universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„A graph-based Intrusion Detection System for defending MQTT Brokers against slow DoS Attacks“

verfasst von / submitted by

Thorsten Steuer B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik UG200
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Dr.Gerald Quirchmayr
Mitbetreut von / Co-Supervisor:	

Zusammenfassung

In den letzten zehn Jahren gab es viele Bestrebungen, um Cyber-physische Systeme zu verbessern. Dieser Trend mündete letztendlich in der Vision vom Interenet der Dinge. Das Internet der Dinge kann man sich als ein globalen dynamischen Netzwerk vorstellen, welches aus selbstkonfigurierenden Geräten besteht. Diese Geräte werden die physische und virtuelle Welt weiter miteinander verschmelzen lassen. Die Umsetzung dieser Vision dürfte sowohl für Verbraucher als auch für Unternehmen viele Vorteile bringen. Um diese Vision umzusetzten wird jedoch noch weitere Forschungsarbeit sowie zusätzliche praktische Anwednungserfahungen benötigt. Eine der wichtigsten Herausforderungen, die es zu meistern gilt, ist die Gewährleistung der Gerätesicherheit gegen Cyber-Attacken. Durch die Verbindung von materielle Vermögenswerte mit dem Internet sind diese vielen neuen Bedrohungen ausgesetzt. die zuvor nicht präsent waren. Vielen Maschienenbauingenieuren, die solche Systeme entwerfen, besitzten nich das nötige Wissen, um geeignete Maßnahmen zum Schutz vor Cyberangriffen zu entwickeln. Zur Gewährleistung der Sicherheit am Arbeitsplatz und an öffentlichen Plätzen müssen jedoch geeignete Software-Tools entwickelt werden, die den neuen Herausforderungen gewachsen sind.
Um die Netzwerksicherheit zu erhöhen können Intrusion-Detection-Systeme verwenden werden, da sie Netzwerkdaten sammeln, analysieren und geeignete Gegenmaßnahmen ergreifen sobald eine Attacke erkannt wird. Diese Systeme wurden bereits in zahlreichen Varianten entwickelt, jedoch bringt die neue Geräteheterogenität im Internet der Dinge eine Reihe neuer Herausforderungen mit sich, die gelöst werden müssen. Um zur Sicherheit zukünftiger Internetanwednungen beizutragen, präsentiert diese Thesis ein graphbasiertes Intrusion-Detection-System, welches Domänenwissen modelliert und langsame Denial of Service (DoS)-Angriffe erkennen kann, die auf Basis des weitverbreitete Message Queuing Telemetry Transport (MQTT)-Protokoll ausgeführt werden.

Abstract

In the last decade, much effort has been focused on enhancing Cyber-Physical System (CPS). This trend ultimately lead to the vision of the Internet of Things (IoT). The IoT can be seen as a future Internet that consists of a global dynamic network of self configuring devices to merge the physical and virtual world. This vision is expected to provide many advantages for consumers and companies alike. However, as for all new technology adaptions, further research work and real-world application knowledge is needed. One of the mayor challenges is securing IoT applications against malicious activities. Connecting physical assets to the Internet exposes them to many kind of threats which they have not been expose to before. Therefore, mechanical engineers designing such systems still lack the knowledge to develop appropriate measures to protect CPSs against cyber attacks. To maintain safety and security in working environments and public spaces dedicated tools need to be developed to address the newly emerged challenges.

Intrusion Detection Systems (IDSs) are one possibility to improve network security since they collect data from network traffic, analyse them and set appropriate counter measures if an intrusion is detected. Even though these system have been design in numerous variations the newly faced heterogeneity of devices in the newly envisioned Internet provides several new challenges which need to be solved. To contribute to the security and safety of future IoT environments, this thesis proposes a graph-based IDS to model domain knowledge and efficiently detect slow DoS attacks exploiting the commonly used MQTT protocol.

Contents

Acronyms

API Application Programming Interface.

ARP Address Resolution Protocol.

CAPEC Common Attack Pattern Enumerations and Classifications.

CLI Command Line Interface.

CPS Cyber-Physical System.

CPU Central Processing Unit.

CRUD Create, Read, Update and Delete.

CSV Comma Sepearted Values.

CVE Common Vulnerabilities and Exposures.

DDoS Distributed Denial of Service.

DNS Domain Name System.

DoS Denial of Service.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IDS Intrusion Detection System.

IEEE Electrical and Electronics Engineers.

IIoT Industrial Internet of Things.

IoT Internet of Things.

IoTSec IoT Security.

IP Internet Protocol.

ISO International Organization for Standardization.

JSON JavaScript Object Notation.

M2M Machine to Machine.

MB Mega Byte.

MQTT Message Queuing Telemetry Transport.

NBA Network Behavior Analysis.

OASIS Organization for the Advancement of Structured Information Standards.

OSI Open Systems Interconnection.

QOS Quality of Service.

RAM Random Access Memory.

RDF Resource Description Framework.

SlowITe Slow DoS against Internet of Things Environments.

SMTP Simple Mail Transfer Protocol.

SPARQL SPARQL Protocol and RDF Query Language.

SSH Secure Shell.

SSL/TLS Secure Sockets Layer/Transport Layer Security.

TCP Transmission Control Protocol.

VPN Virtual Private Network.

W3C World Wide Web Consortium.

WLAN Wireless Local Area Network.

XML Extensible Markup Language.

1 Introduction

The digital integration of sensor and actuators used in CPSs has been in the spotlight for the past decade and has lead to the creation of a new paradigm called IoT. The vision of IoT firstly emerged more than two decades ago and formed a conceptual umbrella for developed methods and technologies concerned with linking sensors and machine controllers to the World Wide Web[32]. The paradigm is strongly influenced by the increasing efforts to further enhance the capabilities of CPSs. The term CPS refers to a new generation of physical systems which are capable of collaborating with humans in an intuitive way [3]. In general, CPSs use new information technologies to enhance their interaction possibilities with the physical world around them. Hence, these systems aim to close the gaps between computer science and engineering disciplines. To further extend the capabilities of CPS, an essential step is to connect them to the Internet, which ultimately resulted in a new research field called IoT. Over the years, the scope of IoT further widened and different definitions have been proposed, but a precise consensus has not been reached up until today[22]. An attempt to comprehensively define the scope of IoT has been made the by European Commission [20] which defines it as:

> "A dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."

This definition is commonly adopted by researches [22] and will be used for the discussions presented in this work as well.
Based on a study conducted by McKinsey in 2015 [28], the economic value of IoT applications has the potential to reach $11 trillion. Consequently, roughly 11% of the world's economic value will be generated by the new emerging technologies enabling the paradigm shift.
Initiatives like Industry 4.0, which aim to connect production equipment with the Internet to enable more sophisticate control and data collection methods, have been adopting IoT concepts in recent years. The resulting concepts and technologies produced by these efforts have led to the creation of a new subbranch of IoT known as the Industrial Internet of Things (IIoT). However, as for IoT, a precise definition of IIoT has not yet been commonly agreed on. Boyes et al. [10] reviewed different definitions provided by the scientific community and concluded that IIoT can be defined as:

> "A system comprising networked smart objects, cyber-physical assets, associated generic information technologies and optional cloud or edge computing platforms, which enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of process, product and/or service information, within the industrial environment, so as to optimise overall production value.

This value may include; improving product or service delivery, boosting productivity, reducing labour costs, reducing energy consumption, and reducing the build-to-order cycle."

In comparison to the definition of IoT, the scope of IIoT has been chosen tighter and is linked to the overall goal in manufacturing to increase production value. Furthermore, emphasis is put on improving processes, services and products. The general goal of further closing the gap between virtual and physical assets can be found in both definitions and highlights the tight connection between both topics.
Integrating new technologies in existing real world applications always includes overcoming a variety of challenges. As highlighted by Mozzaquatro et al. [32], one of the most important challenges to tackle is securing IoT and IIoT applications from cyber attacks. A study conducted by Wolf et al. [47] emphasizes the following security and safety challenges:

- In the IoT, a wide variety of physical assets from different application domains like smart cities, autonomous vehicles, smart supply chains or smart manufacturing will have to be connected to the Internet and communicate with each other. These assets have a wide range of different hardware set ups, use different software configurations or support different protocol standards. Hence, the heterogeneity of the domain is enormous and therefore adds multiple layers of complexity.

- To ensure accurate safety and security measures, the design details of the different devices provide a crucial source of information. However, these documents are often times inconsistent or not available at all.

- CPSs usually interact in a dynamic environment with their users. To ensure CPSs only act as programmed is specifically important in domains where humans can be severely harmed by them. In the manufacturing domain, the safety of the machine operator has to be guaranteed while interacting with a CPS; otherwise, it could lead to fatal injuries.

- The life-cycle of many CPSs can be multiple decades. Providing sufficient security updates to keep these devices secure is difficult since standards change or are obsolete after some years. Keeping track of all these legacy systems is a non-trivial task. For production systems, it is even more cumbersome to provide software updates, since they can not be rebooted easily. Shutting down a production plant may require several hours or days which incurs high costs.

- Combining software and hardware specifications of CPSs with associated security knowledge is mostly cumbersome since many developers lack the necessary knowledge. Each system is unique in its configuration and architecture. Hence, mapping threats, vulnerabilities and security mechanisms to each application is time consuming and only feasible with the appropriate domain knowledge[32].

These challenges raise the need for methods capable of covering a wide range of different application use cases[47]. According to Wolf et al. [47], applying model-based approaches during the design time of applications has the potential to increase safety and security significantly. It will enable precise safety and security property analysis based on semantically rich context information. Furthermore, securing protocols is as well highlighted as crucial for the further success of IoT and IIoT. Even though there is a increasing demand to provide build-in security mechanism in protocol standards, institutions and organizations, which develop these standards, are still trying to adapt to the fast increasing requirements. For example the MQTT protocol, which is widely adopted in IoT and IIoT environments, relies mainly on the underlying security layer provided by Secure Sockets Layer/Transport Layer Security (SSL/TLS). However, recent discoveries by Vaccari et al. [44] prove that these security layers are often not sufficient to mitigate protocol specific vulnerabilities.
The presented discussion motivated this thesis to propose a context-aware IDS to provide network system, service and protocol specific information to secure MQTT brokers against slow DoS attacks. With this thesis, the author aims to support the scientific community with a validation test bed and an approach to combine network data with vulnerability data in a semantically meaningful way. The presented concepts aim to contribute to a more secure future of IoT and IIoT environments.

1.1 Outline of the Thesis

The thesis is structured as follows. The remainder of section 1 presents necessary background knowledge to interpret and understand the introduced concepts. In section 2, related work is reviewed to cover recent proposed approaches concerned with graph-based intrusion detection. Afterwards, section 1.2.2 highlights specific features and types of slow DoS attacks. Additionally, the section introduces the Slow DoS against Internet of Things Environments (SlowITe) attack which is taken as a validation example to evaluate the proposed IDS. Section 3 introduces the system architecture used to implement a prototype of an IDS to validate the proposed data model. Additionally, a test bed has been implemented to simulate senors data and provide a network environment to verify the capabilities of the proposed IDS. The implementation details of the test bed are introduced in section 4. In section 5, the prototypical implementation of the IDS are presented with special emphasis on the proposed graph data model developed to highlight the main contribution of the thesis. Subsequently, the implemented modules were tested by deploying them in the test bed. The results are introduced in section 6. In section 7, the introduced concepts as well as the result of the system capability test are discussed and further development options are suggested. The last section 8 concludes the thesis and suggests directions for future work.

1.2 Technology Background

1.2.1 Denial of Service Attacks

The goal of DoS attacks is to deny access to a victim system over a certain amount of time. As a synonym, the objective is commonly also described as reaching a DoS state on the target system. The attack is carried out by directly or indirectly exhausting resources of a victim system. Typical resources exploited by an adversary are memory and disk space, connection limits and network resources [52]. DoS attacks can cause serious consequences ranging from decreasing customer satisfaction to putting human lives in jeopardy. There are many real-world examples demonstrating the damage such attacks can cause. One recent example is the massive DoS attack reported by the company Imperva[1]. The attack lasted 13 days and involved 400 thousand different IP addresses and flooded the network at its peak with 500 million packets per second. The attack targeted the authentication mechanisms of a website to neglect customers' access to the service [42]. It was the largest DoS attack recorded by Imperva and according to internal analysis has been enabled by exploiting IoT devices. To perform large scale attacks, like the previously mentioned one, adversaries need to allocate large volumes of computational power. Usually, this is achieved by distributing the workload on a large number of previously compromised machines. Consequently, it is categorized as a Distributed Denial of Service (DDoS) attack. A network built of compromised computers is called a botnet [52]. Even though DDoS attacks are the most commonly known and observed DoS attacks, there is a large variety of DoS attacks. Some of them are even designed to be executed from resource constrained devices such as a Raspberry Pi and therefore have no need to firstly construct a botnet to reach the DoS state [44].
Securing systems against these attacks is difficult not only since implementations vary in the magnitude of the involved resources, but also because there are various procedures to generate botnets and methods to exploit different kinds of computer resources. A number of attempts have been carried out by researchers to create a comprehensive topology to classify DoS attacks. One simplistic but popular approach to structure attack patterns is used by Zlomisli et al. [52]. They use the exploited vulnerabilities targeted by an adversary to classify different DoS attacks. Four major liabilities have been identified:

- **Flood Attacks** are characterized by the a vast amount of communication requests sent to a target system. A well known example is the SYN flood attack, which exhausts the target resources by sending Transmission Control Protocol (TCP) SYN requests [9]. The requests are sent from previously spoofed[2] Internet Protocol (IP) client addresses, which do not

[1] `https://www.imperva.com/`

[2] Spoofing refers to a method where an adversary impersonates another computer or person by providing false information. One example to gain access to IP addresses is a technique known as Address Resolution Protocol (ARP) Spoofing. For more information consult the reference [2].

exist at the time of the attack. Therefore, the final ACK[3] message from the victim host will never be send which results in half-opened connections. These half-opened connections are stored in a backlog of the host and bind computational resources of the server. If too many request are sent, the server's resources will eventually be exhausted, and new connections from any host, even legitimate ones, cannot be established anymore.

- **Amplification Attacks** also focus on flooding a victim system with requests but use a more resource efficient approach to reach their goal. As the name suggests, an amplifier is used which in this context is a certain property of a protocol or system to trigger responses significantly larger than the initial transmitted request. The reflective Domain Name System (DNS) attack is part of this family of DoS attacks. First attackers spoof the IP address of a victim, which will redirect all replies from the DNS server to the target system [37]. Hence, the adversary reflects the data traffic from the DNS server to the victim. After the reflection is successful, the attacker scans the responses to determine the ones which have a significant higher response volume than their corresponding request. The higher the difference between the data volume of the request and the response, the greater the amplification factor. If an appropriate amplification request is discovered, the victim server is targeted with a large amount of these requests resulting in a DoS state since processing and preparing the response consumes all computational resources.

- **Protocol Vulnerability Exploitations** are individually tailored to a specific protocol. Vaccari et al. [44] discovered such a vulnerability in the MQTT[4] protocol. The objective of the attack is to occupy all simultaneously maintainable connections of a MQTT broker. In this specific attack scenario, the attack exploits the fact that the keep-alive parameter of the MQTT connection can be set by the client. Therefore, the broker can be flooded with connection requests and will keep them alive nearly as long as the attacker desires. These types of attacks are usually resource efficient for the adversary since exploiting design mistakes included in protocols standards can often be performed with considerably low computational resources.

- **Malformed Packet Attacks** rely on incorrectly formed IP packets to crash a victim's operation system. One way to execute such an attack is to provide each IP packet with the same source and destination IP address [17]. This may result in an abnormal behavior of the system or a crash since the operating system does not know how to handle these packets.

Other scientific papers like Ramanauskaite and Cenys [35] or Dougliergies and Mitrokotsa [17] provide a multidimensional approach to classify DoS attacks.

[3] ACK is an acronym for ACKNOWLEDGE. These type of messages are used to verify the arrival of a message.

[4] For further information about MQTT refer to section 1.2.5.

Even though these papers provide a broad coverage of the domain to follow the concepts introduced in this thesis, the presented attack types are sufficient to follow the argumentation line. Furthermore, in the following subsections the slow DoS and SlowITe attacks will be introduced. Preventing such types of DoS attacks in MQTT networks is the goal of this thesis.

1.2.2 Slow DoS Attacks

As already mentioned in subsection 1.2.1, DoS attacks aim to reduce or completely deny access to a target system by directly or indirectly exhausting resources like Random Access Memory (RAM), Central Processing Unit (CPU), network bandwidth etc. The same applies to slow[5] DoS attacks. However, the procedure to reach that goal are different. The DoS attacks described in subsection 1.2.1 commonly exploit transport layer protocols[6] and send large amounts of bytes to the victim system to reach a DoS state. On the other hand, slow DoS attacks typically exploit vulnerabilities in application layer protocols and are operating on a relatively low byte rate to reach a DoS state. Consequently, these types of attacks are harder to detect and can even be executed by resource constrained devices like mobile phones [12]. SlowITe [44] is a example of a slow DoS attack which aims to deny access to MQTT brokers by exploiting the *KeepAlive* parameter set by a client[7].
According to Cambiaso et al. [12], slow DoS attacks can be categorized into the following six groups:

- **Delayed Responses** characterize attacks which send legitimate requests to a server to trigger a computational expensive response. By flooding a server with such requests, a DoS state can be reached little device resources. An example of such an attack is THC-SSL-DOS which exploits the vulnerability of SSL/TLS handshakes requiring more server side resources than the client side. While a server can process on average 300 handshakes per second, a single client can initiate up to 1000 per second [12].

- **Resource Occupation Planning** attacks attempt to estimate when a server resource is freed to immediately occupy it again to deny access for legitimate users. These types of attacks typically launch a short bulk of requests if connections time out to occupy them again but do not send any or only limited amount of data while the connection is alive. In consequence, the moving average of the occupied bit rate is low and hard to detect. The Shrew attack, for example, sends a bulk of data periodically

[5]The term slow does not necessarily imply that these attacks transmit data slowly. The name was coined by the well known SlowLoris attack. For more information about SlowLoris refer to [15].

[6]For further information about the different network layers, consult the Open Systems Interconnection (OSI) Reference Model [51].

[7]SlowITe will be introduced in detail in subsection 1.2.3.

in synchronization with the retransmission timeout of a target router to occupy as much connections as possible and deny legitimate user access.

- **Resource distortion DoS** attacks deceive a target server about the attacker's resource's capabilities. One example of such an attack is the Slow Read attack which utilises the vulnerability of web servers not closing the server connection as long as data flows. Additionally, the attacker device falsely signals a small receptions buffer which results in the server replying very slowly to not overflow the buffer of the recipient. By opening as many connections as possible, an adversary can block new legitimate connections from being processed and therefore causing a DoS state on the target server.

- **Long Requests** attacks send incomplete requests to a host to occupy the buffer memory while the target server waits for the completion of the request. An example is the SlowLoris attack which never completes the Hypertext Transfer Protocol (HTTP) request sent to a host by neglecting the newline parameter needed to terminate a HTTP request. The DoS state can be reached by keeping the connection alive as long as possible and occupy all connections a server concurrently can handle.

- **Network Oriented** attacks are targeting the network infrastructure instead of the server's resources as the previous discussed attacks. One example of such an attack is the Quiet attack which includes a reconnaissance phase to identify a botnet, target routers, web servers and a victim. The second phase, the execution phase, is characterized by the botnet repeatedly requesting a website from the identified web servers. To disguise the request as being legitimate, the traffic elapses at a random timeout to request afterwards a different web page from the same web servers. Finally, in the network feedback gathering phase, the adversary analyzes how much resources are needed to congest the link and block it for legitimate request.

As can be seen, slow DoS attacks can have a variety of techniques to deny access to system resources or network infrastructures. This rather new category of DoS attacks provides a substantial risk for IoT and IIoT applications since the widely adopted MQTT protocol can also be exploited by a slow DoS attack called SlowITe.

1.2.3 SlowITe Attack

As IoT and IIoT applications are becoming increasingly popular, adversaries expand their efforts to specially target technologies and protocols used within the space. Recently, a new vulnerability of the widely adopted MQTT protocol was discovered by Vaccari et al. [44]. They present the SlowITe attack which takes advantages of the the *Keep-Alive* parameter defined in the MQTT protocol standard. The attack is currently listen under the reference number

CVE-2020-13849[8] in the Common Vulnerabilities and Exposures list maintained by the MITRE Corporation[9]. The goal of the attack is to establish a high number of connections with a MQTT broker until it can not process the amount of concurrent connections anymore.
To illustrate the mechanism behind the SlowITe attack a single connection attempt is used. To authenticate and establish a connection with a MQTT broker, the client needs to send a *CONNECT* package which has to be acknowledged by the server via the *CONNACK* package. Once the connection is established, the server hast to keep the connection alive for at least 1.5 times the Keep-Alive parameter. By default, this value is set to be 60 seconds. Hence, by sending a single *CONNECT* package it is possible for the adversary to occupy a connection for 90 seconds. To reach a DoS state, an adversary just needs to replicate as many connection attempts until all free connection are seized. The number of simultaneously manageable connections is dependent on the MQTT protocol implementation and its instance configuration. A popular implementation of the MQTT protocol is Mosquitto[10] which by default sets the maximum number of possible concurrent connections to 1024 for each broker.[11].
As shown by Vaccari et al. [44], the attack can be preformed by resource constrained devices like a Raspberry PI 3 Model B and takes only three seconds until the DoS state on the Mosquitto broker can be reached. Their results also verified the possibility to maintain the DoS state approximately 1.5 times longer than the provided *Keep-Alive* parameter. In a second scenario, they further exploited the MQTT protocol by utilizing the possibility of the client to set a user defined *Keep-Alive* parameter. The maximum value for the parameter is 65,535 seconds. Some server implementations even allow to void the *Keep-Alive* parameter, leading to an infinite connection time. By sending the maximum value of the *Keep-Alive* parameter, they showed that all connections were opened for 27 hours and 18 minutes, hence denying access to the broker during this time.
The results presented by Vaccari et al. [44] highlight the efficiency of the proposed attack and demonstrate the threat of the SlowITe attack for IoT and IIoT applications relying on the MQTT protocol.
In the following subsections IDSs and their different variations will be introduced.

1.2.4 Intrusion Detection Systems

IDSs are designed to monitor events and reveal security incidences in computer systems and networks. An incidence can be caused by various events, such as spoofing IP addresses, phishing mails, DoS attacks, unauthorized access to the system, misuse by an authorized user etc. Generally, IDSs attempt to detect

[8] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-13849`

[9] MITRE is an American non-profit organization covering aviation, defense, healthcare, homeland security and cybersecurity topics `https://www.mitre.org/`

[10] `https://mosquitto.org/`

[11] For more details refer to the source code `https://github.com/eclipse/mosquitto/blob/master/mosquitto.conf`

anomalies in the confidentiality, integrity or availability of a system under investigation [27]. Alongside detecting undesirable activities, IDSs also record related information from incidences, notify security administrators and produce detailed reports about events of interest.
There are many different possibilities to implement IDSs, but they generally consist of four components: sensors or agents which monitor and evaluate activities, management servers which receive information from agents and sensors and provide tools to supervise them, database servers to store the recorded event history and consoles which provide an interface for users to configure the system. Since all these components can be differently integrated into an existing system, many attempts have been carried out to provide comprehensive overviews of IDSs. Two examples are the work of Liao et al. [27] and Scarfone and Mell [38]. Whereas the former focus on reviewing the scientific research domain and creating a taxonomy the latter provide a guideline to design, implement and maintain IDSs. Both papers organize IDSs by the detection methodologies and technology types used. In the following paragraphs these categories are introduced in detail.
Detection methodologies can be categorized into three main groups: signature-based, anomaly-based, and stateful protocol analysis. IDSs usually adopt a combination of these methodologies to leverage the advantages of each approach. The approaches can be distinguished as follows:

- **Signature-based detection** is performed by developing representation patterns of known threats. Theses patterns are also called the signature of a threat. To detect an intrusion, the signatures are compared to observed system events if an anomaly is detected adequate mitigation action can be triggered. An example of an attack signature is an e-mail containing a subject called "Free pictures!" and an attachment named "freepics.exe". This e-mail pattern could be used to infect computers with malware. Signature-based methods are an effective method to detect known threats since they are specifically built for each threat. Hence, they are also called knowledge-based detection methods. Despite of being simple to implement and effective, signature-based techniques are hard to maintain and are unable to detect new, unknown attacks. Only threats which have been identified and had their corresponding pattern included in the knowledge base are detectable. Variants or mutations cannot be discovered.

- **Anomaly-based detection** can be carried out by defining regular or acceptable network traffic occurring on usual working days. Deviations from these standards can than be classified with statistical methods as anomalies. To describe regular network traffic, anomaly-based detection methods utilise profiles. An example of a profile is the description of the average occupied bandwidth of a Wireless Access Point on a conventional working day. Profiles can be constructed for various events such as authentication attempts, established network connections, host configurations, application status etc. To identify threats, profiles are additionally associated

with thresholds. By applying statistical techniques, anomaly-based detection methods determine if the current footprint of a defined profile is within the defined threshold boundaries. Profiles can change over time, consequently they can be either static or dynamic. Whereas the latter is usually adjusting itself automatically, static profiles need to be changed manually. Even though anomaly-detection based detection is popular, it also has the drawback of being exposed to evasion attacks from adversaries. Evasion attacks are characterized by slowly increasing their attack volume to trick the system in adjusting the thresholds slowly over time, or the malicious activity is unintentionally considered as conventional traffic and is therefore included in a static profile. Creating suitable profiles is the main challenge in anomaly-based detection methods since computing activities can become very complex with increasing network size. Even in small networks, it might be difficult to design suitable profiles since it can happen that employees occasionally transfer large files, which shortly increases the network traffic and most likely will be falsely identified as an intrusion. Another difficulty is to determine the cause of the alarm trigger since profiles are complex constructs of various events correlated with each other and a combination of different events mostly causes an intrusion alarm. However, anomaly-based methods are effective to detect previously unknown attacks and are widely adopted in the domain.

- **Stateful protocol** analysis is also based on comparing profile definitions with actual network events but, contrary to anomaly-based detection, profiles are predefined by vendor-specific protocol specifications. Specifications define how protocols should be applied, how they react when different events occur and in which idle states they can remain. Stateful protocol analysis enables IDSs to understand in which state a protocol is at a given time. For example, a protocol could be in the connection establishment state. Each protocol defines a procedure how client and server applications establish a connection. A series of request an respond messages is usually exchanged. These request-respond pairs are well distinguishable from other messages, hence they can be taken as identifiers for the current protocol state. IDSs relying on this detection approach could trigger an alarm if the same command message is sent repeatedly or if commands are transmitted in the wrong order. The main drawbacks of these approaches are their computationally intensive procedures of tracking and inferring protocol states. Furthermore, protocol analysis techniques do not protect systems against DoS attacks. They can be designed to adhere to the protocol specification, but the volume of the messages sent exhausts the resources of the victim. Another disadvantage is the difficulty of keeping the profiles up to date. Specifications change frequently and vendor or internal enhancements of protocol implementations are very common.

Technology types are characterized by the types of events they can monitor and the methods used to identify incidences. Although there are many different

technologies, a common approach is to structure the types into the following four groups:

- **Network-based IDSs** are dedicated to monitor network traffic of application, transport and network protocols[12]. These systems are usually deployed at the point of contact between two different networks. Such contact points can be firewalls, routers, remote access servers or wireless network access points. Network-based IDSs use two formats of sensors to capture network traffic. Whereas appliance-based sensors have a matching hardware and software architecture, the so called "software only" sensors are not shipped with dedicated hardware components. Additionally, sensors can be distinguished by the location of their deployment. Inline sensors require the network traffic to pass through them while a passive deployment of sensors monitor a copy of the actual network traffic.
  Despite of the fact that network-based IDSs can detect a broad range of malicious activities, they have several disadvantages. They are associated with high false positive and false negative detection rates which is caused by the complexity of host and client traffic a single sensor has to analyze. Furthermore, based on the heterogeneity of network traffic, a long customization time is usually required to tune detection processes and configure the IDS to meet the network requirements. Network-based approaches are also unable to evaluate encrypted traffic like Virtual Private Network (VPN) connections, Hypertext Transfer Protocol Secure (HTTPS) and Secure Shell (SSH) sessions. They can have issues performing analysis during peak traffic times since the processing time of evaluating the traffic is computationally intensive. This characteristic of network-based IDSs exposes them as targets for DoS attacks.

- **Wireless-based IDSs** monitor wireless network traffic and their corresponding networking protocols. Wireless networks enable devices within a certain range to connect to the network infrastructure. The most familiar type of a wireless network is Wireless Local Area Network (WLAN) which enables a group of network devices to exchange data via radio communication. Usually WLANs are implemented based on the Electrical and Electronics Engineers (IEEE) 802.11 standard family. Generally, a WLAN consists of stations, which are endpoint devices like laptop, mobile phones etc., and access points, which connects the stations with the wired network infrastructures. Devices can use WLAN also to communicate directly to each other without an access point. This mode is known as peer-to-peer or ad-hoc networking.
  Wired and wireless networks face the same types of threats as other notworks with one significant difference: Attackers mostly need to place the attacker device in close physical proximity to the wireless network, otherwise they cannot gain access. Even though this represents a significant

[12]For further information about the different network layers consult the OSI Reference Model [51].

obstacle for adversaries, once a malicious device is placed appropriately, it is normally easy to perform a wide range of attacks on WLANs since they use weak forms of authentication. To detect such intrusions, wireless-based IDSs rely on the same components as network-based IDSs. All components provide similar functionality for both approaches except for the sensors. Monitoring wireless networking traffic is a complex task since they can operate on two frequency bands, and each of these bands has multiple channels. This is a challenge because one sensor is not able to track networks over different channels simultaneously. To avoid setting up sensors for each channel, a common approach is to preform channel scanning. Channel scanning is a technique to switch between channels based on a specified frequency rate to evaluate each channel a few times per second. There are different types of sensors available for wireless-based IDSs. Dedicated sensors are most commonly passive and operate in a predefined radio frequency range to sniff wireless network traffic. Sensors bundled with access points or switches provide less comprehensive detection capabilities due to the necessity of balancing resources between providing network access and monitoring multiple channels simultaneously. Considering the main purpose of a switch is assisting in administering wireless devices, integrated sensors typically offer less detection capabilities. The deployment location of a wireless-based IDS sensor should be carefully evaluated since they have to monitor different radio frequency ranges used in the organization, they need to monitor WLAN activity where it is expected and where not and they have to cope with a variety of endpoint devices. Sensors can operate effectively in hallways, conference rooms or other open access areas to evaluate a wide range of geographical space. Such sensors are potentially exposed to physical damage by an intruder. Hence, they are usually designed to withstand physical threats. Another important aspect to consider when placing a sensors is the space range they can cover. Since radio frequency coverage can be effected by walls, doors, office equipment etc., sensors are placed to have a certain percentage of overlap with each other. Additional, in large production or campus environments, a trade-off between the number of sensors and the costs have to be achieved.
Although wireless IDSs provide comprehensive detection features, they also have their limitations. Wireless-based IDS sensors are generally vulnerable against evasion techniques. Attackers can use different approaches to identify the concrete sensor hardware in use and design an attack which bypasses the channel scanning scheme of the specific product. Furthermore, sensors can be the victim of DoS attacks by jamming the WLAN or they can, if accessible, be physically destroyed. Another disadvantage is the inability to detect passive monitoring and offline processing of wireless network traffic.

- IDSs using Network Behavior Analysis (NBA) observe network traffic and statistics to detect suspicious traffic flows. NBA most commonly

uses anomaly-detection to identify DDoS attacks, worms spreading among hosts and security policy violations. Whereas wireless-based and network-based IDSs usually consist of management servers, database servers, consoles and sensors, NBA systems generally only provide sensors and a console. Some sensors of NBA based IDSs operate similarly to network-based sensors and sniff network packets from chosen network segments while others are dedicated to monitor network flows of sessions between hosts. The kind of information provided by such sensors is defined by various standards such as NetFLow [13]. Most sensors are passive appliance sensors integrated in routers, switches and other network devices. These sensors are most commonly located at dividing points between networks and key network areas.
Since NBA systems are largely based on anomaly-based detection approaches, they are efficient in detecting significant deviations from the baseline traffic behavior but are not well suited to identify small-scale attacks or attacks which do not violate any security policy. Furthermore, detection delays caused by the provision of traffic flow data is a common issue. Data is usually transferred in batches from the sensors to the IDS. Depending on the available network bandwidth and the interval settings, a sensors batch transmission interval can vary from every minute to a few times per hour. Thus, fast performed attacks, such as DoS, may not be recognized before they already disrupted the system.

- Host-based IDSs evaluate wired and wireless network traffic, system logs, active processes, user activities and changes in system configurations of a single host machine. In comparison to the previously mentioned technologies, host-based IDSs use agents instead of sensors to collect and analyze data. Agents are software applications directly installed on the host machine. The agents forward the collected data to a management server, which stores the received events on a database server. Additionally, consoles are provided to administer and monitor the system. Agents are not only implemented to monitor operating systems and running applications on a server, they can also be designed to operate on laptops or work stations which run user specific applications like e-mail clients or web browsers. Some agents are even specifically programmed to monitor a single instance of a service like a database access entry point. Agents are usually deployed at critical infrastructure hosts, at publicly available hosts or at hosts holding sensitive information. Nevertheless, agents are available for a wide range of server and workstation operation systems. Hence, organizations usually deploy them on most of their devices. Additionally, agents installed on encryption endpoints are capable of also monitoring the unencrypted activities before passing through the endpoint.
  In general, host-based IDSs require a considerable long time to customize for a specific system. The tuning process intents to reduce the considerable high false positive and false negative rates these systems usually have. Increasing the accuracy rate for host-based IDSs is a complex task

since the evaluation of an event being malicious or not heavily depends on the context it occurs in. For example, installing an application on a host and rebooting it could be done by a scheduled maintenance operation or could have been performed by an adversary. The events can only be correctly identified if the right context information is provided as well. NBA systems usually generate a reporting delay once an incident is detected since they evaluate historical event logs to detect malicious activities. The delay may significantly increase if the predefined frequency rate to transmit data to the management server is too low. Furthermore, contrary to other IDS technologies, agents deployed on hosts can consume a significant amount of resources which can cause a reduction of their availability to other services running on the host.

By introducing the main characteristics of IDSs, it is evident that each approach has it owns advantages and disadvantages. Hence, organizations usually combine multiple approaches to create a comprehensive detection system with a reasonable detection accuracy. In the next subsection, further background knowledge is provided about the MQTT protocol which is widely adopted in IoT and IIoT environments.

1.2.5 Message Queuing Telemetry Transport Protocol

MQTT is a transport protocol originally designed by IBM [25] in 1999. In 2014, it has been revised and published as a Organization for the Advancement of Structured Information Standards (OASIS) standard for Machine to Machine (M2M) communication and in 2016 recognized as an International Organization for Standardization (ISO) standard. The current version 5.0 of the specification defines the protocol as „light weight, open, simple, and designed to be easy to implement“ ([4]). These features enabled MQTT to be the most popular IoT protocol in research and industry applications [30]. Presently, version 3.1.1 and 5.0 are most commonly adopted. Version 5.0 was published in 2019 and includes new updates for better error handling, higher scalability and greater flexibility. In the following paragraph, the general functionality and features of version 3.1.1 are introduced to provide the foundation to understand the key enhancements of version 5.0.
MQTT is based on the client server publish/subscribe pattern. A MQTT network involves two types of agents: clients and brokers. The clients can be divided further into publishers, also known as producers, and subscribers, also known as consumers. Publishers, typically sensors, actuators etc., broadcast data to a broker, where the data can be consumed by a subscriber. The MQTT broker can be seen therefore as intermediary between publishers and subscribers. To transmit data, the protocol uses so-called application messages. These messages are transported trough the TCP/IP protocol. For encryption and authentication, port number 8883 is registered to establish SSL/TLS connections. Port number 1883 instead is used for plain text transmission [5].
Figure 1 shows a simple communication scenario between two MQTT clients and

an MQTT broker. We assume each client is implemented on a different device, one acts as publisher while the other as subscriber[13]. To establish a connection

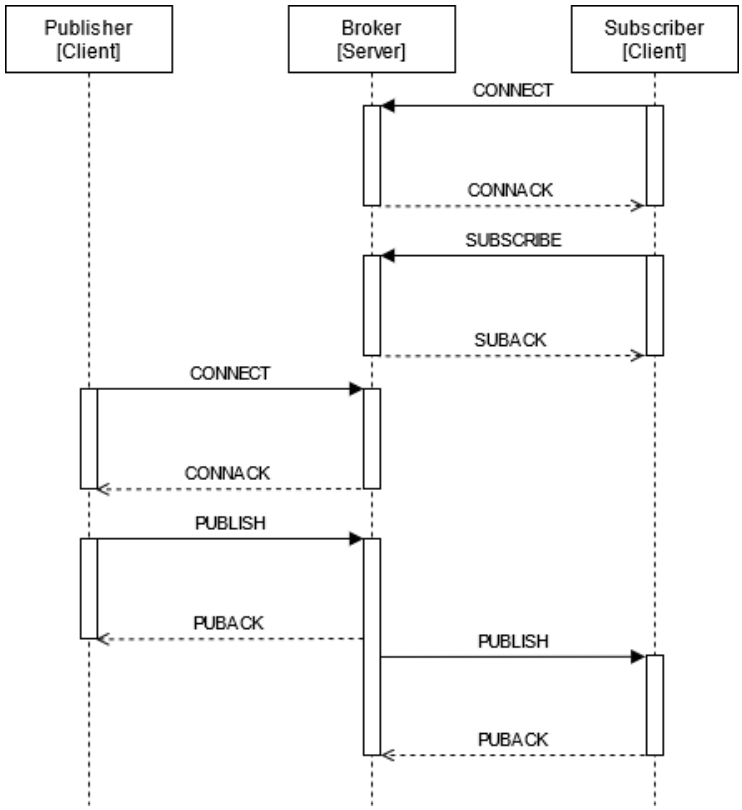

Figure 1: Example of a communication scenario between different agents in an MQTT network [33].

to a broker, the client must transmit a *CONNECT* message to submit at least a unique identifier for the device. Additional parameters like user name, password, keep alive etc. can also be forwarded if required. The broker will return a *CONNACK* message indicating success or failure of the connection attempt. Since we assumed, that the clients are two distinguishable devices, both need to follow this procedure. To publish data, clients need to send a PUBLISH packet containing a topic name and a Quality of Service (QOS) level to the broker. Topic names are used by the broker to determine which information channel should be updated by the incoming data. MQTT enables agents to specify the level of assurance for data delivery. Based on the use case, a user might have the requirement to avoid duplicated application messages, hence they will demand a QOS level of 2. Unfortunately, a higher QOS involves a higher communication

[13]Each device in an MQTT network usually acts as a subscriber and publisher simultaneously. For the sake of simplicity, this case was not considered in this scenario.

overhead since more control packets need to be exchange between agents. The following QOS levels can be used:

- **QOS 0:** There is no acknowledgement from the receiver.
- **QOS 1:** Ensures the messages have arrived at least once at the receiver. The application messages are acknowledged by a *PUBACK* packet.
- **QOS 2:** Enforces messages to be delivered exactly once. Four packets need to be exchanged *PUBLISH*, *PUBREC*, *PUBREL* and *PUBCOMP*.

In figure 1, a QOS of 1 was chosen, as can be seen by the returned *PUBACK* packets. In order to receive data from the broker, the subscribing device needs to sends a *SUBSCRIBE* packet. The message must contain at least one topic filter as well as the associated QOS level. The filter is used to indicate to which topics the client wishes to subscribe, whereas the requested QOS specifies the maximum QOS level the broker has to guarantee to provide the subscriber with data. The broker must confirm the subscription by returning a *SUBACK* packet. This packet contains for each topic filter QOS pair a code indicating which QOS has been granted or if the subscription attempt has failed. Once the subscription has been registered successfully[14] the broker can forward application messages to the client. In addition to the introduced control packets in figure 1, *UNSUBCRIBE*, *PINGREQ* and *DISCONNECT* packets can be used to unsubscribe from topics, indicate that a client is alive and terminate the connection, respectively.
As described by Vaccari et al. [45], MQTT version 5.0 can be seen as a refinement which enhances the protocol with new features while maintaining the core functionalities of the previously introduced 3.1.1 version. In the following, some functional enrichments are highlighted:

- An improved error reporting by adding reason code along with the *DISCONNECT* packet if a malformed packet or protocol error is detected by the client.
- A shared subscriptions feature to distribute traffic load between several clients to process data in parallel.
- Properties have been introduced to enable a request-response interaction between clients and brokers and to add the possibility of attaching user specific information to control packets.
- The maximal packet size and the number of messages transmitted can be configured.
- The authentication algorithm has been enhanced to use a challenge-response mechanism or to include mutual authentication.

[14]The broker is allowed to send application messages before returning the *SUBACK* packet to the client [5]. However, to simplify the communication scenario it was dismissed from figure 1.

- A maximum keep alive parameter can be set by the broker.
- Session expiry intervals are introduced to retain a session during network disconnection.

The new version has introduced more features than listed above. However, based on the fact that version 5.0 is not yet widely adopted and none of the new functional improvements dismisses the need for IDSs, providing further details is out of scope of this thesis.
After introducing the protocol under investigation, the following section will introduce graph databases and their linkage to knowledge management. The section will highlight the possibility to use the label property graph data model to capture context information.

1.2.6 Graph Databases and Label Property Graphs

In recent years, graph databases have become a popular technology to support a variety of applications like fraud detection, social network analysis, recommendation engines etc. They belong to the group of the so-called NoSQL database systems. NoSQL database support a variety of data models. For example, GraphDB[15] supports the Resource Description Framework (RDF) data model whereas Neo4j[16] is based on the label property graph model[17] [8]. Each model has it unique advantages and disadvantages, but native graph databases are specifically designed and optimized to store and query graph data. In the following paragraphs, they will be introduced in detail to provide sufficient knowledge to follow the remainder of this thesis.
Native graph databases most commonly adopt the label property graph data model to store data. A label property graph is a labelled directed multigraph. A graph G can be described by a tuple (V, E) [8]. V contains a set of vertices (also known as nodes or entities) while E is a set of edges (also known as relations) connecting vertices with each other. Hence, a graph can be denoted as $G = (V, E)$. An edge is defined as a tuple of two vertices $e = (u, v) \in E$, if u is defined as the source whereas v is defined as the destination. The corresponding graph is called directed. Additionally, a graph can be weighted if it is modeled as triple (V, E, w), where $w : E \to R$ assigns edges to weights.
This simple graph model is most commonly not used by graph databases since it is not expressive enough to model various real-world problems. Consequently, many native graph databases adopt a extensions called label property graph. A label property graph uses labels to distinguish between different subsets of vertices and edges. Additionally, each vertex and edge can have multiple properties, also known as attributes. Properties are key-value pairs where the key is the identifier for the value and the value holds the corresponding data instance. A label property graph can formally be described as a tuple

[15] `https://www.ontotext.com/products/graphdb/`
[16] `https://neo4j.com/`
[17] Neo4j adopts a variant of the label property graph model since it only allows one label per edge.

$(V, E, L, l_V, l_E, K, W, p_V, p_E)$ where L is a set of labels which are mapped by the labeling functions $l_V : V \to P(L)$ and $l_E : E \to P(L)$ to vertices and edges. $P(L)$ contains all possible subsets of L an is refereed to as the power set of L. As already mentioned, each vertex and edge can be associated with a key-value pair defined in our case as $p = (key, value)$, where $key \in K$ and $value \in W$. K and W together represent all possible key and value sets. p_V represent all sets of key-value pairs associated with a vertex u while p_E contains all key-value pair sets assigned to edge e.

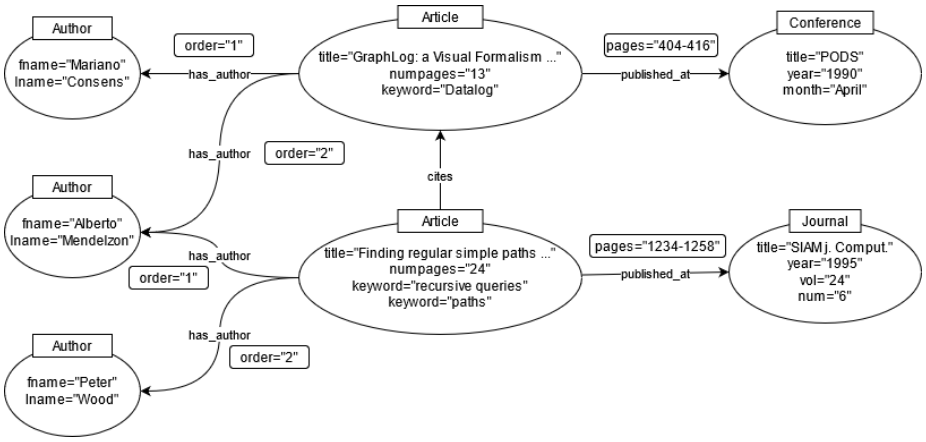

Figure 2: Example of a label property graph data structure for bibliographic information [1].

Figure 2 shows an example of a label property graph for bibliographic information. We can see, for example, that the vertex in the left upper corner has the label "Author" and two properties keys named "fname" and "lname". The property values are "Mariano" and "Consens", respectively. Moreover, the vertex is connected via an edge labelled "has_author" to another the vertex labelled "Article". The edge has a property called "order", indicating who is the primary author and is directed from "Article" to "Author". The article has properties indicating the title number of pages and associated keywords. As can be seen in the example, articles can have multiple authors and are published in different types of media sources.
From a data modelling perspective, the example in figure 2 illustrates that vertices can be seen as entities and edges as relationships connecting entities. The properties can be seen as additional descriptions specifying the features of the entities and relationships. Even though the example in figure 2 is not very complex, the flexibility of the data model enables users to efficiently design arbitrary complex graphs to capture the knowledge of a whole domain. Thus, as claimed by Zhang et al. [50], graph databases which rely on label property graphs are a promising technology to substitute traditional SQL databases to model and store densely connected information such as domain knowledge.

1.3 Research Questions

Detecting slow DoS attacks in IoT or IIoT environments is a non-trivial task since they usually consists of hundreds or thousands of differed sensors and devices with unique characteristics. Therefore, relying solely on anomaly detection approaches may not be sufficient since they are missing the crucial context information needed to interpret network traffic correctly. Consequently, this thesis will investigate the following research questions:

- How can a graph-based IDS be designed to detect slow DoS attacks in the context of MQTT networks?
- What are the advantages of a label property graph data model, especially when encapsulating network context information?
- What are the major challenges of the proposed approach and how can they be dealt with?

2 Related work

The need to further automate CPSs leads to increasingly complex computer systems and network architectures. By adding extra layers of complexity, it becomes more and more difficult to identify and understand cyber-attacks. To be able to still capture the dynamics and dependencies of security threats, attack modelling techniques[18] have been developed by the scientific community. The techniques are used to trace the sequences of events back which are needed to carry out a successfully attack. There are a number of different modelling techniques present in the research field as identified by Lalie et al. [26]. Based on their work, the approaches can be divided into three main categories:

- Use case methods define interactions between users and the system under investigation. Use case diagrams are a popular tool to identify functional requirements but they usually neglect security aspects. Some researchers adapt the idea of use case diagrams and construct so called misuse cases to define which system features should be blocked to ensure system security [40].
- Temporal methods focus on describing cyber-attacks based on used techniques to disrupt a victim system over time. One example of such a temporal method is the diamond model by Caltagirone et al. [11] which attaches activity threads to associated intrusion events. The introduced activity threads are used to capture the execution order of the actions taken by an adversary.

[18] Even though attack modelling techniques do not focus on detecting intrusions the techniques encapsulate knowledge about present vulnerabilities in network systems.

- Graph-based methods design threats patterns in form of a graph where vertices represent vulnerabilities, preconditions, postconditions or a combination of all three entities. Edges on the other hand are used to indicate dependencies between vertices. ADEPTS is a method proposed by Foo et al. [21], which relies on a intrusion goal graph to detect malicious activity. Intrusion goals are modeled as nodes whereas logical dependency between them are model as edges.

Graph-based models will be discussed in detail in the following paragraphs while the other two approaches introduced above have been identified as out of scope of this thesis.
Graph-based vulnerability analysis uses different approaches to construct attack graphs. An attack graph can be defined as S being a set of all possible states, τ as the transition relation set between states, S_0 as a set of initial states and S_S as a set of success states. Therefore, an attack graph can be described with the tuple $G = (S, \tau, S_0, S_S)$ [39].The vertices in an attack graph are used to model attack states while edges are used to represent transition relations between vertices. The concrete definition of entities and relations represented as well as the used vocabulary applied in an attack graph depends heavily on the authors. The same applies for the information required to construct an attack graph [26].
Initially, attack graph needed to be created manually by so-called "Red Teams". Their purpose is to penetrate the system from the perspective of an adversary and identify potential security risk. Based on their gained knowledge, they could construct an attack graph to visually represent the found vulnerabilities. However, even though these teams can be an effective way to discover vulnerabilities in network systems it is usually time-consuming and cost intensive process. As networks of hosts become increasingly complex, generating attack graphs manually becomes impractical or even infeasible.
Over two decades ago, Philips and Swiler introduced one of the first approaches to semi-automatically construct an attack graph [34]. Their approach combined physical network topology information with a set of possible attack scenarios. They used nodes to model attack states which encapsulate information about user access levels, attack effects, hardware configurations etc. The edges have been used to indicate a transition into a new attack state. To generate the attack graph configuration files, attacker profiles and attack templates need to be provided. Relevant information about operating systems, router configurations and the network topology are provided by the configuration file. Attacker profiles encapsulate knowledge about the attackers capabilities, whereas attack templates contain the procedure needed to perform a known attack.
Sheyder et al. [39] proposed to use a finite state machine to capture network topology information. In their case, a state of the network encapsulates information about vulnerabilities, services, active connections or trusted access points. Transitions between states are triggered by an atomic attack of an intruder. The network system is associated with security properties to define crucial aspects of the security system which should not be compromised. Adversaries generally

try to penetrate the network system by exploiting one or multiple security properties. Based on a set of states, a set of initial states, a set of transition relations and a set of security properties, they designed an algorithm to automatically generate an attack graph.
A more recent approach, introduced by Barik and Mazumdar [6], uses graph database technology to model domain knowledge and generate attack graphs. To model malicious attack patterns, they use an exploit dependency graph which is modelling conditions and exploits as vertices while directed edges indicate dependencies. Consequently, an edge $e = (a, s)$ that originates from a and leads to s, indicates that exploit s depends on state a. More specifically, the proposed attack model uses two types of vertices: one to describes attacker capabilities and network conditions and a second one to represent exploits. Edges directed from an exploit to a network condition express a resulting effect or postcondition of the exploit, whereas a reversed directed edge indicates preconditions for executing an exploit. To generate an instance of the introduced attack graph they store network configuration information in a graph database. The data model includes hosts, services, vulnerabilities, attacker and attacker goals as vertices. Each node can have any desired number of properties to further define them. Additionally, edges can indicated various types of dependencies, such as for example defining which services run on which hosts or which preconditions are needed to execute a specific attack etc. Finally the attack graph is constructed by applying simple graph queries to interatively evaluate which exploits are reachable at a certain location in the graph.
Zhang et al. [49] proposed an attack graph generation algorithm for industrial control systems based on the label property graph model. To generate an attack graph, information about the network topology, present devices and associated vulnerabilities are firstly imported into a Neo4j[19] database. In the data model, vertices can either be devices, vulnerabilities or network modules used to structure and protect the system like firewalls or domain configurations. Edges associate devices with vulnerabilities, connect network modules with each other, indicate the network location of devices and highlight which devices are directly connected to each other. Additionally, an attacker device has been introduced to mark the adversaries network penetration point. Based on the collected data, an iterative method is introduce to query information stored in the database and propagate the results into an attack graph.
Even though these approaches collect crucial domain knowledge about network configurations, attack graphs are most commonly used to harden network infrastructures and not to detect intrusion. However, a few approaches like the ones proposed by Roschcke et al. [36] and Wang et al. [46] exist which use attack graph to improve IDSs. They use a method commonly known as alert correlations to map detected events from various sensors to a previously created attack graph. The approaches aim to decrease the false positive rate of IDSs by combining real-time events with attack graphs.
Apart from various efforts to generate and model attack graphs, in recent years

[19] `https://neo4j.com/`

graph models are used to provide context information about network traffic data. In general, IDSs rely on different taxonomies proposed by the research community to classify attacks but lack capabilities to express concepts, relations and axioms about a specific domain [43]. Diederichsen et al. [16], for example, tackled this issue by constructing a graph model to capture information provided by log files. They investigated log files produced by the network security monitoring tool Zeek[20]. The tool monitors network traffic and produces dedicated log files for encountered events. In this particular case, the authors focused on the connection, HTTP and DNS log files. For each log file, a graph representation has been designed which encapsulates the event specific message structure. The DNS graph model, for example, contains nodes representing hosts, IP addresses and DNS connection meta-data. Edges link the nodes to indicate which hosts and IP addresses a request resolves to. By encapsulating the meaning of log statements in a graph structure, the authors facilitate a detailed traffic analysis to detect malicious events based on network traffic.

More commonly, the semantic web technology stack is used to model context information about available network data. The tool stack consists of different standards published and maintained by the World Wide Web Consortium (W3C). The foundation of the technology stack is RDF which is using triples to describe information on the Internet [24]. A triple consists of a subject, predicate and a object. Subject and objects are nodes while predicates are directed edges. Based on this simple model, things in the world can be described since subject and object can refer to any kind of resource such as abstract concepts, documents, multi-media files, physical things, numbers etc. A resource can therefore be seen as an entity in the world of discourse we want to describe. To gain further expressiveness, additional standards were build upon RDF to build powerful ontologies.

Ontologies provide rich semantic constructs to encapsulate context information. Hence, different authors like Undercoffer et al. [43] and Kim et al. [23] introduced different security ontologies to improve network security. Kim et al. [23] derived from existing security ontologies a set of ontologies covering descriptions of security concepts, specification of credentials used for authentication mechanism, definitions of applied security algorithms, definitions of assurance standards, annotation for semantic web services, descriptions of important security information and specifications for input and output parameters of web services. Their goal was to enable intelligent match making of web services based on ontological annotation. To provide a successful matching, not only do the functional dependencies of web services need to be considered, but also security requirements from users and the service itself need to be integrated. They demonstrated that their matchmaking algorithm performs better than similar approaches because it makes extensive use of the defined security property attributes modeled in the ontology. Even though the work by Kim et al. demonstrates how context information can efficiently be integrated to improve web service security, they don't cover its application for IDSs and neither do

[20] `https://zeek.org/`

they consider protocol specific vulnerabilities.
Undercoffer et al. focus on modelling computer attacks and intrusions to use an inference engine to recognize malicious events. Their proposed ontology includes at the top level a host class which is connected to the system components class to track the current system state. In addition, the host can be a victim of an attack which may be directed to compromise different system components. Each attack has a set of different inputs and results in different consequences for the system such as, for example, an adversary gaining root access or achieving a denial of service state. The attack inputs are further distinguished by the methods used by an adversary to perform an attack successfully. An example of such a method are malformed messages which can not be properly processed by the target system and my lead to a DoS state. To validate the ontology, it is imported alongside with instance data into the knowledge base. By querying the knowledge base, specific attacks like Syn Flood can be detected or the scope of the query could be broadened to list all existing DoS attacks. Even though the authors describe the system as being effective in detecting DoS attacks, no emphasis is given to protocol dependent vulnerabilities or the detection of slow DoS attacks.
A number of research projects cover ontology-based security frameworks particularly designed for IoT and IIoT environments. An ontology-based framework consisting of three layers was proposed by Mozzaquatro et al. [32]. One layer is dedicated to support the design process of security services by semi-automatically generate code from high-level definitions of specifications. During run-time, another layer collects events and network information from different sensors to monitor the systems and to detect malicious activities or system vulnerabilities. Additionally, they introduce an integration layer based on the IoT Security (IoTSec) ontology to integrate different data sources into one database. An advantage of their approach is the possibility to utilize the captured knowledge to reason about threats and how to prevent them. The ontology is a one of the central components of their framework and was introduced in their earlier work [31]. It encapsulates assets, threats, security mechanism and vulnerability as top-level concepts. Assets are abstract concepts including all valuable entities essential for the success of the organization such as, for example, human resources, employee capabilities, software, hardware etc. Threats are described as attacks exploiting different vulnerabilities to compromise the system. Tools to detect and protect against malicious activities along with their capabilities are described as security mechanism. Known weakness of M2M technologies and corresponding assets are incorporated by the vulnerability class. Based on an industrial case study, the authors demonstrated that their proposed framework is capable of handling the complexity and diversity of IoT and IIoT environments. However, their approach does not focus on detecting intrusions. They designed the system to support the service design process by suggesting appropriate security mechanisms or recover strategies.
Xu et al. also propose a security model based on the so-called network security situation awareness model [48]. The model builds upon the concepts introduced by Bas et al. [7] and Endsley [19]. It consists of the following three levels:

- The *situation perception level* receives information from heterogeneous data sources to transform it into an appropriate format and preprocesses it to be analysed in the next level.
- The *situation evaluation level* identifies potential correlation between security events and detects malicious activities.
- The *situation prediction level* provides predictions about future security issues based on the current and historical network state.

The research presented by Xu et al. focuses on the first two levels which are dedicated to handle heterogeneous data sources. They propose an ontology which defines context, attack, vulnerability, and network flow as core concepts. To provide proper representation of the context, the ontology provides descriptions for hosts, hardware, software and devices used in IoT environments. Potential system threats can be classified with the ontology since it is based on the concepts defined in the Common Attack Pattern Enumerations and Classifications (CAPEC)[21] provided by the Department of Homeland Security. To identify how adversaries can exploit vulnerabilities, they additionally include concepts to characterize security breaches as well as hardware or software bugs. The ontology encapsulate definitions of network flow protocols to integrate standards like NetFlow v9, Argus, and IPFIX as well. To detect attacks, they defined logical rules representing the characteristics of an attack. By reasoning about the rules, the received instance data from the network sensors can be effectively analysed and intrusion be detected. Even though the authors demonstrate the effectiveness of their approach, they do not show if it can also be effective to detect slow DoS attacks targeting MQTT brokers.
To the best of my knowledge, there has not been an approach introduced in the literature dedicated to detect slow DoS attacks targeting MQTT brokers. To close this gap, this thesis introduces an approach to integrate MQTT log file information into a graph model to effectively query the information and detect slow DoS attacks.

3 System Architecture

This section introduces system architecture requirements to tackle the challenges introduced in section 1.2.2. The gathered requirements are used as a foundation to propose a prototypical IDS architecture capable to detect SlowITe attacks.

3.1 Architecture Requirements

As introduced before, IoT and IIoT applications face a variety of different threats, one of which is slow DoS attacks like SlowITe. To manage the new challenges, the following requirements have been identified based on the discussions presented in sections 1 and 1.2.2:

[21] CAPEC is a dictionary of known attack patterns employed by adversaries.

- **Protocol Heterogeneity:** IoT and IIoT environments apply various types of technologies relying on a variety of different communication protocols. To properly analyse protocol specific information, IDSs need to be capable of properly capturing context information based on used protocol standards. Hence, the system architecture needs to include a flexible data model to integrate protocol dependent information.

- **Topology Model:** To monitor a network effectively, information about the network topology is crucial. A topology model provides detailed information about the network structure of an organization. It includes the amount of hosts in a domain, the location of access points, firewalls and routers, etc. Especially for attacks potentially originating from resource constrained devices, a topology model provides critical information to identify an illegitimate device in the network. Therefore, integrating a topology model into the proposed IDS is essential to easily track suspicious devices and block them if necessary.

- **Attack Graph:** To improve system resilience, attack graphs are a popular tool to identify vulnerabilities. The knowledge encapsulated in attack graphs is a valuable source of information to harden the system and mitigate intrusions. Consequently, integrating attack graph information into IDSs will make them more reliable and improve intrusion discovery by matching attack patterns.

- **Log Data Integration:** To detect protocol dependent exploits and intrusions, it is necessary to not only analyze network traffic data; other sources like log files are valuable information sources as well. Log files consist of natural language statement which are difficult to evaluate automatically. Consequently, to avoid manual work, the proposed IDS should integrate log file information directly into the data model.

- **Connection Monitoring:** SlowITe attacks can hardly be detected by monitoring the bandwidth capacity taken by an adversary. There is a need to monitor the amount of active connections a device established to a specific MQTT broker. Therefore, the suggested IDS should include the capability to track the amount of connections established from a specific host to efficiently detect an attack.

3.2 Architecture Proposal

A typical IDS system consists, as discussed in subsection 1.2.4, of four components: a sensor or agents, a database server, a management server and a console. In figure 3, we can see theses components embedded in the proposed system architecture. On the left side of figure 3, a MQTT broker is shown which receives sensor data from two different rooms. The broker produces a log file which is accessed by an agent to continuously read the log events and forwards the content to a transformation module. The transformation module

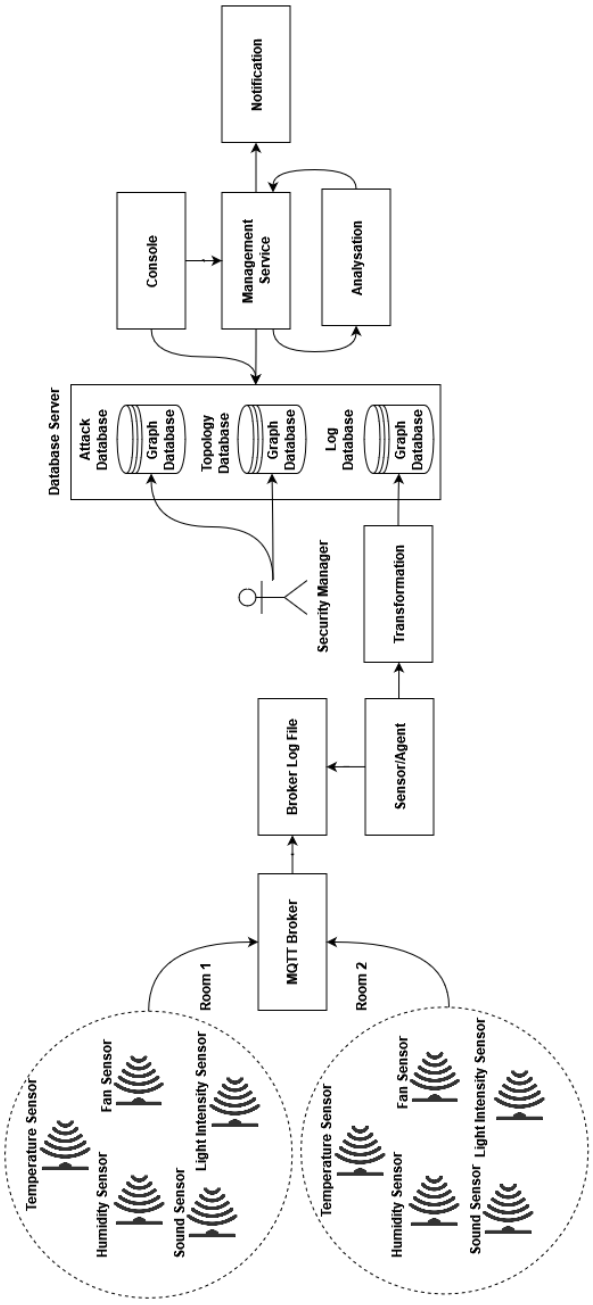

Figure 3: System Architecture of the proposed Intrusion Detection System

is one of the core modules which converts the plain text data from the log file into a graph structure and imports it to the database server. To provide sufficient context information and properly analyse the imported log files, a domain expert needs to model the network topology ahead and connect network nodes with associated vulnerabilities. The created models have to be imported into the graph database before the network can be monitored. To utilise the stored data, a management service is needed to monitor the network data. Based on the incoming data streams, the management service recursively analyses the newly received data and sends a notification to the system administrator if an intrusion was detected. Above the management service, the console module is located. It serves as a user interface to interact with the management service and the database server.
The introduced architecture was designed to acquire, analyse and monitor network relevant log file data. Besides adapting well known IDS modules like agents, database servers, management servers and consoles the architecture has an additional transformation module. The module is linking real-time log events to network topology informations and associated system vulnerabilities. This connections are more sophisticated network analysis since log events can be monitored within a specific network context.
The module oriented architecture facilitates the possibility to enhance the system without rewriting the core of the code base. Currently, the architecture focuses on transforming MQTT log files into a graph data model. However, to support additional IoT and IIoT protocols additional transformation modules can be implemented. Promoting a simple way to extend the system is crucial since IoT and IIoT environments utilize a variety of different protocol standards.
The integration of log events into a graph data model, which encapsulates network context information, is expected to provide a sophisticated foundation to accurately detect slow DoS attacks. Usually DoS attacks as well as slow DoS attacks are being detected by analyzing the network bandwidth utilization. Most commonly a baseline is calculated which represents the averagely occupied network bandwidth. An attack will be detected if the actual network utilization differs significantly from the defined baseline. Unfortunately, slow DoS attacks are designed to occupy relatively little network bandwidth and are therefore hard to detect with such methods. To overcome this challenge the architecture focuses on analysing log events of network services. These log files usually keep track of each connection attempt undertaken by a client. By integrating this information into a graph data model all open connections of each client can be monitored. A malicious device will attempt to open as many connections as possible to block access to the network service. An intrusion should therefore be detectable based on the number of established connections per client.
To accomplish a context-aware log event integration an appropriate graph data model has to be designed to efficiently access the knowledge stored in the database server. In the current architecture proposal the choice has been made to use a graph based data model since following advantages are expected to

emerge from this technology choice[22]. In comparison to relational data models, which store data in tables, graph databases are designed to store densely connected data[29]. To discover complex relations between tables in relational database it requires expensive "JOIN" operations which heavily decrease query performance. Since network topologies can intuitively be a model as a network of connected nodes, a graph data model is a suitable representation to store such data. In addition, modelling domain knowledge as a densely connected graph is a common presentation pattern and was firstly introduced by Stokman et al.[41]. Hence, choosing a graph data model to combine network topology informations with system vulnerabilities and log events is a reasonable choice.
In the following subsection the proposed graph data model will be introduced in detail. The essential concepts and relations to model the information are discussed as well as their purpose.

3.3 Graph Data Model

In the middle of figure 3, we can observe the database server consisting of three different databases. However, all three instances can be harmonized into one by designing an appropriate data model. Figure 4 represents the used data model to combine knowledge about the network topology, attack patterns and log files together. The data model builds upon the ideas and concepts from Diederichesen et al. [16] and Barkik and Mazumdar [6].
Circles in figure 4 depict nodes whereas the rectangles illustrate the properties associated with a node. The directed arrows represent edges between nodes. The edges can be distinguished by their specific name labels. The figure contains the following node and edge types:

- ***Host*** nodes located on the left side of figure 4 are used to model host machines present in the network. They can be identified by their corresponding IP address. Additionally, hosts have a *creation_time* property to specify at which time the host entered the network. The property *notification_sent* illustrates whether a notification message about a suspicious host has been forwarded to the system administrator whereas the *is_blocked* indicates a blocked host not allowed to establish new connections.

- ***HOSTED_AT*** edges illustrate which services are running on which host. Obviously, a single host machine can host multiple services.

- ***STARTS_CONNECTION*** edges are used to demonstrate that a host is starting a connection attempt to a specific service. These edges link host nodes with a connection nodes.

- ***Connection*** nodes were introduced to monitor the active connections

[22]Which data model to adopted is use case dependent. Graph data models are not by default better suited.

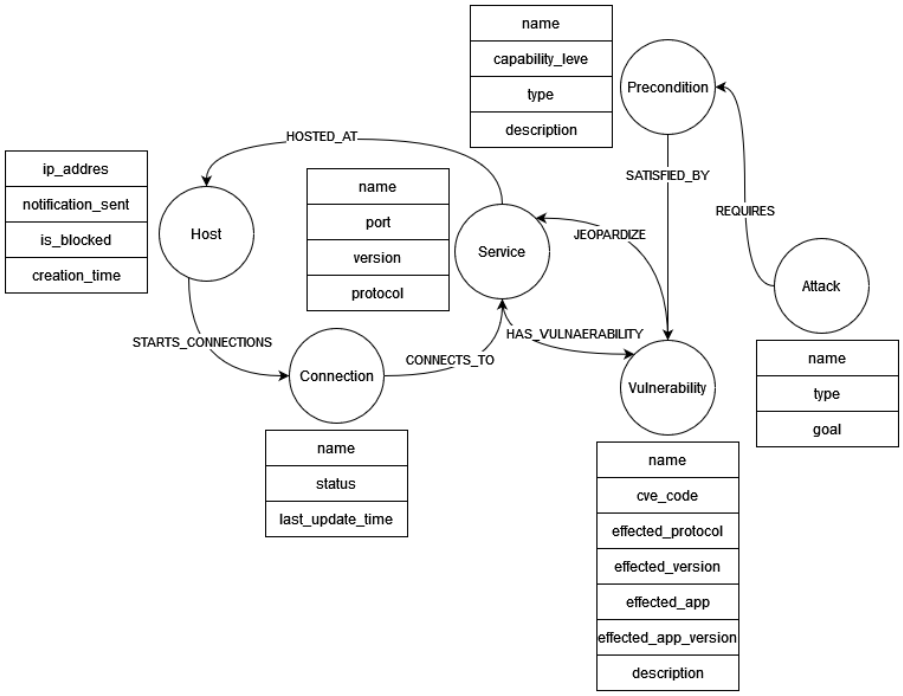

Figure 4: Proposed data model for the IDS introduced in section 3

established to a service[23]. A connection can be identified by their *name* property, which is a unique combination of the service name and a random hash value. Additionally, a connection has a *status* and *last_update_time* property to indicate whether a connection is active or not and to store the last time the connection status was updated.

- ***CONNECTS_TO*** edges indicate to which service the connection was established. They therefore link connection nodes with service instances.
- ***Service*** nodes are used to depict different kinds of software tools running on a host. They can be identified by their name, the exposed port and the corresponding version of the service instance. Furthermore, since many slow DoS attack exploit protocol vulnerabilities, the property *protocol* specifies which communication protocol the service is using.
- ***HAS_VULNERABILITY*** edges are used to associated services with vulnerabilities. Services can be connected to multiple vulnerabilities.

[23]Note that an edge could also have properties and there would not be the need to introduce a specific connection node. However, the node was included to facilitate graph traversal with Neo4j.

- A ***Vulnerability*** node is used to model different system vulnerabilities and can be identified by the *cve_code* property which mirrors the Common Vulnerabilities and Exposures (CVE) code assigned by the MITRE Corporation[24]. In addition, the node contains properties describing the user-defined name of the vulnerability, the effected protocol and application names, the corresponding versions of the protocol and the application, as well as a short description of the vulnerability.

- ***JEOPARDIZE*** edges are the inverse of ***HAS_VULNERABILITY*** edges and were introduced to improve the query capabilities of the data model.

- ***Attack*** nodes represent different kinds of attack categories. Attacks can also be identified by their name which consists of a combination of the user-defined attack name and a random hash value. Furthermore, attack nodes contain a *type* and *goal* property to specific the category of the attack and the primary goal.

- ***REQUIRES*** edges are used to link attacks with associated preconditions to indicate which system features have to be present to conduct a specific attack.

- ***Precondition*** nodes represent different system conditions an adversary has to achieve to exploit a vulnerability. They can be identified by their user-defined name since it adheres to the same format as attack and connection names. The additional properties to describe preconditions are partly taken from Lallie et al. [26] who claims there are three different precondition types: 1) Statuses/services 2) Reachability 3) Perpetrator capability. The precondition property *type* is used to distinguish between the first two types which indicate whether a precondition is dependent on a specific service, software or hardware version whereas reachability illustrates dependencies to access a target device. The *capability_level* property represents the skill level needed to utilise a certain precondition for an attack. To provide further insight, the property *description* has been added to provide a short statement about system consequences once the precondition can be satisfied by a specific vulnerability.

- ***SATIEFIED_BY*** edges link preconditions with vulnerabilities and ultimately to the topology since vulnerabilities are connected to services. Through this relation, the data structure is not only capable to monitor the actual connection states of the different hosts, it can also be used to improve system resilience.

By adapting the introduced data model following advantages are expected: Graph data models alongside with their implementations in graph databases are perfectly suited to extract possible pathways through the network topology. For example, a common analysis is to detect the shortest path between

[24] `https://www.mitre.org/`

two nodes. Such analytical opportunities provide crucial insights for security managers to discover potential threats and organize appropriate mitigation measures. Furthermore, by combining network topology information with associated vulnerabilities, the data model provides a sophisticated overview to increase system resilience. In IDSs information is commonly stored in different locations and is not directly connected. A security manager has to match the relevant information manually. By querying the graph database, this time consuming task can be reduced drastically and the security manager can focus on solving the discovered threats.
The proposed data model further reduces manual labor since network services produce natural language log statements, which can only be interpreted correctly by a human. By extracting relevant information from these statements and importing them into a database, they become machine-readable and can be semi-automatically analyzed. Additionally, the graph data model provides an intuitive representation of a network of hosts which is easier to understand and analyze for security managers.
The introduced concepts enable the database to keep track of each connection established by a client. This feature is expected to be sufficient to detect SlowITe attacks. Since the introduced data model is able to store all connection attempts from each client a distributed SlowITe should also be detectable.
As been discussed in the previous paragraphs the proposed architecture in combination with the introduced data model satisfies the elected requirements in subsection 3.1 by providing the possibility to extend the transformation module for different log file formats, by combining the network topology model with attack patterns, by streaming log file data from the agent directly to the database and by keeping track of the established connections documented in the log file. In the following sections, implementation details of the validation test bed as well as details of the prototypical detection system implementation are presented.

4 Implementation of a Test Bed

Simulating a network of sensors is a common task to test applications dedicated to operate in IoT and IIoT environments. Hence, there are multiple licensed software solutions available to support the validation and testing of IoT and IIoT applications. The most well known solutions are provided from Microsoft[25], Amazon[26] and IBM[27]. However, these software solutions are only available as subscription services. On the other hand free open-source solutions like NetSim [28] are well established and sophisticated solutions to simulate networks. Unfortunately they require a lot of time to master. Another well known

[25] https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-machine-learning-edge-03-generate-data?view=iotedge-2020-11

[26] https://aws.amazon.com/de/solutions/implementations/iot-device-simulator/

[27] https://github.com/IBM/IBMDeveloper-recipes/blob/main/use-the-simulated-device-to-experience-the-iot-foundation/index.md

[28] https://www.tetcos.com/index.html

open-source tool to simulate and manipulate sensor data is Node-Red[29]. This software solution is very flexible and supports a variety of different communication protocols. Nevertheless, to the best of the authors knowledge, Node-Red can not be used to simulate different IP addresses and therefore is not suitable to imitate different IoT or IIoT devices.
Consequently, the decision was made to develop a new IoT and IIoT test bed to simulate sensor data, to provide a network infrastructure and to run essential software services.
In figure 3, two rooms are shown that contain each a device which produces different sensor data and sends it to a MQTT broker. The test bed consists of services providing these functionalities. Additionally, to test an intrusion, two attacker devices are included in the test bed to perform a SlowITe attack.
To provide these functionalities following challenges had to be tackled:

- To simulate a network including different sensor devices, each device has to be equipped with a dedicated IP address.
- Each device needs to be capable to simulate different types of sensors.
- The sensor data exchange has to be done over the MQTT protocol.
- To detect an attack the developed IDS has to have access to the MQTT broker's log file.
- To execute an attack the malicious devices have to be able to connect to the MQTT broker as well.
- To facilitate the usage the test bed should be easily deployable, monitorable and portable to other operation systems.

The test bed was implemented by adopting the following four technologies: Mosquitto[30], an open-source message broker based on the MQTT protocol, Node-RED, an open-source low-code platform for developing event-driven applications, Docker[31], an open-source tool stack to containerize applications and Kubernetes[32], an open-source framework to deploy, manage and scale containerized applications.
Node-RED is used to simulate sensor data and forward it to a MQTT broker. Node-RED was developed by IBM and is a tool for visual programming. The code segments are visualized as flows consisting of interconnected nodes. A node processes data received from previous nodes or external sources and forwards it to other nodes or external services. Node-RED offers a large variety of already implemented node pallets to accomplish diverse tasks. A special type of node is a configuration node which includes a predefined programming logic to achieve a certain task. Configurations nodes come along with defined customization

[29] `https://nodered.org/`
[30] `https://mosquitto.org/`
[31] `https://www.docker.com/`
[32] `https://kubernetes.io/`

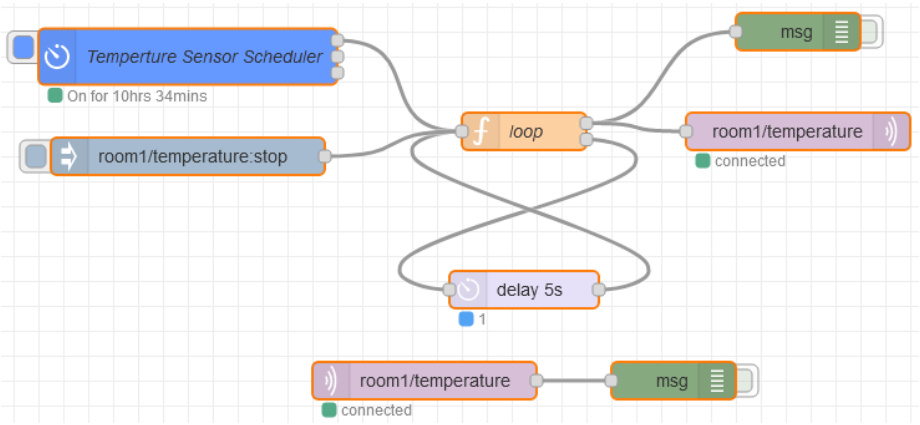

Figure 5: Example flow in Node-RED to simulate temperature data

settings to reduce programming effort. The MQTT Input or Output node for example lets you configure the server IP address, user credentials, topics, etc. Programmers do not need to know the implementation details of configuration nodes, they only need to provide the correct configuration settings.
In figure 5, we can see an example flow created to simulate temperature data in the test bed environment. The flow starts on the left with a scheduler node which controls during which day times the sensor should be active. Underneath it is an injection node which can be used to manually stop the flow to support debugging. In a second step, a function node named "loop" is used to generate temperature data. This node belongs to the standard node palette of Node-RED and can execute arbitrary JavaScript code. In this implementation, the loop uses a random number function to simulate temperature data. The loop node has multiple outgoing edges. The most upper one is connecting the loop to the green debug node which can be used to see the raw HTTP messages produced by the function node. The second edge is connected to the purple MQTT node which is forwarding the data to the MQTT broker which will associate the data stream with the topic name "room1/temperature". At last, the third edge is connected to a delay node which stops the execution of the next iteration of the loop for five seconds. In summary, the flow sends, during predefined day times, a temperature measurement each five seconds to a MQTT broker. Beneath the main flow, there is a small flow located to read the same topic from the broker and display the message content. This flow purely serves as debugging tool.
For ease of use and reusability, all applications used in the test bed have been containerized with Docker. Docker is an operation system virtualization tool. Docker uses so-called containers to be independent of operation system specific dependencies. Each container has its isolated set of software, libraries and configuration files which enables a container to run on any operations system supporting the Docker Engine. Instructions on how to build a specific container

are stored in so-called Docker images. They are created by compiling the source code contained in a Dockerfile.
Additionally, Docker provides a repository server called Docker Hub[33] to share ready-to-use container images. Most major open-source platforms have their own repository on Docker Hub just like Node-RED[34]. Therefore, to run a Node-Red instance on an operating system, the Docker Engine must be installed and the latest branch of the repository from Docker Hub has to be pulled. Afterwards, the containerized application can be started with a simple Command Line Interface (CLI) command.
For the attacker devices, a Docker image based on Ubuntu Desktop 20.04[35] and the MQTTSA[36] tool published by the Security & Trust Research Unit in Fondazione Bruno Kessler has been created and published[37].

Code Listing 1: Dockerfile to create a Docker Image of the MQTTSA application.

```
1  FROM ubuntu:20.04
2
3  LABEL maintainer="thorsten.steuer@gmx.net"
4  LABEL version="0.1"
5  LABEL description="Docker Image for https://github.com/stfbk/mqttsa"
6
7  ENV DEBIAN_FRONTEND=noninteractive
8
9  RUN apt-get update
10 RUN apt-get install -y git python3-pip tshark curl mosquitto-clients && \
11     rm -rf /var/lib/apt/lists/* && \
12     apt clean
13
14 RUN git clone https://github.com/stfbk/mqttsa.git
15
16 WORKDIR /mqttsa
17 RUN make
```

In the code listing 1 we can see the implemented Dockerfile to create images of the MQTTSA application. The image is used to create containerized applications which are capable of executing SlowITe attacks. In the first line the used operation system for the Docker Image is defined. In our case it is based on the Ubuntu version 20.04. After defining the operation system the *LABEL* operators in line three to five are used to add some metadata to the Docker

[33] https://hub.docker.com/
[34] https://hub.docker.com/r/nodered/node-red
[35] https://ubuntu.com/download/desktop
[36] https://github.com/stfbk/mqttsa
[37] https://hub.docker.com/r/bigt1991/mqttsa

Image. Afterwards an environmental variable is set to trigger a non-interactive installation procedure of the operation system. In line nine to fourteen several *RUN* commands are used to update the operation system, to install essential software packages and to clone the repository of the MQTTSA application. The *WORKDIR* operator changes the working directory to install the MQTTSA application with the *make* command.
Even though the Eclipse Foundation[38] provides a ready to use Mosquitto Docker image, a custom image was created by the author to use the powerful Ubuntu CLI to have more control over the Mosquitto service[39].
Since deploying, configuring and monitoring multiple containerized applications can be cumbersome, Kubernetes has been used to facilitate these tasks. Kubernetes is a open-source platform specifically designed to support the following features: service discovery and load balancing, storage orchestration, automated rollouts and rollbacks, automatic bin packing, self-healing and secret and configuration management[40]. To deploy a docker container in a Kubernetes cluster, YAML files need to be created which define the deployment details of the containerized applications. Once these deployment files are created, you can deploy and start multiple applications with a simple CLI command.

Code Listing 2: YAML file to deploy the MQTTSA apllication in Kubernetes.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mqttsa1
5    labels:
6      app: mqttsa1
7    namespace: slowdos-attacker
8  spec:
9    selector:
10     matchLabels:
11       app: mqttsa1
12   template:
13     metadata:
14       labels:
15         app: mqttsa1
16     spec:
17       containers:
18       - name: mqttsa1
19         image: bigt1991/mqttsa:latest
20         command: ["/bin/sleep", "3650d"]
```

[38] https://www.eclipse.org/
[39] https://hub.docker.com/r/bigt1991/mosquitto-broker
[40] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

```
21          imagePullPolicy: IfNotPresent
22        restartPolicy: Always
```

In the code listing 2 we can see the Kubernetes YAML file to deploy the MQTTSA application in a Kubernetes cluster. The first tag specifies the Kubernetes Application Programming Interface (API) version which should be used followed by the definition of the resource type we want to create. In our case we want to create a *Deployment* resource which is used to describe a desired application state. Afterwards in line three metadata is added to describe the application and assign it to a desired namespace. In line nine the tag *selector* controls what kind of applications are managed by the deployment. This deployment therefore manages all applications called *mqttsa1*. The *template* tag in line twelve contains the application's configuration information. We can see the application is labeled with the name, *mqttsa1* and the container will be created based on the docker image *bigt1991/mqttsa:latest*. Additionally, in line twenty a CLI command is used to start the containerized application. Usually a Docker Image will start the services encapsulated in it automatically. However, in this particular case it was more user-friendly to be able to execute the SlowITe manually. Hence, the application is set into a stand by mode until a user wants to interact with it. In the last line a restart policy for the application has been defined. In this YAML file it is set to the default value *Always* which ensures the application restarts once it stops. The policy guarantees a consistent availability of the application.
To implement the test bed kind[41], an open-source tool as been used. Kind enables its users to run local Kubernetes clusters in a docker container based on a configuration file specifying the cluster setup. The test bed has been implemented as a single node cluster to avoid unnecessary complexity of a multi-node cluster. A nice additional feature of Kuberenets is the administration dashboard it is providing. The dashboard serves as an interface for users not familiar with the appropriate CLI commands. The dashboard supports displaying, deleting, updating and configuring applications. As already mentioned earlier, a dedicated console for the IDS was not developed. However, the dashboard enables users to interact with all services and is therefore used as a substitute.
The test bed was designed to be extendable and portable. To extend the current implementation, several options are available:

- The variety of sensor data generated by a Node-RED instance can be extended by copying an existing flow, renaming the MQTT topics and adjusting the function node to simulate data based on the use case.
- Additional Node-RED instances can be created by copying the whole folder containing the deployment YAML files from room one or room two. In the current test bed implementation, each Node-RED instance represents a room as can be seen in the naming conversions of the YAML

[41] `https://kind.sigs.k8s.io/`

files. Consequently, to add a third room, all names in the YAML files containing a room number need to be substituted with the room number three.

- A third possibility to extend the test bed is by defining custom deployment files and running custom services depending on the use case.

The portability of the test bed is ensured since Docker and Kind are supported by Linux, Mac OS and Windows[42]. If both applications are deployed, the test bed can be pulled from GitHub[43]. Afterwards, the test bed can be started with a few CLI commands.The test bed can also be executed if the host machine only supports Kubernetes. However, to have the best experience, it is recommended to use the test bed in combination with kind to ensure proper port forwarding and volume mounting.
After introducing the test bed, implementation details for the actual IDS will be presented in the following section. More specifically the section will highlight the different modules implemented to provide the functionalities introduced in section 3. Furthermore, implementation challenges are introduced alongside with applied solution approaches.

5 Implementation of the Detection System

To realize the introduced IDS in section 3 the following implementation challenges had to be solved:

- The agent module needed to be capable to continuously read the log events produced by the MQTT broker.
- Log events are described in natural language statements which can not be processed easily by an algorithm. Hence, a method to automatically extract the needed information had to be developed.
- To import the log events into the graph database the extracted information components had to be mapped to the defined concepts in the data model.
- To put the real-time log events into context with the topology model and associated system vulnerabilities, a system setup module needed to be implemented. This module imports the essential network information before the log event monitoring can start.
- After combining all information in the database a scheduled monitoring loop needed to be implemented to detect intrusions and send a notification if one is detected.

[42]The test bed was only used in combination with Windows. The author can not ensure a smooth experience with other operation systems even though it is theoretically supported.
[43]`https://github.com/ThorstenBigT/ids_slow_dos`

- To interact with the system a solution for the console had to be found to enable users to monitor the system without solely depending on the CLI.

To implement the different IDS modules the author chose Python[44] and Neo4j[45] as technologies. Both technologies provide rich library support and have a good documentation and community support. The following paragraphs introduce the essential modules to overcome the introduced challenges and present code segments to illustrate the chosen solution approaches.
As the agent, a Python module was implemented to continuously read the log file produced by Mosquitto. The module mimics the functionality of the well known Linux tail CLI command. The Linux tail command is mostly used to print log statements of a service to a CLI. When the command is executed, it jumps to the end of the specified log file and only prints newly written log statements of the service to the CLI. The same program logic has been implemented with the python module named *log_file_access.py*[46].

Code Listing 3: Function definition to mimic the Linux CLI command.

```
1  def tail_file(self,):
2      seek_end = True
3      while True:
4          with self.local_file as file:
5              if seek_end:
6                  file.seek(0, 2)
7              while True:
8                  line = file.readline()
9                  if not line:
10                     try:
11                         if file.tell() > os.path.getsize(self.local_file_path):
12                             file.close()
13                             seek_end = False
14                             break
15                     except FileNotFoundError:
16                         pass
17                     time.sleep(1)
18                 yield line
```

In code listing 3 the function definition is shown which mimics the Linux tail command in Python. From line three to six a while loop is used to constantly seek the end of the submitted log file. Afterwards, in line seven to eight a second while loop is used to constantly read new incoming lines. If there have not been

[44] https://www.python.org/
[45] https://neo4j.com/
[46] https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/local_file_access.py

new log statements written into the file the try statement verifies if the file still exists and throws an exception if that is not the case. After the try-except block is verified the loop waits one second until it reads the next line. At the end of the loop the read log lines are returned as a generator[47].
Log files commonly consist of natural language statements which are separated by new line characters. To extract relevant information from these statements, different regular expressions have been applied. Regular expressions consist of sequences of characters representing a text pattern. There are many libraries which use regular expressions to find text patterns in strings[48]. In this IDS implementation, the python library *re*[49] was used. It provides various functions to match, find and spilt strings based on regular expressions. All regular expressions have been implemented in the *format_log*[50] module.

Code Listing 4: Function definition to extract the IP address from a log statement.

```
1 def extract_ip_address_by_row(self, log_string: str):
2     matches = re.findall(r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b", log_string)
3     if matches:
4         self.host_data["ip_address"] = matches[0]
5     else:
6         logging.info("IP address not found in: %s", log_string)
```

The code listing 4 displays the function definition to extract the IP address from a string. The function takes a string as input parameter. In line two we can observe the function call *findall* from the *re* Python library[51]. The function matches all occurrences in a string which corresponded to the define regular expression highlighted in red[52]. Afterwards, the found matches are saved in a JavaScript Object Notation (JSON) object to be accessed later. If no matches are found a log message will be created for debugging.
The JSON object containing the log file information is imported to Neo4j with different Create, Read, Update and Delete (CRUD) operations. Two modules have been created to realize these features. Firstly, a module named *neo4j_database_access*[53] has been programmed which encapsulates all CRUD operations needed to import a JSON object into a Neo4j database according to the introduced data model in subsection 3.3.

[47] Generators in Python are iterators which can only be iterated once
[48] A string in python, for example, is an array of bytes representing Unicode characters
[49] https://docs.python.org/3/library/re.html
[50] `https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/format_log.py`
[51] `https://docs.python.org/3/library/re.html`
[52] For further information about the meaning of the regular expression consult following link: `https://regexr.com/`
[53] `https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/neo4j_database_access.py`

Code Listing 5: Function definition to extract the IP address from a log statement.

```
def _create_and_return_host(transax, host_json: str):
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ host_json +'\') as host_data '
        'CREATE (h1:Host { ip_address: host_data.ip_address, '
                            'creation_time: host_data.creation_time, '
                            'notification_sent: host_data.notification_sent, '
                            'is_blocked: host_data.is_blocked })'
        'RETURN h1'
    )
    result = transax.run(query)
    try:
        return [{'h1': record['h1']['ip_address']}
                for record in result]

    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise
```

The code listing 5 shows the function definition to create a *Host* instance in the database. The function has two parameters: the *transax* parameter which is a connection object to communicate with the Neo4j database and the *host_json* parameter which encapsulates all information needed to create a host instance. From line two to nine the Neo4j Chyper query is defined. The first line of the query calls a pre-defined procedure, provided by Neo4j to process JSON input. Afterwards in the *CREATE* statement we can access the JSON object with it's assigned name. To retrieve the individual information components the dot operator can be used directly in the query to navigate through the object hierarchy. In line ten the query is executed with the previously provided *transax* object. In the end of the function definition a try-except block is used to check if the database connection is available. If the evaluation is successful the function returns the freshly created host instance to verify it's creation, otherwise it will raise a database service error.
The second module, called *mosquitto_log_transformation*[54], is the core module since it contains the appropriate domain logic to correctly extract the information from the log file and importing it to Neo4j.

Code Listing 6: Code segment of the transformation module to create a *Host* instance.

[54] https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/mosquitto_log_transformation.py

```
1  loglines =  local_file.tail_file()
2  for current_line in loglines:
3      log_formatter.extract_ip_address_by_row(current_line)
4      log_formatter.extract_port_number_by_row(current_line)
5      log_formatter.extract_time_in_seconds_by_row(current_line)
6      log_formatter.extract_connection_name_by_row(current_line)
7      log_formatter.extract_version_by_row(current_line)
8
9      if None not in log_formatter.host_data.values():
10         if not neo4j_driver.check_if_node_exists('Host', 'ip_address',
11                                  log_formatter.host_data['ip_address']):
12             neo4j_driver.create_host(json.dumps(log_formatter.host_data))
```

In code listing 6 a code segment of the transformation module is shown. In the first line a *local_file* object is created which is an initialization of the *local_file_access* module to use the previously described tail function. Afterwards a for loop is used to iterate over the returned log statements. From line three to seven different function calls are used to extract information from the log file. The function call in line three ,for example, uses the previously introduced regular expression to retrieve the IP address. Afterwards, in line nine, a if-statement is used to verify if all information components are available to create a host instance in the database. This verification is needed since not all essential information components are encapsulated in a single log statement. In line ten an additional if-statement has been implemented to check if the host already exists in the database to avoid duplicates. Subsequently, a host can be generated in the database with the extracted information from the log statements. To create other instances of the data model a similar programming logic was applied.
Once the data is stored in the graph database, a Python module named *network_monitoring*[55] monitors all active connections by periodically querying the database. Currently, intrusions will be detected via a threshold-based approach. The current threshold is at a maximum of 50 connections per host to each service. Hence, if a host has more open connections, additional connection attempts will be identified as an intrusion and an e-mail notification will be sent to a system administrator. To send the e-mail, a module named *alarm_notification*[56] has been implemented to handle the communication with the Simple Mail Transfer Protocol (SMTP) server.

Code Listing 7: Code segment of the transformation module to create a *Host* instance.

[55] https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/network_monitoring.py

[56] https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/alarm_notification.py

```
1  def _start(self):
2      while self.monitoring_status is True:
3          query = (
4                  'MATCH (h:Host)-[r:STARTS_CONNECTION]->(c:Connection) '
5                  'WHERE c.status = "active" AND h.notification_sent = "False"'
6                  'RETURN h,count(r) as count'
7                  )
8          result = self.neo4j_driver.execute_and_return_query_result(query)
9
10         if result is not None:
11             message = ""
12             for row in result:
13                 if row['count'] > 50:
14                     message = (
15                                 'Host ' + row['h']['ip_address'] '
16                                 'has ' +  str(row['count']) '
17                                 'active connections'
18                                )
19
20                     self.email_notification.connect_to_smtp_server()
21                     self.email_notification.send_email(message)
```

The code listing 7 demonstrates the function definition used to monitor the network information in the database. The function is designed to run an endless while loop as long as the module variable *monitoring_status* is set to be *True*. From line five to seven we can see the Chpyer query used to extract all active connections from each host present in the network. Subsequently, the query is executed and it is validated if it returns a result set. To check if a SlowITe attack has occurred a for loop is used which iterates over the returned result set and verifies if no host has more than 50 opened connections. If a host with more than 50 connection is detected an e-mail massage is created an send to and administrator, this can be seen in line sixteen to twenty two.
To realize the console included in the proposed architecture no dedicated Phyton module was developed. To interact with the system the user either has to use the CLI or fallback to the Kuberentes dashboard and the Neo4j browser interface. In figure 6 and 7 we can see screenshots of both applications. The first figure shows the Kuberentes dashboard with the corresponding deployments in the *rooms* namespace. On the left side there is a navigation panel to display different cluster resources. In the middle the health status of the deployments is shown. The second screenshot displays in the middle a Chyper query visualised as a graph. Next to the visualization statistics are shown about the instances existing in the database. In addition on the far right of the figure we can also see a navigation panel to administer the database instance.

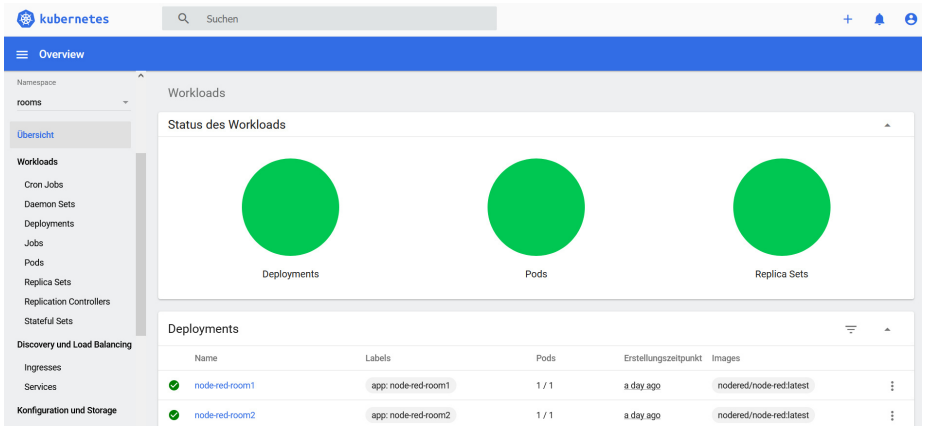

Figure 6: The Kuberntes Dashboard to interact with the cluster resources. Shown are the Node-Red deployments.

In figure 8, we can see the proposed data model populated with sample data, visualized with Neo4j. To create this sample instance, an additional module named *initialize_system*[57] has been implemented. The module has been introduced to validate the system capabilities. To create the instance data, the module depends on different JSON objects containing information about the topology model as well as known vulnerabilities.
Each node type is visualized with a different colour. In this instance, hosts are orange, services are blue, vulnerabilities are brown, attacks are green, preconditions are violet and connections are red[58]. On the left side of figure 8, we can see a host with the IP address *10.244.0.2* which is hosting an *Apache Web Server*. The *Apache Web Server* is a service which is associated with the vulnerability *Log4j*. The green attack node *ACE* requires as precondition *Network Access* which can be satisfied by exploiting the *Log4j* vulnerability. Gaining *Network Access* is also a precondition for the *DoS* attack node. Additionally, the attack requires a service which is using the *3.1.1 MQTT* protocol version. In our sample instance, this is the case because the attack node symbolizes a slow DoS attack which exploits the *SlowITe* vulnerability. The vulnerability SlowITe is connected to the *Mosquitto* service which is hosted on the machine with the IP address *10.244.0.5*. An adversary which has as the goal to preform a slow DoS attack on the *Mosquitto* service might therefore use the *Log4j* vulnerability to gain access to the network and afterwards execute the DoS from host *10.244.0.2*. On the right side of figure 8, two additional hosts are shown, one hosting the *Intrusion Detection System* and the other hosting the *Neo4j* database. We can see the connection node indicating an active connection between the detection

[57] `https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/initialize_system.py`

[58] Neo4j picks the colour schema of the nodes randomly.

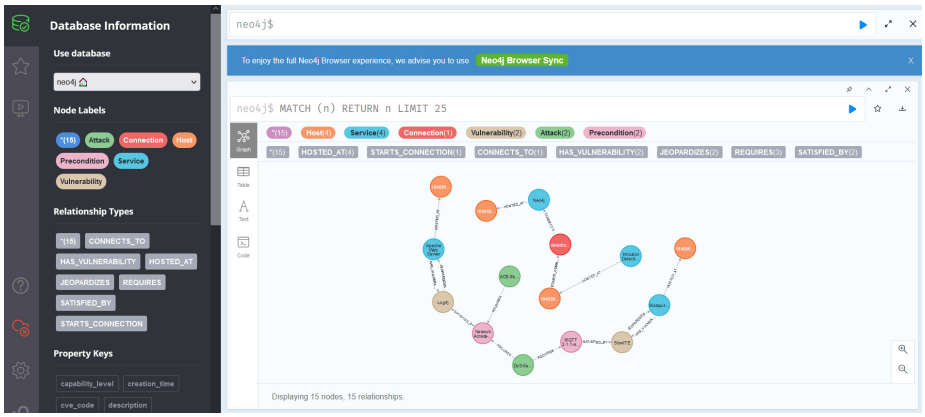

Figure 7: The Neo4j Browser Interface to query and administer the graph database. Shown is the visualization of a Chyper query.

system and the database[59].
The introduced data sample in this section will serve as a running example in the following section to illustrate the capabilities of the proposed system.

6 Test of System Capabilities

To test the capabilities of the data model and the implemented IDS prototype, the test bed introduced in section 4 and the instance data discussed in the previous section have been used. The following paragraphs provide details on the functionality, performance and usability limitations.

6.1 Functionality

Based on the proposed data model, the system cannot only be used to detect SlowITe attacks, it can also be used to increase system resilience by discovering attack pathways and set appropriate mitigation actions.
To detect a SlowITe attack, the built-in query language of Neo4j Cypher can be used. The following Cypher query returns the number of all active connection for each host:

```
MATCH (h:Host)-[r:STARTS_CONNECTION]->(c:Connection)
WHERE c.status = "active" AND h.notification_sent = "False"
AND h.is_blocked = "False"
RETURN h,count(r) as count
```

The query matches all hosts which have a "STARTS_CONNECTION" relation to a connection node where the status of the connection is active, a warning

[59]This specific connection was modelled manually. However, a connection from a host to the Mosquitto service will be generated automatically based on the log file.

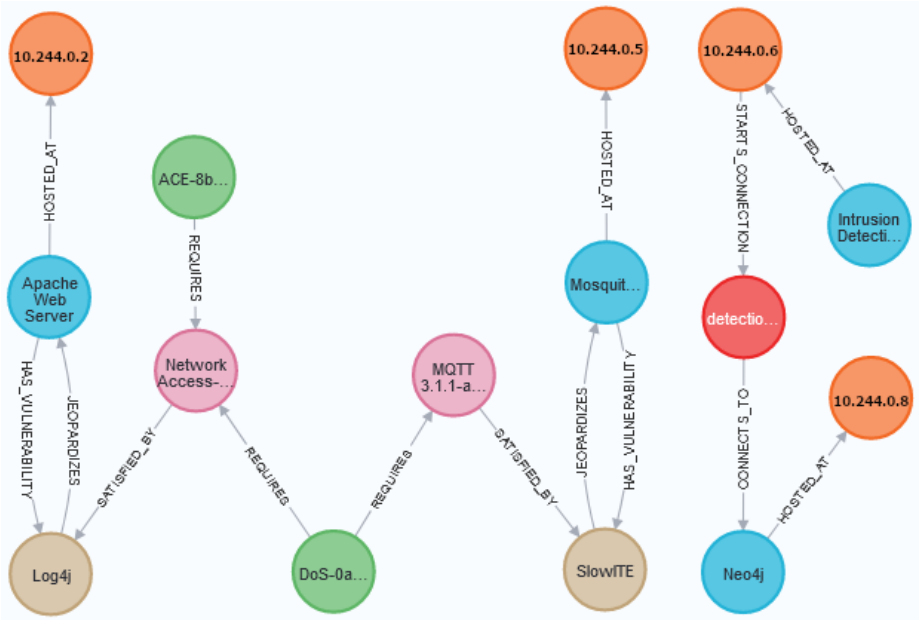

Figure 8: A data sample instance of the discussed data model in Neo4j

notification has not been sent yet and the host is not blocked. The statement returns a list of all hosts fitting the pattern with their corresponding count of active connections. As discussed earlier, a simple threshold rule has been implemented to identify a SlowITe attack; the threshold was set at 50 active connections. However, it could be adjusted to any number or a procedure could be developed to automatically calculate the threshold based on the number of connections established on the previous day. Since the query returns the number of active connections of all host, a distributed SlowITe attack can also be detected. average and standard deviation of the execution time.

In table 1 the result set is shown, which is returned by the previously intro-

"hosts"	**"count"**
{"creation_time":"1649594257","notification_sent":"False", "is_blocked":"False","ip_address":"10.244.0.5"}	51
{"creation_time":"1649594256","notification_sent":"False", "is_blocked":"False","ip_address":"10.244.0.8"}	1

Table 1: Result set of the query used to detect a SlowITe attack.

duced Chyper query. The query returns the host information as a JSON object. In the column named *count* the amount of active connections for each host is shown. Based on this column the SlowITe attack is detected. In our example a intrusion alert would be send since host *10.244.0.5* has 51 opened connections.

Under the assumption that the topology, inserted into the database, includes all known devices in the network, new unknown hosts trying to establish a connection to a service could immediately be blocked. This feature can prove crucial to detect SlowITe attacks performed from resource-constrained devices. Such devices are easy to place within the boarders of WLAN access points and could launch an attack from within the network. These devices will not be part of the original topology model provided for the database, hence connections from unknown devices can be easily detected.
As mentioned above, the proposed data model can also be used to find vulnerabilities in the network structure. Since the information model combines attack types, attack preconditions and system vulnerabilities to indicate potential attack pathways. A security manager can query the graph to verify whether services have been updated to a corresponding version or whether attacks pathways over multiple vulnerabilities can compromise the system. The following Cypher query could be used to investigate which vulnerabilities could be exploited to compromise a specific host:

```
MATCH (host1:Host {ip_address: "10.244.0.5"}),
      (vuls:Vulnerability),
      p = shortestPath((host1)-[*]-(vuls))
RETURN p
```

The query matches a specific host based on a specified IP address and searches for all shortest paths from the host to any vulnerability. In figure 9, we can see

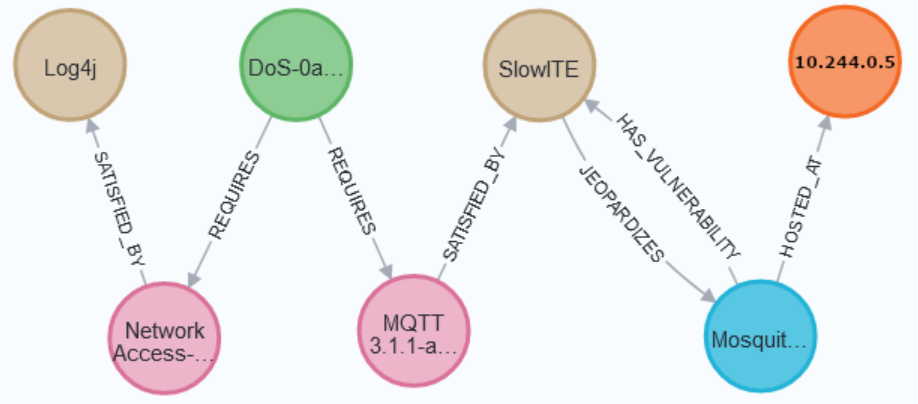

Figure 9: Query result of a shortest path analysis

the query result visualized with Neo4j as a graph. At host 10.244.0.5, a mosquito service is hosted which has the SlowITe attack as vulnerability. Additionally, the Log4j vulnerability is connected to the host via the precondition and attack nodes. We can infer that a DoS attack might be possible since both precondition are satisfied. However, from the current query, we cannot deduce which services might be jeopardized by the Log4j vulnerability. To further investigate, the following query can be used:

```
MATCH (n:Vulnerability {cve_code:"CVE-2021-44228"})
CALL apoc.path.subgraphNodes(n, {relationshipFilter:'>'}) YIELD node
RETURN node
```

The query matches the vulnerability with the provided CVE code and returns all paths which can be reached by outgoing relations like *JEOPARDIZES*. The call to *apoc.path.subraphNodes* executes a predefined procedure which can be customized by configuring predefined parameters. In this case, the *relationshipFilter* parameter was used to only utilize outgoing edges for the path analysis. In figure 10, we can see the resulting graph indicating that the Log4j vulnerability jeopardizes the Apache Web Server running on host 10.244.0.2. We can also observe two links between the service and vulnerability. This behavior is based on the procedure implementation which is reaching the service node *Apache Web Server* by following the relation *JEOPARDIZES*. The current path evaluation has arrived at the service node and starts again the search for outgoing relation from it's current node location. Consequently, the *HAS_VULNERABILITY* edge became as well a outgoing relation and was included in the result set. The query could be adjusted to dismiss the *HAS_VULNERABILITY* relation in the result set. However, for a manual investigation, the presented query is sufficient. The data structure also allows to start the investigation from the attack side and traverse the network to hosts which could be compromised. To perform such an analysis, the following Cypher query can be used:

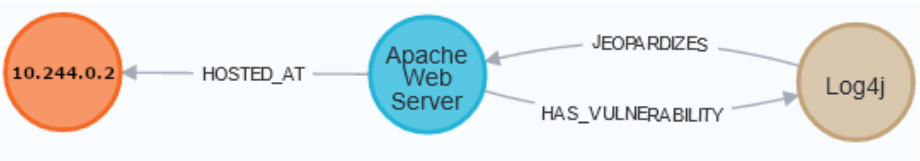

Figure 10: Query result of paths leading from a vulnerability to a host

```
MATCH (n:Attack {name:"DoS-0a61e6f1"})
CALL apoc.path.subgraphNodes(n, {relationshipFilter:'>'}) YIELD node
RETURN node
```

The query has a similar structure to the previous one but is instead matching for a vulnerability. This time the *DoS-0a61e6f1* attack node was used as the starting point for the analysis. In the *apoc.path.subgraphNodes* procedure, we use the filter to only return outgoing relations. We apply the filter to dismiss other attacks from the result set.[60] Figure 11 displays the resulting graph of the query. We can observe that the DoS attack requires two preconditions to be executed. Both conditions are satisfied by the vulnerabilities *Log4j* and *SlowITe*. These vulnerabilities are associated with services hosted on different host machines located in the same network. Hence, based on the query result, a security

[60] Without the filter, the attack node *ACE-8b79e45f* would also be included in the result set.

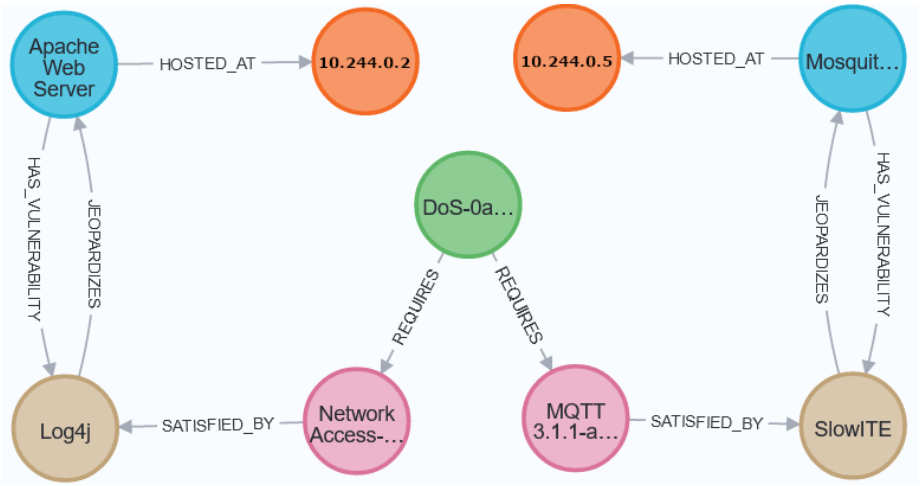

Figure 11: Query result of paths leading from an attack to different hosts

manager could address the threat by applying appropriate mitigation actions. Neo4j offers additional possibilities to analyse data and gain further insights. One notable feature is the Graph Data Science Library[61] which provides, for example, algorithms for node classification and community detection. However, the presented simple queries already demonstrate the efficiency of the proposed data structure. Consequently, this work will not cover additional analysis possibilities provided by Neo4j.
In the following subsection, the performance of the implemented system is discussed in detail.

6.2 Performance

To evaluate the performance of the prototypical implementation of the proposed IDS, the well known Big-O Notation will be used. The notation is widely applied in the computer scientific community to indicate the worst-case running time of an algorithm [14]. To calculate the time complexity of the current implementation, the modules which were identified as bottle necks are reviewed in pseudo code in the following paragraphs. Pseudo code abstracts code lines into natural language statement to establish a common understanding of the functionality of the code passages, independent of the programming language.
The most time-consuming modules of the implementation are the *mosquitto_log_transformation* and *network_monitoring* python files. As explained earlier, the firstly mentioned module handles the transformation of the Mosquitto log file and imports it to the Neo4j database. The auxiliary module *neo4j_database_access*, which is used by the transformation module, will not be considered in the following paragraphs because it is highly dependent on the database implementation which is

[61] `https://neo4j.com/developer/graph-data-science/graph-algorithms/`

not publicly available.
In algorithm 1, the execution logic of the *mosquitto_log_transformation* module

Algorithm 1: Pseudo code representation of `https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/mosquitto_log_transformation.py`

```
for line in mosquitto.log do
    if line is not equal to "Socket error on client" then
        extract host IP address from line;
        extract service port number from line;
        extract service version from line;
        extract date and time for connection and host from line;
        extract connection name from line;
        if host data is available then
            if host does not exist in database then
                create a new host instance in database;
            end
        end
        if service data is available then
            update a new service instance in database;
        end
        if connection data is available then
            if host is not blocked form establishing new connection then
                create a new connection instance in database;
                create a new relation between host and connection;
                create a new relation between connection and service;
            end
            if host or service is disconnecting then
                update connection status to inactive;
                update connection update time;
            end
        end
    end
end
```

is shown. The program iterates over all lines in a log file, as can be seen by the application of the for loop. Afterwards, there is a simple if statement, which was included to improve performance. The statement ensure that log lines containing socket error messages are not processed since they do not contain any information about the effected connections and hosts. The if statement is followed by a few function calls which are retrieving information form the current file line. Subsequently, the next if statement verifies whether sufficient data about a host is available to create it in the database. Before a new host is created, an additional check is made to control whether the host not already exists

in the database. To create a service, another if statement is used to validate the existence of the needed data. However, in contrast to hosts, the services are only updated and not created during run-time since they have been imported ahead with the topology model. Hosts, on the other hand, need to be created during run-time to detect potential new illegitimate hosts accessing a service. The following code segment is used to create connection nodes. Firstly, the if statement verifies whether sufficient data is present to create a connection node. Afterwards, since hosts can be blocked from establishing new connections, another if statement verifies the host status. If the validation succeeds, a new connection node as well as the corresponding relations are created in the database. The connection creation is followed by an additional if statement checking whether a host disconnected from a service. If the statement evaluates to true, the connection status is updated to be inactive and the update time is adjusted.
As discussed previously, the function calls in the code cannot be directly evaluated with regard to time complexity boundaries since their Big-O notations are not published. Hence, we analyse the structure of the code round the functions calls. The used for loop is the most time consuming procedure because the execution time increases with the number of lines provided. Algorithm 1 has therefore a linear time complexity notated as $O(n)$ in Big-O notation. Linear time complexities are a common complexity class in software engineering since iterating over a set of data is a widely encountered use case.
Algorithm 2 displays the pseudo code representation of the implemented network

Algorithm 2: Pseudo code representation of `https://github.com/ThorstenBigT/ids_slow_dos/blob/main/detection_system/code/network_monitoring.py`

```
while status equal to active do
    query database for the amount of active connections per host;
    for row in result set of query do
        extract host IP address from row;
        extract number of active connections from row;
        if number of active connections exceeds 50 then
            create e-mail message;
            send email to administrator;
            update host to be blocked;
        end
    end
    sleep for one second;
end
```

monitoring module. The algorithm starts with a while loop to check whether the monitoring service is set to be active. Afterwards, the database is searched with the query introduced in subsection 6.1. The query returns, as discussed,

a list of all hosts with their corresponding amount of active connections. The query operation is followed by a for loop which iterates through the return list row by row. Inside the for loop, there are two function calls to retrieve the IP address of the host and the associated number of active connections. Subsequently, the following if statement evaluates whether any host has more than 50 active connections. If evaluated to true, an intrusion has been identified. An e-mail is created containing the host details and forwarded to the responsible system administrator. In the last line, a sleep command is used to wait for the next evaluation round for one second.
The while loop in the code segment has a constant execution time since the input of the loop is a simple Boolean which does not increase in size over time. Consequently, the for loop iterating over the query result is the statement most expensive for the time complexity. The time complexity measure for the database function calls as well as for communicating with the SMTP server are not available and are therefore neglected. Similar to the firstly introduced algorithm, the for loop execution time increases linearly with the size of the input. More specifically, it rises with the list size returned by the database query. Therefore, the execution time can be described as $O(n)$ as well.
Even though the Big-O notation provides a good intuition about the general performance of the code segments, it is useful to conduct a performance test in milliseconds to provide a detailed analysis. In the current test scenario, additional performance information is especially useful since the implementation uses external libraries. The following paragraphs include results of conducted performance tests. During the tests, the execution time of the previously introduced bottle neck for loops were measured. The tests have been conducted with a *HP ProBook x360 440 G1* equipped with *16 GB RAM* and a *Intel(R) Core(TM) i5-8250U* CPU.
To measure the performance of the log transformation module (algorithm 1), log files with different sizes were submitted to the loop. The different file sizes were 1, 10, 100 and 1000 Mega Byte (MB). From each test scenario, 100 measurements were taken to calculate the median, average, as well as the standard deviation of the execution times. To evaluate the performance of the monitoring module (algorithm 2), the database has been filled with different topology models varying in the number of hosts, services and connections. Three different model sizes have been chosen: 100, 1 000 and 10 000. For each topology model, one host was created which has more than 50 open active connections to trigger the notification module. Each test scenario was repeated 10 times[62] to calculate the median, average and standard deviation of the execution time.

In table 2, we can see the median, mean and standard deviation recorded for algorithm 1. Even though the file size increases linearly, we can observe the median not changing significantly. The mean and standard deviation also do not change much for the first three file sizes. However, after reaching 1000 MB, both values increase substantially. The median remains stable since it is robust

[62]Because of the SMTP server's configuration, more tests could not be conducted since the requests were blocked based on the security policy applied by the e-mail provider.

File Size	Median	Mean	Standard Deviation
1 MB	38.52 ms	84.46 ms	68.81 ms
10 MB	42.80 ms	87.07 ms	73.38 ms
100 MB	40.06 ms	89.66 ms	75.46 ms
1 000 MB	42.66 ms	162.21 ms	200.68 ms

Table 2: Performance measured in ms for the Mosquitto log transformation module.

against outliers. The recorded behavior is likely caused by the occupied RAM space. To read the log file during run-time, they are kept open in RAM. Hence, the file blocks the memory for other computational operations. Depending on the available memory during a specific iteration, the function calls in the for loop may be processed significantly faster than a subsequent one. Furthermore, to transform the log statements, different CRUD operations have been used which need different amounts of resources. If, during one iteration, little memory is available and a computational intensive operations has to be made, the execution time increases significantly. In the previous test scenarios, there was always enough RAM available to handle these spikes.

The second performance test in table 3 was conducted with the network mon-

Number of Hosts	Number of Services	Number of Connections	Median	Mean	Standard Deviation
100	100	100	3 647.44 ms	5 618.86 ms	4 748.53 ms
1 000	1 000	1 000	3 592.14 ms	5 368.96 ms	3 286.49 ms
10 000	10 000	10 000	4 456.72 ms	6 213.67 ms	4 103.43 ms

Table 3: Performance measured in ms for network monitoring module.

itoring module. Recall that the module is in charge of checking the active connections per host and sending a notification once an anomaly has been detected. We can observe the median and mean being similar in the first two test scenarios, which indicates consistent database performance. The first test scenario has the highest standard deviation, which is most likely rooted in a higher fluctuation of the network speed during the test. Furthermore, the GMX[63] SMTP was used. Consequently, during the test the execution time to send an e-mail highly depends on the traffic arriving at the server as well as on its general configuration. The measured standard deviations decrease the expressiveness of the mean execution time since the measurements are far spread from the mean. The results of the third test indicate a significant increase of the median and mean execution time which might be caused by the processing time of the used query. To further investigate the query performance, a second test has been

[63] `https://hilfe.gmx.net/pop-imap/pop3/serverdaten.html`

conducted measuring only the execution time of the Cypher query itself. The

Number of Hosts	Number of Services	Number of Connections	Median	Mean	Standard Deviation
100	100	100	16.00 ms	16.75 ms	11.01 ms
1 000	1 000	1 000	65.78 ms	68.02 ms	13.19 ms
10 000	10 000	10 000	1 142.59 ms	1 196.11 ms	219.89 ms

Table 4: Performance measured in ms for Cypher query used in network monitoring module.

result of this query performance test can be seen in table 4. The measurements were this time taken again 100 times since the restrictions of the SMTP server were not present anymore. The table displays a steady increase in the execution time along with the number of instances in the database. The standard deviation indicates less variation in the measurements compared to the second performance test conducted with the whole network monitoring module. Hence, the previous deviation can be concluded to be caused by the network speed and the SMTP server. In scenario three with 10 000 instances, we can see a large increase in the execution time of the query. In consequence, the higher median and mean in table 3 have probably been caused by a longer processing time of the query instead of random network fluctuations.
The results indicate a sufficient performance for a prototype implementation. There should be no performance issues with topology models including up to 1000 network devices. Database performance will increase as well if a dedicated database server is used. In the current performance test, all modules and applications ran on the same host machine. As the performance test indicates performance issue may arise in large topology models above 10 000 host machines. However, performance issue will appear already with smaller network typologies since vulnerabilities, attacks and preconditions have not been included in the performance test. Consequently, encapsulating all knowledge about attack patterns and associated vulnerabilities might prove inefficient over time and a suitable retention policy has to be defined to maintain database performance.
In the following section, the results of the presented work are discussed in detail and critically reviewed. The section firstly introduces achieved goals followed by identified limitations and closes with further development options to address the limitations.

7 Results and Discussion

In this section, we will revisit the formulated research questions and address them by reviewing the presented results in section 6. Firstly, the achieved goals are summarized to introduce limitations of the data model afterwards. The

section concludes with further development options to enhance the data model and the proposed IDS.

7.1 Achieved Goals

To validate the system under realistic conditions, a test bed has been implemented to simulate a network of sensors. The test best was realized by combining Node-Red with Kubernetes. Node-Red was used to create different flows to simulate sensor data while Kubernetes was used to handle the network routing and deployment of the implemented services. The test bed can be easily extend by adding new Node-Red flows to simulate more sensor data, by adjusting the function node to create custom sensor data or by adding additional Node-Red instances. Furthermore, Node-Red includes node pallets to use different communication protocols. Therefore, more experienced users can fully customize the capabilities of the test bed to their needs. By containerizing the test bed applications with docker, the reusabiliy and portability of the test bed is ensured.
The results presented in section 6 prove that the introduced data model alongside with the constructed IDS is capable to detect SlowITe attacks. To identify the attack, a simple Cypher has been proposed. Integrating the information encapsulated in the Mosquitto log file proved to be sufficient to keep track of active connections and detect SlowITe attacks. Furthermore, since all active connections of each hist can be tracked, the introduced data model can also be used to detect distributed SlowITe attacks.
To further enhance the capability of the IDS, the data model includes concepts usually encountered in attack graph generation models. However, the proposed data model is not used to generate an attack graph. It utilizes concepts used in the domain to combine them with network topology information to increase system resilience and enhance detection capabilities. A security expert can conduct meaningful analysis by utilizing the features provided by Neo4j. Manually traversing the graph with Cypher queries proves to be efficient to discover potential system threats.

7.2 Usability Limitations

The presented implementation of the IDS in section 5 contains some usability limitations since it was developed as a proof of concept system to validate the proposed data model. The following usability constraints are the most notable ones:

- There is no support for other log files. In consequence, the usability is currently limited to the Mosquitto broker implementation provided by Eclipse.
- The creation of the topology model as well as the attack patterns are currently manual tasks which involve a domain expert. The current im-

plementation provides no interface to graphically model or automate the creation of these models. Therefore, a domain expert can not import or update the models without adjusting the program code.

- Administering and monitoring the current IDS is only possible via the Kubernetes dashboard since a dedicated console was not implemented. Nevertheless, the dashboard provided by Kubernetes enables also inexperienced users to interact easily with the deployed applications.
- Interacting with the Neo4j database is only possible with the Neo4j browser application or via the provided code. Hence, there is no dedicated visualisation or exploration feature available to analyse the stored data without defining Cypher queries.
- Configuring file paths or application passwords has to be done in the code base. There is no command line interface or configuration file provided to adjust the system during run-time or initialization.
- The IDS was implemented and tested on a Windows host machine. Even though Docker and Kubernetes are designed to run on the most common operation systems, application configurations need to be adjusted slightly. Therefore, the local volume mounts defined for the applications in the kind cluster configuration file are Windows-specific and have to be adjusted when ported to a different operation system.

The presented limitations are focused on the usability of the implementation. In following subsection, limitations concerning the proposed data model are discussed in detail.

7.3 Limitations of the Data Model

In section 7.2, limitations regarding the usability of the prototype were already introduced. The following paragraphs focus on the limitations of the proposed data model:

- Currently, the data model does not include information about the raw network traffic transmitted. The information could be included by adding more properties to the connection node. There are different tools available to analyze network packages and save the produced statistics persistently in a file. One of the most widely used tools is the open-source project Wireshark[64]. However, these tools analyze raw network traffic which includes network packages from various communication protocols. Integrating all of the provided information is a time consuming task since each protocol includes different concepts with specific semantics which have to be appropriately represented in the data model. For example, Diederichsen et al.

[64] `https://www.wireshark.org/`

[16] proposed a graph representation of the HTTP protocol. Such models could be integrated into the current data model to also enable detailed analysis of each protocol.

- To analyze attack pathways, usually the system entry point as well as the consecutive order of vulnerabilities used to compromise the system are modeled as well. This information is crucial to identify potential multi-stage and multi-host attacks [6]. However, the introduced data model does not include any temporal information about the execution order to successfully apply an attack. Nonetheless, a big variety of attack paths can be discovered by the path analysis capabilities provided by Neo4j.

- Network topology models usually consist of a variety of different hardware and software modules like switches, access points, firewalls, etc. The data model does not distinguish between these different concepts. Currently, all these hardware devices would have to be modelled as services hosted on a device. Depending on the actual need of the security experts, these concepts could be added. Based on the investigation requirements, suitable database queries might not performant based on the model restriction.

- The data model includes no possibility to store host or service configurations. Even though service nodes have properties to indicate the version and protocol used, the information is rather rudimentary and often not sufficient to evaluate the security risk. Many applications provide a configuration file where plenty of custom settings can be implemented. For example, host machines with different operation systems provide a complex set of possible configurations which could all be exposed to a specific vulnerability. However, integrating this massive amount of configuration data will result in performance issues and potentially decrease the usability of the data model. The current model was designed to provide efficient support to identify potential security risks, to investigate them and set appropriate counter measures. However, identifying appropriate counter measures is still a manual task needed to be done by an security expert.

- Security policies are a crucial part of an organisation's security framework. Security policies are usually defined in semi-structured or unstructured documents, which makes it cumbersome to integrate them into existing system. Therefore, detecting violations against them can be difficult. The presented data model was not designed to include information about defined security policies. Even though a lot of benefits would result from an integration, defining a model which encapsulates the relevant information is a non-trivial task and was out of scope for the current approach.

The presented list provides an overview of the most crucial limitations identified by the author. In the following section, further development options will be presented to provide suggestions to overcome some of the presented limitations.

7.4 Further Development Options

To address the presented limitations in section 7.2 and 7.3, further implementation options to enhance the prototype and the data model are presented in the following paragraphs.
One option to enhance the IDS is to add more transformation modules to cover a broader variety of log files. To achieve this, separate modules for each potential log file type should be implemented. Each module encapsulates the service-specific logic to transform the log statements appropriately and import them into the graph database. To improve efficiency, an abstract class can be designed which defines the general class properties and functions. A more sophisticated solution is to rely on semantic web and implement the introduced solution of Ekelhart et al. [18]. Their approach solves the heterogeneity problem by using an RDF based event extractor. The event extractor provides a SPARQL Protocol and RDF Query Language (SPARQL) endpoint which can be queried by an event integrator module. To adopt these concepts, the current transformation module could be adjusted to use SPARQL to get the desired information from the log files. Neo4j provides an RDF import option, hence the data retrieved form the event extractor could also be directly imported. However, the RDF data model will be different from the one proposed in this work.
A big increase in the usability of the tool could be achieved by automating the creation of the network topology model. There are tools available to scan a network system and create a topology model from the extracted information. An open-source tool which provides such functionality is GRASSMARLIN[65]. It can produce reports which include all host IP addresses in a network and exports them as a Comma Sepearted Values (CSV) file. Another tool to scan the network is Nmap[66] which additionally provides port scanning and service detection alongside with their current version. The gathered information can be exported as an Extensible Markup Language (XML) file. Both tools provide structured format outputs which could be processed by an additional transformation module. The module could extract the needed information and use different CRUD operations to propagate the data to the database.
Similarly, to support the discovery of vulnerabilities and connect them to the corresponding services OpenVAS [67] could be used. The tool was designed to verify authentication mechanisms and support various different Internet and industrial protocols. Greenbone[68], who developed the tool, also maintains a security feed to keep track of newly discovered vulnerabilities. The tool is capable to produce reports in different structured formats, like XML, which in return could be transformed and imported into Neo4j by a dedicated module.
To extend the data model, the most beneficial aspect would be to integrate network traffic information. Network traffic can be tracked by tools like Wire-

[65] https://github.com/nsacyber/GRASSMARLIN
[66] https://nmap.org/
[67] http://www.openvas.org/
[68] https://www.greenbone.net/en/

shark[69] or Zeek https://zeek.org/. These tools provide the capabilities to produce reports and store them in different formats as well. However, integrating the information to the corresponding connections is not straight forward since the raw network traffic contains information about all protocols used in the network environment. Therefore, the information gathered has to be appropriately filtered and mapped to the correct instance in the graph database.
The presented extensions have been identified as most beneficial in terms of expanding the capabilities of the proposed data model and the usability of the IDS. However, to lead the prototype to maturity, all implemented modules would have to be reviewed and adjusted. Furthermore, IDSs are usually shipped with a dedicated console to provide an administration interface for security managers. Therefore, including the presented features is not sufficient to produce a tool appropriate for usage in real-world environments.

8 Conclusion

The introduced data model together with the implemented IDS proved to be effective to detect SlowITe attacks and thus increase network security of IoT and IIoT environments. More specifically, to address the research questions in section 1.3, the following conclusions can be drawn:

- The introduced architecture in section 3 alongside with the proposed data model has been implemented and tested. The results indicate that the chosen design is effective to detect slow DoS attacks. However, the detection capabilities only have been verified by detecting SlowITe attacks as introduced by [44]. The data model has not been tested with other types of slow DoS attacks. Slow DoS attacks exploit protocol specific vulnerabilities. Therefore, the data model would have to be extended to identify other attacks which target other protocol standards. Even if the data model would be sufficient, at least a broader support of log file interrogations would have to be implemented.
  Nevertheless, integrating log events into a graph data model proved to be a suitable approach to automatically analyze them. Moreover, the proposed method to detect SlowITe attacks based on the amount of active connections might prove efficient for other DoS attacks as well. To further validate this hypothesis, additional tests with different attacks need to be performed.

- By modelling attack patterns and associating them with services in the topology, domain knowledge could not only be modelled with the label property graph model, but the appropriate context to run meaningful analysis could also be provided. In comparison to traditional relational data models the label property graph data model has following advantages:

[69] https://www.wireshark.org/

  - Network context data is highly-connected data with many dependencies to each other. Modelling this type of data in tables would result in performance issues since expensive "JOIN" operations would have to be used to analyse them.
  - Network topology and associated vulnerabilities change over time and might require changes in the data model. Relational models are being implemented with a predefined database schema, once set up they are hard to change. Graph data models are made to handle frequent changes in the data model which is crucial to adapt to changes in the network and associated vulnerabilities.
  - Graph models are perfectly suited to exploit graph analytical algorithms such as shortest path analysis between two network nodes. These algorithms are not available in relational databases by default.

- For the introduced approach following challenges remain:
  - One of the major challenges of the introduced approach is the amount of effort needed to model the topology and the associated vulnerabilities. As addressed in section 7.4, further improving the system to generate parts of the model automatically would reduce this time consuming task. However, a security expert would still have to check the generated model to prevent mistakes which could cause serious security breaches.
  - Another challenges of the current approach is to reduce the integration time of further log files produced by different protocols and services. Each service potentially needs its own transformation module. Unless, as mentioned in section 7.4, a general log event extractor as described by Ekelhart et al. [18] is used.
  - Currently, the detection module uses a threshold based method to detect SlowITe attacks. Other IDSs usually combine a number of different approaches to detect potential intrusions. Hence, they can detect a big variety of intrusions. The current approach has been designed to detect SlowITe attacks and has not been used to detect other attacks. Therefore the challenge remains to generalize the detection approach to broaden it's applicability.

As presented in this work, graph data structures can be a suitable and efficient choice to increase network security in IoT and IIoT environments. The approach chosen in this thesis also provided sufficient evidence that the combination of topology models and graph patters yields significant benefits to improve system resilience. Additionally, the conducted validation with the implemented test bed indicates the practical relevance of the introduced methods.
Based on the implications drawn from the research questions, future research should concentrate on solving the heterogeneity problem caused by the lack of standardization of log file events. As explained above, approaches like Ekelhart et al. [18] are very promising to solve this issue. Furthermore, sufficient tool

support to create topology models is still missing. Even though there are tools to retrieve the network structure, embedding tools to integrate these reports in existing graph databases are not present. The same applies for proper interfaces to convert vulnerability reports into graph databases. A goal of this thesis was the integration of attack pattern into a topology model. Further effort should be taken to merge both modelling domain. Specifically, the literature on attack graph generation is well established but lacks approaches to integrate attack patterns into IDSs to enhance their detection capabilities. Another gap in the scientific knowledge is the lack of papers published to investigate slow DoS attacks in detail. Only a few papers exist, which attempt to create a common understanding of the domain by providing taxonomies [12]. Even fewer authors dedicated their work to investigate how these attacks can be mitigated. This thesis, however, provides results that indicated that these attacks can be used to deny access to essential services. Therefore, to secure IoT and IIoT environments, sophisticated methods have to be developed to mitigate these rather new threats.

List of Figures

Listings

List of Tables

List of Algorithms

References

[1] Renzo Angles. The Property Graph Database Model. In *AMW*, 2018.

[2] P. Ramesh Babu, D. Lalitha Bhaskari, and C. H. Satyanarayana. A comprehensive analysis of spoofing. *International Journal of Advanced Computer Science and Applications*, 1(6):157–62, 2010.

[3] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

[4] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. MQTT Version 5.0, March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[5] Andrew Banks and Rahul Gupta. MQTT Version 3.1.1, October 2014. OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[6] Mridul Sankar Barik and Chandan Mazumdar. A Graph Data Model for Attack Graph Generation and Analysis. In Gregorio Martínez Pérez, Sabu M. Thampi, Ryan Ko, and Lei Shu, editors, *Recent Trends in Computer Networks and Distributed Systems Security*, Communications in Computer and Information Science, pages 239–250, Berlin, Heidelberg, 2014. Springer.

[7] Tim Bass. Multisensor data fusion for next generation distributed intrusion detection systems. In *Proceedings of the IRIS National Symposium on Sensor and Data Fusion*, volume 24, pages 24–27. Citeseer, 1999. Issue: 28.

[8] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *arXiv:1910.09017 [cs]*, April 2020. arXiv: 1910.09017.

[9] Mitko Bogdanoski, Tomislav Suminoski, and Aleksandar Risteski. Analysis of the SYN Flood DoS Attack. *International Journal of Computer Network and Information Security (IJCNIS)*, 5(8):1–11, June 2013. Number: 8 Publisher: MECS Publisher.

[10] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (IIoT): An analysis framework. *Computers in Industry*, 101:1–12, October 2018.

[11] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The diamond model of intrusion analysis. Technical report, Center For Cyber Intelligence Analysis and Threat Research Hanover Md, 2013.

[12] Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Slow DoS attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications*, September 2013. Publisher: Inderscience Publishers Ltd.

[13] B. Claise. Cisco Systems NetFlow Services Export Version 9. Technical Report RFC3954, RFC Editor, October 2004.

[14] Thomas H. Cormen, editor. *Introduction to algorithms*. MIT Press, Cambridge, Mass, 3rd ed edition, 2009. OCLC: ocn311310321.

[15] Evan Damon, Julian Dale, Evaristo Laron, Jens Mache, Nathan Land, and Richard Weiss. Hands-on denial of service lab exercises using SlowLoris and RUDY. In *Proceedings of the 2012 Information Security Curriculum Development Conference*, InfoSecCD '12, pages 21–29, New York, NY, USA, October 2012. Association for Computing Machinery.

[16] Lars Diederichsen, Kim-Kwang Raymond Choo, and Nhien-An Le-Khac. A Graph Database-Based Approach to Analyze Network Log Files. In Joseph K. Liu and Xinyi Huang, editors, *Network and System Security*, Lecture Notes in Computer Science, pages 53–73, Cham, 2019. Springer International Publishing.

[17] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, April 2004.

[18] Andreas Ekelhart, Elmar Kiesling, and Kabul Kurniawan. Taming the logs - Vocabularies for semantic security analysis. *Procedia Computer Science*, 137:109–119, January 2018.

[19] Mica R. Endsley. Design and evaluation for situation awareness enhancement. In *Proceedings of the Human Factors Society annual meeting*, volume 32, pages 97–101. Sage Publications Sage CA: Los Angeles, CA, 1988. Issue: 2.

[20] European Commission. Directorate General for the Information Society and Media. *Vision and challenges for realising the Internet of things.* Publications Office, LU, 2010.

[21] B. Foo, Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. Spafford. ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 508–517, June 2005. ISSN: 2158-3927.

[22] Jorge E. Ibarra-Esquer, Félix F. González-Navarro, Brenda L. Flores-Rios, Larysa Burtseva, and María A. Astorga-Vargas. Tracking the evolution of the internet of things concept across different application domains. *Sensors*, 17(6):1379, 2017. Publisher: Multidisciplinary Digital Publishing Institute.

[23] Anya Kim, Jim Luo, and Myong Kang. Security ontology for annotating resources. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1483–1499. Springer, 2005.

[24] Graham Klyne. Resource description framework (rdf): Concepts and abstract syntax. *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/*, 2004.

[25] R. K. Kodali and S. Soratkal. MQTT based home automation system using ESP8266. In *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 1–5, December 2016.

[26] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, 35:100219, February 2020.

[27] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013. Publisher: Elsevier.

[28] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. *The Internet of Things: Mapping the value beyond the hype*, volume 24. McKinsey Global Institute New York, NY, USA, 2015.

[29] Surajit Medhi and Hemanta K Baruah. Relational database and graph database: A comparative analysis. *Journal of Process Management. New Technologies*, 5(2):1–9, 2017.

[30] B. Mishra and A. Kertesz. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access*, 8:201071–201086, 2020. Conference Name: IEEE Access.

[31] Bruno A. Mozzaquatro, Ricardo Jardim-Goncalves, and Carlos Agostinho. Towards a reference ontology for security in the Internet of Things. In *2015 IEEE International Workshop on Measurements Networking (M N)*, pages 1–6, October 2015.

[32] Bruno Augusti Mozzaquatro, Carlos Agostinho, Diogo Goncalves, João Martins, and Ricardo Jardim-Goncalves. An Ontology-Based Cybersecurity Framework for the Internet of Things. *Sensors*, 18(9):3053, September 2018. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[33] Eliseu Pereira, Rui Pinto, Joao Reis, and Gil Gonçalves. MQTT-RD: A MQTT based Resource Discovery for Machine to Machine Communication. In *IoTBDS*, pages 115–124, 2019.

[34] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[35] Simona Ramanauskaite and Antanas Cenys. Taxonomy of DoS attacks and their countermeasures. *Central European Journal of Computer Science*, 1(3):355, September 2011.

[36] Sebastian Roschke, Feng Cheng, and Christoph Meinel. High-quality attack graph-based IDS correlation. *Logic Journal of the IGPL*, 21(4):571–591, August 2013. Conference Name: Logic Journal of the IGPL.

[37] Thijs Rozekrans, Matthijs Mekking, and Javy de Koning. Defending against DNS reflection amplification attacks. *University ofAmsterdam, Tech. Rep.*, February 2013.

[38] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). Technical Report NIST Special Publication (SP) 800-94, National Institute of Standards and Technology, February 2007.

[39] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[40] Guttorm Sindre and Andreas L. Opdahl. Templates for misuse case description. In *Proceedings of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001), Switzerland*. Citeseer, 2001.

[41] Frans N Stokman and Pieter H de Vries. Structuring knowledge in a graph. In *Human-computer interaction*, pages 186–206. Springer, 1988.

[42] NSFOCUS Technologies. 2019 Annual IoT Security Report, 2019. NSFOCUS Technologies Group Co., Ltd.

[43] Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 113–135. Springer, 2003.

[44] Ivan Vaccari, Maurizio Aiello, and Enrico Cambiaso. SlowITe, a Novel Denial of Service Attack Affecting MQTT. *Sensors (Basel, Switzerland)*, 20(10), May 2020.

[45] Ivan Vaccari, Maurizio Aiello, and Enrico Cambiaso. SlowTT: A Slow Denial of Service against IoT Networks. *Information*, 11(9):452, September 2020. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[46] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, September 2006.

[47] Marilyn Wolf and Dimitrios Serpanos. Safety and Security in Cyber-Physical Systems and Internet-of-Things Systems. *Proceedings of the IEEE*, 106(1):9–20, January 2018. Conference Name: Proceedings of the IEEE.

[48] Guangquan Xu, Yan Cao, Yuanyuan Ren, Xiaohong Li, and Zhiyong Feng. Network Security Situation Awareness Based on Semantic Ontology and User-Defined Rules for Internet of Things. *IEEE Access*, 5:21046–21056, 2017. Conference Name: IEEE Access.

[49] Yaofang Zhang, Bailing Wang, Chenrui Wu, Xiaojie Wei, Zibo Wang, and Guohua Yin. Attack Graph-Based Quantitative Assessment for Industrial Control System Security. In *2020 Chinese Automation Congress (CAC)*, pages 1748–1753, November 2020. ISSN: 2688-0938.

[50] Zuopeng Justin Zhang. Graph Databases for Knowledge Management. *IT Professional*, 19(6):26–32, November 2017. Conference Name: IT Professional.

[51] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980. Conference Name: IEEE Transactions on Communications.

[52] V. Zlomislić, K. Fertalj, and V. Sruk. Denial of service attacks: An overview. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, June 2014. ISSN: 2166-0727.

Appendices

A Code Base Test Bed

This appendix contains all code used to implement the test bed introduced in 4.

A.1 Node-Red

A.1.1 nodered-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: node-red-config
  namespace: rooms
data:
  settings.js: |
    var fs = require("fs");
    module.exports = {
      uiPort: process.env.PORT || 1880,
      mqttReconnectTime: 15000,
      serialReconnectTime: 15000,
      debugMaxLength: 1000,
      adminAuth: {
        type: "credentials",
        users: [{
          username: "admin",
          password: "$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
          permissions: "*"
        }]
      },
      httpNodeCors: {
        origin: "*",
        methods: "GET,PUT,POST,DELETE"
      },
      functionGlobalContext: {},
      exportGlobalContextKeys: false,
      logging: {
        console: {
          level: "info",
          metrics: false,
          audit: false
```

```
      }
    },
    editorTheme: {
      projects: {
                  // To enable the Projects feature, set this value to true
        enabled: true
      }
    }
  }
```

A.1.2 nodered-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-red-room1
  labels:
    app: node-red-room1
  namespace: rooms
spec:
  selector:
    matchLabels:
      app: node-red-room1
  template:
    metadata:
      labels:
        app: node-red-room1
    spec:
      containers:
      - name: node-red-room1
        image: nodered/node-red:latest
        ports:
        - containerPort: 1880
          name: node-red-ui
        securityContext:
          privileged: true
        volumeMounts:
        - name: node-red-data-room1
          mountPath: /data
        - name: node-red-config
          mountPath: /data/settings.js
          subPath: settings.js
```

```
        env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: TZ
          value: Europe/Berlin
      volumes:
      - name: node-red-data-room1
        persistentVolumeClaim:
          claimName: node-red-room1-pvc
      - name: node-red-config
        configMap:
          name: node-red-config
```

A.1.3 nodered-pcv.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: node-red-room1-pvc
  namespace: rooms
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

A.1.4 nodered-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: node-red-room1
  namespace: rooms
spec:
  selector:
    app: node-red-room1
  type: NodePort
  ports:
```

```
  - name: node-red-ui
    port: 1880
    protocol: TCP
    targetPort: 1880
    nodePort: 30880
```

A.1.5 flows.json

```
[
   {
      "id":"7c7dc628.08a158",
      "type":"tab",
      "label":"Sensors data",
      "disabled":false,
      "info":""
   },
   {
      "id":"1c4074bd.e3bf8b",
      "type":"function",
      "z":"7c7dc628.08a158",
      "name":"loop",
      "func":"function getRandomInt(min, max) {\n  return Math.floor(Math.random() * (max - min +1)) + min;\n}\n\
      "outputs":"2",
      "noerr":0,
      "initialize":"",
      "finalize":"",
      "libs":[

      ],
      "x":470,
      "y":140,
      "wires":[
         [
            "7b60d417.849f2c",
            "a44244962e51c724"
         ],
         [
            "a380b8b5.5c7f48"
         ]
      ]
   },
   {
```

```
    "id":"a380b8b5.5c7f48",
    "type":"delay",
    "z":"7c7dc628.08a158",
    "name":"",
    "pauseType":"delay",
    "timeout":"1",
    "timeoutUnits":"hours",
    "rate":"1",
    "nbRateUnits":"",
    "rateUnits":"second",
    "randomFirst":"1",
    "randomLast":"5",
    "randomUnits":"seconds",
    "drop":false,
    "allowrate":false,
    "x":470,
    "y":267,
    "wires":[
        [
            "1c4074bd.e3bf8b"
        ]
    ]
},
{
    "id":"7b60d417.849f2c",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":690,
    "y":60,
    "wires":[

    ]
},
{
```

```
"id":"acbea390c095e2b3",
"type":"scheduler",
"z":"7c7dc628.08a158",
"outtopic":"room1/temperature",
"outpayload1":"start",
"outpayload2":"stop",
"name":"Temperture Sensor Scheduler",
"lat":"48.210033",
"lon":"16.363449",
"start":"sunrise",
"end":"sunset",
"starttime":"480",
"endtime":"1020",
"duskoff":"0",
"dawnoff":"0",
"outtext1":"",
"outtext2":"",
"sun":false,
"mon":true,
"tue":true,
"wed":true,
"thu":true,
"fri":true,
"sat":false,
"jan":true,
"feb":true,
"mar":true,
"apr":true,
"may":true,
"jun":true,
"jul":true,
"aug":true,
"sep":true,
"oct":true,
"nov":true,
"dec":true,
"repeat":false,
"atstart":true,
"x":200,
"y":80,
"wires":[
   [
      "1c4074bd.e3bf8b"
```

```
      ],
      [

      ],
      [

      ]
    ]
  },
  {
    "id":"a44244962e51c724",
    "type":"mqtt out",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/temperature",
    "qos":"",
    "retain":"",
    "respTopic":"",
    "contentType":"",
    "userProps":"",
    "correl":"",
    "expiry":"",
    "broker":"7fa102c41ccc6e44",
    "x":690,
    "y":140,
    "wires":[

    ]
  },
  {
    "id":"8679912d16fd9495",
    "type":"mqtt in",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/fan/speed",
    "qos":"2",
    "datatype":"auto",
    "broker":"7fa102c41ccc6e44",
    "nl":false,
    "rap":true,
    "rh":0,
    "x":340,
    "y":720,
```

```
    "wires":[
       [
          "abee4fdfb2b4ad50"
       ]
    ]
 },
 {
    "id":"abee4fdfb2b4ad50",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":590,
    "y":720,
    "wires":[

    ]
 },
 {
    "id":"f463b190bbeece7f",
    "type":"function",
    "z":"7c7dc628.08a158",
    "name":"loop",
    "func":"function getRandomInt(min, max) {\n  return Math.floor(Math.random() * (max - min +1)) + min;\n}\n\
    "outputs":"2",
    "noerr":0,
    "initialize":"",
    "finalize":"",
    "libs":[

    ],
    "x":450,
    "y":500,
    "wires":[
       [
          "4c9f0cd7386b28f3",
```

```
            "09b877f73ea6fed5"
         ],
         [
            "e65f4f0e0befd6d8"
         ]
      ]
   },
   {
      "id":"e65f4f0e0befd6d8",
      "type":"delay",
      "z":"7c7dc628.08a158",
      "name":"",
      "pauseType":"delay",
      "timeout":"10",
      "timeoutUnits":"minutes",
      "rate":"1",
      "nbRateUnits":"",
      "rateUnits":"second",
      "randomFirst":"1",
      "randomLast":"5",
      "randomUnits":"seconds",
      "drop":false,
      "allowrate":false,
      "x":460,
      "y":627,
      "wires":[
         [
            "f463b190bbeece7f"
         ]
      ]
   },
   {
      "id":"4c9f0cd7386b28f3",
      "type":"debug",
      "z":"7c7dc628.08a158",
      "name":"",
      "active":false,
      "tosidebar":true,
      "console":false,
      "tostatus":false,
      "complete":"true",
      "targetType":"full",
      "statusVal":"",
```

```
    "statusType":"auto",
    "x":670,
    "y":420,
    "wires":[

    ]
},
{
    "id":"1be9cb41e67649d8",
    "type":"scheduler",
    "z":"7c7dc628.08a158",
    "outtopic":"room1/fan/speed",
    "outpayload1":"start",
    "outpayload2":"stop",
    "name":"Fan Sensor Scheduler",
    "lat":"48.210033",
    "lon":"16.363449",
    "start":"sunrise",
    "end":"sunset",
    "starttime":"480",
    "endtime":"1020",
    "duskoff":"0",
    "dawnoff":"0",
    "outtext1":"",
    "outtext2":"",
    "sun":false,
    "mon":true,
    "tue":true,
    "wed":true,
    "thu":true,
    "fri":true,
    "sat":false,
    "jan":true,
    "feb":true,
    "mar":true,
    "apr":true,
    "may":true,
    "jun":true,
    "jul":true,
    "aug":true,
    "sep":true,
    "oct":true,
    "nov":true,
```

```
    "dec":true,
    "repeat":false,
    "atstart":true,
    "x":160,
    "y":440,
    "wires":[
        [
            "f463b190bbeece7f"
        ],
        [

        ],
        [

        ]
    ]
},
{
    "id":"09b877f73ea6fed5",
    "type":"mqtt out",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/fan/speed",
    "qos":"",
    "retain":"",
    "respTopic":"",
    "contentType":"",
    "userProps":"",
    "correl":"",
    "expiry":"",
    "broker":"7fa102c41ccc6e44",
    "x":670,
    "y":500,
    "wires":[

    ]
},
{
    "id":"be58ed542515fbbb",
    "type":"mqtt in",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/temperature",
```

```
    "qos":"2",
    "datatype":"auto",
    "broker":"7fa102c41ccc6e44",
    "nl":false,
    "rap":true,
    "rh":0,
    "x":390,
    "y":340,
    "wires":[
        [
            "0c6012c330f77c6e"
        ]
    ]
},
{
    "id":"0c6012c330f77c6e",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":590,
    "y":340,
    "wires":[

    ]
},
{
    "id":"179dd1102686daae",
    "type":"inject",
    "z":"7c7dc628.08a158",
    "name":"",
    "props":[
        {
            "p":"payload"
        },
        {
```

```
            "p":"topic",
            "vt":"str"
         }
      ],
      "repeat":"",
      "crontab":"",
      "once":false,
      "onceDelay":0.1,
      "topic":"room1/temperature",
      "payload":"stop",
      "payloadType":"str",
      "x":200,
      "y":160,
      "wires":[
         [
            "1c4074bd.e3bf8b"
         ]
      ]
   },
   {
      "id":"58cabb92533c62ad",
      "type":"inject",
      "z":"7c7dc628.08a158",
      "name":"",
      "props":[
         {
            "p":"payload"
         },
         {
            "p":"topic",
            "vt":"str"
         }
      ],
      "repeat":"",
      "crontab":"",
      "once":false,
      "onceDelay":0.1,
      "topic":"room1/fan/speed",
      "payload":"stop",
      "payloadType":"str",
      "x":160,
      "y":520,
      "wires":[
```

```
      [
         "f463b190bbeece7f"
      ]
   ]
},
{
   "id":"b9ed74b9542d631b",
   "type":"mqtt in",
   "z":"7c7dc628.08a158",
   "name":"",
   "topic":"room1/humidity",
   "qos":"2",
   "datatype":"auto",
   "broker":"7fa102c41ccc6e44",
   "nl":false,
   "rap":true,
   "rh":0,
   "x":340,
   "y":1140,
   "wires":[
      [
         "8f6e0940a3d419f5"
      ]
   ]
},
{
   "id":"8f6e0940a3d419f5",
   "type":"debug",
   "z":"7c7dc628.08a158",
   "name":"",
   "active":false,
   "tosidebar":true,
   "console":false,
   "tostatus":false,
   "complete":"true",
   "targetType":"full",
   "statusVal":"",
   "statusType":"auto",
   "x":590,
   "y":1140,
   "wires":[

   ]
```

```
},
{
    "id":"5fc47e13db25c729",
    "type":"function",
    "z":"7c7dc628.08a158",
    "name":"loop",
    "func":"function getRandomInt(min, max) {\n  return Math.floor(Math.random() * (max - min +1)) + min;\n}\n\
    "outputs":"2",
    "noerr":0,
    "initialize":"",
    "finalize":"",
    "libs":[

    ],
    "x":450,
    "y":920,
    "wires":[
        [
            "2591c34649c33f63",
            "1174133a1f85d1f9"
        ],
        [
            "930f7e3f3d92be80"
        ]
    ]
},
{
    "id":"930f7e3f3d92be80",
    "type":"delay",
    "z":"7c7dc628.08a158",
    "name":"",
    "pauseType":"delay",
    "timeout":"10",
    "timeoutUnits":"minutes",
    "rate":"1",
    "nbRateUnits":"",
    "rateUnits":"second",
    "randomFirst":"1",
    "randomLast":"5",
    "randomUnits":"seconds",
    "drop":false,
    "allowrate":false,
    "x":460,
```

```
    "y":1047,
    "wires":[
      [
        "5fc47e13db25c729"
      ]
    ]
  },
  {
    "id":"2591c34649c33f63",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":670,
    "y":840,
    "wires":[

    ]
  },
  {
    "id":"1b6028330f34a13e",
    "type":"scheduler",
    "z":"7c7dc628.08a158",
    "outtopic":"room1/humidity",
    "outpayload1":"start",
    "outpayload2":"stop",
    "name":"Humidity Sensor Scheduler",
    "lat":"48.210033",
    "lon":"16.363449",
    "start":"sunrise",
    "end":"sunset",
    "starttime":"480",
    "endtime":"1020",
    "duskoff":"0",
    "dawnoff":"0",
    "outtext1":"",
```

```
    "outtext2":"",
    "sun":false,
    "mon":true,
    "tue":true,
    "wed":true,
    "thu":true,
    "fri":true,
    "sat":false,
    "jan":true,
    "feb":true,
    "mar":true,
    "apr":true,
    "may":true,
    "jun":true,
    "jul":true,
    "aug":true,
    "sep":true,
    "oct":true,
    "nov":true,
    "dec":true,
    "repeat":false,
    "atstart":true,
    "x":170,
    "y":860,
    "wires":[
        [
            "5fc47e13db25c729"
        ],
        [

        ],
        [

        ]
    ]
},
{
    "id":"1174133a1f85d1f9",
    "type":"mqtt out",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/humidity",
    "qos":"",
```

```
    "retain":"",
    "respTopic":"",
    "contentType":"",
    "userProps":"",
    "correl":"",
    "expiry":"",
    "broker":"7fa102c41ccc6e44",
    "x":660,
    "y":920,
    "wires":[

    ]
},
{
    "id":"fa2d35f9a01bdf6e",
    "type":"inject",
    "z":"7c7dc628.08a158",
    "name":"",
    "props":[
        {
            "p":"payload"
        },
        {
            "p":"topic",
            "vt":"str"
        }
    ],
    "repeat":"",
    "crontab":"",
    "once":false,
    "onceDelay":0.1,
    "topic":"room1/humidity",
    "payload":"stop",
    "payloadType":"str",
    "x":150,
    "y":940,
    "wires":[
        [
            "5fc47e13db25c729"
        ]
    ]
},
{
```

```
    "id":"19182b090ec92a71",
    "type":"mqtt in",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/light_intensivity",
    "qos":"2",
    "datatype":"auto",
    "broker":"7fa102c41ccc6e44",
    "nl":false,
    "rap":true,
    "rh":0,
    "x":380,
    "y":1640,
    "wires":[
       [
          "8c6b068bdcadbe36"
       ]
    ]
 },
 {
    "id":"8c6b068bdcadbe36",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":610,
    "y":1640,
    "wires":[

    ]
 },
 {
    "id":"172e908f48d5b609",
    "type":"function",
    "z":"7c7dc628.08a158",
    "name":"loop",
```

```
    "func":"function getRandomInt(min, max) {\n  return Math.floor(Math.random() * (max - min +1)) + min;\n}\n\
    "outputs":"2",
    "noerr":0,
    "initialize":"",
    "finalize":"",
    "libs":[

    ],
    "x":470,
    "y":1420,
    "wires":[
        [
            "0658923d116a9550",
            "2b2869a38d11770d"
        ],
        [
            "9fb992afaf4a44c0"
        ]
    ]
},
{
    "id":"9fb992afaf4a44c0",
    "type":"delay",
    "z":"7c7dc628.08a158",
    "name":"",
    "pauseType":"delay",
    "timeout":"10",
    "timeoutUnits":"minutes",
    "rate":"1",
    "nbRateUnits":"",
    "rateUnits":"second",
    "randomFirst":"1",
    "randomLast":"5",
    "randomUnits":"seconds",
    "drop":false,
    "allowrate":false,
    "x":480,
    "y":1547,
    "wires":[
        [
            "172e908f48d5b609"
        ]
    ]
```

```
},
{
    "id":"0658923d116a9550",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":690,
    "y":1340,
    "wires":[

    ]
},
{
    "id":"b359b9df160c39ba",
    "type":"scheduler",
    "z":"7c7dc628.08a158",
    "outtopic":"room1/light_intensivity",
    "outpayload1":"start",
    "outpayload2":"stop",
    "name":"Light Sensor Scheduler",
    "lat":"48.210033",
    "lon":"16.363449",
    "start":"sunrise",
    "end":"sunset",
    "starttime":"480",
    "endtime":"1020",
    "duskoff":"0",
    "dawnoff":"0",
    "outtext1":"",
    "outtext2":"",
    "sun":false,
    "mon":true,
    "tue":true,
    "wed":true,
    "thu":true,
```

```
    "fri":true,
    "sat":false,
    "jan":true,
    "feb":true,
    "mar":true,
    "apr":true,
    "may":true,
    "jun":true,
    "jul":true,
    "aug":true,
    "sep":true,
    "oct":true,
    "nov":true,
    "dec":true,
    "repeat":false,
    "atstart":true,
    "x":180,
    "y":1360,
    "wires":[
        [
            "172e908f48d5b609"
        ],
        [

        ],
        [

        ]
    ]
},
{
    "id":"2b2869a38d11770d",
    "type":"mqtt out",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/light_intensivity",
    "qos":"",
    "retain":"",
    "respTopic":"",
    "contentType":"",
    "userProps":"",
    "correl":"",
    "expiry":"",
```

```
    "broker":"7fa102c41ccc6e44",
    "x":700,
    "y":1420,
    "wires":[

    ]
},
{
    "id":"316059ce22ab28d6",
    "type":"inject",
    "z":"7c7dc628.08a158",
    "name":"",
    "props":[
        {
            "p":"payload"
        },
        {
            "p":"topic",
            "vt":"str"
        }
    ],
    "repeat":"",
    "crontab":"",
    "once":false,
    "onceDelay":0.1,
    "topic":"room1/light_intensivity",
    "payload":"stop",
    "payloadType":"str",
    "x":200,
    "y":1440,
    "wires":[
        [
            "172e908f48d5b609"
        ]
    ]
},
{
    "id":"40b8167de1d29c75",
    "type":"mqtt in",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/sound_dba",
    "qos":"2",
```

```
    "datatype":"auto",
    "broker":"7fa102c41ccc6e44",
    "nl":false,
    "rap":true,
    "rh":0,
    "x":370,
    "y":2100,
    "wires":[
       [
          "1a006183ba0a43d3"
       ]
    ]
},
{
    "id":"1a006183ba0a43d3",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":610,
    "y":2100,
    "wires":[

    ]
},
{
    "id":"6ba79c180c5bbfbc",
    "type":"function",
    "z":"7c7dc628.08a158",
    "name":"loop",
    "func":"function getRandomInt(min, max) {\n  return Math.floor(Math.random() * (max - min +1)) + min;\n}\n\
    "outputs":"2",
    "noerr":0,
    "initialize":"",
    "finalize":"",
    "libs":[
```

```
    ],
    "x":470,
    "y":1880,
    "wires":[
        [
            "9c541f3edc09beae",
            "778be3321673ace5"
        ],
        [
            "06ccb4c99ac66121"
        ]
    ]
},
{
    "id":"06ccb4c99ac66121",
    "type":"delay",
    "z":"7c7dc628.08a158",
    "name":"",
    "pauseType":"delay",
    "timeout":"10",
    "timeoutUnits":"minutes",
    "rate":"1",
    "nbRateUnits":"",
    "rateUnits":"second",
    "randomFirst":"1",
    "randomLast":"5",
    "randomUnits":"seconds",
    "drop":false,
    "allowrate":false,
    "x":480,
    "y":2007,
    "wires":[
        [
            "6ba79c180c5bbfbc"
        ]
    ]
},
{
    "id":"9c541f3edc09beae",
    "type":"debug",
    "z":"7c7dc628.08a158",
    "name":"",
```

```
    "active":false,
    "tosidebar":true,
    "console":false,
    "tostatus":false,
    "complete":"true",
    "targetType":"full",
    "statusVal":"",
    "statusType":"auto",
    "x":690,
    "y":1800,
    "wires":[

    ]
},
{
    "id":"4b8143bd3863faeb",
    "type":"scheduler",
    "z":"7c7dc628.08a158",
    "outtopic":"room1/sound_dba",
    "outpayload1":"start",
    "outpayload2":"stop",
    "name":"Sound Sensor Scheduler",
    "lat":"48.210033",
    "lon":"16.363449",
    "start":"sunrise",
    "end":"sunset",
    "starttime":"480",
    "endtime":"1020",
    "duskoff":"0",
    "dawnoff":"0",
    "outtext1":"",
    "outtext2":"",
    "sun":false,
    "mon":true,
    "tue":true,
    "wed":true,
    "thu":true,
    "fri":true,
    "sat":false,
    "jan":true,
    "feb":true,
    "mar":true,
    "apr":true,
```

```
    "may":true,
    "jun":true,
    "jul":true,
    "aug":true,
    "sep":true,
    "oct":true,
    "nov":true,
    "dec":true,
    "repeat":false,
    "atstart":true,
    "x":190,
    "y":1820,
    "wires":[
        [
            "6ba79c180c5bbfbc"
        ],
        [

        ],
        [

        ]
    ]
},
{
    "id":"778be3321673ace5",
    "type":"mqtt out",
    "z":"7c7dc628.08a158",
    "name":"",
    "topic":"room1/sound_dba",
    "qos":"",
    "retain":"",
    "respTopic":"",
    "contentType":"",
    "userProps":"",
    "correl":"",
    "expiry":"",
    "broker":"7fa102c41ccc6e44",
    "x":690,
    "y":1880,
    "wires":[

    ]
```

```
},
{
    "id":"246f6431085af417",
    "type":"inject",
    "z":"7c7dc628.08a158",
    "name":"",
    "props":[
        {
            "p":"payload"
        },
        {
            "p":"topic",
            "vt":"str"
        }
    ],
    "repeat":"",
    "crontab":"",
    "once":false,
    "onceDelay":0.1,
    "topic":"room1/sound_dba",
    "payload":"stop",
    "payloadType":"str",
    "x":180,
    "y":1900,
    "wires":[
        [
            "6ba79c180c5bbfbc"
        ]
    ]
},
{
    "id":"7fa102c41ccc6e44",
    "type":"mqtt-broker",
    "name":"",
    "broker":"10.152.183.248",
    "port":"1883",
    "clientid":"",
    "usetls":false,
    "protocolVersion":"4",
    "keepalive":"60",
    "cleansession":true,
    "birthTopic":"",
    "birthQos":"0",
```

```
      "birthPayload":"",
      "birthMsg":{

      },
      "closeTopic":"",
      "closeQos":"0",
      "closePayload":"",
      "closeMsg":{

      },
      "willTopic":"",
      "willQos":"0",
      "willPayload":"",
      "willMsg":{

      },
      "sessionExpiry":""
   }
]
```

A.2 MQTTSA Application

A.2.1 mqttsa-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mqttsa1
  labels:
    app: mqttsa1
  namespace: slowdos-attacker
spec:
  selector:
    matchLabels:
      app: mqttsa1
  template:
    metadata:
      labels:
        app: mqttsa1
    spec:
      containers:
      - name: mqttsa1
        image: bigt1991/mqttsa:latest
```

```
        command: ["/bin/sleep", "3650d"]
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
```

A.2.2 dockerfile

```
# syntax=docker/dockerfile:1

#Download base image ubuntu 20.04
FROM ubuntu:20.04

# LABEL about the custom image
LABEL maintainer="thorsten.steuer@gmx.net"
LABEL version="0.1"
LABEL description="This is custom Docker Image to use https://github.com/stfbk/mqttsa"

# Disable Prompt During Packages Installation
ENV DEBIAN_FRONTEND=noninteractive

# Update Ubuntu Software repository
RUN apt-get update

RUN apt-get install -y git python3-pip tshark curl mosquitto-clients && \
    rm -rf /var/lib/apt/lists/* && \
    apt clean

RUN git clone https://github.com/stfbk/mqttsa.git
WORKDIR /mqttsa
RUN make
```

A.3 Mosquitto

A.3.1 mosquitto-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mosquitto-rooms-config
  namespace: rooms
data:
  mosquitto.conf: |-
    # Port to use for the default listener.
```

```
    port 1884

    # Allow anonymous users to connect?
    # If not, the password file should be created
    allow_anonymous true

    # Types of messages to log. Use multiple log_type lines for logging
    # multiple types of messages.
    # Possible types are: debug, error, warning, notice, information,
    # none, subscribe, unsubscribe, websockets, all.
    # Note that debug type messages are for decoding the incoming/outgoing
    # network packets. They are not logged in "topics".
    log_type error
    log_type warning
    log_type notice
    log_type information

    log_dest file /var/log/mosquitto/mosquitto.log
```

A.3.2 mosquitto-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mosquitto-rooms
  labels:
    app: mosquitto-rooms
  namespace: rooms
spec:
  selector:
    matchLabels:
      app: mosquitto-rooms
  template:
    metadata:
      labels:
        app: mosquitto-rooms
    spec:
      containers:
      - name: mosquitto-rooms
        image: eclipse-mosquitto:1.6.2
        resources:
          requests:
```

```
            cpu: "50m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 1884
        volumeMounts:
            - name: mosquitto-config
              mountPath: /mosquitto/config/mosquitto.conf
              subPath: mosquitto.conf
            - name: mosquitto-log
              mountPath: /var/log/mosquitto/
      volumes:
        - name: mosquitto-config
          configMap:
            name: mosquitto-rooms-config
        - name: mosquitto-log
          persistentVolumeClaim:
            claimName: mosquitto-rooms-log-pvc
```

A.3.3 mosquitto-pcv.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mosquitto-rooms-log-pvc
  namespace: rooms
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

A.3.4 mosquitto-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mosquitto-rooms
  namespace: rooms
```

```
spec:
  selector:
    app: mosquitto-rooms
  type: NodePort
  ports:
  - port: 1884
    targetPort: 1884
    protocol: TCP
    nodePort: 30884
```

B Code Base Intrusion Detection System

This appendix contains all code used to implement the IDS introduced in 5.

B.1 Detection System

B.1.1 detection-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: detection-app
  name: detection-app
  namespace: monitoring-system
spec:
  selector:
    matchLabels:
      app: detection-app
  template:
    metadata:
      labels:
        app: detection-app
    spec:
      containers:
      - name: detection-app
        image: bigt1991/idsslowdos:latest
        volumeMounts:
        - name: mosquitto-log-pv
          mountPath: /var/log/mosquitto/
        imagePullPolicy: IfNotPresent
        command: ["/bin/sleep", "3650d"]
      restartPolicy: Always
      volumes:
      - name: mosquitto-log-pv
        persistentVolumeClaim:
          claimName: mosquitto-log-pvc
```

B.1.2 alarm_notification.py

```
"""This module is a auxiliary class to write a email notification if an attack is detected.
    Adapted from:
```

```
        https://docs.python.org/3/library/email.examples.html
        [last accessed December 2021]

    Author: Thorsten Steuer
    Licence: Apache 2.0
"""
import smtplib
from email.message import EmailMessage
from beartype import beartype

EMAIL = 'ids_masterarbeit@gmx.at'
EMAIL_PASS ='cYqBUB5L'
EMAIL_SUBJECT = 'INTRUSION DETECTED!!!!'

class AlarmNotification:
    """ A class to send a Email Message once an intrusion was detected.
    """

    @beartype
    def stop_smtp(self):
        """Stop smtp client.
        """
        self.email_server.quit()

    @beartype
    def connect_to_smtp_server(self):
        """Connects to a smtp sever.
        """
        self.email_server = smtplib.SMTP(host='mail.gmx.net', port=587)
        self.email_server.starttls()
        self.email_server.login(EMAIL, EMAIL_PASS)

    @beartype
    def send_email(self, message: str):
        """Is Sending an Email with the provided message string.

        Args:
            message (str): [description]
        """
        msg = EmailMessage()
        msg.set_content(message)
        msg['Subject'] = EMAIL_SUBJECT
        msg['From'] = EMAIL
```

```
        msg['To'] = EMAIL

        self.email_server.send_message(msg)

if __name__ == '__main__':
    email_server = AlarmNotification()
    email_server.connect_to_smtp_server()
    email_server.send_email("TEST")
    email_server.stop_smtp()
```

B.1.3 format_log.py

```
"""This module analizes the log entries produced by mosquitto and converts it to json.
    Author: Thorsten Steuer
    Licence: Apache 2.0
"""

import re
import logging

from beartype import beartype
from local_file_access import LocalFileAccess


class FormatLog:
    """This class contains a collection of regular expression functions to extract data
    from the log file and stores it as dataframe.
    """

    def __init__(self):
        """Interface to extract information from the log file. I need to use string as booleans
        since json expext them to be wirtten small an neo4j can handel the conversion.
        """
        self.host_data = {"ip_address": None,
                          "creation_time": None,
                          "is_blocked": "False",
                          "notification_sent": "False"
                          }
        self.connection_data ={"status": "active",
                               "last_update_time": None,
                               "name": None
                               }
```

```
        self.service_data = {"name": None,
                             "port": None,
                             "version": None,
                             "protocol":None,
                             }
        self.vulnerability_data = {"name": None,
                                   "cve_code": None,
                                   "effected_protocol": None,
                                   "effected_prot_version":None,
                                   "effected_app": None,
                                   "effected_app_version":None,
                                   "description": None
                                   }
        self.precondition_data = {"name": None,
                                   "type": None,
                                   "capability_level": None,
                                   "description": None
                                   }
        self.attack_data = {"name": None,
                             "type": None,
                             "goal": None
                             }

    @beartype
    def extract_ip_address_by_row(self, log_string: str):
        """Extract with a regular exprsession the ip address form a string

        Args:
            log_string (str): one row of the log file
        """
        matches = re.findall(r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b", log_string)
        if matches:
            if len(matches) == 1:
                self.host_data["ip_address"] = matches[0]
            else:
                logging.error("Functions: extract_ip_address_by_row else clause not yet implemeneted")
        else:
            logging.info("extract_ip_address_by_row no match was found in: %s", log_string)

    @beartype
    def extract_port_number_by_row(self, log_string: str):
        """Extract with a regular exprsession the port number form a string. Needed to
        dismiss the last two character which is the newline operator and the dot to
```

```
    properly work. In the mosquitto log the port number is everytime at the end of
    the line.
    e.g. 1635015162: New connection from 127.0.0.1 on port 1883.
    There might be a better solution for this.

    Args:
        log_string (str): one row of the log file
    """
    log_string = log_string[:-2]
    matches = re.findall(r"\b([0-9]{1,4}|[1-5][0-9]{4}|6[0-4][0-9]{3}|65[0-4][0-9]{2}|655[0-2][0-9]|6553[0-5]
    if matches:
        if len(matches) == 1:
            self.service_data["port"] = matches[0]
        else:
            logging.error("Functions: extract_port_number_by_row else clause not yet implemeneted")
    else:
        logging.info("extract_port_number_by_row no match was found in: %s", log_string)

@beartype
def extract_time_in_seconds_by_row(self, log_string: str):
    """Extract  the times in seconds form a string by splitting it at the character ':'.
    Putting the timestamp each time also in the connection data start_time to tempoarily store
    in the object and check with the database.

    Args:
        log_string (str): one row of the log file
    """
    groups = log_string.split(":")
    if groups:
        self.host_data["creation_time"] = groups[0]
        self.connection_data["last_update_time"] = groups[0]

    else:
        logging.info("Could not split group missing ':' in: %s", log_string)

@beartype
def extract_connection_name_by_row(self, log_string: str):
    """Extract with a regular exprsession the name of the connections.

    Args:
        log_string (str): one row of the log file
    """
    matches = re.findall(r"(\w{1,}[-|_].[^ (]{1,})", log_string)
```

```
        if matches:
            if len(matches) == 1:
                self.connection_data["name"] = matches[0]
            else:
                logging.error("Functions: extract_connection_name_by_row else clause not yet implemeneted")
        else:
            logging.info("extract_connection_name_by_row no match was found in: %s", log_string)

    @beartype
    def extract_version_by_row(self, log_string: str):
        """Extract with a regular exprsession the version of the service.

        Args:
            log_string (str): one row of the log file
        """
        matches = re.findall(r"version \b([0-9]{1,2}\.[0-9]{1,2}\.[0-9]{1,2})\b", log_string)
        if matches:
            if len(matches) == 1:
                self.service_data["version"] = matches[0]
            else:
                logging.error("Functions: extract_connection_name_by_row else clause not yet implemeneted")
        else:
            logging.info("extract_version_by_row no match was found in: %s", log_string)

    @beartype
    def set_service_port(self, port: str):
        """Set port of service

        Args:
            port (str): port of the service
        """
        self.service_data["port"] = port

    @beartype
    def set_service_protocol(self, protocol: str):
        """Set communication protocol of service

        Args:
            protocpl (str): protocol of the service
        """
        self.service_data["protocol"] = protocol

    @beartype
```

```
def set_service_version(self, version: str):
    """Set version of service

    Args:
        protocol (str): protocol of the service
    """
    self.service_data["version"] = version

@beartype
def set_service_name(self, name: str):
    """Set name of service

    Args:
        name (str): name of the service
    """
    self.service_data["name"] = name

def reset_service_data(self):
    """Set values of service_data json to None
    """
    self.service_data = dict.fromkeys(self.service_data, None)

def reset_host_data(self):
    """Set values of host_data json to None.
    ATTENTION!!! Only set creation_time to None ohter values are default values and the
    ip address is neede late in the code.
    """
    self.host_data["creation_time"] = None

@beartype
def set_host_is_blocked(self, is_blocked: str):
    """Set values of is_blocked in host json to false or ture.
    Is a string since neo4j can't convert python bool to json bool.

    Args:
        is_blocked (str): True or False
    """
    self.host_data["is_blocked"] = is_blocked

@beartype
def set_host_notification_sent(self, sent: str):
    """Set values of is_blocked in host json to false or ture.
    Is a string since neo4j can't convert python bool to json bool.
```

```
    Args:
        sent (str): True or False
    """
    self.host_data["notification_sent"] = sent

@beartype
def set_host_ip_address(self, ip_address: str):
    """Set values of ip_address in host json.

    Args:
        ip_address (str): ip_address of the host
    """
    self.host_data["ip_address"] = ip_address

@beartype
def set_host_creation_time(self, time: str):
    """Set values of creation_time in host json.

    Args:
        time (str): timestamp in second (UTC)
    """
    self.host_data["creation_time"] = time

def reset_connection_data(self):
    """Set values of vulberability_data json to None
    """
    self.connection_data = dict.fromkeys(self.connection_data, None)

@beartype
def set_connection_status(self, status: str):
    """Set status of connection

    Args:
        status (str): status code of connection (inactive/active)
    """
    self.connection_data["status"] = status

@beartype
def set_connection_name(self, name: str):
    """Set name of connection

    Args:
        name (str): name of the connection
```

```
    """
    self.connection_data["name"] = name

@beartype
def set_connection_last_update_time(self, time: str):
    """Set last update time of the connection

    Args:
        time (str): time the connection was last updated
    """
    self.connection_data["last_update_time"] = time

def reset_vulnerability_data(self):
    """Set values of connection_data json to None
    """
    self.vulnerability_data = dict.fromkeys(self.vulnerability_data, None)

@beartype
def set_vulnerability_name(self, name: str):
    """Set name of vulnerablitly.
    Args:
        name (str): name of the vulnerability
    """
    self.vulnerability_data["name"] = name

@beartype
def set_vulnerability_cve_code(self, cve_code: str):
    """Set name of vulnerablitly.
    Args:
        name (str): name of the vulnerability
    """
    self.vulnerability_data["cve_code"] = cve_code

@beartype
def set_vulnerability_effected_protocol(self, effected_protocol: str):
    """Set effected protocol name of vulnerablitly.
    Args:
        effected_protocol (str): name of the effected protocol
    """
    self.vulnerability_data["effected_protocol"] = effected_protocol

@beartype
def set_vulnerability_effected_app(self, effected_app: str):
```

```
    """Set effected app of vulnerablitly.
    Args:
        effected_app (str): name of the effected app
    """
    self.vulnerability_data["effected_app"] = effected_app

@beartype
def set_vulnerability_effected_app_version(self, effected_app_version: str):
    """Set effected app of vulnerablitly.
    Args:
        effected_app_version (str): name of the effected app version
    """
    self.vulnerability_data["effected_app_version"] = effected_app_version

@beartype
def set_vulnerability_effected_prot_version(self, effected_prot_version: str):
    """Set effected protocol version of vulnerablitly.
    Args:
        effected_prot_version (str): version of the effected protocol
    """
    self.vulnerability_data["effected_prot_version"] = effected_prot_version

@beartype
def set_vulnerability_description(self, description: str):
    """Set vulnerablitly description.
    Args:
        description (str): description of the vulnerability
    """
    self.vulnerability_data["description"] = description

def reset_precondition_data(self):
    """Set values of precondition_data json to None
    """
    self.precondition_data = dict.fromkeys(self.precondition_data, None)

@beartype
def set_precondition_name(self, name: str):
    """Set precondition name needed for an attack.
    Args:
        description (str): name of the precondition
    """
    self.precondition_data["name"] = name
```

```python
@beartype
def set_precondition_type(self, cond_type: str):
    """Set precondition type needed for an attack.
    Args:
        cond_type (str): type of the precondition (Network Access/User Credentials)
    """
    self.precondition_data["type"] = cond_type

@beartype
def set_precondition_description(self, description: str):
    """Set precondition description needed for an attack.
    Args:
        description (str): description of the precondition
    """
    self.precondition_data["description"] = description

@beartype
def set_precondition_capability_level(self, capability_level: str):
    """Set precondition capability_level needed for an attack.
    Args:
        capability_level (str): capability_level of the precondition
    """
    self.precondition_data["capability_level"] = capability_level

def reset_attack_data(self):
    """Set values of precondition_data json to None
    """
    self.precondition_data = dict.fromkeys(self.precondition_data, None)

@beartype
def set_attack_name(self, name: str):
    """Set attack name.
    Args:
        name (str): name of the attack
    """
    self.attack_data["name"] = name

@beartype
def set_attack_type(self, attack_type: str):
    """Set attack type.
    Args:
        attack_type (str): name of the attack type (DoS)
    """
```

```
        self.attack_data["type"] = attack_type

    @beartype
    def set_attack_goal(self, goal: str):
        """Set attack goal.
        Args:
            goal (str): name of the attack goal
        """
        self.attack_data["goal"] = goal

if __name__ == "__main__":
    LOCAL_PATH = "C:/Users/Thorsten/Documents/Masterarbeit/Security/ids_slow_dos/detection_system/mosquitto_log_d
    FILE_NAME = "mosquitto_test_1.log"
    local_file = LocalFileAccess(LOCAL_PATH, FILE_NAME)
    log_formatter = FormatLog()
    log_formatter.extract_ip_address_by_row(local_file.get_line(4))
    log_formatter.extract_port_number_by_row(local_file.get_line(4))
    log_formatter.extract_time_in_seconds_by_row(local_file.get_line(4))
    log_formatter.set_connection_status("active")
    log_formatter.extract_connection_name_by_row(local_file.get_line(5))
    log_formatter.extract_port_number_by_row(local_file.get_line(2))
    log_formatter.extract_version_by_row(local_file.get_line(0))
    print(log_formatter.host_data)
    print(log_formatter.connection_data)
    print(log_formatter.service_data)
    local_file.close()
```

B.1.4 initialize_system.py

```
"""This module send a email notification if an attack is detected.

    Author: Thorsten Steuer
    Licence: Apache 2.0
"""
import json
import time
import random
import uuid


from beartype import beartype
from kubernetes import client, config
```

```
from neo4j_database_access import Neo4jDatabaseAccess
from format_log import FormatLog

NEO4J_URI = 'bolt://localhost:30687'
NEO4J_USER = 'neo4j'
NEO4J_PASS = '1234'

class InitializeSystem:
    """ A class to set up the inital database set up with host, vulnearbilities etc.
    """

    def __init__(self):
        """Call initilization with database connection
        """
        self.log_formatter = FormatLog()
        self.neo4j_driver = Neo4jDatabaseAccess(NEO4J_URI, NEO4J_USER, NEO4J_PASS)

    def demo_setup(self):
        """Set up the database for a demo use case. ATTENTION DELETE ALL ENTRIES!!
        """
        query = ('MATCH (n) DETACH DELETE n')
        self.neo4j_driver.execute_and_return_query_result(query)

        self.create_topology_set_up()
        self.create_vulnerabilities()
        self.create_attack_and_preconditions()

    def perfromance_test_setup(self):
        """Set up the database for a demo use case. ATTENTION DELETE ALL ENTRIES!!
        """
        query = ('MATCH (n) DETACH DELETE n')
        self.neo4j_driver.execute_and_return_query_result(query)

        self.create_random_host(100)
        self.create_random_service(100)
        self.connect_each_service_with_a_host()
        self.connects_n_hosts_with_random_service(45,1)
        self.connects_n_hosts_with_random_service(1,55)

    def create_topology_set_up(self):
        """Creates the host and service topology of the demo use case.
        """
```

```
# Create host machine for Neo4j
ip_add_pod = self.get_ip_address_pod("neo4j-db")
self.log_formatter.set_host_ip_address(ip_add_pod)
self.log_formatter.set_host_creation_time(str(int(time.time())))

self.neo4j_driver.create_host(json.dumps(self.log_formatter.host_data))

# Create Service instance for Neo4j
self.log_formatter.set_service_name("Neo4j")
self.log_formatter.set_service_protocol("Bolt")
self.log_formatter.set_service_version("4.3.6-enterprise")
self.log_formatter.set_service_port("7687")

self.neo4j_driver.create_service(json.dumps(self.log_formatter.service_data))

self.log_formatter.reset_service_data()

#Create a at_host relation for the neo4j service

at_host_relation = ('{'
                    '"edge_name": "HOSTED_AT",'
                    '"node1": {  "name":"Service",'
                                '"property_names": ["name"],'
                                '"property_values": ["Neo4j"]},'
                    '"node2": {  "name": "Host",'
                                '"property_names": ["ip_address"],'
                                '"property_values": ["' +  ip_add_pod  + '"]}'
                    '}'
                )

self.neo4j_driver.create_edge(at_host_relation)

# Create host machine for detection system
ip_add_pod = self.get_ip_address_pod("detection-app")
self.log_formatter.set_host_ip_address(ip_add_pod)
self.log_formatter.set_host_creation_time(str(int(time.time())))

self.neo4j_driver.create_host(json.dumps(self.log_formatter.host_data))

# Create Service instance for detection system
self.log_formatter.set_service_name("Intrusion Detection System")
self.log_formatter.set_service_protocol("-")
self.log_formatter.set_service_version("0.0.1")
```

```
self.log_formatter.set_service_port("-")

self.neo4j_driver.create_service(json.dumps(self.log_formatter.service_data))
self.log_formatter.reset_service_data()

#Create a at_host relation for the detection service
at_host_relation = ('{'
                    '"edge_name": "HOSTED_AT",'
                    '"node1": {  "name":"Service",'
                                '"property_names": ["name"],'
                                '"property_values": ["Intrusion Detection System"]},'
                    '"node2": {  "name": "Host",'
                                '"property_names": ["ip_address"],'
                                '"property_values": ["' +  ip_add_pod  + '"]}'
                    '}'
                )

self.neo4j_driver.create_edge(at_host_relation)

# Create a connection to neo4j database
self.log_formatter.set_connection_name("detection-app-0a61e6f1")
self.log_formatter.set_connection_last_update_time(str(int(time.time())))
self.neo4j_driver.create_connection(json.dumps(self.log_formatter.connection_data))

starts_connection_data = ('{'
                          '"edge_name": "STARTS_CONNECTION",'
                          '"node1": {  "name": "Host",'
                                      '"property_names": ["ip_address"],'
                                      '"property_values": ["' +  ip_add_pod  + '"]},'
                          '"node2": {  "name":"Connection",'
                                      '"property_names": ["name"],'
                                      '"property_values": ["' +  self.log_formatter.connection_data["na
                          '}'
                        )

self.neo4j_driver.create_edge(starts_connection_data)

connects_to_data = ('{'
                    '"edge_name": "CONNECTS_TO",'
                    '"node1": {"name":"Connection",'
                           '"property_names": ["name"],'
                           '"property_values": ["' +  self.log_formatter.connection_data["name"]  + '"]}
                    '"node2": {  "name":"Service",'
```

```
                            '"property_names": ["name", "port", "version"],'
                            '"property_values": ["Neo4j",'
                                                '"7687",'
                                                '"4.3.6-enterprise"]}'
                    '}'
                    )
self.neo4j_driver.create_edge(connects_to_data)
self.log_formatter.reset_connection_data()

# Create host machine for mosquitto
ip_add_pod = self.get_ip_address_pod("mosquitto-broker")
self.log_formatter.set_host_ip_address(ip_add_pod)
self.log_formatter.set_host_creation_time(str(int(time.time())))

self.neo4j_driver.create_host(json.dumps(self.log_formatter.host_data))

# Create Service instance for Mosquitto
self.log_formatter.set_service_name("Mosquitto")
self.log_formatter.set_service_protocol("MQTT")
self.log_formatter.set_service_version("1.6.9")
self.log_formatter.set_service_port("1883")

self.neo4j_driver.create_service(json.dumps(self.log_formatter.service_data))
self.log_formatter.reset_service_data()


#Create a at_host relation for the mosquitto service

at_host_relation = ('{'
                    '"edge_name": "HOSTED_AT",'
                    '"node1": {  "name":"Service",'
                                '"property_names": ["name"],'
                                '"property_values": ["Mosquitto"]},'
                    '"node2": {  "name": "Host",'
                                '"property_names": ["ip_address"],'
                                '"property_values": ["' +  ip_add_pod  + '"]}'
                    '}'
                )

self.neo4j_driver.create_edge(at_host_relation)


# Create host machine for one mqttsa
```

```
        ip_add_pod = self.get_ip_address_pod("mqttsa1")
        self.log_formatter.set_host_ip_address(ip_add_pod)
        self.log_formatter.set_host_creation_time(str(int(time.time())))

        self.neo4j_driver.create_host(json.dumps(self.log_formatter.host_data))

        # Create Fake Service Apache Web Service
        self.log_formatter.set_service_name("Apache Web Server")
        self.log_formatter.set_service_protocol("HTTP")
        self.log_formatter.set_service_version("2.4.52")
        self.log_formatter.set_service_port("80")

        self.neo4j_driver.create_service(json.dumps(self.log_formatter.service_data))
        self.log_formatter.reset_service_data()

        #Create a at_host relation for the apache web server service
        at_host_relation = ('{'
                            '"edge_name": "HOSTED_AT",'
                            '"node1": {  "name":"Service",'
                                        '"property_names": ["name"],'
                                        '"property_values": ["Apache Web Server"]},'
                            '"node2": {  "name": "Host",'
                                        '"property_names": ["ip_address"],'
                                        '"property_values": ["' +  ip_add_pod  + '"]}'
                            '}'
                        )

        self.neo4j_driver.create_edge(at_host_relation)

    def create_vulnerabilities(self):
        """Create the vulnerabilites for the demo use case.
        """
        # Create vulnerability
        self.log_formatter.set_vulnerability_name("SlowITE")
        self.log_formatter.set_vulnerability_cve_code("CVE-2020-13849")
        self.log_formatter.set_vulnerability_effected_protocol("MQTT")
        self.log_formatter.set_vulnerability_effected_prot_version("3.1.1")
        self.log_formatter.set_vulnerability_effected_app("Mosquitto")
        self.log_formatter.set_vulnerability_effected_app_version("2.0.11 downwards")
        self.log_formatter.set_vulnerability_description("The MQTT protocol 3.1.1 requires a server to set a time

        self.neo4j_driver.create_vulnerability(json.dumps(self.log_formatter.vulnerability_data))
        self.log_formatter.reset_vulnerability_data()
```

```
#Create a has_vulnerability relation for the mosquitto service
has_vulnerability_relation = ('{'
                    '"edge_name": "HAS_VULNERABILITY",'
                    '"node1": {  "name": "Service",'
                                '"property_names": ["name"],'
                                '"property_values": ["Mosquitto"]},'
                    '"node2": {  "name":"Vulnerability",'
                                '"property_names": ["cve_code"],'
                                '"property_values": ["CVE-2020-13849"]}'
                    '}'
                )

self.neo4j_driver.create_edge(has_vulnerability_relation)

#Create a jeopardizes relation for the mosquitto service
jeopardizes_relation = ('{'
                    '"edge_name": "JEOPARDIZES",'
                    '"node1": {  "name":"Vulnerability",'
                                '"property_names": ["cve_code"],'
                                '"property_values": ["CVE-2020-13849"]},'
                    '"node2": {  "name": "Service",'
                                '"property_names": ["name"],'
                                '"property_values": ["Mosquitto"]}'
                    '}'
                )

self.neo4j_driver.create_edge(jeopardizes_relation)

# Create vulnerability
self.log_formatter.set_vulnerability_name("Log4j")
self.log_formatter.set_vulnerability_cve_code("CVE-2021-44228")
self.log_formatter.set_vulnerability_effected_protocol("-")
self.log_formatter.set_vulnerability_effected_prot_version("-")
self.log_formatter.set_vulnerability_effected_app("Apache Log4j")
self.log_formatter.set_vulnerability_effected_app_version("2.15.0")
self.log_formatter.set_vulnerability_description("An attacker who can control log messages or log message

self.neo4j_driver.create_vulnerability(json.dumps(self.log_formatter.vulnerability_data))
self.log_formatter.reset_vulnerability_data()

#Create a has_vulnerability relation for the mosquitto service
has_vulnerability_relation = ('{'
```

```
                        '"edge_name": "HAS_VULNERABILITY",'
                        '"node1": {  "name": "Service",'
                                    '"property_names": ["name"],'
                                    '"property_values": ["Apache Web Server"]},'
                        '"node2": {  "name":"Vulnerability",'
                                    '"property_names": ["cve_code"],'
                                    '"property_values": ["CVE-2021-44228"]}'
                        '}'
                    )

        self.neo4j_driver.create_edge(has_vulnerability_relation)

        #Create a jeopardizes relation for the mosquitto service
        jeopardizes_relation = ('{'
                        '"edge_name": "JEOPARDIZES",'
                        '"node1": {  "name":"Vulnerability",'
                                    '"property_names": ["cve_code"],'
                                    '"property_values": ["CVE-2021-44228"]},'
                        '"node2": {  "name": "Service",'
                                    '"property_names": ["name"],'
                                    '"property_values": ["Apache Web Server"]}'
                        '}'
                    )

        self.neo4j_driver.create_edge(jeopardizes_relation)


    def create_attack_and_preconditions(self):
        """Create the attacks and precondition for the demo use case
        """
        # Create attack
        self.log_formatter.set_attack_name("DoS-0a61e6f1")
        self.log_formatter.set_attack_type("SlowDoS")
        self.log_formatter.set_attack_goal("Loss of the ability to establish new connections")

        self.neo4j_driver.create_attack(json.dumps(self.log_formatter.attack_data))
        self.log_formatter.reset_attack_data()

        # Create attack
        self.log_formatter.set_attack_name("ACE-8b79e45f")
        self.log_formatter.set_attack_type("Remote code execution")
        self.log_formatter.set_attack_goal("Arbitrary code execution (ACE) is the ability of an attacker to run a
```

```
self.neo4j_driver.create_attack(json.dumps(self.log_formatter.attack_data))
self.log_formatter.reset_attack_data()

# Create precondition Network Access
self.log_formatter.set_precondition_name("Network Access-f78b36b5")
self.log_formatter.set_precondition_type("Reachability")
self.log_formatter.set_precondition_capability_level("Moderate")
self.log_formatter.set_precondition_description("Adversary which gain access to the network are capable t

self.neo4j_driver.create_precondition(json.dumps(self.log_formatter.precondition_data))
self.log_formatter.reset_precondition_data()

#Create a satisfied by relation for the log4j vulnerability
satisfied_by_relation = ('{'
                    '"edge_name": "SATISFIED_BY",'
                    '"node1": {  "name": "Precondition",'
                                '"property_names": ["name"],'
                                '"property_values": ["Network Access-f78b36b5"]},'
                    '"node2": {  "name":"Vulnerability",'
                                '"property_names": ["cve_code"],'
                                '"property_values": ["CVE-2021-44228"]}'
                    '}'
                )

self.neo4j_driver.create_edge(satisfied_by_relation)

#Create a requires relation Network Access to ACE
requires_relation = ('{'
                    '"edge_name": "REQUIRES",'
                    '"node1": {  "name": "Attack",'
                                '"property_names": ["name"],'
                                '"property_values": ["ACE-8b79e45f"]},'
                    '"node2": {  "name":"Precondition",'
                                '"property_names": ["name"],'
                                '"property_values": ["Network Access-f78b36b5"]}'
                    '}'
                )

self.neo4j_driver.create_edge(requires_relation)

#Create a requires relation Network Access to DoS
requires_relation = ('{'
                    '"edge_name": "REQUIRES",'
```

```
                    '"node1": {  "name": "Attack",'
                                '"property_names": ["name"],'
                                '"property_values": ["DoS-0a61e6f1"]},'
                    '"node2": {  "name":"Precondition",'
                                '"property_names": ["name"],'
                                '"property_values": ["Network Access-f78b36b5"]}'
                    '}'
                )

self.neo4j_driver.create_edge(requires_relation)

#Create precondition MQTT
self.log_formatter.set_precondition_name("MQTT 3.1.1-a8h456u7")
self.log_formatter.set_precondition_type("Service/Protocol Status")
self.log_formatter.set_precondition_capability_level("Low")
self.log_formatter.set_precondition_description("Advesaries nee first to gain access to the network befor

self.neo4j_driver.create_precondition(json.dumps(self.log_formatter.precondition_data))
self.log_formatter.reset_precondition_data()

#Create a satisfied by relation for the log4j vulnerability
satisfied_by_relation = ('{'
                    '"edge_name": "SATISFIED_BY",'
                    '"node1": {  "name": "Precondition",'
                                '"property_names": ["name"],'
                                '"property_values": ["MQTT 3.1.1-a8h456u7"]},'
                    '"node2": {  "name":"Vulnerability",'
                                '"property_names": ["cve_code"],'
                                '"property_values": ["CVE-2020-13849"]}'
                    '}'
                )
self.neo4j_driver.create_edge(satisfied_by_relation)

#Create a requires relation Network Access to DoS
requires_relation = ('{'
                    '"edge_name": "REQUIRES",'
                    '"node1": {  "name": "Attack",'
                                '"property_names": ["name"],'
                                '"property_values": ["DoS-0a61e6f1"]},'
                    '"node2": {  "name":"Precondition",'
                                '"property_names": ["name"],'
                                '"property_values": ["MQTT 3.1.1-a8h456u7"]}'
                    '}'
```

```
                    )

        self.neo4j_driver.create_edge(requires_relation)

    @beartype
    def create_random_host(self, n: int):
        """ creates n radom hosts.

        Args:
            n (int): number of host that should be created
        """
        i = 0
        for i in range(0, n):

            ip_add_pod = "196." + str(random.randint(0,10)) + "." + str(random.randint(0,244)) + "." + str(random
            self.log_formatter.set_host_ip_address(ip_add_pod)
            self.log_formatter.set_host_creation_time(str(int(time.time())))

            if not self.neo4j_driver.check_if_node_exists('Host',
                                                          'ip_address',
                                                          self.log_formatter.host_data['ip_address']):
                self.neo4j_driver.create_host(json.dumps(self.log_formatter.host_data))
                i = i + 1

            self.log_formatter.reset_host_data()

    @beartype
    def create_random_service(self, n: int):
        """Creates n random services.

        Args:
            n (int): number of services to be created
        """
        i = 0
        for i in range(0, n):
            service_list = ["Mosquitto", "Neo4j", "SSH", "Firewall", "Docker", "Kubernetes", "Apache Web Server"]
            service_name = random.choice(service_list)+ "-" + str(uuid.uuid4())
            self.log_formatter.set_service_name(service_name)
            self.log_formatter.set_service_protocol("404-Not Available")
            service_version = str(random.randint(0,30)) + "." +  str(random.randint(0,30)) + "." +  str(random.ra
            self.log_formatter.set_service_version(service_version)
            port = str(random.randint(1000,3000))
            self.log_formatter.set_service_port(port)
```

```
            if not self.neo4j_driver.check_if_node_exists('Service',
                                                          'name',
                                                          self.log_formatter.service_data['name']):
                self.neo4j_driver.create_service(json.dumps(self.log_formatter.service_data))
                i = i + 1

            self.log_formatter.reset_service_data()

    def connect_each_service_with_a_host(self):
        """Connect each host with a service bia a HOSTED_AT relation

        Args:
            n (int): [description]
        """
        query = ('MATCH (h:Host) '
                 'WHERE NOT (h)-[:HOSTED_AT]-(:Service) '
                 'RETURN h.ip_address')
        host_result = self.neo4j_driver.execute_and_return_query_result(query)
        if host_result is not None:
            for host_row in host_result:
                query = ('MATCH (s:Service) '
                         'WHERE NOT (s)-[:HOSTED_AT]-(:Host) '
                         'return s.name')
                service_result = self.neo4j_driver.execute_and_return_query_result(query)
                service_name = random.choice(service_result)
                at_host_relation = ('{'
                         '"edge_name": "HOSTED_AT",'
                         '"node1": {  "name":"Service",'
                                    '"property_names": ["name"],'
                                    '"property_values": ["' +  service_name['s.name'] + '"]},'
                         '"node2": {  "name": "Host",'
                                    '"property_names": ["ip_address"],'
                                    '"property_values": ["' +  host_row['h.ip_address']  + '"]}'
                         '}'
                    )

                self.neo4j_driver.create_edge(at_host_relation)

    def connects_n_hosts_with_random_service(self, n_different_hosts: int, amount_of_connections: int):
        """Connect each host with a service via a connection node

        Args:
```

```
    n_different_hosts (int): number of host which should be randomly connected
    amount_of_connections (int): number of connection which should be established for this host
"""
i = 0
for i in range(0, n_different_hosts):
    query = ('MATCH (h:Host) '
             'RETURN h.ip_address')
    host_result = self.neo4j_driver.execute_and_return_query_result(query)
    host_ip_address = random.choice(host_result)

    query = ('MATCH (s:Service) '
             'return s.name')
    service_result = self.neo4j_driver.execute_and_return_query_result(query)
    service_name = random.choice(service_result)
    i = i + 1
    j = 0
    for j in range(0, amount_of_connections):
        connection_name = service_name['s.name'] + "-" + str(uuid.uuid4())
        self.log_formatter.set_connection_name(connection_name)
        self.log_formatter.set_connection_status("active")
        self.log_formatter.set_connection_last_update_time(str(int(time.time())))
        self.neo4j_driver.create_connection(json.dumps(self.log_formatter.connection_data))
        self.log_formatter.reset_connection_data()

        starts_connection_data = ('{'
                        '"edge_name": "STARTS_CONNECTION",'
                        '"node1": {  "name": "Host",'
                                    '"property_names": ["ip_address"],'
                                    '"property_values": ["' +  host_ip_address["h.ip_address"]  + '"]},'
                        '"node2": {  "name":"Connection",'
                                    '"property_names": ["name"],'
                                    '"property_values": ["' +  connection_name  + '"]}'
                        '}'
                    )

        self.neo4j_driver.create_edge(starts_connection_data)

        connects_to_data = ('{'
                        '"edge_name": "CONNECTS_TO",'
                        '"node1": {"name":"Connection",'
                                '"property_names": ["name"],'
                                '"property_values": ["' +  connection_name  + '"]},'
                        '"node2": {  "name":"Service",'
```

```
                                        '"property_names": ["name"],'
                                        '"property_values": ["' + service_name['s.name'] + '"]}'
                                    '}'
                                    )
                    self.neo4j_driver.create_edge(connects_to_data)
                    j = j + 1

    @staticmethod
    @beartype
    def get_ip_address_pod(app_selector: str):
        """Gets ip address of the pod in the k8s cluster

        Args:
            app_selector (str): the app name of the pod the ip address need to be retrieved

        Returns:
            ip_address (str): ip address of the pod
        """
        # it works only if this script is run by K8s as a POD
        #config.load_incluster_config()
        config.load_kube_config()

        v_1 = client.CoreV1Api()
        ret = v_1.list_pod_for_all_namespaces(label_selector='app='+ app_selector)
        for i in ret.items:
            ip_address=i.status.pod_ip

        return ip_address

if __name__ == '__main__':
    system = InitializeSystem()
    system.demo_setup_initialization()
    #system.perfromance_test_setup()
```

B.1.5 local_file_access.py

```
"""This module handles access to local files.

    Adapted from:
    https://medium.com/@aliasav/how-follow-a-file-in-python-tail-f-in-python-bca026a901cf
    [last accessed December 2021]
```

```
    Author: Thorsten Steuer
    Licence: Apache 2.0
"""

import os
import time

from beartype import beartype

class LocalFileAccess:
    """This class defines how local files can be accessed.
    """
    @beartype
    def __init__(self, local_local_file_path: str, local_file_name: str):
        """Initializes the class with a open file object. Create a list for each line and
        reset the fiel to read from the beginning.
        Args:
            local_local_file_path (str): local path to your file
            local_file_name (str): name of the file to be opened
        """
        self.local_file_path = local_local_file_path + "/" + local_file_name
        self.local_file = open(self.local_file_path, "r", encoding="utf-8")
        self.list_of_lines = self.local_file.readlines()
        self.local_file.seek(0)

    def close(self):
        """Closes the file when called
        """
        self.local_file.close()

    @beartype
    def print_n_lines(self, n_times: int):
        """Prints n lines of the file.

        Args:
            n_times (int): number of lines to print
        """
        line_counter = 0
        for lines in self.list_of_lines:
            if line_counter <= n_times:
                print("Line " + str(line_counter) + ": " + lines, end = "")
                line_counter += 1
            else:
```

```
            break

    @beartype
    def get_line(self, line_to_read: int) -> str:
        """Retruns a specific line based on the provided parameter.

        Args:
            line_to_read (int): speficies the line to be returned

        Returns:
            str: returns the requested line
        """
        return self.list_of_lines[line_to_read]

    @beartype
    def get_number_of_rows_of_file(self) -> int:
        """Retruns the amount of lines in the log file.

        Returns:
            int: number of lines
        """
        return len(self.list_of_lines)

    def tail_file(self, ):
        """Follows a file like the unix comand tail. Also works if logrotate is enabled.

        Yields:
            [generator object]: returns a generator function to iterate over log lines.
        """
        seek_end = True
        while True:
            with self.local_file as file:
                if seek_end:
                    file.seek(0, 2)
                while True:
                    line = file.readline()
                    if not line:
                        try:
                            if file.tell() > os.path.getsize(self.local_file_path):
                                file.close()
                                seek_end = False
                                break
                        except FileNotFoundError:
```

```
                    pass
                time.sleep(1)
            yield line


if __name__ == "__main__":
    LOCAL_PATH = "C:/Users/Thorsten/Documents/Masterarbeit/Security/ids_slow_dos/detection_system/code"
    FILE_NAME = "mosquitto.log"
    local_file = LocalFileAccess(LOCAL_PATH, FILE_NAME)
    local_file.print_n_lines(20)
    print(local_file.get_number_of_rows_of_file())
    loglines =  local_file.tail_file()
    counter = 0
    for log_line in loglines:
        print(str(counter) + ": " + log_line)
        counter += 1
        if counter == 10:
            break
    local_file.close()
```

B.1.6 mosquitto_log_transformation.py

```
"""Module to handel the mosquitto log to import it to neo4j
    Author: Thorsten Steuer
    Licence: Apache 2.0
"""
import json

# Was used for performance testing
# import time
# from statistics import median, mean, stdev


from local_file_access import LocalFileAccess
from format_log import FormatLog
from neo4j_database_access import Neo4jDatabaseAccess
from network_monitoring import NetworkMonitoring
from initialize_system import InitializeSystem

LOCAL_PATH = 'C:/kind_persistent_volume/pvc-0dd93242-6f2a-44bf-b4ee-b9879850159d_monitoring-system_mosquitto-log-
FILE_NAME = 'mosquitto.log'
NEO4J_URI = 'bolt://localhost:30687'
```

```
NEO4J_USER = 'neo4j'
NEO4J_PASS = '1234'

if __name__ == "__main__":
    local_file = LocalFileAccess(LOCAL_PATH, FILE_NAME)
    log_formatter = FormatLog()
    neo4j_driver = Neo4jDatabaseAccess(NEO4J_URI, NEO4J_USER, NEO4J_PASS)
    network_monitoring = NetworkMonitoring(NEO4J_URI, NEO4J_USER, NEO4J_PASS)
    initialize_system = InitializeSystem()
    initialize_system.demo_setup()
    network_monitoring.start()

    # Was used for performance testing
    #all_execution_times = []
    #i = 1

    loglines =  local_file.tail_file()
    for current_line in loglines:

        # Was used for performance testing
        #start_time = time.time() * 1000

        if current_line != "Socket error on client <unknown>, disconnecting.":
            log_formatter.extract_ip_address_by_row(current_line)
            log_formatter.extract_port_number_by_row(current_line)
            # Extracts time for host an connection
            log_formatter.extract_time_in_seconds_by_row(current_line)
            log_formatter.extract_connection_name_by_row(current_line)
            log_formatter.extract_version_by_row(current_line)

            if None not in log_formatter.host_data.values():
                if not neo4j_driver.check_if_node_exists('Host',
                                                         'ip_address',
                                                         log_formatter.host_data['ip_address']):
                    neo4j_driver.create_host(json.dumps(log_formatter.host_data))

            # Not sure anymore why this line is needed?
            if "/var/lib/mosquitto/mosquitto.db." in current_line:
                log_formatter.reset_connection_data()
                log_formatter.set_connection_status("active")

            if log_formatter.service_data["port"] is not None and log_formatter.service_data["version"] is not No
                neo4j_driver.update_service_port("Mosquitto", log_formatter.service_data["port"])
```

```
            neo4j_driver.update_service_version("Mosquitto", log_formatter.service_data["version"])
            mosquitto_version = log_formatter.service_data["version"]
            mosquitto_port = log_formatter.service_data["port"]
            log_formatter.reset_service_data()

        if log_formatter.connection_data['name'] is not None and log_formatter.host_data['ip_address'] is not

            if not neo4j_driver.check_if_host_is_blocked(log_formatter.host_data['ip_address']):

                if None not in log_formatter.connection_data.values():

                    if not neo4j_driver.check_if_node_exists('Connection',
                                                             'name',
                                                             log_formatter.connection_data['name']):
                        neo4j_driver.create_connection(json.dumps(log_formatter.connection_data))

                    if not neo4j_driver.check_if_constraint_exists('Connection', ['name']):
                        neo4j_driver.create_unique_property_constraint('Connection', ['name'])

                if "New client connected" in current_line:
                    log_formatter.set_connection_status("active")
                    neo4j_driver.update_connection_status(log_formatter.connection_data['name'],
                                                          log_formatter.connection_data['status'])
                    neo4j_driver.update_connnection_time(log_formatter.connection_data['name'],
                                                          log_formatter.connection_data['last_update_time'])
                    starts_connection_data = ('{'
                                              '"edge_name": "STARTS_CONNECTION",'
                                              '"node1": {  "name":"Host",'
                                                         '"property_names": ["ip_address"],'
                                                         '"property_values": ["' + log_formatter.host_data
                                              '"node2": {  "name":"Connection",'
                                                         '"property_names": ["name"],'
                                                         '"property_values": ["' + log_formatter.connectio
                                              '}'
                                             )
                    neo4j_driver.create_edge(starts_connection_data)

                if "New client connected" in current_line:

                    #Default values if nothing could be read from log file
                    if log_formatter.service_data['port'] is None or log_formatter.service_data['version'] is
                        log_formatter.set_service_version("1.6.9")
                        log_formatter.set_service_port("1883")
```

```
                connects_to_data = ('{'
                                    '"edge_name": "CONNECTS_TO",'
                                    '"node1": { "name":"Connection",'
                                            '"property_names": ["name"],'
                                            '"property_values": ["' + log_formatter.connection_da
                                    '"node2": {"name":"Service",'
                                            '"property_names": ["name", "port", "version"],'
                                            '"property_values": ["Mosquitto",'
                                                                '"'+ log_formatter.service_data['
                                                                '"'+ log_formatter.service_data['
                                    '}'
                                   )
                neo4j_driver.create_edge(connects_to_data)

            if 'disconnected' in current_line:
                log_formatter.set_connection_status("inactive")
                neo4j_driver.update_connection_status(log_formatter.connection_data['name'],
                                                      log_formatter.connection_data['status'])
                neo4j_driver.update_connnection_time(log_formatter.connection_data['name'],
                                                      log_formatter.connection_data['last_update_time'])

            if 'disconnecting' in current_line:
                log_formatter.set_connection_status("inactive")
                neo4j_driver.update_connection_status(log_formatter.connection_data['name'],
                                                      log_formatter.connection_data['status'])
                neo4j_driver.update_connnection_time(log_formatter.connection_data['name'],
                                                      log_formatter.connection_data['last_update_time'])

    log_formatter.reset_host_data()
    log_formatter.reset_connection_data()
    # Keep this here so the default value of the connection is set again
    log_formatter.set_connection_status("active")

    # Was used for performance testing
    #end_time = time.time() * 1000
    #execution_time = end_time - start_time
    #all_execution_times.append(execution_time)
    #if i == 100:
    #    all_execution_times = list(filter(lambda num: num != 0.0, all_execution_times))
    #    print(all_execution_times)
    #    print(median(all_execution_times))
    #    print(mean(all_execution_times))
```

```
            #    print(stdev(all_execution_times))
            #i = i+1
```

B.1.7 neo4j_database_access.py

```
'''This module handles CRUD operation for the Neo4j database.
    Adapted from:
    https://neo4j.com/docs/api/python-driver/current/#quick-example
    [last accessed December 2021]

    Author: Thorsten Steuer
    Licence: Apache 2.0
'''
import logging
from typing import List
import json

from neo4j import GraphDatabase
from neo4j.exceptions import ServiceUnavailable
from beartype import beartype


class Neo4jDatabaseAccess:
    '''This class is initialies with a neo4j database driver to communicate
    with the neo4j database instance.
    '''
    @beartype
    def __init__(self, uri: str, user: str, password: str):
        '''Consturctor to initialize the class with a database connection.

        Args:
            uri (str): [description]
            user (str): [description]
            password (str): [description]
        '''
        self.driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        '''Closes the driver object.
        '''
        self.driver.close()
```

```python
@beartype
def create_host(self, host_json: str):
    '''Creates a host node in the database.

    Args:
        host_json (str): data of the host in json format
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
            self._create_and_return_host, host_json)
        for record in result:
            print(f'Created host node: {record["h1"]}')

@staticmethod
def _create_and_return_host(transax, host_json: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        host_json (str): data of the host in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ host_json +'\') as host_data '
        'CREATE (h1:Host { ip_address: host_data.ip_address, '
                            'creation_time: host_data.creation_time, '
                            'notification_sent: host_data.notification_sent, '
                            'is_blocked: host_data.is_blocked })'
        'RETURN h1'
    )
    result = transax.run(query)
    try:
        return [{'h1': record['h1']['ip_address']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def create_connection(self, connection_json: str):
```

```
        '''Creates a connection node in the database.

        Args:
            connection_json (str): data of the connection in json format.
        '''
        with self.driver.session() as session:
            result = session.write_transaction(
                self._create_and_return_connection, connection_json)
            for record in result:
                print(f'Created connection: {record["c1"]}')

    @staticmethod
    def _create_and_return_connection(transax, connection_json: str):
        '''Executes a create statement and loads in json data for new connection.

        Args:
            transax (driver.session): seesion object to execute the query
            connection_json (str): data of the connection in json format

        Returns:
            Iteratable: A list of dictonaries with the data from the query.
        '''
        query = (
            'WITH apoc.convert.fromJsonMap(\''+ connection_json +'\') as connection_data '
            'CREATE (c1:Connection { status: connection_data.status, '
                                    'port: connection_data.port, '
                                    'name: connection_data.name, '
                                    'last_update_time: connection_data.last_update_time}) '
            'RETURN c1'
        )
        result = transax.run(query)
        try:
            return [{'c1': record['c1']['name']}
                    for record in result]
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
    def create_service(self, service_json: str):
        '''Creates a service node in the database.
```

```
    Args:
        service_json (str): data of the service in json format.
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
            self._create_and_return_service, service_json)
        for record in result:
            print(f'Created service node: {record["s"]}')

@staticmethod
def _create_and_return_service(transax, service_json: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        service_json (str): data of the service in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ service_json +'\') as service_data '
        'CREATE (s:Service { name: service_data.name, '
                            'port: service_data.port,'
                            'protocol: service_data.protocol,'
                            'version: service_data.version }) '
        'RETURN s'
    )
    result = transax.run(query)
    try:
        return [{'s': record['s']['name']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def create_vulnerability(self, vulnerability_json: str):
    '''Creates a vulnerability node in the database.

    Args:
        vulnerability_json (str): data of the vulnerability in json format.
```

```
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
            self._create_and_return_vulnerability, vulnerability_json)
        for record in result:
            print(f'Created vulnerability node: {record["v"]}')

@staticmethod
def _create_and_return_vulnerability(transax, vulnerability_json: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        vulnerability_json (str): data of the vulnerability in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ vulnerability_json +'\') as vulnerability_data '
        'CREATE (v:Vulnerability { name: vulnerability_data.name, '
                            'cve_code: vulnerability_data.cve_code,'
                            'effected_protocol: vulnerability_data.effected_protocol,'
                            'description: vulnerability_data.description,'
                            'effected_app: vulnerability_data.effected_app,'
                            'effected_app_version: vulnerability_data.effected_app_version,'
                            'effected_version: vulnerability_data.effected_prot_version}) '
        'RETURN v'
    )
    result = transax.run(query)
    try:
        return [{'v': record['v']['name']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def create_attack(self, attack_json: str):
    '''Creates a attack node in the database.

    Args:
```

```
        attack_json (str): data of the attack in json format.
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
            self._create_and_return_attack, attack_json)
        for record in result:
            print(f'Created attack node: {record["a"]}')

@staticmethod
def _create_and_return_attack(transax, attack_json: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        attack_json (str): data of the attack in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ attack_json +'\') as attack_data '
        'CREATE (a:Attack { name: attack_data.name, '
                            'type: attack_data.type,'
                            'goal: attack_data.goal}) '
        'RETURN a'
    )
    result = transax.run(query)
    try:
        return [{'a': record['a']['name']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def create_precondition(self, precondition_json: str):
    '''Creates a precondition node in the database.

    Args:
        precondition_json (str): data of the precondition in json format.
    '''
    with self.driver.session() as session:
```

```
        result = session.write_transaction(
            self._create_and_return_precondition, precondition_json)
        for record in result:
            print(f'Created precondition node: {record["p"]}')

@staticmethod
def _create_and_return_precondition(transax, precondition_json: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        precondition_json (str): data of the precondition in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    query = (
        'WITH apoc.convert.fromJsonMap(\''+ precondition_json +'\') as precondition_data '
        'CREATE (p:Precondition { name: precondition_data.name, '
                            'type: precondition_data.type,'
                            'capability_level: precondition_data.capability_level, '
                            'description: precondition_data.goal}) '
        'RETURN p'
    )
    result = transax.run(query)
    try:
        return [{'p': record['p']['name']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise


@beartype
def create_edge(self,  edge_data: str):
    '''Creates a edge between two nodes in the database.

    Args:
        connection_json (str): data of the edge in json format.
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
```

```
            self._create_and_return_edge, edge_data)
        for record in result:
            print(f'Created edge: {record["e1"]}')

def _create_and_return_edge(self, transax, edge_data: str):
    '''Executes a create statement and loads in json data.

    Args:
        transax (driver.session): seesion object to execute the query
        connection_json (str): data of the edge in json format

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    '''
    edge_data = json.loads(edge_data)
    where_clause = self._format_json_array_to_where_clause(edge_data['node1'],
                                                           edge_data['node2'])
    query = (
            'MATCH'
            '   (a: ' + edge_data['node1']['name'] + '), '
            '   (b: ' + edge_data['node2']['name'] + ') '
            '' + where_clause + ' '
            'CREATE (a)-[r:' + edge_data['edge_name'] + ']->(b) '
            'RETURN type(r) '
        )
    result = transax.run(query)
    try:
        return [{'e1': record['type(r)']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise


@beartype
def create_unique_property_constraint(self, node_name: str, property_names: List [str]):
    '''Create a unqiue property constraits for a node property.

    Args:
        node_name (str): name of the node the constraint should be applied on.
        property_name (str): name of property the constraint should be applied on.
    '''
```

```
        with self.driver.session() as session:
            result = session.write_transaction(
                self._create_unique_property_constraint, node_name, property_names)
            if result is None:
                property_names_str = ''.join(property_names)
                print(f'Created unique property for node: {node_name} property: {property_names_str}')
            else:
                for record in result:
                    print(f'Created constraint: {record}')

    def _create_unique_property_constraint(self, transax, node_name: str, property_names: List [str]):
        '''Executes a create statement to add a constraint. The return value of the
        create constraint statement is a NoneType object.

        Args:
            transax (driver.session): seesion object to execute the query.
            node_name (str): name of the node the constraint should be applied on.
            property_name (str): name of property the constraint should be applied on.

        Returns:
            Iteratable: A list of dictonaries with the data from the query. In this case
            it will only return None.
        '''

        property_names = self._format_property_name_comma_list(property_names, 'n')
        query = (
            'CREATE CONSTRAINT ON (n:' + node_name + ') '
            'ASSERT ' + property_names + ' IS NODE KEY'
        )
        result = transax.run(query)
        try:
            return result
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
    def check_if_node_exists(self, node_name: str, property_name: str, property_value):
        '''Check if node was already created. If not checked before the neo4j throws an
        error and terminates the program.

        Args:
```

```
            node_name (str): name of the node type to check.
            property_name (str): name of the property which has unique constarint.
            property_value ([type]): value of the property.

        Returns:
            bool: Ture if node exists False otherwise.
        '''
        with self.driver.session() as session:
            result = session.write_transaction(
                self._check_if_node_exists, node_name, property_name, property_value)
        return result

    @staticmethod
    def _check_if_node_exists(transax, node_name: str, property_name: str, property_value):
        '''Execute the query which returns True or False depending if the node exists

        Args:
            transax (driver.session): seesion object to execute the query.
            node_name (str): name of the node type to check.
            property_name (str):  name of the property which has unique constarint.
            property_value ([type]): value of the property.

        Returns:
            bool: Ture if node exists False otherwise.
        '''
        query = (
            'OPTIONAL MATCH (n:' + node_name + '{' + property_name + ':\"' + property_value + '\"})'
            'RETURN n IS NOT NULL AS exists'
        )
        result = transax.run(query)
        try:
            for record in result:
                exists = record['exists']
                if exists:
                    return True
                else:
                    return False
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
```

```
def check_if_constraint_exists(self, node_name: str, property_names: List[str]):
    '''Checks if a constarint is already present in the database. Uses string
    comparison to check if in the original query was node name and proeprty
    name present.

    Args:
        node_name (str): name of the node type to check.
        property_name (str): name of the property which has unique constarint.

    Returns:
        bool: Ture if node exists False otherwise
    '''
    with self.driver.session() as session:
        result = session.write_transaction(
            self._check_if_constraint_exists, node_name, property_names)
    return result

def _check_if_constraint_exists(self, transax, node_name: str, property_names: str):
    '''Executes query to call db.straints which returns a list of all constraints
    in the database.

    Args:
        transax (driver.session): seesion object to execute the query.
        node_name (str): name of the node type to check.
        property_name (str): name of the property which has unique constarint.

    Returns:
        bool: Ture if node exists False otherwise
    '''
    query = (
       'CALL db.constraints'
    )
    result = transax.run(query)
    try:
        evaluation = False
        property_names = self._format_property_name_comma_list(property_names, node_name.lower())
        constraint = (
            'CONSTRAINT ON ( ' + node_name.lower() + ':' + node_name + ' ) '
            'ASSERT ' + property_names + ' IS NODE KEY'
        )

        for record in result:
            description = record['description']
```

```
            if constraint in description:
                evaluation = True

        return evaluation
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def check_if_host_is_blocked(self, ip_address: str):
    """Checks if a host is blocked for new incoming connection or not.
    Args:
        ip_address (str): ip address of host to verify

    Returns:
        bool: Ture if host is currently blocked
    """
    with self.driver.session() as session:
        result = session.write_transaction(
            self._check_if_host_is_blocked, ip_address)
    return result

def _check_if_host_is_blocked(self, transax, ip_address: str):
    """Executes querz to check the block status of a host.

    Args:
        transax (driver.session): seesion object to execute the query.
        ip_address (str): the ip address of the host to be verified

    Returns:
        bool: Ture if host is currently blocked
    """
    query = (
                    'MATCH (h1:Host) '
                    'WHERE h1.ip_address = "' + ip_address + '"'
                    'RETURN h1'
                    )

    result = transax.run(query)

    try:
        is_blocked = False
```

```
            for record in result:
                is_blocked=record['h1']['is_blocked']

            if is_blocked == "True":
                return True
            else:
                return False
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
    def update_connection_status(self, connection_name: str, status: str):
        """Updates the status of a connection to active or inactive

        Args:
            connection_name (str): name of the connection to be updated
            status (str): status to be set for the connection
        """
        with self.driver.session() as session:
            result = session.write_transaction(
                self._update_and_return_connection_status, connection_name, status)
        for record in result:
            print(f'Updated Connection Status: {record}')

    @staticmethod
    @beartype
    def _update_and_return_connection_status(transax, connection_name: str, status: str):
        """Executes the update query for a specific connection.

        Args:
            transax (driver.session): seesion object to execute the query
            connection_name (str): the connection to be updated
            status (str): the status to be set for the speficied conenction

        Returns:
            Iteratable: A list of dictonaries with the data from the query.
        """
        query = (
                'MATCH (c:Connection {name: "' + connection_name + '"}) '
                'SET c.status = "' + status + '" '
                'RETURN c'
```

```
            )
        result = transax.run(query)
        try:
            return [{'c': record['c']['name']}
                    for record in result]
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
    def update_and_return_host_block(self, ip_address: str, is_blocked: str):
        """ Updates the block status of a host. Will be set to True if a hosts
            creates to many active connections.

        Args:
            ip_address (str): ip_address of the host to be blocked
            is_blocked (bool): state of the block status (True/False)
        """
        with self.driver.session() as session:
            result = session.write_transaction(
                self._update_and_return_host_block, ip_address, is_blocked)
        for record in result:
            print(f'Updated block status of host: {record}')

    @staticmethod
    @beartype
    def _update_and_return_host_block(transax, ip_address: str, is_blocked: str):
        """Execute the query to update the host block status.

        Args:
            transax (driver.session): seesion object to execute the query
            ip_address (str): the hosts ip address to be updated
            is_blocked (str): the status to be set (Ture or False)

        Returns:
            Iteratable: A list of dictonaries with the data from the query.
        """
        query = (
                'MATCH (h:Host {ip_address: "' + ip_address + '"}) '
                'SET h.is_blocked = "' + is_blocked + '" '
                'RETURN h'
                )
```

```
    result = transax.run(query)
    try:
        return [{'h': record['h']['ip_address']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def update_and_return_host_notification_sent(self, ip_address: str, sent: str):
    """ Updates the notification status of a host. Once it was blocked an email
    is sent to the admin to notify the precondition.

    Args:
        ip_address (str): ip_address of the host to be blocked
        sent (bool): state of the notification status (True/False)
    """
    with self.driver.session() as session:
        result = session.write_transaction(
            self._update_and_return_host_notification_sent, ip_address, sent)
    for record in result:
        print(f'Updated notification sent of host: {record}')

@staticmethod
@beartype
def _update_and_return_host_notification_sent(transax, ip_address: str, sent: str):
    """Execute the query to update the notification status of the host.

    Args:
        transax (driver.session): seesion object to execute the query
        ip_address (str): the ip address of the host to be updated
        sent (str): status to be set (Ture or False)

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    """
    query = (
            'MATCH (h:Host {ip_address: "' + ip_address + '"}) '
            'SET h.notification_sent = "' + sent + '" '
            'RETURN h'
            )
    result = transax.run(query)
```

```
    try:
        return [{'h': record['h']['ip_address']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def update_connnection_time(self, connection_name: str, time: str):
    """Updates the connection node with the latest timestamp when it was
    last updated.

    Args:
        connection_name (str): name of the connection
        time (str): the timestamp to set
    """
    with self.driver.session() as session:
        result = session.write_transaction(
            self._update_and_return_connection_time, connection_name, time)
    for record in result:
        print(f'Updated Connection Time: {record}')

@staticmethod
@beartype
def _update_and_return_connection_time(transax, connection_name: str, time: str):
    """Executes the update query for a specific connection.

    Args:
        transax (driver.session): seesion object to execute the query
        connection_name (str): the connection to be updated
        time (str): the time to be set for the speficied conenction

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    """
    query = (
            'MATCH (c:Connection {name: "' + connection_name + '"}) '
            'SET c.last_update_time = "' + time + '" '
            'RETURN c'
            )
    result = transax.run(query)
    try:
```

```
            return [{'c': record['c']['name']}
                    for record in result]
        # Capture any errors along with the query and data for traceability
        except ServiceUnavailable as exception:
            logging.error('%s raised an error: \n %s', query, exception)
            raise

    @beartype
    def update_service_version(self, service_name: str, version: str):
        """Updates the service node with the version from the log file.

        Args:
            service_name (str): name of the service
            version (str): version of the service
        """
        with self.driver.session() as session:
            result = session.write_transaction(
                self._update_and_return_service_version, service_name, version)
        for record in result:
            print(f'Updated service version: {record}')

    @staticmethod
    @beartype
    def _update_and_return_service_version(transax, service_name: str, version: str):
        """Executes the update query for a specific service.

        Args:
            transax (driver.session): seesion object to execute the query
            service_name (str): the service to be updated
            version (str): the version to be set for the speficied service

        Returns:
            Iteratable: A list of dictonaries with the data from the query.
        """
        query = (
                'MATCH (s:Service {name: "' + service_name + '"}) '
                'SET s.version = "' + version + '" '
                'RETURN s'
                )
        result = transax.run(query)
        try:
            return [{'s': record['s']['version']}
                    for record in result]
```

```
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def update_service_port(self, service_name: str, port: str):
    """Updates the service node with the port from the log file.

    Args:
        service_name (str): name of the service
        port (str): port of the service
    """
    with self.driver.session() as session:
        result = session.write_transaction(
            self._update_and_return_service_port, service_name, port)
    for record in result:
        print(f'Updated service port: {record}')

@staticmethod
@beartype
def _update_and_return_service_port(transax, service_name: str, port: str):
    """Executes the update query for a specific service.

    Args:
        transax (driver.session): seesion object to execute the query
        service_name (str): the service to be updated
        port (str): the port to be set for the speficied service

    Returns:
        Iteratable: A list of dictonaries with the data from the query.
    """
    query = (
            'MATCH (s:Service {name: "' + service_name + '"}) '
            'SET s.port = "' + port + '" '
            'RETURN s'
            )
    result = transax.run(query)
    try:
        return [{'s': record['s']['port']}
                for record in result]
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
```

```
        logging.error('%s raised an error: \n %s', query, exception)
        raise

@beartype
def execute_and_return_query_result(self, query: str):
    """Execute a query based on the chyper string provided

    Args:
        query (str): a chyper query

    Returns:
        result (list): a list of dictonary each row of the result a dict
    """
    with self.driver.session() as session:
        result = session.write_transaction(
            self._execute_and_return_query_result, query)
        return result

@staticmethod
@beartype
def _execute_and_return_query_result(transax, query):
    """Exectues a query provided by the query parameter

    Args:
        transax (driver.session): seesion object to execute the query
        query (str): a chyper query

    Returns:
        [type]: [description]
    """
    result = transax.run(query)
    try:
        return result.data()
    # Capture any errors along with the query and data for traceability
    except ServiceUnavailable as exception:
        logging.error('%s raised an error: \n %s', query, exception)
        raise


@staticmethod
@beartype
def _format_property_name_comma_list(data: List[str], short_name: str):
    """Format property name from list to chyper query snytax
```

```
        Args:
            data (List[str]): a list of property name strings
            short_name (str): the acronmy use in the chyper query

        Returns:
            [str]: a string containing the property names in the proper chyper format
        """
        if len(data) == 1:
            return '(' + short_name + '.' + data[0] + ')'
        else:
            entries = ''
            for entry in data:
                entries = entries + short_name + '.' + entry + ', '
            formated_entries = entries[:-2]
            formated_entries = '(' + formated_entries + ')'
            return formated_entries

    @staticmethod
    @beartype
    def _format_json_array_to_where_clause(node1: dict, node2: dict):
        """Format the two node description jason the a where clause to create
        a realtion between two nodes.

        Args:
            node1 (dict): description of the first node to match
            node2 (dict): description of the second node to match

        Returns:
            [str]: properly formated chyper where clause
        """
        if len(node1['property_names']) == 1 and node2['property_names'] == 1:
            where_clause=(
                'WHERE a.' + node1['property_names'][0] + '=\"' + node1['property_values'][0] + '\" '
                'and b.' + node2['property_names'][0] + '=\"' + node2['property_values'][0] + '\"'
                 )
        else:
            temp_str = ""
            for i in range(0, len(node1['property_names'])):
                temp_str = (
                    temp_str + 'a.' + node1['property_names'][i] + '=\"'
                    + node1['property_values'][i] + "\" and "
                )
```

```
            for i in range(0, len(node2['property_names'])):
                temp_str = (
                    temp_str + 'b.' + node2['property_names'][i] + '=\"'
                    + node2['property_values'][i] + "\" and "
                )
            where_clause = temp_str[:-4]
            where_clause = 'Where '+ where_clause
        return where_clause



if __name__ == '__main__':
    DUMMY_DATA_FOLDER_PATH = 'C:/Users/Thorsten/Documents/Masterarbeit/Security/ids_slow_dos/detection_system/cod

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_host_data.json', encoding='utf-8')
    HOST_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_connection_data.json', encoding='utf-8')
    CONNECTION_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_service_data.json', encoding='utf-8')
    SERVICE_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_starts_connection_data.json', encoding='utf-8')
    STARTS_CONNECTION_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_connects_to_data.json', encoding='utf-8')
    CONNECTS_TO_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_hosted_at_data.json', encoding='utf-8')
    HOSTED_AT_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_vulnerability_data.json', encoding='utf-8')
    VULNERABILITY_DATA_TEST = json.loads(f.read())
    f.close()

    f = open(DUMMY_DATA_FOLDER_PATH + '/sample_has_vulnerability_data.json', encoding='utf-8')
```

```
HAS_VULNERABILITY_DATA_TEST = json.loads(f.read())
f.close()

f = open(DUMMY_DATA_FOLDER_PATH + '/sample_attack_data.json', encoding='utf-8')
ATTACK_DATA_TEST = json.loads(f.read())
f.close()

f = open(DUMMY_DATA_FOLDER_PATH + '/sample_exploits_data.json', encoding='utf-8')
EXPLOITS_DATA_TEST = json.loads(f.read())
f.close()

f = open(DUMMY_DATA_FOLDER_PATH + '/sample_precondition_data.json', encoding='utf-8')
PRECONDITION_DATA_TEST = json.loads(f.read())
f.close()

f = open(DUMMY_DATA_FOLDER_PATH + '/sample_requires_data.json', encoding='utf-8')
REQUIRES_DATA_TEST = json.loads(f.read())
f.close()

NEO4J_URI = 'bolt://localhost:30687'
NEO4J_USER = 'neo4j'
NEO4J_PASS = '1234'

neo4j_driver = Neo4jDatabaseAccess(NEO4J_URI, NEO4J_USER, NEO4J_PASS)

if not neo4j_driver.check_if_node_exists('Host', 'ip_address', '127.0.0.1'):
    neo4j_driver.create_host(json.dumps(HOST_DATA_TEST))
if not neo4j_driver.check_if_constraint_exists('Host', ['ip_address']):
    neo4j_driver.create_unique_property_constraint('Host', ['ip_address'])

if not neo4j_driver.check_if_node_exists('Connection', 'name', 'mqtt-explorer-0a61e6f1'):
    neo4j_driver.create_connection(json.dumps(CONNECTION_DATA_TEST))
if not neo4j_driver.check_if_constraint_exists('Connection', ['name']):
    neo4j_driver.create_unique_property_constraint('Connection', ['name'])

if not neo4j_driver.check_if_node_exists('Service', 'name', 'mosquitto'):
    neo4j_driver.create_service(json.dumps(SERVICE_DATA_TEST))
if not neo4j_driver.check_if_constraint_exists('Service', ['name', 'version', 'port']):
    neo4j_driver.create_unique_property_constraint('Service', ['name', 'version', 'port'])

if not neo4j_driver.check_if_node_exists('Vulnerability', 'cve_code', 'CVE-2020-13849'):
    neo4j_driver.create_vulnerability(json.dumps(VULNERABILITY_DATA_TEST))
if not neo4j_driver.check_if_constraint_exists('Vulnerability', ['cve_code']):
```

```
        neo4j_driver.create_unique_property_constraint('Vulnerability', ['cve_code'])

    if not neo4j_driver.check_if_node_exists('Attack', 'name', 'DoS'):
        neo4j_driver.create_attack(json.dumps(ATTACK_DATA_TEST))
    if not neo4j_driver.check_if_constraint_exists('Attack', ['name', 'type']):
        neo4j_driver.create_unique_property_constraint('Attack', ['name', 'type'])

    if not neo4j_driver.check_if_node_exists('Precondition', 'name', 'Network Access'):
        neo4j_driver.create_precondition(json.dumps(PRECONDITION_DATA_TEST))
    if not neo4j_driver.check_if_constraint_exists('Precondition', ['name', 'type']):
        neo4j_driver.create_unique_property_constraint('Precondition', ['name', 'type'])

    neo4j_driver.create_edge(json.dumps(STARTS_CONNECTION_DATA_TEST))
    neo4j_driver.create_edge(json.dumps(CONNECTS_TO_DATA_TEST))
    neo4j_driver.create_edge(json.dumps(HOSTED_AT_DATA_TEST))
    neo4j_driver.create_edge(json.dumps(HAS_VULNERABILITY_DATA_TEST))
    neo4j_driver.create_edge(json.dumps(EXPLOITS_DATA_TEST))
    neo4j_driver.create_edge(json.dumps(REQUIRES_DATA_TEST))

    neo4j_driver.update_connection_status('mqtt-explorer-0a61e6f1', 'active')
    neo4j_driver.update_connnection_time('mqtt-explorer-0a61e6f1', '1635015166')

    COUNT_QUERY = (
            'MATCH (h:Host)-[r:STARTS_CONNECTION]->() '
            'RETURN h, count(r) as count'
             )
    count_result = neo4j_driver.execute_and_return_query_result(COUNT_QUERY)

    if count_result is not None:
        for records in count_result:
            print(records)

    neo4j_driver.close()
```

B.1.8 network_monitoring.py

```
"""This module send monitors the network activity by constnatly querying the database.
    Author: Thorsten Steuer
    Licence: Apache 2.0
"""
#from statistics import median, mean, stdev
import time
```

```
import threading
import sys

from alarm_notification import AlarmNotification
from neo4j_database_access import Neo4jDatabaseAccess

class NetworkMonitoring:

    def __init__(self, uri: str, user: str, password: str) -> None:
        self.neo4j_driver = Neo4jDatabaseAccess(uri, user, password)
        self.email_notification = AlarmNotification()
        self.monitoring_status = True

    def _start(self):

        while self.monitoring_status is True:

            # Was used for perfromance test
            #all_execution_times = []
            #for i in range(0, 100):

            start_time = time.time() * 1000
            query = (
                    'MATCH (h:Host)-[r:STARTS_CONNECTION]->(c:Connection) '
                    'WHERE c.status = "active" AND h.notification_sent = "False" AND h.is_blocked = "False"'
                    'RETURN h,count(r) as count'
                    )
            result = self.neo4j_driver.execute_and_return_query_result(query)

            # Was used for perfromance test
            #end_time = time.time() * 1000
            #execution_time = end_time - start_time
            #all_execution_times.append(execution_time)
            #i=i+1

            if result is not None:
                message = ""
                for row in result:
                    if row['count'] > 50:

                        message = 'Host ' + row['h']['ip_address'] + ' has ' +  str(row['count']) + ' active conn
                        self.neo4j_driver.update_and_return_host_block(row['h']['ip_address'], "True")
```

```
                        self.email_notification.connect_to_smtp_server()
                        self.email_notification.send_email(message)
                        self.email_notification.stop_smtp()
                        self.neo4j_driver.update_and_return_host_notification_sent(row['h']['ip_address'], "True"
                        message = ""

                        # Was used for perfromance test
                        #end_time = time.time() * 1000
                        #execution_time = end_time - start_time
                        #all_execution_times.append(execution_time)
                        #i=i+1

            # Was used for perfromance test
            #all_execution_times = list(filter(lambda num: num != 0.0, all_execution_times))
            #print(all_execution_times)
            #print(median(all_execution_times))
            #print(mean(all_execution_times))
            #print(stdev(all_execution_times))
            #time.sleep(1)

        if self.monitoring_status is False:
            sys.exit()

    def start(self):
        threading.Thread(target=self._start, daemon=True).start()

    def stop(self):
        self.monitoring_status = False

if __name__ == "__main__":
    NEO4J_URI = 'bolt://localhost:30687'
    NEO4J_USER = 'neo4j'
    NEO4J_PASS = '1234'
    network_monitoring = NetworkMonitoring(NEO4J_URI, NEO4J_USER, NEO4J_PASS)
    network_monitoring._start()
```

B.2 Mosquitto

B.2.1 mosquitto-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: mosquitto-broker
  labels:
    app: mosquitto-broker
  namespace: monitoring-system
spec:
  selector:
    matchLabels:
      app: mosquitto-broker
  template:
    metadata:
      labels:
        app: mosquitto-broker
    spec:
      containers:
      - name: mosquitto-broker
        image: bigt1991/mosquitto-broker
        ports:
        - containerPort: 1883
        volumeMounts:
        - name: mosquitto-log
          mountPath: /var/log/mosquitto/
      volumes:
      - name: mosquitto-log
        persistentVolumeClaim:
          claimName: mosquitto-log-pvc
```

B.2.2 mosquitto-deployment.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mosquitto-log-pvc
  namespace: monitoring-system
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

B.2.3 mosquitto-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mosquitto-broker-service
  namespace: monitoring-system
spec:
  selector:
    app: mosquitto-broker
  type: NodePort
  ports:
  - port: 1883
    targetPort: 1883
    protocol: TCP
    nodePort: 30883
```

B.3 Neo4j

B.3.1 neo4j-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: neo4j-db
  name: neo4j-db
  namespace: monitoring-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: neo4j-db
  template:
    metadata:
      labels:
        app: neo4j-db
    spec:
      initContainers:
      - name: init-plugins
        image: "appropriate/curl:latest"
        imagePullPolicy: "IfNotPresent"
```

```
      volumeMounts:
        - name: neo4j-plugins
          mountPath: /plugins
      command:
        - "/bin/sh"
        - "-c"
        - |
          curl -L https://github.com/neo4j-contrib/neo4j-apoc-procedures/releases/download/4.3.0.3/apoc-4.3.0.3
          cp -R apoc-4.3.0.3-all.jar /plugins
    containers:
    - image: neo4j:4.3.6-enterprise
      name: neo4j
      env:
        - name: NEO4J_ACCEPT_LICENSE_AGREEMENT
          value: "yes"
      ports:
          - containerPort: 7474
            name: http
          - containerPort: 7687
            name: bolt
          - containerPort: 7473
            name: https
      volumeMounts:
          - name: neo4j-plugins
            mountPath: "/var/lib/neo4j/plugins"
          - name: neo4j-data
            mountPath: "/var/lib/neo4j/data"
    volumes:
        - name: neo4j-data
          persistentVolumeClaim:
            claimName: neo4j-data-claim
        - name: neo4j-plugins
          persistentVolumeClaim:
            claimName: neo4j-plugin-claim
```

B.3.2 neo4j-pvc.yaml

```
kind: "PersistentVolumeClaim"
apiVersion: "v1"
metadata:
  name: neo4j-data-claim
  labels:
```

```
    app: neo4j-db
  namespace: monitoring-system
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
  kind: "PersistentVolumeClaim"
  apiVersion: "v1"
  metadata:
    name: neo4j-plugin-claim
    namespace: monitoring-system
    labels:
      app: neo4j-db
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
```

B.3.3 neo4j-service.yaml

```
apiVersion: v1
kind: Service
metadata:
    name: neo4j-lb
    namespace: monitoring-system
spec:
    type: NodePort
    ports:
      - protocol: TCP
        port: 7474
        targetPort: 7474
        name: http
        nodePort: 30474
      - protocol: TCP
        port: 7473
        targetPort: 7473
        name: https
```

```
    nodePort: 30473
  - protocol: TCP
    port: 7687
    targetPort: 7687
    name: bolt
    nodePort: 30687
selector:
  app: neo4j-db
```

C Test Protocols of Functionality Test

This appendix contains the test protocol for the presented functionality test in section 6.

C.1 Test Protocol mosquitto_log_transformation.py

Test Case ID	IDS_001	Test Case Description	To measure the performance of the uses for-loop in the log transformation module , log files with different sizes were submitted to the loop. The different file sizes were 1, 10, 100 and 1000 MB. From each test scenario, 100 measurements were taken to calculate the median, average, as well as the standard deviation of the execution times.
Created By	Thorsten Steuer	Version	0.1

Hardware	HP ProBook x360 440 G1 equipped with 16 GB RAM and Intel(R) Core(TM) i5-8250U CPU

Tester's Name	Thorsten Steuer	Date Tested	Dezember 20, 2022

Step #	Prerequisites:
1	Uncomment code lines needed for the testing
2	Set up the function perfromance_test_setup in intiialize_system.py module to generate 100 random instance in Neo4j.
3	Call the function in the mosquitto_log_transformation module

Scenario #	Test Data
1	Mosquitto log file with 1 MB
2	Mosquitto log file with 10 MB
3	Mosquitto log file with 100 MB
4	Mosquitto log file with 1000 MB

Test Scenario	Measure performance of mosquitto_log_tranformation.py with different log file sizes

Scenario #	File Size in MB	Median in ms	Mean in ms	Standard Deviation in ms
1	1	38.52	84.46	68.81
2	10	42.80	87.07	73.38
3	100	40.06	89.66	75.46
4	1000	42.66	162.21	200.6

C.2 Testc 1 Protocol mosquitto_log_transformation.py

Test Case ID	IDS_001	Test Case Description	To measure the performance of the uses for-loop in the log transformation module, log files with different sizes were submitted to the loop. The different file sizes were 1, 10, 100 and 1000 MB. From each test scenario, 100 measurements were taken to calculate the median, average, as well as the standard deviation of the execution times.
Created By	Thorsten Steuer	Version	0.1

Hardware	HP ProBook x360 440 G1 equipped with 16 GB RAM and Intel(R) Core(TM) i5-8250U CPU

Tester's Name	Thorsten Steuer	Date Tested	Dezember 20, 2022

Step #	Prerequisites:
1	Uncomment code lines needed for the testing
2	Set up the function perfromance_test_setup in intiialize_system.py module to generate 100 random instance in Neo4j.
3	Call the function in the mosquitto_log_transformation module

Scenario #	Test Data
1	Mosquitto log file with 1 MB
2	Mosquitto log file with 10 MB
3	Mosquitto log file with 100 MB
4	Mosquitto log file with 1000 MB

Test Scenario	Measure performance of mosquitto_log_tranformation.py with different log file sizes

Scenario #	File Size in MB	Median in ms	Mean in ms	Standard Deviation in ms
1	1	38.52	84.46	68.81
2	10	42.80	87.07	73.38
3	100	40.06	89.66	75.46
4	1000	42.66	162.21	200.6

C.3 Test 2 Protocol mosquitto_log_transformation.py

Test Case ID	IDS_003	Test Case Description	To further investigate the query performance, a second test has been conducted measuring only the execution time of the Cypher query itself. The test with ID IDS_002 was repreated but without the SMTP commands. The measurements were this time taken again 100 times since the restrictions of the SMTP server were not present anymore.
Created By	Thorsten Steuer	Version	0.1

Hardware	HP ProBook x360 440 G1 equipped with 16 GB RAM and Intel(R) Core(TM) i5-8250U CPU

Tester's Name	Thorsten Steuer	Date Tested	Dezember 21, 2022

Step #	Prerequisites:
1	Uncomment code lines needed for the testing
2	Set up the function perfromance_test_setup in intiialize_system.py module to generate the requiered amount of instance in Neo4j.
3	Take care to genearte on host with more than 50 opened conncetions.
4	Call the function in the mosquitto_log_transformation module
5	Uncomment the SMTP commands

Scenario #	Test Data
1	Number of Host, Services and Connections = 100
2	Number of Host, Services and Connections = 1 000
3	Number of Host, Services and Connections = 10 000

Test Scenario	Measure performance of network_monitoring.py with different amount of database instacnes

Scenario #	Number of Host, Services and Connections	Median in ms	Mean in ms	Standard Deviation in ms
1	100	16.00	16.75	11.01

D Installation Instruction to Deploy the Test Bed and IDS

This appendix contains instruction to install and deploy the introduced test bed and IDS on windows.

D.1 Installation of Software dependencies

D.1.1 Install Chocolatey

Source: `https://chocolatey.org/install`

- Ensure that you are using an administrative PowerShell.
- Ensure Get-ExecutionPolicy is not Restricted. Use Bypass to bypass the policy to get things installed.
- Run:

```
Get-ExecutionPolicy
```

  If it returns Restricted run:

```
Set-ExecutionPolicy Bypass -Scope Process
```

- Paste following text into the Powershell and press enter:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1
'))
```

- Wait until the process is finished. If you don't see any errors, you are ready to use Chocolatey!

D.2 Install kubectl

Source: `https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/#install-on-windows-using-chocolatey-or-scoop`

- Ensure that you are using an administrative PowerShell.
- Run following command:

```
choco install kubernetes-cli
```

- Ensure the version you installed is up-to-date:

```
kubectl version --client
```

D.3 Install kind

Source: `https://kind.sigs.k8s.io/docs/user/quick-start/#installation`

- Ensure that you are using an administrative PowerShell.
- Run following command:

```
choco install kind
```

D.4 Install base64

- Ensure that you are using an administrative PowerShell.
- Run following command:

```
choco install base64
```

D.5 Install Git

- Ensure that you are using an administrative PowerShell.
- Run following command:

```
choco install git
```

D.6 Install Docker

- Ensure that you are using an administrative PowerShell.
- Run following command:

```
choco install docker-desktop
```

D.7 Create a kind Cluster

- Ensure that you are using an administrative PowerShell.
- Navigate to the main folder of the repository.
- Run following command to create a kind cluster:

```
kind create cluster --config kind_cluster_config.yaml
```

D.8 Deploy service with kubectl

D.8.1 Kubernetes Dashboard

- Open PowerShell.
- Copy Git repository with following command:

```
git clone https://github.com/ThorstenBigT/ids_slow_dos.git
```

- Navigate to the repository folder app_deployments
- Run following command to deploy the needed namespaces:

```
kubectl apply -f namespaces.yaml
```

- Navigate to the repository folder k8s-dashboard
- Install k8s dashboard with following command:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio
/deploy/recommended.yaml
```

- Create a dashboard user with following command:

```
kubectl apply -f dashboard-admin.yaml
```

- Wait until the dashboard is deployed you can check the pod status with following command:

```
kubectl get pods --all-namespaces
```

- To access the dashboard authentication with a token is necessary to get the token use following command:

```
kubectl get secret -n kubernetes-dashboard $(kubectl get serviceaccount
dashboard-admin-user -n kubernetes-dashboard -o jsonpath="{.secrets[0].name}")
-o jsonpath="{.data.token}" | base64 -d
```

- Forward the dashboard traffic with following command to your localhost:

```
kubectl proxy
```

- Copy the previously extracted token and access the dashboard via following url: `http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/`

D.9 Deploy Node-Red

- Open PowerShell
- Navigate to the repository folder app_deployments/rooms/room1
- Create a persistent volume with following command:

```
kubectl apply -f nodered-pcv.yaml
```

- Navigate to the repository folder app_deployments/rooms/ and deploy the additional resoruces with:

```
kubectl apply -f room1
```

- Now you can check via dashboard when the pods are ready to use.
- Afther the set up you can reach the Node-Red instance via `localhost:30880`
- Log in with USER: admin and PASSWORD: password
- Node-Red has an option to import flows. Use this option and navigate in Node-Red to the repository folder app_deployments/rooms/room1/room1_flow_backup and import the flow.
- If you want to deploy the second Node-Red (room2) instance as well follow the presented step once more only exchange the folder location to room2.

- To send data to the broker it first has to be set up. See subsubsection "Deploy Intrusion Detection System". After deploying the broker it should be available on your local machines IP address with port number 1883. If it is not available try the cluster internal IP assigned by Kubernetes. You can find that in the dashboard in the correct namespace under services.

D.9.1 Deploy MQTTSA

- Open PowerShell
- Navigate to the repository folder app_deployments
- Deploy the application with following command:

```
kubectl apply -f slowdos-attacker
```

- Check via dashboard when the pods are ready to use.
- To perform an attack access the pods CLI via Kubernetes and execute following command:

```
python3 mqttsa.py -sc 2000 {IP_OF_THE_BROKER}
```

D.9.2 Deploy Intrusion Detection System

- Open PowerShell
- Navigate to the repository folder app_deployments/monitoring-system
- Create the persistent volume for the Neo4j database:

```
kubectl apply -f neo4j-pvc.yaml
```

- Create the persistent volume for Mosquitto:

```
kubectl apply -f mosquitto-pvc.yaml
```

- Navigate to the repository folder app_deployments/ and apply the rest of the resoruces:

```
kubectl apply -f monitoring-system
```

- Check via dashboard when pods are ready

- Access the Neo4j Browser Interface via `http://localhost:30474/browser/` and change the connection URL to `bolt://localhost:30687`. Username is neo4j and password is neo4j as well.
- Neo4j will ask you to set a new password use 1234 since the detection system uses this password to import the data to the database.
- Access the CLI of the detection plod via the k8s dashboard and navigate to the folder ids_slow_dos. There you should find a folder named detection_system/code.
- To monitor the Mosquitto log file you will have to change the file location in the file mosquitto_log_transformation.py. Kind automatically created a folder when it was executed under 'C:/kind_persistent_volume/pvc-0dd93242-6f2a-44bf-b4ee-b9879850159d_monitoring-system_mosquitto-log-pvc'. The folder will have the same structure but will have a different hash value.
- In the detection_system/code folder you can start the monitoring with following command:

```
python3 mosquitto_log_transformation.py
```