NIST Technical Note 2066

OpenFMB Proof of Concept Implementation Research

Michael Bartock
Rebecca Herold

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2066

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Technical Note 2066

OpenFMB Proof of Concept Implementation Research

Michael Bartock
Computer Security Division
Information Technology Laboratory

Rebecca Herold
The Privacy Professor Consultancy, LLC
Des Moines, IA

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2066

July 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Technical Note 2066
Natl. Inst. Stand. Technol. Tech. Note 2066, 20 pages (July 2020)
CODEN: NTNOEF

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2066

Abstract

There is a smart grid messaging framework known as an Open Field Message Bus (OpenFMB), which was ratified by the North American Energy Standards Board (NAESB) in March 2016 and has been released as NAESB RMQ.26, *Open Field Message Bus (OpenFMB) Model Business Practices*. OpenFMB focuses on describing a publish-and-subscribe model of communication for smart grid devices to enable efficient communication of data. Subsequent analysis of OpenFMB and its possible implementations will focus on threat analyses of the framework, implementations, cybersecurity recommendations, and a proof of concept implementation of OpenFMB. The OpenFMB framework is being explored as a way to implement publish-subscribe communications between smart grid network nodes. This paper focuses on the cybersecurity risk implications of deployments and a proof of concept implementation of OpenFMB.

Key words

NAESB; NISTIR 7628; OpenFMB; proof of concept implementation; publish-subscribe communications; Smart Grid cybersecurity.

Table of Contents

List of Tables

1. Introduction

Smart Grid architecture development is one of the primary work efforts of the Grid Modernization Initiative.[1] There is currently a Smart Grid messaging framework has been developed by the Smart Grid Interoperability Panel (SGIP).[2] This messaging framework is known as OpenFMB,[3] which was ratified by the North American Energy Standards Board (NAESB) in March 2016 and has been released as NAESB RMQ.26, *Open Field Message Bus (OpenFMB) Model Business Practices*. In April 2017, SGIP merged with the Smart Electric Power Alliance (SEPA) and its work continues under the SEPA name.

The OpenFMB framework is being explored as a way to implement publish-subscribe communications between smart grid network nodes. This paper focuses on the cybersecurity risk implications of deployments and implementations of OpenFMB. The technical requirements for OpenFMB must be understood before successfully enhancing the security of OpenFMB. The objective of this paper is to provide a technical understanding of actual implementations of OpenFMB to identify existing and potential cybersecurity threats.

By understanding the risks associated with OpenFMB, actions can be taken to mitigate those risks and provide more continuous electricity service. If such risks are not mitigated, incidents such as service outages could occur without the knowledge of electric grid administrators, resulting in long-term power outages and potential associated harm to the customers using the electric services.

The following were the two primary goals for the OpenFMB Proof of Concept (PoC) testing that took place at the NIST Gaithersburg campus during a three-week period in the first quarter of 2018:

- Identify technical security controls that can be applied to an OpenFMB installation, and the underlying system it is running on. The application of these security controls will protect against unauthorized access to data being exchanged within the distribution portion of the power grid, to data in storage, or to access and modify the settings of the devices themselves.
- Conduct basic performance testing to understand the overhead of enabling security features within the OpenFMB PoC environment.

The scope of this paper includes the distribution portion of the electricity grid and the associated cybersecurity risks that are not currently covered within the NAESB OpenFMB document. The documents of consideration for meeting these objectives included the following:

[1] See more about this US Department of Energy government initiative at https://www.energy.gov/under-secretary-science-and-energy/grid-modernization-initiative.
[2] See more about the SGIP at https://www.nist.gov/programs-projects/smart-grid-national-coordination/smart-grid-interoperability-panel-sgip.
[3] See more about OpenFMB at https://openfmb.github.io/.

- The North American Energy Standards Board Retail Gas Quadrant Retail Electric Quadrant Model Business Practices Open Field Message Bus (OpenFMB) document, Version 3.1, from March 31, 2016 [1]
- NISTIR 7628, Revision 1, *Guidelines for Smart Grid Cybersecurity Volume 1 – Smart Grid Cybersecurity Strategy, Architecture, and High-Level Requirements*. Created by The Smart Grid Interoperability Panel – Smart Grid Cybersecurity Committee. Published in September 2014 [2]
- NIST SP 800-53, Revision 4, *Security and Privacy Controls for Federal Information Systems and Organizations*. Published: April 2013. Updated 1/22/2015 [3]
- Message Queuing Telemetry Transport (MQTT) Version 3.1.1 Plus Errata 01: Organization for the Advancement of Structured Information Standards (OASIS) Standard Incorporating Approved Errata 01. December 10, 2015 [4]
- NIST Framework for Improving Critical Infrastructure Cybersecurity Version 1.0. February 12, 2014 [5]

Those implementing OpenFMB may need to supplement the security controls specified in NAESB RMQ.26, to protect the entire system OpenFMB is running on. These compensating security controls can be configured within the operating systems of devices running OpenFMB, and by using MQTT cybersecurity controls that are not explicitly indicated by OpenFMB but need to be used to fill critical cybersecurity gaps.

1.1. Identified Cybersecurity Risks

The risks to OpenFMB implementations, including the device operating system and associated publish-subscribe communication applications, are within the scope of this report. As a high-level summary, the risks include the following:

- Unauthorized access to devices and data within the computing environment can occur through the operating system (OS), which can lead to service interruptions, outages, or equipment damage resulting from settings changes.
- Single-factor ID/password authentication is weak and subject to being defeated by unauthorized individuals.
- Admin accounts may be shut out of systems, logs altered or changed, and outages may not be identified as a result of allowing too many concurrent sessions.
- Inappropriate settings for privileged access could lead to inappropriate access to data, logs, settings, devices, and other resources and result in malicious actions, service outages, and damage to devices.
- Unauthorized access to data, device settings, and OS settings could occur without setting appropriate access controls and establishing appropriate settings for session locks.

- When access control changes are not logged, troubleshooting, identifying unauthorized activities, and determining a history of access control changes can become much more time-consuming, difficult, and—in some situations—not possible at all.
- Inadequate security controls can result in data collection, access, changes or sharing that could lead to malicious use, malicious cybersecurity attacks, service interruptions and outages, and possibly even device damage.

1.2. Security Control Configuration

The details of the mitigations for the associated risks to the device operating system and associated publish-subscribe communication applications are within the scope of this report. The following provide high-level descriptions of mitigations to the identified risks in the previous section:

- Use certificates for authentication and encryption.
- Limit the number of concurrent sessions and access to devices and computing environment resources to only those specific administrator accounts that need such access to support their administrative responsibilities.
- Ensure that settings within the devices, operating systems, and applications are established to allow accesses and capabilities to only those specified as being accountable and responsible for maintaining the computing environment and associated devices.
- Establish session lock settings based on the risk levels of the associated network environment.
- Log all access attempts and activities, and establish procedures to regularly review the logs.
- Implement a comprehensive cybersecurity information security program and controls for all devices, systems, data, and applications used within the distribution network environment.

2. Proof of Concept (PoC) Testing Plan Overview

This section of the report will detail and discuss the lab environment setup that was used to implement and test the OpenFMB simulation. It will review the hardware and software components, as well as discuss the plan for implementing and testing security features with the proof of concept implementation.

2.1. Simulation Devices and Software

The following equipment and software were used to perform proof of concept testing of the OpenFMB specification within simulation equipment for electric grid distribution devices:

- Raspberry Pi: Five Raspberry Pi 2 Model B units
- Ubuntu Linux Operating System – Image: ubuntu-16.04.3-preinstalled-server-armhf+raspi2, with the following software:
  - Mosquitto MQTT broker and client
  - OpenSSL
  - Java Development Kit (JDK)
  - Java Runtime Environment (JRE)
- Netgear Switch. One: NETGEAR GS724Tv4 24-Port Gigabit Smart Managed Pro Switch
- Laptop: Windows 10 with software tools for monitoring network traffic and connecting to Linux systems (WireShark, putty, WinSCP, etc.)

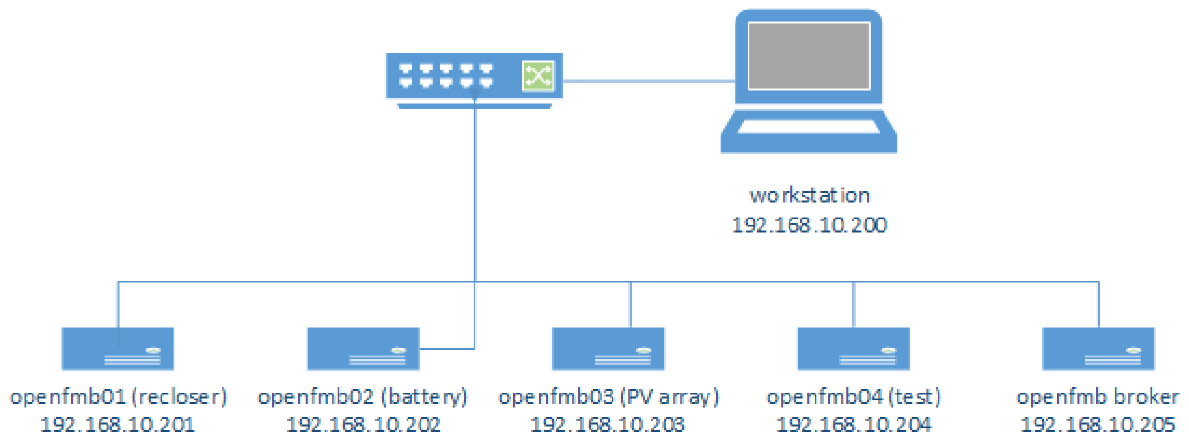

Figure 1. Network Diagram of Lab Setup

2.2. OpenFMB Code

Smart grid device simulation code used for a DistribuTech demo by Duke Energy was obtained from open-source GitHub repositories. At the time of implementation, the following code sets were installed, used for PoC testing, and available from:

- OpenFMB simulators code from https://github.com/openfmb/openfmb-simulators
- OpenFMB common MQTT https://github.com/openfmb/openfmb-common-mqtt

- OpenFMB load-publisher https://github.com/openfmb/openfmb-loadpublisher

The OpenFMB simulation code, as written in Java and each Raspberry Pi in the environment, was deployed to simulate one specific component of a smart grid environment. The smart device modules were installed to simulate a recloser, battery, and solar photovoltaic array, as well as simulate load generation for each class of device. While the open-source simulation code included a Java implementation of the MQTT client, this environment used the Mosquitto library within the host OS. The MQTT broker was installed on a separate Raspberry Pi rather than the simulation devices and was an instance of the Mosquitto broker within the host OS. The devices simulate a scenario in which a microgrid, consisting of a solar photovoltaic array and battery storage, connects to a generation grid. While the recloser is closed, the solar array charges the battery, and when the recloser opens, the microgrid is powered by the stored energy in the battery as well as the solar array. When the recloser returns to closed, the microgrid regains its feed from the generation grid, and the solar array again charges the battery storage. All of the microgrid activity and measurement data are transmitted via publish-subscribe communications and formatted according to the OpenFMB specification.

2.3. Testing Strategy

In order to conduct network performance tests on the simulated OpenFMB nodes, the operating system was first installed and baselined for security configurations. The MQTT Mosquitto software was then installed and configured for various security controls. Finally, the open-source OpenFMB simulation code for a recloser, battery, and photovoltaic (PV) array was installed and configured to use the OpenFMB broker running on its own node. The following structured approach to test the OpenFMB code was established and followed.

2.3.1. Establish baselines on the Raspberry Pis

Out of the box and customizable configurations were examined to determine the capabilities that the operating system underlying OpenFMB supports. A set of common configurations was applied as a baseline to each of the Raspberry Pis (e.g., secure shell (SSH) connection limits, user inactivity timeout, etc.). Applying specific settings to the operating system configurations verified that both:

1) The controls that OpenFMB can possibly be configured to support using the operating system it runs on.
2) The controls can be used to apply security controls to OpenFMB.

The following major activities were planned and performed:

1) Install Ubuntu Linux on the Raspberry Pis.
2) Determine the security controls and capabilities that exist within Ubuntu Linux related to specific security concerns.
3) Provision X.509 certificates to the Raspberry Pis to be used for device authentication and TLS encryption.

2.3.2. Enable MQTT on the Raspberry Pi Simulators

SGIP/SEPA selected MQTT as the "best practice" publish-subscribe protocol.[4] To identify cybersecurity risks, the 2016 DistribuTech demo code[5] was used to determine if similar results were identified and to document any additional findings beyond those reported for that demo.

During this phase of the PoC testing, the Mosquitto[6] implementation was used for the simulation to follow the SGIP/SEPA selection and to parallel the DistribuTech demo. The purpose of this testing was to identify security controls within MQTT, along with identifying and documenting security risks to provide an example of what it takes to secure MQTT.

This testing was performed to verify two major facts:

1) The controls that OpenFMB can possibly be configured to support using the operating system it runs on

2) The controls that can be used to apply security controls to OpenFMB.

The following major activities were planned and performed:

a. Install MQTT on the Raspberry Pis.
b. Determine the security controls and capabilities that exist within MQTT related to specific security concerns.
c. Enable user-based authentication for the MQTT client to the MQTT broker.
d. Enable Transport Layer Security (TLS) security between the MQTT client and the MQTT broker.
e. Enable certificate-based authentication between the MQTT client and the MQTT broker.

2.3.3. Run OpenFMB on the Raspberry Pi simulators

Once the operating system and MQTT software were installed on the Raspberry Pis, the next step was to install and configure the OpenFMB simulation code. This included installing a distinct component on each node. Figure 1 shows that each Raspberry Pi was configured to run a different part of the OpenFMB simulation, which included a recloser, battery, PV array, and an MQTT broker. The following process was followed to configure the lab environment and conduct the network performance tests.

Testing was performed to determine two major facts:

a. The controls that OpenFMB can possibly be configured to support using the operating system it runs on
b. The controls from the operating system (Ubuntu) and MQTT that can be used to apply security controls to OpenFMB.

The following major activities were planned and performed:

a. Install OpenFMB files on the Raspberry Pis
b. Determine the security controls and capabilities of the OpenFMB files that are related to the identified security concerns with NISTIR 7628, Revision 1.

[4] See more discussion about this at the OpenFMB GitHub: https://github.com/openfmb/turnkey-dtech-demo-2016/wiki/MQTT-Security.
[5] See more information about this at the OpenFMB GitHub: https://github.com/openfmb/openfmb-common-mqtt.
[6] Mosquitto is an open-source MQTT broker that can be found at https://www.eclipse.org/mosquitto/download/.

c. Configure the OpenFMB simulation options to use the MQTT broker stand-alone instance in the lab.
d. Measure network performance to determine the overhead of applying security controls on the devices. The device security controls applied will use either no encryption or TLS 1.2 encryption with 2048-bit keys and the following device authentication methods:
    i. None
    ii. Device name- and password-based
    iii. X.509 certificate-based

3. Security Control Implementation

The devices configured in the lab consisted of three main components that were investigated for applying security controls to: the operating system, the MQTT software, and the OpenFMB simulation code. Since the OpenFMB simulation code relied on the underlying MQTT software to handle transport security, the only security control implemented for the OpenFMB software involved configuring it to use the MQTT software. The following subsections describe the security considerations and capabilities that were applied in the lab environment at the operating system and MQTT software levels.

3.1. Operating System Security Configurations

Devices need proper access controls applied to them to ensure that only authorized actions can be performed on them. The following are high-level risk and associated mitigations to be considered when applying access control policies. See Appendix A for details about the associated configuration settings, risks, and associated mitigations.

There are risks and mitigations to be considered for applying access controls at the operating system level.

High-level Risks:

- Unauthorized access to data, device settings, and OS settings could occur without setting appropriate access controls and establishing appropriate settings for session locks.
- When access control changes are not logged, troubleshooting, identifying unauthorized activities, and determining a history of access control changes can become much more time-consuming, difficult, and—in some situations—not possible at all.
- Unauthorized access could occur in active sessions that are not being monitored.
- Allowing too many password attempts could be vulnerable to brute force attacks and allow unauthorized access.
- Not using session timeouts can allow for unauthorized access through active or unsupervised sessions.
- Allowing access from all IP addresses and for all times increases the risk of unauthorized access, distributed denial-of-service (DDOS) attacks, and other cybersecurity incidents that could put data at risk, allow for destructive device settings changes, and cause service outages and disruptions.

The following list consists of the security capabilities that were applied to the operating systems in the lab environment. The capabilities were implemented by applying various settings within the configuration files for the system. See Appendix A for details about the associated configuration settings, risks, and associated mitigations.

Security Capabilities:

- Restrict the groups, IP addresses, and concurrent sessions for root accounts.

- Establish timeout and session lock settings appropriate for the associated computing environment.
- Limit root capabilities to the fewest accounts possible.
- Enable logging, and establish procedures to frequently monitor and review logs and to respond to activities that reveal or could indicate inappropriate activities or problems within the computing environment.
- Establish monitoring procedures for the auth.log file.
- Limit the number of password attempts, and establish settings for automatic session timeouts that are appropriate for the associated network risk environment.
- Limit the IP addresses that can access devices to the specific ones that will be responsible for supporting the devices.
- Limit access for all accounts to only the times when access is necessary for performing administrative activities.
- Log all access attempts and activities, and establish procedures to regularly review the logs.

3.2. MQTT Application Controls

The MQTT application is responsible for transmitting OpenFMB-formatted data to the smart grid devices. Security settings can be applied to the MQTT broker to control access to information that is exchanged between devices. Additionally, the MQTT broker can specify if any encryption needs to be used with transmitting data. The lab environment went beyond the default configuration for MQTT to ensure that security features were enabled. The goal of enabling these security features was to enable device identification and authentication, as well as encryption for data transmitted on the network. Listed below are the security capabilities that were enabled in the lab and the high-level findings that were observed once they were enforced.

Security Capabilities:

- Use TLS authentication and encryption.
- Use username_as_clientID within the mosquitto.conf file to allow for trouble-shooting and forensic investigations.
- Restrict the number of concurrent users allowed to access a topic to one or the lowest number feasible for the associated computing environment.
- Assign a unique username to each actor given access to the system.
- Consider using access control lists to limit access to messages.
- Give access to the $SYS tree to only a very limited number of those with administrator responsibilities, and periodically audit who has these capabilities.
- Do not leave user mosquitto blank; otherwise, it will run as root.
- Turn on TLS encryption for the transmission of messages.

High-level Findings:

- Turning on TLS encrypted the transmission of messages.

- Specify where to log activities using the mosquitto_passwd utility. Use TLS to keep the password encrypted, and do not run as root to mitigate catastrophic accidental or malicious changes to the IDs/passwords.
- Time established within the systems and devices must be synchronized to allow for the successful use of self-signed and SSL certificates.
- The implementation used a 2048-bit key with TLSv1.2 for the certificates.
- Authentication with OpenFMB worked using certificates.
- If defined, only clients that have a clientid with a prefix that matches clientid_prefixes will be allowed to connect to the broker. For example, setting "secure-" here would mean that a "secure-client" could connect but another with the clientid "mqtt" could not. By default, all client IDs are valid. Use clientid_prefixes to further restrict access control.
- TLS confidentiality and authentication were successfully tested on broker, solar array, and battery simulations to the MQTT Broker.

4. Results and Findings

NISTIR 7628, Revision 1, *Guidelines for Smart Grid Cybersecurity*, "presents an analytical framework that organizations can use to develop effective cybersecurity strategies tailored to their particular combinations of smart grid-related characteristics, risks, and vulnerabilities" [2]. A goal of the OpenFMB is to incorporate the use of the GridWise Architecture Council stack to establish a standard to help ensure that cybersecurity is "maintained through all levels of interoperability, from automated control through business transactions."[7]

This project involved performing proof of concept tests to determine the performance impacts of applying the identified NISTIR 7628, Revision 1, security requirements within a proof of concept OpenFMB implementation. Those using OpenFMB are able to supplement the controls available through it with the control capabilities. This can achieved by applying specific configurations in the operating systems of the devices where OpenFMB is running, and by using MQTT cybersecurity controls that are not explicitly indicated by OpenFMB.

Upon enabling device authentication and TLS encryption, the implementation was tested to baseline the performance of varying levels of applying these controls. The possible options for device authentication are None, Device Name and Password, or Certificate Authentication; the possible options for TLS encryption are None or TLS 1.2. Since publish-subscribe communications require devices to publish messages to a broker and receive messages from the broker for topics it is subscribed to, these baseline tests were performed for both scenarios. In the first scenario, the time between a device publishing a message and an acknowledgment from the broker was measured. In the second scenario, the time between a broker sending a published message to a subscriber and an acknowledgement from the subscriber was measured. The following tables show the measurement results.

Table 1. Network Performance of Publisher to Broker with Varying Security Configurations (seconds)

Authentication / Encryption	**None**	**TLS 1.2**
None	.003 242	.109 013
Device Name & Password	.003216	.108 034
X.509 Certificate	N/A	.186 208

[7] "GridWise Interoperability Context-Setting Framework." Created by the GridWise Architecture Council . March, 2008. Page 32.

Table 2. Network Performance of Broker to Subscriber with Varying Security Configurations (seconds)

Authentication / Encryption	None	TLS 1.2
None	.002 855	.108 071
Device Name & Password	.003 223	.108 458
X.509 Certificate	N/A	.201 232

The results of the network performance measurements show that enabling TLS 1.2 encryption and X.509 certificate authentication add the most overhead for MQTT communication. This performance measurement was taken when the Mosquitto clients were establishing new connections with the Mosquitto broker; hence, a TLS handshake had to take place. Other performance baselining measurements for a different MQTT software implementation (HiveMQ) have shown that longer term overhead is negligible since the TLS session can be kept alive, and the handshake overhead will not be felt for every message sent.[8] Further testing can be done within this proof of concept implementation to explore this idea. Additionally, this proof of concept implementation can be supplemented with real-world smart grid devices that implement OpenFMB and the publish-subscribe protocols to obtain more real results instead of using simulators.

[8] See https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/.

References

[1] North American Energy Standards Board (2016) Open Field Message Bus (OpenFMB), Version 3.1. (NAESB, Houston, TX), NAESB Retail Markets Quadrant Task Force RMQ.26 Standard. https://openfmb.ucaiug.org/Pages/Overview.aspx

[2] The Smart Grid Interoperability Panel–Smart Grid Cybersecurity Committee (2014) Guidelines for Smart Grid Cybersecurity. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 7628, Rev. 1. https://doi.org/10.6028/NIST.IR.7628r1

[3] Joint Task Force Transformation Initiative (2014) Assessing Security and Privacy Controls in Federal Information Systems and Organizations: Building Effective Assessment Plans. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53A, Rev. 4, Includes updates as of December 18, 2014. https://doi.org/10.6028/NIST.SP.800-53Ar4

[4] OASIS (2015) MQTT Version 3.1.1 Plus Errata 01 (OASIS) OASIS Standard Incorporating Approved Errata 01, December 10, 2015. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[5] National Institute of Standards and Technology (2018) Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Cybersecurity White Paper, Includes updates as of April 16, 2018. https://doi.org/10.6028/NIST.CSWP.04162018

Appendix A: Acronyms

DDOS	Distributed Denial-of-Service
GWAC	GridWise Architecture Council
JDK	Java Development Kit
JRE	Java Runtime Environment
MQTT	Message Queueing Telemetry Transport
NAESB	North American Energy Standards Board
OpenFMB	Open Field Message Bus
OS	Operating System
PoC	Proof of Concept
SGIP	Smart Grid Interoperability Panel
TLS	Transport Layer Security

Appendix B: Ubuntu Security Configurations

This section provides samples of the operating system security configurations that were applied.

Session Controls Testing

The following examples are how the implementation established access controls for sessions, devices, and user accounts.

1. Ubuntu can restrict SSH access to the host by inbound IP address and enforce password complexity.
    a. SSH allows for SSH to the IP address of the other active sessions.
    b. The SSH daemon can be configured in SSHD_config to use different authentication methods based on the client address/hostname with the following setting:
        i. **DenyUsers:** This setting is set by specifying users to IP addresses (e.g. *userID@IPAddress).*
2. The Ubuntu system can be configured to require a complex password in the SSHD_config file.
    a. The implementation turned on PasswordAuthentication.
    b. Minimum password length can be set.

The following examples are how the systems were configured to control user access attempts and session activity.

1. The configuration file at /etc/profiles:
    a. **TMOUT**: Set the value for the number of seconds for the bash session to automatically end after user inactivity.
2. In the /etc/ssh/sshd_config file, the following values to the following:
    a. **ClientAliveInterval**: The number of seconds for the bash session to automatically end after user inactivity.
    b. **ClientAliveCountMax**: Set to the number of keepalive messages the SSH session will terminate after receiving no response. This setting needs to be used in conjunction with **TCPKeepAlive** set to "yes" so that the SSH service sends the keepalive messages.
3. The /etc/ssh/sshd_config file configures how long a user has to complete an authentication attempt:
    a. **LoginGraceTime**: The number of seconds a user must complete an authentication attempt before being disconnected.
    b. See NIST Special Publication 800-53 (Rev. 4), AC-12 SESSION TERMINATION recommendations for timeouts based on risk values.[9]

[9] See https://nvd.nist.gov/800-53/Rev4/control/AC-12.