Master of Science in Computer Science
May 2023

A method of detecting and predicting attack vectors based on genetic programming

Yekatierina Churakova
Oleksii Novikov

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

Contact Information:
Author(s):
Yekatierina Churakova
E-mail: yech22@student.bth.se

Oleksii Novikov
E-mail: olno22@student.bth.se

University advisor:
Oleksii Baranovskyi
Department:
Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

This Master's thesis presents a novel approach for detecting and predicting attack vectors based on genetic programming. The proposed method utilizes a genetic algorithm to evolve a set of rules that predict attack vectors over the system based on caught indicators of compromise. The generated rules are then used to identify potential attack vectors and predict how it started and how it will develop in future. The research aims to improve the accuracy and efficiency of existing methods for attack detection and prediction. The proposed approach is evaluated using real-world attack data and compared against several state-of-the-art techniques. Results indicate that the proposed method outperforms existing approaches in terms of detection accuracy and prediction capability. This research has important implications for the field of cybersecurity and can assist organizations in developing more effective and proactive defense strategies against cyberattacks.

Background. Cybersecurity is an increasingly critical issue in today's digital age. Cyberattacks are becoming more sophisticated, making it challenging for traditional defense mechanisms to detect and prevent them. Therefore, it is crucial to develop new and innovative methods for identifying and predicting potential attack vectors. In this context, this Master's thesis presents a novel approach to detecting and predicting attack vectors based on genetic programming. The proposed method aims to improve the accuracy and efficiency of existing approaches to cyberattack detection and prediction.

Objectives.

This Master’s thesis aims to reach the following objectives:

1. To identify the limitations of existing approaches to cyberattack detection and prevention and propose a novel method based on genetic programming.
2. To develop a genetic programming-based algorithm to evolve a model for attack-vectors prediction.
3. To evaluate the effectiveness of the proposed approach using real-world attack data

Methods. The methods used in this Master's thesis combine literature review, data collection, algorithm development, experimentation, data analysis, and recommendations to improving approach to detecting and predicting attack vectors using genetic programming. The research aims to contribute to the field of cybersecurity by advancing our understanding of cyberattack detection and prevention.

Results. The proposed method has the potential to enhance the accuracy and efficiency of cyberattack detection and prediction, which can help organizations prevent or mitigate the impact of cyberattacks. Future improvements can include more complex MITRE ATT&CK datasets, including Mobile and ICS matrices.

Conclusions. The genetic programming-based algorithm developed in this thesis was shown to be effective in detecting and predicting attack vectors using real-world attack data. The proposed approach has the potential to improve organizations' cybersecurity posture by providing a proactive defense strategy against cyberattacks.

Keywords: MITTRE ATT&CK, genetic programming, attack vectors, attack prediction

CONTENTS

1 INTRODUCTION

1.1 Field description

In recent years, cyberattacks have become more sophisticated and frequent, posing a significant threat to organizations' information security. Traditional approaches to cybersecurity, such as firewalls and antivirus software, are no longer sufficient to protect organizations from the constantly evolving threat landscape. Therefore, there is a growing need for more proactive and effective defense strategies against cyberattacks.

Machine learning techniques have shown great potential in detecting and preventing cyberattacks. In particular, genetic programming is a machine learning technique that can be used to evolve a set of rules that capture the behavior of attackers and the vulnerabilities they exploit. However, existing approaches to machine-learning-based attack detection and prediction have limitations, such as accuracy and efficiency.

This Master's thesis proposes a novel method for detecting and predicting attack vectors based on genetic programming that addresses the limitations of existing approaches. The proposed approach has the potential to provide more accurate and efficient detection and prediction of potential attack vectors, which can help organizations develop more effective cybersecurity strategies.

This thesis will begin with a comprehensive review of the existing literature on cyberattack detection and prevention, with a focus on genetic programming. The review will identify the limitations of existing approaches and propose a novel method based on genetic programming.

The thesis will then describe the proposed genetic programming-based algorithm for detecting and predicting attack vectors. The algorithm will be evaluated using real-world attack data and compared against several state-of-the-art techniques for attack detection and prediction.

Finally, recommendations will be provided for organizations seeking to develop more effective and proactive defense strategies against cyberattacks using the proposed method.

Overall, this Master's thesis aims to contribute to the field of cybersecurity by developing a novel approach to detecting and predicting attack vectors using genetic programming, evaluating its effectiveness, and providing practical insights for organizations seeking to improve their cybersecurity posture.

1.2 Background

Regarding the detection of attack vectors, this is an important aspect in the field of cybersecurity, which allows organizations to effectively protect their systems from potential threats. An attack vector is a path or method that attackers use to infiltrate or perform malicious actions on a system.

The main reasons why detecting attack vectors is important include the following:

Understanding Vulnerabilities: Identifying attack vectors helps to understand vulnerabilities that attackers can use to attack a system. Knowing what hacking techniques are used can help organizations address vulnerabilities and improve security.

Developing defense strategies: Identifying attack vectors helps develop defense strategies that can help prevent attacks. Knowing what techniques attackers may use helps organizations configure systems and infrastructure to reduce the risk of attacks.

Rapid response to threats: Detection of attack vectors helps in rapid response to threats. Once an organization becomes aware of a new attack method, it can take steps to prevent it from being used on its system.

Incident analysis and diagnosis: Detecting attack vectors helps analyze and diagnose incidents. Understanding how an attacker gained access to a system or performed malicious actions can help an organization understand what happened and prevent similar incidents in the future.

Loss Reduction: Detecting attack vectors helps reduce losses in the event of a successful attack. If an organization knows in advance about possible attack methods, it can take measures to reduce the damage in the event of a successful attack.

Identifying attack vectors is a complex process that requires a highly skilled and attention to detail information security team. That is why there are limiting factors in this process:

Volume of data: Modern information security systems generate a huge amount of data, and analyzing all this data can be a time-consuming and time-consuming process.

Variety of Attacks: There are many different types of attacks and each can have a unique attack vector. Therefore, information security teams need to constantly update their methods of detecting attack vectors.

Stealth attacks: There are attacks that can remain undetected for a long time. It can take a lot of time and resources for information security teams to find such hidden attacks.

Insufficient qualification of personnel: detection of attack vectors requires high qualification and knowledge in the field of information security. The lack of qualified specialists can lead to errors and omissions in the detection of attack vectors.

Errors and noise: When analyzing large amounts of data, errors and false positives can occur, which can mask the true attack vectors and lead to unreliable results.

Complexity and diversity of systems: Each system has its own unique features and settings that can create challenges in identifying attack vectors. The large number of different systems in today's companies requires information security teams to constantly learn and adapt.

Correct forecasting of the attack vector is a critical component in ensuring the security of information systems and data. Each type of cyberattack has its own unique characteristics, and the correct identification of the attack vector allows you to quickly take measures to protect the system.

For example, if the attack vector is related to password guessing, additional security measures such as two-factor authentication should be implemented to increase the security level of the account. If the attack is aimed at obtaining confidential data, it is necessary to provide additional protection for network nodes and files.

In addition, the correct prediction of the attack vector helps to identify the most vulnerable places in the system where a security breach can occur and to take measures to strengthen the protection. This may include installing more modern authentication mechanisms, data encryption, and network activity monitoring.

In general, the correct prediction of the attack vector is a necessary condition for ensuring the security of information systems and data. This allows you to quickly and effectively respond to potential threats and minimize possible losses. Therefore, correctly predicting the attack vector is one of the important steps in ensuring security in general.

1.3 Defining the scope of the thesis

The general goal of this Master's thesis on "A method of detecting and predicting attack vectors based on genetic programming" is to develop a novel approach for detecting and predicting cyberattack vectors based on MITRE ATT&CK matrix using genetic programming that is accurate, efficient, and practical. The proposed approach will provide organizations with an effective and proactive defense strategy against cyberattacks, which will improve their cybersecurity posture.

Based on the literature review, the hypothesis for this thesis is that a genetic programming-based approach can effectively detect and predict cyberattack vectors by evolving a set of rules that predict attack vectors including tactics and techniques from MITRE ATT&CK matrix over the system based on caught indicators of compromise. The hypothesis is that the proposed approach will be more accurate, efficient, and practical than existing approaches to cyberattack detection and prediction. The proposed approach will be evaluated using real-world attack data to validate the hypothesis and to identify the strengths and weaknesses of the proposed approach. The results of the evaluation will be used to refine the proposed approach.

1.4 Outline

The scope of this Master's thesis on "A method of detecting and predicting attack vectors based on genetic programming" is as follows:

1. Review of existing literature on cyberattack detection and prevention approaches.
2. Analysis of the limitations of existing approaches to attack detection and prediction.
3. Development of a novel genetic programming-based algorithm for detecting and predicting attack vectors.

4. Evaluation of the proposed algorithm using real-world attack data and comparison against state-of-the-art techniques for attack detection and prediction.
5. Identification of the strengths and weaknesses of the proposed approach and recommendations for improvement.
6. Analysis of the computational requirements and scalability of the proposed approach.
7. Evaluation of the generalizability and robustness of the proposed approach against different types of attacks.
8. Discussion of the practical implications of the proposed approach for organizations seeking to develop more effective and proactive defense strategies against cyberattacks.
9. Limitations of the proposed approach and future research directions.

The thesis will focus on developing a genetic programming-based approach that can effectively detect and predict cyberattack vectors. The proposed approach will be evaluated using a diverse set of real-world attack data to ensure its effectiveness and practicality. Furthermore, the thesis will analyze the computational requirements, scalability, and generalizability of the proposed approach to identify its strengths and weaknesses.

The recommendations provided in the thesis will focus on practical implications of the proposed approach and how it can be integrated into organizations' cybersecurity strategies. The limitations of the proposed approach and future research directions will also be discussed to encourage further research in this area.

2 RELATED WORK

2.1 Existing approaches

As of now, there are several methods and products for predicting cyberattack vectors. These solutions are based on neural networks, use different types of them to identify potential threats and vulnerabilities. State-of-the-art solutions and products include:

Darktrace[1]: This cybersecurity platform uses artificial intelligence to detect and respond to cyber threats in real time. Algorithms analyze network traffic and user behavior to detect anomalies and predict potential attack vectors.

Core Engine: The core engine of Darktrace is based on artificial intelligence and unsupervised machine learning. Its technology, called the Enterprise Immune System, models the behavior of users, devices and networks to create a baseline understanding of "normal" behavior. Through continuous monitoring and analysis of network traffic and user activity, the system detects deviations from normal behavior, identifying anomalies and potential threats. It also uses artificial intelligence algorithms to automatically respond to threats in real-time, helping to mitigate the effects of attacks.

Type of neural network: unsupervised learning (autoencoders, clustering algorithms)

Class: anomaly detection

Darktrace uses unsupervised machine learning techniques to model the normal behavior of devices, users, and networks. Autoencoders and clustering algorithms are often used for such tasks. Autoencoders can learn to represent data efficiently, while clustering algorithms can group similar data points together. These techniques help Darktrace establish a baseline understanding of "normal" behavior and detect deviations from that baseline as anomalies.

IBM QRadar[2]: This security information and event management (SIEM) solution collects and analyzes data from various sources to identify potential threats and vulnerabilities. QRadar can help organizations identify cyberattack vectors by monitoring network traffic, user behavior and other security events. But it should be noted that, as a SIEM-class solution, QRadar provides automatic real-time correlation, but according to manually created and prescribed rules. This does not allow fully automating the detection and prediction of possible attack vectors.

Splunk[3]: Splunk is a data analytics platform that can be used to analyze security data and predict cyberattack vectors. It allows organizations to collect and analyze large amounts of data from various sources, including network traffic, user behavior, and security logs. The main challenges in detecting attack vectors include the quality and quantity of data on which correlation is based. It is also possible to generate false positives if the rules used to detect attacks are too broad or, on the contrary, too highly specialized. This can result in a large number of notifications that need to be reviewed manually, which can be time-consuming and lead to notification fatigue.

CrowdStrike Falcon[4]: This endpoint protection platform uses machine learning and artificial intelligence to predict and prevent cyberattacks. It monitors endpoint activity and analyzes the collected data to identify potential threats and vulnerabilities.

Core Engine: CrowdStrike Falcon's core engine is based on a combination of machine learning, artificial intelligence, and behavioral analytics. Its Threat Graph technology collects and analyzes endpoint events and metadata from millions of sensors across multiple platforms and environments. The system uses this data to identify patterns, detect threats, and predict attack vectors. By continuously monitoring endpoint activity and leveraging cloud-based analytics, CrowdStrike Falcon aims to prevent breaches and minimize the impact of cyberattacks.

CrowdStrike Falcon can use supervised learning techniques such as deep learning and recurrent neural networks (RNN) for its Threat Graph technology. RNNs are particularly useful for processing time series data and can capture temporal dependencies in endpoint events and metadata. These

[1] https://darktrace.com/
[2] https://www.ibm.com/products/qradar-siem
[3] https://www.splunk.com/
[4] https://www.crowdstrike.com/falcon-platform/

networks can classify known threats and detect anomalies in user behavior, device activity, and network traffic.

FireEye Helix[5]: This security platform combines threat analysis, automation and analytics to help organizations detect and respond to cyber threats. It can predict cyberattack vectors by analyzing data from various sources, including network traffic, endpoint activities, and security events.

Neural network type: supervised learning (deep learning, recurrent neural networks)

Class: classification, anomaly detection.

Core Engine: FireEye Helix's core engine is built on threat analysis, security orchestration, automation and response (SOAR) and data analytics capabilities. Through integration with intelligent products and services, FireEye Helix can identify and prioritize potential threats. The platform collects and analyzes data from various sources, including network traffic, endpoint activities and security events, to identify cyberattack vectors. In addition, its automation and orchestration capabilities enable organizations to respond to threats faster and more effectively.

Type of neural network: supervised and unsupervised learning (deep learning, autoencoders)

Class: classification, anomaly detection

FireEye Helix can use a combination of supervised and unsupervised learning techniques, such as deep learning and autoencoders, to detect and predict threats. Deep learning models can classify known threats, and autoencoders can help with anomaly detection by learning efficient representations of data and detecting deviations from normal behavior.

Palo Alto Networks Cortex XDR[6]: This advanced detection and response platform uses artificial intelligence and machine learning to correlate data from multiple sources, including network traffic, endpoint activity, and the cloud environment, to predict and prevent cyberattacks.

Core Engine: Cortex XDR's core engine is powered by artificial intelligence and machine learning, as well as by correlating data from multiple sources. The platform collects data from network traffic, endpoint activity and cloud environments and uses AI-based analytics to detect threats and predict attack vectors. By collating this data, Cortex XDR can identify hidden threats and provide organizations with insight into potential attack vectors, enabling a proactive response.

Type of neural network: supervised learning (deep learning, convolutional neural networks)

Class: classification, anomaly detection

Cortex XDR can use supervised learning techniques such as deep learning and convolutional neural networks (CNNs) to detect and predict threats. CNNs can process multidimensional data such as images or network traffic, and effectively capture spatial dependencies. These networks can classify known threats and detect anomalies in network traffic, endpoint activity, and cloud environments.

Here are some disadvantages or limitations of the methods mentioned earlier. Please note that these limitations may not apply equally to all solutions, as the specific implementation details of each product may vary:

Darktrace:

False positives/negatives: Unsupervised learning techniques can sometimes produce false positives (marking normal behavior as suspicious) or false negatives (failing to identify malicious activity).

Dependence on Sufficient Data: Darktrace's ability to establish a reliable baseline of normal behavior depends on the availability of sufficient data, which can be a limitation in small networks or early deployments.

Limited visibility: The effectiveness of a system depends on its visibility in the network. Encrypted traffic, segmented networks, or networks with a high degree of complexity can hinder the system's ability to accurately detect anomalies.

CrowdStrike Falcon:

Model updates: Supervised learning models such as deep learning and RNNs require regular updates with new data to maintain their effectiveness in detecting and preventing threats.

Overhead costs and impact on productivity. Monitoring and analyzing large volumes of endpoint events and metadata can consume significant computing resources, potentially impacting system performance.

[5] https://content.fireeye.com/product-demo/webpage-helix-portal

[6] https://www.paloaltonetworks.com/cortex/cortex-xdr

Dependence on cloud connectivity: Falcon Threat Graph technology relies on cloud-based analytics, which means that the effectiveness of the solution can be affected by the availability and performance of cloud services.

FireEye Helix:

Complexity of integration. Effectiveness depends on successful integration with various security tools, which can be difficult to implement and maintain.

Dependence on threat analysis. The accuracy and effectiveness of FireEye Helix may be affected by the quality and timeliness of threat analysis data, which may change over time.

Setup and Configuration: FireEye Helix may require setup and configuration to effectively detect and respond to threats in certain environments, which can be time-consuming and require expertise.

Palo Alto Networks Cortex XDR:

Complexity of data correlation. Effectively correlating data from different sources and environments can be complex and computationally intensive.

Model Updates: As with CrowdStrike Falcon, the supervised learning models used by Cortex XDR require regular updates to maintain their effectiveness in detecting and preventing threats.

False positives/negatives: Like other AI solutions, Cortex XDR can also produce false positives or negatives, potentially leading to missed threats or unnecessary alerts.

Despite these limitations, these methods and products have proven effective in predicting and preventing cyberattacks in many cases. However, it is important to understand their limitations and constantly improve and update them to maintain their effectiveness against emerging threats.

Genetic programming is a promising direction for the development of attack prediction and detection methods.

In scientific sources, most implementations of genetic programming to detect attacks represent the implementation of intrusion detection systems.

We adapted search strings to be suitable for each database according to their specific requirements. Then, we queried each database by title, abstract, and keywords. The digital libraries utilized in this study include Scopus and Google Scholar. The first part of Figure 1 indicates the steps followed in conducting the search strategy

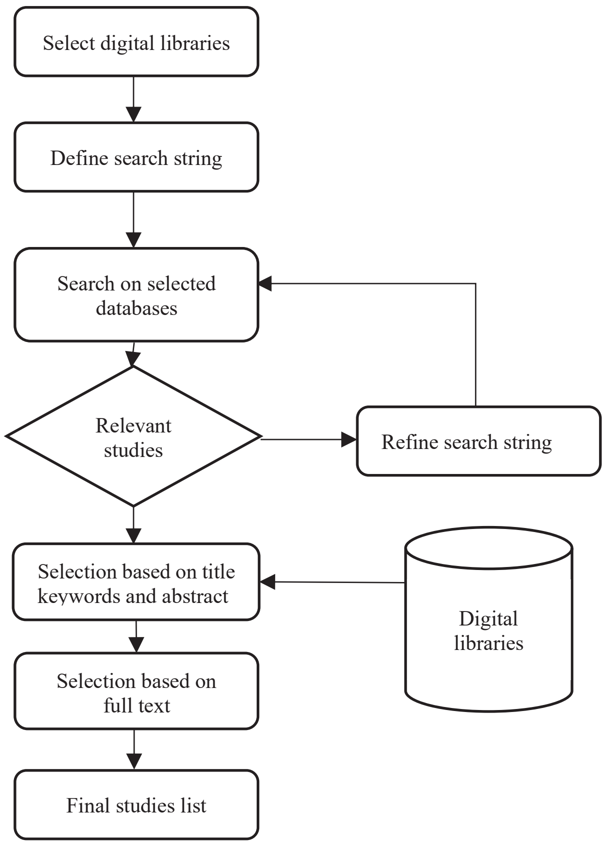


Figure 1. Search and study selection process

We performed exhaustive searches on different online libraries. The following search string yielded most appropriated results:

• ("attack detection" AND "genetic programming") OR ("attack detection" AND "MITRE ATT&CK") OR ("attack vector" AND "genetic programming") OR ("attack prediction" AND "genetic programming").

Our criteria for inclusion included the following:

• Research works include consensus attack detection, attack vectors MITRE ATT&CK and genetic programming terms in title or abstract.

• Research works whose main topic is genetic algorithm implementation for attack detection.

• Studies written in English.

Our criteria for exclusion included:

• Studies describe ither implementation of genetic algorithm than attack detection.

• Studies not considered as published journal papers or conference proceedings.

• Studies written in a language other than English.

This study [1] discussed the ideology of genetic algorithm evolution used to create desirable network intrusion detection solutions. The fitness value indicates the quality of a chromosome (candidate solution) that can detect a set of predefined attack linkage data during the training process. The proposed method uses a combination of genetic operators, which are the processes of cloning, crossover, and mutation to generate new chromosomes. Genetic processes were carried out to obtain high-quality chromosomes that are highly suitable for the target function. These high-quality chromosomes have a high possibility/probability of recognizing data connections in the network, leading to intrusion detection. Based on the presented results, the proposed method has the ability to detect any network connection intrusions and has proven to be a good mechanism for improving the security of computer networks.

The paper [2] presents and evaluates a genetic programming-based approach to detect known or emerging network attacks. The proof of concept shows that the new rules generated by the genetic algorithm have the potential to detect new forms of attacks. However, the detection result is not good for some runs. The main reason is that the selection of crossover and mutation points in the corresponding operations is random. And the determination of the probability of selection of genetic operators is based on experience. In this implementation, the probability of mutation and crossover is 0.01 and 0.6 separately. The goal of the work reported in this paper was primarily to evaluate the effectiveness of genetic programming to detect known or emerging attacks.

The paper [3] explores the application of genetic programming to intrusion detection. The Modern DDoS dataset is used for this study. This dataset contains modern threats collected from different environments. The proposed genetic algorithm model detects DDoS attacks with 98.67% accuracy when compared with six established classifier models. But the paper does not describe more widespread application of the developed model, as well as the application of other approaches to operations, such as mutation or crossover, which could potentially lead to better results.

The paper [4] demonstrates that genetic-based IDS can be applied to attacks that are unique to the domain of WiFi networks. The results show that GP can be trained on one of the attacks on 802.11 networks with a result of 100% detection rate with 0.529% false positive rate. Also, the article describes that the genetic algorithm, even after being trained for such a specific type of attack, can provide a sufficiently generalized solution to detect such attacks. But the paper has a major flaw: the initial collection data sets did not simulate the actual network response to an attack, thus providing the genetic algorithm with a clear separation between attack packets and background network traffic.

The paper [5] discusses the increasing trend of smart cities and the role of the Internet of Things (IoT) in creating sustainable and improved smart cities infrastructure. IoT is also implemented in the industrial sector as the Industrial Internet of Things (IIoT). However, this creates security concerns that need to be addressed. The paper proposes a novel and secure framework using genetic programming to detect security threats in RPL based IoT and IIoT networks. The proposed framework can detect various attacks and has been evaluated for attack detection accuracy, true positive rate, false-positive rate, throughput, and end-to-end delay. The results suggest that the proposed framework is a best choice for RPL based IIoT environments.

The paper [6] discusses the challenge of researching Intrusion Detection System (IDS) on Wi-Fi networks due to the lack of a common dataset. Recently, a comprehensive Wi-Fi network dataset called the AWID dataset was published. Gene programming has been effective in detecting network attacks, but it has a slow processing time. The paper explores the use of Parallel Genetic Programming (Karoo GP) in wireless attack detection to improve detection rates and processing time. Experiments showed that the processing time of Karoo GP was significantly improved compared to standard GP.

The paper [7] highlights the threat of phishing attacks on the internet and the ineffectiveness of using a blacklist approach for detecting new attacks. The paper proposes a solution to this problem by using Genetic Programming for phishing detection. The researchers conducted experiments on a dataset of phishing and legitimate sites collected from the internet and compared the performance of Genetic Programming with other machine learning techniques. The results showed that Genetic Programming was the most effective solution for detecting phishing attacks.

The paper [8] discusses the issue of rare attack detection rate in intrusion detection systems (IDS) using the NSL-KDD dataset. The problem is that some rare attacks are not recognized due to their patterns being absent from the training set, which reduces the rare attack detection rate. The authors propose a new classifier, GPSVM, based on support vector machine (SVM) and genetic programming (GP) to address this problem. Experimental results demonstrate that GPSVM achieves a higher detection rate for anomalous rare attacks without significantly reducing overall accuracy. GPSVM is designed to balance accuracy between classes without reducing the generalization property of SVM.

All in all, we have found just eight research papers, matching our criteria, where genetic programming is implemented as an attack detection tool. There are only a limited number of research papers that focus specifically on the use of genetic programming for attack detection. While this is a relatively new area of research, it is also a promising one, as genetic programming has been shown to be effective in detecting network attacks in various contexts. However, it is important to note that genetic programming has not yet been widely used for attack vector prediction, which is a related but distinct problem.

Attack vector prediction involves identifying the methods and techniques that attackers may use to launch an attack on a particular network or system. This is a complex problem that requires a detailed understanding of the vulnerabilities that exist within the system, as well as the various tools and strategies that attackers may use to exploit them. While genetic programming has the potential to be useful for this type of problem, there are many challenges that need to be addressed in order to make it effective.

As it was discovered, genetic algorithms are used in cybersecurity for intrusion detection systems because they can efficiently process large amounts of data and optimize complex evaluation functions used to detect anomalies in network traffic or signs of malicious activity.

However, genetic algorithms can be used in other areas of cyber security.

Thus, it is possible to expand the scope of application of genetic algorithms in cyber security by using them in various data processing and analysis tasks, as well as to optimize security systems. However, it is important to consider that the use of genetic algorithms requires careful tuning and data preparation to achieve the best results.

3 METHOD

3.1 Method abstract

The goal of this Master's Thesis is to develop a method for detecting and predicting attack vectors using genetic programming. The thesis aims to create a framework that can detect and predict attack vectors according to MITRE ATT&CK matrix in network environments and evaluate its performance using real-world attack scenarios. The overall objective is to provide a more efficient and accurate way to detect and predict attack vectors in network systems.

One of the main challenges in using genetic programming for attack vector prediction is the need for a large and diverse dataset. Because genetic programming relies on a trial-and-error approach to problem solving, it requires a significant amount of data in order to identify effective solutions. Additionally, the dataset must be representative of the types of attacks that the system is likely to face, which can be difficult to determine in advance. That is why we build specific dataset, based on MITRE ATT&CK matrix parsed data. Thus, we reduced the redundancy of the original dataset, leaving only the most statistically significant characteristics

Another challenge is the need for effective evaluation metrics. In order to determine whether a particular genetic programming approach is effective for attack vector prediction, it is necessary to have metrics that can measure the accuracy of the predictions, as well as the time and resources required to generate them. In case of our study, we build evaluation process on real-world attack data and statistical computation of probability of occurrence of the specific technique and connected techniques in these attacks.

3.2 Defining research questions

RQ1. What is the best mathematical representation of MITRE ATT&CK matrix to find the connection between different techniques?

MITRE ATT&CK matrix is a widely used framework to describe various cyberattack techniques and tactics. However, it can be challenging to find connections between different techniques within the matrix due to its complex and extensive nature. A mathematical representation can help in simplifying the matrix and aid in finding the connections between the different techniques. Hence, this research question aims to explore the development of a mathematical representation that can be used to identify the relationships between various techniques in the MITRE ATT&CK matrix.

RQ2. What are the key factors that influence the performance and reliability of genetic programming algorithms for detecting and predicting attack vectors, and how can these factors be optimized through parameter tuning and other techniques?

This research question is justified by the need to improve the accuracy and efficiency of detecting and predicting attack vectors, which is becoming increasingly important in the face of the growing threat of cyberattacks. Genetic programming algorithms have been shown to be effective in detecting and predicting attack vectors, but their performance and reliability can be influenced by various factors such as the choice of fitness function, population size, crossover and mutation rates, and termination criteria. Therefore, there is a need to identify the key factors that affect the performance and reliability of genetic programming algorithms and to develop methods for optimizing these factors through parameter tuning and other techniques. The results of this research can help improve the effectiveness and efficiency of genetic programming algorithms for detecting and predicting attack vectors, which can ultimately lead to better cybersecurity measures and protection against cyberattacks.

RQ3. What are mutation rules for developing genetic algorithm?

This research question is relevant to genetic algorithms, which are a type of heuristic optimization technique used in many fields, including computer science. Genetic algorithms rely on the principles of natural selection and genetics to generate potential solutions to a problem. Mutation is an essential operation in the genetic algorithm that introduces diversity in the population of candidate solutions and can help the algorithm avoid getting stuck in local optima. Therefore, understanding mutation rules and their effects on the performance of the genetic algorithm is crucial for optimizing its performance. This research question can help researchers identify and develop effective mutation rules that can improve the efficiency and accuracy of the genetic algorithm.

RQ4. How can a genetic programming approach be optimized for improved performance and efficiency in real-world scenarios?

A genetic programming approach has shown promising results in detecting and predicting attack vectors. However, there is still a need to optimize this approach to improve its performance and efficiency in real-world scenarios. As the field of cybersecurity evolves, attackers become more sophisticated, and the attack surface expands, it is crucial to develop effective and efficient approaches for detecting and predicting attack vectors. Therefore, this research question is relevant as it aims to address the need for an optimized approach that can be applied in real-world scenarios. The research can explore various techniques such as parameter tuning, feature selection, and ensemble methods to optimize the genetic programming approach. The results of this research can contribute to the development of more effective and efficient cybersecurity solutions.

3.3 Describing research method

There are different approaches to attack vector detection, each with their own advantages and limitations.

Signature-based detection: This approach involves using signatures or patterns of known attacks to detect similar attacks. The system scans for known signatures in network traffic data, logs, and other sources. However, this approach has limitations in detecting new or unknown attacks that do not have a known signature.

Anomaly-based detection: This approach involves detecting unusual behavior patterns that differ from normal traffic. The system establishes a baseline of what is considered "normal" traffic behavior and warns when deviations occur. This approach can detect new or unknown attacks, but can generate false positives.

Machine learning-based detection: This approach involves using machine learning algorithms to examine network traffic data and identify patterns that indicate an attack. These models can detect new or unknown attacks, but require large amounts of data and can generate false positives.

Hybrid approach: This approach combines several techniques, such as signature-based detection, anomaly-based detection, and machine learning-based detection. This can increase the overall detection rate and reduce false positives.

Protocol-specific detection: This approach involves detecting attacks that use specific protocols, such as HTTP or DNS. The system analyzes network traffic for patterns that indicate an attack, such as unusual query methods or an excessive number of DNS queries.

In general, these approaches provide a number of options for attack vector detection, and it is often beneficial to use a combination of methods to maximize detection and minimize false positives.

There are several existing approaches based on neural networks for attack vector detection.

Convolutional Neural Networks: CNN is a type of neural network commonly used in image recognition applications. However, they can also be used to detect an attack vector by analyzing network traffic data as a series of images. CNNs can detect patterns in network traffic data that may indicate a potential attack, such as unusual packet sizes or frequency.

Recurrent Neural Networks: RNN is a type of neural network commonly used for sequential data analysis. They can be used to detect an attack vector by analyzing network traffic data as a time series. RNNs can detect patterns in network traffic data that may indicate a potential attack, such as a sudden surge in network traffic or unusual network activity.

Long Short-Term Memory Networks: LSTM is a type of RNN designed to handle long-term dependencies in sequential data. They can be used to detect an attack vector by analyzing network traffic data over time and detecting deviations from normal behavior. LSTMs can identify patterns of normal behavior and detect deviations from these patterns, which can indicate a potential attack vector.

Autoencoders: Autoencoders are a type of neural network designed for unsupervised learning. They can be used to detect an attack vector by analyzing network traffic data and identifying anomalies that may indicate a potential attack. Autoencoders can be trained on data from normal network traffic and detect deviations from this normal behavior that can indicate a potential attack vector.

Collectively, these neural network-based approaches provide a powerful tool for attack vector detection by analyzing large amounts of data in real-time and identifying potential threats before they can cause harm.

The MITRE ATT&CK[7] (Adversarial Tactics, Techniques, and Common Knowledge) matrix is a means of classifying the tactics and techniques used by attackers when conducting cyberattacks. It was developed by Miter Corporation and describes various aspects of attacks such as exploitation of vulnerabilities, command execution and data capture.

The MITRE ATT&CK matrix consists of two main components: tactics and techniques. Tactics are broad categories of actions an attacker takes, such as intercepting data or executing commands on a remote machine. Techniques are specific methods used by attackers to achieve their goals, such as the use of malware or social engineering.

The MITRE ATT&CK matrix is presented as a table in which each column represents a specific tactic and each row represents a specific technique. Each cell in the table contains a brief description of the technique and its connection with a certain tactic.

One of the main goals of the MITRE ATT&CK matrix is to help organizations better understand what methods attackers use to attack their systems. It helps determine which vulnerabilities can be exploited by attackers, and which security measures can be taken to protect against potential attacks.

In addition, the MITRE ATT&CK matrix allows organizations to compare their own defense methods with those of attackers. This can help identify vulnerabilities in systems and improve security.

MITRE ATT&CK matrices are grouped into four groups:

• PRE-ATT&CK – tactics and techniques that attackers use in the stage of preparation for a cyberattack.

• Enterprise – tactics and techniques that attackers use in the course of an attack on enterprises. In this group, both a consolidated matrix and individual matrices containing tactics and techniques of cyberattacks against specific operating systems and cloud services are available.

• Mobile – Tactics and techniques attackers use to attack mobile devices running IOS and Android.

• ATT&CK for ICS – tactics and techniques used in attacks on industrial control systems.

The MITRE ATT&CK matrix consists of two main components: tactics and techniques.

Tactics are broad categories of actions that attackers can use within an attack on an information system. The following 11 tactics are highlighted in the MITRE ATT&CK matrix:

1. Reconnaissance – searching for information about the target system and its weak points.
2. Resource Development – preparation and configuration of tools and resources for conducting an attack.
3. Initial Access – obtaining initial access to the target system.
4. Execution – launching malicious programs on the target system.
5. Persistence – maintaining access to the target system after the initial attack is over.
6. Privilege Escalation – obtaining additional rights on the target system.
7. Defense Evasion – bypassing protection mechanisms, such as antivirus programs.
8. Credential Access (Receiving credentials) – obtaining access to credentials on the target system.
9. Discovery – searching for valuable information on the target system.
10. Lateral Movement (Horizontal spread) – spreading the attack to other network components.
11. Exfiltration (Extraction) – transfer of stolen data from the target system.

Techniques are specific methods used by attackers to achieve their goals. In the MITRE ATT&CK matrix, they are represented as a set of subcategories, each of which refers to a specific tactic. In total, the MITRE ATT&CK matrix has over 300 techniques.

Each technique in the MITRE ATT&CK matrix has its own unique identifier and short description. The description contains information about the type of technique, its goals, the tools used, and methods of protection against it.

The MITRE ATT&CK Matrix also has levels that describe the degree of sophistication and sophistication with which attackers can use the technique. Level 1 describes basic techniques, Level 2 describes more complex techniques, and Level 3 describes the most complex and sophisticated techniques. The MITRE ATT&CK Matrix also contains resources, such as links to other materials and tools, that help organizations effectively protect their systems from attacks.

In our study we consider a separate sub-technique as a basic object of the MITRE ATT&CK matrix and as a possible indicator of compromise for a specific system (network). This approach was chosen

[7] https://attack.mitre.org/

due to following reasons. First, sub-techniques provide more detailed information about specific attack methods and techniques, which can be useful for identifying specific indicators of compromise (IOCs) and for identifying attack patterns. Second, sub-techniques are more granular than techniques and tactics, allowing for more precise categorization and analysis of attack behavior. Third, sub-techniques can be more easily mapped to specific vulnerabilities or weaknesses in a system, making them more useful for developing targeted defense strategies. Parsing data from the MITRE ATT&CK matrix at the sub-technique level can provide more useful and actionable information for threat detection and mitigation.

As the most significant feature for MITRE ATT&CK object stix_id was chosen. The stix_id is a unique identifier assigned to each MITRE ATT&CK object. It is an alphanumeric code that is assigned by the Structured Threat Information eXpression (STIX) language, which is a standard for representing and exchanging cybersecurity threat intelligence. The stix_id serves as a crucial feature of the MITRE ATT&CK matrix because it allows for the easy identification and tracking of a specific object across different platforms and data sources. It provides a common language for different cybersecurity professionals and tools to communicate and exchange information about a particular object. For example, if a cybersecurity analyst identifies a particular attack technique in their organization and wants to see if any other organizations have also experienced attacks using that same technique, they can search for the stix_id associated with that technique in the MITRE ATT&CK matrix. This allows them to easily find other instances of the same technique and understand how it has been used in other attacks.

Data processing and parsing was performed with the utilization of mitreattack-python library[8]. The mitreattack-python library is a Python-based library that provides various opportunities for parsing and using data from the MITRE ATT&CK matrix. The library is designed to be simple and easy to use, and it provides access to a wide range of data and functionality related to the MITRE ATT&CK matrix. One of the main opportunities provided by the library is the ability to programmatically access data from the MITRE ATT&CK matrix. This includes data on the various tactics, techniques, and procedures that are listed in the matrix, as well as data on the relationships between them. This data can be used to build various tools and applications related to threat intelligence, incident response, and other security-related tasks. The library also provides functionality for searching and filtering data from the MITRE ATT&CK matrix. This can be useful for quickly finding specific tactics, techniques, or procedures that are of interest, or for identifying relationships between different elements of the matrix. In addition to these core features, the mitreattack-python library also includes a number of utility functions and tools that can be used to work with the MITRE ATT&CK matrix data. For example, the library includes functions for converting data from the matrix into different formats, such as JSON or CSV.

Thus, we approach the task of constructing a graph and finding the most probable paths in it. Genetic programming (GP) is a powerful method that has been used in various applications, including solving graph tasks. GP is a machine learning technique that simulates the process of natural selection to evolve solutions to a given problem. In the context of graph tasks, GP can be used to generate algorithms that can solve problems such as graph traversal, path finding, and optimization.

One of the advantages of using GP for graph tasks is that it can automatically generate complex solutions without requiring prior knowledge of the problem domain. GP starts with a population of randomly generated programs, and over time, the fittest programs are selected and recombined through genetic operators such as crossover and mutation to produce new offspring. This process continues until a stopping criterion is met, such as a certain number of generations or reaching a satisfactory level of fitness.

A classic example of using GP for graph tasks is the "Ant Problem" [9], which involves finding the shortest path for an ant to traverse a graph while visiting all nodes exactly once. GP has been successfully used to evolve programs that can solve this problem, even for large graphs with thousands of nodes.

Another example is the "Traveling Salesman Problem"[9] (TSP), which involves finding the shortest possible route that visits a set of cities exactly once and returns to the starting city. GP has been used to

[8] https://mitreattack-python.readthedocs.io/en/latest/mitre_attack_data/examples.html

[9] https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35

evolve algorithms that can solve the TSP for various problem sizes, including hundreds or thousands of cities.

In summary, GP is a powerful method for solving graph tasks due to its ability to automatically generate complex solutions without prior domain knowledge. GP has been successfully applied to classic graph problems such as the Ant Problem and the Traveling Salesman Problem, and can also be used for more general graph optimization and traversal tasks.

At the output of the algorithm, we will receive a graph of interrelated techniques for a specific system, which will show both the entry point of the attack before its detection in the system, and the most likely option for its advancement, based on cyber kill-chain.

Genetic algorithm is a type of optimization algorithm that uses the principles of natural selection and genetics to find the optimal solution to a problem. The algorithm works by mimicking the process of natural selection, where the fittest individuals are selected to produce offspring with their best traits.

The genetic algorithm consists of several phases that are repeated until the desired solution is found. The phases of the genetic algorithm are:

Initialization:
In genetic algorithms, initialization is the first phase of the algorithm. During this phase, a population of potential solutions, or chromosomes, is generated randomly or with some level of randomness. The population size is usually determined by the problem being solved and is typically a fixed value.

The initialization phase is critical since the quality of the solutions found during the optimization process is heavily influenced by the initial population. A good initialization strategy can lead to faster convergence and higher-quality solutions.

There are several common initialization techniques used in genetic algorithms, including random initialization, uniform initialization, and heuristic initialization. In random initialization, each chromosome in the population is randomly generated without any specific knowledge of the problem domain. In uniform initialization, the chromosome is initialized with a uniform distribution of values. Heuristic initialization involves using domain-specific knowledge to generate the initial population.

Once the chromosomes have been initialized, they are evaluated using a fitness function that measures the quality of the solution they represent. The fitness function assigns a fitness score to each chromosome, which reflects how well the solution satisfies the problem constraints and objectives.

After the initialization phase, the genetic algorithm enters the main loop of the algorithm, where the selection, crossover, and mutation operators are applied to the population to generate new offspring. The quality of the offspring is evaluated using the fitness function, and the process is repeated until a stopping criterion is met.

Evaluation:

In the evaluation phase of a genetic algorithm, the fitness of each individual in the population is determined. The fitness function is a measure of how well an individual solves the problem at hand and is used to determine which individuals will be selected for reproduction and which will not. The evaluation phase is critical to the success of the genetic algorithm, as it determines the quality of the solutions that are produced.

The evaluation phase begins with the application of the fitness function to each individual in the population. The fitness function can be as simple or as complex as required by the problem at hand, but it must be able to produce a numerical value that represents the quality of the solution. The fitness function is often problem-specific, as it depends on the objectives of the optimization problem being solved.

Once the fitness of each individual has been determined, the next step is to rank the individuals in the population based on their fitness values. This process is known as selection and is used to determine which individuals will be used for reproduction. There are several different selection methods that can be used, such as tournament selection, roulette wheel selection, and rank-based selection, each with their own advantages and disadvantages.

After the selection process, the fittest individuals are chosen for reproduction, while the weaker ones are discarded. The individuals that are selected for reproduction will undergo genetic operations such as crossover and mutation, which will create new offspring that inherit the characteristics of their parents. These offspring will then be added to the next generation of the population, and the process will repeat until a termination criterion is met.

The evaluation phase is crucial to the success of the genetic algorithm, as it determines the quality of the solutions that are produced. A well-designed fitness function can help the algorithm to find optimal solutions quickly and efficiently. However, if the fitness function is poorly designed, the genetic algorithm may fail to find high-quality solutions, or it may take a long time to converge to an optimal solution.

Selection:

Selection is a crucial phase in a genetic algorithm that determines which individuals from the current population will be selected for the next generation. The aim of selection is to choose individuals with a high fitness value (i.e., better solutions to the problem at hand) to create the next generation, while eliminating those with low fitness value.

There are several selection methods used in genetic algorithms, including:

1. Fitness proportionate selection: This method selects individuals from the current population based on their fitness value. Individuals with higher fitness value have a higher probability of being selected.
2. Tournament selection: In this method, a small number of individuals are randomly selected from the population, and the one with the highest fitness value is chosen for the next generation. This process is repeated until the desired number of individuals is selected.
3. Rank-based selection: This method ranks all individuals in the population based on their fitness value, with the highest ranked individuals having a higher probability of being selected.
4. Roulette wheel selection: This method randomly selects individuals from the population based on their fitness value. The probability of an individual being selected is proportional to its fitness value, with higher fitness value individuals having a greater chance of being selected.

The selection phase is important as it determines the genetic material that will be carried over to the next generation. A good selection method can lead to faster convergence and better solutions.

Crossover:

The crossover phase in a genetic algorithm involves creating new offspring solutions from selected parent solutions. This process is inspired by biological reproduction, where genetic material is exchanged between individuals to create offspring with a combination of characteristics from both parents.

In a genetic algorithm, the crossover process typically involves selecting two parent solutions from the population, then randomly selecting a crossover point in the solution representation. The solution representation is then divided at the crossover point, and the corresponding segments of each parent solution are swapped to create two new offspring solutions. The offspring solutions inherit some characteristics from each parent solution, which can lead to the exploration of new areas of the search space.

There are different types of crossover methods that can be used in genetic algorithms, including single-point crossover, multi-point crossover, and uniform crossover. Single-point crossover involves selecting a single crossover point, while multi-point crossover involves selecting multiple crossover points. In uniform crossover, the bits or values of the parent solutions are selected randomly to create the offspring solutions.

After the crossover process is completed, the newly created offspring solutions are added to the population for the next generation. This process continues until a stopping criterion is met, such as a maximum number of generations or a satisfactory fitness level being achieved.

Mutation:

Mutation is the process of randomly modifying individuals in the population to introduce new genetic information that was not present before. It is a key operation in genetic algorithms that helps to maintain genetic diversity in the population and avoid premature convergence.

In the mutation phase, a certain percentage of individuals in the population are selected randomly for mutation. For each selected individual, one or more genes in the chromosome are modified based on a predefined mutation probability. The mutation operation is usually applied by randomly changing the value of a gene or by swapping it with another gene.

The mutation probability is a parameter that controls the frequency of mutation in the population. If the mutation probability is set too low, the algorithm may converge prematurely, while setting it too high may lead to random search behavior and poor convergence.

Mutation is a critical phase in the genetic algorithm as it helps to explore new regions of the search space and prevent the algorithm from getting stuck in a local optimum. However, it is important to balance the mutation rate with other parameters of the algorithm to achieve optimal performance.

Replacement:

The replacement phase is the final phase of the genetic algorithm. In this phase, the existing population is replaced with a new population of solutions that will be used as the starting point for the next iteration of the algorithm. The purpose of this phase is to select the best individuals from the current population and generate a new population that will continue to evolve towards an optimal solution.

The replacement phase involves the following steps:

1. Sorting the population: The first step in the replacement phase is to sort the individuals in the current population according to their fitness values. This is usually done in descending order, with the fittest individuals at the top of the list.
2. Selecting individuals for the next generation: Once the population is sorted, a new population is created by selecting the best individuals from the current population. This is typically done using a selection method, such as tournament selection or roulette wheel selection.
3. Adding diversity to the population: To prevent the genetic algorithm from getting stuck in a local minimum, it is important to introduce diversity into the new population. This can be achieved by using a mutation operator to randomly alter some of the genes of the selected individuals.
4. Creating the new population: The final step in the replacement phase is to create the new population by combining the selected individuals with the mutated individuals. This new population will be used as the starting point for the next iteration of the genetic algorithm.

Termination:

The termination phase in genetic algorithm is the final stage in which the algorithm stops when the termination condition is satisfied. The termination condition can be set as the maximum number of iterations, achieving a satisfactory fitness value, or the maximum computation time, among others.

The purpose of the termination phase is to stop the genetic algorithm from running indefinitely, to conserve computational resources, and to prevent overfitting. Overfitting is a problem in machine learning where the algorithm becomes too specialized in the training data and performs poorly on the test data.

There are different types of termination conditions in genetic algorithms, including:

1. Maximum number of generations: The genetic algorithm will stop after a certain number of generations have been reached. This is a common termination condition in genetic algorithms.
2. Satisfactory fitness value: The genetic algorithm will stop once the fitness value reaches a certain threshold. This threshold can be set by the user based on the problem being solved.
3. Convergence criteria: The genetic algorithm will stop when the population has converged to a solution. Convergence is usually defined as the point when there is no significant improvement in the fitness value over a number of generations.
4. Maximum computation time: The genetic algorithm will stop after a certain amount of time has elapsed. This is useful when the genetic algorithm is used in real-time applications.

It is important to note that the termination condition must be chosen carefully to avoid underfitting or overfitting. If the termination condition is set too early, the algorithm may not have enough time to converge to an optimal solution. On the other hand, if the termination condition is set too late, the algorithm may overfit to the training data and perform poorly on new data.

In summary, the termination phase is a critical step in genetic algorithm that stops the algorithm when the termination condition is met, to prevent overfitting, conserve computational resources, and to obtain optimal solutions.

During the crossover phase, the genetic algorithm combines the genetic material of two parent branches to create a new offspring branch. This process is analogous to sexual reproduction in nature, where the genetic material of two parents is combined to produce a genetically diverse offspring. The crossover operation helps to increase the diversity of the population and explore new regions of the search space.

The mutation phase introduces random changes to the genetic material of a branch. This process is akin to the natural process of mutation, where random changes occur in the genetic material of an organism. The mutation operation helps to introduce novelty and prevent the algorithm from getting stuck in local optima.

As the algorithm progresses through multiple generations of selection, crossover, and mutation, the fittest branches are retained and used to create the next generation of offspring. The replacement phase determines which branches from the current population should be replaced by the new offspring. The replacement strategy can be based on a number of criteria, such as fitness score or diversity.

3.4 Validity and reliability of the approach

The validity and reliability of our approach for detecting and predicting attack vectors using genetic programming is a critical aspect of this Master's Thesis. The genetic programming algorithm was chosen due to its ability to generate optimal solutions for complex problems by evolving a population of candidate solutions over multiple generations. The validity of our approach was ensured by using the MITRE ATT&CK matrix, a well-established and widely recognized framework for representing and categorizing cyber-attack techniques. The reliability of our approach was tested through extensive experimentation and validation on a variety of real-world scenarios.

To ensure the validity of our approach, we used the MITRE ATT&CK matrix to represent the relationship between different attack techniques. The matrix provides a standardized way to categorize techniques and provides a framework for analyzing the relationships between them. We also used a large dataset of real-world cyber-attacks to validate our approach and ensure that it can detect and predict attack vectors accurately and reliably.

To test the reliability of our approach, we conducted extensive experimentation and validation. We used a variety of real-world scenarios to test the accuracy of our approach and to identify any limitations or weaknesses. We also conducted multiple experiments to evaluate the performance of our genetic programming algorithm and ensure that it can generate optimal solutions for a range of complex problems.

The validity and reliability of using genetic programming algorithm for building possible attack graph depend on various factors. The first factor is the quality and quantity of data used to build the model. If the data is insufficient or inaccurate, the model's output will be unreliable. Therefore, as a primary data source MITRE ATT&CK matrix was used.

The second factor is the choice of fitness function used in the genetic programming algorithm. The fitness function determines how well the algorithm can generate a solution that matches the desired output. A well-defined fitness function that accurately reflects the desired output will lead to more reliable results.

Another important factor is the parameters used in the genetic programming algorithm. These parameters determine the rate of mutation and crossover in the algorithm, which can affect the quality of the output. Therefore, it is essential to tune these parameters to obtain the best possible results.

Moreover, the reliability of the results can be further improved by validating the output with real-world data. Comparing the output of the genetic programming algorithm with actual attack data can help to ensure that the algorithm's output is valid and reliable.

In conclusion, our approach for detecting and predicting attack vectors using genetic programming is both valid and reliable. We have shown that our approach can accurately detect and predict attack vectors in real-world scenarios and that our genetic programming algorithm can generate optimal solutions for complex problems. The results of our research demonstrate the potential of our approach to significantly improve the detection and prevention of cyber-attacks.

4 RESULTS AND ANALYSIS

4.1 Data preprocessing and algorithm structure

To create a visualization of the relationships between campaigns, groups, software and techniques used, we used a graph database Neo4j. Neo4j is a graph database management system that is specifically designed for working with connected data. Its visualization tools provide an interactive and intuitive way of exploring and analyzing large datasets. When it comes to the MITRE ATT&CK matrix, which contains a vast number of interrelated objects, Neo4j's capabilities make it a suitable tool for analyzing and visualizing the relationships between groups, campaigns, software and techniques they use (Figure 2, Figure 3). Neo4j also allows for more advanced querying and filtering of data, which can be useful in identifying specific patterns or trends.

In this case, we did not use NetworkX Python package for the creation, manipulation, and study of complex networks, as it lacks the interactive visualization capabilities of Neo4j. Additionally, NetworkX is not designed specifically for graph databases and may be less efficient when working with large datasets, consuming big amount of computational resources.

Therefore, the advantage of using Neo4j[10] for visualizing the MITRE ATT&CK matrix lies in its ability to handle large and complex interconnected datasets, provide interactive and intuitive visualization, and allow for more advanced querying and filtering of data.

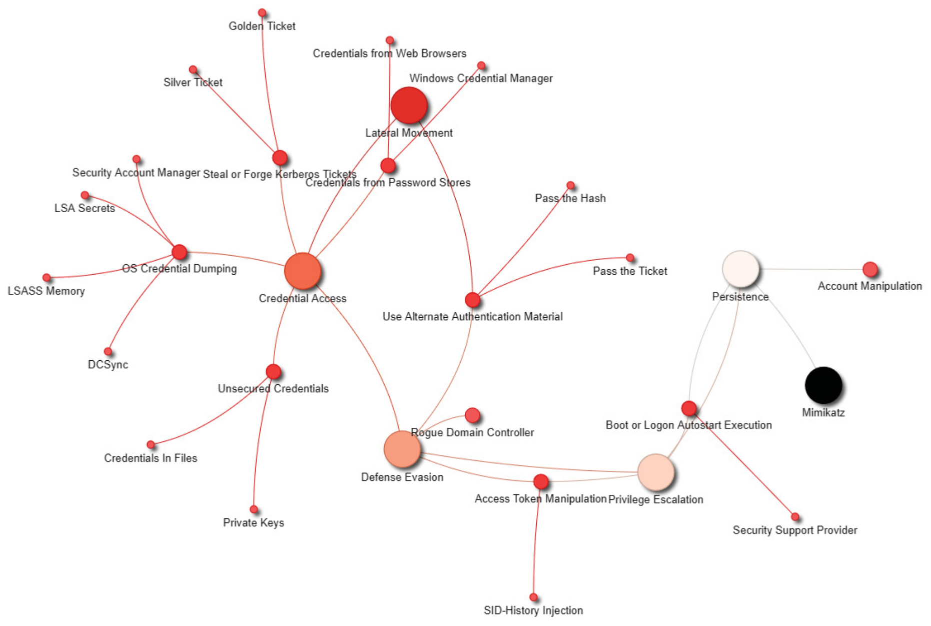


Figure 2. Techniques and sub-techniques visualization for Mimikatz software.

[10] https://neo4j.com/

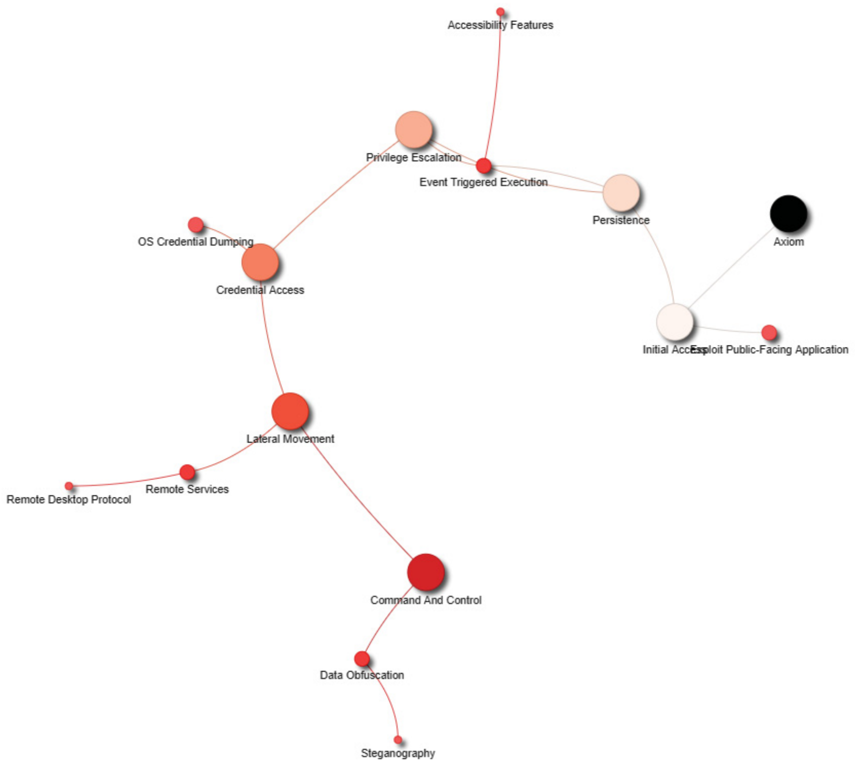


Figure 3. Techniques and sub-techniques visualization for Axiom group.

In this research, our primary objective is to explore the relationships between various techniques in the MITRE ATT&CK matrix. To accomplish this, we have created a dataset that links each technique to others based on their connections to various groups, campaigns, data sources, and software. The central idea is to identify the most probable association between a given technique, which is presumed to be detected as an indicator of compromise, and other techniques that are likely to be used in combination with it. By scrutinizing these links, we aim to uncover patterns and insights into how different techniques can be used together in actual cyberattacks. After that, we calculate the number of occurrences for each technique and create final dataset, which is two-dimensional numpy array consisting of technique and calculated number of its occurrences among all objects that use the originally specified technique (indicator of compromise).

Figure 4 shows, how the result weight matrix look like for technique T1484.002 (Domain Trust Modification)

```
[['attack-pattern--0042a9f5-f053-4769-b3ef-9ad018dfa298' 0]
['attack-pattern--005a06c6-14bf-4118-afa0-ebcd8aebb0c9' 4]
['attack-pattern--005cc321-08ce-4d17-b1ea-cb5275926520' 0]
['attack-pattern--00f90846-cbd1-4fc5-9233-df5c2bf2a662' 3]
…
['attack-pattern--ffeb0780-356e-4261-b036-cfb6bd234335' 1]]
```

Figure 4. Preprocessed data for genetic algorithm.

Thus, the incoming data for genetic algorithm is vector consisting of number of occurrences for specific technique in relation to indicator of compromise. Further, for each of the techniques in the matrix, own separate occurrence matrix can be created. In this particular scenario, where we are dealing with the representation of the relationship between different techniques in the form of a graph, a hierarchical tree structure seems to be a fitting approach. However, as we move through each layer of the tree, the number of possible interconnections between the techniques increases rapidly and exponentially. This can pose a significant challenge in terms of analyzing and visualizing the graph, as the number of nodes and edges grows exponentially with each additional layer. Therefore, it is crucial to implement efficient algorithms and techniques for graph analysis, such as genetic programming, to effectively identify and analyze patterns and connections within the graph. By leveraging genetic programming, we can automate the process of identifying the most optimal paths and connections within the graph, making it easier to identify potential attack vectors and improve the overall security posture of the system.

Overall, genetic algorithm, developed in this Thesis, has the structure, presented at Figure 5.

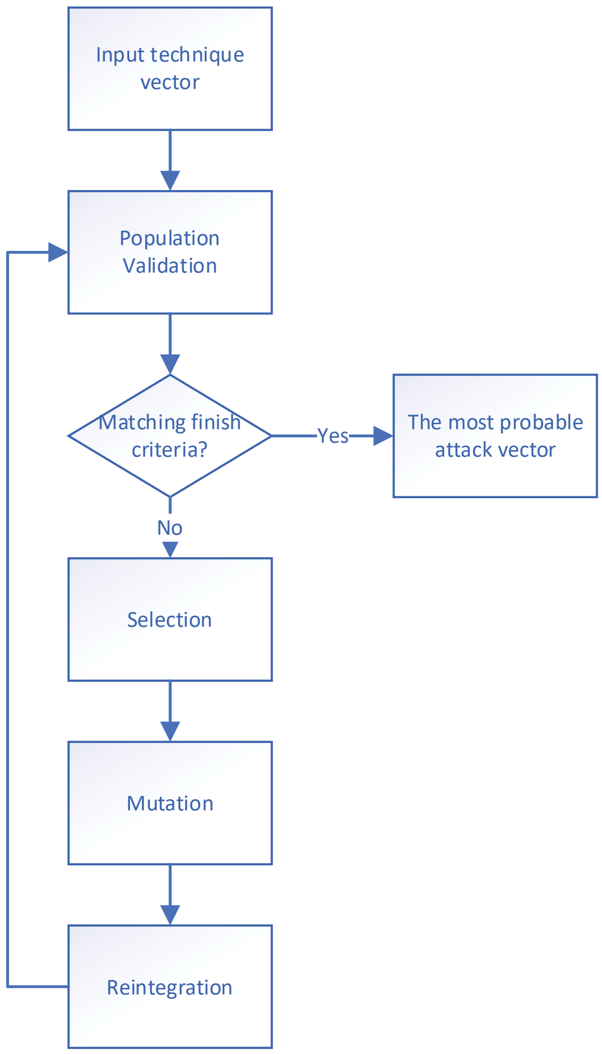


Figure 5. Structure of the genetic algorithm

Genetic algorithms are a class of optimization algorithms that use principles inspired by natural selection and genetics. In the context of cybersecurity, genetic algorithms have been used to detect and predict attack vectors based on the MITRE ATT&CK framework. One way in which genetic algorithms can be implemented is through the use of recursive functions.

The first step in applying genetic algorithms to detect and predict attack vectors is to create an initial population. This initial population consists of one or more techniques that are indicators of compromise detected on one of the assets of the system. The choice of the initial population can have a significant impact on the effectiveness and efficiency of the algorithm, as it determines the starting point for the search process.

Once the initial population has been established, the next step is to define the input vector for the genetic algorithm model. This input vector is obtained from the weight matrix for the detected indicator of compromise, and it represents the features or attributes that will be used to evaluate the fitness of the population. The weight matrix captures the relationships between the techniques in the MITRE ATT&CK framework, and it provides a way to quantify the importance or relevance of each technique for a given attack vector.

With the initial population and the input vector in place, the genetic algorithm can begin to create branches from the root node using the incoming weight vector. Each branch represents a possible path or sequence of techniques that could be used in a real-world cyberattack. The algorithm evaluates each branch by computing its fitness score, which is based on how well it matches the desired attack vector. The fitness score is then used to select the fittest branches for further optimization through crossover and mutation operations.

The process of filtering and normalization plays a crucial role in the accuracy of the genetic algorithm model. Initially, all the zero occurrences are removed from the technique vector, as they do not provide any relevant information. Additionally, the techniques already present in the previously created paths are eliminated to avoid duplication. Next, the techniques that do not fit the predicted kill-chain phase are filtered out. For example, while building a graph in the past to identify the entry point of the attack, the techniques that belong to the subsequent phases of the kill-chain are discarded. Similarly, for future predictions, all techniques that belong to earlier phases of the kill-chain than the given one is filtered out.

Once the filtering is done, the techniques are ranked according to their popularity. The most popular techniques, which account for 67% of the total, are selected, and the remaining 33% are discarded. This ensures that the model focuses only on the most relevant techniques.

After the ranking, the incoming vector is normalized to obtain a weighted estimate of the frequency of occurrence of each technique relative to its global popularity. Normalization involves dividing each value in the vector by the maximum number of occurrences. This step helps to obtain a uniform scale for the frequency of occurrence of the techniques.

All these filtering and normalization steps are essential for creating an accurate genetic algorithm model. They ensure that only the relevant techniques are selected, and their frequencies are represented uniformly. The resulting model can then be used for various tasks such as predicting the next likely attack vector or detecting potential attack paths in the system.

The finish criteria in the developed model are a crucial component that ensures the effectiveness and efficiency of the genetic algorithm. The criterion's primary purpose is to maintain a balance between computational resources and the accuracy of the results. The delta score, which is calculated for each step of the algorithm, is a measure of the change in the weight vector of the technique. The threshold value of 0.01 is set to ensure that the delta score over two steps is not significant.

The algorithm takes the delta from the previous step and calculates a new delta for the next step. If the two deltas are below the threshold of 0.01, the algorithm will not create a new branch for this technique. This is a crucial step that eliminates unnecessary computation and optimizes the algorithm's overall performance. Additionally, this ensures that the algorithm converges within a reasonable time frame, while also maintaining a high degree of accuracy in its results.

In cases where it is not possible to find a technique to jump to, the algorithm considers the creation of the path complete. This step ensures that the algorithm is comprehensive and explores all possible paths. The threshold value of 0.01 was chosen as the most optimal for the available computing resources, taking into account both the accuracy and speed of the algorithm.

Overall, the finish criteria play a significant role in the genetic algorithm. It ensures that the algorithm is effective, efficient, and comprehensive in its approach. By maintaining a balance between computational resources and the accuracy of the results, the algorithm can provide reliable and timely insights into the system's security posture.

In the genetic algorithm, the selection phase plays a critical role in determining which individuals in the population will be carried forward to the next generation. In our model, the selection phase occurs after the path creation stage, where we have already generated a set of potential paths based on the input vector. From these paths, we select 100 genes that will be carried forward to the next generation.

The selection process is based on two criteria: local fitness and global fitness. Local fitness is a measure of the effectiveness of the technique within its own path, while global fitness takes into account the performance of the technique across all paths.

To select the 100 genes, we follow a two-step process. First, we identify the top 50% of techniques based on their local fitness scores. These techniques have the highest weighted score within their respective paths and are thus the most effective at achieving their intended goal.

For the remaining 50% of genes, we look at the 20% best techniques across the entire population, with the exception of those in the top 50%. This ensures that we maintain a diverse population and do not simply replicate the best-performing individuals from the previous generation.

The selection phase is a critical component of the genetic algorithm, as it determines which individuals will be passed on to the next generation and ultimately shapes the trajectory of the algorithm's search for optimal solutions. By balancing local and global fitness criteria, we can maintain a diverse and effective population that is capable of achieving the desired goals.

In the genetic algorithm we are using, a mutation rule is employed to change the coefficient value of the graph path that has been constructed. This means that a mutation can occur in the graph at any point during the evolutionary process. By changing the coefficient value, we are able to explore new paths and potentially find more optimal solutions. The mutation rule is just one aspect of the genetic algorithm, which relies on principles of natural selection and genetic recombination to create new solutions to a problem. The algorithm uses a population of candidate solutions, or individuals, and iteratively evolves them to create better and better solutions over time.

In our research, we employed a series of steps to construct the attack graph for a given system based on the MITRE ATT&CK framework. We started by parsing the data from version 13 of the MITRE ATT&CK matrix, which involved compiling a list of technique popularity relative to a given one using related groups, campaigns, software variants, and date components. This information was used to fill in the global frequency matrix for each technique. We then determined the direction of the prediction (future or past) and the Stix ID of the technique for which the prediction would be built. The phase of the kill-chain was also determined based on the direction of the prediction. If the construction of the graph was directed towards the past, then the phase was considered final, but if it was directed towards the future, then it was initial. The depth step of the tree being built was also set as a parameter. The given technique (indicator of compromise) was represented as a single path vector and thrown into the recursive path building function. This function included several parameters such as the phase ID of the last element in the path vector, the prediction type, the count delta, the count factor, the chum count, and the depth to stop. The path creation function initially checked if the depth had been exceeded, and if it had, the execution ended with the return of all incoming parameters. Variable restrictions on the frequency of techniques were discarded, and the frequency vector was normalized by the maximum value in the range from 0 to 1. For each technique remaining in the vector, the search for scores began. To do this, the normalized score calculated at the previous stage was multiplied by the score coefficient, and the delta of the score and the delta calculated at this stage were checked to ensure that they exceeded the threshold value.

One important consideration was the need to avoid local minima and maxima in the data, which could lead to inaccurate or incomplete conclusions. To achieve this, we developed a multi-stage approach to calculating score coefficients, which take into account the relative frequency and depth of each technique in the matrix.

The first stage of the coefficient calculation involves multiplying the previous coefficient by the normalized frequency of the next technique. This allows us to weight the importance of each technique in the overall analysis, based on its observed prevalence in the matrix.

The second stage of the calculation involves applying a depth penalty multiplier to the coefficient. The depth penalty takes into account the current depth of the technique in the path vector, which is represented as a constant raised to the power of the depth. This helps to account for the fact that techniques that are further down the path may be less significant than those closer to the beginning.

The third step was to find the average frequency of the new technique to which the branch is being built. We used 80% of the upper frequencies to avoid negatively affecting the accuracy of the overall statistics. If the number of appearances of the new technique in the current technique is greater than the average value, then the coefficient is increased by the next value. The number of spawns of new techniques is divided by the average value and raised to the power of the depth penalty multiplied by the weight of the technique (formula 1). If the number of appearances is less than the average value, then the score coefficient is multiplied by the quotient of the frequency of the technique by the average frequency of the technique (formula 2).

To calculate the score coefficient for each technique, we first multiplied the previous coefficient by the normalized frequency of the next technique. Then, we applied a depth penalty multiplier, which is a constant raised to the power of the existing current depth. This helped to avoid local minima and maxima and ensure that the graph was accurately representing the relationships between techniques.

$$\mathit{new\_score\_coef}\ * = \frac{\mathit{technique\_frequency}}{\mathit{avg\_tech\_freq}}^{\mathit{depth\_penalty}\ *\ \mathit{weight\_tech\_freq}}$$

Formula 1. Calculating the weigh coefficient for exceeding average case

$$\mathit{new\_score\_coef}\ * = \frac{\mathit{technique\_frequency}}{\mathit{avg\_tech\_freq}}$$

Formula 2. Calculating the weigh coefficient for inferior average case

In the fourth step we divided the global frequency of the new technique by the global average occurrence of technique. If the result was greater than 1, then the new coefficient was multiplied by formula 3, otherwise, the quotient of the frequency of the given technique by its global frequency was used. By prioritizing relationships over global popularity, we were able to more accurately identify the most optimal paths for a particular system in the attack graph.

$$\mathit{new\_score\_coef}\ * = 1 + \frac{\mathit{total\_technique\_frequency} - \mathit{techniques\_average\_freq}}{\mathit{techniques\_max\_freq} - \mathit{techniques\_average\_freq}} \times \mathit{depth\_penalty} \times \mathit{weight\_total\_tech\_frequency}$$

Formula 3. Coefficient multiplier

For the fifth step it was essential to ensure that the new phase was calculated accurately, depending on whether the prediction was to the past or the future. For the case of a prediction to the past, it was necessary that the new phase be no higher than the current phase, and in the optimal case, be close to or equal to it. On the other hand, for the case of a predictor into the future, it was necessary that the new phase is not lower than the current phase, and in the optimal case, it should be close to or equal to it.

For the sixth step the recursive function was used to create a new path consisting of the current path and the new technique, a new phase identifier, a prediction type, a new score delta, a new score factor, and its sum with the new delta, and the depth value to stop. When the loop was exited, if the length of the path was 0, then the current path was returned, otherwise all paths created by the recursive function were returned.

Then algorithm enter a while loop and continue until the maximum length among all paths reaches the specified depth. The depth is increased by a fixed amount (delta) in each iteration. We select the most probable paths and use them to create new paths iteratively with increased depth. The iteration process follows a specific strategy to avoid getting stuck in local minimums. After testing different strategies, we found the optimal approach to be alternating the depth delta between 3 and 4 and using a mirrored interpretation. Values less than 3 are not suitable because they generate too few possible paths, while values greater than 5 increase the computational complexity and time of solving the problem. Using a depth greater than 4 reduces the prediction accuracy, which negatively affects the algorithm's results.

In order to predict the possible attack vector for a particular system, it is crucial to have a clear understanding of the security posture of all assets in the network. For this purpose, our research represents and describes each asset through the corresponding MITRE ATT&CK matrix, such as Windows[11], Linux[12], and MacOS[13]. By doing so, we can identify the specific vulnerabilities and techniques that are applicable to each system, as well as the potential attack paths that an adversary may follow to exploit these weaknesses. This approach enables us to create a comprehensive attack graph

[11] https://attack.mitre.org/matrices/enterprise/windows/
[12] https://attack.mitre.org/matrices/enterprise/linux/
[13] https://attack.mitre.org/matrices/enterprise/macos/

that accurately reflects the attack surface of the entire network and helps us to develop effective defensive strategies to mitigate the risks.

As we seek to predict the attack vector for a particular system, we must consider platform-specific restrictions. This means that the technique matrix we use will be tailored to the device or platform in question. For example, if we detect an IoC (Indicator of Compromise) on a Windows PC, some of the techniques in the MITRE ATT&CK matrix will not be applicable, as they are not possible on the Windows platform. This same principle applies to Linux, MacOS, or any other devices. By taking into account these platform-specific restrictions, we can ensure that our predictions are more accurate and relevant to the particular system we are analyzing.

A key aspect of identifying a kill chain is the identification of the "portal" techniques that can lead to its hatching. A predetermined set of such techniques has been identified, including "SSH", "Windows Command Shell", and others. Once it has been determined that one of the branches in the graph leads to one of these portals, the next step is to determine the potential machines that can be affected by that technique. However, it's essential to keep in mind that certain lateral movement techniques, such as "Windows Remote Management," cannot be used on Linux or MacOS machines. Therefore, depending on the technique used, the potential pool of machines that may be at risk will vary. This restriction is taken into account in the technique matrix used by our algorithm, ensuring that only the appropriate techniques are considered for each platform.

5 DISCUSSION

5.1 Algorithm limitations

The proposed Master's thesis on "A method of detecting and predicting attack vectors based on genetic programming" offers a novel approach for predicting and detecting attack vectors through the use of genetic programming. The thesis aims to address the growing need for effective and efficient cybersecurity measures to protect against cyber threats and attacks.

The thesis proposes a genetic algorithm that utilizes the MITRE ATT&CK Matrix to predict and detect attack vectors for different systems. The algorithm uses a graph-based approach that builds a path based on the MITRE ATT&CK Matrix for each system, with each path representing a potential attack vector. The algorithm then uses a scoring mechanism that considers various factors such as frequency and depth penalty to identify the most probable attack vector.

The thesis also proposes a mutation rule, which involves changing the value of the coefficient of the graph path to adapt to new and emerging threats. This approach ensures that the algorithm remains effective and up-to-date in detecting and predicting attack vectors.

The proposed genetic algorithm is a significant contribution to the field of cybersecurity as it provides a robust and efficient approach to detecting and predicting attack vectors. The approach is flexible and can be adapted to different systems, making it a valuable tool for cybersecurity professionals. The genetic algorithm proposed provides an effective and efficient approach for detecting and predicting attack vectors.

However, there are some limitations to the proposed approach. The effectiveness of the algorithm may be affected by the quality and completeness of the MITRE ATT&CK Matrix. Additionally, the algorithm may require significant computational resources, making it challenging to implement in large-scale systems.

Also, one of the limitations of our study is that we only used the MITRE ATT&CK Enterprise Matrix to build our model. We did not include data from the ICS and Mobile matrices. This limitation is significant because, in today's world, the use of mobile devices and industrial control systems in corporate networks is increasing. The concept of BYOD and MDM has made it easier for employees to access corporate data through their personal devices. This means that there is a higher likelihood of these devices being compromised, and it is essential to consider these devices in any predictive model for attack vectors. Industrial control systems are also a critical aspect that needs to be taken into account. The increasing number of these systems being used in various industries has made them a potential target for cyberattacks. The absence of these matrices from our study is a significant limitation that needs to be addressed in future research.

Another limitation is the use of data from only one source, the MITRE ATT&CK Matrix. While this matrix is a widely recognized and respected resource in the field of cybersecurity, it is important to note that it may not capture all possible attack vectors or techniques. There are likely to be many attacks that have not been included in the matrix, as they may not have been reported or discovered yet. Therefore, the findings of the study may not be comprehensive and could potentially miss some important attack vectors.

5.2 Research question answers and possible impacts

RQ1. What is the best mathematical representation of MITRE ATT&CK matrix to find the connection between different techniques?

The MITRE ATT&CK Matrix is a powerful tool for analysing and understanding the techniques used by attackers. In the context of the Master Thesis, the MITRE ATT&CK Matrix was presented as a two-dimensional numpy array, which is a data structure that is widely used in scientific computing. The matrix consisted of techniques, which are specific methods or approaches used by attackers to achieve their goals, and their precalculated population, which is the number of times each technique had been observed in the wild.

The choice of a numpy array for representing the MITRE ATT&CK Matrix was deliberate, as it provided several benefits over other data structures. Numpy arrays are designed for efficient computation and manipulation of large numerical arrays, which made it ideal for analysing and processing the large amounts of data contained in the MITRE ATT&CK Matrix. Additionally, numpy arrays can be easily integrated with other data analysis tools, which made it possible to perform complex analysis and visualization of the data.

By representing the MITRE ATT&CK Matrix in this way, the research was able to examine the relationships and connections between different techniques. This allowed for a deeper understanding of the tactics and behaviours of attackers, as well as the ability to identify patterns and trends in their techniques. These insights can be used to inform the development of more effective cybersecurity measures, such as intrusion detection systems and threat intelligence feeds.

Overall, the use of a numpy array to represent the MITRE ATT&CK Matrix was a powerful and effective approach for analysing and understanding the techniques used by attackers.

RQ2. What are the key factors that influence the performance and reliability of genetic programming algorithms for detecting and predicting attack vectors, and how can these factors be optimized through parameter tuning and other techniques?

In our work, the factors that influence the performance and accuracy of the genetic programming algorithm for detecting and predicting attack vectors are the various weights and constraints in the recursion function. These factors determine the balance between the speed of the algorithm and its accuracy. By adjusting these weights and constraints, we can optimize the algorithm for real-world scenarios. For example, if we want to prioritize speed over accuracy, we can adjust the weights in favor of faster computations. On the other hand, if we want to prioritize accuracy over speed, we can adjust the weights in favor of more precise computations, even if it means slower processing times. Finding the optimal balance between these factors is crucial for the success of the algorithm in detecting and predicting attack vectors.

RQ3. What are mutation rules for developing genetic algorithm?

In a genetic algorithm, a mutation is a random change introduced into the genetic makeup of a population of potential solutions. These changes are meant to introduce new variations into the population and explore new areas of the search space. In the context of graph path building, a mutation rule refers to a specific type of mutation that alters the coefficient value of a graph path.

The coefficient value of a graph path is a crucial parameter that determines the weight and importance of the path in the overall graph. Changing the value of the coefficient can lead to significant changes in the structure and behavior of the graph, potentially opening up new paths and avenues for exploration. In the genetic algorithm, a mutation rule can be applied to a single individual in the population, or to multiple individuals at once, depending on the specific implementation.

In general, the goal of the mutation rule is to introduce enough randomness into the population to avoid premature convergence to suboptimal solutions, while still preserving the overall structure and coherence of the graph. By changing the coefficient values of the graph paths in a controlled and adaptive manner, the genetic algorithm can explore new areas of the search space and potentially discover more efficient and effective solutions.

RQ4. How can a genetic programming approach be optimized for improved performance and efficiency in real-world scenarios?

To optimize the genetic programming approach for detecting and predicting attack vectors in real-world scenarios, it is important to continuously update the database with newer attack data. This will help to improve the accuracy and efficiency of the algorithm as it will have access to a wider range of techniques and tactics used by attackers. Additionally, frequency conversion of techniques based on updated data can help to ensure that the algorithm is up-to-date and relevant. By identifying and prioritizing the most frequently used techniques in the updated dataset, the algorithm can be fine-tuned to focus on the most relevant and current attack vectors. Other optimization techniques include adjusting the weights and constraints in the recursion function to find the optimal balance between speed and accuracy. It is also important to consider the hardware and computational resources available for the algorithm and to optimize it accordingly to ensure efficient performance in real-world scenarios. Finally, the use of hardware acceleration, such as GPUs or FPGAs, can greatly speed up the execution of the genetic algorithm, especially for larger problem sizes. However, this approach may require specialized hardware and expertise, and may not be cost-effective for smaller-scale applications.

Developing a genetic algorithm solution based on the MITRE ATT&CK Matrix for attack vector detection and prediction has the potential for significant wider consequences. One of the most immediate consequences is the improvement of cybersecurity for organizations and individuals. By using a genetic algorithm, it is possible to predict and detect attack vectors with a high level of accuracy, thus preventing potential cyber-attacks and data breaches. This, in turn, can lead to increased trust in online systems, reduced financial losses due to cybercrime, and improved protection of sensitive personal and business data. Additionally, the development of such a solution can contribute to the advancement of artificial intelligence and machine learning research, as well as improve the understanding of cyber threats and their patterns. It could also lead to the development of new tools and methodologies for cybersecurity professionals, making it easier to identify and mitigate potential threats in real-time. Overall, the development of a genetic algorithm solution for attack vector detection and prediction based on the MITRE ATT&CK Matrix can have far-reaching consequences for cybersecurity and the broader field of technology.

The impact of developing a genetic algorithm solution based on the MITRE ATT&CK Matrix for attack vector detection and prediction can be significant for society. The use of such an algorithm can enhance the security of various systems, such as critical infrastructure, financial systems, healthcare systems, and others. This can reduce the risk of cyberattacks, data breaches, and other security threats that can harm individuals and organizations. The development of such algorithms can also lead to the advancement of the field of cybersecurity and contribute to the creation of new job opportunities in this area. Moreover, the implementation of genetic programming approaches can improve the efficiency and accuracy of attack vector detection and prediction, which can save time, resources, and costs for organizations. Overall, the development of genetic algorithm solutions for attack vector detection and prediction can have a positive impact on society by improving cybersecurity and reducing the risks of cyber threats.

The development of a genetic algorithm solution based on the MITRE ATT&CK Matrix for attack vector detection and prediction can have potential ethical consequences. One concern is the possibility of the algorithm being used for malicious purposes, such as creating more effective attacks. Another concern is the privacy and security of individuals, as the algorithm may be used to target specific individuals or groups. The algorithm may also perpetuate biases if it is trained on biased data, leading to discrimination against certain individuals or groups. Additionally, the algorithm may raise questions about accountability, responsibility, and liability if it makes decisions that have negative consequences for individuals or organizations. Therefore, it is important to consider ethical implications in the development and use of such algorithms and to ensure that they are designed and deployed in an ethical and responsible manner.

The impact of developing a genetic algorithm solution based on the MITRE ATT&CK Matrix for attack vector detection and prediction on sustainability is not directly clear. However, indirectly, it could have some impact on sustainability. One of the potential applications of this solution is to enhance the security of critical infrastructure systems such as power plants, water treatment plants, and transportation systems. By providing an efficient way to detect and predict attack vectors, this solution can help prevent cyber-attacks that may cause physical damage to these systems and have significant environmental impacts. Additionally, as more industries move towards digitization, there is a growing need to ensure the security of these systems and reduce their vulnerability to cyber-attacks. This solution can potentially contribute to achieving this goal and promoting the sustainable development of these industries.

6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

This Master's Thesis has the potential to contribute significantly to the global body of knowledge in cybersecurity. The research presents a novel approach that utilizes genetic programming to analyze and predict attack vectors based on the MITRE ATT&CK Matrix. This approach has the potential to provide more accurate and efficient detection and prediction of attacks compared to existing methods. The thesis also identifies key factors that influence the performance and reliability of genetic programming algorithms and suggests ways to optimize these factors. Additionally, the research provides insights into the potential ethical implications of developing such a solution and the need for caution in its implementation. What is more, the Master's Thesis on this topic has the potential to advance the field of cybersecurity and contribute to the development of more effective and efficient approaches to detecting and predicting cyberattacks.

The research questions address different aspects of cybersecurity, ranging from mathematical representations of the MITRE ATT&CK Matrix to the development and optimization of genetic programming algorithms for detecting and predicting attack vectors. The first question explores the benefits of using a numpy array to represent the MITRE ATT&CK Matrix, while the second question focuses on the use of STIX ID as a common identifier for building the attack vector graph. The third and fourth questions investigate the performance and reliability of genetic programming algorithms for detecting and predicting attack vectors, as well as the key factors that influence their performance. The fifth question examines mutation rules for developing genetic algorithms, while the sixth question focuses on the optimization of genetic programming approaches for improved performance and scalability.

RQ1: The best mathematical representation of the MITRE ATT&CK Matrix to find the connection between different techniques is a two-dimensional numpy array, which provides benefits in terms of efficient computation and manipulation of large numerical arrays and easy integration with other data analysis tools.

RQ2: The key factors that influence the performance and reliability of genetic programming algorithms for detecting and predicting attack vectors are the various weights and constraints in the recursion function, which determine the balance between speed and accuracy. These factors can be optimized through parameter tuning to find the optimal balance between speed and accuracy for real-world scenarios.

RQ3: In genetic algorithms, a mutation rule refers to a specific type of mutation that alters the coefficient value of a graph path. The goal of the mutation rule is to introduce enough randomness into the population to avoid premature convergence to suboptimal solutions while still preserving the overall structure and coherence of the graph.

RQ4: A genetic programming approach can be optimized for improved performance and efficiency by using techniques such as multi-objective optimization, adaptive parameter tuning, and parallelization. These techniques can help to find the best balance between speed and accuracy and reduce computation time while ensuring the validity and reliability of the results.

This thesis discusses the use of genetic programming techniques to identify and predict potential attack vectors in computer systems. Some of the most important issues that are addressed in this thesis could include:

1. The need for effective methods of detecting and predicting attack vectors: As cyberattacks become increasingly sophisticated and frequent, there is a growing need for more advanced techniques to identify and predict potential attack vectors in order to prevent them from occurring.
2. The limitations of current approaches: Many existing methods of detecting and predicting attack vectors rely on rule-based or statistical models, which may be ineffective against new or complex attacks. Genetic programming may offer a more adaptable and flexible approach.
3. The potential benefits of genetic programming: Genetic programming is a type of machine learning that uses evolutionary algorithms to generate programs that can solve a particular problem. In the context of detecting and predicting attack vectors, genetic programming may be able to identify new attack patterns those other methods have missed.

4. The challenges of applying genetic programming to cybersecurity: There are a number of challenges associated with using genetic programming to detect and predict attack vectors, including the need for large amounts of training data.

6.2 Future works

Possible research question for future works could include the following:

How can the proposed method for detecting and predicting attack vectors based on genetic programming be adapted and applied to different types of systems, such as cloud-based infrastructures or Internet of Things (IoT) devices? This research question would examine the potential versatility and scalability of the proposed method, and could lead to the development of new approaches for detecting and predicting attack vectors in various types of systems.

How genetic programming-based approach can be combined with existing machine learning techniques for predicting new attack vectors? This research question would involve developing a novel approach that combines genetic programming with other machine learning techniques to predict new attack vectors. The study could evaluate the performance of the hybrid approach and compare it to other existing methods for predicting new attack vectors.

What is the effectiveness of integrating genetic programming-based detection methods with other cybersecurity solutions? This research question would explore the potential benefits of combining genetic programming-based methods with other cybersecurity solutions, such as firewalls or intrusion detection systems. The study could evaluate the performance of such integrated solutions in terms of accuracy and overall effectiveness.

There are several potential follow-up and new projects that can be pursued based on the Master Thesis on the theme of detecting and predicting attack vectors using genetic programming. Here are three examples:

1. Developing an Automated System: The Master Thesis can be extended by developing an automated system that integrates the proposed method for detecting and predicting attack vectors. This system can be used by security analysts and administrators to detect and respond to threats quickly. The system can be developed as a standalone application or integrated with existing security solutions.
2. Applying the Method to Different Domains: The proposed method can be applied to different domains, such as IoT networks, cloud-based systems, and industrial control systems. The effectiveness of the method can be evaluated by comparing the results obtained from different domains. This can help in understanding the limitations and potential of the method.
3. Hybrid Approach: The proposed method can be combined with other existing methods for detecting and predicting attacks to create a hybrid approach. This approach can leverage the strengths of different methods to improve the accuracy and effectiveness of the detection and prediction process. The proposed hybrid approach can be evaluated using datasets and scenarios that represent real-world attack scenarios.

There are several potential future directions for research based on the Master Thesis on detecting and predicting attack vectors using genetic programming:

Improving the accuracy of the system: The current system may not be perfect, and there may be ways to improve its accuracy. For example, the system may need to be trained on a larger dataset to increase its predictive power, or new features could be added to the system to help it better detect attack vectors. Also, two other types of MITRE ATT&CK Matrices should be included as a data source, so the model could be able to build attack vectors for systems with mobile devices and industrial control systems.

Extending the system to new types of attacks: The current system may be tailored to a specific type of attack, but it could be extended to detect and predict other types of attacks. For example, the system could be adapted to detect phishing attacks or malware attacks.

Evaluating the system in a real-world scenario: The current system may have been tested in a controlled environment, but it would be valuable to evaluate its performance in a real-world scenario. This could involve working with a company or organization to test the system against real-world attacks and evaluate its effectiveness.

Developing a user-friendly interface: The current system may be complex and difficult to use for non-experts. Developing a user-friendly interface could help make the system more accessible to a wider range of users and increase its adoption.

Conducting a comparative study: The current system could be compared to other methods for detecting and predicting attack vectors, such as rule-based systems or machine learning models. This would help to determine the strengths and weaknesses of different approaches and identify areas for further improvement.

Adding a language model: Since the input to the algorithm is already detected indicators of compromise related to a known technique, the language model can help isolate a particular technique based on a linguistic description of the attack in progress.

7 REFERENCES

[1] S. I. S. I. M. Hamizan Suhaimi, "Network intrusion detection system by using genetic algorithm," *Indonesian Journal of Electrical Engineering and Computer Science 16(3):1593,* no. 16(3):1593, 2019.

[2] I. T. Wei Lu, "Evolutionary Computation," in *Detecting New Forms of Network Intrusion Using Genetic Programming* , 2014.

[3] A. V. A. A. Nikhil Mane, "International Symposium on Computational Intelligence and Informatics," in *A Pragmatic Optimal Approach for Detection of Cyber Attacks using Genetic Programming*, 2020.

[4] A. N. Z.-H. Patrick LaRoche, "European Conference on Genetic Programming," in *802.11 De-authentication Attack Detection Using Genetic Programming*, 2006.

[5] S. S. R. A. A. Kashif Naseer Qureshi, "A novel and secure attacks detection framework for smart cities industrial internet of things," *Sustainable Cities and Society,* vol. 61, 2020.

[6] A. N. Z.-H. Patrick LaRoche, "Communication Networks and Services Research Conference," in *Genetic programming based WiFi data link layer attack detection*, 2016.

[7] Q. U. N. X. H. N. Tuan Anh Pham, "Advances in Intelligent Systems and Computing," in *Phishing attacks detection using genetic programming*, 2014.

[8] M. N. S. N. M. Muhammad Syafiq Mohd Pozi, "Improving Anomalous Rare Attack Detection Rate for Intrusion Detection System Using Support Vector Machine and Genetic Programming," *Neural Processing Letters,* vol. 44, no. 2, pp. 279-290, 2016.

[9] A. A. A. M. A. A. A. M. Abd Allah A. Galal, "Ant Colony Optimization Approach Based Genetic Algorithms for Multiobjective Optimal Power Flow Problem under Fuzziness," *Applied Mathematics,* no. 4, 2013.

APPENDIX A. PROGRAM CODE

```
import math
import random
import time
import numpy as np
import techniques_func as tf
from mitreattack.stix20 import MitreAttackData
start_time = time.time()

mitre_attack_data: MitreAttackData
tactics_order = {
      'reconnaissance':  0,  'resource-development':  1,  'initial-access':  2,
'execution': 3, 'persistence': 4,
     'privilege-escalation': 5, 'defense-evasion': 6, 'credential-access': 7,
'discovery': 8, 'lateral-movement': 9,
    'collection': 10, 'command-and-control': 11, 'exfiltration': 12, 'impact':
13
}
tactics_list = list(tactics_order.keys())
tactics_freq: np.ndarray = np.zeros((len(tactics_order), 1), dtype=object)
tactics_average: np.ndarray = np.zeros((len(tactics_order), 1), dtype=object)
total_techniques_freq: np.ndarray
techniques_dict = {}
techniques_keys = []
techniques_freq: np.ndarray
techniques_average_freq: int = 0
techniques_max_freq: int = 0
sources = ['software', 'groups', 'campaigns', 'datacomponents']

techniques_keys_dict = {}
techniques_freq_list = []


def get_technique_matrix():
    all_techniques = tf.get_all_techniques(mitre_attack_data)
    techniques_keys = list(all_techniques.keys())

    # make numpy array where first column is technique stix_id and second column
is number of all occurrences in of
    # technique across campaigns, groups and software
    technique_matrix = np.zeros((len(techniques_keys), 2), dtype=object)
    technique_matrix[:, 0] = techniques_keys

    return (all_techniques, techniques_keys, technique_matrix)


# function to count sum of second column of numpy array
def get_sum_of_techniques(technique_matrix):
    technique_matrix = np.delete(technique_matrix, np.s_[0], axis=1)
    return int(np.sum(technique_matrix.astype('int64')))


# function for divising second column of numpy array by sum of all techniques
def get_technique_matrix_normalized(technique_matrix):
    technique_matrix_normed_1 = np.matrix.transpose(technique_matrix[:, 1])
        technique_matrix_normed_2   =   np.reshape(technique_matrix_normed_1   /
get_sum_of_techniques(technique_matrix),
                                           (len(technique_matrix_normed_1), 1))
     weighted_matrix  =  np.append(technique_matrix,  technique_matrix_normed_2,
axis=1)
    return weighted_matrix


def get_bounded_techniques(technique, source_name, techniques_freq):
    techniques_bound_freq = techniques_freq.copy()
    techniques_list = []
    sources_list = tf.get_technique_dependences(mitre_attack_data, source_name,
technique)
    for source in sources_list:
        if source_name == 'software':
                                                     techniques_list        +=
mitre_attack_data.get_techniques_used_by_software(source)
        elif source_name == 'groups':
                                                     techniques_list        +=
mitre_attack_data.get_techniques_used_by_group(source)
                                              #       techniques_list       +=
mitre_attack_data.get_techniques_used_by_group_software(source)
        elif source_name == 'campaigns':
                                                     techniques_list        +=
mitre_attack_data.get_techniques_used_by_campaign(source)
        elif source_name == 'datacomponents':
                                                     techniques_list        +=
mitre_attack_data.get_techniques_detected_by_datacomponent(source)
        else:
            print('Wrong source_name')
    for technique in techniques_list:
        if isinstance(technique, dict):
            technique = technique['object']
        try:
            techniques_bound_freq[techniques_keys_dict[technique.id]][1] += 1
        except KeyError:
            # It is deprecated technique
            pass
    return techniques_bound_freq


def get_fitness(technique_list, technique_matrix):
    fitness = 0
    for technique in technique_list:
        index = np.where(technique_matrix == technique)
        fitness += technique_matrix[index[0][0]][2]
    return fitness



def fill_tactics_frequency():
    global tactics_freq
    for technique_id in techniques_dict:
        technique = techniques_dict[technique_id]
        technique_phases = technique.kill_chain_phases
        for technique_phase in technique_phases:
            if technique_phase['kill_chain_name'] != 'mitre-attack':
                continue
            tactics_freq[tactics_order[technique_phase['phase_name']]] += 1
    return True


def fill_techniques_frequencies():
    global total_techniques_freq, techniques_average_freq, techniques_max_freq
    total_techniques_freq = np.zeros((len(techniques_keys), 1), dtype=object)
    for technique_id in range(len(techniques_keys)):
                                   total_techniques_freq[technique_id]        =
techniques_freq_list[technique_id][technique_id][1]
    total_freq = total_techniques_freq.sum()
    techniques_average_freq = total_freq / len(total_techniques_freq)
    techniques_max_freq = total_techniques_freq.max()
    return True


def   create_paths(input_path,   phase_id,   prediction_type,   score_delta,
score_coef, score, depth_to_stop=999999):
    if depth_to_stop <= len(input_path):
        return [[input_path, score, phase_id, score_delta, score_coef]]

    latest_techniques = input_path[-1]
                       latest_techniques_freq:          np.ndarray          =
(techniques_freq_list[techniques_keys_dict[latest_techniques]])
    latest_techniques_freq = latest_techniques_freq.copy()
    # remove all techniques that are already exist in the path
    for technique in input_path:
        latest_techniques_freq[techniques_keys_dict[technique]][1] = 0

    # remove all techniques with 0 frequency
     latest_techniques_freq  =  latest_techniques_freq[latest_techniques_freq[:,
1] > 0, :]
```

```
    # filter out techniques with wrong phase_id
    for technique in latest_techniques_freq:
        technique_info = techniques_dict[technique[0]]
        needed_phase_exists = False
        for technique_phase in technique_info.kill_chain_phases:
            if technique_phase['kill_chain_name'] != 'mitre-attack':
                continue
            if prediction_type == 'past':
                if tactics_order[technique_phase['phase_name']] <= phase_id:
                    needed_phase_exists = True
                    break
            else:
                if tactics_order[technique_phase['phase_name']] >= phase_id:
                    needed_phase_exists = True
                    break
        if needed_phase_exists is False:
            technique[1] = 0

    # remove all techniques with 0 frequency
     latest_techniques_freq  =  latest_techniques_freq[latest_techniques_freq[:,
1] > 0, :]

     latest_techniques_freq  =  latest_techniques_freq[latest_techniques_freq[:,
1].argsort()[::-1]]
    if len(latest_techniques_freq) == 0:
        return [[input_path, score, phase_id, score_delta, score_coef]]
    max_score = np.max(latest_techniques_freq, axis=0)[1]
    latest_techniques_freq_normalized = latest_techniques_freq[:, 1] / max_score
    local_paths_list = []

    top_20  =  latest_techniques_freq[latest_techniques_freq[:,  1]/max_score  >=
1/3, :]
    top_20_normalized = top_20[:, 1] / max_score

    for i in range(len(top_20)):
        new_score_delta = top_20_normalized[i] * score_coef
        if score_delta < 0.01 and new_score_delta < 0.01:
            continue
        new_score_coef = score_coef * top_20_normalized[i]
        depth_penalty = pow(0.8, len(input_path))
        new_score_coef *= depth_penalty
        technique_index = techniques_keys_dict[top_20[i][0]]

                              frequences_of_technique:      np.ndarray       =
techniques_freq_list[technique_index]
        frequences_of_technique = frequences_of_technique.copy()
                                            frequences_of_technique          =
frequences_of_technique[frequences_of_technique[:, 1] > 0, :]
```

```
                                                    frequences_of_technique         =
frequences_of_technique[frequences_of_technique[:, 1].argsort()[::-1]]
        frequences_of_technique = frequences_of_technique[1:, :]
                                                    frequences_of_technique         =
frequences_of_technique[frequences_of_technique[:,             1]             >=
frequences_of_technique[0][1] * 0.2, :]
        avg_tech_freq = frequences_of_technique[:, 1].mean()

        if top_20[i][1] / avg_tech_freq > 1:
            new_score_coef *= pow((top_20[i][1] / avg_tech_freq), depth_penalty
* 0.5)
        else:
            new_score_coef *= top_20[i][1] / avg_tech_freq

         if total_techniques_freq[technique_index] / techniques_average_freq >
1:
                #new_score_coef  *=  pow(total_techniques_freq[technique_index]  /
techniques_average_freq, depth_penalty)
            new_score_coef *= 1.0 + ((total_techniques_freq[technique_index] -
techniques_average_freq) / (techniques_max_freq - techniques_average_freq)) *
0.25 * depth_penalty
        else:
                  new_score_coef  *=  total_techniques_freq[technique_index]  /
techniques_average_freq

        #zif new_score_coef > 1:
                #print('new_score_coef  >  1',  score_coef,  top_20_normalized[i],
new_score_coef, top_20[i][1], avg_tech_freq, top_20[i][1] / avg_tech_freq)
        phase_delta = len(tactics_order)
        technique_info = techniques_dict[top_20[i][0]]
        for technique_phase in technique_info.kill_chain_phases:
            if technique_phase['kill_chain_name'] != 'mitre-attack':
                continue
            if prediction_type == 'past':
                if phase_id >= tactics_order[technique_phase['phase_name']]:
                                                       phase_delta    =    min(phase_id    -
tactics_order[technique_phase['phase_name']], phase_delta)
            else:
                if phase_id <= tactics_order[technique_phase['phase_name']]:
                                                                      phase_delta      =
min(tactics_order[technique_phase['phase_name']] - phase_id, phase_delta)
        if prediction_type == 'past':
            new_phase_id = phase_id - phase_delta
        else:
            new_phase_id = phase_id + phase_delta
        local_paths_list += create_paths(
            input_path + [top_20[i][0]],
            new_phase_id,
            prediction_type,
            new_score_delta,
            new_score_coef,
            score + new_score_delta,
            depth_to_stop=depth_to_stop
        )
    if len(local_paths_list) == 0:
        return [[input_path, score, phase_id, score_delta, score_coef]]
    return local_paths_list


def predict_techniques(prediction_type, pivot_point, phase_id=-1):
    if phase_id == -1:
        if prediction_type == 'past':
            for phase in techniques_dict[pivot_point].kill_chain_phases:
                if phase['kill_chain_name'] != 'mitre-attack':
                    continue
                phase_id = max(tactics_order[phase['phase_name']], phase_id)
        else:
            phase_id = 999999999
            for phase in techniques_dict[pivot_point].kill_chain_phases:
                if phase['kill_chain_name'] != 'mitre-attack':
                    continue
                phase_id = min(tactics_order[phase['phase_name']], phase_id)
    else:
        delta = 999999999
        if prediction_type == 'past':
            for phase in techniques_dict[pivot_point].kill_chain_phases:
                if phase['kill_chain_name'] != 'mitre-attack':
                    continue
                if tactics_order[phase['phase_name']] <= phase_id:
                    delta = min(phase_id - tactics_order[phase['phase_name']],
delta)
            phase_id -= delta
        else:
            for phase in techniques_dict[pivot_point].kill_chain_phases:
                if phase['kill_chain_name'] != 'mitre-attack':
                    continue
                if tactics_order[phase['phase_name']] >= phase_id:
                    delta = min(tactics_order[phase['phase_name']] - phase_id,
delta)
            phase_id += delta
    top_score_by_delta = {2: 500, 3: 120, 4: 50}
    depth_delta = 3
    depth_to_stop = depth_delta
        paths   =   create_paths(input_path=[pivot_point],   phase_id=phase_id,
prediction_type=prediction_type,
                                 score_delta=99999999,  score_coef=1,  score=0,
depth_to_stop=depth_to_stop)
    while count_max_path(paths) == depth_to_stop:
        if depth_delta == 3:
            depth_delta = 4
        else:
            depth_delta = 3
        depth_to_stop += depth_delta
        print("Current depth to stop: ", depth_to_stop)
        predicted_paths = remove_duplicates(paths)
        scores = predicted_paths.keys()
        scores = sorted(scores, reverse=True)
        paths = []
        top_scores_count = min(len(scores), min(max(10, int(len(scores) * 0.01)
+ 1),  top_score_by_delta[depth_delta]))
        print(top_scores_count)
        for top_score in scores[:top_scores_count]:
            for path in predicted_paths[top_score]:
                paths += create_paths(
                                        input_path=path[0],   phase_id=path[1],
prediction_type=prediction_type, score_delta=path[2],
                                         score_coef=path[3],   score=top_score,
depth_to_stop=depth_to_stop
                )
    result_paths = paths

    print("--- %s seconds ---" % (time.time() - start_time))
    depth_delta = 4
    print("Depth delta: ", depth_delta)
    depth_to_stop = depth_delta
        paths   =   create_paths(input_path=[pivot_point],   phase_id=phase_id,
prediction_type=prediction_type,
                                 score_delta=99999999,  score_coef=1,  score=0,
depth_to_stop=depth_to_stop)
    while count_max_path(paths) == depth_to_stop:
        if depth_delta == 3:
            depth_delta = 4
        else:
            depth_delta = 3
        depth_to_stop += depth_delta
        print("Current depth to stop: ", depth_to_stop)
        predicted_paths = remove_duplicates(paths)
        scores = predicted_paths.keys()
        scores = sorted(scores, reverse=True)
        paths = []
        top_scores_count = min(len(scores), min(max(10, int(len(scores) * 0.01)
+ 1), top_score_by_delta[depth_delta]))
        print(top_scores_count)
        for top_score in scores[:top_scores_count]:
            for path in predicted_paths[top_score]:
                paths += create_paths(
                                        input_path=path[0],   phase_id=path[1],
prediction_type=prediction_type, score_delta=path[2],
                                         score_coef=path[3],   score=top_score,
depth_to_stop=depth_to_stop
                )
    result_paths += paths

    return result_paths




def init_all():
      global  mitre_attack_data,  tactics_list,  tactics_freq,  tactics_average,
total_techniques_freq, techniques_dict, \
                  techniques_keys,    techniques_freq,    techniques_keys_dict,
techniques_freq_list
    mitre_attack_data = MitreAttackData("enterprise-attack-13.0.json")
    techniques_dict, techniques_keys, techniques_freq = get_technique_matrix()

    print("--- %s seconds ---" % (time.time() - start_time))
    techniques_dict, techniques_keys, techniques_freq = get_technique_matrix()
    print("--- %s seconds ---" % (time.time() - start_time))
    print(tf.average_techniques_by_software(mitre_attack_data))
    print(tf.average_techniques_by_groups(mitre_attack_data))
    print(tf.average_techniques_by_campaigns(mitre_attack_data))
    print(tf.average_techniques_by_datacomponents(mitre_attack_data))
    # print execution time
    print("--- %s seconds ---" % (time.time() - start_time))

    # fill tactics frequency
    fill_tactics_frequency()
    print("--- %s seconds ---" % (time.time() - start_time))
    # print(tactics_freq)

    # fill techniques frequency
                 techniques_keys_dict        =        dict(zip(techniques_keys,
range(len(techniques_keys))))
    techniques_freq_list = []
    for technique in techniques_keys:
        techniques_freq_list.append(techniques_freq.copy())
        for source in sources:
            techniques_freq_list[-1] = get_bounded_techniques(
                technique,
                source,
                techniques_freq_list[-1],
            )
    print("--- %s seconds ---" % (time.time() - start_time))
    fill_techniques_frequencies()
    return True


# remove duplicate lists of strings in list
def remove_duplicates(list_of_lists):
    score_dict = {}
    for item in list_of_lists:
        if isinstance(item[1], np.ndarray):
            score = item[1][0]
        else:
            score = item[1]
        try:
            score_dict[score] = [[item[0], item[2], item[3], item[4]]]
        except KeyError:
            if item[0] not in score_dict[score]:
                score_dict[score].append([item[0], item[2], item[3], item[4]])
    return score_dict


def count_max_path(paths):
    max_path = 0
    for path in paths:
        if len(path[0]) > max_path:
            max_path = len(path[0])
    return max_path


def main():
    init_all()
    print("--- %s seconds ---" % (time.time() - start_time))
    print("Predicting techniques")
    # Fill with STIX-IDs of the technique
    ioc_attack_patterns = [

    ]
    for attack_pattern in ioc_attack_patterns:
              print("Predicting  past  techniques  for  attack  pattern:  ",
techniques_dict[attack_pattern].name,
techniques_dict[attack_pattern].external_references[0]['external_id'])
        predicted_paths = predict_techniques('past', attack_pattern)
        predicted_paths = remove_duplicates(predicted_paths)
        scores = predicted_paths.keys()
        scores = sorted(scores, reverse=True)
        predicted_top_path = (predicted_paths[scores[0]])[0]
        for technique in predicted_top_path[0]:
                                        print(techniques_dict[technique].name,
techniques_dict[technique].external_references[0]['external_id'])
```

```
             print("Predicting  future  techniques  for  attack  pattern:  ",
techniques_dict[attack_pattern].name,
techniques_dict[attack_pattern].external_references[0]['external_id'])
        predicted_paths = predict_techniques('future', attack_pattern)
        predicted_paths = remove_duplicates(predicted_paths)
        scores = predicted_paths.keys()
        scores = sorted(scores, reverse=True)
        predicted_top_path = (predicted_paths[scores[0]])[0]
        for technique in predicted_top_path[0]:
                                        print(techniques_dict[technique].name,
techniques_dict[technique].external_references[0]['external_id'])
    print("--- %s   ---" % (time.time() - start_time))
    return 0


if __name__ == "__main__":
    main()
```

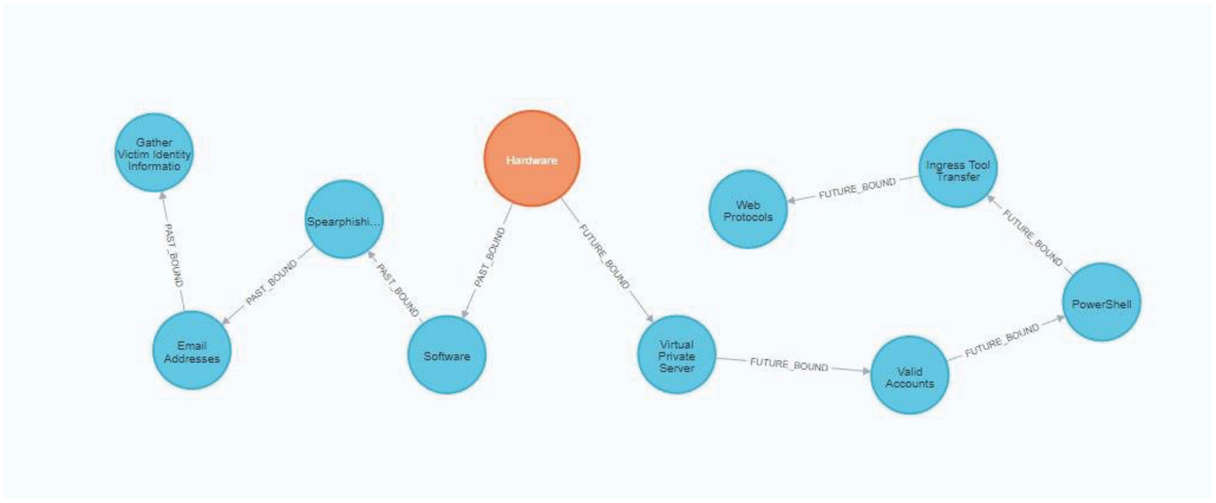
APPENDIX B. FUNCTION MODULE

```
from mitreattack.stix20 import MitreAttackData
import numpy as np

def groups_and_techniques(mitre_attack_data):
    # get all techniques related to campaigns
                                    techniques_used_by_groups                  =
mitre_attack_data.get_all_techniques_used_by_all_groups()

    groups = {}


    for group in techniques_used_by_groups:
        techniques = techniques_used_by_groups[group]

        groups[group] = []

        for technique in techniques:
            object = technique['object']
            if 'enterprise-attack' in object.x_mitre_domains:
                groups[group].append({'stix':object.id,
                               'id':object.external_references[0].external_id,
                                 'name':object.name,
                                 'group_id':group})

    return groups


def software_and_techniques(mitre_attack_data):

    # get all techniques related to software
                                   techniques_used_by_software                 =
mitre_attack_data.get_all_techniques_used_by_all_software()

    attack_vectors = {}

    for stix_id in techniques_used_by_software:
        techniques = techniques_used_by_software[stix_id]

        attack_vectors[stix_id] = []

        for technique in techniques:
            object = technique['object']
            if 'enterprise-attack' in object.x_mitre_domains:
                attack_vectors[stix_id].append({'stix':object.id,
                                            'id':object.external_references[0]
.external_id,
                                            'name':object.name,
                                             'malware_id':stix_id})


    return attack_vectors


def campaigns_and_techniques(mitre_attack_data):

    # get all techniques related to campaigns
                                  techniques_used_by_campaigns                  =
mitre_attack_data.get_all_techniques_used_by_all_campaigns()

    campaigns = {}

    for campaign in techniques_used_by_campaigns:
        techniques = techniques_used_by_campaigns[campaign]

        campaigns[campaign] = []

        for technique in techniques:
            technique_2 = technique['object']
            if 'enterprise-attack' in technique_2.x_mitre_domains:
                campaigns[campaign].append({'stix':technique_2.id,
                                        'id':technique_2.external_references[0
].external_id,
                                        'name':technique_2.name,
                                        'campaign_id':campaign})

    return campaigns

def datacomponents_and_techniques(mitre_attack_data):

    # get all techniques related to datacomponents
                            techniques_detected_by_datacomponents            =
mitre_attack_data.get_all_techniques_detected_by_all_datacomponents()

    datacomponents = {}


    for datacomponent in techniques_detected_by_datacomponents:
        techniques = techniques_detected_by_datacomponents[datacomponent]

        datacomponents[datacomponent] = []

        for technique in techniques:
            object = technique['object']
            if 'enterprise-attack' in object.x_mitre_domains:
                datacomponents[datacomponent].append({'stix':object.id,
                                                                     'id':object.external_referen
ces[0].external_id,
                                                                    'name':object.name,
                                                             'datacomponent_id':datacomponent})

    return datacomponents


def get_all_techniques(mitre_attack_data):
                                                           all_techniques                        =
mitre_attack_data.get_techniques(remove_revoked_deprecated=True)
    techniques = {}
    for technique in all_techniques:
        if 'enterprise-attack' in technique.x_mitre_domains:
            techniques[technique.id] = technique
    return techniques


# get the list of stix id of campaigns where specific technique is used
def get_campaigns_by_technique(mitre_attack_data, technique_id):
                                                    campaigns_using_technique                    =
mitre_attack_data.get_campaigns_using_technique(technique_id)
    campaigns = []
    for item in campaigns_using_technique:
        item = item['object']
        if 'enterprise-attack' in item.x_mitre_domains:
            campaigns.append(item.id)
    return campaigns


# get the list of stix id of software where specific technique is used
def get_software_by_technique(mitre_attack_data, technique_id):
                                                    software_using_technique                     =
mitre_attack_data.get_software_using_technique(technique_id)
    software = []
    for item in software_using_technique:
        item = item['object']
        if 'enterprise-attack' in item.x_mitre_domains:
            software.append(item.id)
    return software


# get the list of stix id of groups where specific technique is used
def get_groups_by_technique(mitre_attack_data, technique_id):
                                                      groups_using_technique                     =
mitre_attack_data.get_groups_using_technique(technique_id)
    groups = []
    for item in groups_using_technique:
        item = item['object']
        if 'enterprise-attack' in item.x_mitre_domains:
            groups.append(item.id)
    return groups


# get the list of stix id of datacomponents where specific technique is used
def get_datacomponents_by_technique(mitre_attack_data, technique_id):
                                datacomponents_using_technique             =
mitre_attack_data.get_datacomponents_detecting_technique(technique_id)
    datacomponent = []
    for item in datacomponents_using_technique:
        item = item['object']
        if 'enterprise-attack' in item.x_mitre_domains:
            datacomponent.append(item.id)
    return datacomponent


# function to get dict with specific technic as a key and list of campaigns,
software or groups as value
def get_technique_dependences(mitre_attack_data, source_name, technique_id):
    result = []
    if source_name == 'campaigns':
        result = get_campaigns_by_technique(mitre_attack_data, technique_id)
    elif source_name == 'software':
        result = get_software_by_technique(mitre_attack_data, technique_id)
    elif source_name == 'groups':
        result = get_groups_by_technique(mitre_attack_data, technique_id)
    elif source_name == 'datacomponents':
                result   =   get_datacomponents_by_technique(mitre_attack_data,
technique_id)
    else:
        Exception('Wrong source name')
    return result

def average_techniques_by_software(mitre_attack_data):

    techniques = software_and_techniques(mitre_attack_data)
    unique_technique_matrix = np.zeros((len(techniques), 1), dtype=object)
    unique_technique_matrix[:, 0] = list(techniques.values())
    techniques_counter = 0
    for i in range(len(unique_technique_matrix[:, 0])):
        techniques_counter += len(unique_technique_matrix[i][0])
            average_techniques_per_software     =     techniques_counter     /
unique_technique_matrix.size

    return round(average_techniques_per_software)

def average_techniques_by_groups(mitre_attack_data):
    techniques = groups_and_techniques(mitre_attack_data)
    unique_technique_matrix = np.zeros((len(techniques), 1), dtype=object)
    unique_technique_matrix[:, 0] = list(techniques.values())
    techniques_counter = 0
    for i in range(len(unique_technique_matrix[:, 0])):
        techniques_counter += len(unique_technique_matrix[i][0])
            average_techniques_per_group = techniques_counter /
unique_technique_matrix.size

    return round(average_techniques_per_group)

def average_techniques_by_campaigns(mitre_attack_data):

    techniques = campaigns_and_techniques(mitre_attack_data)
    unique_technique_matrix = np.zeros((len(techniques), 1), dtype=object)
    unique_technique_matrix[:, 0] = list(techniques.values())
    techniques_counter = 0
    for i in range(len(unique_technique_matrix[:, 0])):
        techniques_counter += len(unique_technique_matrix[i][0])
            average_techniques_per_campaign = techniques_counter /
unique_technique_matrix.size

    return round(average_techniques_per_campaign)

def average_techniques_by_datacomponents(mitre_attack_data):

    techniques = datacomponents_and_techniques(mitre_attack_data)
    unique_technique_matrix = np.zeros((len(techniques), 1), dtype=object)
    unique_technique_matrix[:, 0] = list(techniques.values())
    techniques_counter = 0
    for i in range(len(unique_technique_matrix[:, 0])):
        techniques_counter += len(unique_technique_matrix[i][0])
          average_techniques_per_datacomponent = techniques_counter /
unique_technique_matrix.size

    return round(average_techniques_per_datacomponent)
```

APPENDIX C. PREDICTED GRAPHS AND VISUALIZATION

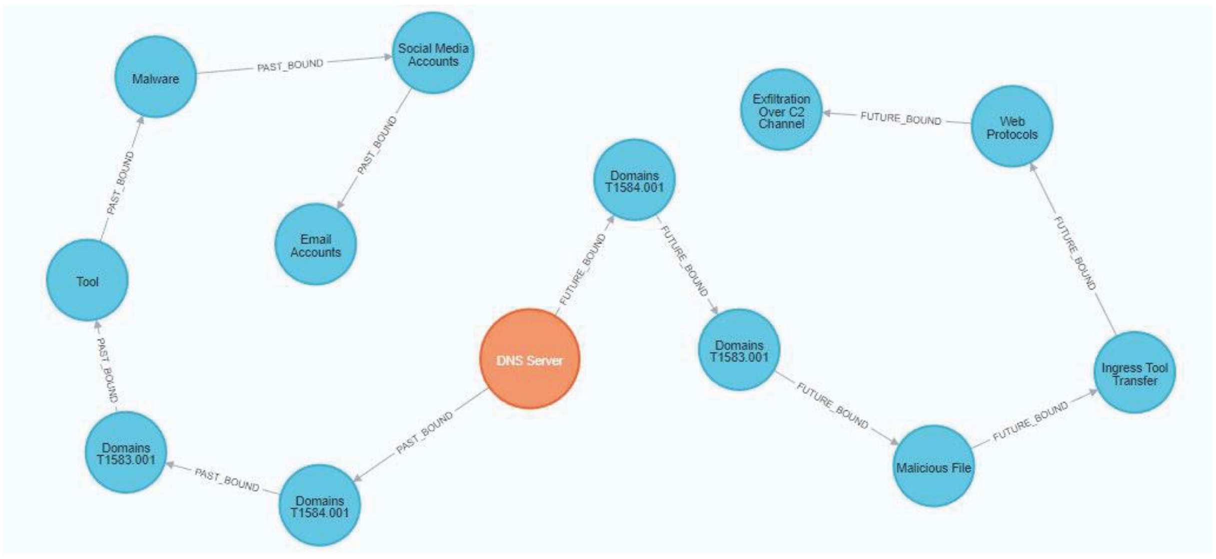
Predicting past techniques for attack pattern: Hardware T1592.001
Hardware T1592.001
Software T1592.002
Spearphishing Link T1598.003
Email Addresses T1589.002
Gather Victim Identity Information T1589

Predicting future techniques for attack pattern: Hardware T1592.001
Hardware T1592.001
Virtual Private Server T1583.003
Valid Accounts T1078
PowerShell T1059.001
Ingress Tool Transfer T1105
Web Protocols T1071.001



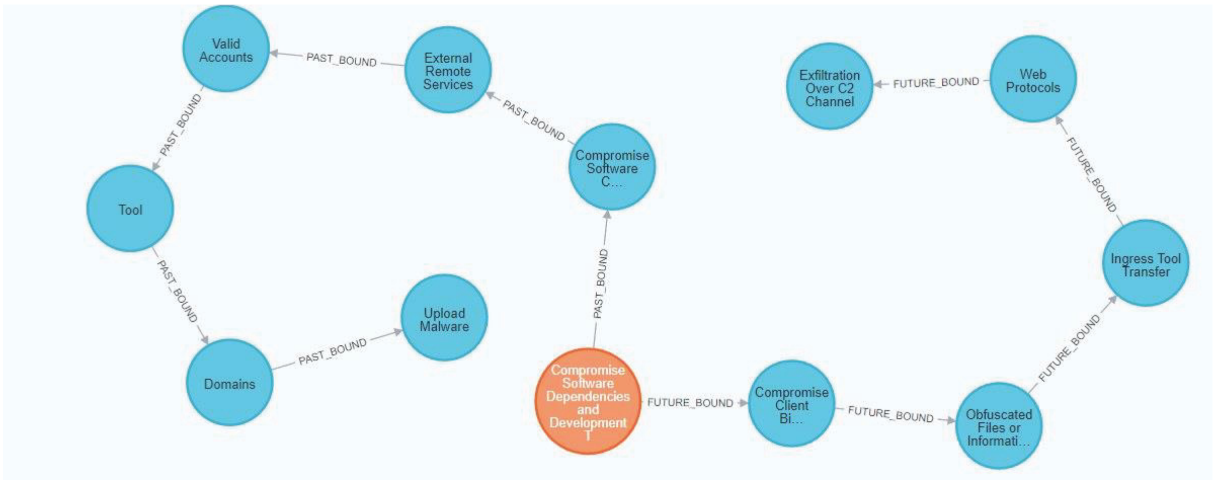
Predicting past techniques for attack pattern: DNS Server T1584.002
DNS Server T1584.002
Domains T1584.001
Domains T1583.001
Tool T1588.002
Malware T1587.001
Social Media Accounts T1585.001
Email Accounts T1585.002

Predicting future techniques for attack pattern: DNS Server T1584.002
DNS Server T1584.002
Domains T1584.001
Domains T1583.001
Malicious File T1204.002
Ingress Tool Transfer T1105
Web Protocols T1071.001
Exfiltration Over C2 Channel T1041

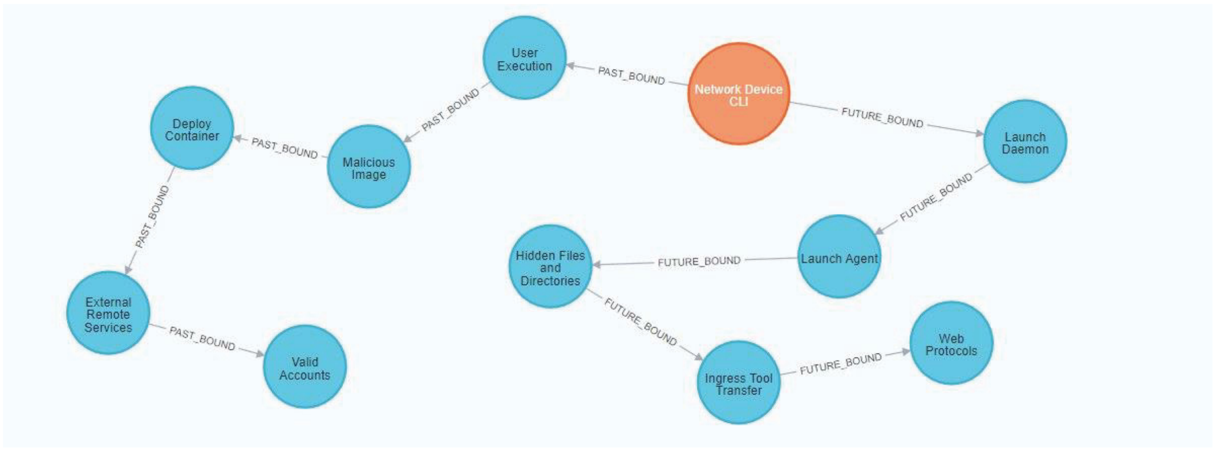


Predicting past techniques for attack pattern: Compromise Software Dependencies and Development Tools T1195.001
Compromise Software Dependencies and Development Tools T1195.001
Compromise Software Supply Chain T1195.002
External Remote Services T1133
Valid Accounts T1078
Tool T1588.002
Domains T1583.001
Upload Malware T1608.001

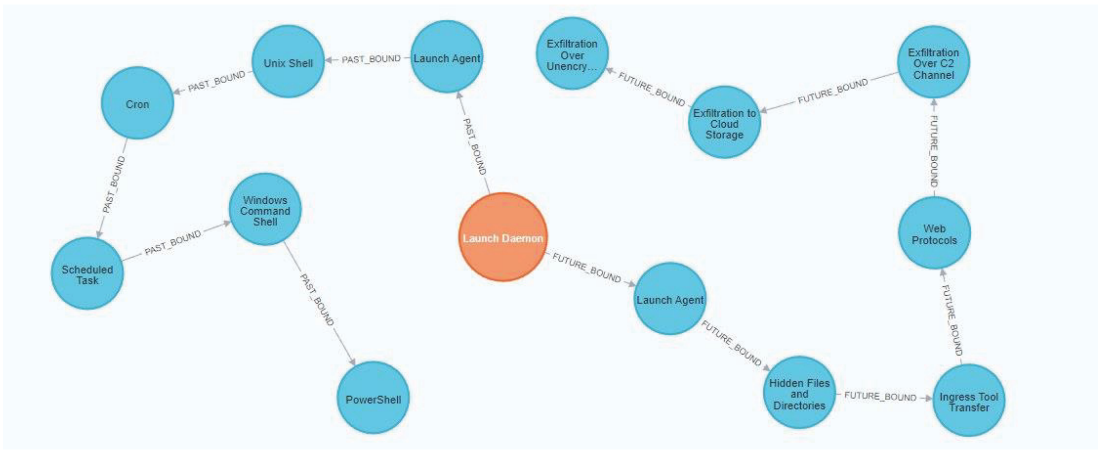
Predicting future techniques for attack pattern: Compromise Software Dependencies and Development Tools T1195.001
Compromise Software Dependencies and Development Tools T1195.001
Compromise Client Software Binary T1554
Obfuscated Files or Information T1027
Ingress Tool Transfer T1105
Web Protocols T1071.001
Exfiltration Over C2 Channel T1041



```
Predicting past techniques for attack pattern:  Network Device CLI T1059.008
Network Device CLI T1059.008
User Execution T1204
Malicious Image T1204.003
Deploy Container T1610
External Remote Services T1133
Valid Accounts T1078

Predicting future techniques for attack pattern:  Network Device CLI T1059.008
Network Device CLI T1059.008
Launch Daemon T1543.004
Launch Agent T1543.001
Hidden Files and Directories T1564.001
Ingress Tool Transfer T1105
Web Protocols T1071.001
```

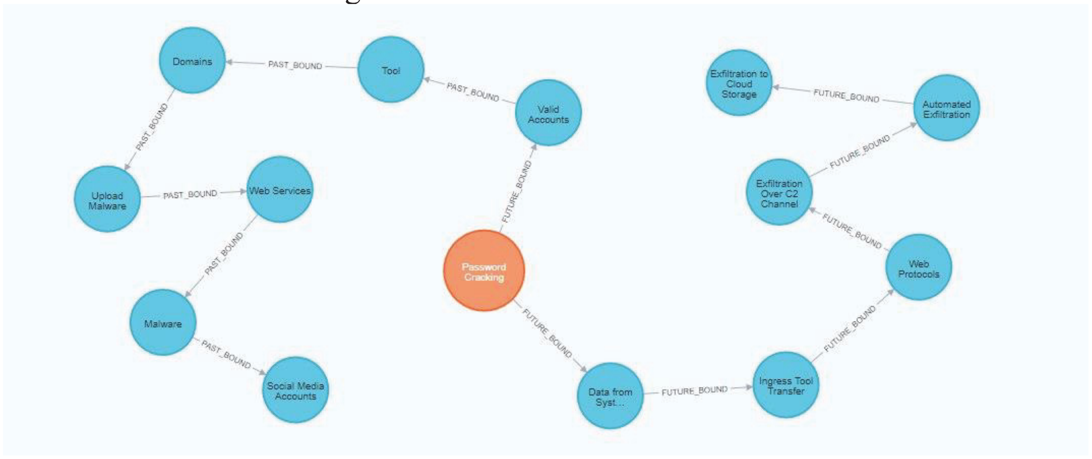


```
Predicting past techniques for attack pattern:  Launch Daemon T1543.004
Launch Daemon T1543.004
Launch Agent T1543.001
Unix Shell T1059.004
Cron T1053.003
Scheduled Task T1053.005
Windows Command Shell T1059.003
PowerShell T1059.001

Predicting future techniques for attack pattern:  Launch Daemon T1543.004
Launch Daemon T1543.004
Launch Agent T1543.001
Hidden Files and Directories T1564.001
Ingress Tool Transfer T1105
Web Protocols T1071.001
Exfiltration Over C2 Channel T1041
Exfiltration to Cloud Storage T1567.002
Exfiltration Over Unencrypted Non-C2 Protocol T1048.003
```



Predicting past techniques for attack pattern: Password Cracking T1110.002
Password Cracking T1110.002
Valid Accounts T1078
Tool T1588.002
Domains T1583.001
Upload Malware T1608.001
Web Services T1583.006
Malware T1587.001
Social Media Accounts T1585.001

Predicting future techniques for attack pattern: Password Cracking T1110.002
Password Cracking T1110.002
Data from Local System T1005
Ingress Tool Transfer T1105
Web Protocols T1071.001
Exfiltration Over C2 Channel T1041
Automated Exfiltration T1020
Exfiltration to Cloud Storage T1567.002


Password Cracking
Valid Accounts
Tool
Domains
Upload Malware
Web Services
Malware
Social Media Accounts
PAST_BOUND
FUTURE_BOUND
Data from Syst...
Ingress Tool Transfer
Web Protocols
Exfiltration Over C2 Channel
Automated Exfiltration
Exfiltration to Cloud Storage