VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE

Information Technology 4'th year Bachelor's Thesis

Simulation and Detection of Cybersecurity Offenses Using a SIEM Framework

Kibernetinio saugumo pažeidimų simuliacija ir aptikimas, remiantis SIEM sistemos karkasu

Done by:
Evelina Kabišaitytė signature

Supervisor:
Lekt. Eduardas Kutka

Vilnius
2026

List of Figures

Contents

Keywords

Virtual machine - in its simplest form, a virtual machine (VM) is a digital replica of a physical computer. Virtual machines can run programs and operating systems, store data, connect to networks, and perform other computing functions.

Ansible - is an open-source IT automation engine that automates provisioning, configuration management, application deployment, orchestration, and many other IT processes.

OpenNebula - is an open-source cloud computing platform for managing heterogeneous datacenter, public-cloud, and edge-computing infrastructure resources.

Tcpdump - a command-line packet analyzer for network traffic capture.

PCAP - or packet capture, refers to the process of intercepting and making a record of data packets, which are the units of data and payload that make up network traffic.

SIEM (Security Information and Event Management) - refers to systems that collect, aggregate, and analyze security-relevant data from multiple sources, including system logs, network traffic, and packet captures.

Offense - in SIEM systems, an offense is a correlated security threat created when one or more security events and/or network flows indicate potential malicious activity.

Abstract

This project simulates three types of cyber security offenses on two virtual machines. It is designed as a simplified approach to cyberattack analysis, using the SIEM (Security Information and Event Management) framework, and allows the user to experience how a SIEM operates from start to finish: beginning with threat generation, followed by analysis of the data produced and concluding with offense generation correlated with that threat.

Machine 1 functions as the threat actor "Attacker" and Machine 2 as the target "Victim", while a third virtual machine, referred to as the "Master," executes the scripts and collects network data packets. The virtual machines are deployed within the OpenNebula environment, which enables two tcpdump instances to run simultaneously on separate machines to gather network traffic data.

From a menu displayed on the Master machine, the user can select one of three different types of offenses inspired by the most common offense alerts found in SIEM tools: SSH brute force, a large number of TCP connections to external internet resources, and a malicious file detected but not deleted.

The network traffic generated during the attack is captured from both machines and saved in PCAP (Packet Capture) format. These PCAP files are then processed by a detection tool that identifies the unique patterns associated with each offense type based on a rule system. The analyzed results are presented to the user in an offense-style format, following the traditional layout of SIEM tool offenses.

Santrauka

Šis projektas imituoja tris kibernetinių atakų tipus dviejose virtualiose mašinose. Jis sukurtas kaip supaprastintas kibernetinių atakų analizės metodas, pasitelkiant SIEM sistemą, ir leidžia naudotojui patirti, kaip SIEM veikia nuo pradžios iki pabaigos - pradedant grėsmės generavimu, tęsiant sugeneruotų duomenų analize ir baigiant su ta grėsme koreliuoto incidento generavimu.

1-oji mašina veikia kaip grėsės vykdytojas („Atakuotojas“), 2-oji, kaip tikslas („Auka“), o trečioji virtualioji mašina, vadinama pagrindinė virtualioji mašina, vykdo scenarijus ir fiksuoja tinklo paketų duomenis. Virtualios mašinos sukurtos OpenNebula aplinkoje, kuri leidžia tuo pačiu metu paleisti dvi tcpdump operacijas skirtingose mašinose tinklo srautui fiksuoti.

Iš pagrindinės virtualiosios mašinos rodomo meniu naudotojas gali pasirinkti vieną iš trijų atakų tipų, įkvėptų dažniausiai pasitaikančių incidentų įspėjimų SIEM įrankiuose: SSH prisijungimų bandymų (brute-force) ataka, didelis kiekis TCP jungčių į išorinius interneto resursus, ir aptiktas, bet neištrintas kenksmingas failas.

Atakos metu sugeneruotas tinklo srautas fiksuojamas abiejose mašinose ir saugomas PCAP (Packet Capture) formatu. Šie PCAP failai vėliau apdorojami detekcijos įrankiu paremtu taisyklėmis, kuris identifikuoja kiekvienam atakos tipui būdingus bruožus. Analizuoti rezultatai pateikiami naudotojui incidento formatu, atitinkančiu tradicinę SIEM įrankių incidentų struktūrą ir logiką.

Introduction

Today, information technology has become a fundamental aspect of modern life, however, like any technology, it can be exploited, abused, or used with malicious intent. As organizations increasingly rely on interconnected networks and cloud-based infrastructures, the number and complexity of cyberattacks have grown significantly. Recent statistical evidence [1] shows the rapid growth and severity of cybercrimes in the United States. Reported cases of fraud, identity theft, and other cyberattacks increased dramatically between 2010 and 2021, with identity theft alone rising by over 470% and other cyber-attacks increased by 285.76%. Cybercrimes have become just as serious as traditional crimes, posing risks to both individuals and organizations. Detecting these attacks has therefore become a big challenge in maintaining the security of information systems. To address such challenges, SIEM systems have been developed to assist in identifying and responding to potential threats.

This thesis presents a practical simulation of cyber security and their detection through the use of a SIEM framework and virtual machines. The project aims to provide a more educational approach to understanding how SIEM systems operate. From the initial generation of a threat to its detection and the creation of correlated offenses. The simulation consists of three virtual machines: an Attacker machine responsible for executing the attacks, a Victim machine that serves as the target, and a Master machine that coordinates the process and captures network traffic for analysis. The system simulates three common categories of attacks inspired by real SIEM alerts to provide a realistic representation of real-world threats: an SSH brute-force attack, one of the most elementary yet common attack types in the cybersecurity field, a large number of TCP connections to external internet resources, representing seemingly mundane network behavior that can quickly escalate into a serious security concern and the detection of a malicious file that is not removed, a more classical scenario illustrating how SIEM systems integrate with tools such as antivirus software.

The captured network traffic is retrieved and then processed using a detection tool created to identify the patterns of each offense type. The processed data is then presented in an offense style format, following the structure used in SIEM tools.

The main objective of this project is to demonstrate, in a safe and controlled environment, how different types of cyberattacks can be detected and then correlated using SIEM-inspired methods. The system serves both as a learning tool for understanding network based threat detection and as a foundation for further research into automated anomaly detection techniques using machine learning

In this paper, the first chapter is dedicated to the analysis of related work. The second chapter provides an overview of the frameworks of the main components: the overall project framework, the offense simulation framework, and the detection algorithm framework. The third chapter delves deeper into the project, reviewing all components in detail, starting with the setup of the master machine, automatic VM creation, and the offenses, which describe the logic behind choosing the three specific attack types. Then an analysis of the three offenses, explaining the methods used for their simulation, followed by data retrieval, detection tool, and rules management, which examines the possibilities available to the user with the detections and results. The final chapters present the conclusions and recommendations, as well as a roadmap for future research.

1 Analysis

This project aims to create a practical simulation-based approach to understanding the processes behind cyberattack detection and analysis using the SIEM framework. By automating the setup and execution of controlled attacks in a virtualized environment, the system demonstrates how real-world security threats can be generated, detected, and correlated into offenses.

Some methodologies and technologies in this project are inspired by the reviewed literature. Foundational studies such as "A Survey of Network Anomaly Detection Techniques" [2] and "Machine Learning Solutions in Network Traffic Classification" [3] make the theoretical and also practical frameworks for analyzing network traffic and detecting anomalies.These works show the need for preprocessing heterogeneous network data and applying classification. Such principle was adopted in this project's approach to PCAP data processing and anomaly recognition.

Futher on, SIEM-focused works such as "Improving SIEM Alert Metadata Aggregation with a Novel Kill-Chain Based Classification Model" [4] inspired the detection algorytim system this project reflects those principles by replicating the SIEM process for event detection and offense generation from gathered data. and "Going Beyond the Ordinary: How SIEM Does Incident Detection, 2020 [5] was used to reflect the most common offenses. The inclusion of documentation sources such as IBM QRadar SIEM Documentation [6] and YARA Documentation [7] were helpful for threat generation. Yara was used for malicious object detected but not deleted simulation.

The paper on "Raw Network Traffic Data Preprocessing and Preparation for Automatic Analysis" [8]: highlights a crucial but often overlooked phase in the cybersecurity workflow: preparing raw network data for effective analysis.

The study on "Mitigation of Multi-vector Network Attacks via Orchestration of Distributed Rule Placement" [9] showcasw the importance of coordinated defense mechanisms when anomalies have been detected.

Finally, studies such as "Classification and Analysis of Malicious Code Detection Techniques Based on the APT Attack" [10] and "Shielding Web Applications Against Cyber-Attacks Using SIEM "[11] show the persistence of modern cyber threats, bettering the educational importance of practical simulations. These works showcase the necessity of adaptive detection mechanisms that can handle a variety of attack types, a core objective achieved in this project through the simulation of three distinct, SIEM-inspired cyberattacks with the ability to change and mold SIEM rules to be able to take in new simulations for detection.

Related work

J. H. Mohiuddin Ahmed et al. [2] provide an analysis of various approaches used for detecting anomalies in network traffic. It categorizes anomaly detection methods into four main groups: classification-based, statistical, information-theoretic, and clustering-based techniques. The paper shows the principles and effectiveness of each method in identifying network intrusions and shows their respective advantages and limitations. It also discusses the challenges related to the datasets used for training and evaluating these detection systems, emphasizing the need for high-quality and representative data in improving anomaly detection accuracy.

F. Pacheco et al. [3] provide insight on the steps to achieve traffic classification by using ML techniques. The main aim is to understand and to identify the procedures followed by the existing works to achieve their goals. As a result, this survey paper finds a set of trends derived from the analysis performed on this domain; in this manner, the authors expect to outline future directions

for ML-based traffic classification.

B. D. Bryant et al. [4] present a hybrid kill-chain framework that is implemented as a novel configuration of SIEM software to address the limitations of current intrusion detection systems. The main objective is to enhance the accuracy and relevance of security alerts by improving SIEM correlation and data normalization. To achieve this, a new log ontology was developed to standardize data collected from multiple security sensors in alignment with modern threat research. Based on this ontology, new SIEM correlation rules were designed and tested against a baseline configuration. The evaluation demonstrated that the proposed configuration improves detection rates, produces more descriptive alerts, and reduces false positives compared to existing system

D. K. M. Dimolianis et al. [9] propose a framework for mitigating detected multi-vector anomalies in typical enterprise networks via the distribution of Access Control Rules. Its distributed, non-proprietary approach takes advantage of the capabilities offered by all devices along an attack path enhancing their mitigation potential.

for Automatic Analysis B. Alothman et al. [8] present an explanation of several steps required to make sure the data is in good shape for analysis and automatic detection of malicious traffic. The steps are explained in a tutorial like manner and demonstrated by being executed to analyse a publicly available network traffic dataset that contains safe and malicious data. The steps and analysis illustrate that the procedure helps in making tasks such as automatic classification and clustering easy.

A. Yushko et al. [11], examine the security challenges of modern web applications, focusing on the increasing risks associated with rapid online service deployment. Web applications have become essential platforms for communication, commerce, and information exchange, yet many are implemented without adequate security evaluation, exposing organizations to threats that compromise data integrity, reliability, and availability. Most web application incidents result from vulnerabilities caused by misconfiguration, improper data handling, weak authentication, and insufficient access control. To address these issues, the study explores the application of SIEM systems as an effective approach to web application protection. The findings highlight SIEM's potential to enhance situational awareness and improve the overall resilience of web applications against cyberattacks.

[7], introduce YARA as a tool designed to assist malware researchers in identifying and classifying malware samples. The tool enables the creation of descriptive rules that define malware families based on textual or binary patterns. Each rule consists of a set of strings and logical conditions that determine whether a given file matches the defined characteristics. The documentation gives examples demonstrating how YARA rules can be crfeated and applied to detect malicious files, from anything like simple string matches or logical expressions.

IBM corporation presents [6], IBM QRadar SIEM as an advanced Security Information and Event Management platform designed to collect, analyze, and correlate security event data from diverse sources across an organization's infrastructure. The documentation set provides an overview of QRadar's system architecture, deployment procedures, configuration options, and administration capabilities.

Positive Technologies (PT Security) present a report [5] that showcases the results of 23 pilot projects involving the MaxPatrol SIEM platform, conducted in the second half of 2019 and early 2020. The study shows how information from various sources can be aggregated and analyzed through a SIEM system to detect corporate security incidents. It shows SIEM's role not only in puting the data in to one central area event data but also in automating incident detection and delivering timely alerts to security teams. The report also discusses the system's ability to identify a varianty of incidents, including policy violations, malware infections, anomalous user behavior,

and suspected attacks.

K. Lee et al. [10] provide a review of malicious code detection methods in the context of Advanced Persistent Threats (APTs). The study analyzes real-world APT attack cases to analyze the corresponding attack scenarios and techniques used by adversaries. Based on this , the authors put the malicious code detection approaches into several groups, including security management systems, pattern-based detection, heuristic-based detection, reputation-based detection, behavior-based detection, virtualization-based detection, anomaly detection, and data analysis-based detection such as big data and machine learning approaches.

2 Framework

2.1 Project Framework

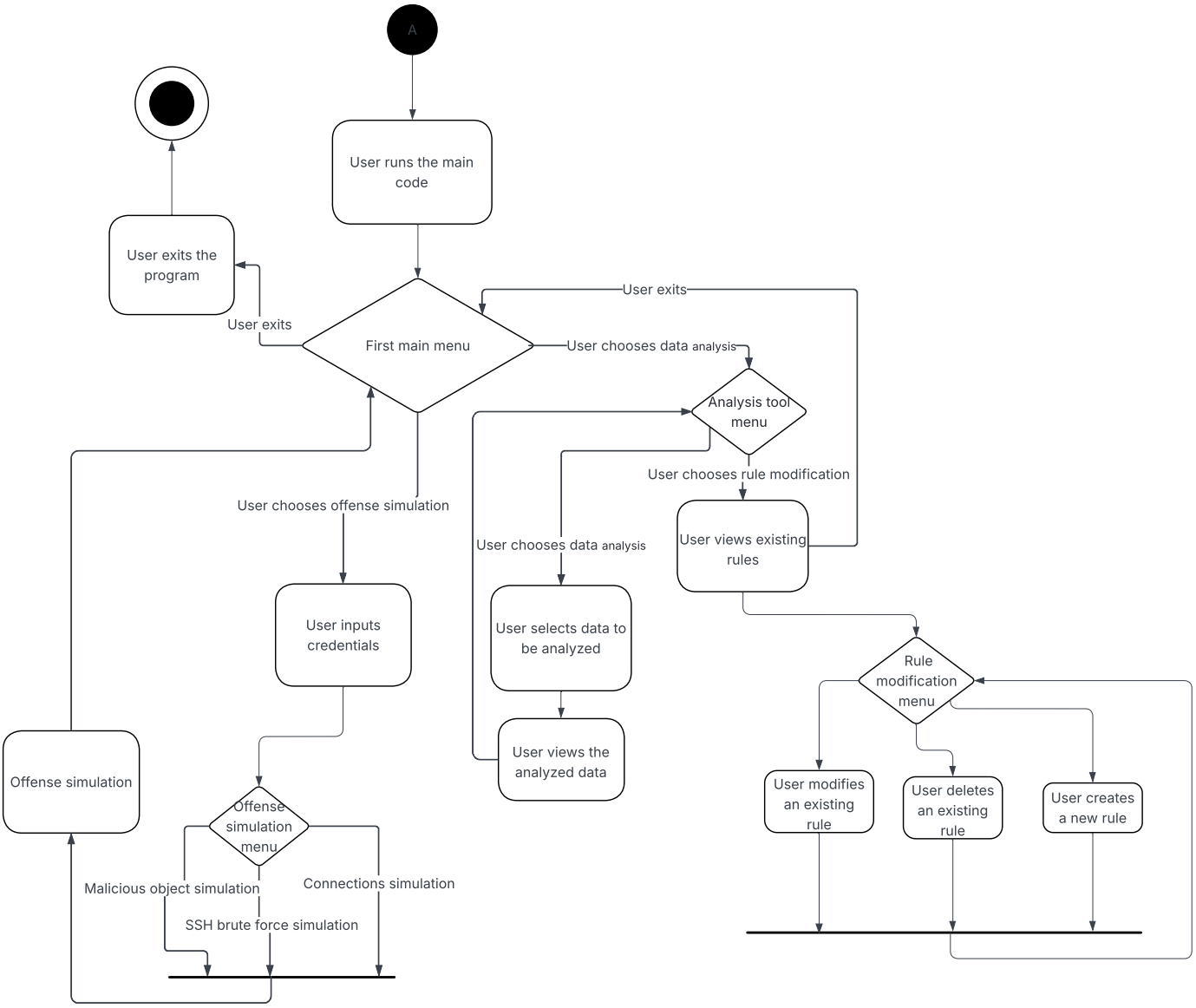

Figure 1. A UML activity diagram from the user's perspective

This overview examines the main components of the project and their structure (Figure 1) from the view of a user, with a more detailed analysis of each component provided in the subsequent chapters. The project is divided into two primary parts: offense generation and offense detection, drawing inspiration from a SIEM tool.

The project starts with a main menu that allows the user to choose between offense simulation, data analysis, or exiting the program overall. When the user selects offense simulation, an additional menu is displayed, presenting three cybersecurity offense simulations: SSH brute force, malicious file detected but not deleted, and a large number of TCP connections to an external internet resource. After one of the simulations is selected, the system requests the user to enter their credentials in order to create virtual machines within the OpenNebula environment. For project testing, credentials from the Vilnius University Faculty of Informatics and Mathematics were used.After

the user enters their credentials, virtual machines are created using OpenNebula's tools. The Bash controller instantiates two Debian 12 virtual machines. The sequence of creation depends on the selected simulation, whether the attacker or the victim machine is created first is based on the type of simulation. During the offense simulation, two data packets in PCAP format are generated: one containing attacker network information and the other containing victim network information. These files are then sent to the master machine. Upon completion of the simulation, the user may repeat the same simulation, choose a different one, or proceed to data analysis. The analysis menu consists of two main options: data analysis and rule modification. When the user selects data analysis, a list of available PCAP files is displayed. The data is then analyzed using either single-packet or stateful rules, depending on whether an offense consists of a single event or multiple correlated events.The analysis is based on SIEM principles. The detection tool performs detection based on predefined rules, which in full-scale SIEM systems may be accompanied by machine learning or other algorithms. However, for this project, the system is intentionally left simple to focus on core functionality, specifically rule matching. Rule management starts with its dedicated menu, allowing rules to be added, edited, or deleted. When a rule is matched, an offense is generated and displayed in a SIEM-style dashboard. After reviewing the detected offenses, the user can then proceed to analyze different data, or reanalyze the same data.

2.2 Offense Simulation Framework

This overview describes the offense simulations and their main components (Figure 2), with a more detailed analysis provided in the following chapters. The order of execution of the simulations is not significant.

The SSH brute force simulation begins with the creation of the victim virtual machine. After which tcpdump and the OpenSSH server are installed. The victim then creates a target user account with a password for the attacker to target. Afterwards, tcpdump is left to run for an extended period to capture the brute-force activity. Following this, the attacker virtual machine playbook starts. On the attacker machine, tcpdump and sshpass are installed. The attacker starts network data capture and performs a brute-force attack against the victim with 25 authentication attempts. Once the simulation is complete, both virtual machines transfer their captured data to the master machine.

The large number of TCP connections to an external internet resource simulation begins with the creation of the attacker virtual machine. After a 45-second boot period, tcpdump, Python 3, and Apache2 are installed. The attacker starts a Python-based HTTP server and creates a script designed to force the victim to establish multiple connections to the attacker. Packet capture is then initiated using tcpdump. The victim installs and runs tcpdump, downloads the attacker's script, and executes it, generating a large number of TCP connections. After the simulation concludes, both machines send their captured data to the master machine.

The malicious file detected but not deleted simulation begins with the creation of the attacker virtual machine. After completing the standard boot process, tcpdump and Apache2 are installed.The attacker starts packet capture and writes an EICAR test file to the web server's root directory. The victim virtual machine is then created, completes its boot process, installs tcpdump, and creates a YARA rule designed to detect the EICAR test file.The victim initiates packet capture and performs a YARA scan, which results in a malware detection alert. Following the detection, data from both virtual machines is collected and transferred to the master machine.

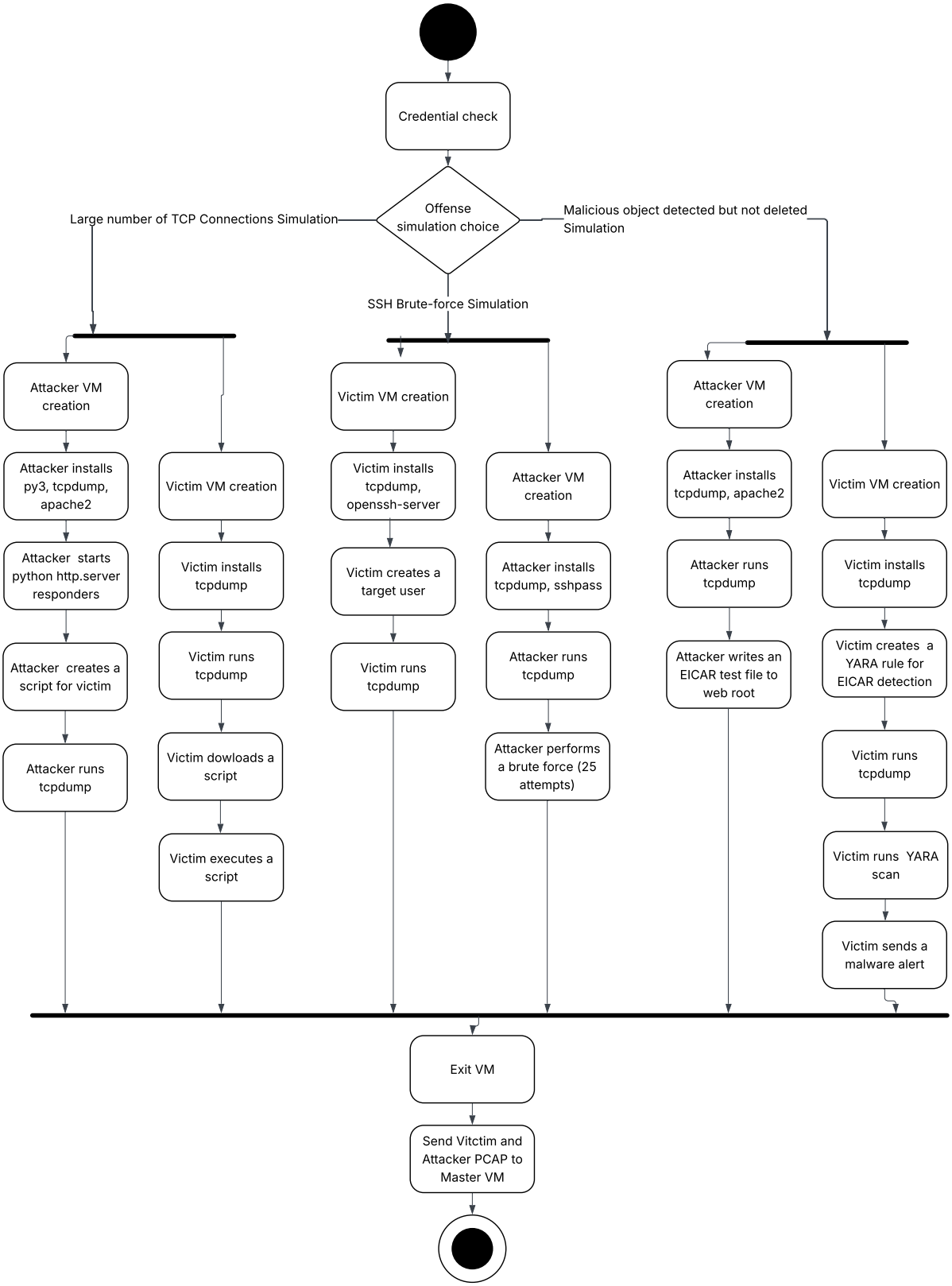

Figure 2. A UML activity diagram of the simulated offense processes

2.3 Detection Algorithm Framework

This project's offense detection system uses a generic framework for network anomaly detection, specifically the framework described by Ahmed et al. [2] in their article "A Survey of Network Anomaly Detection Techniques" (2016).

The project's detection algorithm is designed based on this framework (Figure 3) to simulate a primitive network anomaly detection tool, taking inspiration from the functionality of a SIEM system. It processes input data derived from the packet captures of simulated threats and converts it into a unified format. As noted by Ahmed et al. [2], "The input data requires processing because the data are of different types; for example, IP addresses are hierarchical, whereas the protocols are categorical and port numbers are numerical in nature."

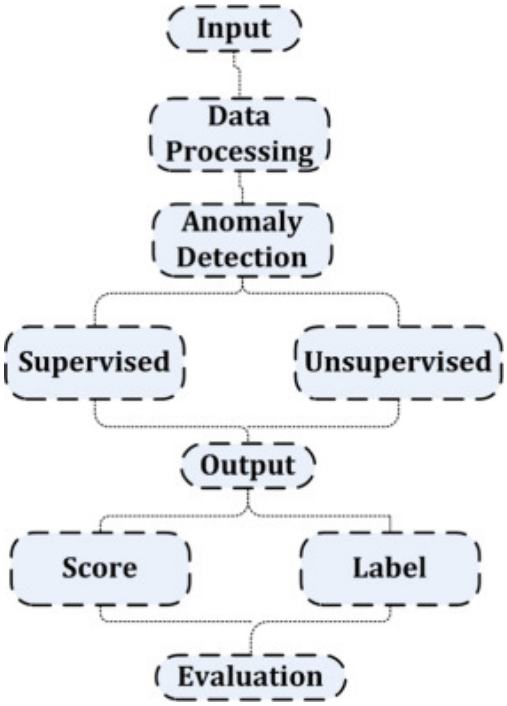

Figure 3. Generic framework for network anomaly detection diagram from the article by Ahmed et al. "A Survey of Network Anomaly Detection Techniques" (2016)

After preprocessing, anomaly detection is performed using a predefined set of rules that determine what constitutes an anomaly. In this project, three distinct scenarios are defined within the algorithm and their defining features:

- SSH brute-force attack - 10 distinct connections to port 22, made in 30 seconds.
- Large number of TCP connections to an external ineternet recourse - 9 HTTP GET requests to the external resource
- Malicious file detected but not deleted - a POST message that the victim sends back to the attacker when the YARA rule detection triggers

After preprocessing, anomaly detection is performed using a predefined set of rules that determine what constitutes an anomaly. In this project, three distinct scenarios are defined within the algorithm and their defining features:

- SSH brute-force attack - 10 distinct connections to port 22, made in 30 seconds.
- Large number of TCP connections to an external ineternet recourse - 9 HTTP GET requests to the external resource
- Malicious file detected but not deleted - a POST message that the victim sends back to the attacker when the YARA rule detection triggers

The detection algorithm applies three predetermined rules corresponding to these specific offenses. If an event matches any of the defined rules, the system generates an offense this represents the “Anomaly Detection” step in the framework.

The terms “supervised“ and “unsupervised“ are used to explain how machine learning models are trained to identify suspicious behavior in network traffic. “Supervised methods“ use labeled data, that means the model is trained on examples that are already categorized as normal or anomalous. The system learns patterns from known “normal” and “attack” traffic and can then classify new network events. “Unsupervised methods“, use unlabeled data, meaning the model does not know in advance what is normal or malicious. Instead, it detects deviations from typical network behavior. In this project this is not used but it can be expanded to use machine learning in future research.

The output consists of an offense label (the event name) and a score. The evaluation step can be done either manually or automatically by an algorithm, depending on the stage and purpose of the analysis. In this project, evaluation is made by a user, who evaluates the generated offense and its score.

3 Project

3.1 The Master Machine

The initial execution of the script requires a third, manually configured Master machine. The purpose of this machine is to work as a functional control workstation capable of interacting with the VU MIF OpenNebula cloud environment and executing Ansible playbooks. The machine also needs at least 15 GB of disk space to store the PCAP files and other required files.

Objective: Create and configure a third virtual machine that functions as the Master controller, installed with the OpenNebula command-line tools, Ansible, and the necessary SSH utilities for secure, passwordless management of remote virtual machines.
Tools and Technologies:

- **Ansible** is used to execute configuration and orchestration of playbooks.
- **OpenNebula CLI tools** is used to instantiate and manage the attacker and victim VMs within the cloud environment.
- **SSH, SCP, and SFTP** is used to provide secure access, remote command execution, and retrieval of PCAP files from the simulation VMs.
- **Bash** the controller is implemented as a shell script executed directly on the Master machine.
- **Python 3.8** is required to run the analysis and detection script.

- **scapy, pandas** PCAP parsing, event aggregation and rule evaluation for the detection algorithm.

Methodology:
A Master VM is instantiated on the OpenNebula environment used for the simulations. Once the Master VM is available, it must be prepared through few steps:

1. Install required packages: Ansible, OpenNebula CLI tools, Python,scapy, pandas and SSH utilities.

2. Configure the OpenNebula authentication credentials to enable non-interactive use by the controller script.

This setup is the preparation that the Master machine can configure, and manage the attacker and victim VMs, which will perform the simulations.

3.2 Automatic VM Creation

The master machine is the machine that carries the main script, it coordinates the attack simulations between the attacker and the victim. It automates the creation of virtual machines, the execution of three simulations, and the fetching of packet captures (PCAPs) from both VMs it also houses the detection algorithm which analyzes data . It is the core of the system, handling all virtual machine management and Ansible playbook execution.

Objective: Without user interaction and fully automatically create two VMs (attacker and victim) in the VU MIF OpenNebula cloud, execute the selected simulation, capture PCAP files, and retrieve them to the master machine via SCP for later analysis.
Tools and Technologies:

- **Ansible** is used to create the VMs and deploy required configurations via ansible-playbook.

- **SSH, SCP, and SFTP** is used to establish secure connections and transfer files between the master system and remote VMs.

- **OpenNebula** provides the cloud infrastructure management platform. Tools such as onetemplate and onevm interact with the OpenNebula RPC endpoint CENDPOINT to instantiate and control VMs.

Methodology:
The project creates attacker and victim virtual machines within the VU MIF OpenNebula cloud to make repeatable, cybersecurity simulations in an ethical and controlled environment.

First, the script (Figure 4) prompts the user to log in with their VU MIF cloud credentials. If authentication is skipped, or the credentials are invalid the process is aborted. Using OpenNebula's CLI tools onetemplate and onevm, the Bash controller instantiates two Debian 12 VMs, injects SSH public keys for passwordless authentication, and appends the VMs' private IP addresses to the Ansible inventory (Figure 6).

If no SSH key exists, a new one is generated and added to the SSH agent. Two virtual machines are created: one as the attacker and the other as the victim.

The user is then able to select one of three simulations (Figure 5):

1. Large number of TCP connections to an external resource
2. Brute-force attempt via SSH
3. Malicious object detected but not deleted

The user selects a simulation by entering 1, 2, or 3. If an invalid input is provided, the system defaults to option 1 (large number of TCP connections). Ansible playbooks configure both VMs and initiate the chosen scenario.

For simulations 1 and 2, the attacker playbook runs first, followed by the victim playbook. For simulation 3 ("malicious object detected but not deleted"), the order is reversed, the attacker playbooks are executed first, then the victim's.

After the simulation completes, the script automatically retrieves the TCP capture files (PCAPs) from both VMs: labeled `A` for attacker and `B` for victim, using SFTP The captures are stored on the master machine for analysis.

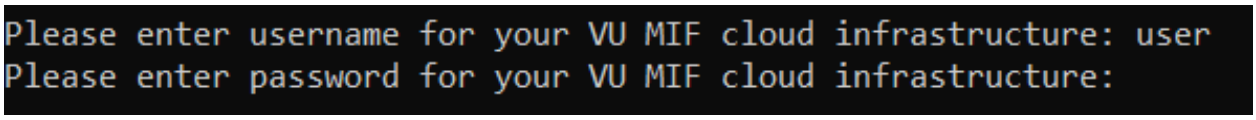

Figure 4. User logs in to OpenNebula with their credentials

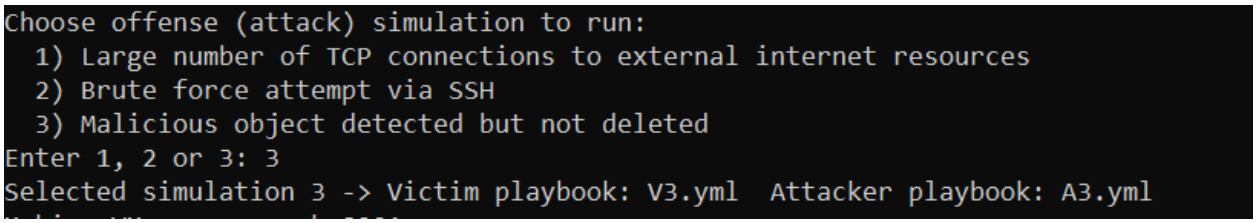

Figure 5. User chooses between simulations

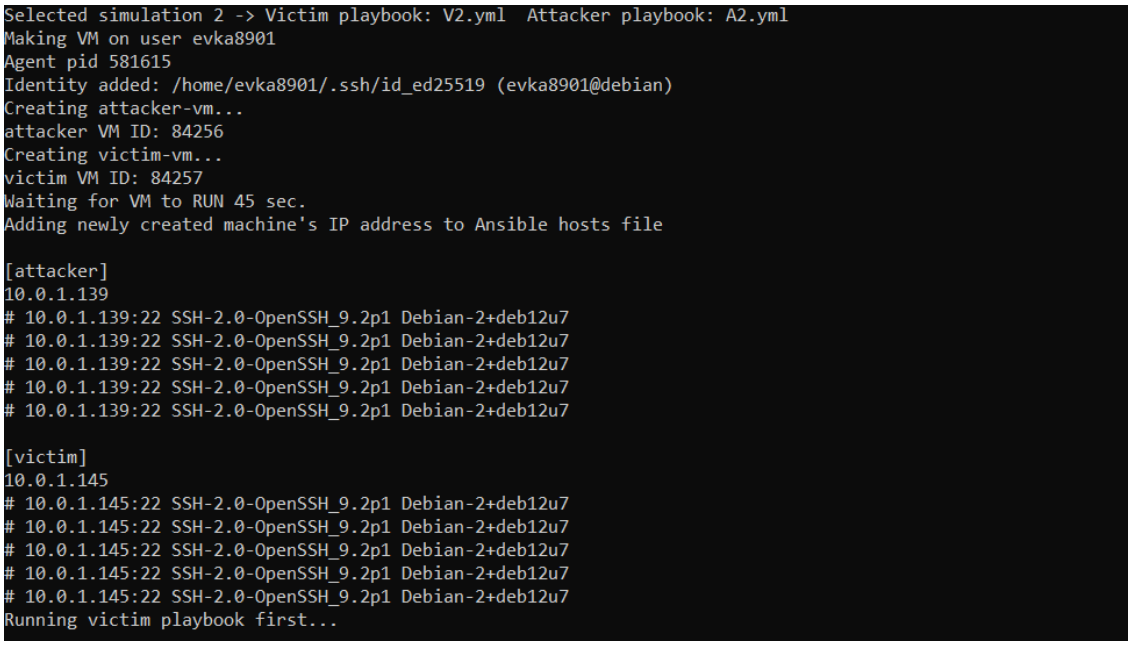

Figure 6. Virtual Machine creation

3.3 Offenses

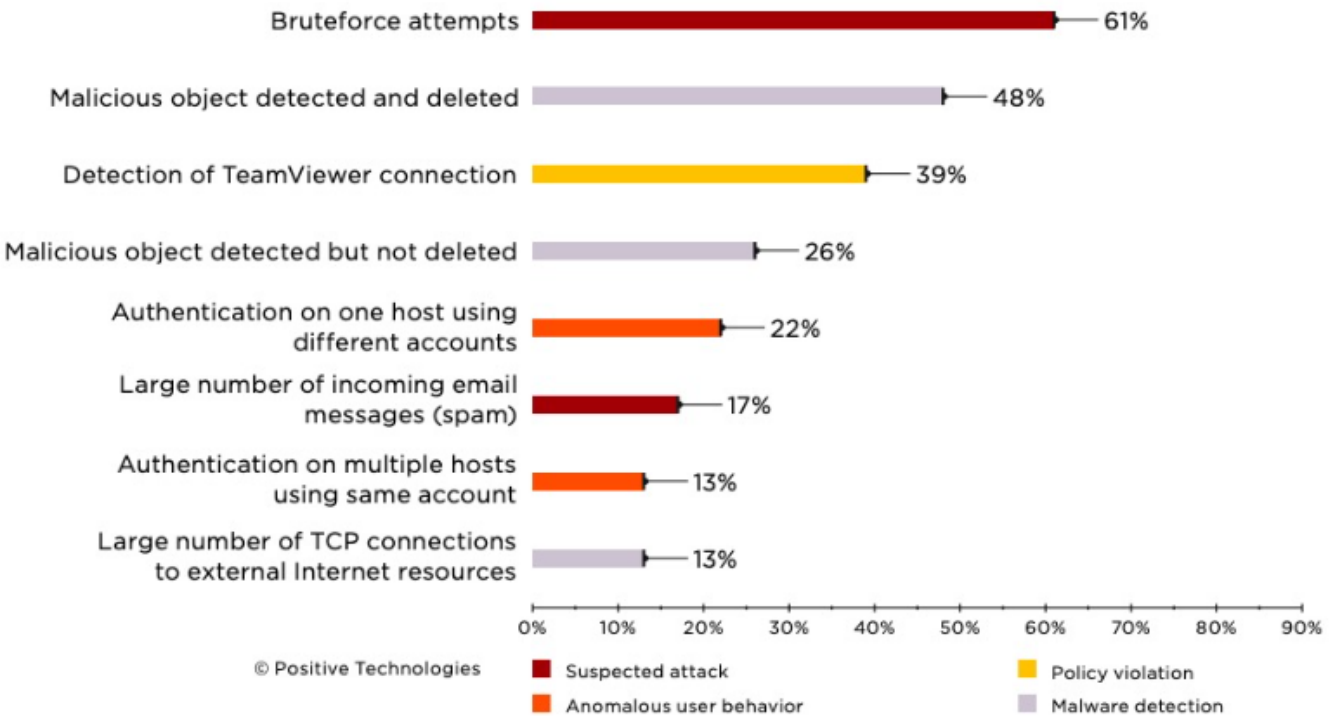

Figure 7. Figure from the report by Positive Technologies (PT Security) "Going beyond the ordinary: how SIEM does incident detection", (2020).

Offense in SIEM systems is a security incident or suspicious activity detected by correlating events and flows from various sources. It is made up of three aspects. To make the explanation of an offense more understandable, we can create a scenario: an employee from company "X" failed to log in to his device 10 times within 5 minutes. This created an Event - "10 failed logins in a short time frame." The employee used a remote work computer assigned the IP 10.10.X.X and attempted to connect to a device on the company's network with its own IP 10.1.X.X. This information is defined as a Flow - a record of network communication between hosts (the source and the destination). This allows us to collect as many details as possible describing the event.

Prior to this situation, the company's security personnel had defined a rule in their local SIEM tool, stating that 10 failed logins to a company device within 5 minutes should trigger a detection. This led to the correlation between the user activity and the SIEM rule, which triggered an Offense. This offense can then be analyzed either by another software component, a user, or an analyst to determine what actions should be taken next.

To get the most accurate picture of how SIEM technologies work, the attack types in this project were chosen to closely resemble the most common types of offenses. As a reference for the most frequent offenses, a real-life publicly available SIEM tool report was used [5] from Positive Technologies (PT Security). It is the MaxPatrol SIEM produced by Positive Technologies. The report consists of real detections from the second half of 2019 and early 2020. The figure 7 shows what MaxPatrol SIEM detected in real security incidents in 100 percent of the projects with the dataset. This dataset was publicly available and simple to understand for these reasons it is a suitable candidate to reference. From top to bottom, we can see eight of the most commonly triggered offenses, which are:

- brute force attempt - occurs when someone repeatedly tries different passwords to gain access to a system.
- malicious file detected and deleted - means security software found and removed harmful code.
- detection of TeamViewer connection - an alert showing that a remote-access session using TeamViewer software was detected.
- malicious object detected but not deleted - describes when a threat is found on a system but remains because it could not be deleted.
- authentication on one host using different accounts: when authentication occurs on one host using different accounts.
- large number of incoming emails (spam) - a spike in received emails classified as unwanted or spam.
- large number of TCP connections to an external internet resource - a host initiates an unusually high number of outbound network connections to external systems.

From these offenses, three were chosen. Detection of TeamViewer connection could not be replicated due to being out of project scope for a script-based project and due to VM resource limitations. The project was defined to require as little user interaction as possible. Malicious objects could not be replicated in this project because it would require advanced external software that could perform actions like deletion, which were out of scope due to resource limitations and not aligned with the main project idea - analyzing network attacks. Authentication on one host using different accounts was deemed redundant and too similar to the detection of brute-force attempts in its pattern; the project aimed to have a variety of offenses. A large number of incoming email messages (spam) required an email inbox, which was deemed out of scope for this project.

This left three offense types: Brute-force attempts - deemed in line with the project's goals and replicable using scripts. Malicious object detected but not deleted - achievable using scripts without the need for external software such as an EDR or antivirus solution. Large number of TCP connections - feasible, aligned with the project's scope, and executable without external dependencies.

3.4 SSH Brute-force Simulation

One of the most elementary attacks in cyberspace is a brute-force attack, characterized by many attempts to log in to a system using different username and password combinations until one succeeds. It is usually performed automatically with the help of tools. In this project a variant of this is simulated: an SSH brute-force attack. SSH (Secure Shell) is a network protocol used to securely connect to and manage remote computers over an encrypted channel, it replaces older, insecure methods like Telnet or rlogin that transmit data in plain text. When a user connects to a remote system via SSH, the user authenticates to the server using either a password or an SSH key pair (a public and private key). Once the connection is established, the user can run commands, transfer files, or perform other required actions.

In a real-world scenario, a threat actor might have a username and try many different passwords, or might have a password and try many usernames, or might try combinations of both. In this

simulation the threat actor knows the username and uses a random password generator to find a password. The attacker playbook generates 25 candidate passwords consisting of four lowercase letters each and attempts to log in with each candidate.

In this project the attack and victim behaviours are implemented as separate Ansible playbooks; examining each playbook gives better insight into the simulation.

Victim objective: Create a test user account with a password and enable SSH so that password-based login attempts can be observed in the network capture.

Tools/Technologies:

- **Ansible** for orchestration.
- **openssh-server** the program that accepts incoming SSH connections.
- **tcpdump** for network capture.

Methodology: The victim playbook installs the SSH server and the packet-capture tool tcpdump. The SSH daemon (sshd) is started and enabled so that remote SSH logins are possible (default port 22 unless changed). The playbook enables password authentication is for the duration of the simulation (so password-based attempts are accepted by sshd). A test user account is created and the password is stored as a hashed value. Tcpdump is let to run in the background.

Attacker objective: Simulate a brute-force attack consisting of 25 password attempts against the test user.

Tools/Technologies:

- **Ansible** for orchestration.
- **sshpass** for non-interactive password input.
- **tcpdump** for network capture.

Methodology: The attacker playbook installs tcpdump and sshpass and runs tcpdump in the background for the same capture interval as the victim so their captures overlap. Before attempting logins the playbook checks that the victim's SSH port is reachable and aborts if it is not. The brute-force simulation is done with a loop that repeats 25 times; each iteration generates a random four-letter lowercase password and then calls sshpass to attempt a non-interactive SSH login with that password. A brief delay is also implemented between attempts to avoid overwhelming the service as this might cause not catching all of the log in attempts in the networkk data. Because the attacker generates only four-letter lowercase candidates but the victim password is 10 letters, a successful login never occurs. The goal of the simulation is to produce SSH authentication traffic rather than to break in.

Getting deeper into the password-generation command:

```
1 tr -dc 'a-z' </dev/urandom | head -c4
```

This pipeline reads bytes from `/dev/urandom`, a non-blocking kernel source of pseudo-random bytes, then filters the stream with `tr -dc 'a-z'` to remove every character not in the range `a-z`. The filtered stream is passed to `head -c4`, which takes the first four characters and exits. The result is a random four-character lowercase string (for example, *qzmd*).

Below (Figure 8) is an example network flow for this type of offense. The image shows the source IP, destination IP, protocol, length, source port, destination port, and packet info. A notable feature of this traffic is many connections to port 22, over a short interval.

Source	Destination	Protocol	Length	s port	d port	Info
10.0.0.185	10.0.1.1	TCP	74	49742	22	49742 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909264 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49742	22 → 49742 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173123 TSecr=2656909264 WS=128
10.0.0.185	10.0.1.1	TCP	66	49742	22	49742 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909265 TSecr=1514173123
10.0.0.185	10.0.1.1	TCP	74	49752	22	49752 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909265 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49752	22 → 49752 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173124 TSecr=2656909265 WS=128
10.0.0.185	10.0.1.1	TCP	66	49752	22	49752 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909266 TSecr=1514173124
10.0.0.185	10.0.1.1	TCP	74	49754	22	49754 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909266 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49754	22 → 49754 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173125 TSecr=2656909266 WS=128
10.0.0.185	10.0.1.1	TCP	66	49754	22	49754 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909267 TSecr=1514173125
10.0.0.185	10.0.1.1	TCP	74	49766	22	49766 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909272 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49766	22 → 49766 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173131 TSecr=2656909272 WS=128
10.0.0.185	10.0.1.1	TCP	66	49766	22	49766 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909272 TSecr=1514173131
10.0.0.185	10.0.1.1	TCP	74	49782	22	49782 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909273 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49782	22 → 49782 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173132 TSecr=2656909273 WS=128
10.0.0.185	10.0.1.1	TCP	66	49782	22	49782 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909274 TSecr=1514173132
10.0.0.185	10.0.1.1	TCP	74	49798	22	49798 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909274 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49798	22 → 49798 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173133 TSecr=2656909274 WS=128
10.0.0.185	10.0.1.1	TCP	66	49798	22	49798 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909275 TSecr=1514173133
10.0.0.185	10.0.1.1	TCP	74	49802	22	49802 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909275 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49802	22 → 49802 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173134 TSecr=2656909275 WS=128
10.0.0.185	10.0.1.1	TCP	66	49802	22	49802 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909276 TSecr=1514173134
10.0.0.185	10.0.1.1	TCP	74	49810	22	49810 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909280 TSecr=0 WS=128
10.0.1.1	10.0.0.185	TCP	74	22	49810	22 → 49810 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1514173138 TSecr=2656909280 WS=128
10.0.0.185	10.0.1.1	TCP	66	49810	22	49810 → 22 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=2656909280 TSecr=1514173138
10.0.0.185	10.0.1.1	TCP	74	49818	22	49818 → 22 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=2656909281 TSecr=0 WS=128

Figure 8. Network flow example of an SSH brute force offense.

3.5 Malicious Object Detected But Not Deleted Simulation

The phrase malicious object detected but not deleted means that defensive software identified an object that shows characteristics of a known malicious file, process, or script, but did not remove it. The object can be anything from an infected file or a trojan executable to a script containing exploit code, a registry entry, or an in-memory artifact related to malware. In this project, a malware file was chosen to represent the malicious object.

Within incident classification in SIEM systems, the table below (Table 1) reproduces the low-level malware event categories from the official IBM QRadar documentation [6]. It illustrates how detected items (such as the EICAR test file) are normalized, categorized, and prioritized by the platform, shows the different types of malware the SIEM can identify, and provides an indication of how many distinct categories exist.

In our scenario it was aimed to simulate a case where a victim downloaded malware and an antivirus-like detection mechanism identified its presence but did not delete it. This was implemented by creating a file evil.exe on the attacker host containing the EICAR [12] test string and serving it with Apache2. The EICAR test string is a non-malicious, specially crafted ASCII sequence that antivirus and anti-malware products are designed to detect as if it were actual malware. It is a standard test file published by the European Institute for Computer Antivirus Research (EICAR) and Computer Antivirus Research Organization (CARO). An example EICAR string is:

```
1 X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Apache2 is an open-source web server that serves files and web applications to clients over HTTP and HTTPS.

Prior to the test, in our scenario the play was that the Security Team created a YARA rule to detect this threat. YARA [7] is a pattern-matching and malware identification tool; it is not directly an antivirus product and does not quarantine or delete files. YARA was chosen because it allows writing flexible rules to describe suspicious files and, in this constrained lab environment, it provides a light-weight and reliable way to flag the test file without running a full antivirus that scans everything. The YARA rule is written to reliably match the EICAR string so the detection step consistently succeeds; because YARA only reports matches, the detected file remains on disk.

Table 1. Low-level malware Event Categories

Low-level Event Category	Description
Unknown Malware	Indicates an unknown virus.
Backdoor Detected	Indicates that a back door to the system was detected.
Hostile Mail Attachment	Indicates a hostile mail attachment.
Malicious Software	Indicates a virus.
Hostile Software Download	Indicates a hostile software download to your network.
Virus Detected	Indicates that a virus was detected.
Misc Malware	Indicates miscellaneous malicious software.
Trojan Detected	Indicates that a trojan was detected.
Spyware Detected	Indicates that spyware was detected on your system.
Keylogger	Indicates that a key logger was detected.
Adware Detected	Indicates that Ad-Ware was detected.
Malware Infection	Indicates that a malware infection was detected.

When the YARA rule matches, the victim playbook sends a POST to the attacker's endpoint so the detection event is visible in the network capture (this is part of the simulation and not normal defensive behaviour).

Attacker objective: Distribute a malicious object to the victim.

Tools/Technologies:

- **Ansible** for orchestration.
- **Apache2** for hosting the malicious object.
- **tcpdump** for network capture.
- **EICAR** Malware test file.

Methodology: The attacker installs tcpdump and runs it, then sets up an Apache2 server to host the malicious object. The attacker creates /var/www/html/evil.exe containing the EICAR test string. Because Apache2 uses /var/www/html as the default document root, the file will be served via HTTP (e.g., http://<vm>/evil.exe). After this step the attacker playbook finishes.

Victim objective: Download the malicious object from the attacker and detect it with a detection tool.

Tools/Technologies:

- **Ansible** for orchestration..
- **YARA** for pattern-based detection.
- **tcpdump** for network capture.

Methodology: The victim installs tcpdump and YARA, then writes a YARA rule file to /tmp/eicar.yar. The rule looks for the EICAR test string and will match if the target file contains that exact byte sequence. The victim downloads evil.exe from the attacker and saves it to /home/evil.exe. The

victim then runs YARA against the downloaded file; the scan will match and return exit code 0. When YARA finds a match, the victim sends a POST to the attacker so that the detection event appears in the PCAP and the attacker is signalled that the payload was retrieved and detected. Since the objective was to detect but not delete the malicious file, no further remediation is taken. After this the victim playbook finishes.

Below is the YARA rule used to detect the EICAR string. This YARA rule detects the EICAR Standard Antivirus Test File by matching its unique, test string. It includes metadata such as description, author, and date to help keep track of the rule. The dest parameter specifies the target location where the file eicar.yar will be created. The mode parameter defines the file's permissions 0644, allowing read access for all users but write access only for the owner.

```
name: Create YARA rule for EICAR detection
copy:
content: |
rule EICAR_Test_File
{
meta:
description = "EICAR Standard Antivirus Test File"
author = "Security Team"
date = "2025-09-30"
strings:
$eicar = "X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE
    !$H+H*"
condition:
$eicar
}
dest: /tmp/eicar.yar
mode: '0644'
```

Below (Figure 9)is an example network flow for this offense type. The image shows the source IP, destination IP, protocol, length, source port, destination port, and packet info. A notable feature in this network flow is the POST message that the victim sends back to the attacker when the YARA rule detection triggers.

Source	Destination	Protocol	Length	s port	d port	Info
10.1.1.113	10.1.1.112	TCP	74	53754	80	53754 → 80 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=782231221 TSecr=0 WS=128
10.1.1.112	10.1.1.113	TCP	74	80	53754	80 → 53754 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1884290148 TSecr=782231221 WS=128
10.1.1.113	10.1.1.112	TCP	66	53754	80	53754 → 80 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=782231223 TSecr=1884290148
10.1.1.113	10.1.1.112	HTTP	185	53754	80	GET /evil.exe HTTP/1.1
10.1.1.112	10.1.1.113	TCP	66	80	53754	80 → 53754 [ACK] Seq=1 Ack=120 Win=62208 Len=0 TSval=1884290149 TSecr=782231223
10.1.1.112	10.1.1.113	HTTP	398	80	53754	HTTP/1.1 200 OK (application/x-msdos-program)
10.1.1.113	10.1.1.112	TCP	66	53754	80	53754 → 80 [ACK] Seq=120 Ack=333 Win=62208 Len=0 TSval=782231231 TSecr=1884290156
10.1.1.112	10.1.1.113	TCP	66	80	53754	80 → 53754 [FIN, ACK] Seq=333 Ack=120 Win=62208 Len=0 TSval=1884290156 TSecr=782231223
10.1.1.113	10.1.1.112	TCP	66	53754	80	53754 → 80 [FIN, ACK] Seq=120 Ack=334 Win=62208 Len=0 TSval=782231232 TSecr=1884290156
10.1.1.112	10.1.1.113	TCP	66	80	53754	80 → 53754 [ACK] Seq=334 Ack=121 Win=62208 Len=0 TSval=1884290158 TSecr=782231232
10.1.1.113	10.1.1.112	TCP	74	53758	80	53758 → 80 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=782232940 TSecr=0 WS=128
10.1.1.112	10.1.1.113	TCP	74	80	53758	80 → 53758 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=1884291866 TSecr=782232940 WS=128
10.1.1.113	10.1.1.112	TCP	66	53758	80	53758 → 80 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=782232941 TSecr=1884291866
10.1.1.113	10.1.1.112	TCP	262	53758	80	53758 → 80 [PSH, ACK] Seq=1 Ack=1 Win=62464 Len=196 TSval=782232941 TSecr=1884291866 [TCP PDU reassembled in 15]
10.1.1.113	10.1.1.112	HTTP	74	53758	80	POST /MALWARE_DETECTED HTTP/1.1 (application/x-www-form-urlencoded)
10.1.1.112	10.1.1.113	TCP	66	80	53758	80 → 53758 [ACK] Seq=1 Ack=197 Win=62208 Len=0 TSval=1884291867 TSecr=782232941
10.1.1.112	10.1.1.113	TCP	66	80	53758	80 → 53758 [ACK] Seq=1 Ack=205 Win=62208 Len=0 TSval=1884291867 TSecr=782232941
10.1.1.112	10.1.1.113	HTTP	518	80	53758	HTTP/1.1 404 Not Found (text/html)
10.1.1.113	10.1.1.112	TCP	66	53758	80	53758 → 80 [ACK] Seq=205 Ack=453 Win=62080 Len=0 TSval=782232945 TSecr=1884291870
10.1.1.112	10.1.1.113	TCP	66	80	53758	80 → 53758 [FIN, ACK] Seq=453 Ack=205 Win=62208 Len=0 TSval=1884291870 TSecr=782232941
10.1.1.113	10.1.1.112	TCP	66	53758	80	53758 → 80 [FIN, ACK] Seq=205 Ack=454 Win=62080 Len=0 TSval=782232945 TSecr=1884291870
10.1.1.112	10.1.1.113	TCP	66	80	53758	80 → 53758 [ACK] Seq=454 Ack=206 Win=62208 Len=0 TSval=1884291871 TSecr=782232945

Figure 9. Network flow example of a malicios object detected but not deleted offense.

3.6 Large Number of TCP Connections to External Internet Resource Simulation

A large number of TCP connections to external Internet resources refers to a situation where a host opens many TCP connections to addresses on the public Internet. This can be many simultaneous connections or a very high rate of new connections over a short time window. This offense could indicate malware activity, data exfiltration, or the early stages of a denial-of-service attack. In this simulation the attacker is emulated so that it connects back to the attacker host itself to create a safe simulation environment and prevent causing a real DoS (denial of service) attack to an external internet resource.

To configure a TCP connection limit per session

1. Go to **DoS Protection > Application > Malicious IPs**.

   To access this part of the web UI, your administrator's account access profile must have **Read** and **Write** permission to items in the **Web Protection Configuration** category. For details, see Permissions.

2. Click **Create New**.
3. Configure these settings:

Name	Type a unique name that can be referenced in other parts of the configuration. Do not use spaces or special characters. The maximum length is 63 characters.
TCP Connection Number Limit	Type the maximum number of TCP connections allowed with a single HTTP client. The valid range is from 1 to 1,024. The default is 1. Fortinet suggests an initial value of 100. For details, see Reducing false positives.

Figure 10. Recommended TCP connection number limit from FortiWeb user manual, 2025.

The distinction between what is considered a DoS and what is a large number of external TCP connections can be surprisingly small. American cybersecurity company Fortinet and their produced FortiWeb web application firewall appliance, in their user manual (Figuren 10) [13], Fortinet recommends a TCP connection limit of 100 connections per single HTTP client to help prevent DoS attacks. This simulation is not intended to create an actual DoS attack but to model a large number of TCP connections to an external resource; therefore, the victim makes nine connections in total (three connections to each of three different ports) back to the attacker host. Prior to creating the script for the victim, the attacker starts fake "listener" web servers on multiple ports so that the victim machine can connect to them during the simulated test and those listeners act as the external Internet resource.

Attacker objective: Create a host that accepts incoming connections so that the victim can connect to it, and create a script forcing a connection.

Tools/Technologies:

- **Ansible** for orchestration.
- **apache2** web server used to host the script so a victim can download it.

- **tcpdump** for network capture.
- **Python3** to run simple HTTP servers on multiple ports.

Methodology:
The attacker defines the ports where it will start simple HTTP responders, and installs tcpdump, python3, and apache2. A shell loop runs three instances of Python's built-in HTTP server bound to ports 8001, 8002, and 8003 on 0.0.0.0 (listen on all interfaces). Each server is started in the background so the Ansible task does not stop. The attacker now has processes listening at TCP ports 8001-8003 on its network interfaces. Any host that can reach the attacker's IP and those ports can open TCP connections to them. This is the initial setup for the attacker to act as the "external Internet resource."

Next, the attacker writes a script to the Apache document root (`/var/www/html/script.sh`) for the victim to download and execute. The script is made executable with mode 0755 and contains the attacker host's IP address at the time the play runs.The mode value 0755 sets the file permissions to allow the owner to read, write, and execute the file, while other users are granted read and execute permissions only. The script performs three rounds; in each round it launches three background curl requests (one to each port 8001/8002/8003) with a 3-second maximum timeout, producing a total of 9 connections. Apache serves script.sh so a victim can download it with curl and then execute it. The Python servers on ports 8001-8003 act as responder endpoints for the curl connections. The attacker also runs tcpdump to capture the resulting network traffic.

Victim objective: Download a script from the attacker and execute it.

Tools/Technologies:

- **Ansible** for orchestration.
- **tcpdump** for network capture.

Methodology:
The victim installs tcpdump and configures it to filter out traffic on port 22 (to reduce noise in the pcap, since port 22 is used for Ansible management). The victim then downloads a script from the attacker and saves it as an executable. The victim makes the script executable and runs it. The script issues the 9 curl connections described above. Tcpdump runs during the scripted activity to capture the network traffic.

Below (Figure 11)is an example of the network flow for this offense type. The image shows source IP, destination IP, protocol, length, source port, destination port, and packet information. A notable feature in this network flow is the sequence of TCP handshakes followed by many HTTP GET requests to the external resource.

Source	Destination	Protocol	Length	s port	d port	Info
10.1.1.46	10.1.1.50	TCP	80	8002	53704	8002 → 53704 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=3549692840 TSecr=99446821 WS=128
10.1.1.50	10.1.1.46	TCP	72	53704	8002	53704 → 8002 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=99446821 TSecr=3549692840
10.1.1.50	10.1.1.46	HTTP	150	53694	8002	GET / HTTP/1.1
10.1.1.50	10.1.1.46	HTTP	150	50586	8003	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	80	8001	51780	8001 → 51780 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=3549692840 TSecr=99446821 WS=128
10.1.1.50	10.1.1.46	TCP	72	51780	8001	51780 → 8001 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=99446822 TSecr=3549692840
10.1.1.46	10.1.1.50	TCP	72	8002	53694	8002 → 53694 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692840 TSecr=99446821
10.1.1.50	10.1.1.46	HTTP	150	53704	8002	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	72	8003	50586	8003 → 50586 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692840 TSecr=99446821
10.1.1.46	10.1.1.50	TCP	72	8002	53704	8002 → 53704 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692840 TSecr=99446822
10.1.1.50	10.1.1.46	TCP	80	50596	8003	50596 → 8003 [SYN] Seq=0 Win=62370 Len=0 MSS=8910 SACK_PERM TSval=99446822 TSecr=0 WS=128
10.1.1.50	10.1.1.46	HTTP	150	50594	8003	GET / HTTP/1.1
10.1.1.50	10.1.1.46	HTTP	150	51780	8001	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	80	8003	50596	8003 → 50596 [SYN, ACK] Seq=0 Ack=1 Win=62286 Len=0 MSS=8910 SACK_PERM TSval=3549692841 TSecr=99446822 WS=128
10.1.1.50	10.1.1.46	TCP	72	50596	8003	50596 → 8003 [ACK] Seq=1 Ack=1 Win=62464 Len=0 TSval=99446822 TSecr=3549692841
10.1.1.50	10.1.1.46	HTTP	150	53686	8002	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	72	8003	50594	8003 → 50594 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446822
10.1.1.50	10.1.1.46	HTTP	150	51768	8001	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	72	8001	51780	8001 → 51780 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446822
10.1.1.50	10.1.1.46	HTTP	150	50596	8003	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	72	8002	53686	8002 → 53686 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446822
10.1.1.46	10.1.1.50	TCP	72	8001	51768	8001 → 51768 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446822
10.1.1.50	10.1.1.46	HTTP	150	51776	8001	GET / HTTP/1.1
10.1.1.46	10.1.1.50	TCP	72	8003	50596	8003 → 50596 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446823
10.1.1.46	10.1.1.50	TCP	72	8001	51776	8001 → 51776 [ACK] Seq=1 Ack=79 Win=62208 Len=0 TSval=3549692841 TSecr=99446823

Figure 11. Network flow example of a large number of TCP connections to an external Internet resource.

3.7 PCAP Retrieval

This step is the retrieval of PCAPs from both the attacker and victim VMs after a simulation. These PCAPs are the primary dataset for the followig network traffic analysis.

Objective: Retrieve the PCAP generated during the offense simulation.

Tools and technologies: The retrieval procedure relies on Ansible and an SFTP-based transfer to copy the PCAP files from the virtual machines to a Master VM directory.

Methodology. Upon completion of each simulation, the system calls for a helper routine that connects to the attacker and victim machines and copies their PCAP files into the working Master VM directory for analysis (Figure 12). The naming for the captures includes a label to differentiate that multiple runs do not overwrite one another and that each capture can be traced back to its originating simulation. Capture A refers to attacker data and Capture B to victim data. The TCP ending represents a large number of TCP connections to an external internet resource, SSH ending stands for an SSH brute-force attack, and file ending refers to a malicious object that was detected but not deleted.

```
Retrieving captureA_tcp.pcap from attacker (10.0.0.163)...
captureA_tcp.pcap                                          100% 1521    88.4KB/s   00:00
Copied via scp to /home/evka8901/attacker_captureA_tcp.pcap
Retrieving captureB_tcp.pcap from victim (10.1.0.240)...
captureB_tcp.pcap                                          100%   15KB   1.1MB/s   00:00
Copied via scp to /home/evka8901/victim_captureB_tcp.pcap
Done. If present, PCAPs saved to /home/evka8901:
  /home/evka8901/attacker_captureA_tcp.pcap
  /home/evka8901/victim_captureB_tcp.pcap
```

Figure 12. PCAP files from both the attacker and victim VMs are retrieved via SFTP

3.8 Detection Algorithm

The detection algorithm begins with a user prompt that allows the user to select between three options (Figure 13): analyzing collected data (PCAPs gathered after a chosen simulation from both

the attacker and the victim), modifying the rules (which will be discussed in the following chapter), or exiting the program.

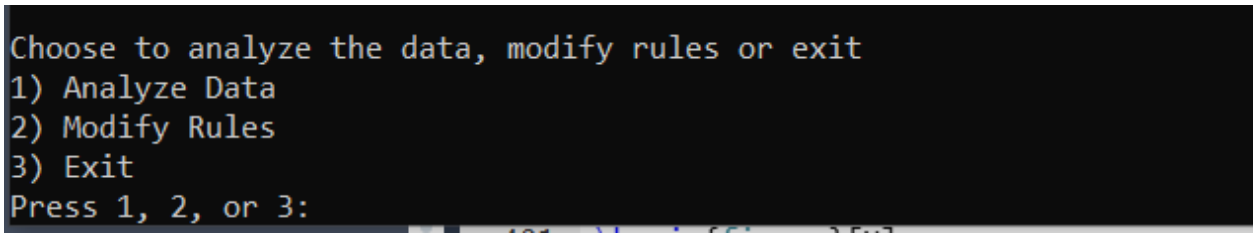

Figure 13. User menu of the analysis algorithm.

Once the user selects the data analysis option, all available PCAP files are displayed (Figure 14). The list of PCAP files updates dynamically as new captures are added or existing ones are removed from the directory in which the main code is.

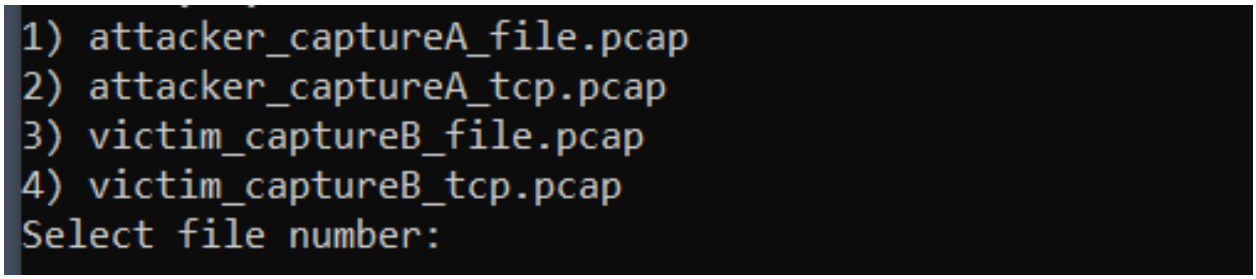

Figure 14. List of available PCAPs.

After selecting a PCAP file, the analysis is performed. The PCAP is evaluated against the rules defined in the SIEMrules.json file, and if a match is found, an offense is generated. Each offense includes a description, a magnitude level (high, medium, or low), the source and destination IP addresses, the protocol involved, the timestamp, and information extracted from the payload such as Accept-Encoding, Host, User-Agent, and connection status. A raw payload extract is also provided. Once the analysis results are displayed, the user can run another analysis, modify the rules, or exit the program.

Analysis objective: Provide SIEM-framework-based detection and analysis of the offense.

Tools/Technologies:

- **Python 3** primary programming language
- **Scapy** packet capture parsing and network protocol inspection
- **Pandas** data analysis and event aggregation
- **JSON** rule storage and configuration format
- **Standard Python Libraries:** os, time file system interaction, timestamps
- **Command-Line Interface (CLI)** for user interaction
- **PCAP** as input data source

The main script, called SIEM.py, first provides a simple menu to run analysis, open a rules editor or exit. If analysis is selected, it lets the user pick a file: the algorithm lists all files ending in .pcap that exist in the working directory, which is the directory where the code is located.It returns the chosen filename or None if no files are found. After this, it reads the selected PCAP file containing captured network traffic (using Scapy) and converts each packet into a structured event containing the timestamp, source and destination IP addresses and ports, protocol information, and payload.This constitutes the input and data-processing stage. Following this, the algorithm applies a set of detection rules loaded from the JSON file SIEMrules.json. The rules are stored as JSON and loaded into memory before analysis. This seperates the detection logic from the code and makes the system extensible, allowing the user to create their own rules and manage them as needed. The rules can be either packet rules meaning stateless checks that match a single packet or stateful rules that look for patterns across multiple packets (for example, many connections from one IP in a short time). The algorithm evaluates the rule-matching criteria against the packet's attributes and raw payload; this is the anomaly-detection component. It is used both by stateless rules and as a preliminary filter for stateful evaluation. Stateful detection performs temporal aggregation per entity and uses a sliding window to detect bursts that exceed a threshold within a specified time window. This mechanism is used for both SSH brute-force attempt detection and the detection of a large number of TCP connections to an external Internet resource, as it matches and aggregates multiple triggers across the dataset. The analysis algorithm converts raw packet captures into structured events, applies stateless (packet-based) detection first, then performs stateful aggregation, and finally presents the flagged events. It then prints the detected events in an offense-style layout; this is the output stage, which includes a score and a label. If no flagged events are found, it prints "No suspicious events detected." The system avoids duplicate alerts by leaving only one alert per offense and preventing repeated notifications for the same behaviour.The algorithm looks up the relevant rule object in the loaded rule set to print the description and magnitude.The display emulates a SIEM offense view, presenting event summary blocks followed by a raw-payload log for investigation. The offense information includes the description of the offense, its magnitude (high, medium, or low), source IP, destination IP, protocol, timestamp, and information contained in the payload such as Accept-Encoding, Host, User-Agent, and connection status. After this, a raw payload of the offense is provided

The detection rules are stored in a JSON file and are loaded by the analysis script at runtime. Each rule is defined by several fields: name, which specifies the human-readable title of the rule; type, which indicates whether the rule is a "packet" (stateless) rule or a "stateful" (pattern-based) rule; description, which explains the security event the rule represents; magnitude, which defines the severity level; match, which contains the packet-level filtering conditions; and aggregate, which is used exclusively by stateful rules to identify temporal patterns.

The rule "Malicious object detected but not deleted" identifies where a malicious object is reported in network traffic but no mitigation action is taken. As a packet-level rule, it examines individual packets and triggers when the packet uses the TCP protocol and contains both the HTTP method "POST" and the string "/MALWARE DETECTED", which serves as a simulated malware alert indicator. When both conditions are met, the offense is marked with high severity.

The "SSH Brute-force" rule identifies repeated attempts to access an SSH in a short period. This is a stateful rule which first filters for TCP packets targeting port 22. It then groups packets by source IP address and takes in an activity within 30-second windows. If 10 or more connection attempts from the same source IP are detected within that time 30 second frame, the rule is triggered. Although packet captures do not reveal login failures, repeated TCP SYN packets closely

approximate brute-force behaviour, making this a practical detection method.

The rule "Large Number of TCP Connections to External Internet Resources" detects excessive outbound communication to the same external destination. This stateful rule detects TCP packets with the string "GET ", indicating HTTP GET requests, and groups events by both source and destination IP addresses. If at least nine GET requests occur within a 60-second window, an offense is generated. This repeated outbound activity could impy automated behaviour from a malicious software. This is assigned a low severity.

3.9 Rules Management

When the Modify Rules option (Figure 15) is selected, the user is presented with several features for managing SIEM detection rules. These include the ability to add a custom rule, allowing users to introduce new detection rules that integrates with either existing simulations or with the introduction of new simulations. This lets users to expand the detection capabilities without modifying the underlying code. Users can also edit an existing rule, making it possible to adjust detection parameters directly within the program rather than editing the source files manually. This way the system is more flexible and user user-friendly. The user can also delete rules that are no longer nesecery. When all the actions are done, the user can save the updates and return to the main menu.

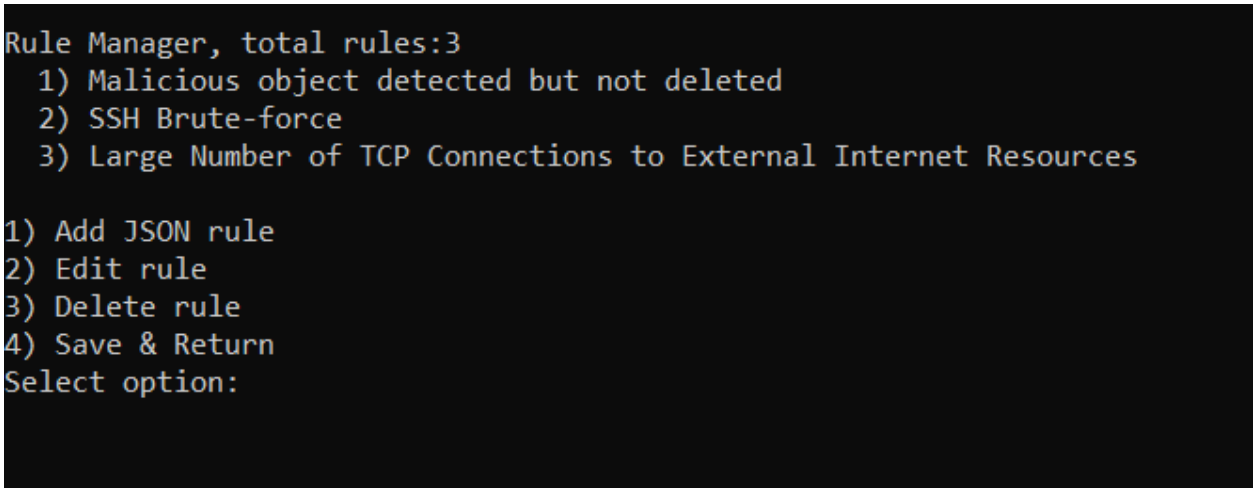

Figure 15. Rules modification menu.

Objective: Ability to add new rules, modify existing rules, delete rules, and save completed actions before returning to the main menu.

Tools/Technologies:

- **Python 3** The programming language used to implement the rule management system.
- **JSON** Used for loading, validating, and saving SIEM detection rules stored in JSON format.
- **os** Used to check file existence, detect the operating system, read environment variables, and remove temporary files.
- **tempfile** Used to create temporary JSON files during rule editing in an external text editor.
- **subprocess** Used to open a system text editor (e.g., nano, notepad, or the editor specified by EDITOR for rule modification.

Methodology: Upon initial start, the Rule Manager class ensures that the rule repository file exists. If it does not, the system automatically creates a new JSON file containing an empty list. This is a check that the rules part is always a properly formatted list, simplifying later operations. After initialization, the system loads all rules from the JSON file and showcases them to the user for review.

Rules are loaded directly at runtime, and any modification made to the rule list can be saved back to the JSON file when the user decides to save their actions. Saving rewrites the file in a JSON format.

When adding a new rule (Figure 16), the program accepts a JSON input entered directly by the user. The rule entry is considered complete when an empty line is entered. The system then parses the JSON into a Python dictionary and runs a set of minimal validation checks. It verifies the presence of essential fields such as the rule's name, type, and either a match section (for packet-based rules) or an aggregate section (for stateful rules). Only rules that successfully pass validation are added to the active rule list, preventing malformed or incomplete rules from being introduced into the SIEM.

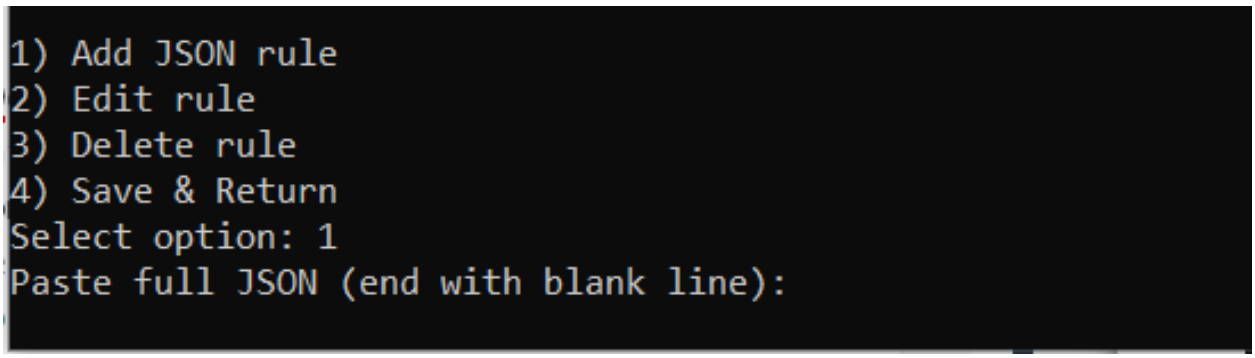

Figure 16. Adding a new rule.

When editing an existing rule (Figure 17), the selected rule is exported to a temporary JSON file. This file is then opened in the user's available text editor. By default, Linux systems use nano, while Windows systems open Notepad; however, if the EDITOR environment variable is set, that editor will be used instead. This feature was implemented so that the analysis part could be run on any OS without requiring a Linux system, mainly for troubleshooting outside of it. This is particularly useful because the virtual machine resources are quite limited, making it more practical in this case to troubleshoot on a local computer, which happened to run Windows. After the user finishes editing and closes the editor, the temporary file is reloaded and validated, and if the updated rule is valid, the original rule in memory is replaced. Finally, the temporary JSON file is deleted to avoid leaving unnecessary files behin

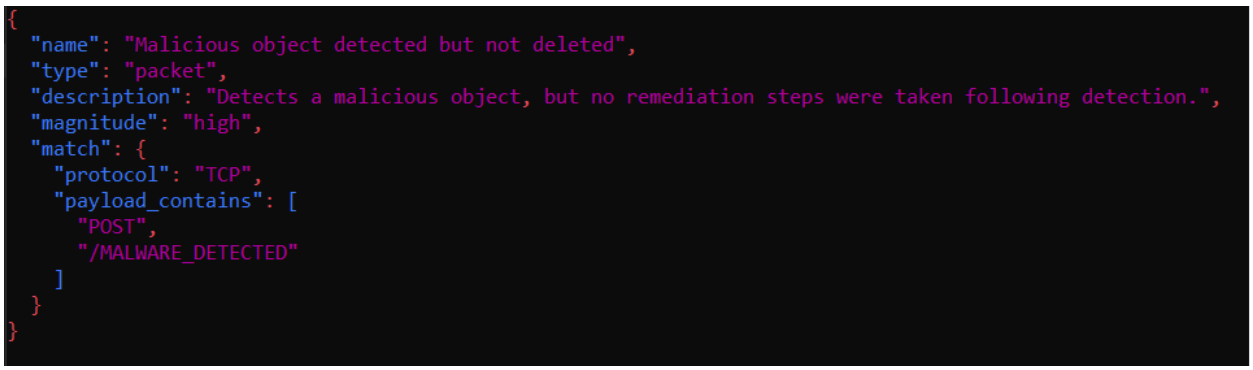

Figure 17. Editing an existing rule in a Linux environment.

The deletion process is straightforward. When the user selects a rule to delete (Figure 18), the system checks whether the selected index is valid and then removes the rule from the internal list. This removal occurs only in memory until the user chooses to save the changes, allowing the user to experiment safely before committing permanent modifications.

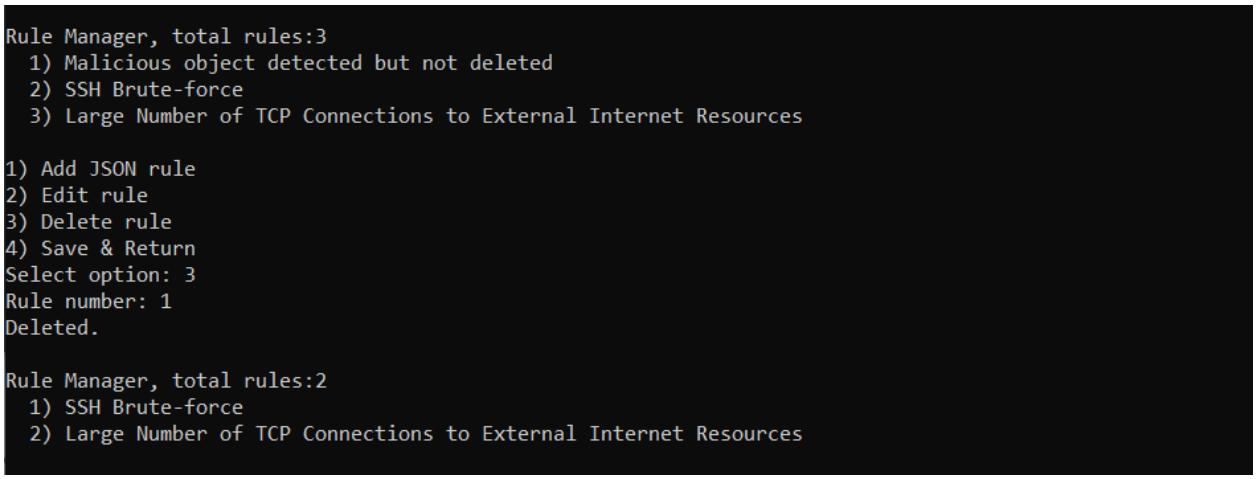

Figure 18. Deleting a rule.

3.10 Results

The results of the project showcase a successful autonomous simulation and detection of three offense types: SSH brute-force, a large number of TCP connections to an external internet resource, and a malicious file detected but not deleted. The tool proved to be an educational SIEM technology imitation, consistently reliable whenever a simulation was executed or analyzed, without crashing or producing false positives. The project demonstrated how an offense is generated on both the attacker and the victim machine, with the ability to analyze each perspective. It also provided a user-friendly and way to expand the analysis component at runtime, enabling users to edit, modify, view, delete, and add new rules when experimenting with new detection patterns. The main limitation for fully integrating new cyber threat detections is the absence of an easier method for adding new simulations at runtime. Addressing this challenge was outside the scope of the current project, but future research could improve upon this.

Below are the showcased detection outputs from the victim PCAPs. The data was extracted by the analysis algorithm. The first result (Figure 19) demonstrates the successful detection of an SSH brute-force attack. Few things can be seen such as: Event Type: SSH Brute-force, Description - The

system detected a source host sending 10 or more TCP SYN packets to port 22 within 30 seconds, meeting the threshold for a brute-force attempt, Magnitude - Medium severity, Protocol - TCP, Timestamp: 2025-11-24 20:41:04 (the actual time the simulation was run) This activity indicates automated or repeated connection attempts toward an SSH service, a sign of brute-force tools or bots. The network information shows: Source IP - 10.0.0.215, Destination IP: 10.0.0.254:22 (SSH service) Meaning that a device within the internal network repeatedly attempted to initiate SSH connections to another internal host. The payload details show repetitive SSH handshake attempts. In one of the payload lines the SSH banner is : SSH-2.0-OpenSSH 9.2p1 Debian-2+deb12u7, indicating that the destination system was running OpenSSH 9.2p1 on Debian 12 during the attempt. Additional payload entries list the key exchange algorithms, ciphers, and MAC options advertised by the server examples include: Key Exchange Algorithms - curve25519-sha256, ecdh-sha2-nistp256, diffie-hellman-group14-sha256, Ciphers: aes128-ctr, aes256-ctr, aes128-gcm, aes256-gcm, MAC Algorithms: hmac-sha2-256, hmac-sha2-512, umac-128

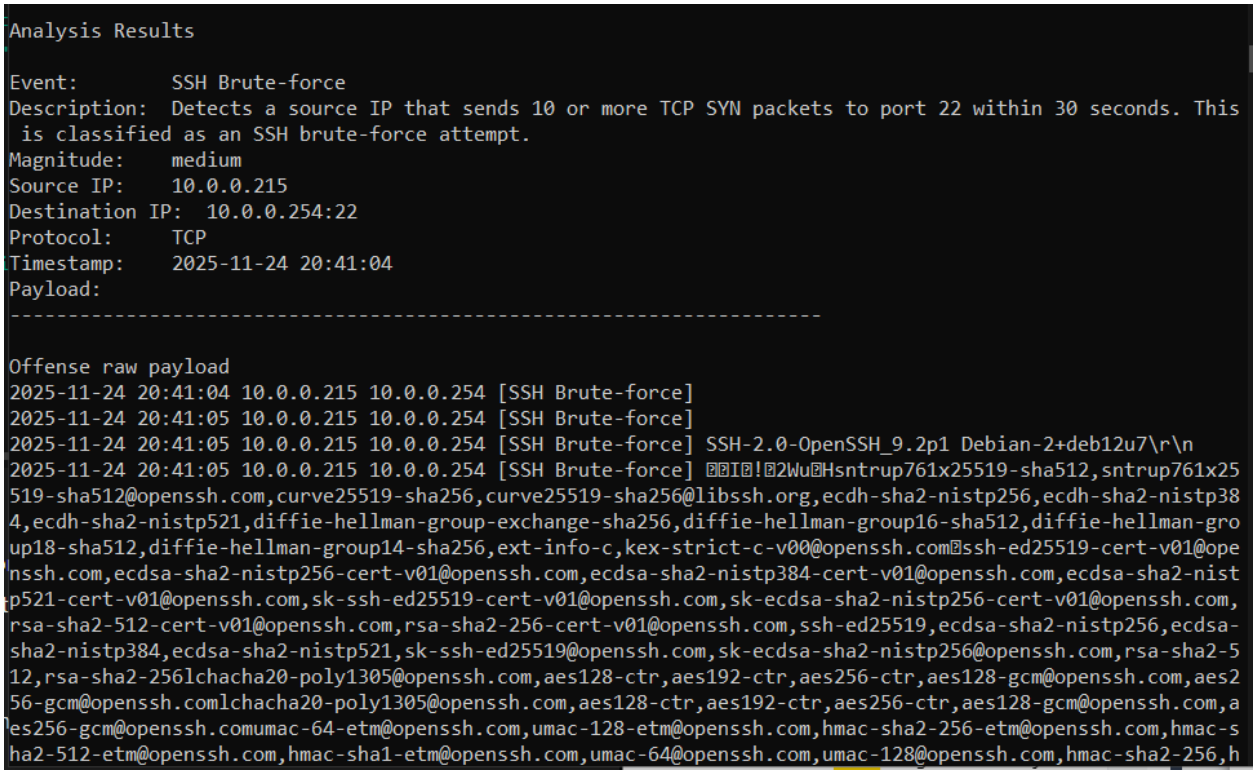

Figure 19. Detection of an SSH brute-force attack in a victim PCAP.

The second successful detection concerns a “Large Number of TCP Connections to External Internet Resources” (Figure 20). The detection descriptions provides information as to what constitues a detection - reports any case where nine or more HTTP GET requests are sent to the same resource within 60 seconds, and classifying it as a low-severity event. Such details can be seen: the source system (10.0.0.234) repeatedly made HTTP GET requests to 10.0.0.218 on port 80. The timestamp is 2025-11-15 22:50:57. The payload shows the requested file script.sh. The initial request uses the user-agent “ansible-httpget,” indicating that the activity was generated by an Ansible playbook or automated configuration task. From further payloads GET requests can be seen from the same source but with different user agents (e.g., curl/7.88.1) targeting multiple ports on the same destination (e.g., 8002, 8001, 8003).

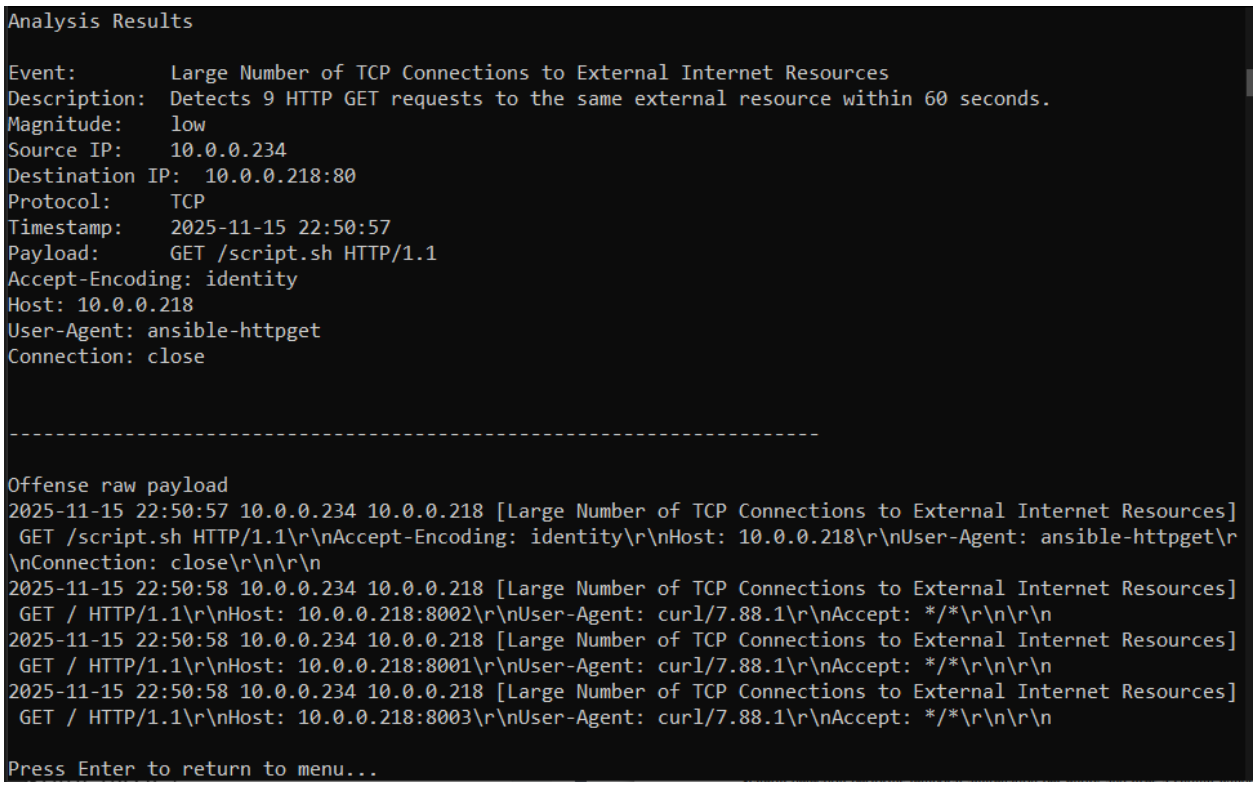

Figure 20. Large number of TCP connections to an external internet resource in a victim PCAP.

The final successful detection, “Malicious object detected but not deleted” (Figure 21), iis when the system identifies malicious content on a host but did not perform automatic remediation. Event details show traffic originating from 10.0.1.233 to 10.0.1.221 over TCP port 80, with a timestamp of 2025-11-24 19:56:37. The captured payload contains an HTTP POST request to the endpoint MALWARE DETECTED, using the user-agent ansible-httpget. The request includes headers such as Content-Type: application/x-www-form-urlencoded and Content-Length: The destination host (10.0.1.221) received the malware detection post. The raw payload shows that this was a single POST event rather than ongoing suspicious activity. Overall, this alert highlights a security concern: a malicious file or behavior was detected on host 10.0.1.233, logged properly, but no cleanup action followed.

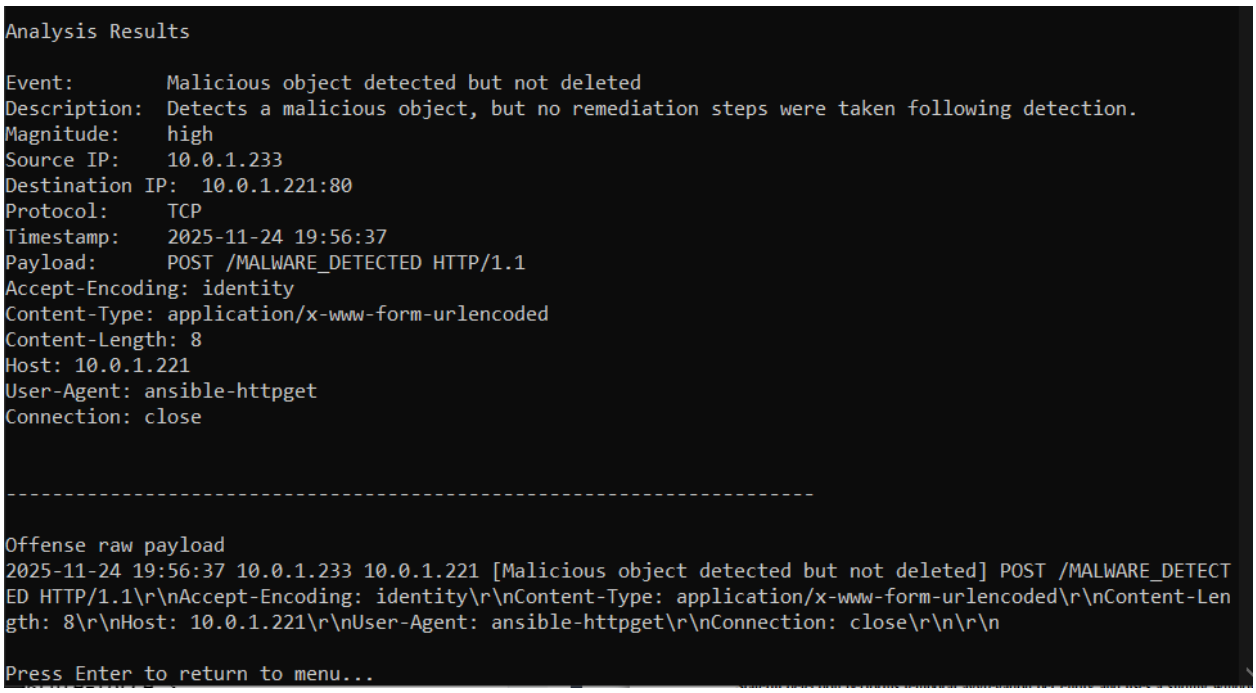

Figure 21. Malicious file detected but not deleted in a victim PACP.

3.11 Testing

The efficiency and reliability of the system were evaluated through manual testing, during which each functional component was executed approximately 40 times. This approach allowed for the observation of both functional correctness and execution stability. The virtual machine creation process achieved successful results in 100 % of the test cases. No simulation failures were observed that could be directly attributed to the scripts. Any errors encountered during the evaluation phase were external to the system and included constraints such as exceeding the maximum number of virtual machines permitted by the credentials used for testing.

The simulation component consistently produced the expected results, achieving a success rate of 100 %. Depending on the selected simulation scenario, the complete simulation process required around 2–3 minutes to complete (including VM creation and PCAP retrieval). These execution times were primarily influenced by the number of generated network events and booting time of the VMs (45 seconds). The analysis component also achieved a 100 % success rate, with an average execution time of approximately 1 second per run.

The rule management component produced the required results in 100 % of test cases. Changes to detection rules were applied immediately. Overall, the evaluation results show that the system is both reliable and efficient for the intended scope.

Conclusions and Recommendations

The results of this project were a successful simulation and detection of cyberattacks using a SIEM inspired framework. By creating two VMs Attacker and Victim the system successfully simulated three types of cyber security offenses including: SSH brute-force attempts, a large number of TCP connections to an external resource, and malicious file detection but not deletion. The captured network packets, stored in PCAP format, were collected by the Master machine and processed using a detection algorithm designed to identify and showcase these three offenses, which it did without any false positives.An advantage of this setup is the ability to capture traffic data from both the attacker and victim perspectives, giving a complete view of network behavior during threats.This capture method provides a look into how an attack appears from both ends. The automation of offenses and data collection ensures that the system is consistent and scalable to accommodate the needs of the person who might use the system

The detection algorithm is flexible of incorporating new rules, increasing the overall usability of the system. In the three specific cases tested, it was entirely free of false positives and consistently detected each simulated offense. The rule management system allows users to dynamically view, modify, remove, and add additional rules, enabling customization without requiring direct modification of the main code.

The project demonstrates the practical use of combining simulation, automation, and analysis to better understand SIEM. As a foundation for future work, the system could be expanded even more to include more elaborate threats to more closely resemble a real world SIEM, such as multi-stage intrusions, advanced persistent threats (APTs), and most notably suspicious user behavior. Machine learning and anomaly detection methods would improve the accuracy of the system, bridging the big gap between theory and practice of SIEM operations.

Recommendations for future development of this project include cutting down the simulation runtime, as each simulation at the moment takes around 2–3 minutes to complete, which may be inconvenient.Additionally, the inclusion of informational pages describing each offense, along with explanations of the corresponding simulations and analyses, would improve the usefulness of the tool and be more beginner-friendly. At present, users are required to conduct their own research prior to using the tool. The primary recommendation a more flexible mechanism for new simulations at runtime, allowing users to define and execute their own simulations. This would allow users to explore and discover new offenses, promoting a deeper understanding through hands-on experimentation and learning.

Roadmap for future research

Future research could improve this system by expanding the analysis component and expanding the automated simulation environment. Simulation number could be greatly improved, and a possibility to create your own custom simulations would also be a wast update to the system. The current detection model may be enough for a primitive detection of 3 very specific attacks but it is not suited for more complex detections, extending it through the use of more advanced techniques such as machine learning, deep neural networks, or anomaly detection models that are better suited to identifying subtle attack patterns. These methods could help reduce false positives and improve classification accuracy across network behaviors. Additionally, the automation framework can be scaled to support multiple virtual machines not just two, enabling the system to simulate larger and more realistic network environments. By combining machine learning algorithms with scalable and dynamic simulations the tool could become a real-time, adaptive SIEM-like threat monitoring tool.

AI Usage Declaration

The project used the help of ChatGPT AI [14] to summarize research articles for easier understanding, find flaws and bugs in the source code, and provide assistence when errors occurred.

Grammarly AI [15] was used to help with grammar and writing style in the written parts of the project, performing spell checks and making sure that the writing style aligns with the standards of a research paper.

References

[1] M. H. U. Sharif and M. A. Mohammed, "A literature review of financial losses statistics for cyber security and future trend," *World Journal of Advanced Research and Reviews*, vol. 15, no. 1, pp. 138–156, 2022, cited by: 160. [Online]. Available: https://www.researchgate.net/profile/Md-Haris-Sharif/publication/362363620_A_literature_review_of_financial_losses_statistics_for_cyber_security_and_future_trend/links/62edad4f505511283e94b5ed/A-literature-review-of-financial-losses-statistics-for-cyber-security-and-future-trend.pdf

[2] J. H. Mohiuddin Ahmed, Abdun Naser Mahmood, "A survey of network anomaly detection techniques, journal of network and computer applications," *Journal of Network and Computer Applications)*, pp. 193–198, 2019, cited by: 2242. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515002891

[3] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2019, cited by: 219. [Online]. Available: https://ieeexplore.ieee.org/document/8543584

[4] B. D. Bryant and H. Saiedian, "Improving siem alert metadata aggregation with a novel kill-chain based classification mode," *Computers Security*, vol. 94, p. 101817, 2020, cited by: 40. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S016740482030095X

[5] Positive Technologies (PT Security), "Going beyond the ordinary: how siem does incident detection, 2020," 2020, accessed: 2025-09-05. [Online]. Available: https://global.ptsecurity.com/en/research/analytics/incidents-siem-2020

[6] IBM Corporation, "IBM QRadar SIEM 7.4.3 documentation," 2025, accessed: 2025-10-21. [Online]. Available: https://www.ibm.com/docs/en/qsip/7.4?topic=SS42VS_7.4/com.ibm.qradar.doc/c_qradar_pdfs.htm

[7] VirusTotal and Contributors, "Yara documentation," 2025, accessed: 2025-10-21. [Online]. Available: https://yara.readthedocs.io/en/stable

[8] B. Alothman, "Raw network traffic data preprocessing and preparation for automatic analysis," p. 1–5, 2019, cited by: 6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8885333

[9] D. K. M. Dimolianis, A. Pavlidis and V. Maglaris, "Mitigation of multi-vector network attacks via orchestration of distributed rule placement," p. 162–170, 2019, cited by: 1. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8717838?casa_token=uYMajmqMGzwAAAAA:epXAV4K9o5nby9j07Aw0vy0Gujeea5UWifMZ33gvFb-DYlpzdEuVzCzaeGrUqmQ5FRoDbc47

[10] K. Lee, J. Lee, and K. Yim, "Classification and analysis of malicious code detection techniques based on the apt attack," *Applied Sciences*, vol. 13, no. 5, 2023, cited by: 47. [Online]. Available: https://www.mdpi.com/2076-3417/13/5/2894

[11] A. Yushko, R. Shevchuk, K. Łopaciński, M. Leszczynska, O. Yashchyk, and T. Yurchyshyn, "Shielding web application against cyber-attacks using siem," pp. 393–396, 2023, cited by: 3. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S016740482030095X

[12] European Institute for Computer Antivirus Research, "The eicar anti-virus test file," European Institute for Computer Antivirus Research, 2025, accessed: 2025-10-21. [Online]. Available: https://www.eicar.org/download-anti-malware-testfile

[13] Fortinet Inc., "DoS Prevention – FortiWeb Administration Guide," 2025, accessed: 2025-10-21. [Online]. Available: https://help.fortinet.com/fweb/586/Content/FortiWeb/fortiweb-admin/dos_prevention.htm

[14] OpenAI. (2025) Chatgpt. Used to assist in implementing and refining ideas. [Online]. Available: https://www.openai.com/

[15] Grammarly, Inc. (2025) Grammarly. Used to assist with grammar and writing style. [Online]. Available: https://www.grammarly.com/

A Appendix

Below (Figures 22, 23, 24, 25, 26, and 27) showcase victim and attacker playbook outputs during offense generation. The order is as follows: attacker playbook in action during the simulation of an offense involving a large number of TCP connections to an external internet resource; victim playbook in action during the same simulation; attacker playbook in action during the simulation of an SSH brute-force offense; victim playbook in action during the SSH brute-force simulation; victim playbook in action during the simulation of a malicious file detected but not deleted; and attacker playbook in action during the same malicious file simulation.

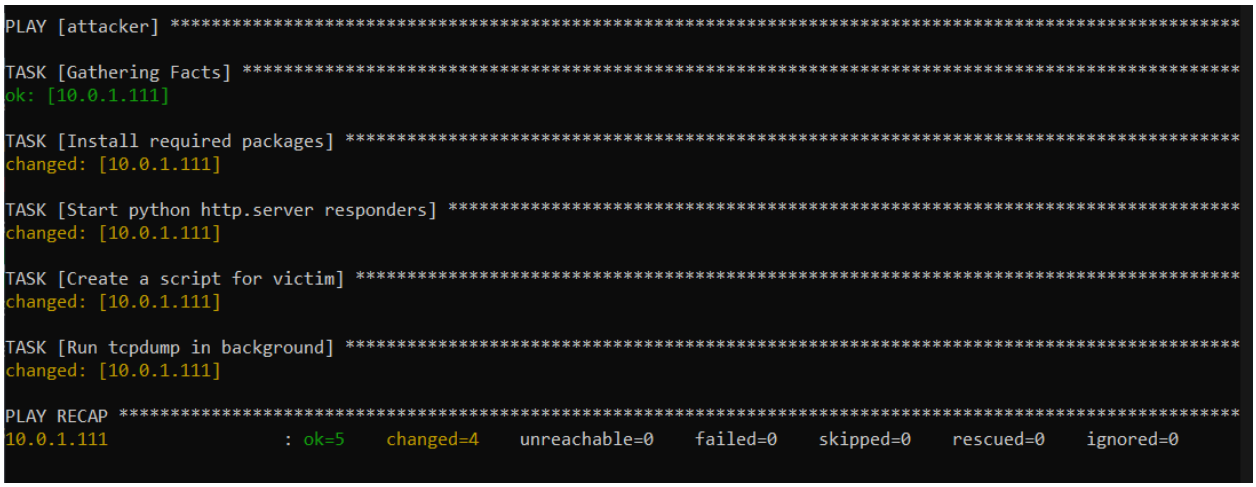

Figure 22. Attacker playbook in action during the simulation of an offense large number of TCP connections to an external internet resource.

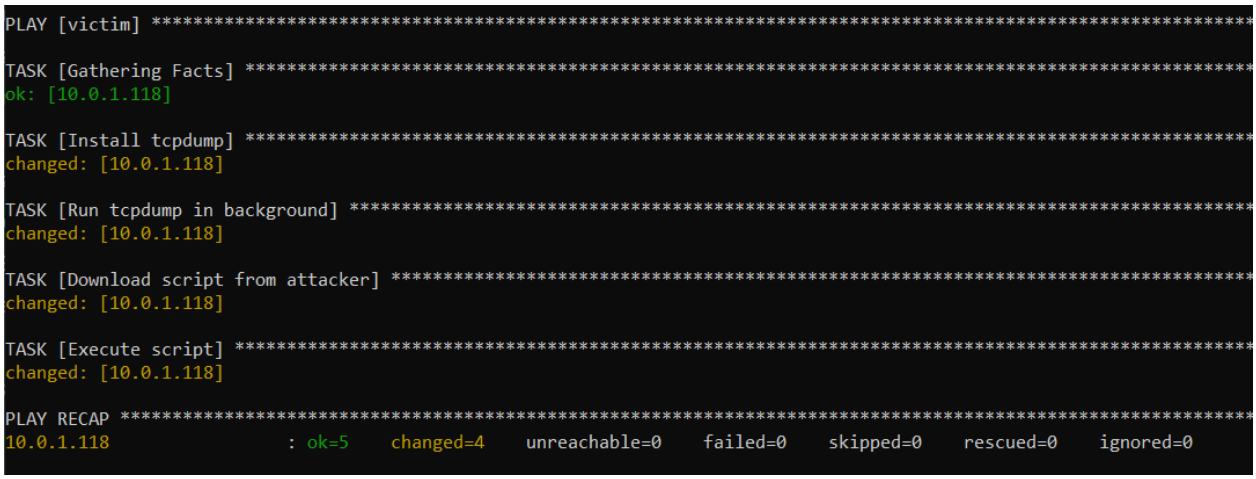

Figure 23. Victim playbook in action during the simulation of an offense large number of TCP connections to an external internet resource.

```
PLAY [attacker] ****************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [10.0.1.139]

TASK [Install tcpdump and sshpass] *********************************************************************************
changed: [10.0.1.139]

TASK [Run tcpdump in background] ***********************************************************************************
changed: [10.0.1.139]

TASK [Ensure SSH reachable] ****************************************************************************************
ok: [10.0.1.139]

TASK [Abort if SSH not reachable] **********************************************************************************
skipping: [10.0.1.139]

TASK [Brute force with random 4 letter passwords (25 attempts)] ****************************************************
changed: [10.0.1.139]

PLAY RECAP *********************************************************************************************************
10.0.1.139                 : ok=5    changed=3    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

Figure 24. Attacker playbook in action during the simulation of an offense SSH brute force.

```
PLAY [victim] ******************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [10.0.1.145]

TASK [Install ssh server and tcpdump] ******************************************************************************
changed: [10.0.1.145]

TASK [Ensure sshd is running] **************************************************************************************
ok: [10.0.1.145]

TASK [Create target user] ******************************************************************************************
changed: [10.0.1.145]

TASK [Run tcpdump in background] ***********************************************************************************
changed: [10.0.1.145]

PLAY RECAP *********************************************************************************************************
10.0.1.145                 : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 25. Victim playbook in action during the simulation of an offense SSH brute force.

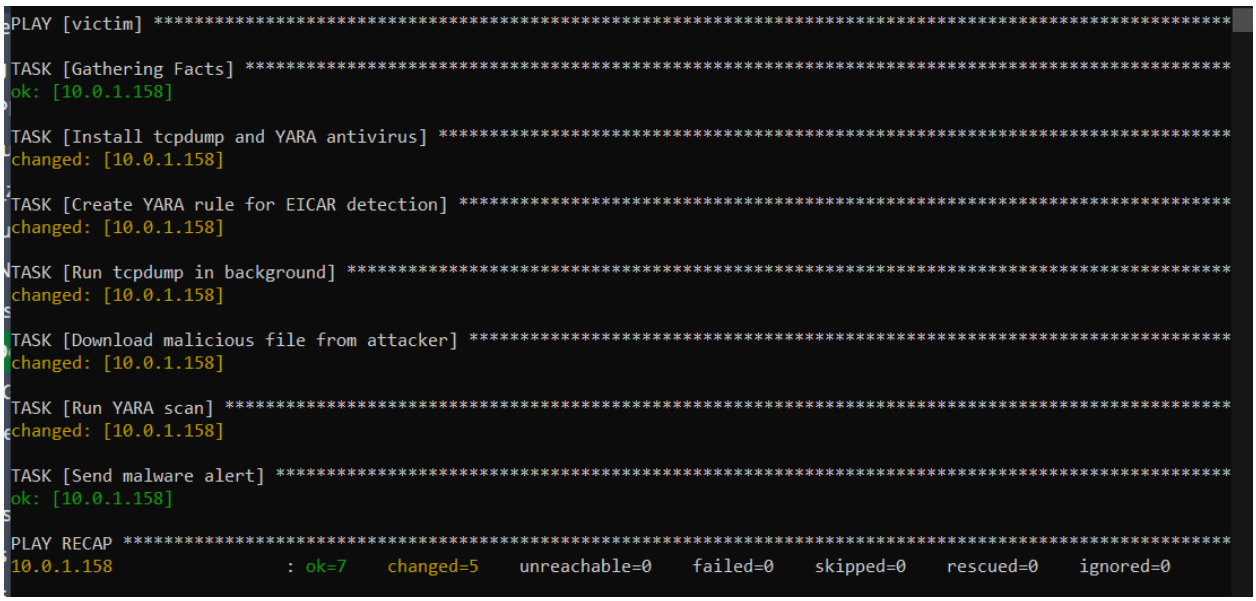

Figure 26. Victim playbook in action during the simulation of an offense malicious file detected but not deleted.

Figure 27. Attacker playbook in action during the simulation of an offense malicious file detected but not deleted.