Intrusion Detection and Prevention Systems for Use on Remote Servers

Leon Anderson
2204060

BSc Hons Ethical Hacking, 2025

School of Design and Informatics
Abertay University

Table of Contents

Contents

Table of Figures

Table of Tables

Acknowledgements

A special thank you to my supervisor Jaime O'Hare for supporting me throughout the project and giving me guidance about how to do this academic piece.

And a special thank you to my girlfriend Eden for supporting me and unravelling the mystery of how to use "a" and "an". It really shouldn't have taken me this long to learn this but thank you anyways.

Abstract

Intrusion detection and prevention systems (IDPS) are a well-known example of technical security controls and are widely used by companies to help protect systems inside their network. However, most companies use IDPS developed by third-party companies, which can present an issue when the software is unstable and becomes unavailable; this is where custom-made intrusion detection systems can become critical for a company's security. This project aims to research, develop, and evaluate the effectiveness of a system capable of the detection and remediation of threats on remote servers.

The intrusion detection system was developed using an agile development methodology and focused on using log analysis to detect intrusions within the system and on detecting remote SSH connections and file/folder monitoring. The system created was a host-based system with a central management system responsible for receiving alerts via AWS and SCP protocols. These alerts were then parsed into a Slack bot to be sent to the correct channels. The IDS was then tested and evaluated using manual functional testing, which would reveal that the folder monitoring feature was not functional, though the SSH and file monitoring were.

Finally, the system focused on detection and alerting, with automated remediation not being implemented due to the design options such as using log analysis making it not real time. Due to this, preventative actions could not be taken.

Abbreviations, Symbols and Notation

IDS- Intrusion detection system

IDPS- Intrusion detection and prevention system

IPS- Intrusion prevention system

ML- Machine learning

LSM- Linux security module

PoLP- Principle of least privilege

AWS- Amazon Web Services

VM- Virtual Machine

FTP- File Transfer Protocol

Chapter 1 Introduction

As cybercrime against organisations is becoming increasingly common with more sophisticated payloads and techniques used, so is it that intrusion detection and prevention systems play a critical role in the upkeep of security in a company. To clarify, an Intrusion detection and prevention system, or IDS/IDPS, is a piece of software or hardware that is specifically designed to detect and subsequently stop malicious activity from occurring on a host system or network. Historically, this has been done in many ways, such as blocking the IP of potential attackers as a direct method or more indirect methods, such as informing a network administrator or security expert that an attack has occurred. (Fuchsberger, 2005)

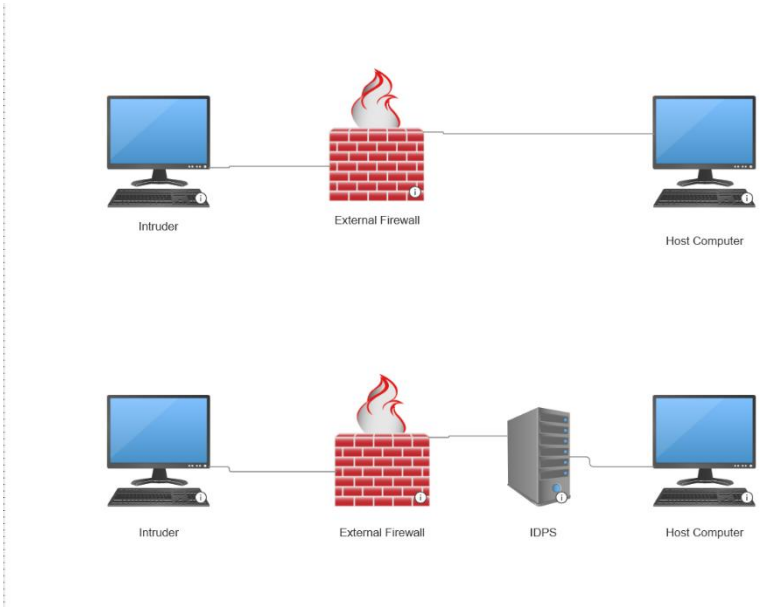

Figure 1- Example of Defence in Depth With and Without IDPS

These systems are not meant to be the first line of defence in hindering cybercrime; rather, the systems are a tool in what is known as the ‘defence in depth’ approach to security. (Mosteiro-Sanchez *et al.*, 2020) The 'defence in depth' approach to security entails multiple layers of security, which are, in a sense, stacked on top of each other to make the breach of

security more difficult for the attacker, should they break through the first line of defence. This would stop them from going deeper, denying them access to sensitive data. For example, one would have multiple locks on a door so that if an intruder were to break one to gain access to the house, the intruder would not be able to enter through the doorway without breaking through each subsequent lock. With the multiple locks, in this case, more attempts at getting through each lock only increase the likelihood of being caught. In cyber security, this allows the system to have more opportunities to catch the attackers through each layer of security. An example of defence in depth can be seen in Figure 1 above.

For this project, the goal is to create a system that watches and learns the remote server's regular routine and identifies when an operation is out of the ordinary before working to resolve the issue. This technique is commonly known as anomaly-based detection and works using systems, such as machine learning, to determine what can be seen as normal behaviour and what is not.

Cybercrime in the UK has remained an issue for medium to larger businesses, with 70% of medium businesses and 74% of larger businesses suffering a cyber attack, where 90% of attacks are phishing attempts. (United Kingdom Government, 2024) This is when a criminal attempts to extract credentials and information from the company's employees to gain access to the company network through a legitimate account. Since the attacker could access the company network, they can access remote servers and information without setting off intrusion alarms by pretending to be a legitimate user. Bypassing firewall rules and other technical controls. This is why it is vital for companies to use intrusion detection and prevention systems.

Although commercial IDPS systems are available to companies, there are risks to using premade solutions. A significant challenge with current commercial IDPS solutions is their lack of customizability, stability issues, and potential for unnecessary costs. An example of this would be the

CrowdStrike Falcon Platform. CrowdStrike is an American cyber security solutions company that offers multiple tiers of security for a monthly subscription. It is estimated that 45% of Fortune 100 companies rely on CrowdStrike. (George, 2024). On July 19th 2024, an unstable update was released for CrowdStrike's IDS solution. The update caused a technical malfunction across all systems that had the software installed. Causing widespread outages worldwide. Microsoft estimates that 8.5 million Windows devices were affected (Tidy, 2024), and the damages caused by this outage have an estimated cost of between 4-6 billion USD. (George, 2024). This problem with commercial IDPS is that far too little attention has been paid to how IDPS can be made more stable, customisable and cost efficient. One solution for this would be for a company to produce their own custom IDPS.

1.1 Research aims and objectives

This project aims to research, develop, and evaluate the effectiveness of a system capable of the detection and remediation of threats on remote servers. Three key objectives have been identified for the project's aim.

- The first objective is that a method of detecting anomalies from observing the server will be researched and implemented.

- The second is that appropriate and automated remediation and alerts for detected threats will be implemented, allowing the system to run without user interaction.

- The third objective will be to evaluate the effectiveness and accuracy of the project, reporting on how successful the project was.

1.2 Chapter Overview

This paper will be split into five sections, each detailing a stage in the research, development, and evaluation of an IDPS system. The first

section of the paper will be a literature review. This section will research the core principles of Linux security and techniques associated with Linux security, as well as methods, existing tools, and past research. The paper will then move on to the methodology of the project, where a MoSCoW analysis will be done to further identify objectives, and then the development and testing of the IDS will take place. Following this, a section detailing the results from testing will be completed, and then a section discussing the project will be done. This section will be used to evaluate what went well throughout the project and what did not. Detailing possible remediations for the issues that occurred during the project. Finally, there will be a conclusion determining if the project was a success and recapping the processes taken throughout the project.

Chapter 2 Literature Review

2.1 Introduction

This chapter of this project aims to provide a comprehensive literature review of the current state of security monitoring of security monitoring on the Linux operating system. The chapter will be split into two sections, with each focusing on its own corner of security monitoring on Linux. The two sections of this literature review will be the Core Principles of Linux Security and Techniques associated with Linux Security. These two sections will help establish the project's baseline and support the IDPS development.

2.2 Core Principles of Linux Security

When thinking about cyber security, some common principles are used by security teams worldwide. This literature review section aims to explain and discuss some of these core principles and how they can be applied to the Linux operating system to enhance its security.

2.2.1 Principle of Least Privilege

The Principle of Least Privilege (PoLP) is a security practice that entails that all users be granted the minimum number of permissions possible in order for them to complete their work. (Jero *et al.*, 2021). If a user has a security breach on their account, the damages caused by this would be minimised as the attacker would have access to limited resources.

PoLP is one of the most common methods used in cyber security and spans multiple operating systems; in Ubuntu OS, when the operating system is first installed, it comes with three account types: Root, Administrator and Basic. (Motiee, Hawkey and Beznosov, 2010). The administrator account is the most used account by users. The Administrator account allows users to perform tasks such as changing the time and using the operating system at an upper level; however, it will not

allow the user to perform system tasks or Root tasks without a root account through the sudo command. (Kenlon, 2022).

On Windows OS, The PoLP is much easier to enforce, thanks to Active Directory. The active directory is a feature in Windows that allows administrators to create accounts and control what each account can do and see. The authors (Desmond *et al.*, 2008) describe an active directory as "a central repository that can be globally distributed" " which is true as the active directory is easy to manage and deploy enterprise-wide for system administrators. However, ease of use comes at a cost. The lack of security.

In an article published by Microsoft, the author discusses how PoLP is often incorrectly enforced and that large groups of users have “broad and deep“ privileges. (Foulds, 2023). The article discusses how administrators often take the easiest road when creating new user accounts instead of assigning a new group of users to their own groups with their own PoLP enforcement. The administrators often assign them to an existing group with privileges above what they must complete for their jobs. This highlights an issue with the PoLP in which administrators often incorrectly implement it and can be a low-hanging fruit in security.

2.2.2 Protection and Detection

The Detection and Protection principle states that half of security is detection and half of security is protection. Detection is the idea that when a cyber attack occurs, the victim can know it is happening. Meanwhile, protection is the idea that when an attack occurs, appropriate steps have been taken beforehand to minimise the damage. The authors Sewak et al discuss this in their paper on Deep reinforcement learning in advanced cyber security threat detection and protection. (Sewak, Sahay and Rathore, 2023) During the paper, the authors discuss how machine learning can be used to help improve the detection of cyber attacks. They propose that deep reinforcement learning can be used to improve the

detection rate for cyber attacks and implement one such model into an IDPS system. During their research, they discovered that when the detection rate of the IDPS went up due to ML, the protection rate also went up. Highlighting the protection and detection principle well.

2.2 Techniques Associated with Linux Security

Many techniques are commonly used to monitor Linux operating systems. This section of the literature review aims to discuss some of the more popular and common techniques.

2.2.1 Log analysis

Logs are one of the easiest to implement features when adding security to a software product. Logs are useful for many things throughout the development cycle and production of a product and can be simply added using simple print commands in code; however, although this may help within the development cycle of software with debugging, this method of logging is volatile and disappears when the software is finished executing. Due to this, developers realised early on that software needs to be logged into files for cold storage, as the authors Buckley and Siewiorek discussed in their paper in 1995. (Buckley and Siewiorek, 1995) The authors discuss how log analysis can be used for fault management and trend analysis;. However, this paper focuses on using log analysis for system monitoring rather than security monitoring, showing the capabilities of log analysis.

The Linux system was released four years before this paper, and an in-built in-depth logging system was implemented into it. The "/var/log" directory in the kernel is widely used by the system and by developers to log events that have taken place on the OS. This is a commonly monitored folder for security and is used in many forensic systems, as discussed by Dusane and Sujantha (2021). In the paper, the authors discuss how the /var/log folder contains system logs (syslog), Daemon logs (daemon.log), and authorisation logs (auth.log). The author discusses different methods

of log analysis, such as using historical data and comparing logs, but talks about the challenges associated with this method of forensic analysis. They scrutinise this analysis method as log analysis can take a long time to complete, and automation tools can have difficulties analysing these files due to logging not having an industry standard. This scrutiny of log analysis is an opinion shared by the authors (Svacina *et al.*, 2020), who further dissect the issues of log analysis. Their paper discusses how log analysis is a very effective strategy for security monitoring; however, the authors describe it as a "post mortem" method of analysis. Which can only be done after an attack has taken place. This highlights a key issue with log analysis for use in an IDPS: it cannot be a guaranteed way of catching attackers in the process of an attack.

2.2.3 Network Monitoring

Networking in Linux has always been highly accessible, with the openness of the Linux kernel, allowing system administrators and security professionals to keep a close eye on network traffic flowing in and out of the device. (Sánchez, Alvarado-Nava and Martínez, 2012), This has become a key staple in the cyber security industry as most researchers consider network traffic to be a critical part of security monitoring.

In a paper that focuses on integrating IDS with network monitoring, the authors Kumar, Angral and Sharma (2014) support this theory by describing how network monitoring is a vital part of real time detection of threats. The authors describe network monitoring use in IDS as the ability to identify malicious packets and drop it from the network. Umar, Anral and Sharma discuss in their paper why this is needed for IDS, highlighting the aforementioned; however, they don't go into detail about how network monitoring would function. However, this was discussed in 2015 by the researchers Svoboda, Ghafir and Prenosil.

The authors (Svoboda, Ghafir and Prenosil, 2015) discuss in their paper how different methods of network monitoring can be used. In Svoboda's

paper, they discuss how five different approaches to security monitoring can be used and talk about the advantages and disadvantages of each approach; the authors discuss methods such as traffic duplication. Packet capture. Deep packet Inspection and Flow Observation. At the end of the paper, the authors highlight how each method has its pros and cons; however, no method is better than others—highlighting to the user the importance of choosing a method that is right for their software.

2.2.4 Intrusion Detection Systems and Intrusion Prevention Systems

Intrusion detection systems and intrusion prevention systems (IDS, IPS) are a critical part of any security monitoring system and historically had fantastic integration with the Linux operating system due to the open-source nature of Linux, allowing the IDS to have a deeper integration with the operating system. (Wang and Chang, 2022)

Intrusion detection systems have a wide range of capabilities and uses depending on the feature set needed for security. One such feature that is widely used is anomaly-based detection. As explored by the authors Depren *et al.*, (2005). The authors explore using machine learning inside of an IDS for anomaly detection. The authors use decision trees as a classifier to detect the type of attacks on the system. When testing the data, the researchers discovered that their anomaly-based detection method had a 98.96% detection rate and a false positive rate of 0.2%. The author's research shows how successful machine learning can be when integrated into an IDS system and highlights how IDS systems can successfully stop attacks.

A key principle of IDS systems is where the IDS will be based; for network security, a network-based IDS (NIDS) can be used, and for host security, a host-based IDS (HIDS) can be used. The concept of this is explored further in a paper by Wang and Zhang, (2012) who propose to combine both methods for a hybrid IDS. The authors discuss how both HIDS have issues with taking up system resources on each node and only having the

capability to detect issues on themselves. The authors also discuss the downsides of NIDS and how it is limited to the network and can't detect intruders if they are only one computer. The author also highlights the limitations of NIDS with encryption, discussing how the IDS can be broken by using this feature. The authors highlight in their study that a combination of the two host types should be used rather than one or the other.

2.2.5 File Integrity Monitoring

File integrity monitoring (FIM) is a critical part of the Linux system and ensures that files are not modified or changed. This can be useful To the Linux operating system itself as if a critical file such as a boot configuration file was corrupted, then the operating system could have issues booting. However, this issue could be corrected through the use of FIM. (Loscocco *et al.*, 2007) Although this technology is used to verify files it can also be used for security monitoring of files, as explored by Kim and Spafford, (1994). This paper is an early example of how file integrity monitoring has been used for security since the 1990s and introduces the idea that FIM can be used as a "tripwire", as described by the author. The authors discuss how this technology can be used alongside other monitoring methods, such as log analysis, and how this can alert administrators when files are modified or deleted. This research is developed by the authors (Al-Muntaser, Mohamed and Tuama, 2023) In a paper where the writers discuss how FIM can be used in a real-time IDS to detect changes in critical files. The researchers tested the FIM with 100 simulated scenarios where files were modified, and the FIM had high accuracy in the tests; however, it faced an issue where there were a small amount of false positives, This highlights an issue with file integrity monitoring in security systems, Some files that are being monitored can be modified by genuine users or systems rather than malicious attackers however FIM systems would not be able to detect the difference.

Chapter 3 Methodology

This chapter of the paper will focus on how the aims and objectives of the project were met during the development of the Intrusion Detection and Prevention System. (IDPS). This chapter will be split into sections, each showing a step taken during the development. Each section will consist of subsections detailing different approaches taken during the development and the reasons behind each design decision.

3.1 Design Requirements of the Project

Before the project's development began, the project's requirements needed to be sorted out. Through talks with the clients the requirements of the project were laid out. After this, a MoSCoW analysis was completed on the requirements, and they were sorted into two sections: "must haves" and "could haves". Must halves are best described as something the client requires for the project to succeed. Meanwhile, “could haves” are features and functionalities the client would like to have but are not deemed essential to the project's success.

3.1.1 Must Haves

Discussions with the client occurred, and the project needs were laid out. The project aimed to develop an Intrusion Detection System capable of detecting threats on remote servers. After this further, smaller requirements were set out. The IDS would be running on Ubuntu Linux, so it must be compatible with Ubuntu systems. This would be an essential requirement as software incompatibility would leave each agent vulnerable to cyber attacks. Another requirement set out by the client was that the system would have to have an agent per virtual machine (VM). This requirement would shape the development of the IDS into a HIDS-based system. The IDS system also would have two main detection requirements to be classified as a success: The ability to detect SSH and remote connections and the ability to detect file modification.

Table 1- Description of the "Must Haves" From the Client

Must Haves	Description
Develop an Intrusion Detection System	Develop a functional Intrusion Detection system.
Compatibility with Ubuntu Linux	Allow the IDS to be fully compatible with the Ubuntu operating system.
Host Based System	IDS must be a host-based intrusion detection system with an agent per virtual machine.
Network Authentication Detection	Agents must be able to detect remote connections to the machine.
File modification Detection	Agents must be able to detect when files and folders are modified.

3.1.2 Could Haves

Discussions with the client took place and the "could haves" of the project were laid out. The project aimed to develop an Intrusion Detection System capable of detecting threats on remote servers. After this further, smaller "could haves" were set out. These goals are not required for the project to be deemed a success but would rather serve as a nice to have for the clients. The clients asked for a way for the bot alerts to be handled and decided that they would like to use a Slack bot for the project; however, it would be okay if other methods of alerts were used. The client also wanted the IDS to use as few ports as possible to avoid long and manual deployment times. However, stated that they could open specific ports if required for the project. The clients stated they would like the IDS system to have a central management system. This would be where all the alerts were sent to and saved. The clients desired an easy deployment process as they wanted to be able to deploy the software as simply as possible through the use of simple scripts. So, the IDS would require minimal setup.

Table 2- Description of the "Could Haves" From the Client

Could Haves	Description
Simple Alerts	A Slack alert bot that could alert the security team of any intrusions.
Simple Deployment	The ability to deploy the software through the use of simple scripts
Minimum Ports	The client desires the IDS to use as few ports as possible for the system to function.
Central Management System	A central "command" system that could be used to monitor the IDS agents and handle alerts.

3.2 Development Methodologies

When developing large, complex software, development teams often use development methodologies to help guide product development. (Despa, 2014). Although there are a lot of development methodologies, it was decided that the Agile Software Development cycle would be used for this project. Although Agile was chosen for this project, the original plan was to use the Waterfall Methodology; however, due to the project's timeframes and changing goals, it was decided that the adaptability of the Agile framework was more in line with the project's goals.

3.2.1 Agile Development Process

The Agile Development Process is a software development methodology popular amongst developers and product managers. The development process is more flexible than other methodologies, highlighting adaptability and working software over documentation and planning. This methodology fits this project well, as different features were modified and changed throughout the development cycle.

The Agile software development cycle follows a loop of meetings, planning, designing, development, testing and evaluation; however, for this project, the meetings with the client were stepped back, so this methodology was

modified, as highlighted in Figure 2 below. This methodology will be used throughout the project to help develop every required feature.

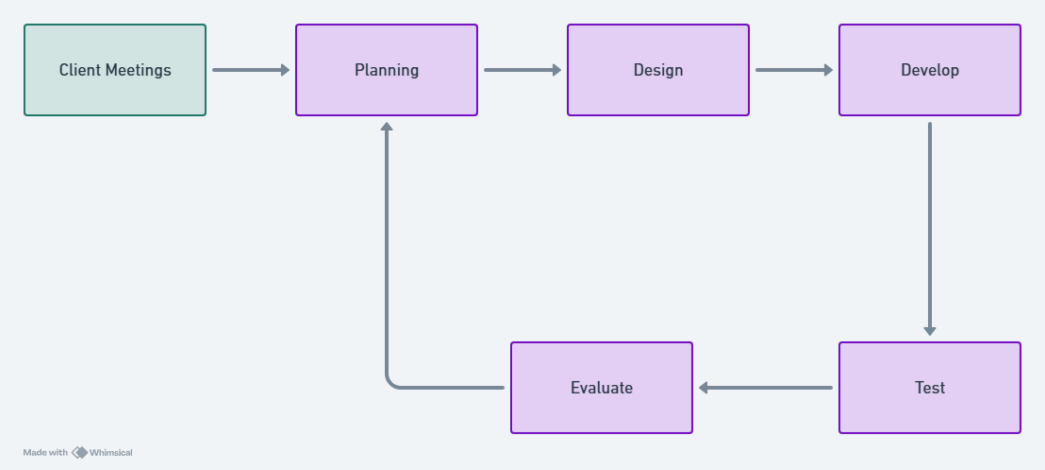

Figure 2- Modified Agile Software Development Process

3.2.2 Waterfall Model

The Waterfall Model is another popular development methodology used by software developers. The model focuses on following a planned sequential sequence of steps when developing software and requires a lot of planning in the early stages of the development process. However, the waterfall model was not deemed viable for this project due to the volatility of the project's requirements. An example of the Waterfall Model can be seen below in Figure 3.

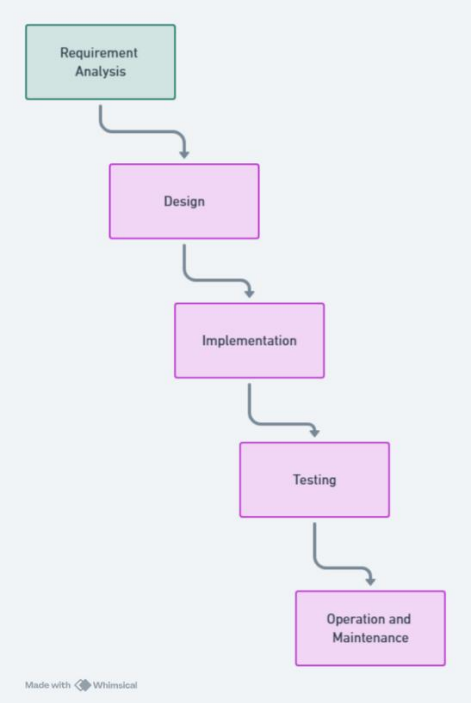

Figure 3- Waterfall Development Process

3.3 Planning and Design

When following the Agile development process, the second step of the cycle is planning; due to this, the initial wireframe for how the IDS would function was drawn up. This wireframe shows how the IDS would be structured, as shown in Figure 4 below. This diagram shows the initial idea of how the IDS would be structured. Each node would be placed onto an agent, and all would share one “command” server, which a security team would control. This approach of a one-to-many design would allow ease of scalability by introducing modularity into the system. Introducing new command servers could allow administrators in different offices to have their own clusters.

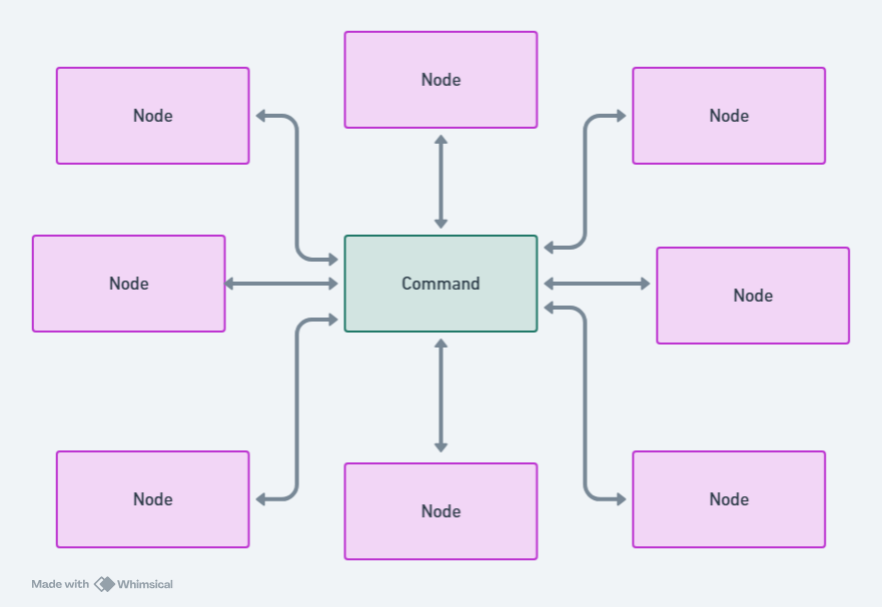

Figure 4- Wireframe of IDS Structure

3.3.1 Node Design

When designing the IDS, one of the “must haves" of the project was for the IDS to be a HIDS-based system. This would be accomplished through the use of “Nodes”, with each node being an agent that would be on the monitored system. Before starting development on the nodes, an initial wireframe design of how the node would function would be drawn; this wireframe would attempt to incorporate all of the project's requirements as well as the optional tasks of the project to help shape the project's design. The wireframe would further detail a potential way that each feature would be implemented. The node wireframe can be seen below in Figure 5.

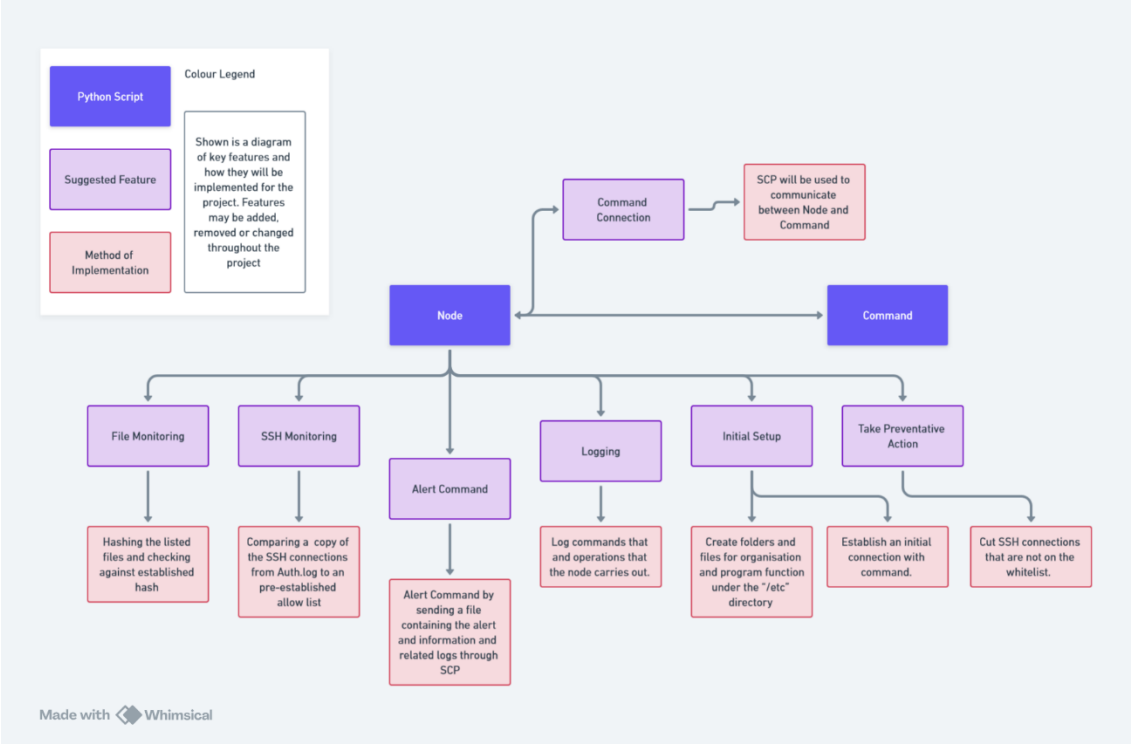

Figure 5- Initial Wireframe Design of Agent Nodes

3.3.2 Command Design

When designing the IDS system, it was decided that a "command" node would be used; this node would be responsible for controlling nodes and act as a central node to agent nodes that would send alerts when a rule violation occurred. Using a Command node would benefit the project by allowing the central management of nodes to aid security teams. Before starting active development on the command node, a wireframe of the node was made. This would show the structure of the command node and its features, such as how it would handle alerts and node communication. The wireframe design of the command node can be seen below in Figure 6.

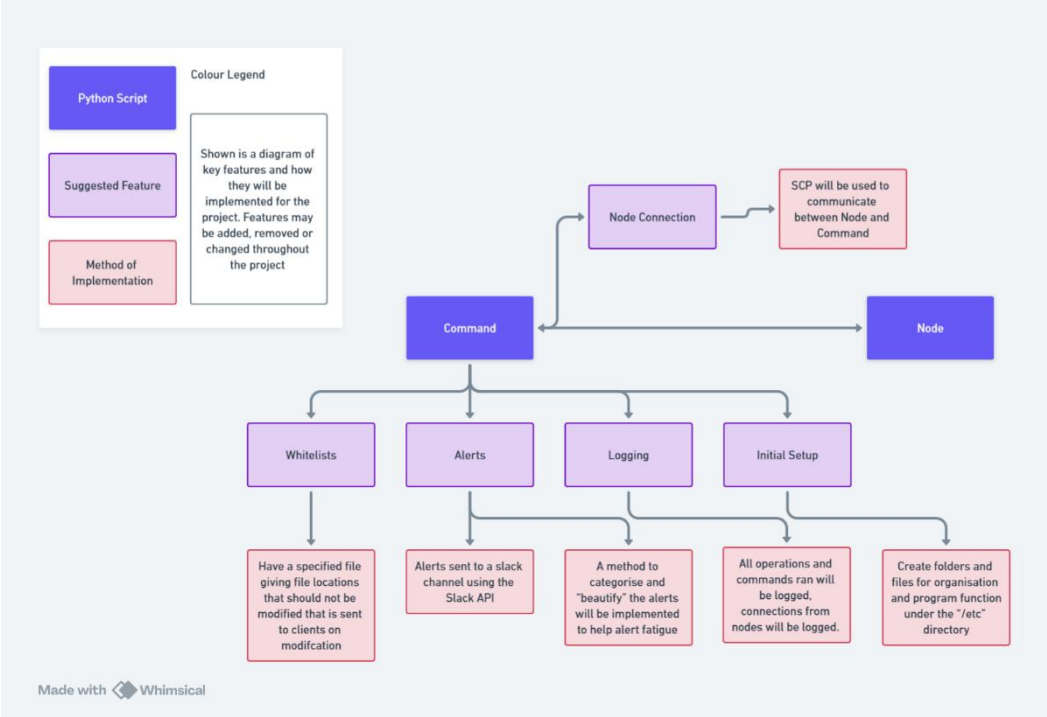

Figure 6- Initial Wireframe Design of Command Node

3.4 Transferring Data Between Nodes and Command

When monitoring each node, the command and nodes must be able to communicate with each other. This is required as when nodes detect a rule violation, the alert will be sent to the command for a security team to see. When deciding on a method of communication, there are multiple things to consider that influence what communication protocol will be used. Race conditions, system resources, security, and scalability can be strict barriers that different protocols face and should be considered when choosing one.

The first step in creating the communication method between the node and command was to decide what protocols and methods would be used to develop the code. When analysing the downsides and upsides of each method, three methods of communication were implemented: Sockets, Secure Copy Protocol and AWS. However, it was decided that sockets would be scrapped after prototyping due to security concerns.

3.4.1 Sockets

When deciding on methods to use for communication between nodes, sockets were the first method prototyped. Sockets were chosen for their wide compatibility with different machines, and because sockets are part of the Python standard library, they would require little setup. When designing the socket prototype for the IDS, The first step was to obtain a form of SSL certification; This was a vital part of the communication as without SSL, the information would not be encrypted and would be vulnerable to man-in-the-middle attacks. (Bhushan, Sahoo and Rai, 2017). For prototyping, a self-signed certificate was created using the code displayed below in Figure 7.

```
def sslCert():
    if not os.path.exists("server.crt") or not os.path.exists("server.key"):
        openssl_cmd = "openssl req -x509 -newkey rsa:2048 -keyout server.key -out server.crt -days 365 -
nodes"  os.system(openssl_cmd)
```

Figure 7- Code to Create an Self Signed SSL Certification

When deploying code into production, it is important not to use self-signed certifications but to use certification from certificate authorities. This is because certifications verify the legitimacy of web servers, and when using self-signed certifications, there is no way of knowing if the certification being received is genuine or not. (Radif, 2018) This was one of the main security concerns when prototyping sockets, so this communication method was not chosen for further development.

3.4.2 Secure Copy Protocol

The second prototype method was using a secure copy protocol (SCP). This method has a lot of positives, such as not requiring ports to be opened up, unlike sockets. SCP relies on the SSH protocol, so it only uses port 22. This is a benefit as this was one of the “could haves” that the client wanted to have and would reduce the setup time of the IDS.

```
def commandConnection(ip_address, ssh_username, data, filereason):
    currentTime = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    filename = f"NINJAEYE-{ip_address}-{currentTime}-{filereason}"
    filepath = f"/etc/NinjaEye/logs/alerts/{filename}"


    # Write data to file
    with open(filepath, "w") as f:
        f.write(data)

    #Sends data to server
    os.system(f"scp {filepath} {ssh_username}@{ip_address}:/tmp")
    return
```

Figure 8- Function Responsible for Command Connections

As shown above, in Figure 8, is the code that was developed for sending files to the command node. The code starts off by creating a file and naming it “NINJAEYE-{IP}-{TIME}-{FILEREASON}”. This was done so that information could easily be extracted from the file name when the command node receives the file. After the file is made, the data received by monitoring functions is written into the file and then sent to the command server "/tmp” folder. A flowchart showing this progress can be seen below in Figure 9.

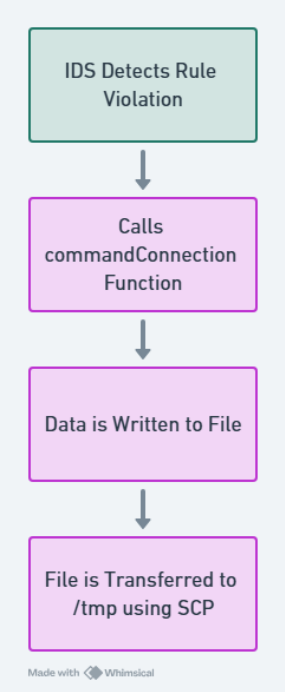

Figure 9- Flow Chart of the SCP File Transfer

When the file is successfully transferred to the Command node, a method of monitoring the /tmp folder is created using the following code shown in Figure 10.

```
def alertmonitoring():

    folderpath = "/tmp/"

    while True:
        for filename in os.listdir(folderpath):
            filename.startswith("NINJAEYE"):
            source = (folderpath + filename)
            shutil.move(source, "/etc/NinjaEye/alerts")
            print(f"ALERT! See /etc/NinjaEye/alerts/{filename} for more information!")
        time.sleep(5)
```

Figure 10- /tmp Folder Monitoring Function

The code above is a basic form of alert monitoring on the command node; the code starts by monitoring files inside the "/tmp" folder, reading each filename and checking if the filename starts with “NINJAEYE”. If that filename is present, then the file Is moved to the alerts folder using the shutil.move command, and an alert is printed.

3.4.3 Amazon Web Services S3 Buckets

The third method of transferring data between the agent nodes and the command server that was chosen was the use of Amazon web services (AWS) S3 Buckets. The AWS S3 cloud service provides simple storage solutions and allows developers to use Amazon's storage servers. This can allow developers to use the service as middleware, allowing them to use the service to transfer data between two devices. This can benefit developers by saving system resources and allowing developers to easily transfer data across the internet rather than just the local network. Another benefit of using AWS is the ability to scale and use elasticity. Although cloud services can benefit companies, they also come with trade offs, such as operational costs and the requirement for additional ports to be opened up on each machine, going against the could haves objectives in the MoSCoW analysis.

When prototyping how AWS would be integrated into the IDS, the first stage of the process was to make a flow chart detailing how the process would function. The flowchart can be seen in the figure below.

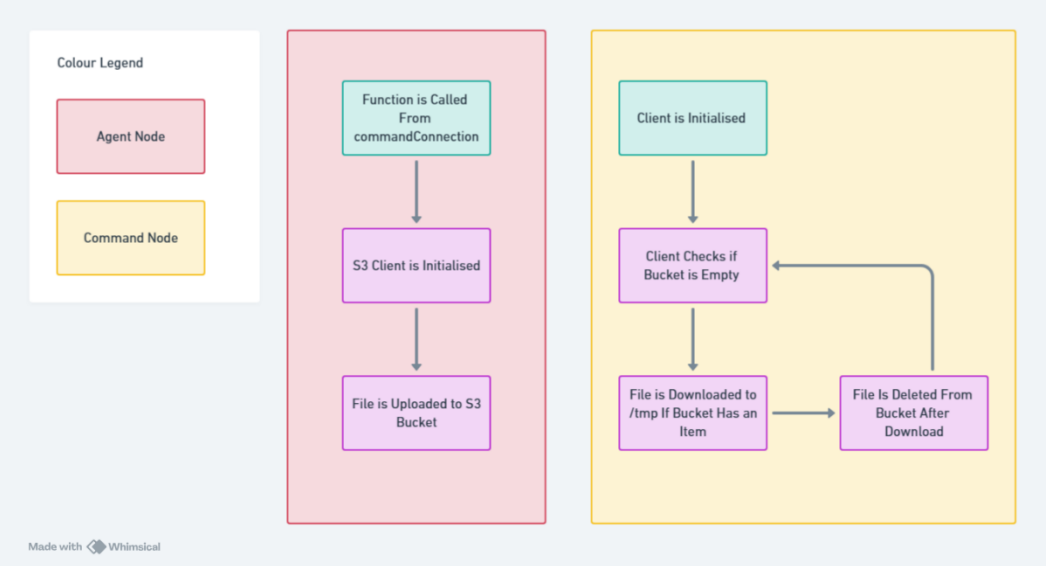

Figure 11- Flowchart of Data Transferring Using AWS S3

The abovementioned method is a simple method by which data can be transferred between two modes. The flowchart highlights how the data will be removed from the S3 bucket once downloaded; however, this is a feature that the client could remove if they wished. Once the flowchart was done, the development of the code began. The first iteration of the code for the agent node that was produced can be seen below in Figure 12.

```
def upload_to_s3(file_name, object_name=None):

    Access_Key = os.getenv('AccessKey')
    Secret_Key = os.getenv('SecretAccessKey')
    BUCKET_NAME = os.getenv('BUCKET_NAME')
    REGION = os.getenv('REGION')

    if object_name is None:
        object_name = file_name

    # Initialize S3 client
    s3 = boto3.client(
        's3',
        aws_access_key_id=Access_Key,
        aws_secret_access_key=Secret_Key,
        region_name=REGION
    )

    try:
        s3.upload_file(file_name, BUCKET_NAME, object_name)
        print(f" Upload successful: {file_name} to s3://{BUCKET_NAME}/{object_name}")
        return True
    except FileNotFoundError:
        print(" File not found")
        return False
    except NoCredentialsError:
        print(" Credentials not available")
        return False
```

Figure 12- Agent Nodes Code For Uploading Files to AWS

The code above shows the function written for the agent nodes that allows them to upload files to the S3 buckets; the code starts by initialising the S3 client and then attempts to upload the file, which is passed into the function by "commandConnection". Once completed, the code prints a message telling the user that the code has uploaded the file; however, if there is an issue, the code will return out of the function and display an error message. After this code was developed, the code for the command function was developed. The code can be seen below in Figure 13.

```python
def downloadFromS3():

    Access_Key = os.getenv('AccessKey')
    Secret_Key = os.getenv('SecretAccessKey')
    BUCKET_NAME = os.getenv('BUCKET_NAME')
    REGION = os.getenv('REGION')

    s3 = boto3.client(
        's3',
        aws_access_key_id=Access_Key,
        aws_secret_access_key=Secret_Key,
        region_name=REGION
    )

    try:
        bucketObjects = s3.list_objects(Bucket=BUCKET_NAME)

        for obj in bucketObjects.get('Contents', []):
            key = obj['Key']
            file_path = os.path.join("/tmp/", key)

            s3.download_file(BUCKET_NAME, key, file_path)
            s3.delete_object(Bucket=BUCKET_NAME, Key=key)
            print(f"Downloaded {key} to /tmp/ and removed from S3.")
            return True

    except NoCredentialsError:
        print("Credentials not Valid")
        return False

    except:
        print("Error Downloading Files")
        return False
```

Figure 13- AWS Download Function for Command Node

3.5 SSH Monitoring

Having shown how different methods of data transfer can occur, one common theme that all the methods faced was the use of the “data” and “filereason” variables. These variables are created using the monitoring functions, Which detect when a rule violation occurs and send this information to the “commandConnection” function. One of the “must haves” of the clients was for the IDS to have an SSH monitoring system; this system would check to ensure that no SSH connections occur and report if one is active. Although there are multiple ways to accomplish this, such

as process monitoring or network monitoring the method investigated for this project was the use of Log Analysis based on the research done within the literature review.

3.5.1 Log Analysis

Following Buckleys and Siewioreks research, the use of log analysis was going to be implemented. Linux monitors when ssh connections happen using the “auth.log” file. The file contains all attempts at authentication connections on the system; this includes failed connections and successful ones and is not limited to SSH. The auth.log files also contain information on all authentication methods on the system, including connections from other services and the use of the sudo command. (Isaiah, 2025). This means that the code produced for SSH monitoring could be easily modified for other monitoring methods, such as sudo or FTP.

The first step when implementing log analysis was to design a generic flowchart of how the code would function. The flowchart would help guide the development of the log analysis feature. The flowchart can be seen below in Figure 14.

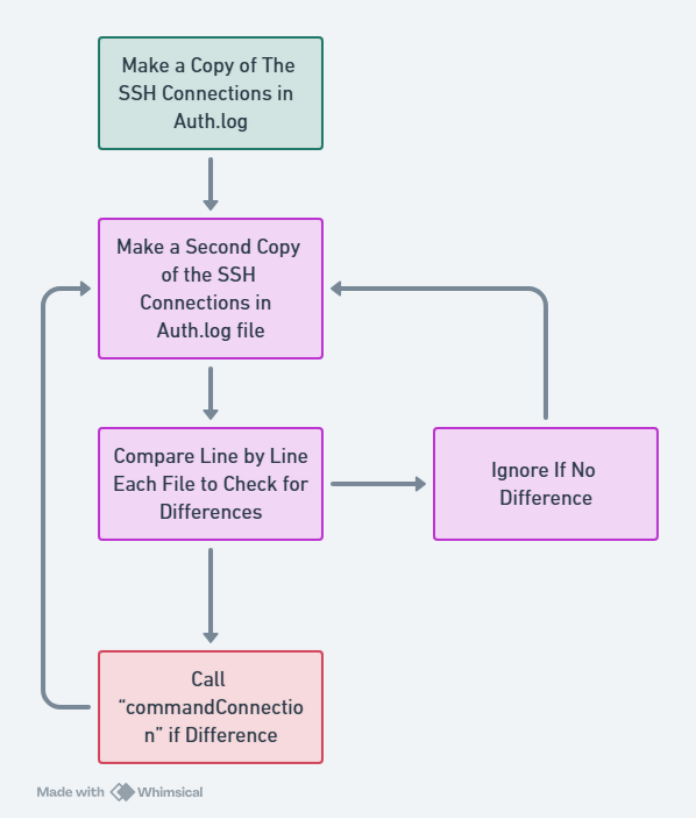

Figure 14- Flowchart of SSH Monitoring using Log Analysis

When planning how the log analysis would function, it was planned that the function would gather two copies of the ssh connections from the auth.log files. The first copy would be a control file that would serve as the baseline, and the second file would be compared to the baseline every 30 seconds. If a change were detected between the two files, then this information would be sent to the alert function to be sent to the command. The code developed for this can be seen in Figure 15 below.

```
def sshCompare():
    filereason = "unauthorizedSSH"
    while True:

        sshCmd = "grep sshd /var/log/auth.log > /etc/NinjaEye/logs/sshCompare.txt"
        os.system(sshCmd)

        afterLog = open("/etc/NinjaEye/logs/sshCompare.txt")
        beforeLog = open("/etc/NinjaEye/logs/ssh.txt")

        beforeLog_data = beforeLog.readlines()
        afterLog_data = afterLog.readlines()

        before_set = set(beforeLog_data)
        after_set = set(afterLog_data)

        differences = after_set - before_set

        if differences:
            alert_data = ""
            for line in differences:
                print("\t ALERT! ", line, end="")
                alert_data += line #combines the alert data


            commandConnection(ip_address, ssh_username, alert_data, filereason)

         else:
            print("No differences found.")

        afterLog.close()
        beforeLog.close()
        time.sleep(30) #checks every 30 seconds
```

Figure 15- Code for SSH Monitoring Using Log Analysis

The code above follows the flowchart design made in Figure 14. The code was developed to be made modular at a date and easily modified for other monitoring purposes. This could be done by changing the “grep” command to look for a different feature instead of “sshd”.

3.6 File Monitoring

One of the project's requirements was to allow the IDS to monitor files and folders for change or access. The system would require the ability to monitor when a file or folder changes; this can be from things such as the addition or removal of files, modification to the contents of the file or if a system or person has accessed the file.

Although there are many commercial tools available for this function, many of these tools cost money and are ill-suited for the project, and could require the opening of ports or other configurations which would go against the project's aims and objectives. When designing how this feature would function, the first stage, as detailed by the agile development process, would be to plan and design the feature, exploring different ways to implement the feature. When planning out the feature, one main way of completing this goal was considered, the python “OS.Stat” function. Although this method would face its own benefits and downsides, which are detailed below.

3.6.1 Stat

The OS.Stat function is a part of the Python standard library (Lundh, 2001) and provides information about a file or folder when used; the command returns an object which contains information about the file. The two main items that were used for this stage in the project, however, were the "st_atime” and the st_mtime" These details can be seen below in Table 3.

Table 3- Table Detailing the Used OS.Stat Features

Feature	Description
st_atime	Details the last time a user or system accessed a file or folder.
st_mtime	Details are provided on the last time that the file was modified. This includes whether files were modified inside a folder.

When planning how the os.stat feature would be implemented, a flowchart was made detailing how the code would be developed. The flowchart can be seen below in Figure 16.

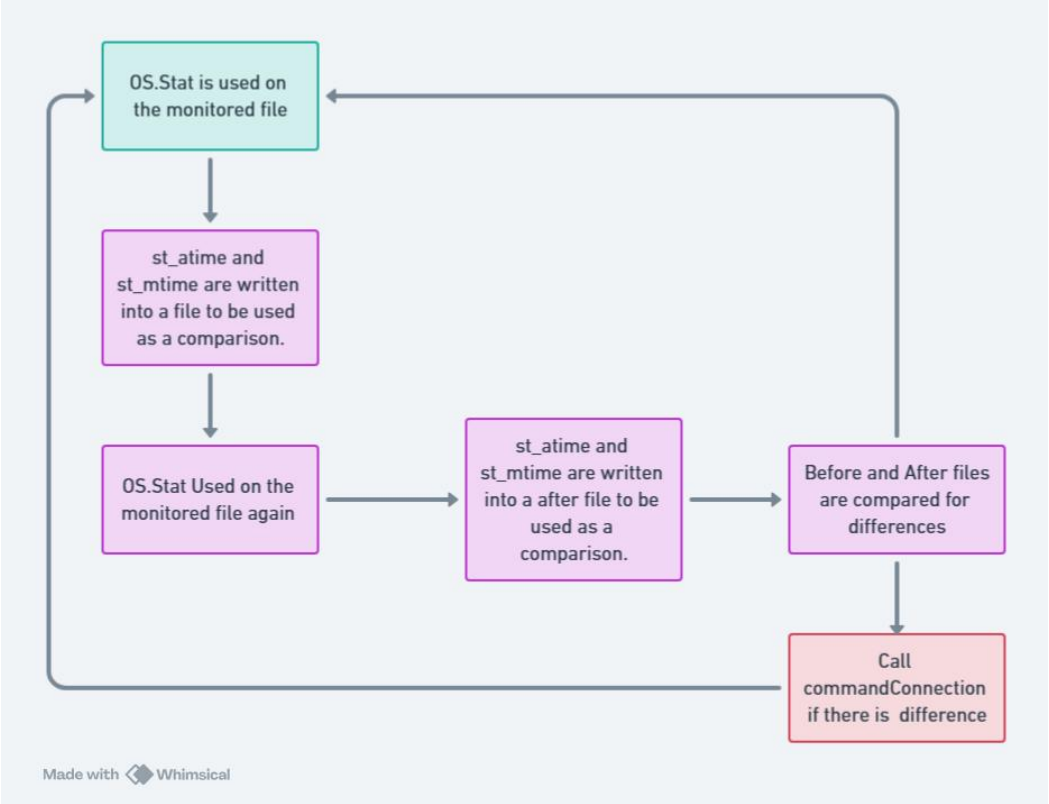

Figure 16- Flowchart Detailing How os.stat File Monitoring Would Function

The method chosen for the file monitoring using os.stat was similar to the SSH Log analysis method, where the code would function by comparing two files and checking if there was a difference. The full code for this can be seen below in Figure 17.

```
def fileChecker():  #helper function
    test =  os.stat("control_file.txt")
    statfile = "/etc/NinjaEye/file_compare/beforefile.txt"
    with open(statfile, "w") as file:
        file.write(f"{test.st_atime} : {test.st_mtime} \n")

    fileCompare()


def fileCompare():
    filereason = "unauthorizedAccess"
    while True:

        control = os.stat("control_file.txt")
        statfile = "/etc/NinjaEye/file_compare/afterfile.txt"
        with open(statfile, "w") as file:
            file.write(f"{control.st_atime} : {control.st_mtime} \n")

        afterLog = open("/etc/NinjaEye/file_compare/afterfile.txt")
        beforeLog = open("beforefile.txt")

        beforeLog_data = beforeLog.readlines()
        afterLog_data = afterLog.readlines()

        before_set = set(beforeLog_data)
        after_set = set(afterLog_data)

        differences = after_set - before_set
        if differences:
            alert_data = ""
            for line in differences:
                print(f"\t ALERT! folder accessed ", line, end="")
                alert_data += line #combines the alert data

            commandConnection(ip_address, ssh_username, alert_data, filereason)

        else:
            print("No differences found.")

        afterLog.close()
        beforeLog.close()

        with open("beforefile.txt", "w") as file:
            file.write(f"{control.st_atime} : {control.st_mtime} \n")

        time.sleep(30)
        fileChecker()
```

Figure 17- os.stat File Monitoring Code

3.7 Alert Monitoring

When Alerts are sent to the command server using commandConnection, the alerts are first stored inside the /tmp folder. The tmp folder is a folder used in Linux to store temporary files used by the program and is wiped at boot time. (Sharma, 2024). Due to this, a way to monitor the /tmp folder and move files from it into a permanent folder is required.

When designing the nodes, a method for identifying what an alert file is was created using file names. Each Alert file starts with “NINJAEYE”, This identifier would allow the command node to know through string concatenation what files it would be required to move and what files it could ignore. This method was chosen for its ease of implementation and efficiency, as the command node would not be required to read the file's contents but instead just read the file's name. The code that was developed for this can be seen below in Figure 18.

```
def alertmonitoring():
    CHANNEL_ID = os.getenv('CHANNEL_ID')
    SLACK_BOT_TOKEN = os.getenv('SLACK_BOT_TOKEN')
    folderpath = "/tmp/"

    while True:
        for filename in os.listdir(folderpath):
            if filename.startswith("NINJAEYE"):
                source = os.path.join(folderpath, filename)
                destination = "/etc/NinjaEye/alerts"
                shutil.move(source, destination)
                print(f"ALERT! See {destination}/{filename} for more information!")

                parts = filename.split(':')
                if len(parts) >= 4:
                    prefix = parts[0]
                    ip_address = parts[1]
                    current_time = parts[2]
                    filereason = parts[3]
                else:
                    prefix = "NINJAEYE"
                    ip_address = "Unknown"
                    hostname = "Unknown"
                    current_time = "Unknown"
                    filereason = "Unknown"

                reason = (f"ALERT! A issue has been detected and moved to {destination}\n"
                          f"Prefix: {prefix}\n"
                          f"IP Address: {ip_address}\n"
                          f"Time: {current_time}\n"
                          f"File Reason: {filereason}")

                send_message(CHANNEL_ID, SLACK_BOT_TOKEN, ip_address, current_time,
reason) time.sleep(5)
```

Figure 18- Code Used to Monitor the /tmp Folder

3.8 Slack

When discussing with the client how they would like to receive alerts, the client highlighted how they would like a simple method to receive them. When discussing further through the MoSCoW analysis of the project, it

was identified that a Slack bot would best be used to deliver the alerts. When designing the Slack bot, the bot would need to overcome issues identified by the client, such as alert fatigue and ease of setup.

3.8.1 Basic Bot

Development teams widely use Slack bots as a method of alert monitoring. Due to this, Slack has developed a wide range of API features that help design bots, and many community Python libraries have been created to make APIs more accessible. When designing the Slack bot, it was decided that it would use the slack_sdk to aid development. The official Slack development team develops the SDK so the library has greater integration with the API and helps ease the Slack bot's implementation.

When designing how the Slack bot would function, the first stage was deciding its features. It was decided that the Slack bot would feature a simple alert with the information and a picture. An image of what this looked like can be seen below in Figure 19.

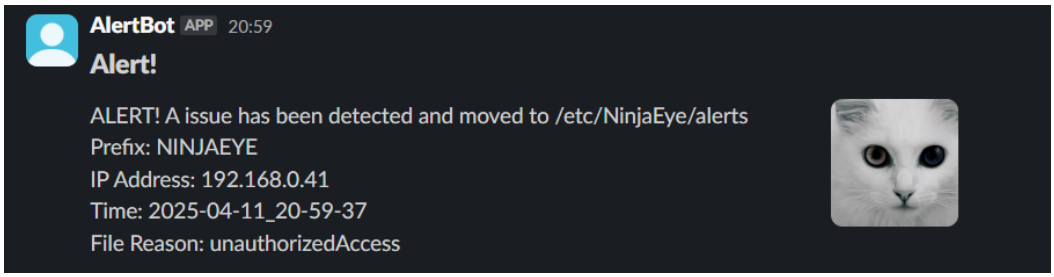

Figure 19- Slack Alert Example

When implementing the Slack bot, it was decided to keep it simple. This was so that if the client wanted to modify the bot to implement it into an existing bot, they were able to. The code for the basic bot can be seen below in Figure 20.

```
def send_message(channel, SLACK_BOT_TOKEN, ip_address, current_time, reason):
    client = WebClient(token=SLACK_BOT_TOKEN)
    cat_image_url = get_cat_image()
    if not cat_image_url:
        cat_image_url = "https://pbs.twimg.com/profile_images/625633822235693056/lNGUneLX_400x400.jpg"  ## fallback
image
    blocks = [
        {
            "type": "header",
            "text": {
                "type": "plain_text",
                "text": "Alert!",
                "emoji": True
            }
        },
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": f"{reason}\n"


            },
            "accessory": {
                "type": "image",
                "image_url": cat_image_url,
                "alt_text": "Random cat image"
            }
        },

            ]

    try:
        response = client.chat_postMessage(channel=channel, blocks=blocks, text="SSH Breach Alert!")
        print(f"Message sent: {response['message']['text']}")
    except SlackApiError as e:
        print(f"Error sending message: {e.response['error']}")
```

Figure 20- Slack Bot Code

The code for the bot is simple and starts off by authenticating using the “SLACK_BOT_TOKEN”. The code then starts a new client using the credentials and creates the format for the message using the “blocks” UI builder included in the Slack SDK. The code then posts the message using the “chat_postMessage” function and throws an error message if any issues are detected.

3.8.2 Cat API

During the conversation with the client, the client mentioned that they would mute the bot's channel if the bot were too "spammy". This highlights a common issue in the cyber security and medicine industry. Alert fatigue. Alert fatigue is when a person hears so many alerts that the person starts not paying attention to the alerts anymore. (Ban *et al.*, 2021; Wang *et al.*, 2024) This can be caused by bots sending too many messages alerts, or by a high rate of false positives provided by IDS. Although there are many ways to limit alert fatigue, the one chosen for use in this project was for the use of cat pictures.

Research completed by Nittono *et al.* (2012) revealed that looking at cute pictures of animals before completing focused tasks helped improve mental performance by 15%. This research supports the theory that the use of cat pictures can assist in reducing alert fatigue. Moreover, it helped shape the design of the Slack bot. One tool that can provide cat pictures is the use of the Cat API. The Cat API is an API that provides over 60k pictures of cats for free and would be implemented into the IDS using the code below in Figure 21.

```
def get_cat_image():
    CAT_API_KEY = os.getenv('CAT_API_KEY')
    CAT_API_URL = os.getenv('CAT_API_URL')
    headers = {"x-api-key": CAT_API_KEY}
    response = requests.get(CAT_API_URL,
headers=headers)
    if response.status_code == 200:
        return response.json()[0]["url"]
    return None
```

Figure 21- Cat API Implementation

The code shown above highlights how the Cat API was implemented. The code starts by fetching the API Key URL from an env file and then uses Python's request library to request the URL. Once the request is received, the code will return the URL, which will be passed into the “send_message” function. Providing the cat picture. This method is effective because no files are downloaded and stored on the machine, which saves resources. As well as no ports need to be opened besides port 80 which keeps the port numbers minimised. Supporting the objectives of the project.

3.9 Logs

When building IDS systems, a form of logging must be implemented for both the agent and command nodes. Logs can help provide replays of

events that can assist SOC analysts in analysing rule violations. Logs can also help ensure that the integrity of software stays intact and provide a detailed report of how the software performs and if there are any issues. (Kent and Souppaya, 2006). Although there are many standards of logs and implementation methods, the method chosen for this project was to use the “logging" Python library. The logging library is a part of Python's standard library, which allows it to be easily implemented into the code, assisting in the ease of maintainability of the code.

When designing how the logs would function, the first step is to set up how the logs will be configured. This step is important as it is where features such as log locations and the structure of the logs are located. The code that was designed for this can be seen below in Figure 22.

```
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)

handler = logging.handlers.RotatingFileHandler(
    "/etc/NinjaEye/log.txt", maxBytes=1000000,
backupCount=10
)

formatter = logging.Formatter("%(asctime)s - %
(name)s - %(levelname)s - %(message)s")

handler.setFormatter(formatter)

if not logger.hasHandlers():
    logger.addHandler(handler)
```

Figure 22- Logging Configuration

The code above is a modified version of code found in the blog post made by Verma, (2023). The logging code configures the format of the log messages to include the timestamps and the log level. The code also introduces rotating log files, which assists in ease of use by creating new

log files and archiving old files when the file reaches 10MB in size. After the configuration of the files is added to both agent and command nodes, the next stage is to implement the logging throughout the code. This can be done by using the code shown in Figure 23.

```
    logger.info("Attempting to fetch random cat
image")
```

Figure 23- Example Log Code

When implementing the logger, it is important that the code is placed in strategic positions in the code and that the logs highlight what is happening in the code. This helps highlight what is happening throughout the code executions and debug errors. When deciding what type of logger to use, it is important to use the correct flag associated with the message. The code highlighted below in Figure 24 shows how the info flag is used to portray information on what is happening in the code, whereas the error flag is used to portray if an error has occurred. The use of the three flags throughout the code: warning, error and info. It can support highlighting critical information in the logs such as software failure and assist in log analysis of the IDS.

```
try:
    response = client.chat_postMessage(channel=channel, blocks=blocks, text="NinjaEye Alert!")
    logger.info(f"Message sucesfully sent to slack")

except SlackApiError as e:
    logger.error(f"Error sending message to slack: {e.response['error']}")
```

Figure 24- Example of Info vs Error Flags in Logger

3.10 Initial Setup

Before the code can be deployed, a method of setting up files and folders must be developed. Premade folders where files and materials could be

stored would assist in maintaining the software and would allow administrators to easily find files and logs. When deciding where the files would be stored and what folders to use, it was decided that they would all be stored underneath the "/etc." directory. The /etc directory is a common directory developers use to store files that do not fit under other categories. (Academy, 2023). The files that are included in the directory typically lean towards configuration files; however, log files may also be stored. This would allow a centralised location for all the project files, including logs and alerts. The code that was developed for this can be seen below in Figure 25.

```
def firstsetup():

    folderpath = "/etc/NinjaEye/"
    folders = ["logs", "logs/alerts", "file_compare"]
    file_paths = ["logs/log.txt"]

    ##creates the folders and files needed for the program to run
    try:

        for folder_name in folders:
            folder_path = os.path.join(folderpath, folder_name)
            os.makedirs(folder_path, exist_ok=True)
            logger.info(f"Created directory: {folder_path}")


        for file_name in file_paths:
            file_path = os.path.join(folderpath, file_name)
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            if not os.path.exists(file_path):
                logger.info(f"Creating file: {file_path}")
                with open(file_path, 'w') as f:
                    pass

        print(f"Directories and files set up successfully in {folderpath}.")
    except PermissionError:
        print("Permission denied: Please run the script with elevated privileges.")
        exit(1)
```

Figure 25- Function for Setting up Folders and Files

The code shown above shows how the folder and file setup were created. The code takes folders and files names stored in the arrays and iterates through them one at a time using os.makedirs to create the folders and files. If the files already exist, the code will pass over them and not attempt to override them. Once this is completed, the function will end and move

on to the next stage of the code; however, if the function has an error at any stage in the code, the code will exit and print to the terminal an error.

3.11 Configuration

When designing the file monitoring system, the function would only monitor one file or folder that was hard coded. That issue represented a problem in which the code would be complex to modify. Going against the project's aims. A solution that was decided upon was to make use of a configuration file. Developers use configuration files to allow simple customizability in the code without modifying the main code itself. This can be accomplished in many ways, such as the use of XML, YAML, INI or JSON. (Kenlon, 2021). Although each file type provides its own advantages and disadvantages, it was decided that for the project, the use of JSON was going to be implemented into the project. This was due to JSON's portability, ease of use, and implementation within the Python standard library.

When developing how this JSON would be implemented into the code, the first stage of the design process was deciding what would be included in the JSON file. It was decided that two main parts would be implemented into it. These would be the file paths to be monitored and the time interval between the functions running. This allows the code to be customised based on performance with a lower time being run on a faster machine. Furthermore, the ability to have the code easily scaled with the ability to add more files for the file monitoring function. These two features allow end users to easily customise the code based on their needs for the project. The full JSON can be seen below in Figure 26.

Figure 26- Example JSON File

When modifying the file monitoring function to integrate the JSON, a new method of calling the function was required. The first stage of prototyping a new method was to make a flowchart detailing possible methods that could be used to complete this. The flowchart can be seen below in Figure 27. After designing an outline of how the config file would be integrated, the code was developed, which can be seen below in Figure 28.

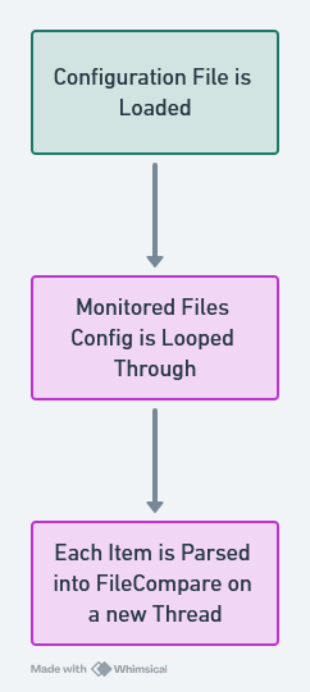

Figure 27- FileCompare Integration with Config.json

```
if __name__ == "__main__":

    firstsetup()
    ip_address, ssh_username = get_env_variable()

    sshThread = threading.Thread(target=sshLog, daemon=True)  # Create a thread for the sshCompare function
    sshThread.start() ## Start the thread

    time_interval = config["monitored_files"]["time_interval"]

    for x in config["monitored_files"]["file_paths"]:
        label = x["filename"]
        path = x["filepath"]

        fileThread = threading.Thread(target=fileCompare, args=(path, label, time_interval), daemon=True)
        fileThread.start() ## Start the thread

    sshThread.join()  ##waits for both threads to finish before exiting the program.
    fileThread.join()
```

Figure 28- Main Function For Agent Nodes With Config.json Implementation

3.12 Testing

Testing is completed throughout the code development when following the Agile development methodology. This method of continuous testing allows the code to be evaluated often and helps create higher-quality code to be used; however, throughout the project's development, a central testing method is required to help keep tests repeatable. For this project the use of manual testing was used to complete this.

Manual testing has the benefit of being time efficient on smaller projects such as this and can allow customizability to the testing procedures. The downside of manual testing, however, is that the testing methods can become more time-restrictive throughout the code development as the project's complexity increases. The requirements for this method were focused on functional testing. The theory is that the software should be tested to see if it can complete specified requirements. (Beizer, 1995)

3.12.1 SSH Testing

The requirement for the SSH testing was that the software would be able to detect if an SSH connection had occurred. This was completed by connecting to the Agent nodes VM from an external SSH connection and testing if the SSH could be detected. A flowchart detailing this can be seen below in Figure 29.

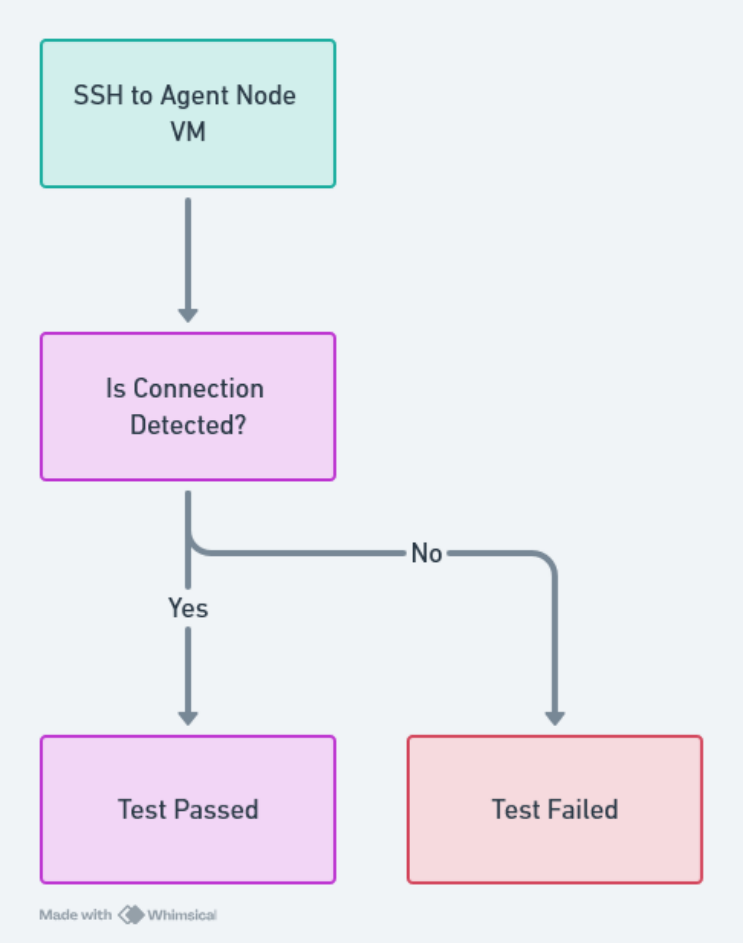

Figure 29- SSH Functional Testing Flowchart

3.12.2 File Monitoring Testing

The requirement for the file monitoring feature to be considered a success was that the software would be able to detect if a folder/file had been accessed as well as if a folder/file had been modified. This would be tested by accessing a file using the nano command, modifying it, and reading the contents of a file using the cat commands. Both of these methods were chosen as the tools cat and nano come preinstalled on Ubuntu Linux. (*Ubuntu*, no date) The operating system the IDS is tailored towards. A flowchart of the testing method for file/folder monitoring can be seen below in Figure 30.

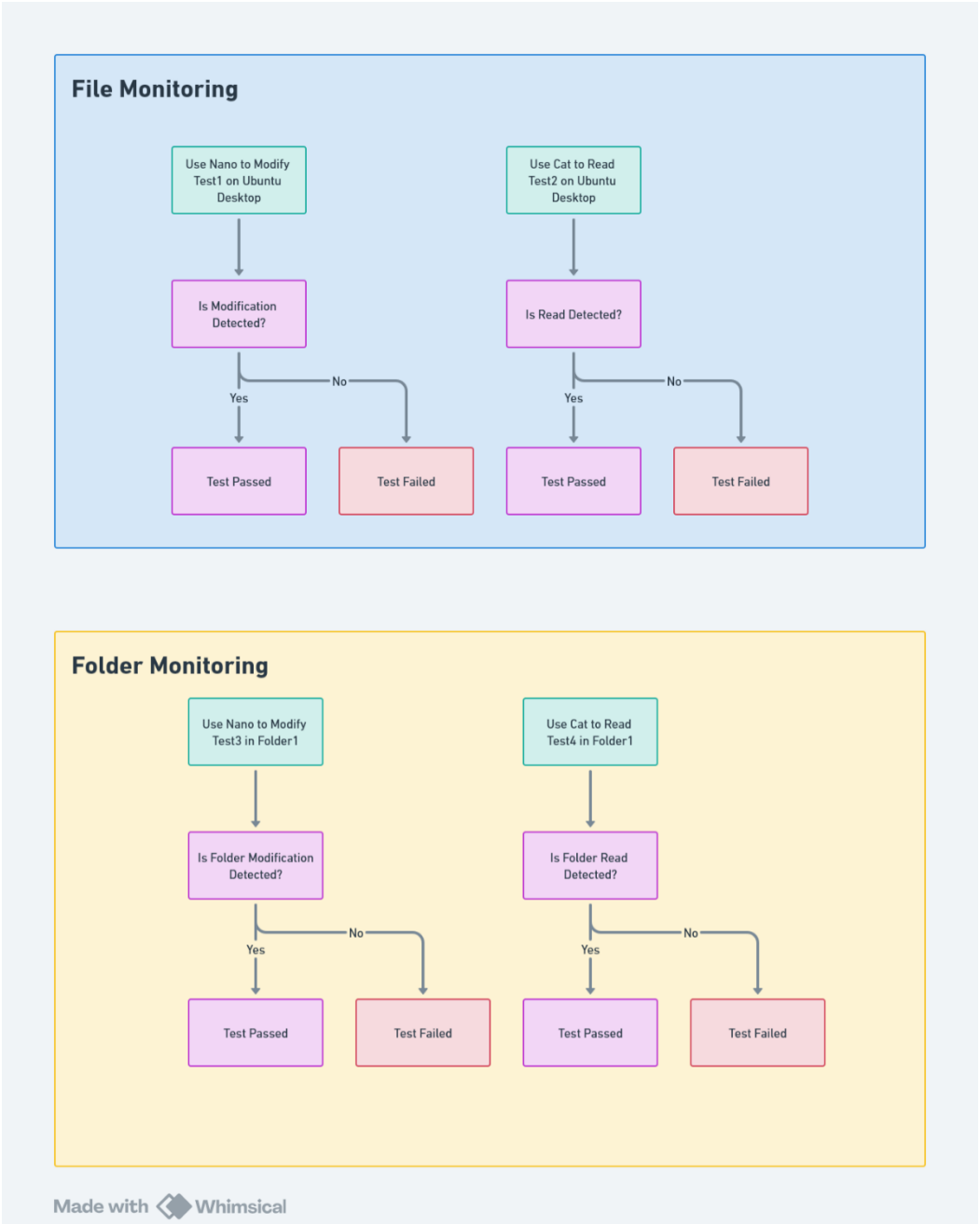

Figure 30- File and Folder Monitoring Testing Methods

3.12.3 Command Node Testing

When the command node was tested, the requirements were to successfully move the correct files from /tmp to their designated directories and to send an alert to Slack when it detected a file. This would be tested by placing files within the /tmp folder, checking if an alert was sent, and if the files had been moved into the correct folders.

Chapter 4 Results

This section of the project will highlight the findings and results of the testing completed in Chapter 3, where each feature was tested using functional testing to determine if it was viable. The criteria for success for each feature was that it could successfully detect if a rule violation had occurred and successfully report it to the command node. However, the success criteria for the command node were that it could successfully detect when an alert file was located inside its /tmp directory and move it into the correct location then send an alert to Slack.

4.1 SSH Testing

When testing if the SSH monitoring was working, the results showed that the software successfully detected when a user had connected to the machine. Fulfilling one of the project objectives in the introduction and MoSCoW analysis. This is evidenced by the log files showing the connection to the host from the IP 192.168.190.128. The full log of this can be seen below in Figure 31.

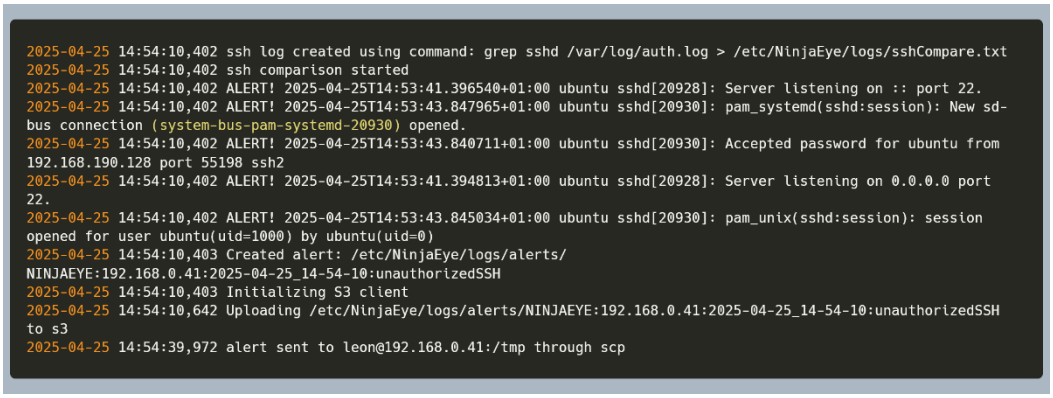

Figure 31- Logs Showing SSH Connection

4.2 File Monitoring Testing

When testing the file/folder monitoring, the testing was split into two main sections. File testing and folder testing. Both sections were tested using

the nano command and the cat command. The results showed that the file monitoring worked correctly and was successfully detected when files were modified and accessed. Evidence of this can be seen in the logs below in figures 32 and 33, which show that the IDS detected the access and modifications. However, the folder testing revealed that the software did not successfully detect the folder's content being accessed or modified, revealing a logic bug in the software and a limitation of the os.stat command.

Figure 32- Log Files Showing Evidence of Nano

Figure 33- Log Files Showing Evidence of Cat

4.3 Command Node Testing

When testing, the command node files were placed into the /tmp folder, and the command node would need to determine if a file was an alert and then move the file into the designated directory and send a slack alert to be considered a success. During the testing of the agent nodes, files were uploaded to s3 and then downloaded into the /tmp folder of the command node. Once this happened, the command node successfully determined what files were alerted, transferred them to their designated directories, and sent a Slack alert. This is evidenced by the log file below in Figure 34 and the Slack alert in Figure 35.

```
2025-04-25 15:18:07,243 - __main__ - INFO - Attempting to access S3 bucket
2025-04-25 15:18:07,776 - __main__ - INFO - Files found on s3 bucket
2025-04-25 15:18:07,776 - __main__ - INFO - Downloading NINJAEYE:192.168.0.41:2025-04-25_15-18-03:unauthorizedAccess to
/tmp/NINJAEYE:192.168.0.41:2025-04-25_15-18-03:unauthorizedAccess
2025-04-25 15:18:08,183 - __main__ - INFO -  NINJAEYE:192.168.0.41:2025-04-25_15-18-03:unauthorizedAccess removed from
S3.
2025-04-25 15:18:08,671 - __main__ - INFO - Detected alert and moved to /etc/NinjaEye/alerts/
NINJAEYE:192.168.0.41:2025-04-25_15-18-03:unauthorizedAccess
2025-04-25 15:18:08,671 - __main__ - INFO - attempting to send message to slack channel
2025-04-25 15:18:08,671 - __main__ - INFO - Attempting to fetch random cat image
2025-04-25 15:18:09,468 - __main__ - INFO - Fetched cat image URL: https://cdn2.thecatapi.com/images/a0i.jpg
2025-04-25 15:18:09,802 - __main__ - INFO - Message sucesfully sent to slack
```

Figure 34- Log File Showing Evidence of Successful File Detection and Alerts

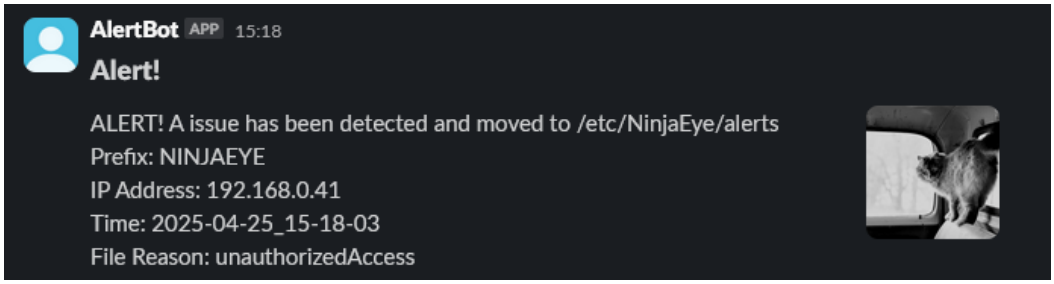

Figure 35- Image of Slack Alert Sent by Command

Chapter 5 Discussion

This chapter of the project aims to evaluate and discuss the methods and results of the project. Then, they will be compared to the project's aims and objectives, highlighting the positives and negatives that the project faced and discussing further where the project had shortcomings. This section will further discuss how the project could have been improved in future work.

5.1 Project Aims and Objectives

This section of the discussion aims to evaluate how the project aims and objectives were met throughout the project and discuss any shortcomings, highlighting the issues that caused these shortcomings. As mentioned previously, the aims and objectives of the project were as follows:

- This project aims to research, develop, and evaluate the effectiveness of a system capable of the detection and remediation of threats on remote servers. Within the aim of the project, there have been three key objectives identified.

- The first objective is that a method of detecting anomalies from observing the server will be researched and implemented.

- The second is that appropriate and automated remediation and alerts for detected threats will be implemented, allowing the system to run without user interaction.

- The third objective will be to evaluate the effectiveness and accuracy of the project, reporting on how successful the project was.

During testing for the intrusion detection system, the software successfully detected SSH connections and modifications to files; however, the software failed to detect modifications and access to files within the

monitored folders. This presents a software limitation that failed to meet the project's aims of successfully detecting threats on remote servers. After the project was completed, the code was reviewed, and it was discovered that there was a logic bug within the folder monitoring section of the code. The logic bug could have been fixed with further work on the project, but this was not possible due to the project's time limitations.

The project's second objective was that there would be automated steps of remediation and alerting that would allow the software to run without interaction. This objective was mostly met with Slack alerts; however, the project failed to take preventative measures. This highlighted an issue in the software's design: using log analysis. When using log analysis, the IDS no longer becomes a real-time monitoring system and cannot detect threats that are currently happening; therefore, it cannot prevent the attacks. This issue was highlighted in the project's literature review by authors Svacina *et al.* (2020) who described log analysis as a “post mortem” method of monitoring. This monitoring method directly contradicts the proactive purpose of IDPS and was a major issue for the project. If further work on the project were to be done, then a different monitoring technique would be implemented, such as process monitoring, which would allow the system to become a real-time system, allowing preventative actions to be taken. This issue was further elaborated during the project because pattern-based detection was used instead of anomaly-based detection. This meant that the first objective of the project was not fully achieved. The difference between anomaly-based detection and pattern-based detection is that anomaly-based detection relies on systems such as machine learning to set a baseline and detect if anything has changed; however, for this project the decision to use logs were used instead, contradicting the aims of the project and limiting the capabilities of the IDS.

5.2 MoSCoW Analysis

While the project had the overall aims and objectives, there was also a set of secondary objectives identified through a MoSCoW analysis done with the project client. The “Could Haves” and “Must Haves” of the projects were objectives that the client discussed and were used to shape the features included within the software. The full MoSCoW analysis can be seen below in Table 4 and Table 5.

Table 4- Description of the “Must Haves” From the Client

Must Haves	Description
Develop an Intrusion Detection System	Develop a functional Intrusion Detection system.
Compatibility with Ubuntu Linux	Allow the IDS to be fully compatible with the Ubuntu operating system.
Host Based System	IDS must be a host-based intrusion detection system with an agent per virtual machine.
Network Authentication Detection	Agents must be able to detect remote connections to the machine.
File modification Detection	Agents must be able to detect when files and folders are modified.

Table 5- Description of the “Could Haves” From the Client

Could Haves	Description
Simple Alerts	A Slack alert bot that could alert the security team of any intrusions.
Simple Deployment	The ability to deploy the software through the use of simple scripts
Minimum Ports	The client desires the IDS to use as few ports as possible for the system to function.

Central Management System	A central “command” system that could be used to monitor the IDS agents and handle alerts.

As shown in Table 4, the "Must Haves" of the project are as follows: The objectives set out by the client were required for the project to succeed. As discussed before, the project failed to detect when the contents of folders were modified, which goes against one of the "must haves" of the project. However, all other secondary objectives of the project were successful, including the project's optional "could haves". The software was fully compatible with Ubuntu Linux, as evidenced by the Ubuntu-focused file structuring. The project was also a HIDS-based system with a central management system used to send simple Slack alerts, fulfilling two optional objectives. The project was also simple to deploy, requiring the user to run the script as the software had an auto-setup feature for files and folders. It also contained a configuration file so that each agent node could be updated by modifying the config.json file, assisting ease of use and allowing flexibility within each node. However, a downside of this method is that if different machines used different configurations, an array of config files would need to be made depending on each machine's requirements. Another highlight of the software was the use of a minimal port design. The software was designed so that only two ports would be required to be opened for the software to be functional—port 22 For SCP and port 80 for AWS. However, if the client did not wish to use AWS, they could close port 80 on the agent nodes, and the software would still be functional due to the redundancy of AWS and SCP.

5.3 Data Transfer Redundancy

When designing the “commandConnection" function, AWS and SCP were implemented into the code. This would allow the software to transfer the alert file through either S3 or SCP. The SCP method allows data to be sent quickly through local networks easily; however, it has the downside of scalability and difficulty setting up. While testing the software, an issue was

that when the software would send data through SCP, the software would require a password to be entered. This would go against the project's objectives of no user interaction with the software. The method of mitigation that would be implemented for this would be for SSH Keys to be set up in the agent nodes. However, this has the downside of complicating the setup of the software and can present security integrity issues, such as the possibility for the keys to be stolen if an attack occurred, which can be a major issue when it comes to scalability and goes against the “could have” objective of simple deployment.

AWS was implemented into the “commandConnection” function. Using AWS would overcome the issues plaguing the SCP method of data transfer. AWS allowed the project to be scalable due to the elasticity of cloud services. AWS also allows users to easily move files worldwide rather than to a local network. A downside of S3, however, is the cost of storage. AWS can be a low-cost solution with minimal setup for developers, but the costs can quickly rise as the storage capacity goes up and the data retrieved goes up. The default cost per GB of data stored is 0.023$ per month. (*AWS*, 2025). To support offsetting storage costs, a method of deleting the data once downloaded was implemented. This would minimise storage. Keeping costs low. However, this has the downside of keeping the stored alerts on a local machine running the command node, an option that may not be viable for some users.

Using both AWS and SCP allows the system to have a redundant data transfer method. Although this is not one of the project's objectives, redundancy allows the software to keep alerting if something has gone wrong with AWS of SCP and positively impacts the project.

5.4 Evaluating the Testing Methodology

While the use of manual testing allowed a greater understanding of how the software functioned, the use of functional testing presented issues throughout the project. One such issue is that minor bugs would happen

and go unnoticed due to the functional software. This presented an issue within the project where minor bugs would go unnoticed after the feature was complete and, due to the testing methods, would not be discovered later. One example of this is the “ip_address” variable throughout the project. The variable was used as the SCP command node address, and the IP address told the command node where the file came from in the file name. Instead, two variables should have been named “scp_ip_address” and “source_address”. This issue was not identified until after the project was completed due to functional testing and if other testing methods were used, such as unit testing, where components are tested in isolation. This issue would likely not be present in the final product.

5.5 Future Work

While the project was a success, further work is needed to polish the software and simplify and improve methods within the software. One such example of this is the config.json file. The file allowed the file monitoring to be configured by the user to state what files and folders they would like to be monitored. However, this file could have been improved to allow other features to be configured. With modification to the SSH monitoring function, the file could easily allow the user to specify if they would like to monitor only SSH or if they would like to monitor other features that are stored in the auth.log, such as FTP and Sudo events. Another modification to the software would be to allow features to be turned on and off if they were not needed, helping to save system resources. This is another software feature that could be implemented within the config.json.

Furthermore, improvements to anomaly-based detection could be implemented, such as ML classifiers, process monitoring, and network analysis. These systems support the IDS and allow it to become a more real-time application, further allowing preventative methods to be added to the project. This would help support the project aims and objectives more effectively; however, it would come with individual trade-offs, such as potentially high false positive rates and network congestion. However, it

would be up to the end user to decide if the trade-offs are worth it for a higher detection rate and implementation of IPS into the software.

Chapter 6 Conclusion

This project set out to research, develop, and evaluate the effectiveness of a system capable of the detection and remediation of threats on remote servers. The project started with a literature review detailing core principles of Linux security, such as the Principle of Least Privilege (PoLP) and protection and detection.

Following this, there was a discussion regarding the techniques associated with Linux security, which included log analysis, network monitoring and intrusion detection and prevention systems (IDPS). The project was guided by the general techniques found and discussed in the literature review to provide a baseline for the project before developing secondary objectives. The objectives were set out using a MoSCoW analysis based on talks with the client. Afterwards, an intrusion detection system was created using an Agile development methodology to monitor files for changes and access. It also could detect remote SSH access on the machine before reporting it to a central command system. This would then send alerts to a Slack bot. Furthermore, the IDS also had AWS integration to allow a variety of possibilities for data transfer to assist with scalability for the project and to help simplify the software deployment. The IDS was then tested and evaluated using manual functional testing, which would reveal that the folder monitoring feature was not functional, though the SSH and file monitoring were.

Finally, the system focused on detection and alerting, with automated remediation not being implemented due to the design options such as the use of log analysis making it not real time and due to this, preventative actions could not be taken.

List of References

Academy, E. (2023) ‘What is the purpose of the “/etc” directory in the Linux file system?’, *EITCA Academy*, 5 August. Available at: https://eitca.org/cybersecurity/eitc-is-lsa-linux-system-administration/linux-filesystem/filesystem-layout-overview/examination-review-filesystem-layout-overview/what-is-the-purpose-of-the-etc-directory-in-the-linux-file-system/ (Accessed: 23 April 2025).

Al-Muntaser, B., Mohamed, M.A. and Tuama, A.Y. (2023) ‘Real-Time Intrusion Detection of Insider Threats in Industrial Control System Workstations Through File Integrity Monitoring’, *International Journal of Advanced Computer Science and Applications*, 14(6). Available at: https://doi.org/10.14569/IJACSA.2023.0140636.

Ban, T. *et al.* (2021) ‘Combat Security Alert Fatigue with AI-Assisted Techniques’, in *Cyber Security Experimentation and Test Workshop*. *CSET ’21: Cyber Security Experimentation and Test Workshop*, Virtual CA USA: ACM, pp. 9–16. Available at: https://doi.org/10.1145/3474718.3474723.

Beizer, B. (1995) *Black-box testing: techniques for functional software and systems* testing. USA: John Wiley & Sons, Inc.

Bhushan, B., Sahoo, G. and Rai, A.K. (2017) ‘Man-in-the-middle attack in wireless and computer networking — A review’, in *2017 3rd International Conference on Advances in Computing,Communication & Automation (ICACCA) (Fall)*. *2017 3rd International Conference on Advances in Computing,Communication & Automation (ICACCA) (Fall)*, pp. 1–6. Available at: https://doi.org/10.1109/ICACCAF.2017.8344724.

Depren, O. *et al.* (2005) ‘An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks’, *Expert Systems with Applications*, 29(4), pp. 713–722. Available at: https://doi.org/10.1016/j.eswa.2005.05.002.

Desmond, B. *et al.* (2008) *Active Directory: Designing, Deploying, and Running Active Directory*. O’Reilly Media, Inc.

Despa, M.L. (2014) ‘Comparative study on software development methodologies.’, *Database systems journal*, 5(3).

Ell, M. (2024) *Cyber security breaches survey 2024*, *GOV.UK*. Available at: https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2024/cyber-security-breaches-survey-2024 (Accessed: 3 March 2025).

Foulds, I. (2023) *Implementing Least-Privilege Administrative Models*. Available at: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/implementing-least-privilege-administrative-models (Accessed: 24 March 2025).

Fuchsberger, A. (2005) ‘Intrusion Detection Systems and Intrusion Prevention Systems’, *Information Security Technical Report*, 10(3), pp. 134–139. Available at: https://doi.org/10.1016/j.istr.2005.08.001.

George, D.A.S. (2024) ‘When Trust Fails: Examining Systemic Risk in the Digital Economy from the 2024 CrowdStrike Outage’, *Partners Universal Multidisciplinary Research Journal*, 1(2), pp. 134–152. Available at: https://doi.org/10.5281/zenodo.12828222.

Isaiah, A. (2025) *Monitoring Linux Authentication Logs: A Practical Guide | Better Stack Community*, *Better Stack*. Available at: https://betterstack.com/community/guides/logging/monitoring-linux-auth-logs/ (Accessed: 11 April 2025).

Jero, S. *et al.* (2021) ‘Practical Principle of Least Privilege for Secure Embedded Systems’, in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 1–13. Available at: https://doi.org/10.1109/RTAS52030.2021.00009.

Kenlon, S. (2021) *What is a config file? | Opensource.com*. Available at: https://opensource.com/article/21/6/what-config-files (Accessed: 23 April 2025).

Kenlon, S. (2022) *Linux superuser access, explained*, *Red Hat*. Available at: https://www.redhat.com/en/blog/linux-superuser-access (Accessed: 23 March 2025).

Kent, K. and Souppaya, M.P. (2006) *Guide to computer security log management*. 0 edn. NIST SP 800-92. Gaithersburg, MD: National Institute of Standards and Technology, p. NIST SP 800-92. Available at: https://doi.org/10.6028/NIST.SP.800-92.

Kim, G.H. and Spafford, E.H. (1994) ‘The design and implementation of tripwire: a file system integrity checker’, in *Proceedings of the 2nd ACM Conference on Computer and communications security - CCS ’94*. *the 2nd ACM Conference*, Fairfax, Virginia, United States: ACM Press, pp. 18–29. Available at: https://doi.org/10.1145/191177.191183.

Kumar, N., Angral, S. and Sharma, R. (2014) ‘Integrating Intrusion Detection System with Network monitoring’, 4(5).

Loscocco, P.A. *et al.* (2007) ‘Linux kernel integrity measurement using contextual inspection’, in *Proceedings of the 2007 ACM workshop on Scalable trusted computing*. *CCS07: 14th ACM Conference on Computer and Communications Security 2007*, Alexandria Virginia USA: ACM, pp. 21–29. Available at: https://doi.org/10.1145/1314354.1314362.

Lundh, F. (2001) *Python Standard Library*. O’Reilly Media, Inc.

Mosteiro-Sanchez, A. *et al.* (2020) ‘Securing IIoT using Defence-in-Depth: Towards an End-to-End secure Industry 4.0’, *Journal of*

Manufacturing Systems, 57, pp. 367–378. Available at: https://doi.org/10.1016/j.jmsy.2020.10.011.

Motiee, S., Hawkey, K. and Beznosov, K. (2010) 'Do windows users follow the principle of least privilege?: investigating user account control practices', in *Proceedings of the Sixth Symposium on Usable Privacy and Security*. *SOUPS '10: Symposium on Usable Privacy and Security*, Redmond Washington USA: ACM, pp. 1–13. Available at: https://doi.org/10.1145/1837110.1837112.

Nano - Community Help Wiki (no date) *Ubuntu*. Available at: https://help.ubuntu.com/community/Nano (Accessed: 25 April 2025).

Nittono, H. *et al.* (2012) 'The Power of Kawaii: Viewing Cute Images Promotes a Careful Behavior and Narrows Attentional Focus', *PLoS ONE*. Edited by K. Paterson, 7(9), p. e46362. Available at: https://doi.org/10.1371/journal.pone.0046362.

Radif, M.J. (2018) 'Vulnerability and Exploitation of Digital Certificates', in *2018 Al-Mansour International Conference on New Trends in Computing, Communication, and Information Technology (NTCCIT)*. *2018 Al-Mansour International Conference on New Trends in Computing, Communication, and Information Technology (NTCCIT)*, pp. 88–92. Available at: https://doi.org/10.1109/NTCCIT.2018.8681179.

Sánchez, A.R., Alvarado-Nava, O. and Martínez, F.J.Z. (2012) 'Network monitoring system based on an FPGA with Linux', in *2012 Technologies Applied to Electronics Teaching (TAEE)*. *2012 Technologies Applied to Electronics Teaching (TAEE)*, pp. 232–236. Available at: https://doi.org/10.1109/TAEE.2012.6235441.

Sewak, M., Sahay, S.K. and Rathore, H. (2023) 'Deep Reinforcement Learning in the Advanced Cybersecurity Threat Detection and Protection', *Information Systems Frontiers*, 25(2), pp. 589–611. Available at: https://doi.org/10.1007/s10796-022-10333-x.

Sharma, sagr (2024) *The /tmp directory in Linux: What You Should Know*, *linuxhandbook*. Available at: https://linuxhandbook.com/tmp-directory/ (Accessed: 13 April 2025).

Svacina, J. *et al.* (2020) 'On Vulnerability and Security Log analysis: A Systematic Literature Review on Recent Trends', in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. *RACS '20: International Conference on Research in Adaptive and Convergent Systems*, Gwangju Republic of Korea: ACM, pp. 175–180. Available at: https://doi.org/10.1145/3400286.3418261.

Svoboda, J., Ghafir, I. and Prenosil, V. (2015) 'Network Monitoring Approaches: An Overview', *International Journal of Advances in Computer Networks and Its Security– IJCNS*, 5, pp. 88–93.

Tidy, J. (2024) *CrowdStrike IT outage affected 8.5 million Windows devices, Microsoft says*, *BBC News*. Available at: https://www.bbc.com/news/articles/cpe3zgznwjno (Accessed: 3 March 2025).

Verma, P. (2023) *10 Best Practices for Logging in Python | Better Stack Community*, *Better Stack*. Available at: https://betterstack.com/community/guides/logging/python/python-logging-best-practices/ (Accessed: 18 April 2025).

Wang, S.-Y. and Chang, J.-C. (2022) 'Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel', *Journal of Network and Computer Applications*, 198, p. 103283. Available at: https://doi.org/10.1016/j.jnca.2021.103283.

Wang, X. *et al.* (2024) 'Combating alert fatigue with AlertPro: Context-aware alert prioritisation using reinforcement learning for multi-step attack detection', *Computers & Security*, 137, p. 103583. Available at: https://doi.org/10.1016/j.cose.2023.103583.

Wang, Z.Q. and Zhang, D.K. (2012) 'HIDS and NIDS Hybrid Intrusion Detection System Model Design', *Advanced Engineering Forum*, 6–7, pp. 991–994. Available at: https://doi.org/10.4028/www.scientific.net/aef.6-7.991.

Appendices

Appendix A- Complete Node Code

```
import os
import datetime
import time
from dotenv import load_dotenv
import threading
import boto3
from boto3 import client
import logging
import subprocess
import json

load_dotenv()

##code below is resposible for setting up the logging system

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
handler = logging.handlers.RotatingFileHandler(
    "/etc/NinjaEye/logs/log.txt", maxBytes=1000000, backupCount=10
##rolling logs to save space
)
formatter = logging.Formatter("%(asctime)s - %(name)s -
%(levelname)s - %(message)s")
handler.setFormatter(formatter)
if not logger.hasHandlers():
    logger.addHandler(handler)



print(r"""
```

```
.-------------------------------------------------------.
|                                                       |
|                                                       |
|  _________   ___  _________       ___  _________      |
| |\   ____  \|\  \|\   ____ \     |\  \|\   __  \     |
| \ \  \\ \  \ \  \ \  \\ \  \    \ \  \ \  \|\  \    |
|  \ \  \\ \  \ \  \ \  \\ \  \  __\ \  \ \  \   __  \  |
|   \ \  \\ \  \ \  \ \  \\ \  \|\  \\_\  \ \  \ \  \  |
|    \ \__\\ \__\ \__\ \__\\ \__\ \_________\ \__\ \__\ |
|     \|__| \|__|\|__|\|__| \|__|\|_________|\|__|\|__| |
|                                                       |
|                                                       |
|                                                       |
|  _________       ___    ___ ________                  |
| |\  ___ \     |\  \  /  /|\  ___ \                    |
| \ \   __/|    \ \  \/  / | \   __/|                   |
|  \ \  \_|/__   \ \    / / \ \  \_|/__                 |
|   \ \  \_|\ \   \/  /  /   \ \  \_|\ \                |
|    \ \_______\__/  / /      \ \_______\               |
|     \|_______|\___/ /        \|_______|               |
|              \|___|/                                  |
|                                    V1.0               |
|                               Leon Anderson           |
'-------------------------------------------------------'


""")

def configsetup(path="/home/ubuntu/Desktop/config.json"):
    ##loads configuration json to assist file monitoring
    try:
        with open(path, "r") as file:
            config = json.load(file)
```

```
        logger.info("config.json loaded sucessfuly")
        return config

    except Exception as e:
        logger.error(f"Failed to load configuration: {e}")
        exit(1)

config = configsetup()

def firstsetup():
    ## function that is responsible for setting up the directories and files
needed for the program to run
    folderpath = "/etc/NinjaEye/"
    folders = ["logs", "logs/alerts", "file_compare"]
    file_paths = ["logs/log.txt"]

    ##creates the folders and files needed for the program to run
    try:

        for folder_name in folders:
            folder_path = os.path.join(folderpath, folder_name)
            os.makedirs(folder_path, exist_ok=True)
            logger.info(f"Created directory: {folder_path}")



        for file_name in file_paths:
            file_path = os.path.join(folderpath, file_name)
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            if not os.path.exists(file_path):
                logger.info(f"Creating file: {file_path}")
                with open(file_path, 'w') as f:
                    pass
```

```
        print(f"Directories and files set up successfully in {folderpath}.")
    except PermissionError:
        print("Permission denied: Please run the script with elevated
privileges.")
        exit(1)



def get_env_variable():
    ## function that loads the enviroment variables from the .env file
needed for scp
    try:
       ip_address = os.getenv('IP_ADDRESS')
       ssh_username = os.getenv('SSH_USERNAME')

       logger.info("loaded enviroment variables")
       return ip_address, ssh_username

    except Exception as e:
        logger.error(f"An error occurred when loading the enviroment
variables: {e}")
        return None

def commandConnection(ip_address, ssh_username, data,
filereason):
    ##main function used for connecting to command node using scp
and S3
    currentTime = datetime.datetime.now().strftime("%Y-%m-%d_%H-
%M-%S")
    filename = f"NINJAEYE:{ip_address}:{currentTime}:{filereason}"
    filepath = f"/etc/NinjaEye/logs/alerts/{filename}"

    ## Write "data" to file
    with open(filepath, "w") as f:
```

```
        f.write(data)
    logger.info(f"Created alert: {filepath}")

    upload_to_s3(filepath) ##uploads to aws
    subprocess.run(f"scp {filepath} {ssh_username}@{ip_address}:/tmp",
shell=True)
    logger.info(f"alert sent to {ssh_username}@{ip_address}:/tmp
through scp")
    return

def sshLog(): ## gathers ssh logs and places them into a compare file
        sshCmd = "grep sshd /var/log/auth.log >
/etc/NinjaEye/logs/ssh.txt"
        subprocess.run(sshCmd, shell=True)
        logger.info(f"ssh log created using command: {sshCmd}")
        sshCompare() ## calls the ssh compare function to compare the
logs

def sshCompare(): ## comparison function of ssh logs to predefined
ssh log
    filereason = "unauthorizedSSH"
    while True:

        sshCmd = "grep sshd /var/log/auth.log >
/etc/NinjaEye/logs/sshCompare.txt"
        subprocess.run(sshCmd, shell=True)
        logger.info(f"ssh log created using command: {sshCmd}")

        afterLog = open("/etc/NinjaEye/logs/sshCompare.txt")
        beforeLog = open("/etc/NinjaEye/logs/ssh.txt")

        logger.info("ssh comparison started")
```

```
        beforeLog_data = beforeLog.readlines()
        afterLog_data = afterLog.readlines()

        before_set = set(beforeLog_data)
        after_set = set(afterLog_data)

        differences = after_set - before_set ##diffrences between the two
files

        if differences:
            alert_data = ""
            for line in differences:
                logger.warning(f"ALERT! {line.strip()}")
                alert_data += line ##combines the alert data

                ##sends information to command
            commandConnection(ip_address, ssh_username, alert_data,
filereason)  ##call helped function too parse information to command
            updatedLog = open("/etc/NinjaEye/logs/ssh.txt", "w")
            updatedLog.writelines(afterLog_data)
            updatedLog.close()
            logger.info("ssh log /etc/NinjaEye/logs/ssh.txt updated with new
data")

        else:
            logger.info("No differences found in ssh logs.")

        afterLog.close()
        beforeLog.close()
        time.sleep(time_interval) ##checks after time interval set in
config.json
```

```
def fileCompare(path, label, time_interval):
    ##function responsible for comparing files/folders using os.stat

    filereason = "unauthorizedAccess"
    beforefile = f"/etc/NinjaEye/file_compare/{label}_before.txt"
    afterfile = f"/etc/NinjaEye/file_compare/{label}_after.txt"

    open(beforefile, "a").close()
    open(afterfile, "a").close()

    test =  os.stat(path) ## the file that is being accessed to test the function using nano
    logger.info("files are being checked using stat")

    with open(beforefile, "w") as file:
        file.write(f"{test.st_atime} : {test.st_mtime} \n")
    logger.info(f"{beforefile} modified wth updated timestamp")

    while True:

        monitoredfile = os.stat(path) ## the file that is being accessed to test the function using nano
        logger.info("files are being checked using stat")

        with open(afterfile, "w") as file:
            file.write(f"{monitoredfile.st_atime} : {monitoredfile.st_mtime} \n")
        logger.info(f"{afterfile} modified with updated timestamps")

        afterLog = open(afterfile) ## the file that has the current time the file was accessed
        beforeLog = open(beforefile)
```

```
        beforeLog_data = beforeLog.readlines()
        afterLog_data = afterLog.readlines()

        before_set = set(beforeLog_data)
        after_set = set(afterLog_data)

        differences = after_set - before_set

        if differences:
            alert_data = ""
            for line in differences:
                logger.warning(f"ALERT! {line.strip()}")
                alert_data += line #combines the alert data

                ##sends information to command
            commandConnection(ip_address, ssh_username, alert_data,
filereason) ##call helper function too parse information to command

        else:
            logger.info("No differences found in file compare.")

        afterLog.close()
        beforeLog.close()

        with open(beforefile, "w") as file:
            file.write(f"{monitoredfile.st_atime} : {monitoredfile.st_mtime}
\n")
        logger.info(f"{beforefile} modified with updated timestamps")

        time.sleep(time_interval)
```

```
def upload_to_s3(file_name, object_name=None): ##Modified code from AWS S3 Documentation:  Code examples for Amazon S3 using AWS SDKs - Amazon Simple Storage Service (no date). Available at: https://docs.aws.amazon.com/AmazonS3/latest/API/service_code_examples_s3.html (Accessed: 02 April 2025).
##function that uploads the alert file to S3 bucket using boto3

    Access_Key = os.getenv('AccessKey')
    Secret_Key = os.getenv('SecretAccessKey')
    BUCKET_NAME = os.getenv('BUCKET_NAME')
    REGION = os.getenv('REGION')

    if object_name is None:
        object_name = os.path.basename(file_name)

    # Initialize S3 client
    logger.info("Initializing S3 client")
    s3 = boto3.client(
        's3',
        aws_access_key_id=Access_Key,
        aws_secret_access_key=Secret_Key,
        region_name=REGION
    )

    try:
        logger.info(f"Uploading {file_name} to s3")
        s3.upload_file(file_name, BUCKET_NAME, object_name)

        return True

    except FileNotFoundError:
        print(" File not found")
        logger.error("File not found")
```

```
        return False

    except Exception as e:
        logger.error(f"Error uploading file: {e}")
        return False



if __name__ == "__main__":

    firstsetup()
    ip_address, ssh_username = get_env_variable()

    sshThread = threading.Thread(target=sshLog, daemon=True)  # Create a thread for the sshCompare function
    sshThread.start() ## Start the thread

    time_interval = config["monitored_files"]["time_interval"]

    for x in config["monitored_files"]["file_paths"]:
        label = x["filename"]
        path = x["filepath"]

        fileThread = threading.Thread(target=fileCompare, args=(path, label, time_interval), daemon=True)  # Create a thread for each name
        fileThread.start() ## Start the thread

    sshThread.join()  ##waits for both threads to finish before exiting the program.
    fileThread.join()
```

Appendix B- Complete Command Code

```
import logging.handlers
import os
import time
import shutil
from dotenv import load_dotenv
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError
import requests
import boto3
from boto3 import client
import threading

load_dotenv()

##code below is resposible for setting up the logging system
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
handler = logging.handlers.RotatingFileHandler(
    "/etc/NinjaEye/logs/log.txt", maxBytes=1000000, backupCount=10
##rolling logs to save space
)
formatter = logging.Formatter("%(asctime)s - %(name)s -
%(levelname)s - %(message)s")
handler.setFormatter(formatter)
if not logger.hasHandlers():
    logger.addHandler(handler)

print(r"""

.-------------------------------------------------------.
|                                                       |
|                                                       |
```

```
|  _________  ___  _________       ___  _________      |
| |\  ___  \|\  \|\  ___  \     |\  \|\  __  \     |
| \ \  \\ \  \ \  \\ \ \  \\ \  \    \ \  \ \  \|\  \    |
|  \ \  \\ \  \ \  \\ \ \  \\ \  \  __ \ \  \ \  __  \   |
|   \ \  \\ \  \ \  \\ \ \  \\ \  \|\  \\_\  \ \  \ \  \  |
|    \ \__\\ \__\ \__\\ \__\\ \__\ \_________\ \__\ \__\ |
|     \|__| \|__|\|__|\|__| \|__|\|_________|\|__|\|__|  |
|                                          |
|                                          |
|                                          |
|  _______      ___    ___ _______                 |
| |\  ___ \    |\  \  /  /|\  ___ \               |
| \ \  __/|   \ \  \/  / | \  __/|              |
|  \ \  \_|/__  \ \    / / \ \  \_|/__            |
|   \ \  \_|\ \  \/  /  /   \ \  \_|\ \           |
|    \ \_______\__/  / /     \ \_______\          |
|     \|_______|\___/ /       \|_______|          |
|           \|___|/                        |
|                                  V1.0    |
|                            Leon Anderson     |
'-----------------------------------------------------'


""")

def firstsetup():
    ## Setup required directories and files.
    folderpath = "/etc/NinjaEye/"
    folders = ["logs", "logs/alerts"]
    file_paths = ["logs/log.txt"]

    try:

        for folder_name in folders:
```

```
            folder_path = os.path.join(folderpath, folder_name)
            os.makedirs(folder_path, exist_ok=True)
            logger.info(f"Created directory: {folder_path}")



        for file_name in file_paths:
            file_path = os.path.join(folderpath, file_name)
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            if not os.path.exists(file_path):
                logger.info(f"Creating file: {file_path}")
                with open(file_path, 'w') as f:
                    pass

        print(f"Directories and files set up successfully in {folderpath}.")
    except PermissionError:
        print("Permission denied: Please run the script with elevated
privileges.")
        exit(1)



def alertmonitoring():  ## Monitors for incoming files in tmp folder and
alerts the user if there is one then stores the file
    CHANNEL_ID = os.getenv('CHANNEL_ID')
    SLACK_BOT_TOKEN = os.getenv('SLACK_BOT_TOKEN')
    folderpath = "/tmp/" ##where files are stored

    ## Code belows splits the file name into sections and stores sets
them to "reason" variable to allow ease of passing in slack
    while True:
        for filename in os.listdir(folderpath):
            if filename.startswith("NINJAEYE"):
                source = os.path.join(folderpath, filename)
                destination = "/etc/NinjaEye/alerts"
```

```
            shutil.move(source, destination)
            print(f"ALERT! See {destination}/{filename} for more
information!")
            logger.info(f"Detected alert and moved to
{destination}/{filename}")
            parts = filename.split(':')
            if len(parts) >= 4:
                prefix = parts[0]
                ip_address = parts[1]
                current_time = parts[2]
                filereason = parts[3]
            else:
                prefix = "NINJAEYE"
                ip_address = "Unknown"
                current_time = "Unknown"
                filereason = "Unknown"

            reason = (f"ALERT! A issue has been detected and moved to
{destination}\n"
                      f"Prefix: {prefix}\n"
                      f"IP Address: {ip_address}\n"
                      f"Time: {current_time}\n"
                      f"File Reason: {filereason}")

            send_message(CHANNEL_ID, SLACK_BOT_TOKEN,
reason)
        time.sleep(5) ##sleep for performance


def get_cat_image():
    ## Code below helper function resposible for getting a random cat
image from the API and returning the URL to be passed into slack
```

```
    logger.info("Attempting to fetch random cat image")
    CAT_API_KEY = os.getenv('CAT_API_KEY')
    CAT_API_URL = os.getenv('CAT_API_URL')
    headers = {"x-api-key": CAT_API_KEY}
    response = requests.get(CAT_API_URL, headers=headers)
    if response.status_code == 200:
        immage_url = response.json()[0]["url"]
        logger.info(f"Fetched cat image URL: {immage_url}")
        return response.json()[0]["url"]
    return None

def send_message(channel, SLACK_BOT_TOKEN, reason):
    #function responsible for sending slack messages.

    logger.info(f"attempting to send message to slack channel")
    client = WebClient(token=SLACK_BOT_TOKEN)
    cat_image_url = get_cat_image()
    if not cat_image_url:
        cat_image_url =
"https://pbs.twimg.com/profile_images/625633822235693056/lNGUneL
X_400x400.jpg"  # Fallback image
        logger.info(f"Failed to fetch cat image, using fallback image
{cat_image_url}")

    ##blocks used to format the slack message.
    blocks = [
        {
            "type": "header",
            "text": {
                "type": "plain_text",
                "text": "Alert!",
                "emoji": True
            }
```

```
        },
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": f"{reason}\n"



            },
            "accessory": {
                "type": "image",
                "image_url": cat_image_url,
                "alt_text": "Random cat image"
            }
        },
    ]

    try:
        response = client.chat_postMessage(channel=channel,
blocks=blocks, text="NinjaEye Alert!")
        logger.info(f"Message sucesfully sent to slack")

    except SlackApiError as e:
        logger.error(f"Error sending message to slack:
{e.response['error']}")

def monitor_S3(): ##Modified code from AWS S3
Documentation:  Code examples for Amazon S3 using AWS SDKs -
Amazon Simple Storage Service (no date). Available at:
https://docs.aws.amazon.com/AmazonS3/latest/API/service_code_exa
mples_s3.html (Accessed: 02 April 2025).

    Access_Key = os.getenv('AccessKey')
```

```
    Secret_Key = os.getenv('SecretAccessKey')
    BUCKET_NAME = os.getenv('BUCKET_NAME')
    REGION = os.getenv('REGION')

    ##code itterates over each function in the bucket and downloads it to the /tmp folder then deletes it to save storage.
    while True:
        try:
            logger.info("Attempting to access S3 bucket")
            client = boto3.client(
                's3',
                aws_access_key_id=Access_Key,
                aws_secret_access_key=Secret_Key,
                region_name=REGION
            )

            bucketItems = client.list_objects(
                Bucket=BUCKET_NAME
            )

            if 'Contents' not in bucketItems:
                logger.info("No files in bucket")

            else:
                logger.info("Files found on s3 bucket")
                for item in bucketItems['Contents']:
                    key = item['Key']
                    file_path = os.path.join("/tmp/", key)
                    logger.info(f"Downloading {key} to {file_path}")

                    client.download_file(BUCKET_NAME, key, file_path)
```

```
                client.delete_object(Bucket=BUCKET_NAME, Key=key)
##reccomended to delete the file after download to save space,
however optional and can be removed retention is needed.
                logger.info(f" {key} removed from S3.")

            time.sleep(5)  ##sleep for performance

        except client.exceptions.NoSuchBucket:
            logger.error("The specified bucket does not exist.")
            return False
        except client.exceptions.ClientError as e:
            logger.error(f"Client error: {e}")
            return False

        except Exception as e:
            logger.error(f"Error accessing S3 bucket: {e}")
            return False



if __name__ == "__main__":

    firstsetup()
    alertmonitoringThread = threading.Thread(target=alertmonitoring,
daemon=True)  ## Create a thread for the alertmonitoring function
    alertmonitoringThread.start() ## Start the thread
    monitor_S3Thread = threading.Thread(target=monitor_S3,
daemon=True)  ## Create a thread for the monitor_S3 function
    monitor_S3Thread.start() ## Start the thread

    alertmonitoringThread.join() ## waits for both threads to finish before
exiting the program.
    monitor_S3Thread.join()
```