. FreeRADIUS[51]; as the RADIUS server; was configured according to credential-files provided in the Chewie repository and connected to the virtual switch.

2. wpa_supplicant[52]; as the supplicant; configured according to the supplicant folder in the Chewie repository and specific tests that were being attempted. On successful startup of the network the supplicant would automatically attempt to authenticate with the network using the EAP-method defined in the test.

3. Chewie; as the EAP Authenticator; configured by Faucet according to the Faucet configuration.

4. Faucet; as the SDN controller; The Faucet configuration was defined in a configuration file that was generated using a template by the specific tests being run. If no tests were being run, the developer is dropped into a bash environment after network set-up is complete.

7.2 Granular Traffic Filtering

Once the testing framework was in place, I developed a mechanism for Faucet to apply fine-grained filter policies after the start-up process. As Faucet allocates the matching resources required to perform it's filtering based on the configuration file at startup, setting filter options was only available on start-up.

By defining ACLs in the configuration file and adding a small amount of code to the ACL.py file, I was able to mark two ACLs for use with 802.1x. One was to be used for authenticated clients and the other to be used with unauthenticated ports. Resources for these ACLs are allocated on start-up for all switches containing controlled ports and the unauthenticated ACL is applied to controlled ports on port-up.

On successful authentication, flow-rules are built for the authenticated ACL and applied to the authenticated port matching on the MAC address of the successful client. This was achieved through using Faucet's ACL Manager, converting the defined ACLs into OpenFlow[25] flow-rules, applying the MAC as extra matching criteria.

By setting the priority of the authenticated flow-rules at a higher level than the unauthenticated rules and including the MAC address as matching criteria, I was able to allow for multiple clients to attach to a single port.

The port remains closed for all unauthenticated users but is opened for individual users on authentication. This provides access control even if a non-SDN switch is attached to a controlled port, minimising the ability to piggyback on an authenticated session.

7.3 Mac Authentication Bypass

During the development process with Chewie, a number of feature requests for mac authentication bypass (MAB) were placed by Faucet-users. Without the ability to perform MAC address authentication Chewie was unable to be deployed in IoT environments hindering the adoption of Chewie by Faucet users.

I built a new state machine for Chewie to process incoming DHCP requests and trigger a MAB authentication. Attempting to extract common functionality between MAB and EAPoL authentication, I build the abstract state machines from which both MAB SM and EAP SM inherit.

7.3.1 Abstract State Machine

The goal of the Abstract State Machine (ASM) was to extract out the RADIUS communication process into a separate state machine and apply this to both child objects, but due to the high-coupling inside the EAP state machine, I was unable to extract all of RADIUS logic, instead using the A SM to extend the state machines, providing a template for state machine development.

As the Chewie state machines were built using a state machine library; Transitions[53] by PyTransitions; I was able to leverage the library to build state diagrams which were added to the documentation. The A SM was useful in enforcing the separation of core-pipeline state transition logic from transitions into failure states. This allowed for building cleaner diagrams, as all states return to disabled if a port-down event occurs, making the unfiltered diagrams difficult to read. Diagram generation has been added to documentation build process and diagrams are now included in the ReadTheDocs[48] documentation.

7.3.2 MAB State Machine

The MAB State Machine (MAB SM) receives all DHCP requests from the NFV port and performs authentication on behalf of the connecting client.

A second socket has been created on the NFV port for consuming DHCP broadcast requests and forwarding these to the MAB SM, similar to the implementation of the EAPoL socket for the EAP State Machine.

On receiving these packets, the MAB SM is required to complete an authentication process with the RADIUS server. I opted for using a non-challenged authentication process, proving all information in RADIUS attributes allowing for a 2-packet authentication process. This required the addition of the RADIUS User-Password attribute and encryption algorithm[44] as it was not currently supported by Chewie.

By adding the User-Password attribute I was able to keep the MAB SM implementation simple with it only requiring 7 states.

Along with the MAB SM, Faucet also required minor extensions; I added an addition configuration option for allowing MAB to be applied to controlled ports. This allowed me to add flow-rules to ports with MAB applied that forward DHCP requests along with EAPoL requests to the NFV port.

I developed unit-tests and integration tests for MAB which were passing successfully, but while testing Faucet integration I noticed inconsistent behaviour. I was able to perform successful MAB authentication but an unintended side-effect occurred where all ports on the same VLAN as the NFV port triggered the MAB authentication process.

For further information about this flooding issue refer to Section 8.1.

7.4 Downloadable ACL (dACL)

The primary focus of this thesis was the implementation of a NAC solution that can apply RBAC policies in an SDN environment. These policies were to be stored with the credentials in an authentication server, such as a RADIUS

or LDAP server.

My initial design of this aimed to provide individual ACL rules to the authenticator similar to Cisco's dACL functionality[54]. Unlike Cisco's implementation, the individual filter rules would consist of a standard format, similar to the HPE implementation[20]. These control rules would then deployed as flow-rules by Faucet on successful authentication.

After familiarising myself with the Faucet code-base, I started investigating a way to pass filter rules between Chewie and the RADIUS server. The RADIUS attribute NAS-Filter-Rule [55] is not a vendor-specific attribute and appears to be intended for transport of individual filter rules. I added the attribute to Chewie and passed the individual rules to Faucet on successful authentication.

Attempting to have Faucet build flow-rules based on the provided filter rules was harder than expected. Discussing the problems with other Faucet developers, the errors being experienced were due to limitations with Faucets pre-allocation of switch matching resources. I was attempting to allocate new matching criteria on switches without them being allocated, which both deviated from the light-weight requirement of Faucets design principles and caused the switches to return OF Errors.

A simple solution would be to pre-allocate a large number of matching resources on all switches containing controlled ports allowing for a large number of different types filter rules to be applied, but does not satisfy the light-weight requirements of Chewie and Faucet and is not a scalable solution.

Instead this approach was abandoned and an alternative approach was investigated.

7.5 Credential-based ACLs

After failing to find a way to successfully build and apply individual flow-rules based on RADIUS information, I was forced to redesign the way RBAC

policies will be applied to authenticated sessions. I had developed ability to apply pre-defined ACLs on authentication status changes, which could be used for applying user-specific ACLs, provided the ACLs were defined in the Faucet configuration files.

I expanded upon the current approach and defined a way for Chewie to pass ACL identifiers to Faucet on authentication. Faucet could then use the provided ACL identifier to apply the appropriate flow-rules to the new session.

This required the definition of another RADIUS attribute in Chewie, RADIUS Filter-ID. Using the Filter-ID attribute, I was able pass ACL identifiers to Chewie and have them applied on successful authentication.

Also, as this was developed using RADIUS attributes, the provided filter identifiers were able to be provided with both MAB and EAPoL authentication. Allowing credential-based session restrictions for all supported authentication methods, provided the ACLs were defined in the Faucet configuration files.

Chapter 8

Implementation Difficulties

During my development towards Chewie there were a number of unexpected events that hindered my ability to accomplish my original design goals. In this chapter I have listed some of difficulties experienced throughout the development process in the hope that it may be beneficial for future developers.

8.1 NFV - VLAN Flooding issues

Chewie receives DHCP requests from controlled ports for MAB functionality. These packets are sent over the existing data network. This network is controlled by Faucet and Faucet provisions a number of flow rules to redirect DHCP request to the NFV ports attached to Chewie.

By default, Faucet places all ports in the same VLAN, placing all ports in the same flood table. When frames are sent to broadcast or multicast addresses they are forwarded to all listening devices on the same VLAN.

As a DHCP request is a broadcast request, it is forwarded to all ports on the same VLAN including the NFV port. This side-effect was overlooked during the design process and created issues when more than one client is connected to a switch.

As clients connect to non-802.1x controlled ports they send out DHCP requests. These requests were being received by Chewie and initiating the MAB authentication request process for all connecting devices.

This issue was resolved by moving the NFV port into a separate VLAN from the data network. As all inter-VLAN routing of the packets is done through Faucet's defined flows, we are able to forward packets inter-VLAN without a router. This removed the chance of broadcast packets being received by Chewie unintentionally.

8.2 Stated Radius Connections

The initial design for the EAP-RADIUS packet translation process was intended to be stateless.

On receiving an EAPoL frame, the Ethernet layer would be removed and the EAP message placed in a RADIUS packet. Likewise, on receiving a RADIUS packet, the EAP message would be extracted and forwarded to the Supplicant.

A number of issues stopped this from being implementable, the main of which was as RADIUS packets are returned to the Authenticator there is no clear mechanism for resolving the intended client, without maintaining the state of communication between a the Supplicant and RADIUS Server.

Resolving the intended client has been achieved by storing RADIUS message id numbers and associating them with Supplicant origin information. This allows Chewie perform a lookup to find the required information to send EAPoL frames to the Supplicant.

8.3 Faucet - Allocation of Resources

As part of Faucet's start-up process, switch matching resources are allocated according to the network configuration defined in the Faucet configuration file.

All flows must not match on criteria that is not reserved during this period. If an attempt is made to match on extra criteria the action performed by the switch is ambiguous. Switches may respond with an OF Error or not respond and drop the new flow-rule. This limitation is a design feature of Faucet and

is a crucial component to its lightweight design philosophy.

The initial design for Chewie's fine-grained NAC was to accept individual filter rules using the NAS-Filter-Rule RADIUS attribute and deploy flow-rules accordingly. These filter rules could be grouped together to create ACLs and associated with groups and users through LDAP relationships.

This approach was not feasible with the current Faucet implementation, and instead available ACLs must be defined in the Faucet configuration. The ACLs that will be used to limit the available network resources to a controlled port are defined in the RADIUS sever and shared to Chewie using the (IETF Filter ID) attribute. This is forwarded to Faucet and the flow rules are built to apply the appropriate limitations.

Chapter 9

Evaluation

This chapter provides an evaluation of the achievements from this paper, reflecting on the goals presented in Chapter 1 and discussing if they have been accomplished.

9.1 Goals Overview

The primary focus of this project was the development of a NAC solution for an SDN environment. The validation goals discussed in Chapter 1 have been listed below.

The goals of this project:

- Use open and standard authentication protocols to allow interoperability between vendors whilst avoiding vendor lock-in.
- Apply role-based access control policies based on authentication details for allocating access to network resources.
- Light-weight and not overly hinder normal management of the network.

9.2 Open and Standard Authentication Protocols

During development, all vendor-specific protocols and attributes were removed as options for use in the NAC system. By using RADIUS, EAP and MAB for authentication Chewie will have the widest support for clients, avoiding vendor lock-in.

The RBAC solution was also designed with open-standards as a primary focus. No vendor-specific RADIUS attributes, such as Cisco's AV-Pair attributes were used for sending information to Chewie. Instead open RADIUS attributes were used to provide session filter and VLAN information.

This goal has been achieved as no proprietary protocols, attributes or close-source software is required by this project to provide NAC functionality.

9.3 Apply Role-Based Access Control Policies

Although Chewie is able to apply session filters based on a provided credentials, it is limited to predefined restrictions only. All ACLs used for session restrictions must be defined in the Faucet configuration file at startup. For the purposes of this research project, restriction policies are able to be applied on a per-user basis and are dependent on supplied credentials however there extensions can be applied to the current implementation and are discussed further in Chapter 10.

9.4 Light-Weight Management

Chewie is managed and configured in same way as Faucet, using the Faucet.yaml configuration file. Chewie's logging is also achieved in a similar fashion to Faucet, using the same logging manager and activity monitor.

Chewie provides thorough logs producing log entries for all state changes, allowing for quick debugging on errors.

Chewie also provides per-user VLAN allocation, allowing network administrators to move freely around the network whilst providing access to any control VLANs without manual configuration.

As Faucet is an SDN controller, switches do not require individual management and no additional setup is required for adding Chewie to the network.

As Chewie does not require the addition of any management tools and attaches to all management techniques provided by Faucet, Chewie does not hinder the light-weight management style of Faucet.

Chapter 10

Further Research

This chapter discusses possible extensions to the completed research in this project. I have listed a number of areas that future developers could expand upon to provide a more featured NAC solution.

10.1 Downloadable ACLs

Further investigation can be done into Faucet's resource allocation system and implementing a mechanism for adding additional matching criteria to a switch flow-table after the initial setup. This would allow for dynamic allocation of filter-rules based on user credentials using the NAS-Filter-Rule RADIUS attribute.

Another mechanism for adding additional matching criteria could be adding new ACL requirements to the Faucet configuration file and forcing a hard-restart, however this may create issues with the current implementation of Chewie.

Further investigation is required to state that this is not possible.

10.2 Encrypted Channels

802.11i, a specification for wireless key exchange provides a key-sharing mechanism that can be used alongside 802.1x. This is used in wireless technologies

such as WPA to provide an encrypted tunnel between the wireless AP and the client, limiting the ability of malicious users to piggy-back on authenticated sessions and removes the ability for traffic to be sniffed from the network.

With Chewie, if a hub attached to a controlled port a malicious actor is able to sniff all traffic that is sent over the hub. By implementing a system similar to the 802.11i protocol, sessions could be encrypted removing the ability for the malicious user from understanding the traffic. This will also limit the ability for hubs to provide a mechanism to circumvent the NAC filtering, stopping the ability to piggy-back on authenticated sessions through MAC address spoofing.

A proposed design for this could be through using Faucet's co-processor functionality, exposing an NFV port that receives encrypted traffic and outputs decrypted traffic to a separate switch for forwarding through the rest of the network, separating all decrypted traffic from the encrypted data network.

Chapter 11

Conclusion

This paper has investigated the feasibility of producing a network access control system for a Faucet-driven software defined network. It has presented a system for providing network authentication and control and has successfully achieved the initial investigation goals. The provided solution supports the deployment of fine-grained network access control without the requiring unreasonable maintenance overhead. The solution also provides a mechanism for authenticating devices typically unable to authenticate to secure networks through the use of mac authentication bypass by performing authentication on behalf of devices that are unable to connect to secure networks using EAP methods.

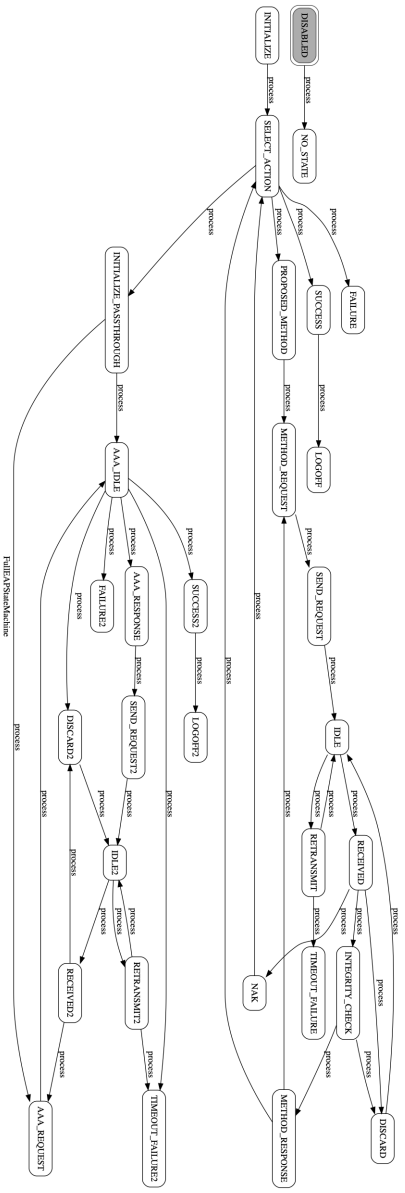

Figure .1: EAP State Machine

.1 EAP State Machine

Provided above is a high-definition copy of the EAP State Machine diagram as reference.

.2 Source Code

All code produced in this paper is available open source at www.github.com/faucetsdn/chewie

References

[1] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 475–488. [Online]. Available: http://doi.acm.org/10.1145/2663716.2663755

[2] A. Kakisim, M. Nar, N. Carkaci, and I. Sogukpinar, *Analysis and Evaluation of Dynamic Feature-Based Malware Detection Methods: 11th International Conference, SecITC 2018, Bucharest, Romania, November 89, 2018, Revised Selected Papers*, 01 2019, pp. 247–258.

[3] A. Hezam, D. Konstantas, and M. Mahyoub, "A comprehensive iot attacks survey based on a building-blocked reference mode," *International Journal of Advanced Computer Science and Applications*, vol. Vol. 9, 04 2018.

[4] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, Nov 2011, pp. 4490–4494.

[5] M. ONeill, "Insecurity by design: Today's iot device security problem," *Engineering*, vol. 2, pp. 48–49, 03 2016.

[6] M. Patton, E. Gross, R. Chinn, S. Forbis, L. Walker, and H. Chen, "Uninvited connections: A study of vulnerable devices on the internet of things

(iot)," in *2014 IEEE Joint Intelligence and Security Informatics Conference*, Sep. 2014, pp. 232–235.

[7] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[8] Fortinet. The evolution of network access control (nac). [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/white-papers/wp-evolution-of-network-access-control.pdf

[9] IANA. Ieee 802 numbers. [Online]. Available: https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml#IEEE

[10] Cisco. (2019) What is network access control? [Online]. Available: https://www.cisco.com/c/en/us/products/security/what-is-network-access-control-nac.html

[11] J. Networks. Understanding psk authentication. [Online]. Available: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director3.0/topics/concept/wireless-wpa-psk-authentication.html

[12] I. E. T. Force. Internet engineering task force. [Online]. Available: https://www.ietf.org/

[13] N. W. Group. Rfc 3748 - extensible authentication protocol. [Online]. Available: https://tools.ietf.org/html/rfc3748

[14] A. Networks. About aaa. [Online]. Available: https://www.arubanetworks.com/techdocs/ClearPass/Aruba_DeployGd_HTML/Content/802.1XAuthentication/About_AAA.htm

[15] N. W. Group. Rfc 2904 - aaa authorization framework. [Online]. Available: https://tools.ietf.org/html/rfc2904

[16] I. A. N. Authority. (2019) Radius types. [Online]. Available: https://www.iana.org/assignments/radius-types/radius-types.xhtml#radius-types-14

[17] N. W. Group. Rfc 2904 - ieee 802.1x remote authentication dial in user service (radius) usage guidelines. [Online]. Available: https://tools.ietf.org/html/rfc3580

[18] J. Networks. Standard and vendor-specific radius attributes. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/radius-std-attributes-vsas-support.html

[19] C. Networks. Radius centralized filter management. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_usr_rad/configuration/xe-16/sec-usr-rad-xe-16-book/sec-rad-central-filt.html

[20] H. P. Enterprise. Radius filter-id. [Online]. Available: https://techhub.hpe.com/eginfolib/networking/docs/switches/K-KA-KB/15-18/5998-8150_access_security_guide/content/v32013119.html

[21] Cisco. Cisco ios: Authentication authorization and accounting configuration guide. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_usr_aaa/configuration/15-e/sec-usr-aaa-15-e-book/sec-usr-mac-auth-bypass.pdf

[22] E. B. C. World. What to look for in an sdn controller. [Online]. Available: https://www.computerworld.com.au/article/533230/what_look_an_sdn_controller

[23] C. Networks. Software defined networking. [Online]. Available: https://www.cisco.com/c/en_au/solutions/software-defined-networking/overview.html

[24] TechTarget. What is control plane? [Online]. Available: https://searchnetworking.techtarget.com/definition/control-plane-CP

[25] O. N. Foundation. A high-speed routing engine for software defined network. [Online]. Available: https://www.opennetworking.org/software-defined-standards/specifications/

[26] F. Developers. (2019) Faucet configuration. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/faucet/latest/faucet.pdf

[27] S. Gao, S. Shimizu, S. Okamoto, and N. Yamanaka, "A high-speed routing engine for software defined network," 2012.

[28] F. Developers, "Faucet architecture diagram," 2019, [Online; accessed October 10, 2019]. [Online]. Available: https://faucet.readthedocs.io/en/latest/_images/faucet-architecture.svg

[29] B. Developers. Beka codebase. [Online]. Available: https://github.com/faucetsdn/beka

[30] V. Dangovas and F. Kuliesius, "Sdn-driven authentication and access control system," 06 2014.

[31] F. Kuliesius and V. Dangovas, "Sdn enhanced campus network authentication and access control system," 07 2016, pp. 894–899.

[32] S. T. Yakasai and C. G. Guy, "Flowidentity: Software-defined network access control," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 115–120.

[33] T. F. S. Project. Freeradius - modules. [Online]. Available: https://freeradius.org/modules/

[34] J. Networks. Mac radius authentication. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/mac-radius-authentication-switching-devices.html

[35] ——. Static mac bypass. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/static-mac-bypass-mac-radius-authentication.html

[36] H. P. Enterprise. Configuring mac authentication. [Online]. Available: https://techhub.hpe.com/eginfolib/networking/docs/switches/WB/15-18/5998-8152_wb_2920_asg/content/ch03s06.html

[37] F. Developers. Faucet codebase. [Online]. Available: https://github.com/faucetsdn/faucet

[38] C. Developers. Chewie codebase. [Online]. Available: https://github.com/faucetsdn/chewie

[39] N. W. Group. Radius - why udp? [Online]. Available: https://tools.ietf.org/html/rfc2865#section-2.4

[40] F. Developers. (2019) Faucet configuration. [Online]. Available: https://faucet.readthedocs.io/en/latest/configuration.html#x-dp

[41] gRPC Community. grpc. [Online]. Available: https://grpc.io/

[42] N. W. Group. Radius accounting. [Online]. Available: https://tools.ietf.org/html/rfc2866

[43] C. Developers. Chewie - radius accounting branch. [Online]. Available: https://github.com/MichaelWasher/chewie/tree/radius_accounting

[44] N. W. Group. Radius - user-password. [Online]. Available: https://tools.ietf.org/html/rfc2865#section-5.2

[45] ——. State machines for extensible authentication protocol (eap) peer and authenticator. [Online]. Available: https://tools.ietf.org/html/rfc4137

[46] S. Authors. Sphinx - python documentation generator. [Online]. Available: http://www.sphinx-doc.org/en/master/

[47] T. Developers. Travisci. [Online]. Available: https://travis-ci.org/

[48] C. Developers. Chewie documentation. [Online]. Available: https://chewie.readthedocs.io/en/latest/

[49] D. Inc. Docker. [Online]. Available: https://www.docker.com

[50] T. L. Foundation. (2016) Openvswitch. [Online]. Available: https://www.openvswitch.org/

[51] F. Developers. Freeradius. [Online]. Available: https://freeradius.org/

[52] J. Malinen. Wpa_supplicant. [Online]. Available: https://linux.die.net/man/8/wpa_supplicant

[53] T. Yarkoni. Transitions. [Online]. Available: https://github.com/pytransitions/transitions

[54] C. Networks. 802.1x dacl, per-user acl, filter-id, and device tracking behavior. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/lan-switching/8021x/119374-technote-dacl-00.html

[55] N. W. Group. Radius filter rule attribute. [Online]. Available: https://tools.ietf.org/html/rfc4849

[56] R. Toghraee, *Learning OpenDaylight*. Packt Publishing, May, 2017.

[57] M. Team, "Mininet: An instant virtual network on your laptop (or other pc)," *Google Scholar*, 2012.

[58] V. V. D. Solutions. Eapol - ex. [Online]. Available: https://www.vocal.com/secure-communication/eapol-extensible-authentication-protocol-over-lan/

[59] S. Microsystems. Sun's xacml implementation. [Online]. Available: http://sunxacml.sourceforge.net

[60] Oracle. Oracle entitlements server. [Online]. Available: https://www.oracle.com/technetwork/middleware/oes/overview/index.html

[61] T. A. S. Foundation. Openaz project. [Online]. Available: http://incubator.apache.org/projects/openaz.html

[62] A. Hesham, F. Sardis, S. Wong, T. Mahmoodi, and M. Tatipamula, "A simplified network access control design and implementation for m2m communication using sdn," in *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, March 2017, pp. 1–5.

[63] C. Networks. Openflow agent for nexus 3000 and 9000 series. [Online]. Available: https://www.overleaf.com/project/5d2c46c792569a05969931b1

[64] ——. Mac authentication bypass deployment guide. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/identity-based-networking-services/config_guide_c17-663759.html