COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Degree in Telecommunications Technology Engineering

Bachelor's final project

Attack Traffic Network Analysis of a Brute-Force Attack against a MySQL Server

Author
Elena Conderana Medem

Supervised by
Maximilian Stephan

Munich
July 2023

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Attack Traffic Network Analysis of a Brute-Force Attack against a MySQL Server

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

[illegible]

Fdo.: Elena Conderana Medem   Fecha: 02/ 07/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

[illegible]

Fdo.: Prof. Wolfgang Kellerer   Fecha: 03/ 07 / 2023

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Degree in Telecommunications Technology Engineering

Bachelor's final project

Attack Traffic Network Analysis of a Brute-Force Attack against a MySQL Server

Author
Elena Conderana Medem

Supervised by
Maximilian Stephan

Munich
July 2023

RESUMEN DEL PROYECTO

Esta tesis trata de sentar las bases para aplicar métodos basados en datos para el análisis del tráfico de ataque de una red. Las redes de comunicación proporcionan la estructura básica para el funcionamiento del mundo moderno, pero no sin enfrentarse a algunos retos. Las ciberamenazas, como los ataques de fuerza bruta, son un vector de ataque habitual para intentar obtener acceso no autorizado a dispositivos y datos. Debido a la singularidad de cada red, disponer de una visión personalizada de cada una de ellas es clave para prevenir y mitigar eficazmente el impacto de los ataques y las filtraciones de datos. La tesis utiliza un entorno de testeo de desarrollo propio que implementa un ataque de fuerza bruta de forma segura y observable. El banco de pruebas archiva en conjuntos de datos el tráfico del host y de la red central. Mediante el exhaustivo análisis exploratorio de datos a nivel de host y de red de ambas muestras de tráfico, la tesis identifica patrones para detectar un ataque de fuerza bruta basado en densidad de paquetes en el tráfico de red. Los modelos estocásticos de bloques también demuestran su capacidad para detectar ataques de robo de credenciales basados en fuerza bruta contra un servidor de MySQL.

Palabras clave: análisis de tráfico de red, banco de pruebas, ataque de fuerza bruta, servidor MySQL, Modelos estocásticos de bloques

1. Introducción

Las redes de comunicación constituyen una de las tecnologías que permiten al mundo moderno funcionar digitalmente y entorno a todo el planeta. Sin embargo, esta relevancia no está exenta de desafíos. Entre otros, las redes de comunicación se han convertido en objetivos de la ciberdelincuencia.

Los ciberataques pueden adoptar una gran variedad de formas, que difieren en las tácticas, técnicas y procedimientos en los que se basan. Los ataques de fuerza bruta son uno de los vectores más comunes para obtener acceso no autorizado a sistemas y datos, debido a su naturaleza simple de ensayo y error y a su alta efectividad. Para minimizar su impacto y garantizar la seguridad de las redes de comunicación es de suma importancia detectar y mitigar estos ataques lo antes posible.

Para prevenir los ataques es importante conocer la red en cuestión. Cada red tiene características diferentes y no pueden tratarse como una caja negra. Por lo tanto, los métodos de detección y mitigación deben adaptarse a una red específica para lograr su máxima eficacia.

2. Definición del proyecto

En este contexto, la tesis propuesta se centra en sentar las bases para aplicar métodos basados en datos para el análisis del tráfico de ataque de una red de un campus. El proyecto implementa un banco de pruebas de desarrollo propio, en el que se puede llevar a cabo un ataque de fuerza bruta contra un servidor MySQL de forma segura y observable. Se recopilan los datos de red reales de los hosts implicados y los datos de la monitorización central de la red. El conjunto de datos resultante se somete a un exhaustivo análisis exploratorio de datos a nivel de host y de red.

Los conocimientos adquiridos deberían contribuir a responder a la pregunta sobre "¿Cómo debe diseñarse la monitorización para permitir la detección de ataques a la red?", contribuyendo así directamente a la mejora de la seguridad de la red. La identificación de patrones en el tráfico de red podría allanar el camino para automatizar la supervisión y detección de ataques de fuerza bruta en una red de campus.

3. Arquitectura

El proyecto consta de tres fases principales: desarrollo del banco de pruebas; ataque y recogida de datos; y análisis de datos. La primera parte consiste en configurar el entorno de testeo con las máquinas virtuales. La máquina atacante utiliza el framework de Metasploit [met23] para realizar el ataque. Metasploit es un proyecto de código abierto orientado a las pruebas de penetración y es una de las herramientas más populares en este campo. Target1, la víctima del ataque, contiene el servidor MySQL con varias bases de datos. Una vez configurado el entorno, se procede a la aplicación del módulo auxiliar de metasploit para realizar ataques de login de fuerza bruta. Los comandos del ataque se programan en python para automatizar el proceso.

Después de codificar con éxito el ataque, se escribe el código para simular el comportamiento del usuario real trabajando con las bases de datos de MySQL. El script final combina tanto el ataque como el comportamiento del usuario. Durante la ejecución del script, los datos de los hosts implicados y los datos de la monitorización central de la red se recopilan en conjuntos de datos. Por último, una vez que los datos recopilados son suficientes, se procede al análisis para extraer información de valor. El análisis exploratorio a nivel de host y de red consiste en identificar comportamientos extraños en base a diferentes parámetros, como la frecuencia de paquetes, el tamaño de los paquetes o el número de bytes enviados. AwareNet [SKK22], un modelo de bloques estocásticos ponderados, analiza el tráfico a partir de los datos de monitorización central agrupando nodos. Los patrones y estructuras de comunicación se manifiestan en la pertenencia de un nodo a un grupo y en las relaciones entre grupos descritas mediante matrices de bloques.

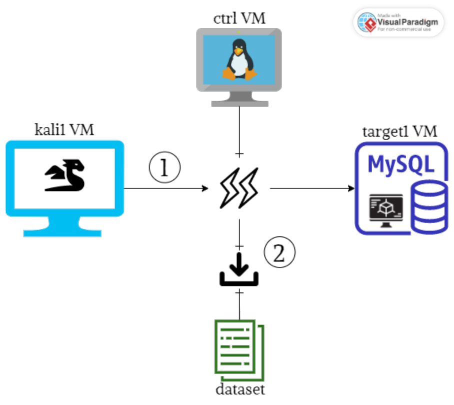


Figure 1: Testbed environment schema

La figura 4.1 representa un esquema del banco de pruebas. El diagrama muestra dos hilos principales correspondientes a los diferentes roles de las máquinas virtuales durante la fase de ataque. El primer hilo representa tanto el ataque de inicio de sesión por fuerza bruta como el uso por parte del usuario legítimo de las bases de datos MySQL, y la correspondiente recopilación de datos a nivel de host. El segundo hilo corresponde al almacenamiento de la monitorización central en conjuntos de datos.

4. Resultados

El proyecto adopta tres enfoques diferentes para analizar el ataque de inicio de sesión por fuerza bruta. El análisis a nivel de host revela que el ataque en contraste con el comportamiento normal, presenta un notorio aumento de la frecuencia de paquetes por unidad de tiempo en la red. Dicho comportamiento también se observa en el tráfico de la monitorización central. La figura **??** representa en azul la densidad de paquetes promedia de todas las conexiones transmitiendo en ese instante en la red. La zona azul claro marca el margen hasta la desviación típica. La línea naranja representa el flujo de paquetes entre el atacante y la víctima. Se observa que durante la franja correspondiente al ataque, los paquetes enviados por minuto superan el margen de la desviación típica, presentando un comportamiento fuera de los márgenes del tráfico esperado.

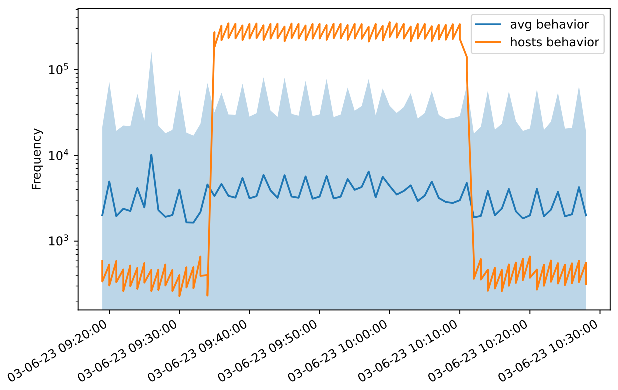


Figure 2: Densidad de paquetes por minuto durante ataque

Para concluir se demuestra la efectividad de detectar el ataque de fuerza bruta mediante un modelo de bloques estocásticos. Para ello AwareNet utiliza una serie de datos iniciales para calcular las log-verosimilitudes del comportamiento esperado de la red, que se sitúan entre -6 y -22 para la red en cuestión. Comparando el tráfico de red restante con esos valores, AwareNet intenta detectar comportamientos anómalos. La Fig. 5.24 y la Fig. 5.25 representan las log-verosimilitudes de los grupos a los que pertenecen el atacante y la víctima respectivamente. En las gráficas se aprecia perfectamente cómo ambas conexiones presentan un comportamiento absolutamente anómalo con valores muy alejados de los esperados por el modelo inicial.

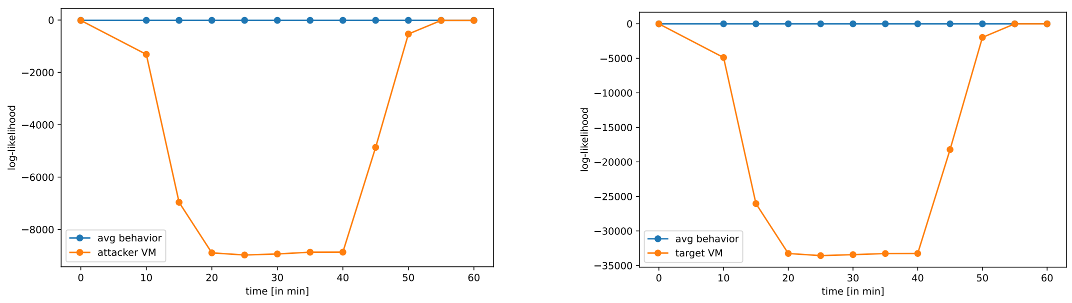


Figure 3: log-likelihoods of group 105 and 102 over time

5. Conclusiones

Los resultados adquiridos durante el análisis del tráfico de red ayudan a responder a la pregunta inicial sobre ”¿Cómo debe diseñarse la monitorización para permitir la detección de ataques a la red?”. Además, los gráficos resultantes de la aplicación del modelo de bloques estocásticos muestran que AwareNet es capaz de detectar ataques de fuerza bruta contra servidores MySQL. Como trabajo de futuro podría estudiarse la capacidad de este modelo para detectar otros ataques.

6. Referencias

[met23] Metasploit Documentation Penetration Testing Software, Pen Testing Security. `https://rapid7.github.io/metasploit-framework/docs/pentesting/metasploit-guide-mysql.html`, 2023.

[SKK22] Maximilian Stephan, Patrick Krämer, and Wolfgang Kellerer. AwareNet: using WSBMs for network traffic analyis. *Proceedings of the 3rd International CoNEXT Student Workshop*, pages 35–36, 2022.

ABSTRACT

This thesis tries to lay the groundwork for applying data-driven methods to network attack traffic analysis. Communication networks provide the basic structure for the functioning of the modern world, but not without facing some challenges. Cyberthreats, such as brute-force attacks are a common attack vector to try and gain unauthorized access to devices and data. Due to the uniqueness of every network, having tailored insight of every individual network is key to efficiently prevent and mitigate the impact of attacks and data breaches. The thesis uses a self-developed testbed environment that implements a brute-force login attack in a secure and observable manner. The testbed collects the host and central network traffic of a campus network into datasets. By scrutinizing both traffic samples with a thorough exploratory data analysis at host and network level, the thesis identifies patterns to detect a brute force-login attack based on packet frequencies in the network traffic. Stochastic Block Models also demonstrate their ability to detect credential stealing attacks based on brute-forcing against a MySQL server.

Keywords: network traffic analysis, testbed environment, brute-force attack, MySQL server, Stochastic Block Model

1. Introduction

Communication networks constitute one of the enabling technologies that allow the modern world to function digitally and around the globe. However, this relevance does not come without its challenges. Among others, communication networks have turned into targets of cybercrime.

Cyberattacks can take on a vast variety of forms, differing in the tactics, techniques and procedures they are based on. Brute-force attacks are one of the most common vectors to gain unauthorized access to systems and data due to their simple trial-and-error nature and high effectiveness. To minimize their impact and guarantee the security of communication networks it is of utter importance to detect and mitigate these attacks as early as possible.

To prevent attacks it is important to have insight into the network itself. Every network has different characteristics and cannot be treated as a black box. Detection and mitigation methods should therefore be tailored to a specific network to achieve their maximum efficiency.

2. Project definition

In this context, the proposed thesis focuses on laying the groundwork for applying data-driven methods to network attack traffic analysis. The project implements a self-developed environment, where a brute-force login attack can be carried out in a secure and observable manner. Ground truth network data from the involved hosts and the related data from central network monitoring is collected. The resulting dataset undergoes a thorough exploratory data analysis at host and at network level.

The knowledge acquired should contribute towards answering "How should monitoring be designed to enable detection of network attacks?", thus directly contributing to the improvement of network security. The identification of patterns in the network traffic could pave the way to automate the monitoring and detection of brute-force login attacks in a campus network.

3. Architecture

The project has three major phases: testbed environment; attack and data collection; and data analysis. The first part consists on setting up the testbed environment with the virtual machines. The attacker machine uses the Metasploit [met23] framework to perform the attack. Metasploit is an open-source project geared towards penetration testing and is one of the most popular tools in the field. Target1 hosts the MySQL server with various databases. Once the environment is set, the application of the auxiliary module of metasploit to carry out brute-force login attacks follows. The attack queries are scripted in python.

After successfully coding the attack, the code to resemble the behavior of the actual user working on the MySQL databases is scripted. The final script combines both the attack and the user behavior. During the execution of the script the data from the involved hosts and the related data form the central network monitoring is gathered in datasets. Lastly, once the collected data is sufficient, the analysis to extract valuable insight follows. The exploratory analysis at host and network level consists on identifying odd behaviors in different parameters, such as packet frequency, packet sizes or number of bytes sent. AwareNet [SKK22], a Weighted Stochastic Block Model, analyzes traffic from the central monitoring data by grouping nodes. Communication patterns and structures manifest in node to group membership and in group-to-group relations described through block matrices.

Figure 4.1 portrays a schema of the testbed environment. The diagram displays two main threads corresponding to the different roles of the virtual machines during the attack phase. The first thread represents both the brute-force login attack and the use by the rightful user of the MySQL databases, and the corresponding data gathering at host level. The second thread resembles the storing of the central monitoring into datasets.

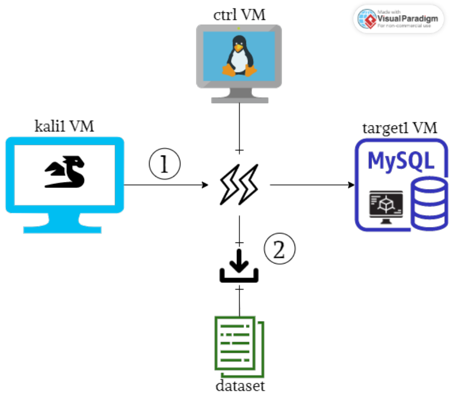


Figure 4: Testbed environment schema

4. Results

The project takes three different approaches to analyze the brute-force login attack. The host-level analysis reveals, that the attack in contrast to the normal behavior, presents a notorious increase in packet frequency per time unit. This behavior also manifests in the data flow from the central monitoring. Figure 5.14 illustrates with a dark blue line the average density of packets per minute of all connections communicating at every instant through the network. The light blue region corresponds to the standard deviation. The orange line depicts the average packets per minute between the attacker and the victim. During the interval that corresponds to the attack, the average packets sent per minute are beyond the standard deviation. The packet density experiences an unexpected amount of data flow that translates into outliers in comparison to the total network traffic.

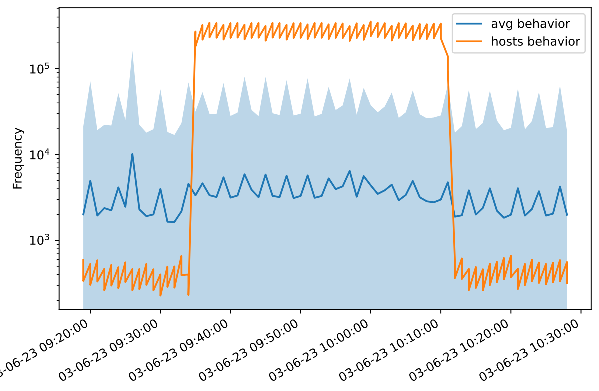


Figure 5: Average packets per minute during attack

To conclude Stochastic Block Models prove their capacity to detect brute-force attacks. AwareNet uses some initial data to calculate the log-likelihoods of the expected network behavior, which lie between -6 and -22. Comparing the remaining network traffic with those values AwareNet tries to detect abnormal behaviors. Fig. 2 and Fig. 3 represent the log-likelihoods of the groups the attacker and the victim belong to respectively. From the graphs depict perfectly how both connections present an absolut abnormal behavior.

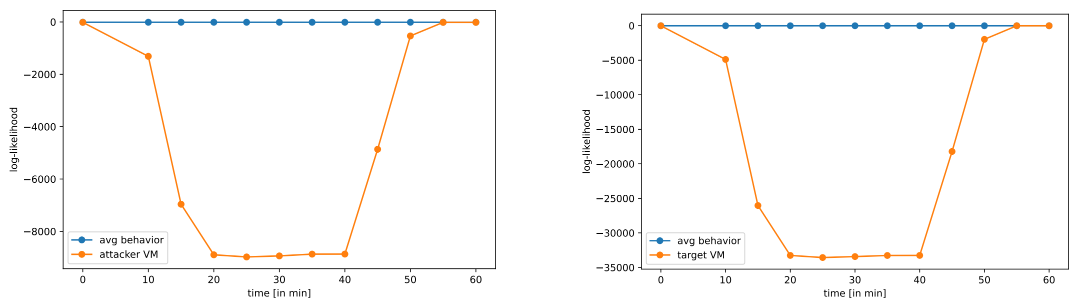


Figure 6: log-likelihoods of group 105 and 102 over time

5. Conclusions

The results help answer the initial question about "How should monitoring be designed to enable detection of network attacks?". Further, the resulting plots of the Weighted Stochastic Block Model showcase that AwareNet is capable of detecting brute-force login attacks against MySQL servers. Future work could explore the capacity of the model to effectively detect other attack types.

6. References

[met23] Metasploit Documentation Penetration Testing Software, Pen Testing Security. `https://rapid7.github.io/metasploit-framework/docs/pentesting/metasploit-guide-mysql.html`, 2023.

[SKK22] Maximilian Stephan, Patrick Krämer, and Wolfgang Kellerer. AwareNet: using WSBMs for network traffic analyis. *Proceedings of the 3rd International CoNEXT Student Workshop*, pages 35–36, 2022.

Contents

List of Figures

Listings

Chapter 1

Introduction

Communication networks are the backbone of today's society. They provide a critical infrastructure in the everyday life of the majority of the population. From banking transactions and social interaction to controlling vital infrastructures and satellite monitoring. Communication networks constitute one of the enabling technologies that allow the modern world to function digitally and around the globe. However, this relevance does not come without its challenges.

Among others, communication networks have turned into targets of cybercrime. Cyberattacks can take on a vast variety of forms, differing in the tactics, techniques and procedures they are based on. Some prominent examples are denial of service, man in the middle attacks, or gain of unauthorized access to machines and data. Brute-force attacks are one of the most common vectors to gain unauthorized access due to their simple trial-and-error nature and high effectiveness. One of its main purposes is credential stealing. Credential theft allows an attacker to gain access to machines and steal critical data. The main advantage of credential based attacks is, that the attacker bypasses an organizations security measures, such as firewalls, effortless. To minimize their impact and guarantee the security of communication networks it is of utter importance to detect and mitigate these attacks as early as possible.

To prevent attacks it is important to have insight into the network itself. Networks differ in the amount and kind of traffic they process, the size of the system and the amount and interconnection of the devices included. Given the different characteristics of any network, they cannot be treated as a black box. Detection and mitigation methods should therefore be tailored to a specific network to achieve their maximum efficiency.

In this direction, the proposed thesis focuses on laying the groundwork for applying data-driven methods to a campus' network attack traffic. The project implements a self-developed environment, where a brute-force login attack can be carried out in a secure and observable manner. Ground truth network data from the involved hosts and the related data from central network monitoring is collected. The resulting dataset undergoes a thorough exploratory data analysis at host and at network level. The knowledge acquired should contribute to the development of methods for identifying traffic patterns of a brute force-login attack, thus directly contributing to the improvement of network security. The identification of patterns in the network traffic could pave the way to automate the monitoring and detection of brute-force login attacks in campus' network.

Chapter 2

Background

The following chapter explains the technologies implemented in the project and reviews different approaches to tackle the detection and mitigation of network attacks. Section 2.1 reviews the attack under study as well as the technologies employed to extract valuable insight. Section 2.2 analyzes some of the researches carried out on the field and exposes their strengths and weaknesses.

2.1 Description of the technologies

Network attacks have become a critical issue for the security of computer networks. Communication networks are constantly exposed to cyberthreats and are often the victim of intrusion attacks. A set of events which have the ability to compromise the principles of computer systems: availability, authority, confidentiality and integrity. One of the most prevalent intrusion attacks that threaten computers connected to the network are brute force attacks. The detection and mitigation of network attacks takes place through monitoring and analysis of relevant traffic data at network or at host level. The variety of models to analyze monitored data is countless. The thesis implements some basic exploratory data analysis and Stochastic Block Models to that end.

2.1.1 Brute-force attack and Credential stealing

A brute force attack uses a trial-and-error approach to systematically guess login information and encryption keys to gain unauthorized access to systems. Although it is an old attack method, brute forcing remains effective. There exist different types of attack. Simple brute-force attacks, where the attacker tries to logically guess the credentials without assistance of any available tools or means. Dictionary attacks run dictionaries containing lists of passwords on a chosen target. Hybrid brute-force attacks, where the attacker combines external means such as preexisting dictionaries with their logical guesses. Reverse brute-froce attacks reverse the attack by knowing the password and trying to guess the corresponding username. Lastly, credential stuffing uses a known username-password tuple in tons of other websites and applications since users often reuse login credentials.

This thesis uses a hybrid brute-force attack. The password list contains 1.000.001 entries plus the correct password. The list contains a selection of the most frequent passwords, variations and permutations of those passwords and passwords related to the chair of communication networks and their abbreviation. The reason is that the virtual machines belong to that chair, therefore it is logical to think that the password may be related to an acronym or a word related to it, facilitating the access to any member of the chair. The passwords vary from only letters, to the combination of letters with special characters and numbers and it uses capital and lowercase characters.

Credential theft is among the popular brute force attacks, since the actor acquires the account privileges of the victim. Once successful, the attacker can access the system masquerading as the legitimate user and remain inside until detected. The intent is to steal user credentials that will grant access to critical data, enable access to other devices on the network, install back doors, wipe backups and data, move laterally and gain remote access to third-party services, among others. Credential theft often takes place in form of internal attacks. Threats provoked by people with authorized access to or knowledge of an organization's resources that deliberately or accidentally expose or help expose confidential information, intellectual property, systems, money and more.

2.1.2 Host-based and Network-based intrusion detection systems

An Intrusion Detection System (IDS) monitors network traffic and issues alerts when it detects suspicious activities or known threats. IDSs sniff packets and detects anomalies in network traffic. The roles comprise system monitoring, assessment of firewalls, servers, etc., researching system logs and identify underlying patterns in the data flow of the network to detect and tackle cyberthreats.

Host-based intrusion detection systems (HIDS) monitor and analyze local incoming and outgoing traffic of a specific host. Network-based intrusion detection systems (NIDS) involve the entirety of the system and are therefore usually located in main intersection points, such as routers or servers. NIDS examine the traffic flow of all systems and devices in the network. A frequently used tool to capture all packets in the network traffic is tcpdump[tcp23]. A powerful command-line packet analyzer that compiles a pcap file of raw data derived from its sniffing process.

These studies take place on physical testbeds to train and test cyberattacks.

2.1.3 Testbed environment

A testbed is an environment to carry out replicable and rigorous testing of experimental research and new product development in a safe and controlled environment. The testbed can be tailored to the needs of the project at hand. Testbed environments focus on a subset from the total real-world system that can range from small pieces to complete prototypes of a subsection. They are used to gain insight into specific aspects the project at hands aims to study. The rationale behind using a testbed environment is to represent a realistic hardware-software environment where the testing results are actually representative for the system under scrutiny.

2.1.4 Stochastic Block Models

This thesis uses stochastic block models (SBM) to try to recognize a brute force attack within a campus' network data flow, given that community detection is generally very insightful in evaluating the structure of large complex networks. SBMs are unsupervised random graph models used in statistical analysis for community structure identification and clustering purposes. It is currently a useful benchmark in statistics, network science and ML to retrieve community structure in graph data. This approach uses properties of edges within and between communities, which makes it more suitable than simple clustering to understand networks.

Let n be the amount of data points, referred as nodes, in the set. Stochastic Block Models partition the nodes into k disjoint subsets called communities. Each node is interconnected to the remaining nodes through particular edge densities. The nodes belonging to the same community share some underlying characteristic. The point of differentiation towards simple clustering is the description of edge probabilities between groups through a symmetric block matrix of size kxk [FB19]. Figure 2.1 represents an example of the resulting graph and block matrix of a SBM model. The colors in the graph denote the group each node belongs to. The matrix on the figure depicts in a grey scale the group-to-group edge probabilities.

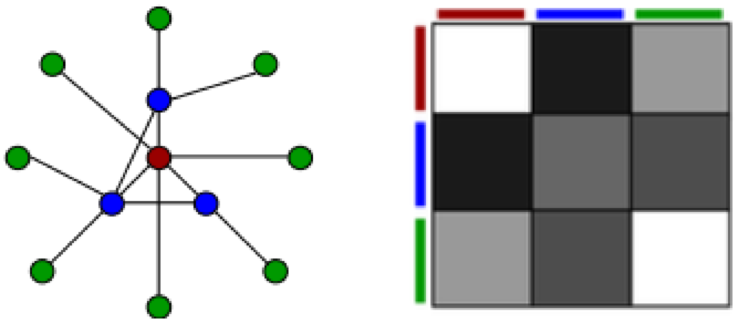


Figure 2.1: Node-to-group membership and group-to-group block matrix of SBM [FB19]

2.2 Current research

To assess the efficiency of intrusion detection systems there exists a vast variety of intrusion detection datasets (IDSs) and testbed environments. Platforms to conduct replicable testing of computing tools and new technologies and algorithms. The IDSs are extensively used for attack prediction approaches and anomaly detection in networks. Some publicly available benchmark datasets for the field of network security are the: KDD Cup '99, NSL-KDD, Kyoto 2006+, UNSWNB15, CIC-IDS2017 and CSE-CIC-IDS2018 datasets.

Both the KDD Cup '99 and its upgraded version, the NSL-KDD dataset, are comprised of over 40 features and can distinct between 25 attack types. But their inability to reflect the modern environment makes them unsuitable for current real network traffic analysis[GB19].

The CIC-IDS2017 and its newer version, the CSE-CIC-IDS2018, implement a complete network configuration (including Modem, Firewall, Switches, Routers an a variety of operating systems). The implementation topology tries to recreate a real client-server network, but the variety of protocols and attacks compared to real-world traffic is very limited. Real network conditions such as packet loss and different TTLs and applications like social networking are not present in these datasets. These shortcomings lead to an oversimplified scenario compared to the real world [RCCL22].

In contrast to the previous datasets, the UNSWNB15 and the Kyoto 2006+ reflect modern low foot print attacks. The former is a hybrid of real modern and contemporary attack activities of network traffic. The latter was developed in the real time environment between 2006 and 2009. But although the attack families of both datasets address current attacks, the Kyoto 2006+ does not provide information about the attack type[MS15].

Datasets are one of the main components to train network attack detection systems and judge the reliability of its algorithms. In 2021 Aljabri [AAM+21] reviews and analyzes research studies that use contemporary intelligent models to detect different cyberattacks. The majority of the models were trained on the KDD Cup '99, NSL-KDD, Kyoto 2006+, UNSWNB15, CIC-IDS2017 and CSE-CIC-IDS2018 benchmark datasets.

The studies use supervised and unsupervised Machine Learning (ML) and Deep Learning (DL) algorithms. The ML models include classification regression; clustering techniques such as logistic regression and random forest; support vector machines; naÏve Bayes; and K-nearest neighbor. The Deep Learning techniques mainly concern the development of different neural networks such as deep neural network or multi-layer perceptron.

The algorithms learn from both labeled and unlabeled data. The use of labeled data enables the analysis and detection of insider threat, DDoS attacks, phishing attacks, malware attacks, zero-day attacks and botnet attacks. The aim of using unlabeled data is to detect intrusions at network and at DNS level and to classify malicious traffic to its corresponding attack.

The techniques exposed in the different research papers have proven to have very high accuracy within the specific domains. But non of the techniques has proven useful in mitigating all kinds of network attacks [AAM+21]. The results of the research demonstrate the existing gap between controlled environment and real-world network attack detection.

In contrast to the previous research papers, Malecot et al. [ELML08] and Najafabadi et al. [NKK+11] focus their research on effectively identifying brute force attacks. Malecot et al. [ELML08] aim to detect distributed brute force attacks through information visualization. Each local host generates a quad tree. A mapping structure that resembles the hosts attempting to connect to the local host in question. Coordinated attackers should appear in the quad trees of multiple hosts enabling their detection. However, this approach has two major downsides. It requires a network expert's analysis and it is unable to automatically detect the attacking hosts.

To find simpler, more effective and automatic approaches Najafabadi et al. [NKK+11] study the application of machine learning methods to detect brute force attacks at network level by studying traffic data. The dataset to train the classifiers contains real world flow data labelled by network experts. The investigation trains 4 different models and evaluates the results of each of them based on average AUC values obtained across 4 runs of 5-fold cross-validation. The conclusions state that ML algorithms can achieve pretty accurate predictions in the detection of SSH brute force attacks. Additionally, the study tries to include ports in the dataset. Though results show that ports can improve the classification accuracy, they also suggest that classifiers, such as decision trees, can neglect other meaningful features when building the model. This tread off can lower the performance in case of new ports due to the categorical nature of this ports. The wide range of potential values they can take, biases the decision tree classifier to develop specific rules for each port, instead of generating more general rules based on other discriminating features. The challenges of this research lie within correctly choosing the most appropriate features to train the classifiers, the need to label traffic data to train the models and collecting additional network traffic to evaluate the performance of the fitted models on real-world unknown data.

A solution to avoid having to label flow data is to use unsupervised machine learning algorithms. These models identify underlying hidden structures of unlabelled data. Clustering is a very wide spread technique in the field. Cluster analysis groups data points that share similarities through the dataset to reveal hidden connections and patterns that are undetectable to infer from single data points. According to [KGB20] unsupervised learning has a higher detection rate than supervised learning; however, they are prone to a high false-positive rate.

Wang et al. [WYWA17] use a clustering technique called Seed-Expanding (SE). This algorithm employs the Two-Seed-Expanding network traffic clustering scheme, which clusters attack traffic into phases. Attacks are divided into multi-steps and all attacks are presumed to share default steps, allowing the algorithm to detect attacks before they damage the system. Experimental results show that the implementation of discretization methods and asymmetric binary attributes to process traffic data, greatly improves the clustering performance of SE. Seed-Expanding manages to outdo K-Means and other seed-expanding with different seed numbers. The remaining challenge for the seed-expanding model in this research is to identify attack flows from normal flows based on the clustering results.

Another clustering technique to detect network attacks is the MeanShift algorithm. A non-parametric clustering technique that tries to find the dense areas in a set. Non-parametric clustering means there is no need to specify the amount of clusters or their shape. Since it automatically detects clusters based on data density, the development of empty clusters is minimal. Kumar et al. [KGB20] apply the MeanShift algorithm with the KDD 99 dataset. The research throws promising results achieving high performance rates of the algorithm, but solely tested on the KDD 99 set. The results could be non-representative for other datasets. Further work could involve analyzing the application of other normalization approaches to the dataset, that could potentially improve the detection rate and accuracy of the MeanShift algorithm.

Roeling et al. [RN18] apply Stochastic Block Models (SBM) to botnet data with the aim of identifying infected clusters without the need of a labelled dataset.

The rationale is to apply an unsupervised method to discover blocks of nodes in the network given the connectivity pattern and identify latent malicious traffic. Simulation results show that SBM can be insightful in networks with multiple infected users visiting the same addresses, whether or not malicious. However, in real world applications, the performance is notoriously lower, due to some limitations in the training of the stochastic block model. The training data might not provide an adequate snapshot of real botnet network activity, since only one cluster includes botnet data and is entirely treated as infected in the simulation. In real life multiple clusters or nodes across clusters would be infected. The lack of this feature would lead to a very high false positive rate in a real life network. Further, the data comes from separate environments. Instead of collecting the data from the same setting, the botnet data is collected with Virtual Machines (VM) and afterwards put together through IP mapping to resemble one network.

Stephan et al. [SKK22] deploy AwareNet, a system based on Weighted Stochastic Block Models (WSBMs), in a campus network. In the study AwareNet learns an internal representation of the network structure to detect scanning attacks that they initiate. Results show that AwareNet successfully detects anomalies like targeted host scans.

Chapter 3

Description of the Project

3.1 Motivation

External threats are what usually comes to people's minds when discussing cybersecurity. But external attackers are not the only threat modern organizations need to worry about. Malicious, negligent and compromised users are a growing risk. The 2022 Cost of Insider Threats Global Report [Ins22a] reveals that insider threats have increased in cost, frequency and time to contain since 2020. Over the past two years incidents have risen 44% with costs up to $15.38 million. There exist three insider threat profiles, negligent insider, which is the root cause of most incidents; malicious insiders; and credential theft incidents, which have almost doubled since 2020 and are the costliest to remediate.

Credential theft remains one of the top attack methods employed by hackers. The last report from the Ponemon Institute [Ins22b], a pre-eminent research center dedicated to privacy, data protection and information security policy, shares that stolen or compromised credentials remain the most common cause of data breaches. Figure 3.1 portrays the average cost and frequency of the most common attacks in 2022. Credential theft was the most common primary attack vector, making up to 19% of successful attacks in 2022. The average cost of attacks caused by credential theft amounted to $4.50 million. The most critical factor to reduce the damage of a security breach is time. Time to discover and to disarm and isolate the intruder are strongly correlated to the damaging impact of a breach. Figure 3.2 shows that attacks initialized through credential theft have the longest lifecycle. It took an average of 243 days to identify the breach and another 84 days to contain it in 2022, which is 16.6% greater than the overall mean time to identify and contain a breach. Credential stealing is one of the most appealing initial attack vectors, since it allows hackers to operate undetected throughout the network for the longest average time. This thesis aims to develop mechanisms to detect credential theft through brute-force attacks. Hence, trying to contribute to mitigate its impact on organizations' data breaches and monetary losses.

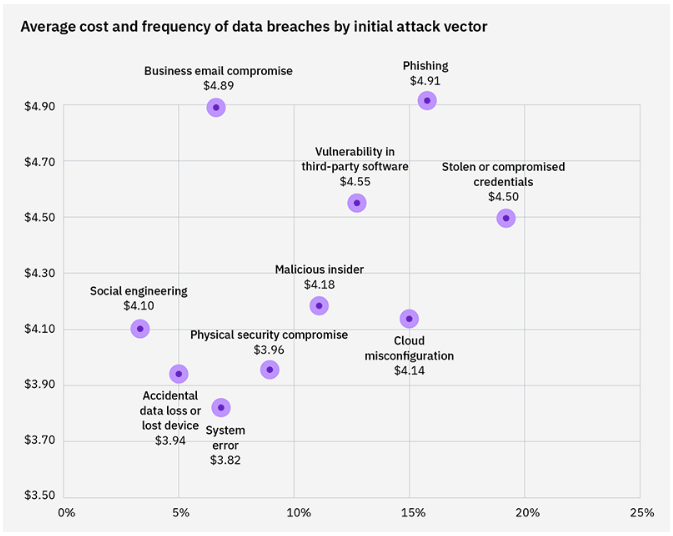


Figure 3.1: Average cost and frequency of data breaches by initial attack vector [in USD millions]

There exist two main approaches to detect malicious activities and ensure the intended functioning of a network and the secure transfer of user data. The alternatives comprise monitoring data at host or at network level. A key limitation of host-based mechanisms is their limited view of the network, which incapacitates them to detect distributed attacks that are prevalent in today's world. Network-based detection is more scalable and provides additional protection to hosts. However, encrypted traffic data and high speed networks make detection only more challenging. This study will use both approaches to profit from their respective upsides. The host-based approach will monitor the activity of the host. The aim is to find out if a brute-force login attack against a MySQL server leaves any tracks of anomalous behavior in the host flow data. The network level analysis will monitor the traffic passing through the central monitoring. The objective is to detect a brute-force login attack within the campus' network traffic.

The monitoring at host level, but specially at network level, is most insightful if all the traffic under analysis stems from the same environment as is the case in real life. One of the major drawbacks of [RN18] is that the data is gathered from separate environments and afterwards combined through IP mapping to resemble a single network. To prevent the combination of different data flows, the thesis automates the brute-force login attack and samples possible behaviors of the real user working on the MySQL server before and after the attack. The purpose is

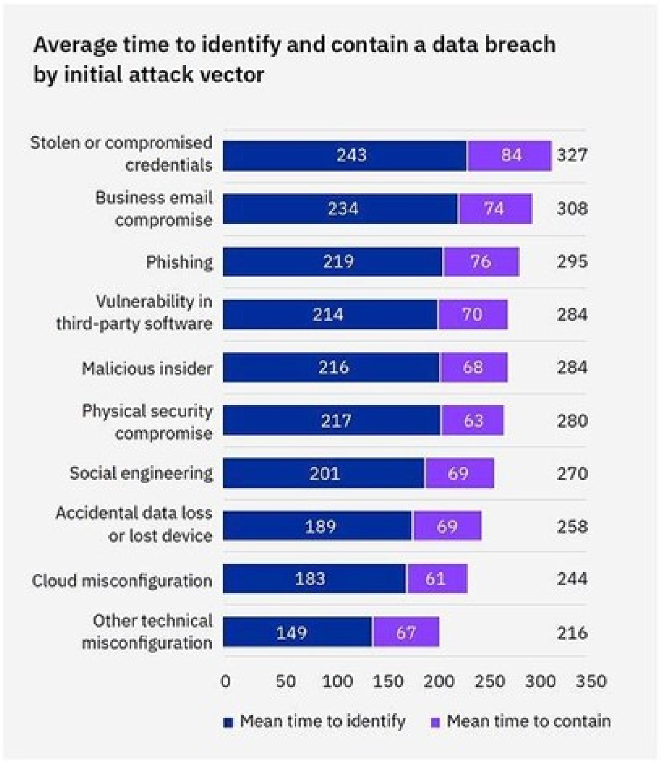


Figure 3.2: Average time to identify and contain a data breach by initial attack vector

that the malicious and non malicious traffic originates in the same network and under the same circumstances. This avoids the problems or conflicts that can arise when putting together data from different environments or when neglecting the existing links between traffic generated within a network. Traffic data is inherently dependent, due to unobserved latent factors that act locally on each network. Disregarding these dependencies can provide limited validity to the detection of abnormal network behaviour or other malicious traffic.

Another problem exposed in research papers such as [KGB20] is the utilization of benchmark intrusion detection datasets to train models that report excellent performances of their detection mechanisms on training and validation datasets, but have not been tested in real life scenarios. The purpose of datasets is to store information. However, sometimes it might be necessary to adjust the amount or type of data collected to answer the question "what should be monitored?". The KDD Cup '99, the Kyoto 2006+ and the CSE-CIC-IDS2018, for example, are very standardized datasets in the field of cybersecurity. But, as their name implies, they store fixed sets of data. The specificity and immutability of the data may turn them uninteresting or useless for certain purposes or circumstances. In addition, for some of the available datasets the testbed environments are not accessible for researchers. The data is therefore collected in an unknown environment and may not be representative for other networks, potentially misleading about the efficiency of the methods tested once implemented in real networks.

Datasets are contributing to develop algorithms and methods to detect network attacks. But their contribution could have a higher impact if researchers had access to the testbeds. Monitoring the environment and network traffic directly generates tailored datasets. Customed datasets give real actual insight into what is specifically taking place in the network, leading to the development of more efficient detection methods. Aljabri et al. [AAM+21] address the existing gap between controlled environments and real world network attack detection systems. Training models on fixed sets of data gathered in unknown networks often do not reliably represent conditions suitable to the later implementation of the algorithm.

The testbed environment of the thesis mimics the internal attack scenario of a campus network with three major goals: provide a topological description on how a credential theft occurs; achieve attack pattern extraction from raw sniffed data; and establish attack pattern identification as a parameter to visualize real-time attacks at host and network level. The advantage of using a self-developed testbed is to have all knowledge around the traffic generated, the conditions of the network, the devices involved, etc. and to have the capacity to unlimitedly apply any changes to make it more suitable to any needs. The stochastic block model trains with data directly collected from the central monitoring of the network. Given the real life conditions of its training, the performance should not experience much fluctuation, hence closing the aforementioned gap between controlled environments and real world network data.

Empirical results show that machine learners are quite successful in detecting brute force attacks with high detection rate and low false alarm. Still, the seed-expanding model of Wang et al. [WYWA17] exposes the main challenge for many of these researches. The inability to distinguish between attack flows and normal flows. Additionally, researches like Najafabadi et al. [NKK+11] use different machine learning classifiers, such as decision trees, that rely heavily on the availability of a labelled dataset and on a closed set of features to detect a brute force attack. Having to label the data and choosing the features to train the algorithm are major disadvantages. It creates the need for a network's expert to properly label the data and to accurately and correctly choose the features, which supposedly are more relevant or insightful to detect the attack.This can potentially lead to the disregard of other meaningful features.

The thesis applies stochastic block models to identify infected clusters without the need of a labelled dataset or a set of predefined features. The rationale is to apply an unsupervised approach to discover blocks of nodes in the network given the connectivity pattern and identify latent malicious traffic automatically. Unlike Malecot et al. [ELML08] the investigation provides a new insight of brute force attacks with the main goal of coming up with automatic attack patterns visualization that may help the network administrator to analyze easily any similar attacks.

Lastly, the motivation behind targeting a MySQL server is the widespread usage of MySQL servers that makes it a frequent target of attacks. MySQL alongside Oracle is the most popular database management system worldwide. Many large companies such as Facebook, NASA, Google and Boeing, as well as small businesses and individual developers use MySQL [Ora23]. The reason is that MySQL is very versatile and easy to use. From powering a simple website to running large and complex e-commerce sites, MySQL has endless applications.

3.2 Project objectives

The goal of the thesis is to lay the groundwork for applying data-driven methods to network attack traffic, through the collection and analysis of network flow at a campus network.

To achieve the final purpose of the project some previous stages need to be fulfilled. The first step consists on setting up a testbed environment where known attacks can be carried out in a secure and observable manner. In this environment attacker hosts target VMs with dedicated vulnerabilities using pentesting tools like Metasploit. The attack is scripted in Python to allow the automatic execution at any time.

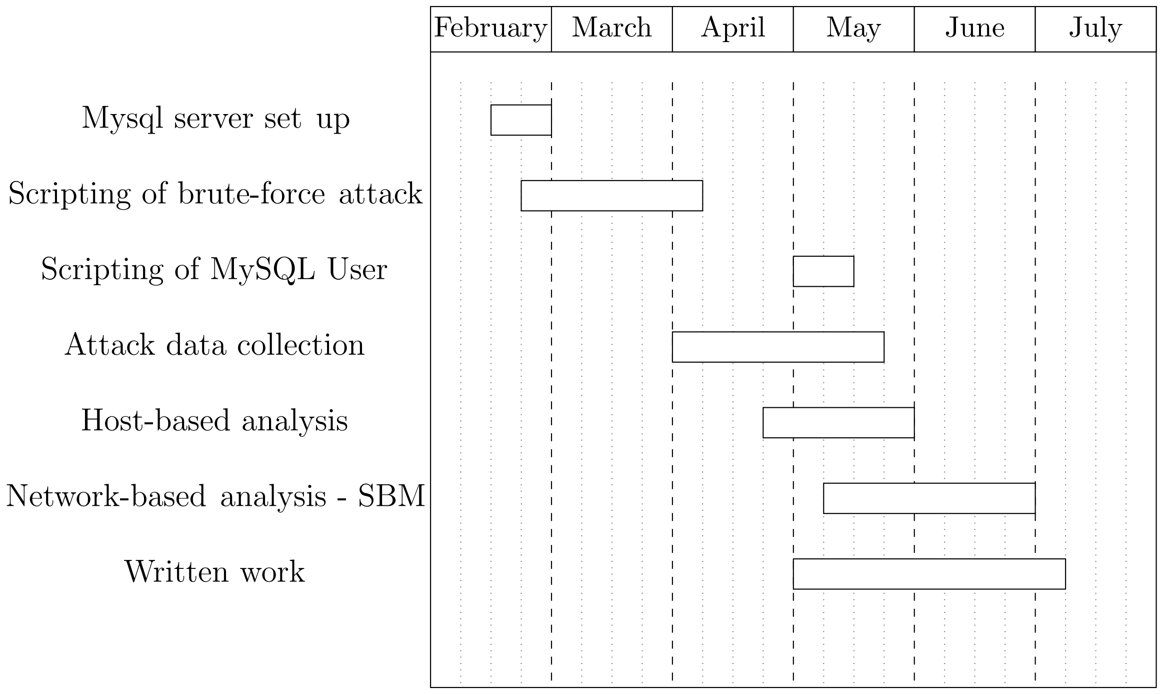
Ground truth network data from the involved hosts and the related data from central network monitoring is collected and should result in an extensive dataset containing network traffic from a brute-force login attack. The data needs to be processed and prepared to be suitable for its analysis.

As last step a thorough exploratory data analysis of the collected data takes place. The results should support identifying suitable data-driven methods for autonomous network attack detection and help towards answering the question "How should monitoring be designed to enable detection of network attacks?".

3.3 Methodology and Planning

The project has three major phases: testbed environment; attack and data collection; and data analysis. The first part consists on setting up the testbed environment with the virtual machines. Within the environment the attacker VM needs the installation of the metasploit framework to perform the MySQL brute-force login attack. The target VM needs to install the MySQL server that will suffer the attack. Once the environment is set the application of pentesting tools to carry out attacks follows. The attack queries are scripted in python. With the script the attack can run automatically at any time with a simple command. After the success of the attack, the code to resemble the behavior of the actual user working on the MySQL databases is scripted as well. The final script combines both the attack and the user behavior after working as expected independently.

During this phase the data from the involved hosts and the related data form the central network monitoring is gathered in a dataset with tcpdump [tcp23] and caplon respectively. Lastly, once the data collected is sufficient, the analysis to extract valuable insight follows. The exploratory analysis at host and network level uses the python library matplotlib [mat23]. This part consists mainly on visualizing the behavior over time, the sizes of the packets and the protocols involved in the network flow. Stochastic block models analyze traffic data from the central monitoring data. Throughout the development of the thesis the results, challenges and conclusions met are written down in a report. The following Gantt diagram roughly portrays the time distribution of the course of actions throughout the whole project.



3.4 Economic Estimation

An economic estimation is a crucial step to grant the feasibility of the thesis. The estimation considers all costs incurred during the development of the project and writes them off over the duration of the project. The first project-related issues began in November 2022. The real work though did not start until February 2023. For that reason the timespan considered for the project starts in February 2023 and lasts until June 2023.

The elements employed during the thesis are the computer, where the whole project was scripted, visualized and set up; the server of the campus network; and different tools, libraries and applications for the brute-force attack, data gathering and visualization. Still, only two of the elements mentioned have a contribution to the economic estimation. The laptop had an acquisition price of 1.082,31€ and a lifespan of 6 years. Distributing the price of the lifespan of the laptop the amortization price amounts to 180,39€/year. Since the duration of the project is of 5 months, the total cost is of 75,16€. The other element that adds to the cost of the project is the server of the campus that hosts the virtual machines. To the advantage of this project the server of the campus wrote off last year and is pending to be exchanged by a new one. Hence, the contribution to the economic estimation is cero. All programming and pentesting tools used throughout the whole project, such as matplotlib, metasploit, AwareNet and tcpdump, are publicly available and do not add any costs either.

Lastly, the labour costs need to be considered. In Germany the hourly salary for a junior engineer adds up to 26.92€/h. Taking into account that the estimated invested time approximates 330h, the labour costs reach 8.883,60€. Taking into account the aforementioned costs, the total economic estimation of the thesis amounts to 8.958,76€. It may seem a fairly high cost for a Bachelor's Thesis. However, it should be bared in mind that the thesis resembles an investment in the enhancement of the security of the campus network and in the application of probabilistic algorithms for the detection of network attacks.

Chapter 4

Architecture

The project implements a brute-force login attack to generate attack traffic and analyse it for pattern recognition. The attack has three parts. An initial run, that resembles the legitimate user working on the MySQL server. The proper attack, where the password is cracked and again the behavior of a legitimate user. The purpose is to generate "normal" and "attack" traffic on the network. Having both types of traffic enables the comparison between behaviors and facilitates the identification of odd behaviors and patterns. During the whole execution the VMs store the data for the posterior data analysis.

4.1 Implementation

The thesis uses a small-scale network testbed consisting primarily of 3 virtual machines: attacker as kali1, control as ctrl and victim as target1. In addition to the hardware, the testbed uses supporting software such as MySQL, Metasploit and tcpdump. The control and target virtual machines use the Linux operating system. The attacker machine, kali1, uses Kali Linux and runs the Metasploit [met23] framework on top of it. Metasploit is an open-source project geared towards penetration testing and is one of the most popular tools in the field. Target1 hosts the MySQL server with various databases. Tcpdump runs on the attacker VM to gather the data for the host level analysis. The control VM collects the network traffic from the central monitoring and pseudomizes it with a docker container. The central monitoring uses caplon and special capture hardware to collect the traffic mirrored from the core switches of the datacenter. Caplon provides an extensive overview of all the traffic in the network and complete control over IT and OT-infrastructures.

Figure 4.1 portrays a schema of the testbed environment. The diagram displays two main threads corresponding to the different roles of the virtual machines during the attack phase. The first thread represents both the brute-force login attack and the use by the rightful user of the MySQL databases, as well as the corresponding data gathering at host level. The second thread resembles the data collection and pseudonymization that stores the data from the central monitoring into datasets.

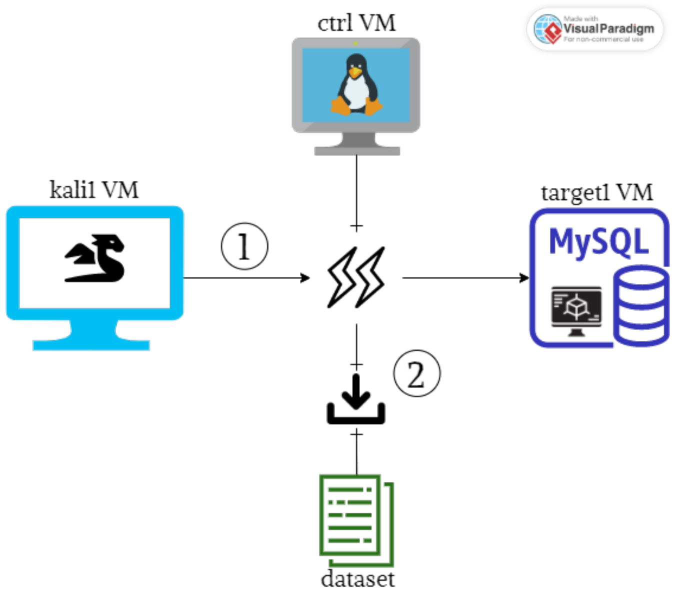


Figure 4.1: Testbed environment schema

4.2 Algorithms

The thesis makes use of different tools to accomplish the goals. The different implementations of algorithms and tools are further explained in the following section. The section is divided in 4 parts. The setting of the brute-force attack, the scripting of the attack and normal behavior, the visualizations with python and lastly the application of Weighted Stochastic Block Models.

4.2.1 Metasploit Framework

Msfconsole is one of the most frequently used command-line user interfaces to access the Metasploit framework and work with all the available tools for exploiting vulnerabilities, collecting data or scanning targets. There are two types of modules in Metasploit, exploit and auxiliary modules. Though none of the modules provides a script to gain control over a remote machine, they are extremely valuable for performing penetration testing. Auxiliary modules allow the framework to extend to a variety of purposes different than mere exploitation, such as DoS, scanning, fuzzing, sniffing or data gathering. The credential theft against the MySQL server of the target1 VM uses the scanner of the auxiliary module. More precisely it employs the mysql_login auxiliary module, a brute-force login tool for MySQL servers. To conduct the brute-force login some parameters need to be defined. The command 'show options' shows the name, current setting, whether or not the parameter is required and the description of all tunable parameters.

The 4.1 listing portrays the default configuration of the brute-force login extension. The only parameters of interest for the credential theft of the project are PASS_FILE, RHOSTS, STOP_ON_SUCCESS, USERNAME and VERBOSE. USER_FILE is not specified, since the knowledge of the username or lack of it cannot be a requirement to preserve the security of a system. Lines 14 to 20 of listing 4.2 contain the command to execute the attack with the specification of each of the parameters. Conducting the attack itself is quite simple, since it only requires 4 steps. Accessing the msfconsole; entering the module of interest, in this case the command needed is 'use auxiliary\scanner\mysql\mysql_login'; specify the parameters of interest; and lastly let the console run the attack.

RHOSTS holds the IP address of the target. The list of passwords assigned to PASS_FILE for the attack is a million passwords long and contains very diverse and frequently found passwords. Since the project recreates an insider attack, knowing the username is a plausible assumption. Possessing the username facilitates the attack, considerably reducing the time required and the possibility of triggering any security parameter, such as maximum connections allowed. USERNAME is set to the known username. Specifying these three parameters is the minimum requirement to conduct the brute-force login attack. The remaining settings are non-essential, but contribute to a more efficient attack. VERBOSE prints only the successful attempts, when it is set to false, as is the case. This reduces the attack time significantly, since the command line only prints out a couple of results instead of the over a million attempts. The STOP_ON_SUCCESS setting stops guessing as soon as one of the credentials is successful. To the purpose of the thesis guessing one password is enough, since it already provides full access to the MySQL server. The mysql_login module tries to connect to the specified host and user with all of the passwords in the PASS_FILE. The attack lasts until it either succeeds or runs out of passwords, in which case the credential stealing has failed. Something worth mentioning is the fact that RPORT automatically points to port 3306. The reason behind is that the module in use is the mysql_login and the default port for the classic MySQL protocol is number 3306. Though it could be possible to run a MySQL server on a different port, the module makes this assumption, since usually the port of the MySQL server is rarely changed.

Listing 4.1: Default mysql_login setting

```

Module options (auxiliary/scanner/mysql/mysql_login):

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   BLANK_PASSWORDS   true             no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                yes       How fast to bruteforce, from 0 to 5
   DB_ALL_CREDS      false            no        Try each user/password couple stored
   in current DB
   DB_ALL_PASS       false            no        Add all passwords in current
   DB to the list
   DB_ALL_USERS      false            no        Add all users in current DB
   to the list
   DB_SKIP_EXISTING  none             no        Skip existing credentials
   in current DB
   PASSWORD                           no        A specific password to
   authenticate with
   PASS_FILE                          no        File containing passwords,
   one per line
   Proxies                            no        A proxy chain of format
   type:host:port
   RHOSTS                             yes       The target host(s)
   RPORT             3306             yes       The target port (TCP)
   STOP_ON_SUCCESS   false            yes       Stop guessing when credential works
   THREADS           1                yes       Number concurrent threads
   (max 1 per host)
   USERNAME          root             no        A specific username to
   authenticate as
   USERPASS_FILE                      no        File containing users and passwords
   separated by space, one pair per line
   USER_AS_PASS      false            no        Try username as password
   for all users
   USER_FILE                          no        File containing usernames,
   one per line
   VERBOSE           true             yes       Whether to print output
   for all attempts
```

4.2.2 Python Scripting of Linux commands

To automate and accelerate the execution of the brute-force attack, the commands are scripted. The script takes over all the tasks: opening the tmux sessions, running the commands, accessing the msfconsole, among others. For the purpose of simulating the credential theft insider attack by a host embedded in normal traffic, the attack is divided into 3 main time-based stages, normal traffic (preAttack), attack (msfconsole and flush-hosts) and normal traffic (postAttack and tshark). This division allows for a clear distinction between both types of data flow, which should pop up in the exploratory host-based and network-based data analysis. Listing 4.2 holds the python function create_attack_list(). The name is not very accurate, since it creates the list of all the actions to simulate both the attack and the normal behavior of the host. The attack list contains five different actions in total. The preAttack, that resembles the normal behavior of the host before the brute-force attack. The msfconsole, which does the actual credential stealing. Flush-hosts as a precaution in case the MySQL server blocks the host because of too many connection errors. The postAttack, which performs the same action as the preAttack, but after the brute-forcing attack has taken place. Lastly, tshark converts the pcap dataset into a csv file to facilitate the exploratory analysis with python.

Listing 4.2: Attack code

```
def create_attack_list(self):
    """
    Creates the attack list.
    Checks for the attack type and adds attack command to the list.
    Same attack can be run multiple times behind each other
    and is specified in the iterations key.
    """
    self.attack_list = []
    for i in range(len(self.config_data["attack_set"])):
        if self.config_data["attack_set"][i]["type"] == "preAttack":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('mysql -u username -p password
                -h dst_ip -e \'' + self.random_commands(0) + '\'')
        if self.config_data["attack_set"][i]["type"] == "msfconsole":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('msfconsole -q -x \'use
                auxiliary/scanner/mysql/mysql_login;
                set RHOSTS dst_ip; set VERBOSE false;
                set USERNAME username; set PASS_FILE passwords.txt;
                set STOP_ON_SUCCESS true; run; exit\'')
        if self.config_data["attack_set"][i]["type"] == "flush-hosts":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
            self.attack_list.append('mysqladmin -u username -p password
            -h dst_ip flush-hosts')
        if self.config_data["attack_set"][i]["type"] == "postAttack":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('mysql -u username -p password
                -h dst_ip -e \'' + self.random_commands(0) + '\'')
        if self.config_data["attack_set"][i]["type"] == "tshark":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('tshark -r ' + self.log_path
                +'attack_kali1_' + self.overall_start_time +'.pcap -t ud
                -T fields -e _ws.col.Time -e ip.src -e ip.dst
                -e frame.len -e _ws.col.Protocol -E separator=,
                -E occurrence=f > ' + self.log_path + 'attack_kali1_'+
                self.overall_start_time +'.csv')

    self.total_attack_set_iterations = self.config_data["attack_set_iterations"]

def start_attack_handler(self):
    """
    Main attack handler.
    Traverses through the attack list and runs the attack commands.
    Checks if current attack has finished. If yes next attack command
    of the list will be started.
    """
    for _ in range(self.total_attack_set_iterations):

        self.attack_set_iteration_counter = 0
        while self.attack_set_iteration_counter < len(self.attack_list):
            if self.current_attack_finished:
                self.current_attack_start_time = datetime.datetime.now(
                datetime.timezone.utc)
                self.start_new_attack()
                self.set_attack_state_to_running()

            if self.check_attack_state():
                self.create_attack_list()
                self.set_attack_state_to_finished()
                self.add_current_attack_timestamps()
                self.string_start_time = datetime.datetime.strptime(
                self.overall_start_time, "%Y_%m_%d-%I_%M_%S_%p_UTC")
                self.current_timestamp = datetime.datetime.strptime(
                datetime.datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p_UTC"),
                "%Y_%m_%d-%I_%M_%S_%p_UTC")
                if (self.current_timestamp - self.string_start_time >
                datetime.timedelta(hours=1) and self.attack_set_iteration_counter
                ==0)
                or (self.current_timestamp -  self.string_start_time >
                datetime.timedelta(hours=3) and self.attack_set_iteration_counter
                ==3)
                or self.attack_set_iteration_counter in [1,2,4]:
                    self.attack_set_iteration_counter += 1

            time.sleep(1.0)

    self.stop_tmux()
```

The preAttack and the postAttack aim to simulate actions the legitimate user performs on the MySQL server. To do so, both actions connect to the MySQL server and run the function random_commands() 4.3. The function adds some randomness to the script and avoids repeating in each execution of the script the exact same steps and attack. The brute-force attack lasts around 40 minutes (see Figure 5.2). The pre and postAttack are designed to last more or less one hour each. The idea is to generate traffic over a similar amount of time. This should allow to better draw parallelisms and compare both events under the same conditions. The function random_commands() is very basic and executes different commands during the specified timespan. The function selects in every run a random number between 0 and 5. With an index value of 0 the fucnion executes a SELECT query. The function select() uses another random process to select the database, table and column to read the data from. Index 1 drops and creates a table from a database. Again the creation of the table follows a separate random process started with the function dropCreate(). Lastly for the remaining values insertions on different tables take place. The value of the index chooses the table that will suffer the new INSERT queries. The queries themselves, however, are chosen with the fuction insert(). The actions nor the syntax are very complicated, but the function random_commands() serves the purpose of generating a constant flow of traffic in the network to enable the comparison between different parameters. To see all the MySQL queries executed during these stages turn to Appendix C.

Listing 4.3: Normal Behavior

```
def random_commands(self, typeAttack):
    randomIndex = random.randint(0,5)
    if randomIndex == 0:
        return self.select()
    elif randomIndex == 1:
        return self.dropCreate(self.config_data["attack_set"][typeAttack]
        ["details"][randomIndex]["Command"])
    elif randomIndex in range(2,7):
        return self.insert(self.config_data["attack_set"][typeAttack]
        ["details"][randomIndex]["Command"])
```

The msfconsole action carries out the whole brute-force attack, following the steps explained at the beginning of the section. It connects to the msfconsole, it specifies the parameters of interest for the project in the auxiliary module, RHOSTS, VERBOSE, USERNAME, PASS_FILE and STOP_ON_SUCCESS. Finally it runs the attack and once it ends it exists the console.

Flush-hosts empties the host cache and unblocks any blocked hosts. In the script the command is a precaution to ensure the correct development of all the actions. If the host at some point fails too many times to connect, the MySQL server will block it, preventing the host from connecting at all. If the host cannot connect, the traffic will not hold much data of interest, since any connection will be rejected.

The kali1 VM records the host traffic with tcpdump. Tcpdump directly captures all network traffic packets using the libcap library and produces raw data stored in a pcap file. The pcap file contains a huge amount of information on each of the packets. For the preliminary exploratory data analysis of this project a lot of that information is superfluous. Listing 4.6 shows the raw data of a single packet captured and stored in the pcap file. To extract insight of the data flow the information is translated into visualizations through matplotlib. To work more easily with the traffic in the pcap file, tshark transforms the data into a csv file keeping only the fields of interest to the exploratory analysis. The specifications for the tshark command are in lines 31 to 36 of listing 4.2. The fields extracted are the source and destination ip, the timestamp of each packet, the packet size and the protocol in the highest layer. In the command in listing 4.2 the protocol and time fields are not directly imported, but have the suffix _ws.col. This suffix makes both fields more understandable to the user. It resolves the name of the protocol instead of loading it as a number and it translates the UNIX timestamp to UTC format.

The other function in listing 4.2 is start_attack_handler(). This is the main attack handler and it traverses through the previously defined attack list running the five explained actions. The function checks if the attack has finished (line 51). If the statement is true the next attack command of the list that corresponds to the self.attack_set_iteration_counter starts executing with self.start_new_attack()

(line 54). The function self.check_attack_state() reads the last line of the current_attack.log file to check whether or not the current action has finished. If the current action finishes the last line of the log file holds the string: "Checked: Attack finished!". This condition is enough to increase the self.attack_set_iteration_counter if the commands are either msfconsole, flush-hosts or tshark. If the actions are preAttack or postAttack a second condition exists in order to move on, namely the time condition. The preAttack lasts one hour. Until that timespan does not elpase the self.start_new_attack() function will keep executing random queries with self.random_commands. The situation repeats with postAttack.

Tmux is a terminal multiplexer that allows to start several "pseudo terminals" from a single terminal. This is very helpful, since the connection to the virtual machines of the testbed is remote. The connection may fail or be interrupted. Tmux ensures that the attack and further actions take place regardless of any interruption or failure. The script launches two tmux sessions during the attack 4.4. The sessions involve the attack and the gathering of network flow data, since the non-interruption of both processes is crucial. The first pane of listing 4.4 is responsible for the execution of the attack per se. Meanwhile the second pane records the traffic data through tcpdump and stores it at the end of the attack in a pcap file. The function start_attack_handler() kills the tmux sessions once all the actions on the attack lists finish (line 75 of listing 4.2).

Listing 4.4: Tmux sessions

```
def initialize_tmux_server(self):
        """
        Start the Tmux server and session.
        Opens two panes for tcpdump and attack commands.
        """
        self.tmux_server = libtmux.Server()
        self.tmux_session = self.tmux_server.new_session(
        session_name="kali1_attack",
        kill_session=True, attach=False)
        self.tmux_window = self.tmux_session.new_window(attach=True,
        window_name="kali1_attack")
        self.tmux_pane1 = self.tmux_window.attached_pane
        self.tmux_pane2 = self.tmux_window.split_window(vertical=False)
        self.tmux_window.select_layout('even-horizontal')
```

The details regarding all functions involved in the process are in Appendix C.

4.2.3 Python packages for Visualization

The visualization, preparation and transformation of the data use different python libraries. Only two are really worth mentioning, matplotlib for visualization and pandas for data treatment. Matplotlib [mat23] is a powerful python library to create static, animated, and interactive visualizations. The project uses this python module to do different representations of the traffic at host and network level. The loading of the data into python uses pandas [pan23]. A powerful, easy and flexible open source tool to perform data manipulation and analysis.

At host level after loading the data, the filtering process keeps only the data flow between the attacker VM and the victim. The reason is to solely focus on the parameters of the brute-force login and the normal behavior of the user and have the clearest picture possible. The visualizations focus on the three fields extracted with tshark besides the ip addresses. The visualizations illustrate the behavior over time of the traffic, the packet sizes and their frequency, and the frequency of each protocol.

At network level the visualizations combine average behaviors of all the connections with the traffic generated only between the attacker and target VM. The plots focus on the parameters packet frequency, number of bytes sent and average packet size. The resulting visualizations are discussed in section 5.

4.2.4 Weighted Stochastic Block Models

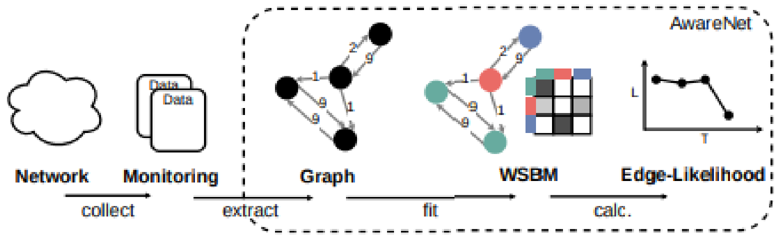


Figure 4.2: AwareNet steps
[SKK22]

For the application of Weighted Stochastic Block Models (WSBM) this thesis uses AwareNet [SKK22]. AwareNet implements probability theory to generate a probabilistic model for network flows and behaviors. Figure 4.2 portrays the architecture of AwareNet. The attack traffic collected from the central monitoring nurtures the model to construct different graphs. As a starter AwareNet takes the first 10 minutes of the monitored data to develop a graph of the network flow. In the graph IP addresses represent nodes and data flows the edges of the communications over that first 10 minute slice. The edges also have some associated statistics, such as amount of packets sent. Consecutively, AwareNet fits the parameters of a WSBM with the library GraphTool [gra23]. An efficient Python module to manipulate and statistically analyse graphs. This fitting process results in a partition z of all nodes in the graph into k groups. Figure 4.2 illustrates the architecture of AwareNet. The partition the model uses pairs nodes with similar communication behaviors into the same group. The grouping can already reveal some underlying network-inherent structures that are imperceptible at first sight. The matrix in Figure 4.2 portrays in a grey-scale the statistics about edge existence and weight distribution at a group-to-group level. These statistics are employed to calculate edge likelihoods. The following formula calculates the log-likelihood of an edge e_{ij} between node i and node j with weight w_{ij} of the fitted WSBM model θ:

$$log\mathcal{L}(e_{ij}|\theta) = logPois_{\lambda_{z_i,z_j}}(1) + log\mathcal{N}_{\mu_{z_i,z_j},\sigma_{z_i,z_j}}(w_{ij}) \tag{4.1}$$

The first summand of the equation 4.1 assumes a Poisson distribution for edge existence. λ represents the ratio of observed edges from the group of node i,z_i to the group of node j,z_j and the amount of possible edges between both groups, $|z_i|\cdot|z_j|$. The second summand assumes the edge weights to follow a normal distribution and represents through μ and σ for edges from group z_i to z_j the related maximum likelihood estimates of all their weights. Since the edge existence and weight are also assumed to be independent, the total edge log-likelihoods for the fitted model result from adding the log-likelihoods for existence and weight. The memory complexity for AwareNet is of $\mathcal{O}(k^2 + n)$. The computational complexity to calculate edge log-likelihoods on the other hand is linearly dependent on the amount of edges in upcoming observations.

The rationale behind using a 10 minute slice to generate the initial graph is to calculate the values for what will be considered the "normal" and expected behavior of the network traffic. After this first fitting process AwareNet processes the complete data flow and assigns nodes to groups and calculates edge log-likelihoods. The likelihoods that greatly deviate from the initial 10 minute slice values need to be deeply looked into, to identify the reason for portraying an odd and unexpected behavior.

4.3 Data

The function random_commands() [4.3] emulates the normal behavior of the legitimate user of the MySQL server. The function uses two different repositories to implement the different actions of selecting, inserting, dropping and creating tables. Both repositories contain simple multi-purpose data sets with various different tables. Since the information transmitted during the normal behavior is irrelevant, the repositories selected are star-wars-data and hogwarts-sql. The upside of using these repositories is the abundance of rows of data and the different query sizes.The star wars repository [Rol18] has very large queries, while the hogwarts [Sud16] queries are rather short. The packet length of the traffic is therefore heterogeneous, displaying a non-uniform behavior over the MySQL server. The star wars repository has two tables, planet and people. The hogwarts database has

6 tables, houses, parents, students, teachers, classes and class_rosters. Both repositories have a vast amount of entries for the tables, offering wide and verstatile possibilities for the random_command() function.

The data used to mirror the normal behavior with the MySQL server is the only one that stems from external sources. The rest of the data employed in the project is generated internally in the testbed environment. Listing 4.5 and 4.6 show an extraction of the csv file and of the pcap file respectively at host-level. The first packet in listing 4.5, index 8, is the transformation and simplification of the raw packet presented in the pcap file 4.6. The data in the csv file leads to the visualizations of the host level exploratory data analysis. More details can be extracted from the pcap file, but the five fields selected are enough key indicators to detect a brute-force login attack, as demonstrated in section Results.

Listing 4.5: Extraction of csv file

```
   Time                        IP_src       IP_dest      Size  Protocol
---------------------------------------------------------------------------
8  2023-05-30 11:10:30.684731 ip_attacker  ip_target    74    TCP
9  2023-05-30 11:10:30.685345 ip_target    ip_attacker  74    TCP
10 2023-05-30 11:10:30.685370 ip_attacker  ip_target    66    TCP
11 2023-05-30 11:10:30.686563 ip_target    ip_attacker  161   MySQL
12 2023-05-30 11:10:30.686581 ip_attacker  ip_target    66    TCP
13 2023-05-30 11:10:30.686756 ip_attacker  ip_target    294   MySQL
14 2023-05-30 11:10:30.686977 ip_target    ip_attacker  66    TCP
15 2023-05-30 11:10:30.687122 ip_target    ip_attacker  114   MySQL
16 2023-05-30 11:10:30.687128 ip_attacker  ip_target    66    TCP
17 2023-05-30 11:10:30.687163 ip_attacker  ip_target    90    MySQL
```

Listing 4.6: Example of raw packet information in pcap file

```
Packet (Length: 74)
Layer ETH:
 Destination: dst_eth
 Address: add_eth
 .... ..1. .... .... .... .... = LG bit: Locally administered address
 (this is NOT the factory default)
 .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
 Source: src_eth
 .... ..1. .... .... .... .... = LG bit: Locally administered address
 (this is NOT the factory default)
 .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
 Type: IPv4 (0x0800)
 Address: add_eth
Layer IP:
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 0000 00.. = Differentiated Services Codepoint: Default (0)
 .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
 Total Length: 60
 Identification: 0xa0f7 (41207)
 010. .... = Flags: 0x2, Don't fragment
 0... .... = Reserved bit: Not set
 .1.. .... = Don't fragment: Set
 ..0. .... = More fragments: Not set
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 64
 Protocol: TCP (6)
 Header Checksum: 0x5966 [validation disabled]
 Header checksum status: Unverified
 Source Address: ip_attacker
 Destination Address: ip_target
Layer TCP:
 Source Port: src_port
 Destination Port: 3306
 Stream index: 0
 Conversation completeness: Incomplete (0)
 TCP Segment Len: 0
 Sequence Number: 0    (relative sequence number)
 Sequence Number (raw): 389017798
 Next Sequence Number: 1    (relative sequence number)
 Acknowledgment Number: 0
 Acknowledgment number (raw): 0
 1010 .... = Header Length: 40 bytes (10)
 Flags: 0x002 (SYN)
 000. .... .... = Reserved: Not set
 ...0 .... .... = Accurate ECN: Not set
 .... 0... .... = Congestion Window Reduced: Not set
 .... .0.. .... = ECN-Echo: Not set
 .... ..0. .... = Urgent: Not set
 .... ...0 .... = Acknowledgment: Not set
 .... .... 0... = Push: Not set
 .... .... .0.. = Reset: Not set
 .... .... ..1. = Syn: Set
 Expert Info (Chat/Sequence): Connection establish request (SYN): server port 3306
 Connection establish request (SYN): server port 3306
 Severity level: Chat
 Group: Sequence
 .... .... ...0 = Fin: Not set
 TCP Flags:           S
 Window: 64240
 Calculated window size: 64240
 Checksum: 0x408d [unverified]
 Checksum Status: Unverified
 Urgent Pointer: 0
 Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps,
 No-Operation (NOP), Window scale
 TCP Option - Maximum segment size: 1460 bytes
 Kind: Maximum Segment Size (2)
 Length: 4
 MSS Value: 1460
 TCP Option - SACK permitted
 TCP Option - Timestamps
 Timestamp value: 664962153: TSval 664962153, TSecr 0
 Timestamp echo reply: 0
 TCP Option - No-Operation (NOP)
 TCP Option - Window scale: 7 (multiply by 128)
 Shift count: 7
 Multiplier: 128
 Timestamps
 Time since first frame in this TCP stream: 0.000000000 seconds
 Time since previous frame in this TCP stream: 0.000000000 seconds
 Kind: SACK Permitted (4)
 Kind: Time Stamp Option (8)
 Kind: No-Operation (1)
 Kind: Window Scale (3)
 Length: 2
```

```
Length: 10
Length: 3
```

The central monitoring captures all traffic flowing through the campus network. The ctrl VM uses a docker container to save the traffic into csv files. The csv files contain the information about all the packets detected between two end-to-end devices in the network during one minute The csv file saves the Unix timestamp as name of the file. Listing 4.7 holds an extraction of one of the csv files with all relevant information for the project. A total of 8 fields describe the traffic flow. The index, which indicates the total amount of different communications during the time interval; source IP; destination IP; direction; the timestamp of the captured csv file in UNIX format; the time_interval, which equals the duration of the gathering process and should always equal 60 seconds; the amount of packets detected; and the number of bytes counted. The docker container encrypts the raw source and destination IP out of security reasons. The kali1 IP corresponds to 037df6969ca851b7c627ff07f47b90d3 and the target1 to 4f895c4848ee12cc2f3aa0b12bd3f9c9 for example in this pseudomization.

Listing 4.7: Extraction of central monitoring data

```
,src,dst,direction,time,time_interval,packets,nb_bytes
0,3dcfdcdfe6f9b59467faaccf6242499c,35fb3b62c33d1388ff1659a876c2ed8c,MAIN,
1686124800,60,677864,1022547400
1,c04c5dc8db5f14ab279bcbc9463b93d6,35fb3b62c33d1388ff1659a876c2ed8c,MAIN,
1686124800,60,383776,547959184
2,35fb3b62c33d1388ff1659a876c2ed8c,c04c5dc8db5f14ab279bcbc9463b93d6,MAIN,
1686124800,60,121282,164282688
3,35fb3b62c33d1388ff1659a876c2ed8c,3dcfdcdfe6f9b59467faaccf6242499c,MAIN,
1686124800,60,62874,7811912
4,38d2bfaee1ec2497d2b8a7d12a16dfba,35fb3b62c33d1388ff1659a876c2ed8c,MAIN,
1686124800,60,34872,50273336
5,75633afcc4b8c3b28fac99aae683c7c3,35fb3b62c33d1388ff1659a876c2ed8c,MAIN,
1686124800,60,32440,46326552
```

Chapter 5

Results

The testbed environment hosts two kinds of behavior, brute-force attack and regular work on a MySQL server. The thesis analyses the behaviors with three different approaches to discover features and patterns to target during the future monitoring of the network. First, a host-level exploratory data analysis aims to reveal what protocols, packet sizes and packet frequencies appear on the host traffic and how they vary during a brute-force login attack. At network level two different procedures scrutinize the data flow. A network-level exploratory analysis investigates the amount of packets and bytes sent, as well as average packet sizes. Stochastic Block Models try to discover groups or clusters of connections that reveal the brute-force attack.

5.1 Host-level exploratory data analysis

The host based analysis uses the data collected through tcpdump by the attacker VM. The purpose is to identify odd behaviors during the brute-force attack and differences in the data flow between the normal actions of the legit user and the attacker. The odd behaviors can manifest as sudden spikes and drops or as repetitive behaviors, for example. The parameters visualized on the plots are packet frequency, protocol sizes, protocol frequencies and their respective behaviors over time.

Tcpdump captures data over a timespan of 190 minutes. Graphs 5.1, 5.2, 5.6, 5.10 and 5.7 illustrate the behavior over the complete time interval. These representations show the contrast between the two traffic types. Figures 5.3, 5.11 and 5.12 dissect the time behavior and packet size graphs into normal usage and attack traffic. The separation of both scenarios allows to more precisely analyze the patterns and characteristics of both behaviors and determine which features to target during the monitoring of the network.

The CDF of the average number of packets sent per minute over the whole interval appears in Figure 5.1. The CDF of the traffic flow from the attacker to the victim, blue line, and the traffic flow from the victim to the attacker, orange line, have a similar shape. The difference between both cumulative distribution functions is the range of the average packets per minute. The packets/minute stemming from the target VM top out at 125.000, whereas the ones sourcing from the attacker VM reach a maximum frequency of 175.000. The difference in packet density resembles that the whole process presents an asymmetric network traffic between the hosts.

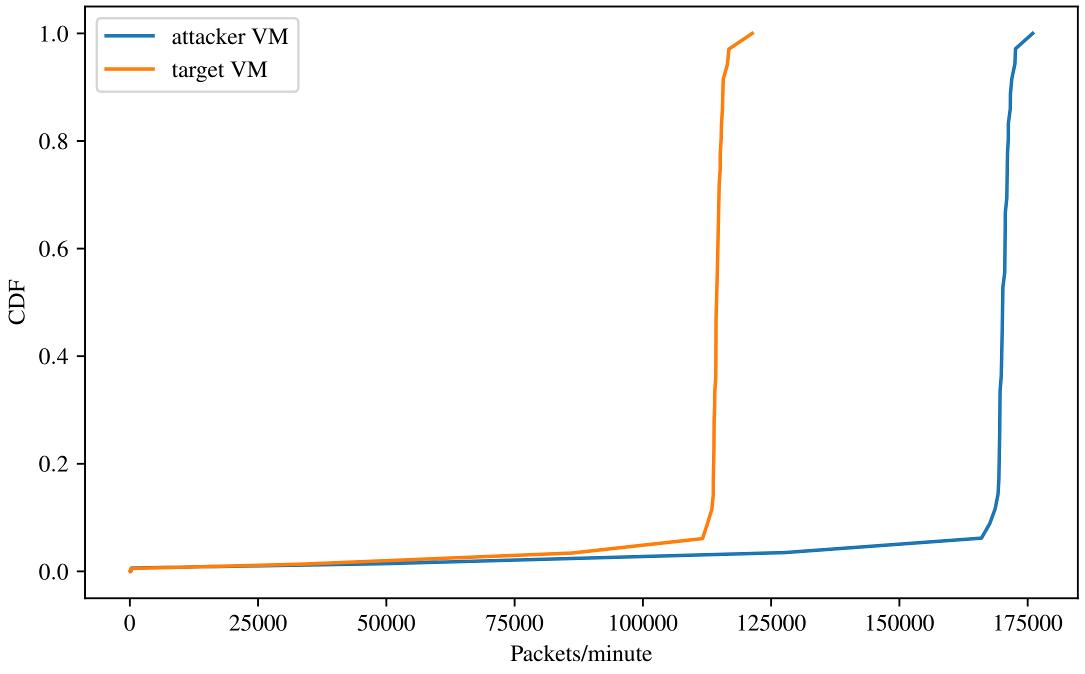


Figure 5.1: CDF packets/minute

Figure 5.2 portrays the clear distinction between the three phases of the scenario under study. The first hour features the legitimate user working on the MySQL server. Immediately afterwards the credential stealing begins and takes almost 40 minutes to guess the correct credential for the known user. Once the auxiliary module cracks the password, for the remaining 80 minutes the simulation of the legit user repeats. The attack works with a list of 1.000.002 different passwords. The credential stealing stops as soon as one of the passwords succeeds. Given this configuration and the position of the correct password in the list of passwords, the brute-force login tries only about half of the passwords in the file.

The mysql_login module of Metasploit tries to crack the password as quickly as possible. The constant testing of passwords generates a very high network flow in the host during the brute-forcing. The network traffic of the normal usage on the other side is very low. Figure 5.3 illustrates the data flow prior and posterior to the attack, without featuring the credential stealing. The space between the dashed lines corresponds to the omitted brute-force traffic. Despite the randomness of the MySQL queries the data flow has a very similar and fairly constant behavior.

To compare the absolute difference in packet frequency per minute, histograms 5.4 and 5.5 represent the packet flow of both hosts during the attack and during the normal behavior. During the normal behavior the traffic flow concentrates on a pretty narrow range between 350 and 500 packets per minute. On the other side, the packet rate of the attack ranges between 282250 and 287750 packets per minute, which is a significantly wider interval.

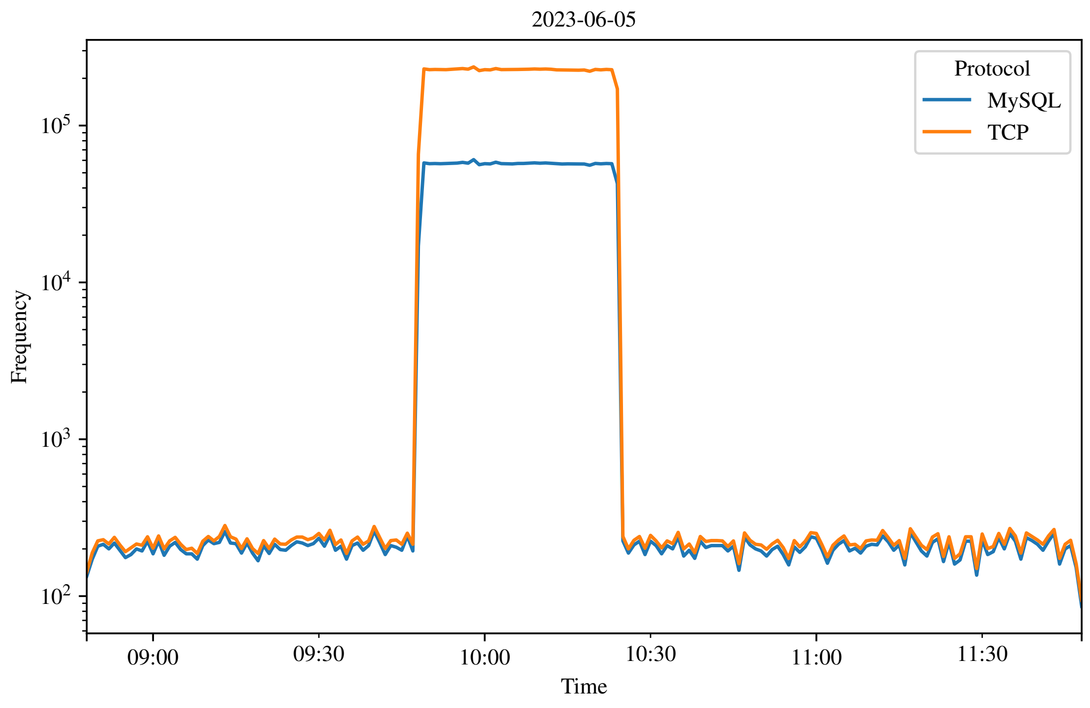


Figure 5.2: Behavior over time of protocols

Both scenarios share similarities and differences in their behaviors. The most noticeable difference is the packet frequency per time unit of the attack and the normal usage. A possible explanation for the divergence in packet rate could be the difference in query sizes of both scenarios. During the normal behavior the user is actually working with the databases, not only trying to access them. The queries range from selecting information, to entering new information or creating and dropping tables. Any of these queries will require more execution time than simply trying out a password. Taking up more execution time means lower packet processing capacity per time unit, leading to a drop in packet density per minute. Further, the brute-force attack tries to crack the password as quickly as possible. The quicker and the higher the amount of packets are sent, the earlier the attack succeeds. Another remarkable difference is the range of packets per minute. The normal behavior has a fairly constant traffic flow. The brute-force attack however, can vary its traffic flow in over 5.000 per minute. But despite the differing density of packet flows in the network, the TCP and MySQL protocol present a parallel behavior throughout the whole process. Both protocols spike and collapse at the same timestamps.

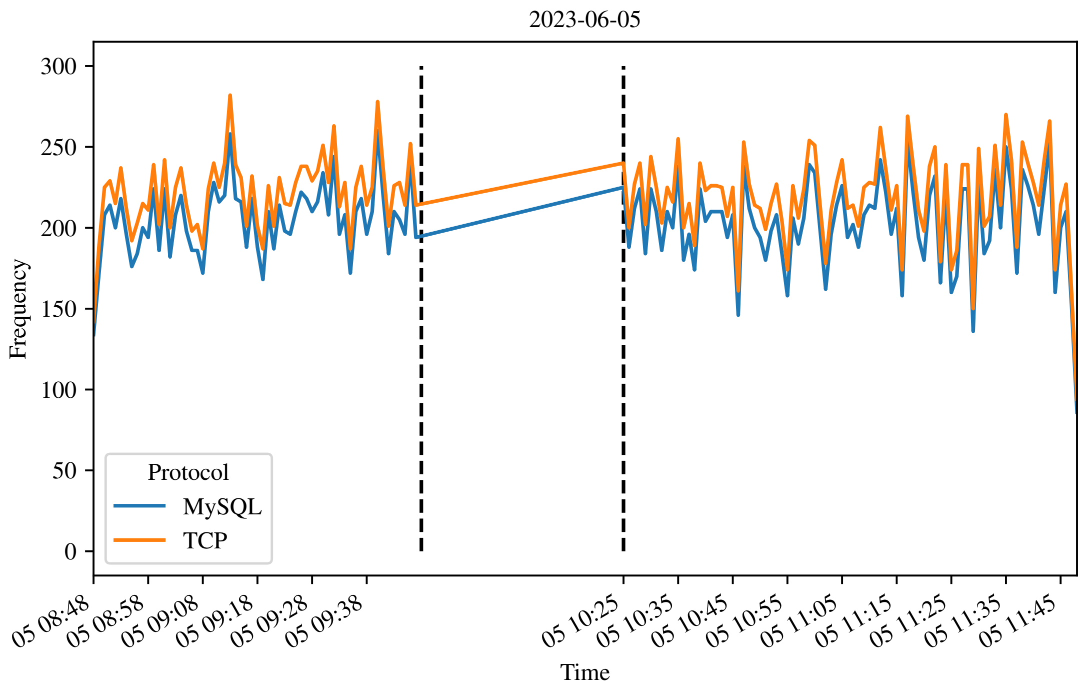


Figure 5.3: Behavior over time of protocols during normal behavior

Another distinction is the the gap between the packet frequency of both protocols during both behaviors. The attack sends considerably more TCP packets than MySQL ones, whereas the normal usage sends a very similar amount of packets of both protocols. As Figure 5.6 illustrates, the amount of packets of the MySQL protocol reaches 2.065.222 in contrast to the TCP protocol, which quadruples the quantity with an amount of 8.203.949 packets. The diagram shows the combined network traffic of the attack and the normal behavior.

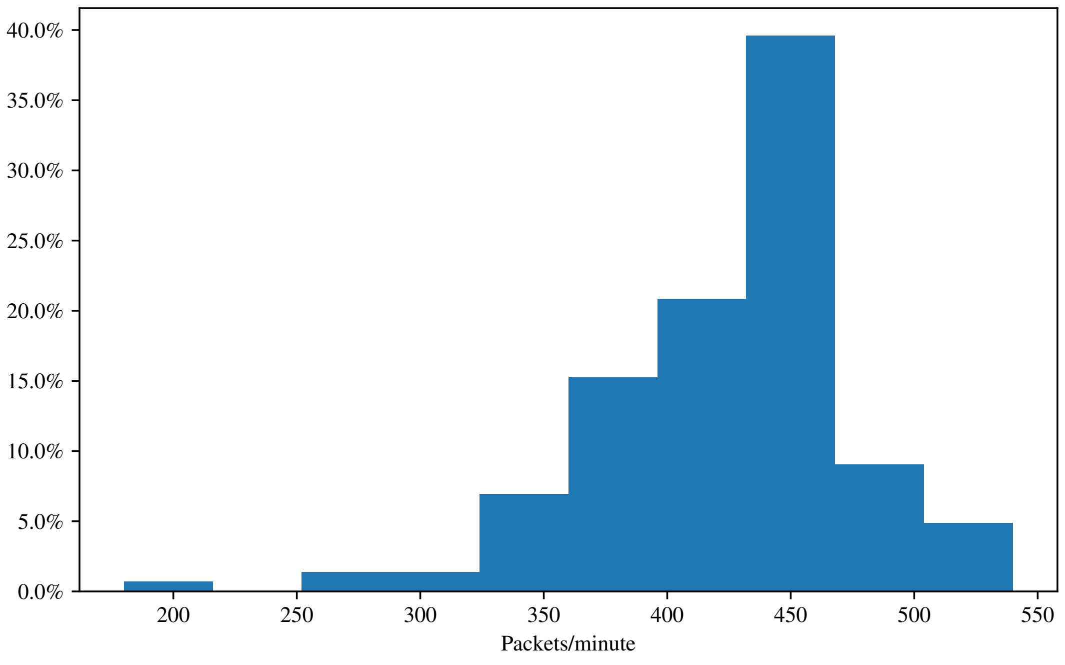


Figure 5.4: Packets per minute during normal behavior

Figure 5.7 offers a narrower picture of the protocols depicting the behavior of both protocols with respect to their source IP over time. During the normal behavior the MySQL traffic stemming from the target VM seemingly disappears. There does exist packet flow of the MySQL protocol from the target to the attacker. The problem is that the packet frequency is so similar, that both MySQL flows overlap, hiding the MySQL flow starting from the victim. One of the most interesting aspects of this graph is the change in protocol behavior from normal to attack behavior. During the normal behavior both MySQL threads have an equal distribution. Meanwhile, the TCP packets starting from the target VM present a lower frequency than the TCP from the attacker. During the attack however, the plot experiences an abrupt change. The four features keep presenting a parallel behavior, but the packet frequencies experience considerable changes. The TCP packets of both actors increase by a factor around 10^3, keeping a similar ratio between their respective packet frequencies. The MySQL protocol of the different connections on the other side no longer overlaps. The target is responsible for a considerably higher flow of MySQL packets.

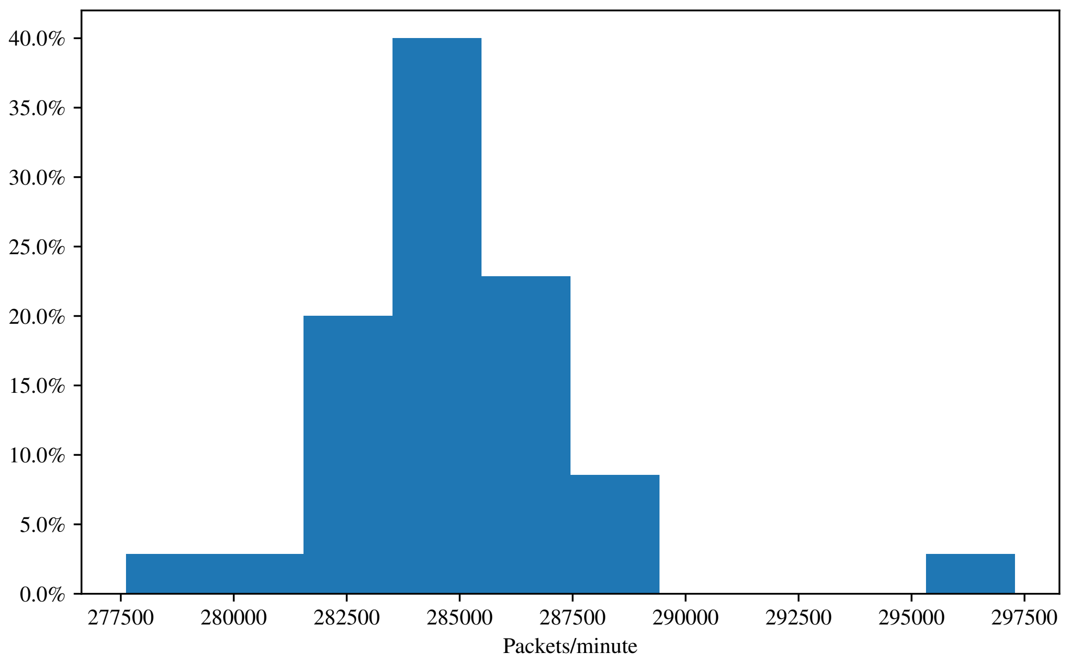


Figure 5.5: Packets per minute during brute-force attack

The behavior of the MySQL protocol resembles the scenario illustrated in figures 5.4 and 5.5 perfectly. During the normal behavior the MySQL packet flow overlaps corresponding to the narrow margin the histogram exposes. During the attack however the traffic density varies significantly. This behavior seems almost counterintuitive. As if the behavior during normal usage and the attack had been exchanged. It would seem logical that queries handling different amount of data, executing different actions and working on different databases would generate varying frequency responses. Whereas the brute-force attack, that sends packets only containing a username and a password, would cause immediate responses. Hence, what can be inferred is that the normal behavior generates a more constant flow of packets for both protocols and a symmetric MySQL traffic flow. Whereas the attack showcases a completely asymmetric traffic with considerable fluctuations in its packet density flow.

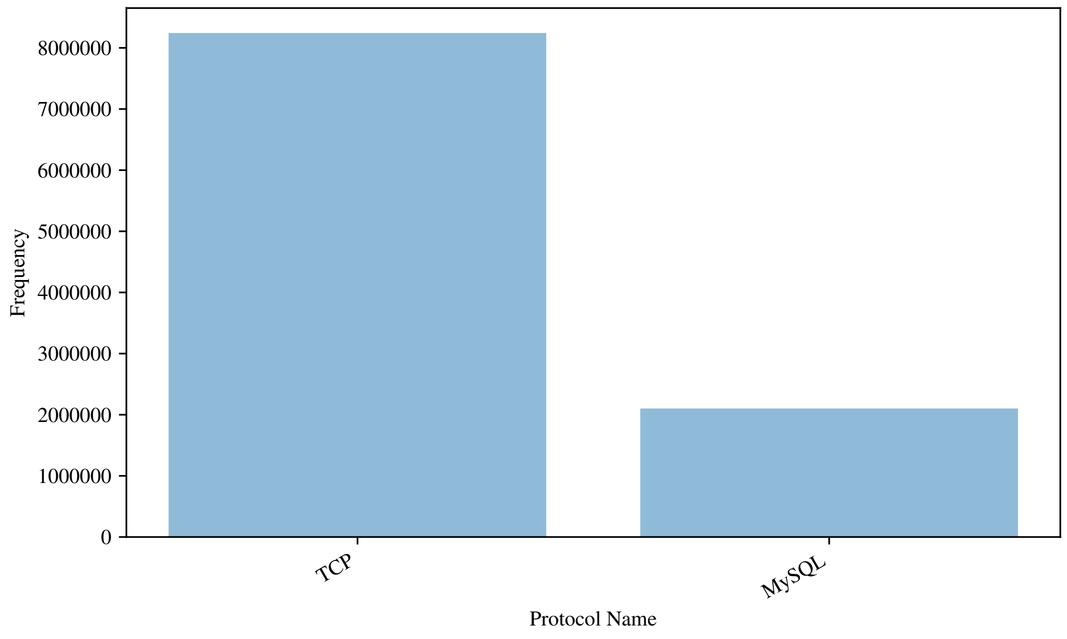


Figure 5.6: Packet frequency of protocols

The only protocols that intervene during the whole procedure are the TCP and the MySQL protocol. The Transmission Control Protocol (TCP) is a connection-oriented transport layer protocol. It establishes the connection prior to the communication allowing packet transmission between computing devices within a network. The MySQL protocol connects MySQL Clients with a MySQL Server and is stateful. MySQL runs on top of the TCP protocol and stays alive until the connection terminates. The analysis works with the highest layer protocol to gain deeper insight into all the protocols involved in the host traffic. Not doing so would reduce the detection possibilities for the attack. The TCP protocol appears constantly on the network traffic and is the basis for the majority of connections. Recognising patterns or odd behaviors in the MySQL protocol is a very useful discovery. The MySQL protocol is very specific for MySQL server connections and less frequent in data flows, making it easier to detect there odd behaviors.

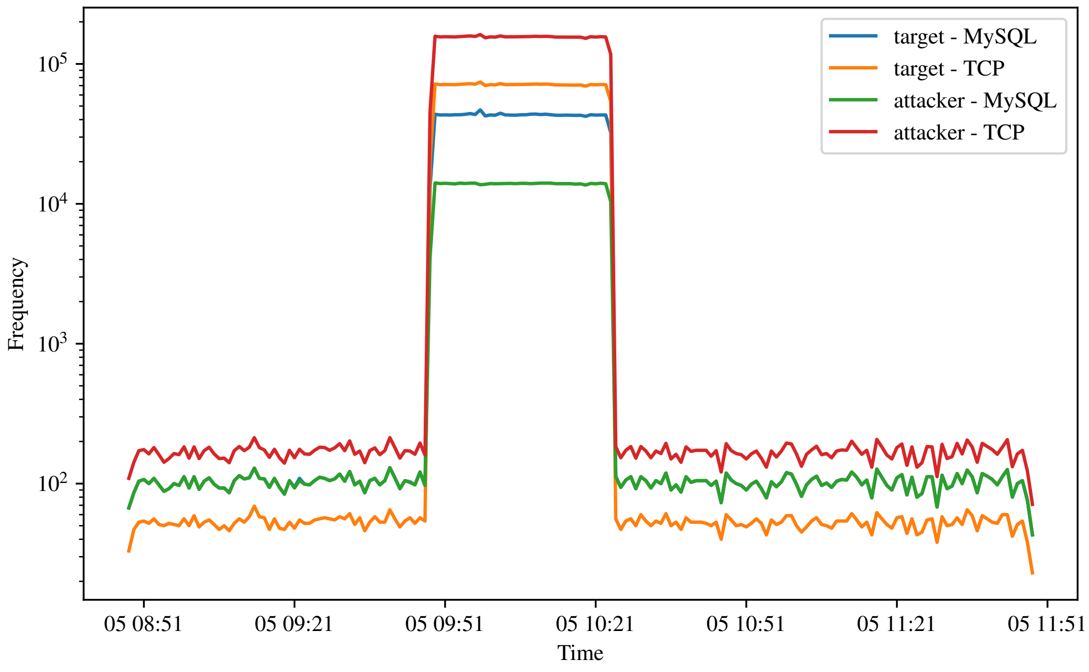


Figure 5.7: Behavior over time of protocols differentiated by source IP

Protocol frequency is not the only feature that experiences changes during the brute-force attack. The packet sizes flowing through the network also present changes in their behavior. As opposed to packet frequency, the range in packet size during the normal behavior is much wider than during the attack. Figures 5.8 and 5.9 provide an overall picture of the most frequent packet size ranges in the traffic. Almost the complete traffic flow during the normal behavior concentrates between 60 and 110 bytes 5.8. The range of packet sizes though reaches to over 3000 bytes. These cases are left out, since they are outliers. But the range of common packet size values goes from 50 to 600 bytes. Figure 5.9 shows the distribution of the attack packets. 80% of the packets concentrate between 65 and 80 bytes. The remaining ones have larger packet sizes around 160 bytes. But the scope of values is of 120 bytes as opposed to the 550 byte range of the normal behavior. The following graphs narrow down on the specific packet sizes of the network traffic.

Figure 5.10 provides a more exact representation of the packet size values of the whole attack, but differentiating by protocol. Diagrams 5.11 and 5.12 break the distribution down into both scenarios. The protocols present different distributions in their packet sizes. Neither of the protocols though works with big packet sizes, since the maximum size of a TCP packet is 65.535 bytes.

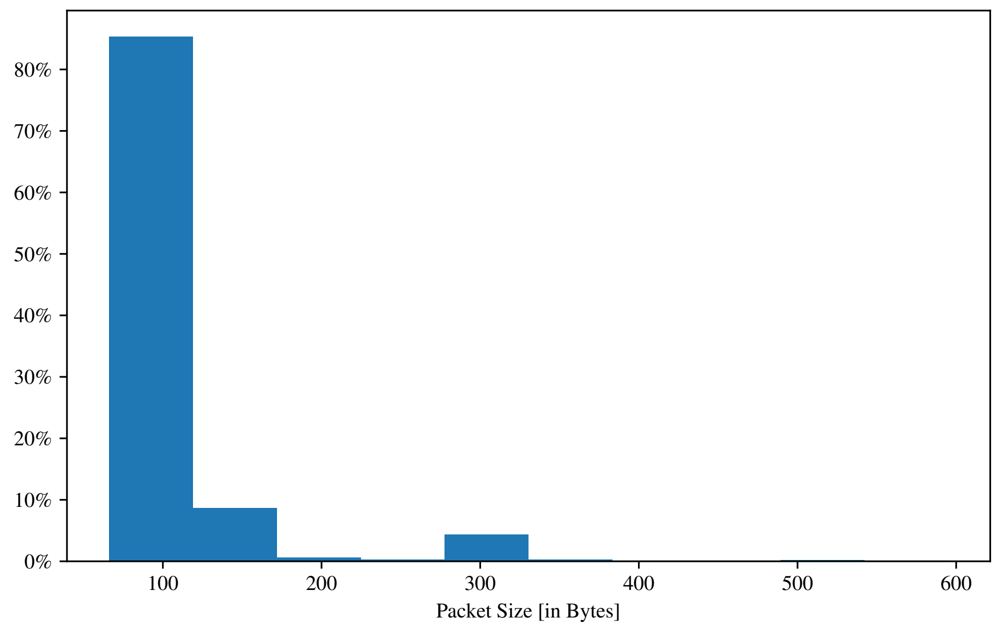


Figure 5.8: Packet sizes during normal behavior

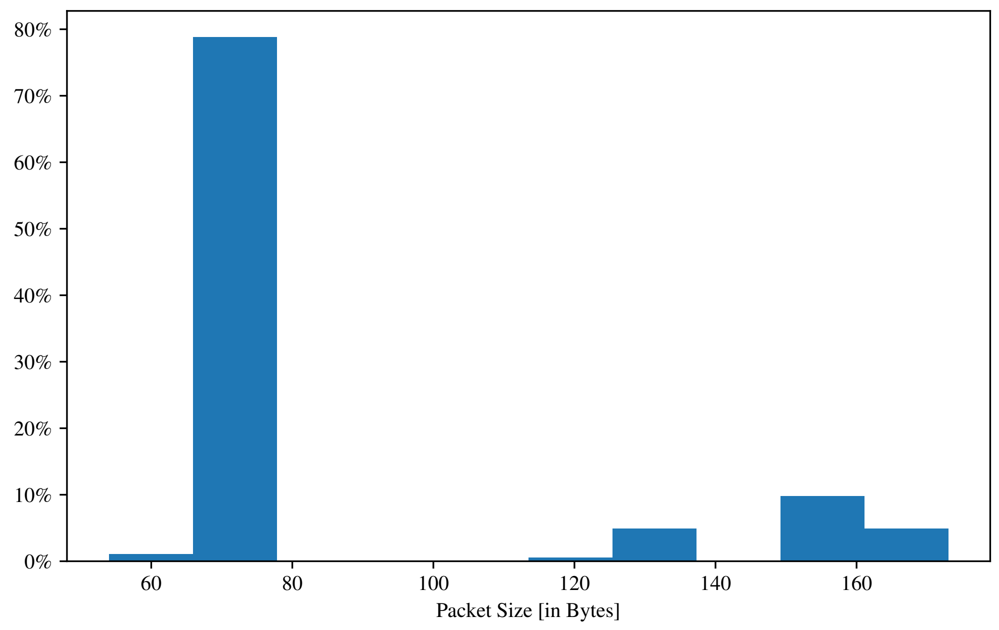


Figure 5.9: Packet sizes during brute-force attack

3/4 of the TCP packets have a length of 66 bytes and close to the remaining quarter are 74 bytes long. The MySQL protocol packets present slightly bigger lengths with around two times more bytes than the TCP packets. The protocol distributes the bytes reasonably even among 128, 161 and 173 bytes. There are more than 24 packet sizes in the host traffic, but the graph only depicts the most frequent ones. Despite the selection of the 24 most frequent packet sizes, some are so insignificant in comparison to 66 and 74 bytes, that their frequency is imperceptible in the graph.

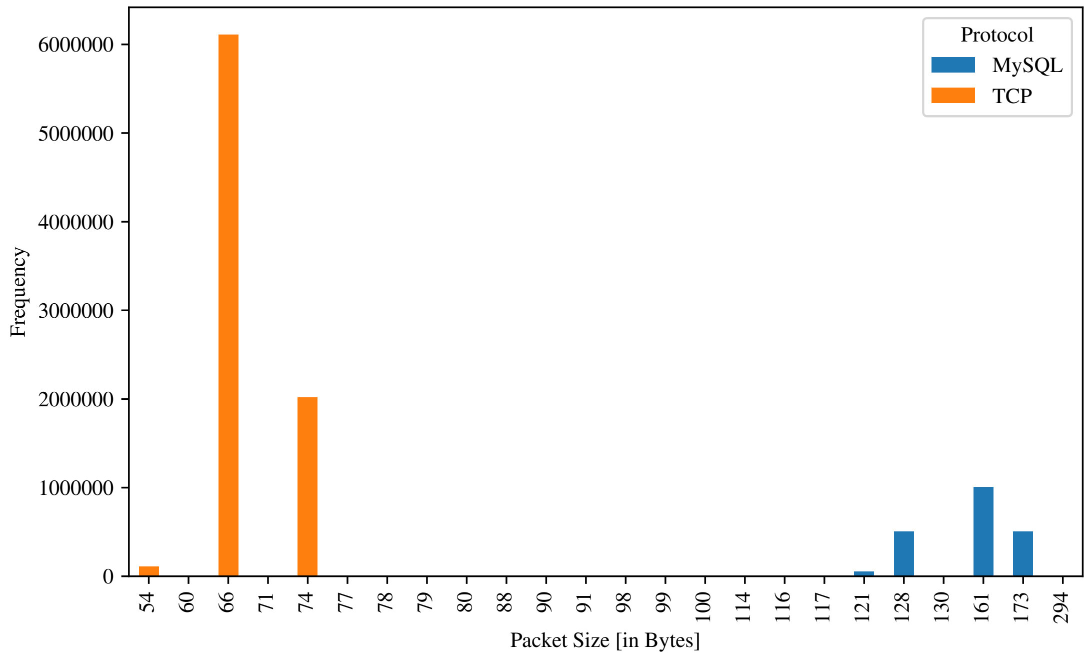


Figure 5.10: Packet sizes of protocols

As seen before in the time behavior of the normal usage, the quantity of MySQL and TCP protocols is very similar. The TCP protocol concentrates the majority of its traffic in 66 bytes long packets and the rest in 74 bytes long packets. MySQL distributes the packet sizes fairly even among all the packet sizes. The most frequent packet size is 90 bytes. But there is no length that predominates over the rest. Again, the MySQL protocol works with marginally bigger packet sizes, though in general terms the sizes are rather small.

During the attack the packet sizes of the TCP protocol are very similar to the ones of the normal usage, only with a much higher frequency. 66 bytes packets clearly dominate the network traffic when communicating with the MySQL server. The MySQL protocol uses in this case somewhat bigger packet sizes, but as is the case with the TCP protocol, the distribution of the packets between the different sizes is very similar. In this scenario 161 bytes predominates, and 128 and 173 bytes split evenly the remaining half. On the overall graph 5.10 with both types of traffic, the packet sizes of the normal usage are negligible. The low frequency of the normal behavior makes its impact on plot 5.10 imperceptible.

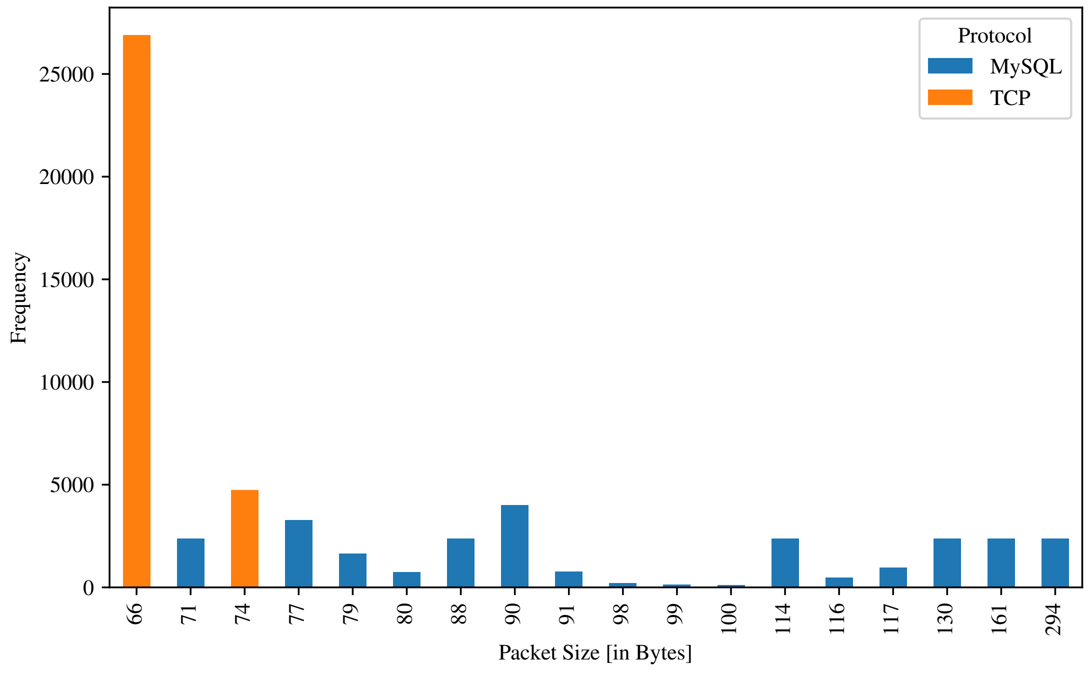


Figure 5.11: Packet sizes of protocols during normal behavior

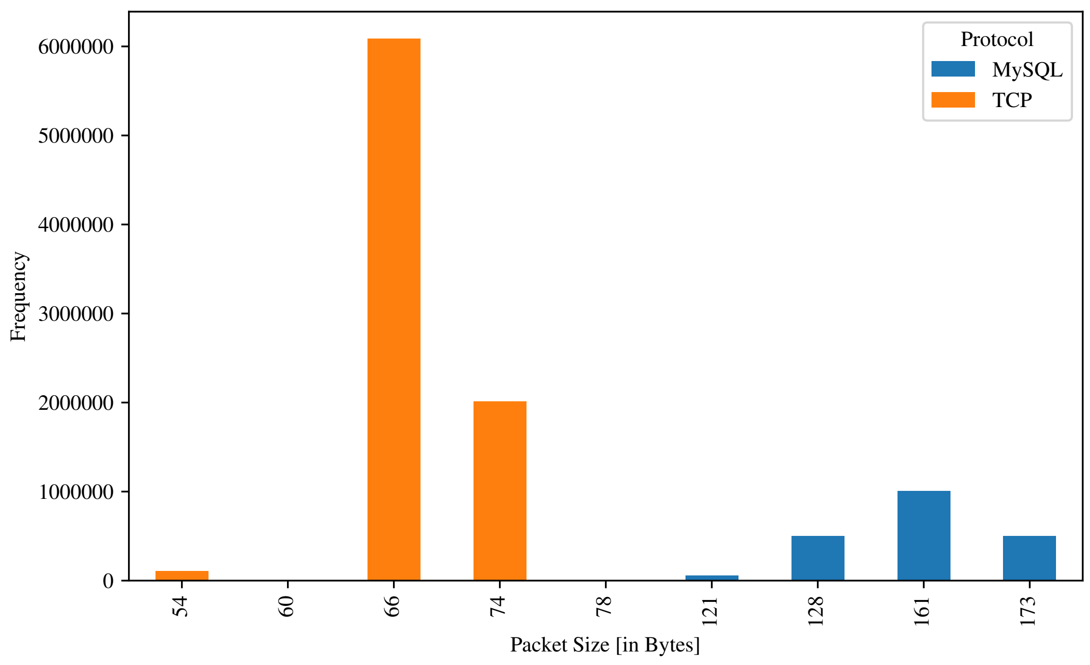


Figure 5.12: Packet sizes of protocols during attack

5.2 Network-level analysis

The network level analysis uses the data collected from the central monitoring. The control VM uses a docker container to anonymize the source and destination IP addresses from the network traffic of the central monitoring and stores the data flow. The exploratory data analysis studies different visualizations of the data flow featuring different parameters and tries to extract valuable patterns. The Stochastic Block Model uses some initial traffic to establish the values that should correspond to the expected behavior of the network. Afterwards, the model groups the traffic into different communities and analyzes the relations within and between said groups.

5.2.1 Exploratory data analysis

The exploratory analysis develops different visualizations of the traffic network based on packet frequency, number of bytes sent and average packet sizes. The visualizations throughout the section go from broader to more specific graphs.

The traffic flow of the central monitoring contains information about hundreds of connections and thousands of packets per minute. For that reason many visualizations employ more generic approaches working with averages. The analysis of the average behaviors leads to the development of more specific and revealing plots.

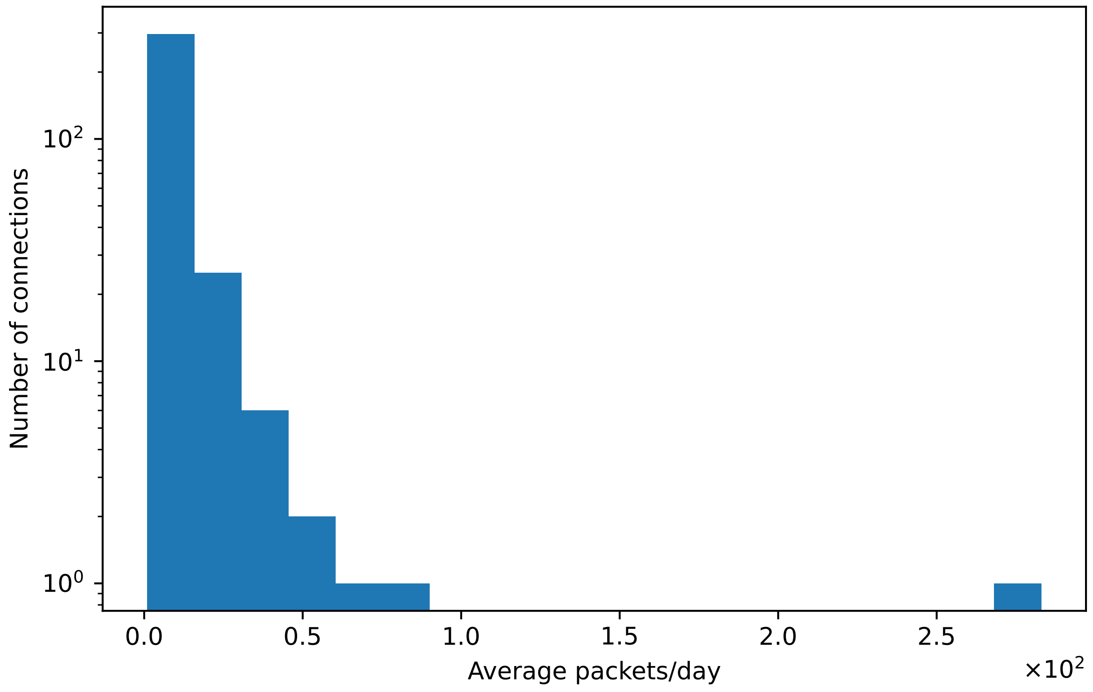


Figure 5.13: Amount of connections with same average packets per day

The histogram above 5.13 portrays the average packets a connection sends per minute during a day. The curve of the bars follows more or less a negative exponential distribution. The majority of connections send on average between 10 to 50 packets per minute during the whole day. Hence, the average sending rate of the majority of connections is of one packet every 2 seconds. The results of the histogram represent a normal and expected traffic flow in the network, with no remarkable odd behavior.

Figure 5.14 changes the perspective and displays the average amount of packets per minute over a whole day with regards to time. The dark blue line represents the average and the light blue region the standard deviation. It is remarkable to observe higher packet frequencies at night rather than over the day. Throughout the day the traffic flow is not very consistent and experiences constant fluctuations. The graph displays the average packets of all hosts transmitting at the different timestamps during the 3rd of June. However, it is difficult to recognise any peculiarity from such a general view.

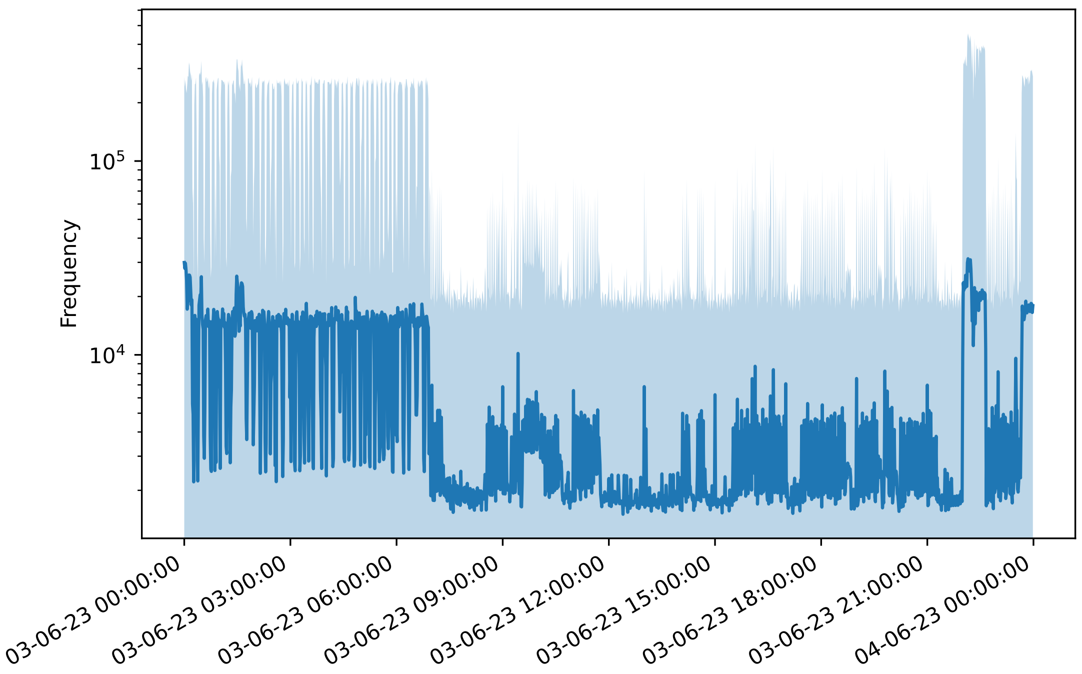


Figure 5.14: Average packets per minute over a day

Figure 5.15 zooms on the time interval of the brute-force login attack on the 3rd of June. The graph displays the time of the attack and fifteen minutes of normal usage of the MySQL server before and after the attack. This allows to compare the different impacts of both behaviors. The dark blue line still resembles the average number of packets sent on each minute through the network and the light blue region its respective standard deviation. The orange line represents the bidirectional traffic flow between the attacker and the victim. During the normal usage the amount of packets sent per minute is considerably below average and the distribution is non-uniform. On the contrary, all frequencies related to packets in the brute-force traffic are outliers. The quantity of packets sent per minute is very constant in contrast to the normal behavior.

Given the homogeneous behavior of the outliers of the attack traffic, studying the outliers on the network could provide insightful patterns. For that purpose figure 5.16 represents all the outliers detected during the same period of time. A green line with constant spikes and drops clouds the graph. Graph 5.17 filters

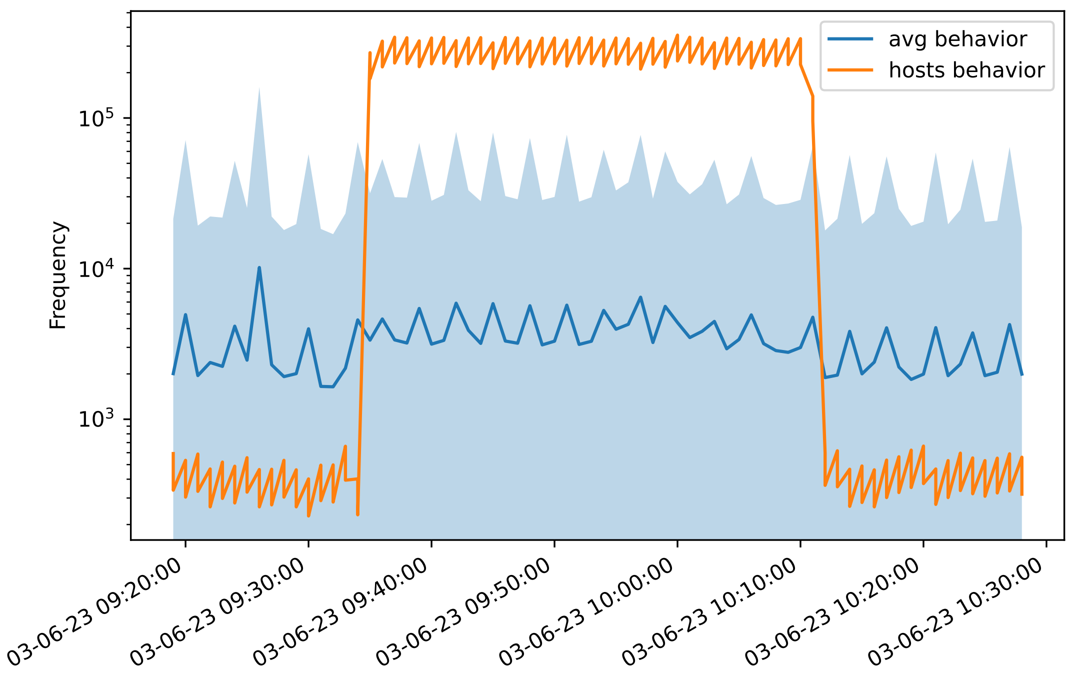


Figure 5.15: Average packets per minute during attack

this connection, leading to a clearer graph. There are two lines belonging to two different connections that portray the exact same behavior over a certain amount of time, though with differing packet frequencies. Both connections correspond to the attack. The slightly higher frequencies, the dark blue line, corresponds to the traffic flowing from the attacker to the target VM. The turquoise line displays the traffic flowing in the opposite direction.

After looking at the traffic flow from a more generic view through averages and standard deviations, Figure 5.18 scatters the connections during the duration of the attack and the 15 minutes prior and posterior to it. The original scatter plot portrays a lot of connections and does not provide any clear picture that allows to extract any patterns. Figure 5.18 represents a filtered selection of the connections. The filtering process uses two conditions to consider a connection valid. A valid connection is one that shares some basic characteristics with the brute-force attack. The first condition is that the connection does not cease over a certain period of time. The length of the interval is irrelevant, since a brute-force login attack can be of any length, from a couple of minutes to days. What matters is that over that period the connection remains uninterrupted. The other condition is that the difference in the average packet size does not surpass 10 bytes. This

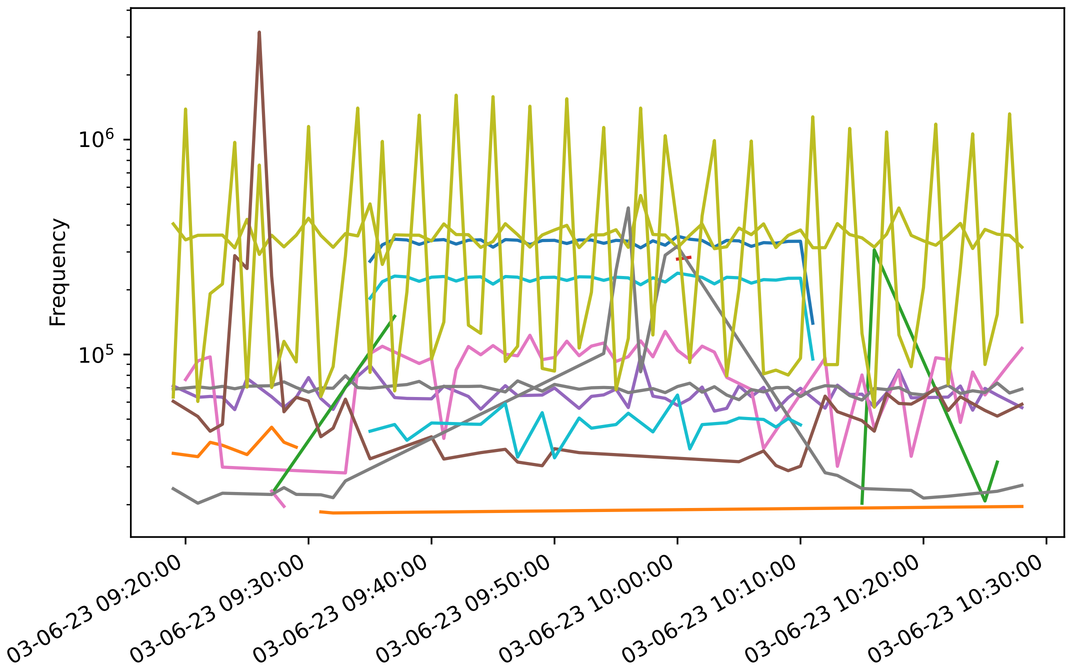


Figure 5.16: Outliers during attack

condition is drawn from the host level analysis, where the packet sizes during the attack are mainly 66 bytes long and the range of differing values is fairly narrow. Further, the central monitoring presents the total amount of packets and number of bytes of each connection per minute and not individually, which also diminishes the variance in packet sizes.

In graph 5.18 every color represents a different connection. The dark green connection at a frequency slightly over 200.000 packets and the olive green connection at a little over 300.000 packets share a parallel behavior. The olive green connection belongs to the traffic from the attacker VM and the other connection corresponds to the traffic from the victim VM. The shape of both traffics is exactly the same. The only difference is the downwards shift of the victim's traffic. This behavior reminds of the one that appears on the outliers graph 5.17. The shapes are very similar and the parallelism between both traffic flows match. The scatter plot offers an alternate way to discover a brute-force login attack in the central network traffic without assuming that the attack is going to produce outliers. Among the rest of connections there does not seem to exist any relation or pattern.

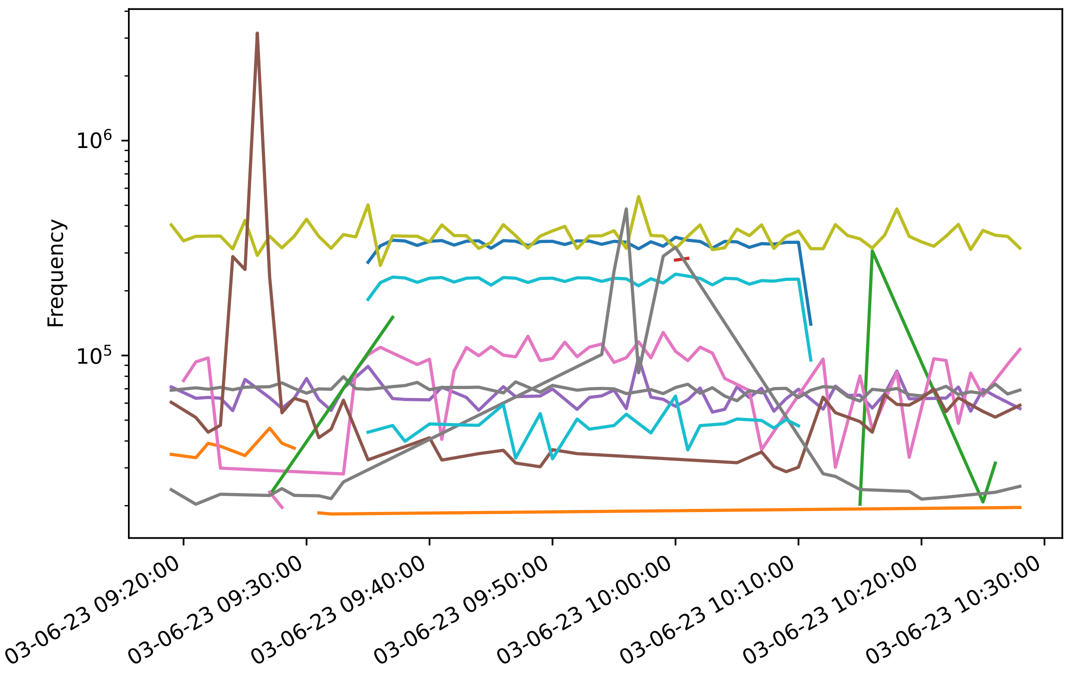


Figure 5.17: Outliers during attack filtered

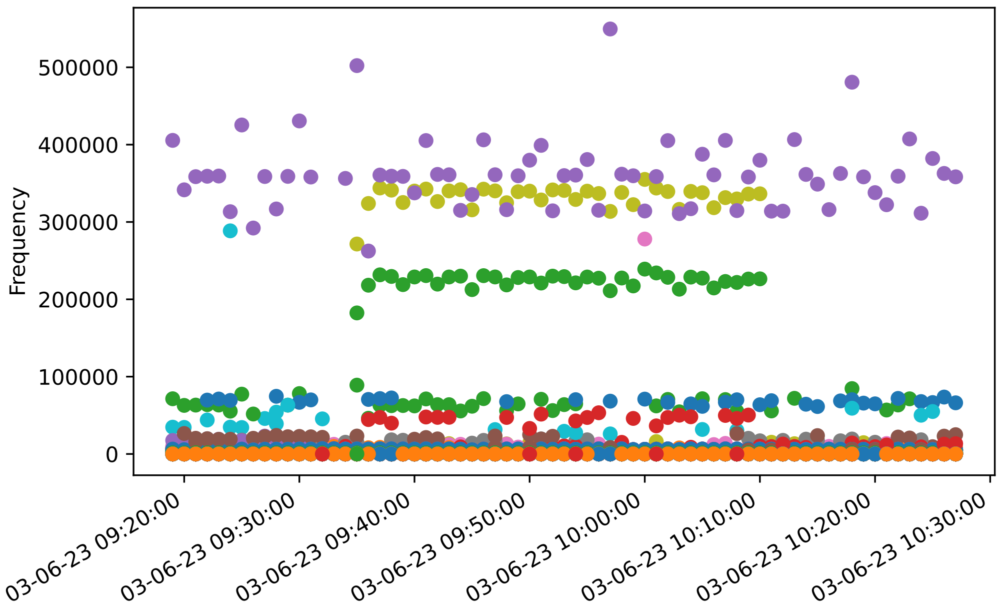


Figure 5.18: Filtered packets per minute during attack

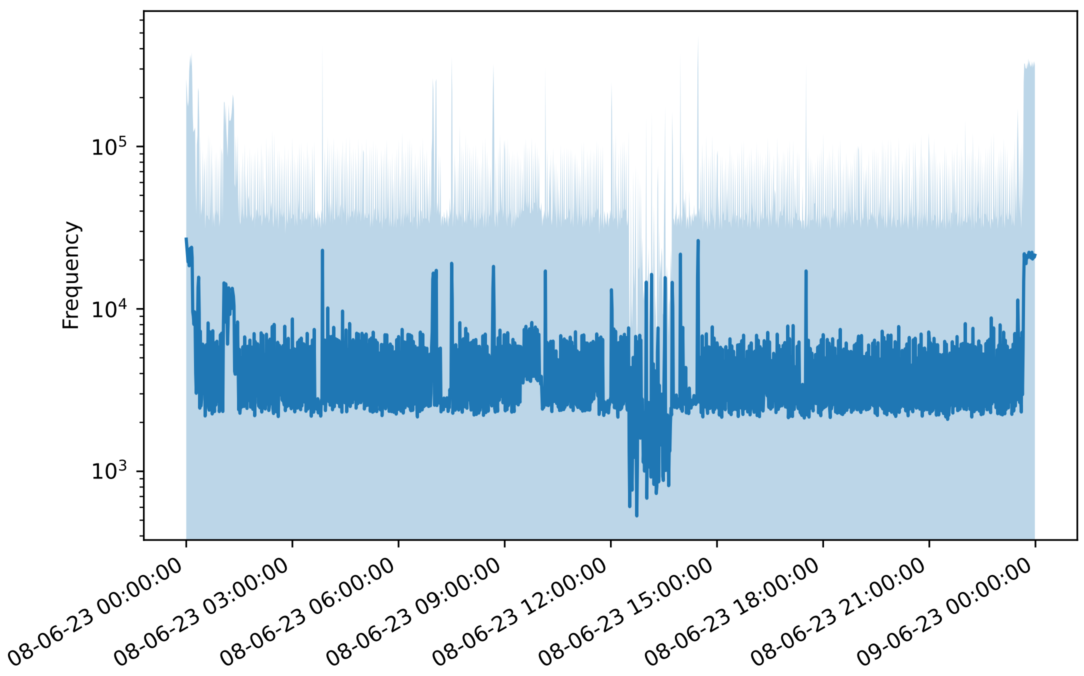


Figure 5.19: Average packets per minute over a different day

Figures 5.19 and 5.20 illustrate again the average packets per minute and the average packets per minute during the brute-force attack, but on a different day. The network traffic on the 8th of June is substantially different to the network flow of the 3rd of June. The amount of packets sent throughout the whole day remains fairly constant, regardless of night or day, except for the lapse between 12:00 and 15:00. The aim is to make sure that the brute-force login can still be detected and that the previous analysis on the outliers is consistent. Figure 5.20 displays in dark blue the average packet frequency at each moment for all the connections in the network traffic. The light blue region represents its respective standard deviation. The orange line depicts the average packet frequency of the attacker and the target. Again the attack traffic between the hosts presents a fairly constant and repetitive behavior, considered outliers in comparison to the average traffic. Therefore, the parallel behavior of two connections when scrutinizing the outliers of the traffic flow is a reasonably valid indicator. Identifying this feature should set off an alarm that a potential brute-force login attack against a MySQL server is taking place.

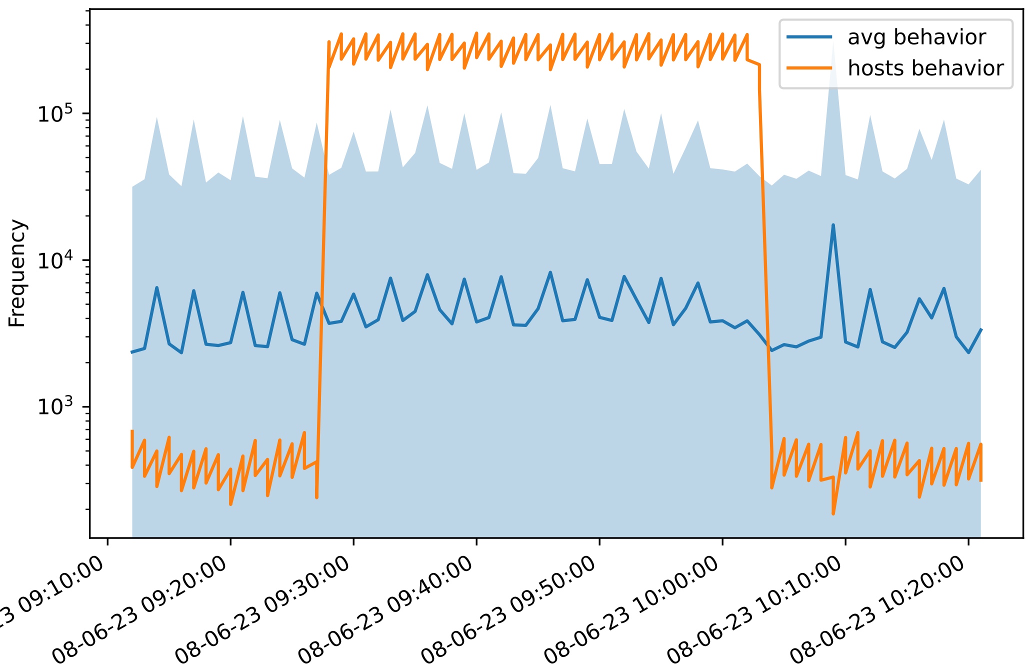


Figure 5.20: Average packets per minute during attack of a different day

The central monitoring also gathers information about the number of bytes sent per minute 5.21 and the average packet size 5.22. Graph 5.21 displays the number of bytes sent every minute during the attack and shares some similarities with figure 5.15. The frequency, as is logical, is much higher, but the shape of the average behavior of all connections is akin. The hosts traffic though remains within the standard deviation. The resolution of Figure 5.21 is smaller than the resolution of Figure 5.15, which does not allow to recognize as clearly the peaks and drops in the number of bytes per minute during the brute-force attack. During the normal behavior the number of bytes remains on a range of $3{\cdot}10^5$ to $7{\cdot}10^5$, but the shape is completely random and does not respond to any pattern or uniformity.

The similarity and proportionality between the number of packets and number of bytes sent per minute during the attack phase leads to graph 5.22. The average behavior of all the connections during the whole interval is flatter than the averages of Figure 5.15 and 5.21. The logarithmic scale of the chart may misguide about the variance of the traffic of the normal behavior and the attack. But comparing the values of the y-axis, it becomes clear, that the most constant values belong to the average packet size. During the brute-forcing the average packet size presents a periodical shape, that repeats itself throughout the duration of the brute-force login in a fairly narrow margin of bytes. The host-level analysis may provide a

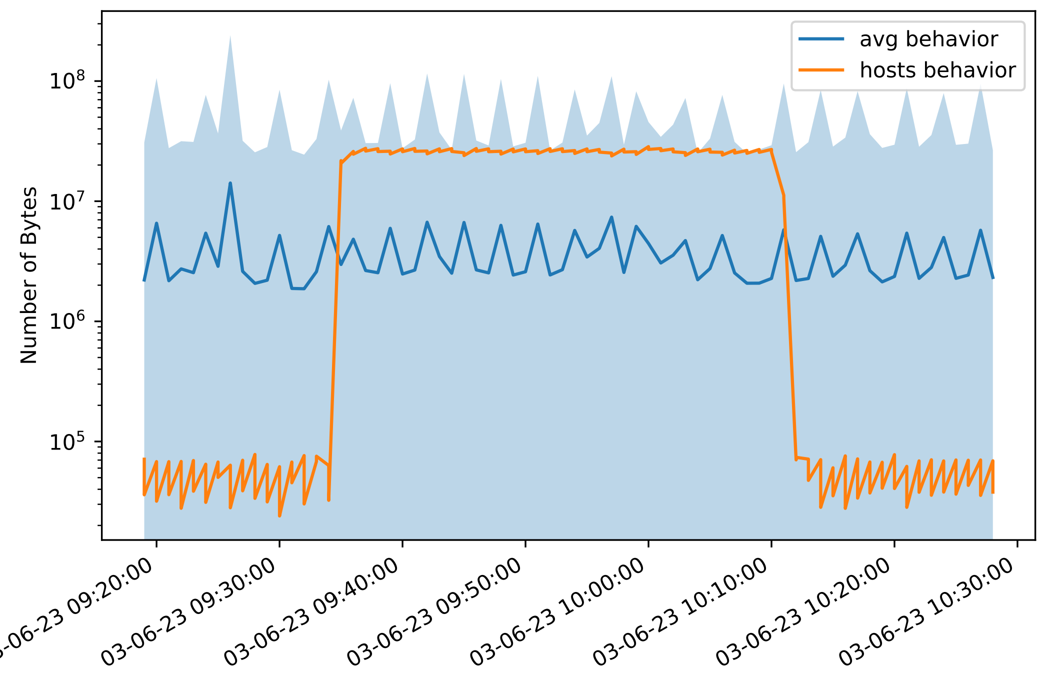


Figure 5.21: Number of bytes sent during attack

feasible explanation for this behavior. The packet sizes during the brute-force attack don't vary much, specifically 60 and 74 bytes long packets make up for 80% of the packets sent. An increase in packet frequency per minute directly increases the number of bytes sent. Given the fairly constancy of the packet sizes, the ratio between packet frequency and number of bytes is likely to work within a narrow range of values. The range of values lies within the standard deviation of the average packet size of all the connections. However, the zigzag shape is certainly peculiar and presents an odd behavior. But the behavior gets lost within all of the connections, regardless of the application of any filters to the central network monitoring traffic.

Neither of the parameters, number of bytes and average packet size, present any outliers. The number of bytes scratch the limit of the standard deviation and the average packet size lies considerably lower than the average behavior of the connections. Though they lie within the average region, they show a characteristic and constant behavior over the attack phase. Contrary to the frequency of packets, the peculiarity of the number of bytes and the average packet size is not identifiable in the network flow. Despite applying a filter, the overall traffic flow concentrates the majority of the packets around the same values as the parameters do during the credential stealing. The closeness and similarity of the values makes it very difficult to filter out any further data to recognise any patterns within the traffic. None of the parameters lead to the identification of any underlying pattern or succession of data points that stands out from the rest.

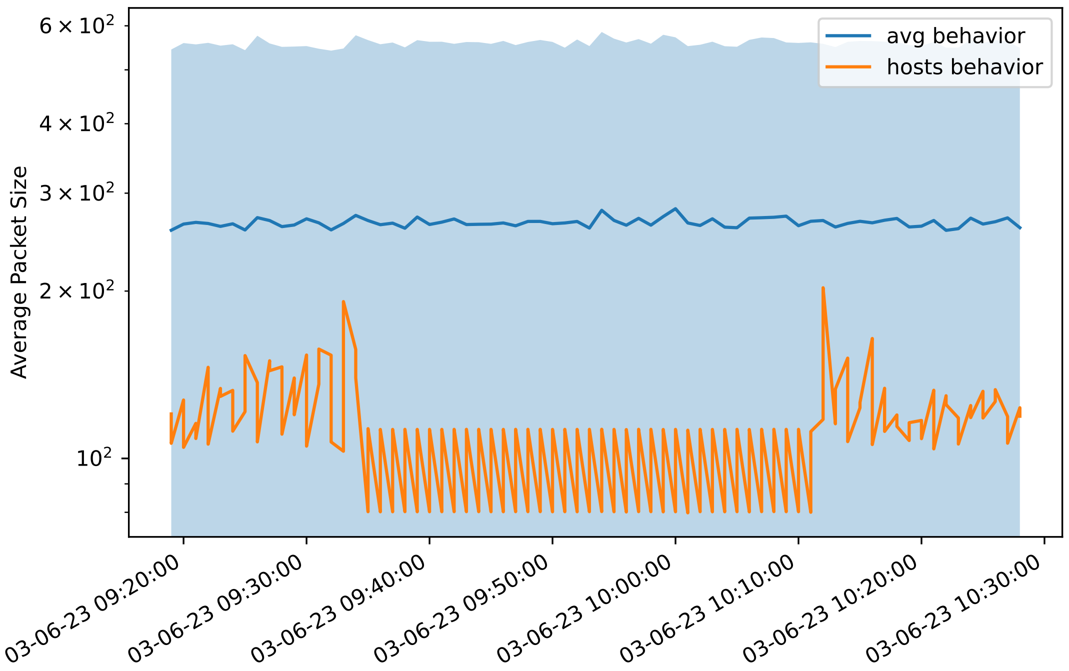


Figure 5.22: Average packet size during attack

5.2.2 Evaluation of using WSBMs for Detection of Brute Force Attack

Previous work used Weighted Stochastic Block Models (WSBMs) to detect targeted host scanning attacks [SKK22]. Weighted Stochastic Block Models are latent generative probabilistic graph models often used to generate new graphs with certain properties. WSBMs fit to observed traffic can be used to represent network communication behaviors. Communication patterns and structures manifest in node to group membership and in group-to-group relations described through block matrices. The comparison of network traffic to the model allows to calculate likelihoods and therefore detect abnormal behaviors. As a final evaluation, we use AwareNet as proposed in [SKK22] to detect brute-force login attacks against a MySQL server.

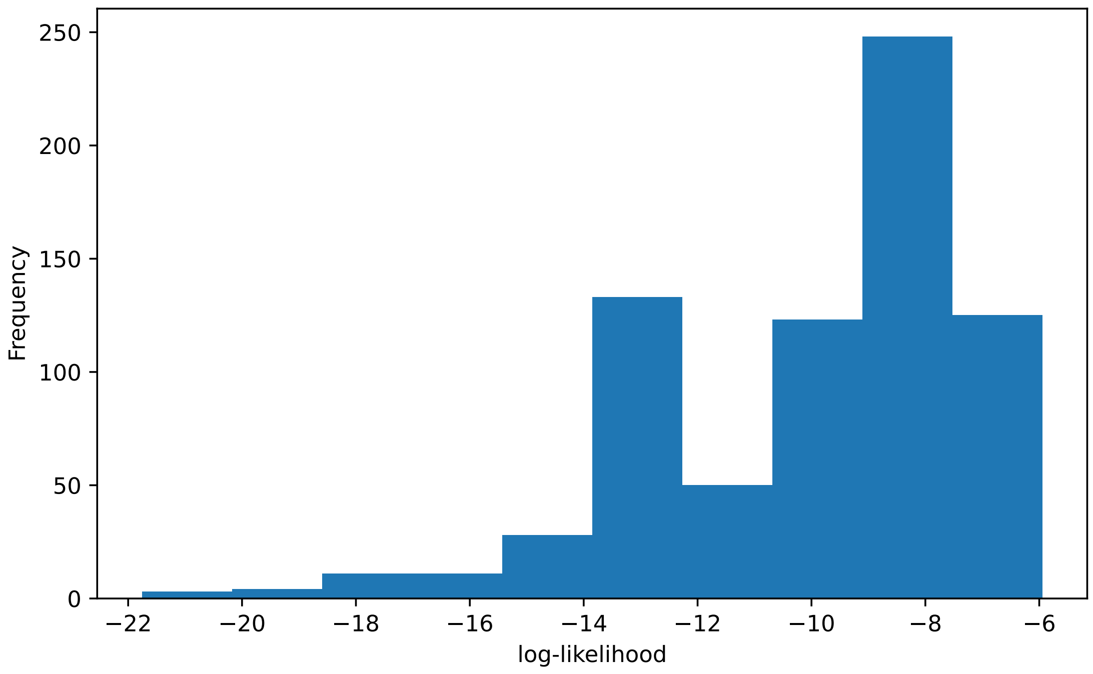


Figure 5.23: Initial distribution log-likelihoods

Figure 5.23 illustrates the distribution of edge log-likelihoods for the initial data. The initial data is a slice of 10 minutes of the traffic flow in the network. All the values fall between -6 and -22. These values establish a notion of the values that the model expects to encounter in the traffic based on the initial data. By themselves the values are meaningless. After fitting the model, AwareNet calculates the edge log-likelihoods for subsequent 10 minute blocks of the central monitoring data. The data contains the brute-force attack, which lasts almost 40 minutes, as well as 10 minutes of normal traffic prior and posterior to the attack. Figure 5.24 displays edge log-likelihoods from source group 105 over time. The graph plots the nodes assigned to the group considering the source address. Figure 5.25 does the same with the nodes of group 102. As a note, the numbering of the groups does not represent the amount of groups that exist. However, the assignment and calculation of the group numbering goes beyond the scope of the thesis. The target VM belongs to group 102 as source and the attacker VM to group 105 .

The blue line in both figures uses the average value of all nodes except the attacker in group 105 and the target in group 102. The line represents the average values of all ”normal” edges. The rationale for averaging the log-likelihoods of the

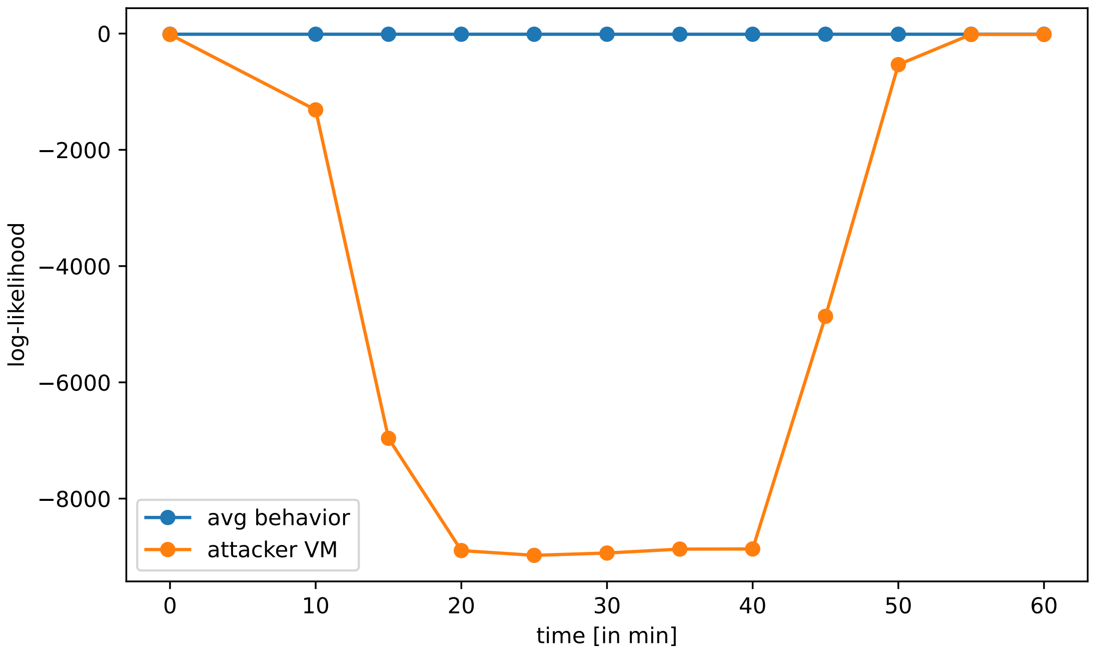


Figure 5.24: log-likelihoods group 105 over time

nodes instead of plotting each node separately is that all present a very ”normal” behavior and all lines overlap. The average behavior of group 105 does not drop below -11.95 and group 102 stays above -12.20 all the time. Both values are close to the mean of the initial log-likelihoods in Figure 5.23. For that reason the edges are considered ”normal” and the traffic ordinary.

In contrast, the log-likelihood values for the attack edge, shown in orange, drop under -8k for group 105 and under -35k for group 102 and never exceed -15.20 and -12.21 respectively. The numeric value itself of the log-likelihood lacks any special interest. The only important feature for the purpose of the project is whether or not the log-likelihood has a similar value to the expected behavior or not. The resulting plots showcase that AwareNet is capable of detecting brute-force login attacks against MySQL servers. This discovery encourages the possibilities to use probabilistic models for network traffic analysis and the application of AwareNet to try to detect other attack types.

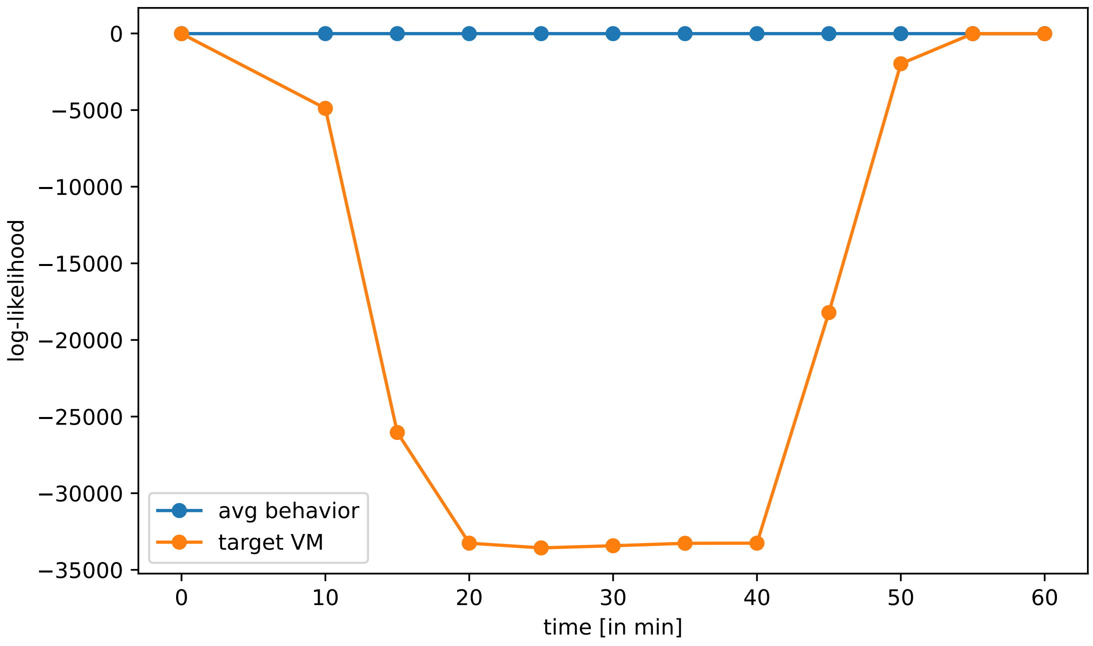


Figure 5.25: log-likelihoods group 102 over time

Chapter 6

Conclusions and Outlook

To reduce the impact of breaches in cybersecurity the development of detection and mitigation mechanisms plays a key role. Due to the wide range of attack forms and vectors and the constant increase in cyberthreats, detection and mitigation is one of the main research fields in cybersecurity. There exist countless researches on different attacks and detection approaches with promising results. The high accuracy of the algorithms and methods under study, however, often apply only within specific domains. When implemented on other networks or attacks the effectiveness of the technologies tend to drop considerably. This phenomenon exposes the existing gap between controlled environment and real-world network attack detection.

One of the rationales for the inability of the algorithms to work on real-world environments is the training data to develop the algorithms. Many studies use benchmark intrusion detection datasets to train models that report excellent performances of their detection mechanisms on training and validation datasets, without including testing in real life scenarios. Benchmark IDDs, such as the Kyoto 2006+ or the CSE-CIC-IDS2018 dataset, are very useful for preliminary training and modelling of the algorithms. But they do not come without their shortcomings. One of the most important ones most of the datasets share is the inability to represent a valid real life scenario with all its complexity. Further, the purpose of datasets is to store information. Sometimes though, it is necessary to adjust the data collected answering the question "what should be monitored?". These benchmark datasets are very standardized in the field of cybersecurity, but the specificity and immutability of their data limits the usage and utility for certain purposes.

In addition, for some of the available datasets the testbed environments are not accessible. The data is therefore collected in unknown environments. Traffic data is inherently dependent, due to unobserved latent factors that act locally on each network. Disregarding these dependencies and neglecting the existing links between traffic generated within a single network can provide limited validity to the detection of abnormal network behaviour or other malicious traffic.

Machine learning algorithms are frequently chosen for detection purposes. An important challenge for many of the algorithms is the inability to distinguish between attack traffic flows and normal traffic flows. Moreover, many ML algorithms for example rely heavily on the availability of a labelled dataset. Data labelling and the complexity of the results of certain algorithms require a network expert, which is not very optimal.

The thesis focuses on laying the groundwork for applying data-driven methods to network attack traffic analysis. To overcome some of the stated problems the project implements a self-developed environment, where a brute-force login attack can be carried out in a secure and observable manner. Ground truth network data from the involved hosts and the related data from central network monitoring is collected. The resulting dataset undergoes a thorough exploratory data analysis at host and at network level. The knowledge acquired should contribute to the development of methods for identifying traffic patterns of a brute force-login attack against a MySQL server, directly contributing to the improvement of network security.

One of the key elements of this project is the self-developed testbed environment.The testbed environment of the thesis mimics the internal attack scenario of a campus network with three major goals: provide a topological description on how a credential theft occurs; achieve attack pattern extraction from raw sniffed data; and establish attack pattern identification as a parameter to visualize real-time attacks at host and at network level. The advantage of using a self-developed testbed is being able to access all knowledge around the traffic generated, the conditions of the network and the devices involved, as well as having the capacity to unlimitedly apply any changes to the testbed to make it more suitable to any specific needs.

The exploratory analysis both at host and at network level generates different visualizations of the traffic data. The plots do not require a deep knowledge of communication networks and have the purpose that at simple sight any viewer recognizes any patterns or odd behaviors. The visualizations apply some basic filtering, but do not rely on any algorithm or machine learning method. The Weighted Stochastic Block Models, on the other hand, are based on slightly more complicated algorithms. But the resulting graphs, as is the case with the exploratory data analysis, are fairly simple to interpret and provide valuable insight for brute-force login attacks recognition.

The exploratory analysis at host level reveals that, while working with a MySQL server, the TCP and MySQL protocol present a parallel behavior with simultaneous drops and spikes. During a brute-force login attack against a MySQL server there is a sudden and prolonged spike of the TCP and MySQL packet protocols. During the normal usage of the server both protocols present a fairly low packet frequency, with bare packet density changes within a narrow margin. During the attack, however, the packet frequency increases substantially and experiences considerable fluctuations. In regards to packet lengths, the most common packet sizes are 66 bytes for the TCP and 161 bytes for the MySQL protocol during both behaviors. All in all, the packet flow concentrates on the lower packet sizes between 54 and 80 bytes during the attack, and between 66 and 110 bytes for the normal behavior.

At network level, anew the most relevant and insightful parameter is the packet frequency. The average packets sent per minute during the attack rank as outliers in comparison to the average behavior of the packet frequency of all the involved connections in the central monitoring. When picturing all outliers in the traffic, two connections stand out because of their parallel behavior. The connections correspond to the traffic flowing from the attacker to the target VM and viceversa. The same behavior is also perceivable among the whole traffic from the central monitoring after a filtering process. The filtering keeps the connections that generate an uninterrupted flow of data over a period of time and whose packet frequency does not fluctuate much. These are two characteristics of the brute-force login attack inferred from the host-level data exploratory analysis. Within the turmoil of the filtered network traffic the same parallel behavior pops up. The other two features under scrutiny are the number of bytes sent and the average size of the packets. Despite portraying some odd behaviors, the values are too close to the average values of the rest of connections and are non-conclusive in the traffic flow of the brute-force login attack.

The WSBM fits a model that allows to calculate log-likelihoods based on some initial traffic data from the central monitoring. The model divides the traffic network into groups and compares afterwards the traffic from each group to the fitted model. Both virtual machines, attacker and victim, present abnormal behaviors in their traffic, with log-likelihoods very far from the values the model considers "normal" or expected network traffic. This showcases that WSBMs have the capability to detect a brute-force login attack against a MySQL server.

The thesis manages to build a testbed environment to perform a brute-force attack in a secure and observable manner. The data analysis successfully uses the monitoring data at host and network level to visualize the traffic and identify traces of the brute-force attack. The Weighted Stochastic Block Model AwareNet evaluates the data and manages to distinguish the anomaly case, meaning the brute-force attack, from the expected normal network traffic.

One improvement for the project could involve designing the MySQL behavior to be more specific and particularize the actions, instead of performing random and basic queries. Resembling a specific behavior or usage may have a different outcome in the network traffic. Future work may include the application of other probabilistic models or machine and deep learning methods for network traffic analysis. Another possibility could include broadening the scope of attacks and testing the ability of AwareNet to detect them. Since AwareNet has the capability to detect brute-force attacks, it could be interesting to test its effectiveness with distributed brute-force attacks. The attacks are increasing in popularity and are more challenging to detect, since multiple machines carry out the attack instead. Other attack possibilities include DoS or man-in-the-middle attacks, which are very frequent attack vectors.

As final remark it is worth mentioning that the fact that some parameters reveal odd behaviors is non-conclusive for other attacks or networks. Each network has its own traffic flow. Though the thesis provides answers to the question ”How should monitoring be designed to enable detection of network attacks?”, there is no universal truth. For that reason it is important to not treat networks as a black-box and design tailored systems. The thesis has found some parameters and features that partially answer the question for the specific campus network under study, since the monitoring to enable network attack detection is a very broad subject. For other traffic flows the insightful features for this network may not provide any valid results and vice versa.

Bibliography

[AAM$^+$21] Malak Aljabri, Sumayh S. Aljameel, Rami Mustafa A. Mohammad, Sultan H. Almotiri, Samiha Mirza, Fatima M. Anis, Menna Aboulnour, Dorieh M. Alomari, Dina H. Alhamed, and Hanan S. Altamimi. Intelligent Techniques for Detecting Network Attacks: Review and Research Directions. *Sensors (Basel)*, 21(21):7070, 2021.

[cyb18] cybercrimemag. Cybercrime To Cost The World $10.5 Trillion Annually By 2025. `https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/`, 2018.

[ELML08] K. Sakurai J. Ryou E. L. Malecot, Y. Hori and H. Lee. (Visually) Tracking Distributed SSH BruteForce Attacks? *3rd International Joint Workshop on Information Security and Its Applications (IJWISA 2008)*, pages 1–8, 2008.

[FB19] Thorben Funke and Till Becker. Stochastic block models: A comparison of variants and inference methods. *PLoS ONE*, 14:e0215296, 2019.

[GB19] Gaddam Venu Gopal and Gatram Ramamohan Babu. A Detailed Study on A Benchmark Intrusion Dataset - Kyoto 2006+. *International Journal of Emerging Trends in Engineering Research*, 8(10):7228–7231, 2019.

[gra23] graph-tool: Efficent network analysis with python. https://graph-tool.skewed.de/, 2023.

[Ins22a] Ponemon Institute. 2022 cost of insider threats global report. 2022.

[Ins22b] Ponemon Institute. Cost of a Data Breach Full Report 2022. `https://www.ibm.com/downloads/cas/3R8N1DZJ`, 2022.

[KGB20] Avinash Kumar, William Glisson, and Ryan Benton. Network attack detection using an unsupervised machine learning algorithm. *Hawaii International Conference on System Sciences*, 2020.

[mat23] Matplotlib documentation — matplotlib 3.7.1 documentation. https://matplotlib.org/stable/index.html, 2023.

[met23] Metasploit Documentation Penetration Testing Software, Pen Testing Security. `https://rapid7.github.io/metasploit-framework/docs/pentesting/metasploit-guide-mysql.html`, 2023.

[MS15] Nour Moustafa and Jill Saly. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *IEEE Xplore*, pages 1–6, 2015.

[NKK$^+$11] Maryam M. Najafabadi, Taghi M. Khoshgoftaar, Clifford Kemp, Naeem Seliya, and Richard Zuech. *2014 IEEE International Conference on Bioinformatics and Bioengineering*. IEEE, 2014-11.

[Ora23] Oracle. Mysql :: Mysql customers. `https://www.mysql.com/customers/`, 2023.

[pan23] pandas - python data analysis library. `https://pandas.pydata.org/`, 2023.

[RCCL22] Arnaud Rosay, Eloïse Cheval, Florent Carlier, and Pascal Leroux. Network Intrusion Detection: A Comprehensive Analysis of CIC-IDS2017:. *SCITEPRESS - Science and Technology Publications*, pages 25–36, 2022.

[RN18] Mark Patrick Roeling and Geoff Nicholls. Stochastic block models as an unsupervised approach to detect botnet-infected clusters in networked data. *WORLD SCIENTIFIC (EUROPE)*, 03:161–178, 2018.

[Rol18] Alexis Rolland. Star wars data. `https://github.com/alexisrolland/star-wars-data/blob/d72f819e8309c1c508be23c879204277977c61f1/database.sql`, 2018.

[SKK22] Maximilian Stephan, Patrick Krämer, and Wolfgang Kellerer. AwareNet: using WSBMs for network traffic analyis. *Proceedings of the 3rd International CoNEXT Student Workshop*, pages 35–36, 2022.

[Sud16] Dave Sudia. Hogwarts PG. `https://github.com/thedevelopnik/hogwarts-sql/blob/6e7fc523bd9a331f1150f29e305ad2c2decae4ea/sql/seed.sql`, 2016.

[tcp23] tcpdump. Home | TCPDUMP & LIBPCAP. `https://www.tcpdump.org/`, 2023.

[WYWA17] Jie Wang, Lili Yang, Jie Wu, and Jemal H. Abawajy. Clustering analysis for malicious network traffic. *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.

Appendix A

Alignment of the Project with SDGs

All United Nations Member States adopted the 2030 Agenda for Sustainable Development. The Agenda sets a shared blueprint to promote peace and prosperity across the planet for every individual. The Sustainable Development Goals (SDGs) at the core of the Agenda are 17 issues calling for an urgent action worldwide. The thesis tries to contribute to the promotion of the SDGs within its possibilities.

Cyberattacks are one of the biggest challenges companies, individuals, governments and organizations are currently confronted with. It is therefore of utter relevance to develop detection and prevention mechanisms to detect these attacks, that are not only highly reliable, but also accessible to everybody. Cybersecurity Ventures, one of the world's leading researcher and publisher covering the global cyber economy, expects the costs of cybercrime worldwide to grow by 15 percent per year, from 3 trillion USD in 2015 to 10.5 trillion USD annually by 2025, without any prospects of cybercrime stagnating or slowing down in the near future [cyb18].

The main objective of the thesis focuses on developing mechanisms to detect attacks to contribute to the network security of a campus. The project therefore aligns with SDGs 8, Decent work and Economic growth; 9, Industry, Innovation and Infrastructure; and 16, Peace, Justice and Strong Institutions. The prevention of attacks allows entities to allocate part of the money lost due to security breaches in other sectors. The investment in Industry, Infrastructure and Innovation would thereby increase, leading to more development in these branches, which automatically translates into economic growth.

Besides, developing detection methods fosters innovation and leads to the protection of infrastructures. Not only private information is threatened by hackers, ports, airports, in brief, modern infrastructures, have also proven to be vulnerable to cyberattacks. Preventing the threats or at least mitigating them would greatly benefit these sectors contributing to their increase in safety and further development.

With regards to the 16th goal, the project contributes to the enhancement of network security in a campus. The idea is to reduce the risk of suffering severe data breaches through the development of detection and monitoring mechanisms. Lastly, the 5th SDG, Gender equality, is also promoted as a side-effect of conducting the thesis. Male figures predominate in the IT field. Having a woman carry out this research is a statement that gender does not play a role, regardless of the field of work. Hopefully this will open up the door or at least encourage more female students to pursue researches and careers in this area.

Appendix B

Notation and Abbreviations

CDF	Cumulative Distribution Function
DoS	Denial of Service
IDD	Intrusion Detection Dataset
IDS	Intrusion Detection System
ML	Machine Learning
SBM	Stochastic Block Model
SDG	Sustainable Development Goals
SQL	Structured Query Language
TCP	Transmission Control Protocol
UN	United Nations
VM	Virtual Machine
WSBM	Weighted Stochastic Block Model

Appendix C

Attack code

Listing C.1: Complete Code

```python

import libtmux
import subprocess
import paramiko
import time
import datetime
import json
import os
import random
import csv


class AttackHandler:
    """
    AttackHandler class will handle the whole attack. This consists of:
    - creating tmux server and session
    - starting tcpdump inside tmux session
    - starting attacks by sending the attack commands into tmux session
    - checking whether attack has finished
    - saving logfiles and timestamps of the attacks
    """
    def __init__(self):
        """
        Initialize the AttackHandler object parameters.
        :param tmux_server: current tmux server
        :param tmux_session: current tmux session
        :param tmux_window: current tmux window
        :param tmux_pane1: tmux pane for attack commands
        :param tmux_pane2: tmux pane for tcpdump

        :param overall_start_time: start time of script/attack
        :param log_path: path for log files
        :param config_path: path of config.json file
        :param config_data: json config data will be loaded into this variable
        :param current_attack_start_time: start time of the current attack
        for timestamps
        :param current_attack_end_time: end time of the current attack
        for timestamps
        :param current_attack_finished: elapsed time of the current attack
        :param attack_counter: counter for total number of executed attacks
        :param total_attack_set_iterations: amount of times the attack
        set should be ran
        :param attack_set_iteration_counter: counter for iterating through
        the attack list
        :param attack_list: list with all the attack commands that
        will be executed

        Calls the intialization method.
        """
        self.tmux_server = None
        self.tmux_session = None
        self.tmux_window = None
        self.tmux_pane1 = None
        self.tmux_pane2 = None

        self.overall_start_time = datetime.datetime.now().
        strftime("%Y_%m_%d-%I_%M_%S_%p_UTC")
        self.log_path = '/home/lkn/attack-traffic-testbed/kali1/logs/'
        self.mysql_path = '/home/lkn/attack-traffic-testbed/kali1/mysql/'
        self.config_path = '/home/lkn/attack-traffic-testbed/kali1/
        config_mysql.json'
        self.config_data = None
        self.current_attack_start_time = None
        self.current_attack_end_time = None
        self.current_attack_finished = True
        self.string_start_time = None
        self.current_timestamp = None
        self.attack_counter = 0
        self.total_attack_set_iterations = 0
        self.attack_set_iteration_counter = 0
        self.attack_list = []

        self.initialization()

    def initialization(self):
        """
        Initialization calls the methods which should be run
        once before starting the attacks.
        """
        self.create_json()
        self.create_logfile()
        self.read_config()
        self.create_attack_list()
        self.initialize_tmux_server()
        self.start_tcpdump()

    def create_json(self):
        """
        Creates an empty json skeleton, to which the timestamps
        of the several attack can be added.
        """
        json_skeleton = {"attacks": []}
        with open(self.log_path + 'attack_timestamps_' + self.overall_start_time +
        '.json', 'w') as f:
            json.dump(json_skeleton, f, ensure_ascii=False, indent=4)

    def create_logfile(self):
        """
        Creates empty logfiles.
        Current_attack logfile is used to check the state (running or finished)
        of the current attack.
        Attack logfile logs the commandline output of an attack.
        """
        with open(self.log_path + "current_attack_" + self.overall_start_time +
        ".log", 'w') as f:
            pass

        with open(self.log_path + "attack_" + self.overall_start_time +
        ".log", 'w') as f:
            pass

    def read_config(self):
        """
        Reads the config.json and save the data to self.config_data variable.
        Data will be used to create the attack list.
        """
        with open(self.config_path) as json_file:
            self.config_data = json.load(json_file)

    def select(self):
        databases = {'star_wars': {'planet': ['*', 'name', 'rotation_period',
        'orbital_period', 'diameter', 'climate', 'gravity', 'terrain',
        'surface_water', 'population', 'created_date', 'updated_date','url','id'],
        'people': ['*', 'name', 'height', 'mass', 'hair_color', 'skin_color',
        'eye_color', 'birth_year', 'gender', 'planet_id', 'created_date',
        'updated_date', 'url', 'id']},
        'hogwarts': {'parents': ['*', 'name'],
        'students': ['*', 'name', 'year', 'house_id'],
        'teachers': ['*', 'id', 'name', 'house_id'],
        'class_rosters': ['*', 'class_id', 'student_id'],
        'houses': ['*', 'id', 'name'],
        'classes': ['*', 'id', 'subject', 'teacher_id']}}

        rndDB = random.randint(0, len(databases.keys())-1)
        database = list(databases.keys())[rndDB]
        rndTable = random.randint(0, len(databases[list(databases.keys())
        [rndDB]])-1)
        table = list(databases[list(databases.keys())[rndDB]].keys())[rndTable]
        rndColumn = random.randint(0, len(databases[database][table])-1)
        column = list(databases[database][table])[rndColumn]
        return 'USE '+ database + '; ' + 'SELECT '+ column + ' FROM ' + table +';'

    def insert(self, file):
        lista = []
        with open(file, 'r') as doc:
            csvreader = csv.reader(doc, delimiter=')')
            for row in csvreader:
                lista.append(row[0]+')')
        lista = random.sample(lista, random.randint(1, min(len(lista),25)))
        lista = str(lista).replace('[','')
        lista = lista.replace(']','')
        if file == 'mysql/planetsSW.csv':
            return 'USE star_wars; INSERT INTO planet(name,rotation_period,
            orbital_period,diameter,climate,gravity,terrain,surface_water,
            population,created_date,updated_date,url,id) VALUES ' +
            lista.replace('\'','') + ';'
        elif file == 'mysql/peopleSW.csv':
            return 'USE star_wars; INSERT INTO people(name,height,mass,hair_color,
            skin_color,eye_color,birth_year,gender,planet_id,created_date,
            updated_date,url,id) VALUES ' + lista.replace('\'','') + ';'
        elif file == 'mysql/parentsHw.csv':
            return 'USE hogwarts; INSERT INTO parents (name) VALUES ' +
            lista.replace('\'','') + ';'
        elif file == 'mysql/studentsHw.csv':
            return 'USE hogwarts; INSERT INTO students (name, year, house_id)
            VALUES ' + lista.replace('\'','') + ';'
        elif file == 'mysql/class_rostersHw.csv':
            return 'USE hogwarts; INSERT INTO class_rosters (class_id, student_id)
            VALUES ' + lista.replace('\'','') + ';'

    def dropCreate(self, file):
        create = []
        with open(file, 'r') as doc:
            csvreader = csv.reader(doc, delimiter=';')
            for row in csvreader:
                create.append(row[0])
        create = random.sample(create,1)
        create = str(create).replace('\'', '')
        create = create.replace('[', '')
        create = create.replace(']', '')
        table = create[create.find('table')+5:create.find('(')]
        if 'planet' in table or 'people' in table:
            return 'USE star_wars; DROP TABLE IF EXISTS'+table+'; '+ create +';'
        else:
            return 'USE hogwarts; DROP TABLE IF EXISTS'+table+'; '+ create +';'
    def random_commands(self, typeAttack):
        randomIndex = random.randint(0,5)
        if randomIndex == 0:
            return self.select()
        elif randomIndex == 1:
            return self.dropCreate(self.config_data["attack_set"][typeAttack]
            ["details"][randomIndex]["Command"])
        elif randomIndex in range(2,7):
            return self.insert(self.config_data["attack_set"][typeAttack]
            ["details"][randomIndex]["Command"])

    def create_attack_list(self):
    """
    Creates the attack list.
    Checks for the attack type and adds attack command to the list.
    Same attack can be run multiple times behind each other
    and is specified in the iterations key.
    """
    self.attack_list = []
    for i in range(len(self.config_data["attack_set"])):
        if self.config_data["attack_set"][i]["type"] == "preAttack":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('mysql -u username -p password
                -h dst_ip -e \'' + self.random_commands(0) + '\'')
        if self.config_data["attack_set"][i]["type"] == "msfconsole":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('msfconsole -q -x \'use
                auxiliary/scanner/mysql/mysql_login;
                set RHOSTS dst_ip; set VERBOSE false;
                set USERNAME username; set PASS_FILE passwords.txt;
                set STOP_ON_SUCCESS true; run; exit\'')
        if self.config_data["attack_set"][i]["type"] == "flush-hosts":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
            self.attack_list.append('mysqladmin -u username -p password
            -h dst_ip flush-hosts')
        if self.config_data["attack_set"][i]["type"] == "postAttack":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('mysql -u username -p password
                -h dst_ip -e \'' + self.random_commands(0) + '\'')
        if self.config_data["attack_set"][i]["type"] == "tshark":
            for _ in range(self.config_data["attack_set"][i]["iterations"]):
                self.attack_list.append('tshark -r ' + self.log_path
                +'attack_kali1_' + self.overall_start_time +'.pcap -t ud
                -T fields -e _ws.col.Time -e ip.src -e ip.dst
                -e frame.len -e _ws.col.Protocol -E separator=,
                -E occurrence=f > ' + self.log_path + 'attack_kali1_'+
                self.overall_start_time +'.csv')

    self.total_attack_set_iterations = self.config_data["attack_set_iterations"]
    def check_attack_state(self):
        """
        Reads the last line of current_attack.log file to check whether
        the current attack has finished.
        Current attack has finished when last line holds the string:
        "Checked: Attack finished!"
        Returns True or False accordingly.
        """
        with open(self.log_path + "current_attack_" + self.overall_start_time +
        ".log", "rb") as file:
            try:
                file.seek(-2, os.SEEK_END)
                while file.read(1) != b'\n':
                    file.seek(-2, os.SEEK_CUR)
            except OSError:
                file.seek(0)
            last_line = file.readline().decode()

            print(last_line)

            if last_line == "Info: Attack finished.\n":
                print("Checked: Attack finished!")
                with open(self.log_path + "current_attack_" +
                self.overall_start_time + ".log", 'w') as f:
                    pass
                return True
            else:
                return False

    def set_attack_state_to_finished(self):
        """
        When attack has finished, current_attack_finished will be set to True,
        current_attack_end_time will be set to the current time and skipfish file
        will be removed from the directory, so that attack can be started again.
        """
        self.current_attack_finished = True
        self.current_attack_end_time =datetime.datetime.now(datetime.timezone.utc)

    def set_attack_state_to_running(self):
        """
        New attack started, current_attack_finished will be set to False.
        """
        self.current_attack_finished = False

    def add_current_attack_timestamps(self):
        """
        Gets the attack start and attack end timestamp and calculates
        the elapsed time.
        Timestamps of the current attack will be appended to the
        attack_timestamps.json file.
        """
        self.current_attack_start_timestamp = self.current_attack_start_time.
        timestamp()
        self.current_attack_end_timestamp = self.current_attack_end_time.
        timestamp()
        attack_time_elapsed = self.current_attack_end_timestamp -
        self.current_attack_start_timestamp

        current_attack_timestamps = {"id": self.attack_counter, "type":
        self.attack_list[self.attack_set_iteration_counter].split()[0], "t_start":
        str(self.current_attack_start_time), "t_end": str(
        self.current_attack_end_time), "t_elapsed": str(attack_time_elapsed)}
        self.attack_counter += 1

        with open(self.log_path + 'attack_timestamps_' + self.overall_start_time +
        '.json') as json_file:
            overall_attack_timestamps = json.load(json_file)
            temp = overall_attack_timestamps["attacks"]
            temp.append(current_attack_timestamps)

        with open(self.log_path + 'attack_timestamps_' + self.overall_start_time +
        '.json', 'w') as f:
            json.dump(overall_attack_timestamps, f, ensure_ascii=False, indent=4)

    def initialize_tmux_server(self):
        """
        Start the Tmux server and session.
        Opens two panes for tcpdump and attack commands.
        """
        self.tmux_server = libtmux.Server()
        self.tmux_session = self.tmux_server.new_session(session_name=
        "kali1_attack", kill_session=True, attach=False)
        self.tmux_window = self.tmux_session.new_window(attach=True,
        window_name="kali1_attack")
        self.tmux_pane1 = self.tmux_window.attached_pane
        self.tmux_pane2 = self.tmux_window.split_window(vertical=False)
        self.tmux_window.select_layout('even-horizontal')

    def start_tcpdump(self):
        """
        tcpdump will be started in Tmux pane 2.
        """
        self.tmux_pane2.send_keys('sudo tcpdump -ni eth0 -w ' + self.log_path +
        'attack_kali1_' + self.overall_start_time + '.pcap')

    def start_new_attack(self):
        """
        Attack commands will be run in Tmux pane 1.
        """
        self.tmux_pane1.send_keys(self.attack_list[
        self.attack_set_iteration_counter] + ' | tee ' + self.log_path +
        'current_attack_' + self.overall_start_time + '.log >> ' + self.log_path +
        'attack_' + self.overall_start_time + '.log && echo "Info:
        Attack finished." | tee ' + self.log_path + 'current_attack_' +
        self.overall_start_time + '.log >> ' + self.log_path + 'attack_' +
        self.overall_start_time + '.log')

    def stop_tmux(self):
        """
        Kills tmux server to stop tcpdump on both VMs.
        """
        # kill server on hmp1 VM to stop tcpdump
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect('ip_dst', username='username', key_filename=
        '/home/username/.ssh/id_rsa')
        ssh_stdin, ssh_stdout, ssh_stderr =client.exec_command('tmux kill-server')

        # kill on this kali1 VM to stop tcpdump
        self.tmux_server.kill_server()

    def start_attack_handler(self):
    """
    Main attack handler.
    Traverses through the attack list and runs the attack commands.
    Checks if current attack has finished. If yes next attack command
    of the list will be started.
    """
    for _ in range(self.total_attack_set_iterations):

        self.attack_set_iteration_counter = 0
        while self.attack_set_iteration_counter < len(self.attack_list):
            if self.current_attack_finished:
                self.current_attack_start_time = datetime.datetime.now(
                datetime.timezone.utc)
                self.start_new_attack()
                self.set_attack_state_to_running()

            if self.check_attack_state():
                self.create_attack_list()
                self.set_attack_state_to_finished()
                self.add_current_attack_timestamps()
                self.string_start_time = datetime.datetime.strptime(
                self.overall_start_time, "%Y_%m_%d-%I_%M_%S_%p_UTC")
                self.current_timestamp = datetime.datetime.strptime(
                datetime.datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p_UTC"),
                "%Y_%m_%d-%I_%M_%S_%p_UTC")
                if (self.current_timestamp - self.string_start_time >
                datetime.timedelta(hours=1) and self.attack_set_iteration_counter
                ==0)
                or (self.current_timestamp -  self.string_start_time >
                datetime.timedelta(hours=3) and self.attack_set_iteration_counter
                ==3)
                or self.attack_set_iteration_counter in [1,2,4]:
                    self.attack_set_iteration_counter += 1

            time.sleep(1.0)

    self.stop_tmux()


if __name__ == "__main__":
    attack = AttackHandler()
    attack.start_attack_handler()
```