FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Vulnerability Prioritization in the Cloud Environment
Student:	Bc. Ondřej Vokoun
Supervisor:	RNDr. Daniel Joščák, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2022/2023

Instructions

Vulnerability management (VM) is essential for organizations to prioritize threats and minimize their attack surface. VM is ongoing, regular process which consists of five core parts:
identification, assessment, prioritization, remediation and measurement of progress. Those are usually well covered and supported by tools from cloud providers, except the prioritization.

The aim of this work is to develop a tool that will provide insight into the cloud network infrastructure and identity access for VM teams in regards to prioritizing vulnerabilities. We will consider the following vectors that could contribute to the prioritization process: vulnerability severity scoring, endpoint reachability, identity access and custom tags.

Steps to cover:
1. Analyze current solutions and tools for analyzing network reachability and identity accessibility - focus on AWS.
2. Design own solution or enhance existing one.
3. Implement the proposed solution.
4. Test the solution on sample data.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 7 February 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Vulnerability Prioritization in the Cloud Environment

Bc. Ondřej Vokoun

Department of Information Security
Supervisor: RNDr. Daniel Joščák, Ph.D

May 2, 2022

Acknowledgements

First of all, I would like to thank the supervisor, RNDr. Daniel Joščák, Ph.D., for guidance, valuable advice and suggestions. I gratefully acknowledge the help of my colleagues Štěpán Šimek and Zdeněk Sloupenský for their patience and practical suggestions. Most importantly, I would like to thank my parents and my brother for their endless support and encouragement during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for nonprofit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 2, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vokoun, Ondřej. *Vulnerability Prioritization in the Cloud Environment.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce se zabývá prioritizací zranitelností v prostředí cloudu, zejména pak v AWS. Na začátku je popsán proces vulnerability managementu následovaný VM procesem v cloudu a jeho odlišnostmi. Následuje analýza frameworků pro hodnocení závažnosti zranitelností a nástrojů pro jejich prioritizaci. S využitím předchozích znalostí je navržena a implementována aplikace CloneM, která dokáže prioritizovat zranitelnosti nalezené na EC2 instancích. Výstupem aplikace je risk skóre, které bere v potaz CVSS skóre, možnost vzdálené exploitace zranitelnosti či důležitost aktiva.

Klíčová slova management zranitelností, prioritizace zranitelností, skenování zranitelností, cloud, AWS

Abstract

The focus of this thesis is on prioritizing vulnerabilities in the cloud environment, specifically in AWS. The general vulnerability management process is described, followed by the specifics of the process in the cloud. Risk scoring frameworks and tools available for prioritization in the cloud are discussed. The application called CloneM is designed using the knowledge gained from the previous research. As a result, the developed application allows prioritization of vulnerabilities found on EC2 instances based on the risk factors such as CVSS score, vulnerability exposure and asset importance. The application is evaluated on sample data.

Keywords vulnerability management, vulnerability prioritization, vulnerability scanning, cloud, AWS

Contents

List of Figures

List of Tables

List of Listings

Introduction

In today's world, modern businesses have data centers with servers, firewalls, load balancers, routers, switches and other kinds of devices. Some use the cloud for hosting their applications and workloads, and their employees have business laptops, smartphones and other devices. All these devices do have vulnerabilities, without exception. National Vulnerability Database (NVD) reported over 20000 new vulnerabilities found in 2021, with over 2600 with critical severity [1]. These numbers are alarming and are likely to grow even more as we adopt more technologies.

Critical vulnerabilities are at constant risk of being exploited by attackers to steal sensitive user data, intellectual property, put ransomware, keyloggers or other malware onto devices. Since so many critical vulnerabilities are out there, organizations simply cannot patch them all. Instead, they need to focus their resources on remediating those that pose the highest risk to the organization. This thesis aims to analyze current tools for prioritizing vulnerabilities in the cloud and proposes new application CloneM, that was developed.

The thesis is divided into the following chapters. The first chapter, *Introduction to Vulnerability Management*, is dedicated to the introduction to vulnerability management, describing basic concepts and terms. The second chapter, *VM in the Cloud*, discusses the VM process in the cloud and how it deviates from the general VM process. The third chapter, *Design of the CloneM*, introduces the main goals of the application and proposes its design. Chapter *Implementation of the CloneM* describes the implementation details of the application. The last chapter, *Evaluation, use-cases and testing of the CloneM*, is dedicated to evaluating the implemented application on the sample data.

Chapter 1

Introduction to Vulnerability Management

Vulnerability management (VM) is a continuous process of automated asset discovery, vulnerability scanning, prioritization, reporting and remediation. Typically, a security team leverage a vulnerability management tool to detect vulnerabilities and utilize different processes to report and remediate them. Well-developed VM programs utilize IT and business operations knowledge to prioritize risk and address high-impact vulnerabilities as quickly as possible.

1.1 Key terms used in VM

Essential terms relevant to security and VM are shortly explained. There are countless definitions of these terms in various acts and standards. The following were chosen.

Stakeholder

"Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations." [2]

Asset

"The term asset refers to an item of value to stakeholders. An asset may be tangible (e.g., a physical item such as hardware, firmware, computing platform, network device, or other technology component) or intangible (e.g., humans, data, information, software, capability, function, service, trademark, copyright, patent, intellectual property, image, or reputation). The value of an asset is determined by stakeholders in consideration of loss concerns across the entire system life cycle. Such concerns include but are not limited to

business or mission concerns. The meaning of loss and the associated consequences of loss vary based on the nature of the asset. For example, a data or information asset will have a different loss interpretation than a capability or function. The value of an asset can also be represented in different ways to include criticality, irreplaceability, and the degree to which the asset is relied upon to achieve stakeholder objectives. From these characteristics, the appropriate protections are engineered to provide the requisite system security performance and effectiveness and to control, to the extent reasonable and practical, asset loss and the associated consequences." [3, p. 13]

Vulnerability

"A known weakness in a system, system security procedures, internal controls, or implementation by which an actor or event may intentionally exploit or accidentally trigger the weakness to access, modify, or disrupt normal operations of a system resulting in a security incident or a violation of the system's security policy." [4, p. 212]

Exploit

"A technique to breach the security of a network or information system in violation of security policy." [5]

Threat

Threat is an event, either intentional or unintentional, capable of causing harm to an asset, undesirable consequences or impact from such loss. [3, p. 175]

Risk

Risk expresses the impact of a threat that exploits a vulnerability. It is measured in terms of the probability of occurrence and its consequence. [6]

Incident

A security incident, cybersecurity, or computer security incident is a violation of information security in an information system, information processed by the system, security policies or procedures. Its occurrence results in actual or potential jeopardy to the confidentiality, integrity, or availability of the information. [7, 8]

1.2 Vulnerability Management Life Cycle

VM life cycle faces numerous challenges. For easier understanding, it can be broken down into stages, where each tackles a different problem. As discussed

in the NIST publication [9], managing assets on a large scale can be challenging due to the vast diversity of hardware and software.

With such a diverse ecosystem, vulnerability assessment of all devices poses another challenge for the security teams. For the assessment, teams utilize vulnerability scanners. However, scanners can output several false-positive findings and 'noise', which would slow down response to problems that need to be fixed. Hence, prioritization of such findings is needed.

Moreover, an organization can have outstanding results from scanning and prioritizing; reporting is a key to a successful vulnerability management program. Effective reporting to the asset owners helps the organization act faster upon found vulnerabilities and improve the organization's security posture. Once the asset owners are informed about the findings, it is up to them to remediate the findings in a timely manner following the service level agreements (SLAs) defined by the organization. Last but not least, verification of the remediated findings helps the organization understand the effectiveness of the processes and the vulnerability program.

All the steps described above form a vulnerability management life cycle, which is visualized as a cycle in Figure 1.1. The following sections describe each stage.

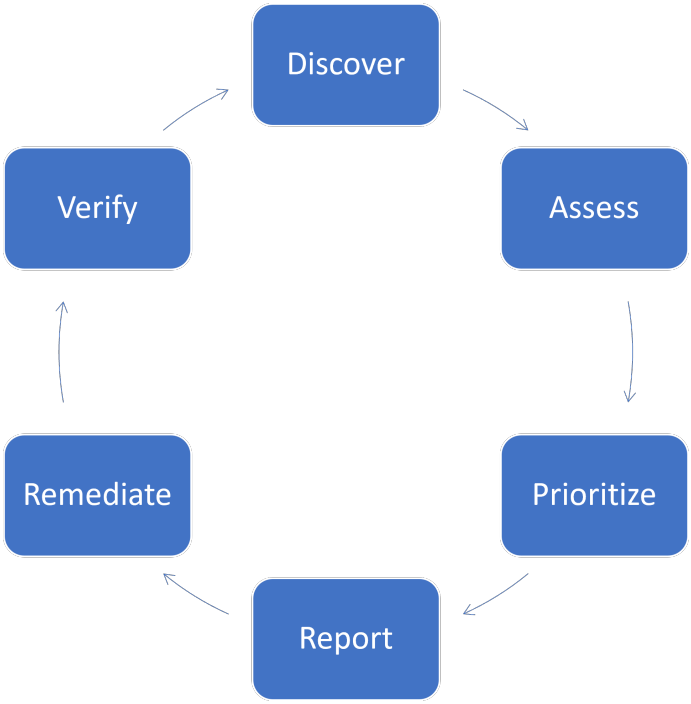

Figure 1.1: Vulnerability Management Life Cycle diagram.

1.3 Asset Discovery

An organization has to know its attack surface before taking action to protect it adequately. The environment can be very diverse, including on-premise servers, network devices, cloud infrastructure, portable and mobile devices, Internet of Things (IoT) devices, Industrial IoT (IIoT) devices and other kinds of devices, making inventorying difficult. The objective of asset discovery is to map every hardware and software asset across the organization. It helps to assess risk and identify asset owners who are responsible for mitigating the risk in later stages. Not only is the discovery important for vulnerability management, but it is also part of security frameworks like NIST Cyber Security Framework [10, p. 24], ISO-IEC 27000 [11], or CIS Controls [12].

1.3.1 Service asset and configuration management

There are few strategies that can be utilized to cover asset discovery and reporting steps, such as service asset and configuration management (SACM), IT service management (ITSM), configuration management or IT asset management (ITAM). They are quite similar to each other with the goal to properly track assets and configuration items (CIs) in their lifecycle with their respective owners.

Configuration management is enabled by CMDB (configuration management database) and stores CIs within their lifecycle and manages relationships between them. ITAM is a process to ensure that assets are properly accounted for, deployed, maintained, upgraded and disposed at the end of their lifecycle [13]. SACM (or ITSM) combines both processes, asset management and configuration management. There are many vendors of these platforms, just to mention a few, such as ServiceNow (IT Asset Management, IT Service Management, CMDB), Jira (Service Management), BMC (IT Service Management), or SolarWinds (IT Asset Management, IT Service Management).

1.4 Vulnerability Assessment

Vulnerability assessment refers to the assessment stage of the VM Life Cycle. Committee on National Security Systems (CNSS), in their glossary [4, p. 212], describes vulnerability assessment as a "systematic examination of an information system or product to determine the adequacy of security measures, identify security deficiencies, provide data from which to predict the effectiveness of proposed security measures, and confirm the adequacy of such measures after implementation."

Assessing an organization's attack surface via vulnerability scanning helps understand the risk and adequately protect it. In the NIST glossary [14], vulnerability scanning is referred to as "a technique used to identify hosts or

host attributes and associated vulnerabilities." Scan results are then compared against a database of known vulnerabilities to identify possible gaps in security.

However, there are some challenges with the scanning. Diverse asset types might need different tools and scanners for assessment. This increases the complexity that needs to be handled by VM teams. Another thing that teams need to have in their mind is that scanning can pose a load on the network and its core devices like switches, routers or firewalls. Scanning can also disrupt the operation of the devices. In that case, passive monitoring might be a better approach. Last but not least, it is worth mentioning that vulnerability scanning is excellent for regular testing. However, penetration tests should be included in the organization's security program for more thorough tests.

1.4.1 Scan direction

Vulnerability scans can be differentiated based on the network from which they scan. Internal scans are deployed inside the corporate network and identify vulnerabilities within it. Internal scans are useful in scenarios where an attacker has already compromised some devices and has access to the network. These scans then help to prevent from attacker moving laterally to other devices on the network. As opposed to internal scans, the external scans are targeted at the Internet-facing infrastructure of the company. [15]

1.4.2 Level of access privileges

Regarding the level of access privileges, there are two general types of scans, one being an unauthenticated (or sometimes called non-authenticated or non-credentialed) scan, the other one authenticated (also known as credentialed). As the names suggest, an unauthenticated scan does not have system privileges. Its main goal is to discover and enumerate services, open ports, and protocols that are exposed on the host. Unauthenticated scans can identify vulnerabilities including, but not limited to, expired certificates, exposed services unpatched software, weak passwords, or poor encryption protocols. This approach emulates the attacker from the outside. It is limited by missing client-side vulnerabilities such as detailed patch information. [16]

On the other hand, authenticated scans are more in-depth as they can log in to the device with a regular user. The most significant advantage is that the authenticated scan can directly look up installed software, including the version number and applied patches. It can also help with misconfigurations such as bad password policies. Authenticated scans are required when doing compliance scans on the host.

It is worth noting that authenticated scans add complexity to the credentials management. This means that there must be a valid credential with adequate permissions for every scanned host. Teams also need to monitor if the authentication was successful or not. Authentication can fail due to many

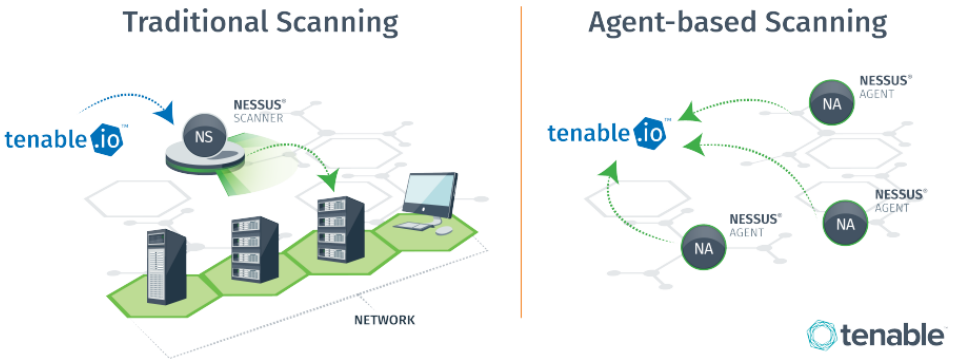

Figure 1.2: The difference between traditional and agent scanning [17].

reasons, like expired password or certificate, wrong password, removed user from the device or any other reason that could prevent from authenticating. When the permissions were mentioned, it is common for the scanner to log in with the regular user rather than with a root or admin. In many scenarios, however, a regular user does not have adequate permissions, which means an escalation of privileges is needed, and it might need a different password for escalation from the one that the scanner uses for login.

Besides these two types mentioned above, another type that stands alone is agent scan. The main difference between traditional network-based scan and agent scan is that in the traditional approach, the scanner reaches out to the target over the network, while agent-based scanning works the opposite way. The agent runs on the host, performs the scan and then reports back to some manager or collector, which will collect the output. To make better sense of it, the difference between these two can be seen in Figure 1.2, in which Tenable.io acts as a scanner manager.

1.4.3 Scope of scanned devices

Besides the types that were mentioned in the previous text, vulnerability scans can be categorized based on their primary focus. Their focus might be limited to just some specific devices, like user workstations, mobile devices, web applications, databases, network devices, containers, etc. Or, it can be a comprehensive scan that scans all kinds of devices connected to the network.

1.4.4 Scheduling

How often an organization scans the assets is not a simple question. Most security frameworks do not specify the frequency of how often the scans should run. And even though some do, like PCI DSS, which requires quarterly scans

[18], that might not be enough to stay secure. There are multiple approaches that can be considered.

It is important to run scans regularly in a more static environment such as on-premises. It can be on a monthly, weekly or even daily basis, which depends on the maturity of the cybersecurity program and on the organization itself. Some organizations like financial or healthcare institutions might be more sensitive to cyberattacks, requiring scans more often.

In the change-based approach, scans run after every minor change made to the application or system. This suits well for cloud environments, which tend to implement continuous integration and continuous deployment (CI/CD) pipelines, and there might be new features introduced on a daily basis. Putting it together in a cloud environment or any other dynamic environment is a sensible approach to run a scan after each change [19]. When the environment is so dynamic, it is also worth considering including the scans in the pipeline itself, which could prevent deploying a new application build when high-risk vulnerabilities are present.

That was all about vulnerability scans, but the vulnerability management team might run compliance scans as well. This might be needed for audit reasons, where an annual scan could be enough, or compliance with standards that define the frequency of the scans.

1.4.5 Environments

There are few noticeable differences between scanning on-premises and the cloud. There are two ways how to detect vulnerabilities in the cloud assets. Either by using a third-party tool, like Nexpose, Nessus, etc. or using a cloud provider's native tool. This might be Qualys in Microsoft Azure, Amazon Inspector for AWS and so on. Both approaches have their own pros and cons.

Native tools tend to be a cheaper option with a payment model pay-as-you-go, built into the cloud and integrated well with cloud provider services. Usually, they are less feature-rich than third-party tools, results of scans might lack information or be incomplete, and also, capabilities of remediating and reporting might be limited.

On the other hand, non-native tools are way more expensive if they are not open-source. They might lack the support of some cloud services. It might take more time to integrate with services for vendors of these solutions. On the pros side, the companies behind these tools are usually companies with years of experience with on-premise scanning. Thus, the scans provide better information about found vulnerabilities, have more capabilities in terms of reporting, etc. Another advantage is also using the same tool for multi-cloud setups or hybrid setups (scanning both on-premise and cloud).

1.4.6 Vulnerability scanning tools

There are tens of scanning tools available on the market, some of which are open-source and some commercial only. Providing a complete list of available vulnerability scanners here would not make much sense. Instead, it is provided with a link to the OWASP web page, on which the OWASP project maintains a list of vulnerability scanning tools.[1] Just to mention a few popular options:

- Burp Suite
- Tenable Nessus
- Rapid7 Nexpose
- Nikto
- Nmap
- Greenbone OpenVAS
- Qualys Guard

1.5 Prioritization

As the next step comes prioritization step. Prioritization of vulnerabilities is important due to one fact. It is impossible for organizations to patch and remedy every single vulnerability found. And it might even be a waste of resources. Prioritization helps teams to focus on vulnerabilities with high impact. After all, according to Kenna Security research [20], only approximately 2% of published vulnerabilities have exploits in the wild.

There are multiple risk factors associated with the vulnerability. Common strategies of vulnerability prioritization discussed in [21] consider the following factors:

- vulnerability severity score,
- threat context,
- exposure,
- business impact.

Vulnerability scoring plays a key role in vulnerability management as it is used to determine a potential risk that vulnerability may have on a system and therefore is important for prioritization. There are multiple scoring systems and frameworks. Some are briefly explained in the following text.

[1] List can be found at `https://owasp.org/www-community/Vulnerability_Scanning_Tools`

The Real Risk prioritization scoring framework is developed by Rapid7 and used in their products like InsightVM. The Real Risk Score (RRS) considers the CVSS score, malware exposure, exploit exposure, exploit complexity, and vulnerability age to prioritize vulnerabilities. The output of the RRS is a granular risk score on a scale from 1 to 1000. [22]

Skybox Risk Score is used in the Skybox Vulnerability Control product developed by Skybox Security. It is a complex risk scoring system with a customizable risk formula. The risk formula supports the most important aspects of the prioritization, such as severity score, exposure in the network, the exploitability of a vulnerability, and the asset's importance. Each customer can adjust the formula and the weights of each category which contributes to the final risk score. The given visibility to the network by Vulnerability Control helps to determine exposed vulnerabilities with active exploitation and thus provides risk-based vulnerability management. [23]

1.5.1 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) is an open framework managed by FIRST.Org, Inc, a US-based non-profit organization that provides principal characteristics of a vulnerability. Its part is a numerical score reflecting vulnerability's severity for software, hardware or firmware vulnerabilities and also produces qualitative severity ratings such as None, Low, Medium, High or Critical (for CVSS v3). [24, 25]

As described in [25], CVSS consists of three metric groups:

- **Base:** represents the innate characteristics of each vulnerability that are constant over time and environment.
- **Temporal:** adjusts the Base severity of a vulnerability depending on factors that change over time, such as the availability of exploit code.
- **Environmental:** adjusts the Base and Temporal severities to a specific computing environment. They consider factors such as the presence of mitigations in that environment.

CVSS has three major versions of which, the most recent, as of writing this thesis, being CVSS v3.1. Version v3.1 slightly differs from v3.0 in guidance on how to clarify vulnerabilities. Version v3.1 is also nowadays used by National Vulnerability Database (NVD) for assigning the score to the new CVEs. [24] Comprehensive documentation for CVSS v3 can be found on the official web page.[2] Mappings between numerical score and qualitative representation are presented in Table 1.1.

The major flaw of the CVSS is that it is designed to identify the technical severity of a vulnerability, but it is more often used for scoring a security

[2] `https://www.first.org/cvss/specification-document`

Severity	CVSSv3 Range
Critical	9.0 to 10.0
High	7.0 to 8.9
Medium	4.0 to 6.9
Low	0.1 to 3.9
None	0

Table 1.1: CVSSv3 severity scoring and quantitative representation [24].

risk since that is what people and organizations need. Why the CVSS fails at scoring a security risk is well explained in the white paper [26] – *Time to Change the CVSS?* Not only the CVSS is not meant for prioritizing, but also the fact that a high percentage of vulnerabilities are scored with critical severity makes CVSS cumbersome. As of 2020, there were more than 16000 vulnerabilities reported with CVSSv2 rated as 9.0 or higher, which makes it 13% of all vulnerabilities. CVSSv3 is no different, with more than 60000 vulnerabilities rated with CVSSv3 which 9400 are rated as 9.0 or higher. [27] This makes it extremely difficult to prioritize based on just a CVSS score, which is why other vendors came up with their scoring for prioritization.

1.5.2 Tenable Vulnerability Priority Rating

Vulnerability Priority Rating (VPR) developed by Tenable, Inc, is a more modern approach to vulnerability risk rating. Tenable is the industry solutions provider for vulnerability management and is also behind the Nessus vulnerability scanner, a de facto industry standard for vulnerability scanning. Unlike CVSS, VPR takes into account the relevancy of the vulnerability and proactively reevaluates the risk score based on the information. Its goal is to solve the enormous number of vulnerabilities with a critical CVSS rating. This does not mean simply reducing the number of critical vulnerabilities but rather observing the likelihood of actual exploiting of the vulnerability and rating the vulnerabilities in a more appropriate manner. The Figure 1.3 shows the comparison in distribution between CVSSv3 and VPR.

Although VPR is a more risk-driven approach to scoring the vulnerabilities, it is not an open-source framework, thus being only part of the Tenable products and cannot be widely adopted by the industry. Furthermore, it is worth mentioning that the VPR does not treat the CIA triad the same way CVSS does, and it considerably does not play a key role in the score. [28]

Tenable calculates a dynamic VPR for most vulnerabilities. It is a dynamic companion to the data provided by the vulnerability's CVSS score since Tenable updates the VPR to reflect the current threat landscape. VPR values range from 0.1 to 10.0, with a higher value representing a higher likelihood of exploitation. [29] Mapping table between numerical score and qualitative

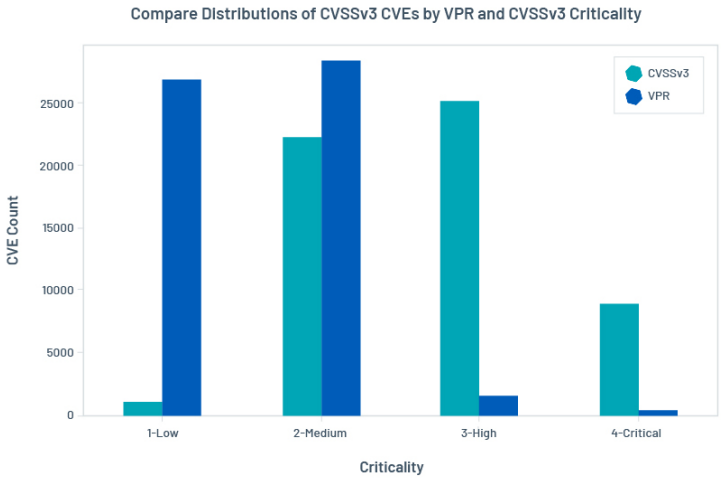

Figure 1.3: Distributions of CVSSv3 CVEs by CVSSv3 and VPR Criticality [27].

VPR Category	VPR Range
Critical	9.0 to 10.0
High	7.0 to 8.9
Medium	4.0 to 6.9
Low	0.1 to 3.9

Table 1.2: VPR Scoring [29].

representation is shown in Table 1.2. Note that VPR does not include *Info* severity.

Furthermore, users cannot adjust the ranges based on their needs. Vulnerabilities without CVEs in the National Vulnerability Database (NVD) do not receive a VPR score and these vulnerabilities are usually assigned with info severity. According to Tenable, they should be remediated based on their CVSS severity [29].

In this section, some scoring frameworks were discussed with their advantages and disadvantages. Some are purely for assigning severity (like Common Vulnerability Scoring System), while others are trying to take into account the threat context, such as Tenable Vulnerability Priority Rating. The exposure context for prioritization is included in tools like Skybox Vulnerability Control with their Skybox Risk Score.

It is important to keep in mind that there is not any one-size-fits-all approach for scoring. Especially if the scoring is not fully aware of the environ-

ment, for example, network exposure, business criticality, identity access and other factors.

1.6 Reporting

Discovering assets, assessing them and prioritizing the found vulnerabilities are fundamental to the VM. But for the organization to be able to reduce the risk and attack surface, reporting to the right people is key part of the process. It is critical to address found vulnerabilities to the asset owners in a timely manner, so they can act upon a findings criticality. This also refers back to the importance of proper asset management.

Each vulnerability after prioritization has its risk rating, which also defines remediation target time. With higher criticality comes shorter timelines for fixing the vulnerability. Reporting can be part of the tool used for scanning and prioritizing, but it can be done externally.

1.7 Remediation

Many vulnerability scanners not only discover vulnerabilities, but can also propose a solution to fix the found vulnerability, which is important in the remediation step. Asset owners are responsible for remediating found vulnerabilities. The remediation process is based on the remediation target timelines, which the organization processes define. Remediation can include patching, blocking network access, removing permissions, removing components of the system or application, or any other solution that prevents exploiting the vulnerability.

If the vulnerability cannot be patched or mitigated any other way, risk acceptance may be required. Risk acceptance has to provide alternate mitigation options, documentation and processes on how to act upon exploitation. It might need approval from a higher security officer in the organization. [30]

1.8 Verification

Verification is the last step of the cycle and is closely connected with the assessment step. Once the vulnerabilities are mitigated, re-scanning has to be done to make sure vulnerabilities are not present in the system anymore.

Chapter 2

VM in the Cloud

Chapter 1 was dedicated to introducing vulnerability management and its key components and processes. This chapter analyzes VM life cycle in the cloud and its specifics. A few paragraphs are dedicated to the identity and access management in the cloud.

2.1 Cloud computing

In simple terms, cloud computing delivers computing services with on-demand access. The access is given over the Internet, thus called 'cloud computing' [31]. Computing resources might be physical and virtual servers, applications, databases, storage, networs, software, analytics or intelligence. A third party typically maintains these resources, known as a cloud provider or cloud service provider (CSP).

In terms of the payment model, most services are provided on a pay-as-you-go basis. Also, there are possibilities where customers can pay for months/years upfront to get a better price. The disadvantage is that once the resources are not needed anymore, it has already been paid for them. Multiple types of clouds exist, like public, private, or hybrid clouds. These types are explained in more detail in Section 2.1.1. Different types of provided services are categorized into Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These are explained in Section 2.1.2.

2.1.1 Types of Cloud

As mentioned earlier, multiple cloud deployment options are public, private, and hybrid. Public clouds are the most common type of cloud deployment. They ease the responsibility for managing the infrastructure and lower the cost of maintenance since they are hosted by a public cloud providers such as AWS, Azure, Google Cloud, IBM Cloud, or any other provider. The cloud

provider owns all hardware, software, and any other infrastructure. An important aspect of the public cloud is that the infrastructure is shared between customers, so-called tenants.

Even though the CSP owns the core infrastructure and guarantees its security with SLA to customers, the customers have several responsibilities too. This model is known as the shared responsibility model. When the company moves to the cloud, standard practice is that DevOps teams take ownership and responsibility of the deployed virtual machines, network appliances, and other services, which means that the teams are responsible for what they put on the infrastructure and its security. [32]

On the other hand, private cloud infrastructure is dedicated to only one customer. Since it is dedicated to one customer, the term on-premises data center can also be used. The organization can own it or manage it by a third-party company. The private cloud is a great choice for customers who need to meet specific requirements. It can apply, for example, to government agencies or financial institutions.

As the name suggests, a hybrid cloud is the combination of the previous two mentioned, public and private clouds. It gives the advantages of both solutions but also brings additional complexity to management. [31]

2.1.2 Types of services

Most cloud services fall into three categories, IaaS, PaaS or SaaS. Together, they form the cloud computing stack. Other types of services could be Data as a Service (DaaS), Analytics as a Service (AaaS), and others. However, these are usually empowered through SaaS, so they will not be described furthermore. The differences between the main three types are described below.

Infrastructure as a Service or IaaS for short, is closest to the traditional on-premises infrastructure. Users of the cloud can rent virtual machines, storage, networks, operating systems, or other infrastructure. Their responsibility includes operating systems, any data, middleware, applications, etc.

PaaS is stepping up further in the model, where the provider takes more responsibilities. CSP owns infrastructure, hardware and software to deliver the platform as an integrated ready-to-use solution accessible via the Internet. Customers do not have to worry about updates or maintenance. Some examples can include services like AWS Elastic Beanstalk, Heroku, or Red Hat OpenShift. [33]

The last type is SaaS, also known as cloud application services. It features a pay-as-you-go payment model or customers paying monthly/yearly fees to use the application. The application and all the underlying infrastructure, like servers, storage, network, etc., are managed by vendor of the application. [34]

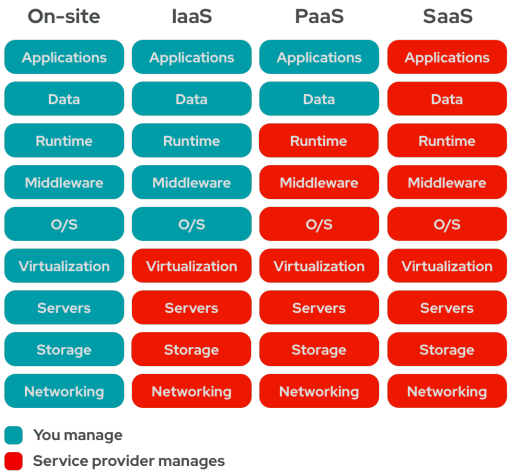

Figure 2.1: Differences in responsibility for different types of cloud service models [33].

2.1.3 Shared responsibility model

Previous paragraphs described the type of services available in the cloud. The crucial part of this segregation is that each type has different responsibilities for both provider and customer. This segregation is known as the shared responsibility model, which defines responsibilities for providers and customers for each type of service. Figure 2.1 shows how the responsibilities are divided for each service category. The main focus of this work is on IaaS cloud services on which customer has the most responsibilities.

2.1.4 Regions and availability zones in AWS

AWS uses the concept of regions and availability zones (AZs). It is important to understand the difference between them and how it impacts deployed resources. The region consists of multiple isolated and physically separate AZs within a geographic area. All AZs within one region are interconnected with high-bandwidth and low-latency networking, giving customers the option to operate applications that need replication or high availability [35]. For some resources, customer cannot specify the availability zone in which the resource will be created. An example could be VPCs or subnets. From a networking perspective, it does not matter in which AZ customer deploys resources such as EC2s. The traffic will be routed between them regardless of AZ.

On the other hand, regions are completely separated between themselves. Customers can deploy two VPCs in different regions, no routing will take place between them by default.

2.2 Specifics of VM in the Cloud

In the first chapter 1, the general VM process was described. This section is dedicated to the VM process in the cloud, its specifics and how it deviates from the general VM life cycle.

2.3 Asset Discovery

ITAM in the cloud is a bit more straightforward than on-premises. Cloud providers provide tools and services to audit and map all resources deployed in one place in a standardized form. For this purpose, CSPs offer a master account. Alternatively, sometimes called an organizational or management account. An example could be a management account in AWS Organizations [36]. This kind of account can manage other accounts in the organization, configure services for the whole organization and collect data from all member accounts. With this feature, it is a much simpler task to manage all assets in the cloud.

When the organization uses multiple CSPs or combines them with on-premises as a hybrid environment, ITAM can get a bit more complex. In that case, an organization needs to import data from every CSP and on-premises, which makes it more difficult.

2.4 Vulnerability Assessment

Vulnerability assessment and scanning do not differ much from scanning on-premises. It can utilize the same set of tools to do so or native tools provided by the CSP. The biggest difference is the life cycle of the assets deployed in the cloud.

Typically, assets in the on-premise environment are deployed with a much longer life cycle than cloud assets. Assets in the cloud can be deployed for months, weeks, days, or even hours. Since cloud assets can be deployed only for a short period, it might require specifics on when the actual asset will get scanned.

Another slight difference is the way how the assets can get scanned. One example could be the infrastructure deployed in the pipeline. Pipeline defines exactly what will be deployed (image of the OS, software and its version, other components). Therefore assessment can be done in the pipeline before actual deployment takes place. It depends on defined processes by the organization, but for instance, if vulnerabilities are found, it can even stop the pipeline and prevent the resource from being deployed.

2.4.1 Amazon Inspector2

Amazon provides a tool called Amazon Inspector for vulnerability management in their cloud. This tool has been available since 2015 in AWS. In November 2021, Amazon introduced a new version of it, which overtook the name Inspector and the previous version got renamed to Amazon Inspector Classic. New Inspector is referred to as Inspector2 in the SDK to get even more confusing, while the old version is just Inspector. From now on, the new version will be referred to as Inspector2, while the previous version will be as Inspector, or explicitly Inspector Classic.

It supports scanning Amazon Elastic Cloud Computing (EC2) instances and Amazon Elastic Container Registry (ECR) images. The downside worth mentioning here is that Inspector2 does not support any Windows operating system as of writing this thesis[3]. Findings are pushed to Amazon Security Hub and Amazon EventBridge, which also helps with automation of reporting and remediating. Inspector2 is integrated with AWS Organizations, contributing to an even better reporting workflow. [37]

Inspector2 is native automated solution which does not require any setup. However, it does not give users any control over it without any options to configure the scans.

In addition to the Inspector2, AWS provides other tools that can help in improving the security posture, such as the Network access analyzer[4] for assessing network access, reachability analyzer[5] for evaluating network reachability or IAM access analyzer[6] analyzing identity access.

2.4.2 Microsoft Defender for Cloud

Microsoft Defender for Cloud is a security solution for Azure with the aim to improve security posture and provide threat protection. Besides security recommendations and alerts, it provides Qualys vulnerability scanner (no license required) for continuous vulnerability assessment and Microsoft Defender for Endpoint for risk-based vulnerability management and assessment. [38]

Qualys solution is quite similar to the Inspector2 in AWS, it requires an extension on the machines, and it is a no setup solution without any configuration. On the other hand, MS Defender for Endpoint is an agent-less solution.

[3] Complete list of currently supported operating systems can be found at `https://docs.aws.amazon.com/inspector/latest/user/supported.html`.

[4] `https://docs.aws.amazon.com/vpc/latest/network-access-analyzer/what-is-vaa.html`

[5] `https://docs.aws.amazon.com/vpc/latest/reachability/what-is-reachability-analyzer.html`

[6] `https://aws.amazon.com/iam/features/analyze-access/`

2.4.3 Other tools

Native tools are not the only vulnerability assessment solutions available. Other tools used to scan on-premise assets, discussed in Section 1.4.6, can also be utilized in the cloud.

2.5 Prioritization

So far, the steps in the cloud were not much different from the classical VM process. The same frameworks for severity and risk scoring can be used for prioritization, such as CVSS or VPR covering the severity scoring and threat context. The following text introduces other possible options for prioritization in the cloud, such as Skybox or Amazon Inspector2. However, prioritization based on asset exposure and importance is relatively new in the cloud and is not covered well by these tools. Thus, the CloneM application was developed to solve this challenge.

Skybox Security claims in [39] that it is integrated with public and private cloud infrastructure platforms. Unfortunately, very little information is publicly available, and since Skybox is a commercial product, it is not easy to evaluate its capabilities regarding prioritization in the cloud.

2.5.1 Amazon Inspector2 Score

Amazon describes the new Inspector2 risk assessment in the documentation [40] as a highly contextualized risk score assessment for each finding, which considers factors such as CVE, network access and exploitability. Amazon Inspector2's score is based on the NVD base score and adjusted to the organization's environment. Its score is in the same format as CVSS, where each finding's severity rating is represented as untriaged, informational, low, medium, high, or critical. Moreover, Inspector2 provides findings based on network reachability, which informs the customer about open ports. However, these findings are only for open ports to the Internet. And since the Inspector2 is only able to detect local vulnerabilities, it cannot prioritize the remote ones based on the open ports (especially to the local network). [40]

Inspector2 uses the NVD/CVSS score for software packages, which is the vulnerability severity score published by the NVD and defined by the CVSS [40]. CVSS was already described in great detail in Section 1.5.1, and the scoring corresponds to the CVSS table. For network reachability scoring, severity is determined by open ports, running services and protocols. These are predefined in Table 2.1.

Even though the documentation states that the risk considers different factors such as network access or exploitability, it is not clear how these factors contributes the final risk score. The documentation does not specify it any further.

Moreover, during the evaluation (described in Chapter 5) vulnerabilities found on the deployed systems had the same Inspector2 score as was their CVSS, even for machines with public IP address! Other issue with the Inspector2 scoring is that the customers do not have any control over it and cannot adjust it according to their needs.

This work aims to address these issues and the CloneM application gives users complete control over the factors contributing to the final risk score. Furthermore, the application provides insight into IAM findings which can contribute to the risk as well.

Service	TCP ports	UDP ports	Internet path rating	Open path rating
DHCP	67, 68, 546, 547	67, 68, 546, 547	Medium	Info
Elasticsearch	9300, 9200	NA	Medium	Info
FTP	21	21	High	Medium
Global catalog LDAP	3268	NA	Medium	Info
Global catalog LDAP over TLS	3269	NA	Medium	Info
HTTP	80	80	Low	Info
HTTPS	443	443	Low	Info
Kerberos	88, 464, 543, 544, 749, 751	88, 464, 749, 750, 751, 752	Medium	Info
LDAP	389	389	Medium	Info
LDAP over TLS	636	NA	Medium	Info
MongoDB	27017, 27018, 27019, 28017	NA	Medium	Info
MySQL	3306	NA	Medium	Info
NetBIOS	137, 139	137, 138	Medium	Info
NFS	111, 2049, 4045, 1110	111, 2049, 4045, 1110	Medium	Info
Oracle	1521, 1630	NA	Medium	Info
PostgreSQL	5432	NA	Medium	Info
Print services	515	NA	High	Info
RDP	3389	3389	Medium	Low
RPC	111, 135, 530	111, 135, 530	Medium	Info
SMB	445	445	Medium	Info
SSH	22	22	Medium	Low
SQL Server	1433	1434	Medium	Info
Syslog	601	514	Medium	Info
Telnet	23	23	High	Medium
WINS	1512, 42	1512, 42	Medium	Info

Table 2.1: Network reachability scoring in Amazon Inspector2 [40].

2.6 Reporting

Reporting can be a bit easier in the cloud. At least the phase of assigning the findings to the right people. It depends on the organization's maturity, but once the member accounts or subscriptions are created, they should be tied to people responsible for deployed resources. Thus, findings should be easily assigned to the account owners, assuming that accounts are not shared between multiple teams.

2.7 Remediation and verification

The last two steps of the VM life cycle in the cloud do not deviate from the classical VM process.

2.8 Identity and Access Management in the cloud

The defense-in-depth approach leverages multi-layer security measures to protect an organization's assets. In an on-premise environment, it is usually performed through network-layer controls [41]. And it is still required in the cloud environment! However, it is not sufficient there. Cloud providers offer many services that need to be secured from the network perspective and from an identity and access management (IAM) point of view.

This section introduces IAM in the cloud and how it works in general. IAM is further discussed in Section 3.9 how it can contribute to the risk assessment of assets.

Every user, application, or service needs some permissions to operate in the cloud. This is where IAM, its policies and roles come in. In the text below, AWS IAM will be taken as an example, and although other vendors might deviate from it a bit, the principles remain the same. The smallest building block for permissions are policies. In its simplest way, policies are defined as a set of permissions (sometimes called statements). Each permission defines:

- its effect, which is either allow or deny,
- list of actions,
- list of resources,
- conditions when it is applicable.

These permissions form a policy, which can be later attached to users or roles.

As roles were mentioned in the previous paragraph, they represent containers for policies. Roles contain permissions described above and something called trust relationships. A trust relationship defines who can use the role (or assume the role), such as the user, other services, or if the trust is set to

an external entity. While the concept of IAM is straightforward, it is challenging to implement regarding the least privilege principle and scaling for organizations.

By default, IAM policies in AWS have deny effect, and users need to add allow statements to allow actions explicitly. Even though there is a default deny, adding explicit deny is a good practice. Permissions within roles are cumulative, which means if one policy does not have some specific action listed while the other policy allows it, the result is that the action is allowed. Nevertheless, if any policy explicitly denies some action, it does not matter on the rest of the policies; they are overridden by the deny statement.

These concepts are easy to understand, but hard to implement in practice. Users will not go through the documentation and spend the day just figuring out, which permissions they need to add in order to set up their workload. Moreover, the need for some permissions can change over time. For this purpose, Amazon provides so-called AWS Managed policies. With their descriptive name and informative descriptions, these policies help users quickly set up what they need. However, they are built in the way to make things work, so usually, they still have too broad permissions!

Achieving the least privilege is a continuous process since the cloud is an ever-evolving environment. Users need to set, monitor, verify and refine permissions to grant the right fine-grained set of them.

2.8.1 Analyzing policies

AWS itself provides tools that can help with analyzing access within IAM policies. IAM Access Analyzer[7] offers a variety of features to set and monitor permissions. There is a feature for validating policies, which checks for syntax errors, missing required elements, constructs, etc. There are over 100 checks for security, which look for bad security practices and patterns in policies [42]. The other one is called policy generation. It can generate a policy template based on the access activity on an entity in some time period. It uses CloudWatch logs to extract this information and then creates the template.

However, a built-in access analyzer is not the only tool out there. Parliament[8] is another excellent open-source linting tool for reviewing IAM policies. It checks for similar things as a policy validator in AWS, such as malformed JSON format or missing actions, and it has its own set of bad policy patterns. As a result, it generates findings that can be later reviewed.

[7] `https://aws.amazon.com/iam/features/analyze-access/`
[8] `https://github.com/duo-labs/parliament`

Chapter 3

Design of the CloneM

Chapter 1 covered vulnerability management in general, essential stages of the VM life cycle, and technical details of vulnerability scanning and prioritizing. Chapter 2 was dedicated to VM in the cloud.

This chapter proposes an application called CloneM capable of prioritizing vulnerabilities in the cloud. Prioritization helps the organization understand the attacker's point of view. The application considers both an outside attacker and an attacker that has already gained access to the network.

3.1 Goals

Whether it be ransomware attacks, data breaches, stealing intellectual property or disrupting the business, the most significant cyber threats are coming from the Internet without any doubt. However, attackers usually do not get direct access to the data as organizations monitor and secure their perimeter. They have to get access to the network first and then move into the network to access the data.

Still, threats are not only coming from the outside Internet. Many organizations outsource some services to third-party providers, which eventually gain access to the company's internal network. Employees of a third party can be vulnerable to social engineering or bribes, and they can be an easier target for the attacker. Nevertheless, internal employees might be a threat as well.

This work focuses on both threats, either coming from the Internet or internal network when an attacker has already gained access and is preparing for lateral movement.

The main goal is to evaluate firewall rules, network access control lists, and routing tables and determine reachability between endpoints based on the evaluation. The focus is on virtual machines running in the cloud.

Based on the possible reachability between virtual machines, the application can update the risk of vulnerabilities found on them to prioritize those with the highest risk. Prioritization includes other factors such as if the endpoint is onboarded into the SIEM program, if it is reachable from the Internet or how critical the system is to the business operations.

A secondary usage of the application is that it can help teams and organizations determine the coverage of scanned devices in the cloud.

3.2 Architecture of the CloneM application

The following section discusses the architecture of the application which is divided into modules. Modules provide functionality for fetching the data, either cloud resources or vulnerabilities and findings, and are responsible for loading them into the database. The core part then is analyzing the data imported to the database. The application uses a graph database called Neo4j, which is described in Section 3.3 in greater detail. The tool aims to be as modular as possible to make it easily extendable in the future. Figure 3.1 shows a simple diagram of the tool's architecture.

Using a graph database came naturally since the network itself is a graph. While there are many graph DBMS available, Neo4j was a choice based on the ranking of DB-engines[9], where Neo4j is all-time number one. There are many available resources for learning, developing and troubleshooting Neo4j and a great community around it. The query language of Neo4j Cypher is also intuitive and easy to get into.

Some data are stored in the document-oriented database MongoDB. The document-oriented database comes in handy since AWS Config output is a structured JSON document that can be directly loaded into MongoDB.

3.3 Neo4j database

Neo4j is a graph database developed by Neo4j, Inc. It is a transactional, ACID-compliant database meant for graph usage, thus having native support for storing the graph data and processing them. Neo4j features two license models. Neo4j Community Edition is fully open source, licensed and distributed under GPL v3. The other one is Neo4j Enterprise Edition, a closed source intended for commercial deployments [44]. Neo4j has an available Neo4j Desktop application, which also includes Neo4j Enterprise Edition for developers.

Neo4j is written in Java and, as such, can run any code that will compile on JVM. That allows users to write their procedures, which can extend the capabilities of the database and Cypher query language. Neo4j can be extended

[9] `https://db-engines.com/en/ranking/graph+dbms`

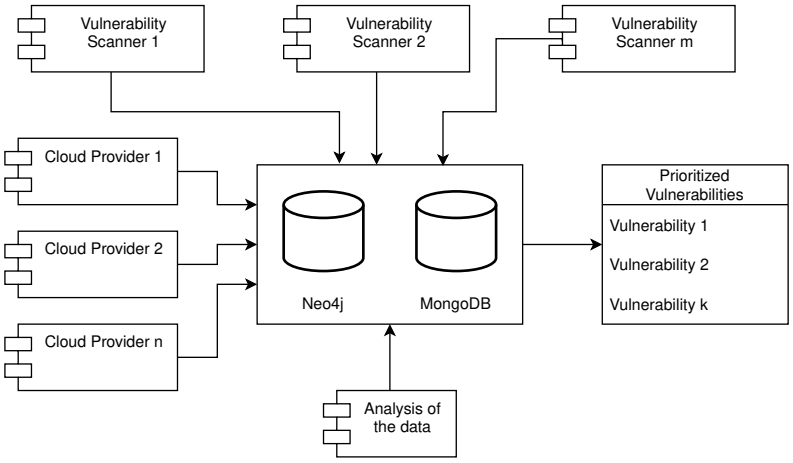

Figure 3.1: Architecture of the application [43, created with Diagrams.net].

in other ways with plugins for authentication or authorization to extend the security framework or with other server extensions. [45]

Neo4j uses its own query language called Cypher. Cypher is mainly inspired by SQL and pattern matching borrowed from SPARQL [46]. It is a powerful query language optimized for graphs. A brief introduction to Cypher can be found in Section 3.3.1.

Neo4j allows for a variety of deployment options. It can run in the cloud, in a cluster, on a personal computer, in a container, or it can be embedded in Java applications.

Information in Neo4j is represented as nodes, relationships and properties. These building blocks make the property data graph model, as shown in Figure 3.2. When building a graph model in Neo4j, nodes can have assigned labels and hold multiple key-value pairs (properties). Nodes labelled with the same label belong to the same set. Labels can also attach metadata to certain nodes, such as indexes or constraints. Similarly, as nodes have labels, relationships between nodes can have types. When working with the database, queries can often work with these node and relationship sets instead of the whole graph, making the execution of the queries much more efficient. While nodes can have multiple labels, relationships can hold up to one relationship type. Relationships can have properties just like the nodes. They are always directed and have a start node and an end node. While the direction of the relationship is mandatory, it can be traversed in both directions. [47]

3.3.1 A brief introduction to Cypher

This section introduces Cypher query language before diving deeper into the modules themselves. As stated earlier in Section 3.3, the Neo4j database used in this project was developed by Neo4j, Inc. The same applies to the

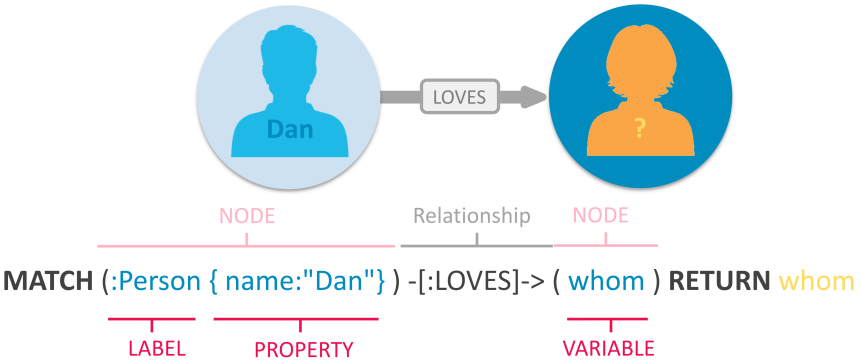

Figure 3.2: Building blocks of the property graph data model [47].

```
(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(johan)-[:KNOWS]->(emil)
```

Listing 3.1: Example of a basic Cypher pattern [48].

Cypher language as well. Note that the other implementations of Cypher query language exist, such as OpenCypher. The following text describes the Cypher implementation made by Neo4j, Inc, and it serves as a high-level overview of the language. For a more comprehensive manual of Cypher, take a look at `https://neo4j.com/docs/cypher-manual/current/`.

Cypher queries are based on patterns, where the user asks the database to match them. As noted in the documentation [48], creating patterns in Cypher is denoted as an ASCII art. Nodes use rounded brackets, while the relationships are expressed in square brackets with an arrow. An example of such a pattern can be seen in Listing 3.1. While Emil, Jim, and Johan represent nodes (for example, with the label Person), relationship (KNOWS) expresses their friendship. The pattern expressed in Listing 3.1 would be drawn as Figure 3.3 on the whiteboard.

3.4 Graph Data Model

The crucial part of the design is the graph data model, which describes how the data are stored and connected. Sometimes, it is referred to as *whiteboard-friendly* since people often draw example data on a whiteboard and connect it to other data while designing the data model. This whiteboard drawing can be put into the graph database exactly as it is [49].

In the case of AWS, resources are represented in the database almost the same as in AWS. Nodes represent resources, and they have an AWS label to distinguish the cloud provider and then a resource label to put the same resources into one set. Most of the information about the resource is then

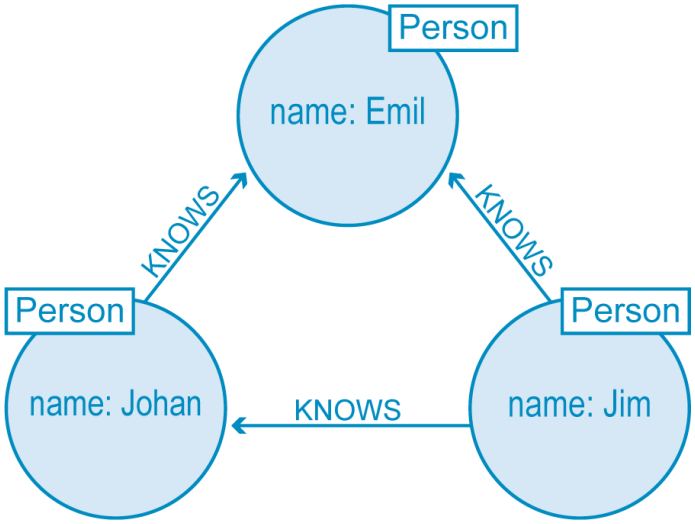

Figure 3.3: Simple friendship data model [48].

kept as properties of the node.

Application Arrows.app[10] was used to create the graph data model. Neo4j develops it to support modelling the graph data models for Neo4j database. It allows exporting the model in different formats like SVG, PNG, JSON, and Cypher query to load the data directly to Neo4j. Since the built graph data model is quite complicated, only some parts are shown.

In some cases, properties are broken down into individual nodes connected to the resource node with a relationship. An example could be AWS Network Access Control Lists (NACLs), which hold information such as NACL id, the default state or region, but they also have the inbound and outbound rules. Instead of being kept as properties, these rules are created as new nodes connected to the NACL by relationship. An example is shown in Figure 3.4.

The graph data model was built with a focus on making the later traversal as easy as possible. It also needs to be able to cover and represent all possible configurations that can be created in AWS. The complete graph data model can be found in Appendix B as an exported JSON file.

3.5 Data synchronization

While this work focuses on prioritizing vulnerabilities, it is crucial to have all the data up to date. In order to solve this, there must be synchronization between the cloud providers, vulnerability scanners and the databases. When the user wants to synchronize the data, the user has to run the module for the selected CSP or vulnerability scanner to download fresh data.

[10] `https://arrows.app/`

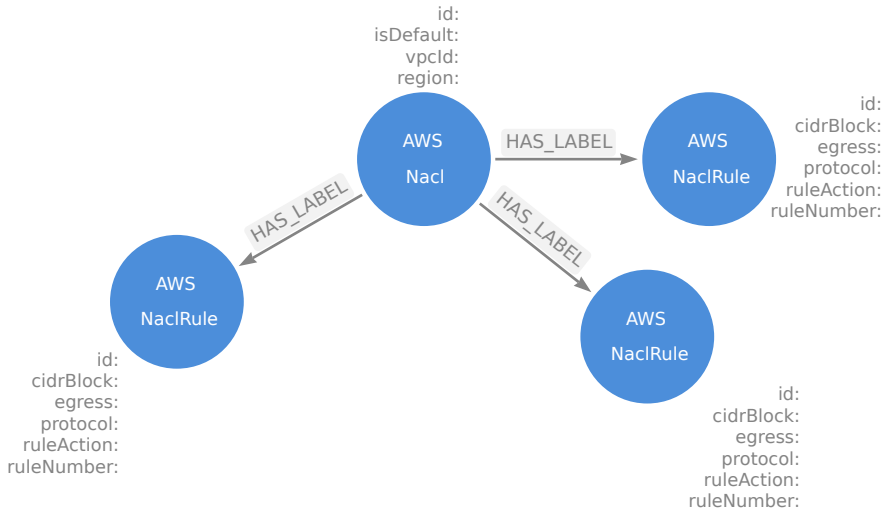

Figure 3.4: Connection between NACL node and its rules [50, created with Arrows.app].

This approach is practical for a certain amount of accounts, regions, and cloud providers. Once the user would like to synchronize hundreds of accounts, multiple environments and regions across the globe, every synchronization could take hours to complete and would make the data immediately out of date. For example, the Cartography tool similarly synchronizes the data, and this issue can be seen and recalled on GitHub issues [51] and [52].

A better approach planned for future work is to have a subscribing process that watches for creating, deleting or modifying events published by the CSP. In the case of AWS, this would be a CloudTrail[11] service, but other vendors have similar services that could be used. This approach would require writing a handler for every such event. The handler would have to create, delete or modify nodes in the Neo4j database or documents MongoDB, create new relationships, or delete the ones not needed anymore in the Neo4j. Such synchronization is beyond the scope of this work.

In the case of vulnerability scanners, downloading the vulnerabilities and findings is much faster and should not take much time. Hence, having full synchronization for them should not be such an issue. Moreover, if it is possible to filter the vulnerabilities upfront based on the assets, data downloaded are limited just to the needed data.

[11] `https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html`

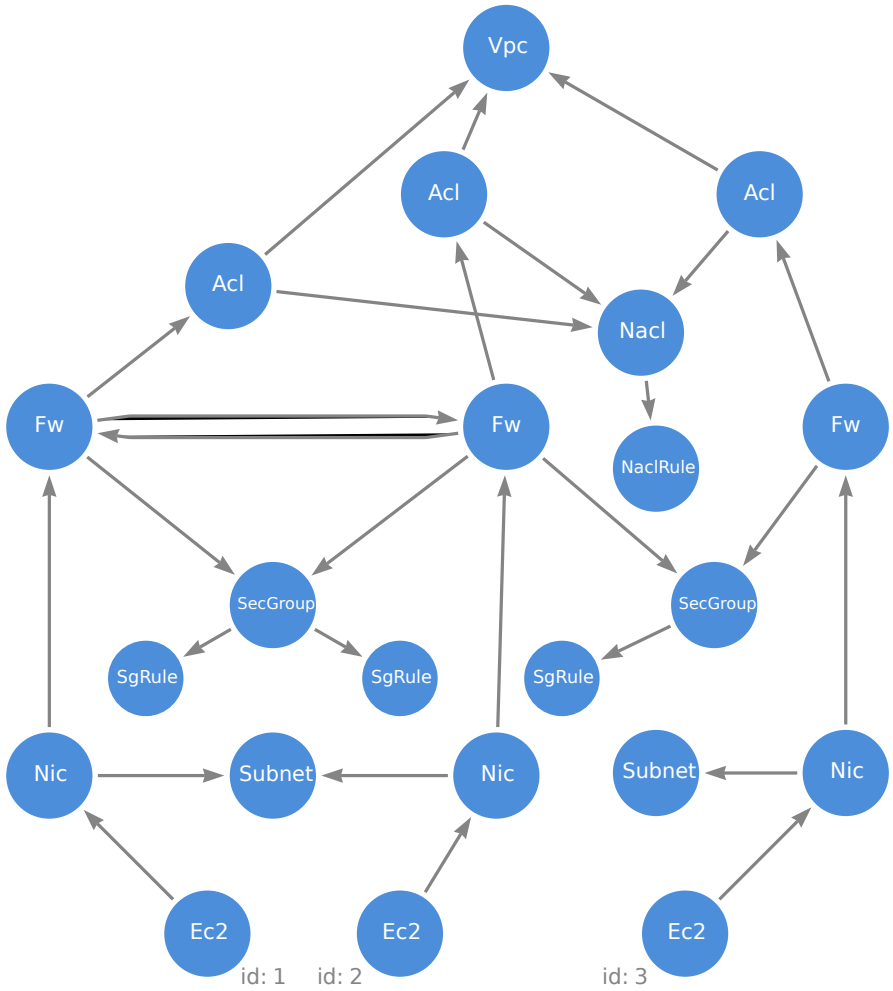

Figure 3.5: Snippet of the graph data model with artificial nodes [50, created with Arrows.app].

3.6 Module for AWS

The module loading the data is closely connected to the graph data model. A few artificial nodes are added to the graph that does not represent any AWS resource to make the traversing easy. These are *FW* and *NetworkAcl*. They are added to every network interface present in the AWS account, as shown in Section 3.5.

This approach makes it easy to represent cases when the network interface is inside multiple security groups or when the security group spans over multiple subnets within VPC. At the same time, it makes it easy to traverse between EC2s within the subnet or outside the subnet with the proper evaluation of network ACLs. It also prevents data duplication as the security group

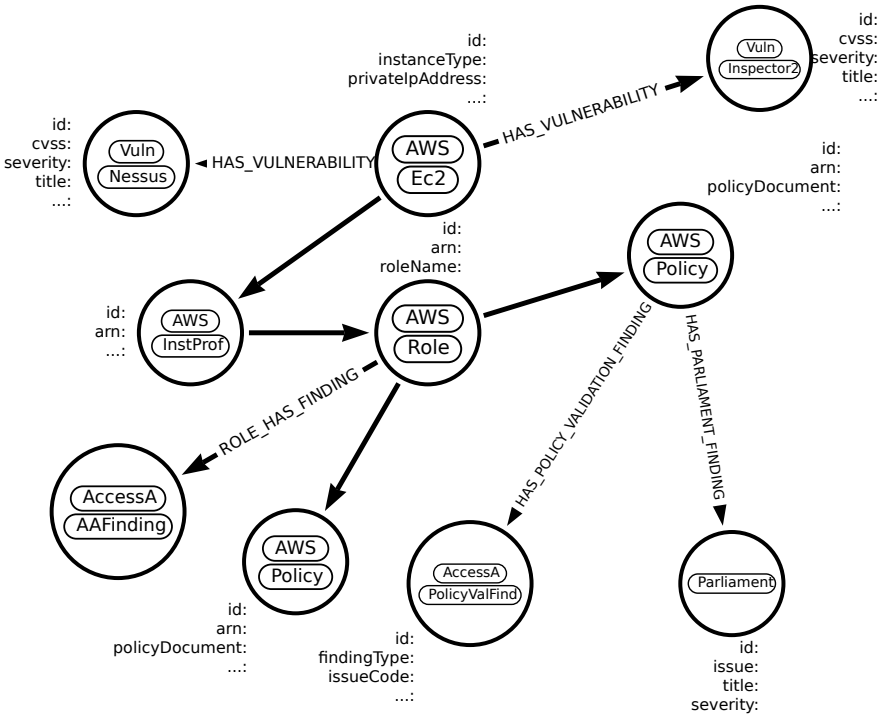

Figure 3.6: Findings connected to their respective nodes [50, created with Arrows.app].

and network ACLs rules are stored only once and referenced by relationships from *Fw* and *NetworkAcl* labelled nodes. Similarly, there is an *Internet* node which can be used for evaluating attacks from the Internet.

3.7 Modules for vulnerabilities and findings

Modules for loading data from vulnerability scanners or tools that look for IAM potential insecure configurations are relatively simple. They only need to retrieve the data from the respective source and connect them in the graph to the correct node. An example can be seen in Figure 3.6, where vulnerabilities are connected to the EC2 instance, Parliament and access analyzer findings to policies and roles.

3.8 Evaluating the network

Once all the data are loaded into Neo4j, the next step is to analyze the data. The first level on which the data can be analyzed is the network level. Evaluating all the routing tables, firewall rules, security groups (for some platforms known as network security groups), network access control lists, and other

security appliances on the network can help determine reachability between endpoints. This could reveal the exposure of some critical vulnerabilities to the rest of the network.

Evaluating the network can be done by traversing the graph and checking the rules. An example of such traversal can be seen in Figure 3.7. Considering the starting node is an EC2 instance with id 2, red arrows display the algorithm's path to reach the other two EC2 instances. Note that as mentioned in Section 3.3, the relationships can be traversed in both directions. Green arrows then show paths to check on rules of security groups and network ACLs.

3.9 Evaluating identity access

The second level on which the graph can be analyzed is based on identity access and management. Unlike the network evaluation, where it is clear and easily determined what should be allowed on the network, evaluating permissions and identity access is much more difficult. There might be scenarios where single permission will not cause any harm but, combined with other permissions, can let the attacker hop on other services, deploy malicious code, etc. Also, it is not rare that some services themselves require more permissions than they would need for their operation. Yet this gives the user no other choice than assign the required permissions.

Given these conditions, it is much more challenging to automate the prioritization of findings and vulnerabilities based on IAM. The application provides all the data connected together and lets an analyst review the findings.

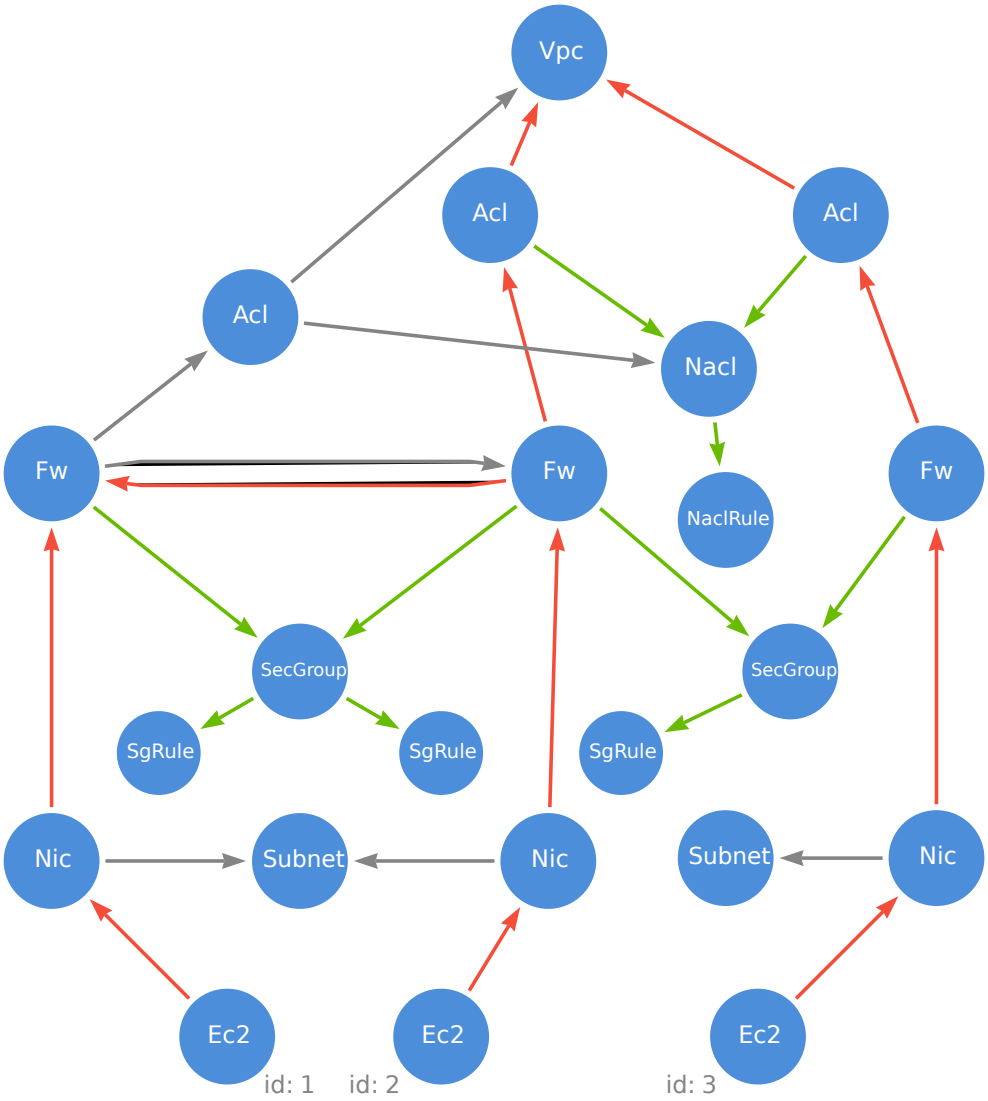

Figure 3.7: Traversal example [50, created with Arrows.app].

Chapter 4

Implementation of the CloneM

In Chapter 3, the tool's design was discussed. This chapter is dedicated to implementation details. The steps needed to extend the CloneM application are described at the end of the chapter.

For the development, the Python language was used. Python is easy to use, fast to develop and enables fast application prototyping. AWS has an extensive SDK for Python known as boto3[12].

As mentioned in Section 3.2, the application is divided into modules, which all form the backend of the app. Modules can be separated into three categories:

- modules for cloud providers (or any provider of the infrastructure)
- modules for vulnerability scanners (or any tool that provides findings)
- modules for analyzing the data in the database

Class `CLI` is the main class to execute the application, parse the arguments and set up the environment for the application. Together with the `Sync` class, it handles stages executed in the current run. The function `run` from the `Sync` class then executes each module. A code snippet of this function can be seen in Listing 4.1.

4.1 Module for AWS

The following paragraphs introduce `aws` module responsible for retrieving data from AWS. In the beginning, the module creates a boto3 session, deletes all data in the database labelled AWS and then the module pulls fresh data from AWS. Concerning synchronization, more details can be found in Section 3.5.

[12] `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`

```
for stage_name, stage_func in self._stages.items():
    try:
        self.logger.info(f'Executing stage: {stage_name}')
        stage_func(self._conn, config, self._log_level)
    except Exception:
        self.logger.exception(f'Unhandled exception during syncing')
        pass
```

Listing 4.1: Run function for executing modules.

```
conf = AWSConfig()
for func, expression in conf.expressions.items():
    self.logger.debug(expression)
    data = config_client.select_resource_config(
                Expression=expression
            )
    func(self, data)
```

Listing 4.2: A snippet of the code to retrieve data via AWS Config.

The key role in the `aws` module is the Config service[13] in AWS, which allows querying the current configuration state of AWS resources. This module heavily depends on it. Since the queries can be written in a (limited) SQL, it makes it easy to retrieve the data and, most importantly, reduces the codebase needed for pulling the data. The code snippet responsible for that operation can be seen in Listing 4.2. It goes through items where each item consists of the query and the corresponding function. Once the data are retrieved from Config, the corresponding function is called, and the data are passed as an argument. Supporting class `AWSConfig` is used to store all the queries and function names responsible for handling the data for each resource.

Despite the fact that AWS Config is used for most queries, it is not capable of retrieving all AWS resources. In those cases, boto3 client for the respective resources is used.

Once the appropriate function is called, it processes the data and loads them to the Neo4j database. In order to do that, every function needs to implement its Cypher query based on the resource the query handles. For some resources, it is fairly straightforward, but some queries are a bit more complicated.

An example of such a complex query can be a case for routing tables. The complete query is shown in Listing 4.3. The routing table holds metadata about itself, such as the table's id, the VPC's id to which it belongs, and then routes and associations of the routing table. The first part of the query

[13] `https://docs.aws.amazon.com/config/latest/developerguide/WhatIsConfig.html`

```
query = """
UNWIND $rts as rt
MERGE (new_rt:
""" + f'{nodes_relationships.nodes.get("aws")}:
    {nodes_relationships.nodes.get("routeTable")}' + """
{id: rt.configuration.routeTableId } )
SET
    new_rt.rtid = rt.configuration.routeTableId,
    new_rt.region = rt.awsRegion
WITH new_rt, rt
MATCH (vpc:
""" + f'{nodes_relationships.nodes.get("aws")}:
    {nodes_relationships.nodes.get("vpc")}' + """
 {id: rt.configuration.vpcId } )
MERGE (vpc)-[r:
""" +
f'{nodes_relationships.relationships.get("hasRouteTable")}' +
"""
]->(new_rt)
WITH rt, new_rt
UNWIND rt.configuration.associations as rtbassoc
MERGE (new_rtbassoc:
""" + f'{nodes_relationships.nodes.get("aws")}:
    {nodes_relationships.nodes.get("routeTableAssociation")}' + """
{id: rtbassoc.routeTableAssociationId })
SET
    new_rtbassoc.main = rtbassoc.main,
    new_rtbassoc.association_state = rtbassoc.associationState.state
MERGE (new_rt)-[r2:
""" +
f'{nodes_relationships.relationships.get("rtHasAssoc")}' +
"""
]->(new_rtbassoc)
WITH rtbassoc, new_rtbassoc
MATCH (subnet:
""" + f'{nodes_relationships.nodes.get("aws")}:
    {nodes_relationships.nodes.get("subnet")}' + """
{id: rtbassoc.subnetId } )
MERGE (subnet)-[r3:
""" +
f'{nodes_relationships.relationships.get("subnetHasRtAssoc")}' +
"""
]->(new_rtbassoc)"""
```

Listing 4.3: Cypher query to load routing tables and all their components.

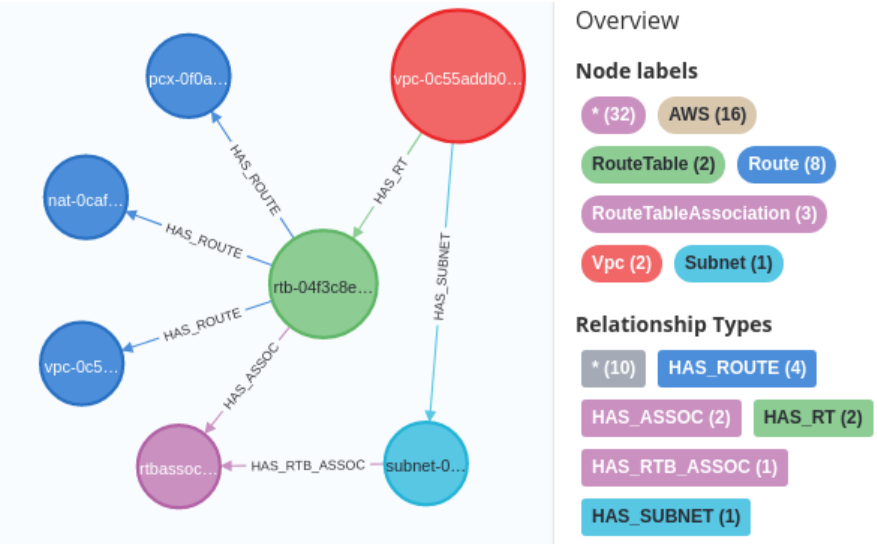

Figure 4.1: Routing table with its relationships [53, taken in Neo4j Browser].

expands the list of routing tables into a sequence of rows with `UNWIND` clause. Routing table nodes are created and connected to the appropriate VPC.

Associations give information about which subnet is associated with which routing table and if it is the main routing table of the VPC. Hence, the query for loading the routing table must take associations into account and associate the routing table with appropriate subnets. The next part of the query is responsible for that. Query expands the list of associations for each routing table, creates a node per association, and connects it to the routing table. The created node then connects the association to the correct subnet.

Another query creates nodes for each route. Figure 4.1 shows the visualization of the result. Other resources are loaded similarly.

4.1.1 Supported resources

The main focus was on the core infrastructure of the network, EC2 instances and IAM roles and related policies. Supported resources include:

- VPCs
- Subnets
- Routing tables (and their associations)
- Internet gateways
- NATs
- Transit gateways

- Peering connections
- Network ACLs (and their associations)
- Security groups
- Network interfaces
- EC2 instances
- IAM policies
- IAM roles
- Tags

However, not all possible configurations are supported. For instance, current state of the application does not support prefix lists for routing tables and security groups.

4.2 Modules for vulnerabilities and findings

Multiple modules were implemented in order to retrieve vulnerabilities and findings. Vulnerabilities are getting assigned CVEs. On the other hand, findings do not have CVEs and typically are misconfigurations. Vulnerabilities are retrieved via module that connects to Tenable.io[14]. AWS native tool Inspector2 is another source of vulnerabilities. For IAM findings, the built-in tool AWS Access Analyzer was used to retrieve IAM findings. Open-source tool Parliament is implemented as well.

Tenable.io

Both Nessus scanners and agents are allowed to connect to Tenable.io. This makes it easy to obtain data from one source while having both types of scans (remote and local).

The module uses the pyTenable library[15], the official Python library developed by Tenable, Inc. Tenable.io is a quite evolving product, so the API changes a lot. The pyTenable library is usually a little behind and does not implement the latest API.

The module is quite simple. It retrieves the list of assets and vulnerabilities and pairs the vulnerabilities with corresponding assets based on the asset id (that is internal to Tenable.io). Once it is done, the Cypher query will match the resulting assets with virtual machines present in the database and load the vulnerabilities with the relationship to the corresponding machine.

[14] `https://www.tenable.com/products/tenable-io`
[15] `https://pypi.org/project/pyTenable/`

Tenable.io aggregates data from multiple accounts. It also supports other cloud providers such as Microsoft Azure or Google Cloud Platform.

AWS Inspector2

AWS Inspector2 is an agent-based scanner, so it can only find local vulnerabilities. The module implementation is very straightforward. This time module uses the boto3 library to retrieve the data from AWS. If the configuration specifies it, the module can also assume other accounts and pull the data from them. Once the data are retrieved, they are loaded into the Neo4j database and vulnerabilities and findings are connected to the respective EC2 instances.

AWS Access Analyzer

AWS Access Analyzer was discussed before in Section 2.8.1. It serves the purpose of checking harmful patterns within IAM policies and access permissions. The module is divided into two core functions, one being `validate_policy` and the other one `retrieve_findings`.

`Validate_policy` retrieves the policies from MongoDB and checks them with the function `validate_policy` provided by the boto3 client for the access analyzer. As a result, if any findings are found, they are loaded into Neo4j and attached to the corresponding policy.

Function `retrieve_findings` retrieves findings via function `list_findings` available for boto3 client for access analyzer. These findings are attached to their corresponding roles in the next step.

Parliament

The last module implemented for checking IAM policies uses the open-source library parliament, already introduced in Section 2.8.1. Parliament is available as a library and can be installed via `pip install parliament`. Parliament stands out because it is great for customization, where users can define their own checks, override the severity of some findings, or skip certain types altogether.

Similar to the `validate_policy` function in Section 4.2, the parliament module retrieves policies from MongoDB and checks them against its checks. Findings are loaded into the Neo4j in the following step.

4.3 Evaluating the data

Python is also the language of choice for evaluating the data. Using Python means that the application needs to query the database server over the network every time it needs the next node. This approach is highly inefficient in terms of performance. On the other hand, the data model can change significantly

during its development, and these changes to the model might require changes to the analyzing algorithm.

The module for evaluating the network is probably the most complicated one. Its goal is to find paths between endpoints and evaluate vulnerabilities on the targeted node if the path exists. Evaluation starts on the node based on the initial configuration. The user has to provide either id of the resource or a valid IP address which tells the algorithm from which node it should start traversing the graph.

Every label has predefined possible options, where the algorithm can go in the next step. For labels such as *Fw* or *NetworkAcl*, the algorithm evaluates their security group rules and network ACL rules, respectively. If the rules permit it, the algorithm continues deeper in the graph. Otherwise, the algorithm returns.

When the target is reached, the algorithm looks up vulnerabilities present on the node. If there is a match between target (which consists of a combination of protocol and port) and a vulnerability's protocol and port, the risk might be increased for such vulnerability. Note that the path is traversed only in one direction. However, the user can run the application in the other direction to discover if the path exists in both.

Nevertheless, other factors might have an impact on risk as well. These include tags on an EC2 whether the instance is critical to business operations, onboarded to a SIEM, or any other user-defined tags.

Finally, all the traversed data are copied over to the second instance of the Neo4j database together with all the exposed vulnerabilities, IAM roles, policies and their respective findings. Later, the analyst can review the copied data, making it much easier to look at it without any noise and disturbance.

Section 4.1.1 discussed supported resources. However, not all possible configurations are supported for traversing. For instance, traversing the graph currently supports only IPv4.

4.4 Steps to extend the project

This last section of the implementation chapter will shortly discuss the steps needed to extend the project. Adding modules for retrieving data, either from a cloud provider or another vulnerability scanner, is quite straightforward. There must be added a parameter to the command line arguments or a section in a configuration file so that the program knows which stage to add to the current run.

Every module is supplied with a connection to the Neo4j instance, a configuration file and a logging level. The module can make use of them or not. Since most of the data are stored in Neo4j, it is highly recommended to use at least the Neo4j connection unless there are specific requirements. As discussed

in Section 5.4, the database engine is currently Neo4j, but this requirement could be lifted in the future.

The evaluation stage requires most of the work. It needs to support new nodes, their labels and properties, and relationships and handle them accordingly. In Section 5.4 is discussed how this could be improved to make it more pleasant to extend.

Chapter 5

Evaluation, use-cases and testing of the CloneM

The following chapter is focused on testing and presenting the capabilities of the developed application. The first part describes the testing environment and setup. In the second, the actual testing of the application and results are shown.

5.1 Setup

In order to create a testing environment and repeatable tests, Terraform tool was utilized. Terraform[16] is a so-called Infrastructure as Code (IaC) tool developed by HashiCorp Inc. IaC automates the provisioning of the infrastructure in a repeatable manner and enables versioning and testing of the infrastructure. Tools such as tfsec[17] can check created templates and spot potential misconfigurations and security holes.

Terraform is not the only tool available on the market. There are plenty of other options, such as CloudFormation for AWS, Azure Resource Manager for Azure or Google Cloud Deployment Manager for GCP.

The AWS setup

A few things need to be set up and enabled on the AWS side. AWS Config has to be enabled in the account to retrieve most of the data. For the Inspector2 module, the Inspector2 service needs to be enabled. Moreover, AWS must have a created resource access analyzer for the Access Analyzer module. All the mentioned services are on a per-region basis, which means that users who want to retrieve data must enable these services per region.

[16] `https://www.terraform.io`
[17] `https://github.com/aquasecurity/tfsec`

The second part of the AWS setup is to have a user/role with permissions to retrieve the data. In order to pull data from AWS from one account, the application must be supplied with the credential that has permissions for the AWS Config service and IAM ListPolicies. Users can use AWSServiceRoleForConfig or AWS_ConfigRole, AWS managed policies defined in the documentation AWSManagedPolicies [54]. Either of them gives enough permissions to access AWS resources, except the permission iam:ListPolicies. As mentioned in Section 4.1, the AWS Config service itself is not capable of listing IAM policies. Hence the permission must be supplied separately.

For Inspector2, users can use AWS managed policy called *AmazonInspector2ReadOnlyAccess*, which gives read permission over all Inspector2 findings. There is no other configuration needed for Inspector2. However, it is worth noting here that the EC2 machines the user wants to include in the scanning must have installed and configured the SSM agent (AWS Systems Manager Agent).

Similarly, AWS managed policy called *IAMAccessAnalyzerReadOnlyAccess* is recommended for Access Analyzer. Notice that the Parliament module is a Python library with built-in checks. There are no calls to the AWS, and thus, Parliament does not need any extra permission.

The database setup

The application uses two databases. The first is Neo4j, the second MongoDB. For the testing, the Neo4j Desktop application was installed on a testing machine, which includes an Enterprise license for developers. This is useful since the application uses one database to store the data and another one to display the analysis result, in total, two databases within one DBMS. An Enterprise license is required to use multiple databases within one DBMS.

However, the application expects the databases to be created beforehand. Thus, the first and default database created was *neo4j*, the second *traversal*.

Another advantage of the Neo4j Desktop is the ease of setup and installation. All the required software (like Java) is bundled with the application. Furthermore, tools such as Neo4j Browser and Neo4j Bloom for querying the database and visualizing the data are part of it as well.

MongoDB was set up on a testing machine as a Docker container. Reference of the setup can be found in Appendix D.

The Nessus setup

Concerning making use of the Tenable.io module, the scanner of choice had to be Nessus. Nessus scanner was used for remote scanning, and Nessus agents were installed on the target machines for local scans, which all were linked to the Tenable.io instance. Tenable.io is a cloud-based vulnerability management

solution that collects data from various environments such as on-premises, AWS, Azure or GCP.

Nessus scanner can be easily installed on a Linux machine with the command:

```
curl -H 'X-Key: your-linking-key'
'https://cloud.tenable.com/install/scanner?name=scanner-name'
| bash}
```

The user has to provide a valid linking key, which can be retrieved via the Tenable.io web interface. Other optional parameters can be set, such as scanner group or network. Installed version of Nessus scanner for testing was 10.1.2 with plugin set 202204231148.

Nessus agent can be installed on Linux in a similar way:

```
curl -H 'X-Key: your-linking-key'
'https://cloud.tenable.com/install/agent?name=agent-name'
| bash
```

The requirement to provide a valid linking key applies here as well. Other parameters like agent groups or network can be provided. For installing Nessus on Windows, refer to the documentation for an agent[18] or scanner[19]. Version of all Nessus agents was 10.1.3 with plugin set 202204231148. When the valid linking key is provided, Nessus scanners and agents are connected to the desired Tenable.io instance, and data can be collected from them.

The infrastructure setup

A few Terraform templates were developed to build an infrastructure and to speed up the process of testing. Deployment was divided into two parts. One is responsible for building an underlying infrastructure, the other one for deploying vulnerable machines and configuring different security group rules and network access control lists.

Terraform template responsible for the deployment of the infrastructure is called *core*. It creates IAM roles necessary for connecting and managing both Windows and Linux machines via AWS Systems Manager (formerly known as AWS SSM). Then it creates two VPCs. The first one, *vpc-core*, has two subnets, one being public, the other one private. The internet gateway (IGW) and NAT gateway are deployed inside the VPC. Subsequently, an EC2 machine with the Red Hat Enterprise Linux (RHEL) operating system inside the private subnet is deployed with an installed Nessus scanner. The second VPC, *vpc-testing*, is connected with the first one over the peering connection. Appropriate routes are added to both to maintain routing between them. No

[18] https://docs.tenable.com/nessus/Content/InstallNessusAgentWindows.htm
[19] https://docs.tenable.com/nessus/Content/InstallNessusWindows.htm

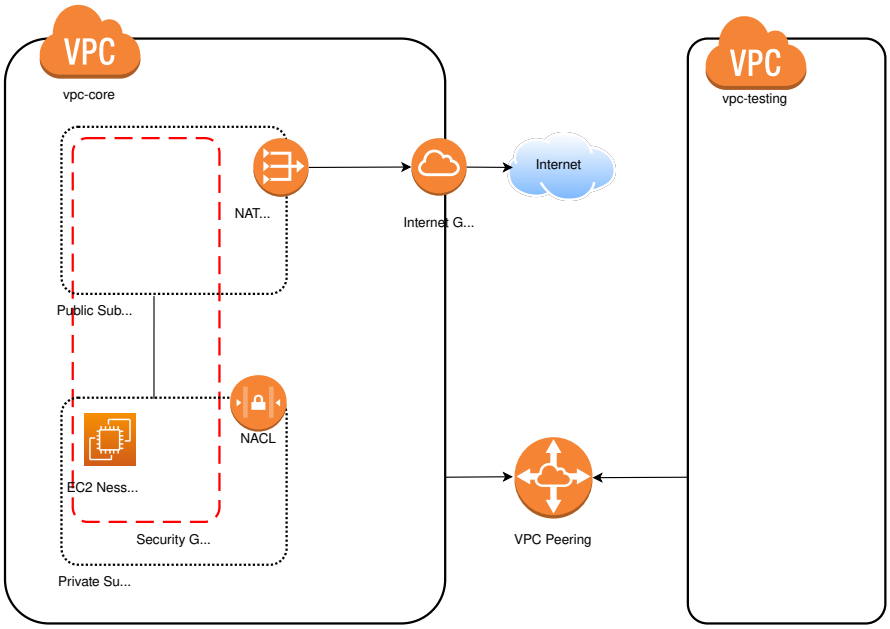

Figure 5.1: Deployed infrastructure with Terraform template [43, created with Diagrams .net].

other resources are deployed in the second VPC by this template. The infrastructure deployed by this template is depicted in Figure 5.1. More details about the template can be found in Appendix D.

Machines setup

Different machines were set up in different ways. Some were configured directly via Terraform, some manually, and some were imported from various templates and sources. In this section, configured machines will be briefly introduced.

Following Windows machine was created manually. As a base image was used Windows Server 2016 with public AMI *ami-0f16f67df3939369c*. On top of it was installed XAMPP[20], which is an Apache distribution containing MariaDB, PHP, and Perl, and it was loaded with the vulnerable application downloaded from GitHub[21]. After the setup was done, a new custom AMI was created from this machine using the built-in feature to do so. This machine is referred to with AMI *ami-0bbe3b1dc84a92226*. In the same fashion was created another Windows machine. Based on the Windows Server 2016, AMI *ami-0f16f67df3939369c* and with XAMPP installed. This time with installed Damn Vulnerable Web Application[22] (DVWA).

[20] `https://www.apachefriends.org/index.html`
[21] `https://github.com/ShinDarth/sql-injection-demo/`
[22] `https://github.com/digininja/DVWA`

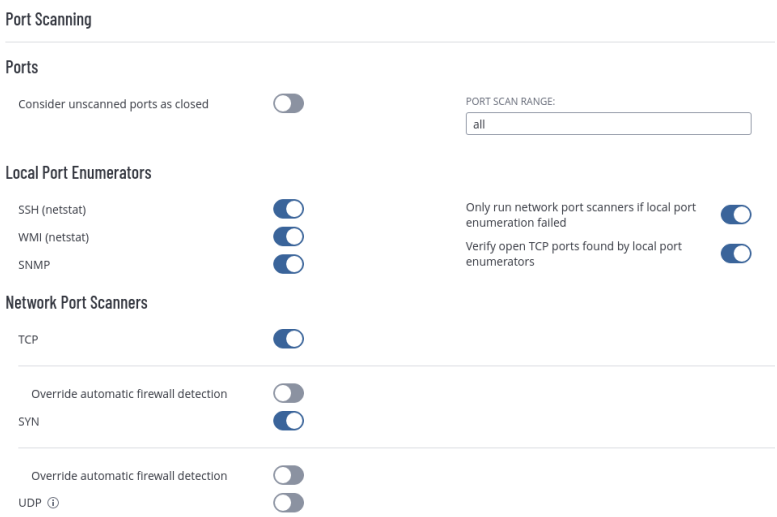

Figure 5.2: Nessus remote scan setup for port scanning [56, taken in Tenable.io].

Some images were created from templates publicly available and imported to AWS. AWS allows importing VMs in various formats and creating images from them. More about importing can be found in the documentation[23]. Web Security Dojo is an open-source training environment for web application penetration testing [55]. Different tools and vulnerable applications are provided in the image, which is freely available as an OVA template to download from `https://sourceforge.net/projects/websecuritydojo/`. This template was imported into AWS and is referred to with AMI *ami-03bc8cab6d501e60c*.

The rest of the machines were configured directly with Terraform templates. Either RHEL 7 or RHEL 8 were used as a base OS. In most cases, the webserver and the database were installed on top of them. Installation scripts are also referenced in Appendix D.

The scan setup

This section briefly introduces scan setups in Tenable.io for both traditional remote scan and agent scans. The complete configuration of the scans can be found in Appendix D. Figures 5.2 and 5.3 depict some of the most important settings of the remote scan. Agent scans are configured similarly.

[23] `https://docs.aws.amazon.com/vm-import/latest/userguide/vmimport-image-import.html)`

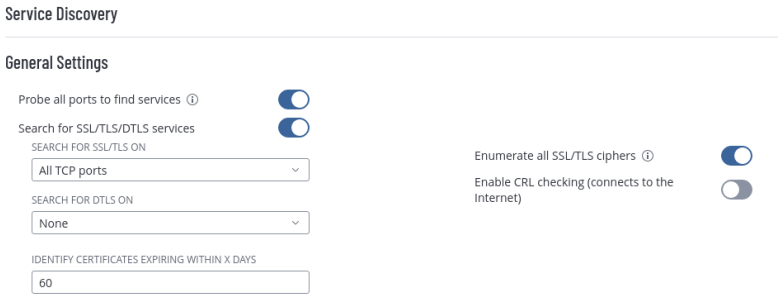

Figure 5.3: Nessus remote scan setup for service discovery [56, taken in Tenable .io].

5.2 Testing

Another Terraform template was created to deploy infrastructure into the VPC designated for testing. The template deploys one public subnet and two private subnets. A few EC2s are deployed in each of the private subnets. The whole infrastructure is depicted in the Figure 5.4, and even though the infrastructure is not complicated, the graph can get dense and hard to read quickly.

The most interesting part of the setup is security groups and network access control lists since they restrict the network-level access to resources. During the testing, their rules were changed to permit or deny access to test different scenarios.

After the infrastructure had been deployed, Nessus scans were run on the hosts to find local or remote vulnerabilities. SSM agent was installed on all machines together with the attached instance profile with required permissions to collect findings and vulnerabilities found by Inspector2. The next step was to collect the data and load them to the Neo4j and MongoDB databases.

Important disclaimer, once the machines were scanned for vulnerabilities, they were stopped to reduce the cost and to actually prevent exploitation from the real world attackers (since some machines were exposed to the Internet). Then, tweaks were made to the NICs, security groups, and NACLs to simulate different scenarios.

With all the data in the DBs, it was time to traverse the graph from different starting points to find vulnerable machines. Together with the results, all the configurations are referenced in Appendix D.

Test 1

In the first test scenario, the attacker was assumed to be outside the network. Particularly there was one EC2 with public IP address with security group allowing outside access. However, the risk calculator was set up and would

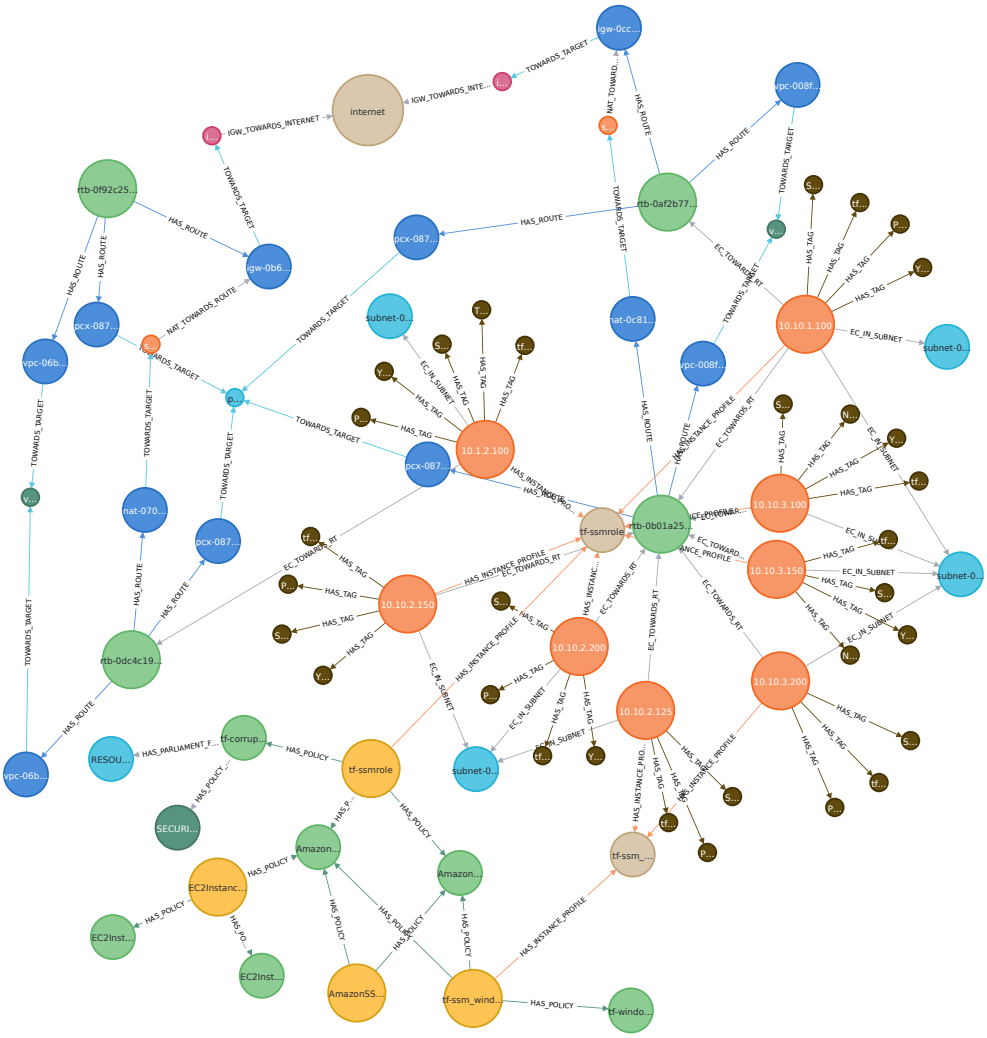

Figure 5.4: Simplified picture of the deployed infrastructure [53, taken in Neo4j Browser].

increase risk on all vulnerabilities with CVSSv3 above 6 on production machines, only one low severity vulnerability was found, thus the final risk score remained the same.

Test 2

The second scenario simulates the attacker that is already inside the network. All machines are reachable within the 10.0.0.0/8 subnet with no other restrictions.

Test 3

The third test simulates an inside attacker similar to the second test scenario. This time, with restrictions on security group and NACL level. Furthermore, the risk is lowered on vulnerabilities found in non-production machines.

5.3 Discussion and outcomes

During the testing period, some pitfalls came concerning selected technologies. On the AWS side, the Config service allowed easier data retrieval with only a few lines of code and SQL queries. On the other hand, Config does not support all resources in AWS. That could be seen in the case of IAM policies, which need to be retrieved via boto3 iam client. Second, Config is not 100% reliable regarding the returned results. Over the testing period, the Config service did not return Network ACL Associations with subnets, even though these associations could be seen in the web console or retrieved via boto3 afterwards. This prevents associating Network ACLs with the subnets and the right resources inside the subnets. This kind of error is hard to detect since there could be Network ACLs without any association.

Neo4j is great for connecting the data via its relationships, allowing insight into the data. However, it is a shame that it does not allow storing more complex data structures in nodes (or even relationships). This could be seen with IAM policies stored in MongoDB as they are complex documents. Moreover, it would take much more effort to break them into simple key-value pairs to store them in Neo4j.

In Neo4j 4.4.3, another missing feature is that Neo4j does not support generating any UUID for nodes.

5.4 Further technical improvements

The application uses the Neo4j database, but there are multiple options that support Cypher query language (more precisely, openCypher); hence this requirement could be lifted, and any database that supports openCypher could be used.

The algorithm to traverse the graph was written in Python, which was great during development. Especially when the data model was changing quite often, the code could be updated quickly. Nevertheless, it also came with the downside of the performance penalty. Once the data model is not expected to change much, the algorithm should be reimplemented in Java (or any JVM supported language) since the Neo4j database can run JVM code natively. It would remove the need to query the database over the network and thus remove this performance hit. Overall this could pose a significant performance improvement.

Not only the performance would be advantageous, but also an improved design for evaluating the graph could be beneficially chained together with normalized data. In the current stage, data are not normalized, and nodes are always specific to the cloud provider or scanner vendor.

The output of the traversed graph can point out remotely found vulnerabilities. However, it does not return the rules that allowed the connection on its path. This would be helpful for later analysis.

Conclusion

The main goal of this thesis was to analyze the current state of vulnerability prioritization in the cloud and implement a tool that would help vulnerability management teams with it. The general vulnerability management process was discussed with its specifics in the cloud. The application design was proposed, and the built prototype was evaluated on sample data and deployed in the corporate environment.

The goals of the thesis were met. The CloneM application was developed in Python and deployed on a few AWS accounts. I suggest methods how the application could be used to reduce the costs of the organization's prioritization process within the vulnerability management. It could save the human resources needed to analyze the vulnerabilities manually and help them to be more efficient.

The CloneM application could serve as a great starting point for further development so that it can accommodate other services within AWS, other cloud service providers and vulnerability scanners.

Future work

The CloneM application has many possibilities to be enhanced. It could add support for other cloud providers such as Microsoft Azure, Google Cloud Platform, OpenStack, DigitalOcean, Oracle Cloud, Linode, or any other. The CloneM application should be thoroughly evaluated in the organization. Lastly, technical improvements are discussed in Section 5.4.

Literature

1. *NVD - Search and Statistics* [online] [visited on 2022-05-01]. Available from: https://nvd.nist.gov/vuln/search.
2. *ISO/IEC/IEEE 15288:2015* [online] [visited on 2022-04-09]. Available from: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/37/63711.html.
3. ROSS, Ron; MCEVILLEY, Michael; CARRIER OREN, Janet. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems* [online]. 2016-11 [visited on 2022-02-25]. NIST SP 800-160. National Institute of Standards and Technology. Available from DOI: 10.6028/NIST.SP.800-160.
4. *Committee on National Security Systems (CNSS) Glossary* [online]. 2022 [visited on 2022-04-09]. Available from: https://www.cnss.gov/CNSS/openDoc.cfm?MoQwsz3tuBdKgIomP5vj+w==.
5. *ISO/IEC 27039:2015* [online] [visited on 2022-04-09]. Available from: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/68/56889.html.
6. EDITOR, CSRC Content. *Risk - Glossary | CSRC* [online] [visited on 2022-02-25]. Available from: https://csrc.nist.gov/glossary/term/risk.
7. *Act No. 181/2014 Sb. - Zákon o Kybernetické Bezpečnosti a o Změně Souvisejících Zákonů (Zákon o Kybernetické Bezpečnosti)* [online]. 2014 [visited on 2022-02-25]. Available from: https://nukib.cz/images/icons/2021-08-31_novelizace_zneni_zakona_181_2014.pdf.
8. EDITOR, CSRC Content. *Computer Security Incident - Glossary | CSRC* [online] [visited on 2022-02-25]. Available from: https://csrc.nist.gov/glossary/term/Computer_Security_Incident.

9. STONE, Michael; IRRECHUKWU, Chinedum; PERPER, Harry; WYNNE, Devin; KAUFFMAN, Leah. *IT Asset Management: Financial Services* [online]. Gaithersburg, MD, 2018-09 [visited on 2022-04-10]. NIST SP 1800-5. National Institute of Standards and Technology. Available from DOI: 10.6028/NIST.SP.1800-5.

10. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1* [online]. Gaithersburg, MD, 2018-04-16 [visited on 2022-02-18]. NIST CSWP 04162018. National Institute of Standards and Technology. Available from DOI: 10.6028/NIST.CSWP.04162018.

11. *ISO/IEC27000: Asset Management* [online]. 2016-06-20 [visited on 2022-05-02]. Available from: https://www.tenable.com/sc-dashboards/isoiec27000-asset-management.

12. *CIS Controls Version 8* [online] [visited on 2022-05-02]. Available from: https://www.cisecurity.org/controls/v8/.

13. ATLASSIAN. *Asset & Configuration Management in Jira Service Management* [online] [visited on 2022-04-13]. Available from: https://www.atlassian.com/software/jira/service-management/product-guide/getting-started/asset-and-configuration-management.

14. EDITOR, CSRC Content. *Vulnerability Scanning - Glossary | CSRC* [online] [visited on 2022-02-18]. Available from: https://csrc.nist.gov/glossary/term/Vulnerability_Scanning.

15. *Vulnerability Scanner: What Is It and How Does It Work?* [Online] [visited on 2022-05-01]. Available from: https://snyk.io/learn/vulnerability-scanner-2/.

16. *Traditional Active Scans (Non-credentialed) (Nessus Agents)* [online] [visited on 2022-03-25]. Available from: https://docs.tenable.com/nessusagent/Content/TraditionalScansUncredentialed.htm.

17. *Benefits and Limitations (Nessus Agents)* [online] [visited on 2022-02-27]. Available from: https://docs.tenable.com/nessusagent/Content/BenefitsAndLimitations.htm.

18. *PCI DSS Quick Reference Guide* [online]. 2018-06 [visited on 2022-02-18]. Available from: https://www.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf.

19. *Vulnerability Scanning Frequency Best Practices* [online] [visited on 2022-02-27]. Available from: https://thehackernews.com/2021/12/vulnerability-scanning-frequency-best.html.

20. KENNA. *Prioritization to Prediction* [online] [visited on 2022-02-26]. Available from: https://website.kennasecurity.com/wp-content/uploads/2020/09/Kenna_Prioritization_to_Prediction_Vol1.pdf.

21. GMBH, CodeShield. *Automated Vulnerability Prioritization in the Context of the Cloud* [online]. 2021-08-04 [visited on 2022-04-24]. Available from: https://codeshield.io/blog/2021/08/04/automated_vulnerability_prioritization/.

22. *Real Risk Prioritization* [online]. 2022-02-04 [visited on 2022-02-04]. Available from: https://www.rapid7.com/products/insightvm/features/real-risk-prioritization/.

23. *Skybox Risk Scoring* [online] [visited on 2022-04-25]. Available from: https://lp.skyboxsecurity.com/rs/440-MPQ-510/images/Skybox_TB_Risk_Scoring.pdf.

24. *NVD - Vulnerability Metrics* [online]. 2019-11-09 [visited on 2019-11-09]. Available from: https://nvd.nist.gov/vuln-metrics/cvss.

25. *CVSS v3.1 Specification Document* [online]. 2019-11-09 [visited on 2019-11-09]. Available from: https://www.first.org/cvss/specification-document.

26. SPRING, Jonathan; HATLEBACK, Eric; HOUSEHOLDER, Allen; MANION, Art; SHICK, Deana. Time to Change the CVSS? *IEEE Security Privacy*. 2021, vol. 19, no. 2, pp. 74–78. ISSN 1558-4046. Available from DOI: 10.1109/MSEC.2020.3044475.

27. *What Is VPR and How Is It Different from CVSS?* [Online]. 2020-04-16 [visited on 2022-02-11]. Available from: https://www.tenable.com/blog/what-is-vpr-and-how-is-it-different-from-cvss.

28. *TryHackMe | Vulnerabilities 101* [online]. 2022-02-02 [visited on 2022-02-02]. Available from: https://tryhackme.com/room/vulnerabilities101.

29. *CVSS vs. VPR (Tenable.Sc)* [online] [visited on 2022-02-11]. Available from: https://docs.tenable.com/tenablesc/Content/RiskMetrics.htm.

30. *Changes to Vulnerability Management - Risk Acceptance Process* [online] [visited on 2022-02-25]. Available from: https://www.it.ucsb.edu/vulnerability-management/changes-vulnerability-management-risk-acceptance-process.

31. *What Is Cloud Computing?* [Online] [visited on 2022-03-02]. Available from: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

32. SAWITSKY, Aaron. *Vulnerability Management in the Cloud: Addressing the AWS Shared Responsibility Model* [online]. 2020-01-22 [visited on 2022-02-04]. Available from: https://www.rapid7.com/blog/post/2020/01/22/vulnerability-management-in-the-cloud-addressing-the-aws-shared-responsibility-model/.

33. *IaaS vs PaaS vs SaaS* [online] [visited on 2022-03-02]. Available from: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas.

34. EDUCATION, IBM Cloud. *IaaS vs. PaaS vs. SaaS* [online]. 2021-09-16 [visited on 2022-03-02]. Available from: https://www.ibm.com/cloud/learn/iaas-paas-saas.

35. *Global Infrastructure Regions & AZs* [online] [visited on 2022-04-13]. Available from: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

36. *AWS Organizations Terminology and Concepts - AWS Organizations* [online] [visited on 2022-04-10]. Available from: https://docs.aws.amazon.com/organizations/latest/userguide/orgs_getting-started_concepts.html.

37. *Automated Vulnerability Management – Amazon Inspector – Amazon Web Services* [online]. 2022-02-04 [visited on 2022-02-04]. Available from: https://aws.amazon.com/inspector/features/.

38. KRIEGER, Elazar; MANSHEIM, Ben. *Microsoft Defender for Cloud - an Introduction* [online] [visited on 2022-04-25]. Available from: https://docs.microsoft.com/en-us/azure/defender-for-cloud/defender-for-cloud-introduction.

39. *Skybox: Security Integrations Provide Attack Surface Visibility and Context* [online] [visited on 2022-04-29]. Available from: https://www.skyboxsecurity.com/products/integrations/.

40. *Severity Levels for Amazon Inspector Findings - Amazon Inspector* [online] [visited on 2022-02-12]. Available from: https://docs.aws.amazon.com/inspector/latest/user/findings-understanding-severity.html.

41. HARNIK, Ron; SCHWARTZ, Bar. *The Role of Identity Access Management (IAM) in Cloud Security* [online]. 2020-02-26 [visited on 2022-04-27]. Available from: https://www.paloaltonetworks.com/blog/2020/02/cloud-iam-security/.

42. *IAM Access Analyzer Update – Policy Validation* [online]. 2021-03-16 [visited on 2022-03-23]. Available from: https://aws.amazon.com/blogs/aws/iam-access-analyzer-update-policy-validation/.

43. *Diagram Software and Flowchart Maker* [online] [visited on 2022-04-09]. Available from: https://www.diagrams.net/.

44. *Neo4j Licensing* [online] [visited on 2022-02-24]. Available from: https://neo4j.com/licensing/.

45. *Extending Neo4j - Java Reference* [online] [visited on 2022-03-27]. Available from: https://neo4j.com/docs/java-reference/4.4/extending-neo4j/.

46. *The Neo4j Graph Data Platform* [online] [visited on 2022-04-15]. Available from: https://neo4j.com/product/.

47. *What Is a Graph Database? - Developer Guides* [online] [visited on 2022-03-19]. Available from: https://neo4j.com/developer/graph-database/.

48. *Graph Databases for Beginners: Why a Database Query Language Matters (More Than You Think)* [online]. 2018-08-15 [visited on 2022-03-26]. Available from: https://neo4j.com/blog/why-database-query-language-matters/.

49. *Graph Modeling Guidelines - Developer Guides* [online] [visited on 2022-03-19]. Available from: https://neo4j.com/developer/guide-data-modeling/.

50. *Arrows.App* [online] [visited on 2022-03-25]. Available from: https://arrows.app/.

51. CHANTAVY, Alex. *Cartography near Real-Time Updates · Issue #420 · Lyft/Cartography* [online] [visited on 2022-03-19]. Available from: https://github.com/lyft/cartography/issues/420.

52. *Feature Request: Threading for Faster Account Processing · Issue #257 · Lyft/Cartography* [online] [visited on 2022-03-19]. Available from: https://github.com/lyft/cartography/issues/257.

53. *Neo4j Browser - Neo4j Browser* [online] [visited on 2022-04-09]. Available from: https://neo4j.com/docs/browser-manual/4.4/.

54. *AWS Managed Policies for AWS Config - AWS Config* [online] [visited on 2022-03-26]. Available from: https://docs.aws.amazon.com/config/latest/developerguide/security-iam-awsmanpol.html.

55. *Web Security Dojo* [online] [visited on 2022-04-06]. Available from: https://www.mavensecurity.com/resources/web-security-dojo.

56. *Vulnerability Management Solution for Modern IT | Tenable.Io®* [online] [visited on 2022-04-09]. Available from: https://www.tenable.com/products/tenable-io.

Appendix A

Acronyms

AaaS Analytics as a Service

ACID Atomicity, Consistency, Isolation, Durability

AMI Amazon Machine Image

ASCII American Standard Code for Information Interchange

AWS Amazon Web Services

AZ Availability Zone

CI Configuration Item

CIA Confidentiality, Integrity, Availability

CIDR Classless Inter-Domain Routing

CI/CD Continuous integration, continuous deployment

CMDB Configuration Management Database

CSP Cloud Service Provider

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DaaS Data as a Service

DBMS Database Management System

EC2 Elastic Cloud Computing

ECR Elastic Container Registry

GPL General Public License

HW Hardware

IaaS Infrastructure as a Service

IaC Infrastructure as Code

IAM Identity and Access Management

IIoT Industrial Internet of Things

IoT Internet of Things

IP Internet Protocol

IR Incident Response

ISMS Information Security Management System

ITAM IT Asset Management

ITSM IT Service Management

JSON JavaScript Object Notation

JVM Java Virtual Machine

NACL Network Access Control List

NAT Network Address Translation

NIST National Institute of Standards and Technologies

NVD National Vulnerability Database

OVA Open Virtualization Appliance

OVF Open Virtualization Format

OWASP Open Web App Security Project

PaaS Platform as a Service

PCI DSS Payment Card Industry Data Security Standard

PNG Portable Network Graphics

PoC Proof of Concept

RRS Real Risk Score

SaaS Software as a Service

SLA Service Level Agreement

SVG Scalable Vector Graphics

SW Software

UUID Universally unique identifier

VM Vulnerability Management

VPC Virtual Private Cloud

VPR Vulnerability Priority Rating

Appendix B

Graph data model

As mentioned in 3.4, the whole graph data model put in the PDF would be unreadable to the reader. Instead, JSON exported from the Arrows app[24] is attached to the flash media in the folder `graph`. To open the graph data model, open the Arrows app in your browser, click the yellow arrow button and select import. There you can import the attached JSON to see the whole graph data model.

[24] `https://arrows.app`

Appendix C

User Guide

In this section, the reader can find a brief user guide and requirements to be able to run the application. Also, a similar user guide is available on the flash media in Markdown format.

Prerequisites

There are prerequisites that the user needs to meet.

- The application uses AWS Config to retrieve AWS data, thus needs to be enabled for all required regions.
- Use Neo4j database version 4.0.x or above (tested to be working on 4.4.3).
- Use Neo4j with Enterprise Edition.
- MongoDB (tested on 5.0.6).
- Python 3.6 or above (tested on Python 3.10)
- AWS Inspector2 must be enabled per region.
- AWS Access Analyzer must be created beforehand per region.

The application relies on a few Python packages, this list can be also found in `requirements.txt` in attached media.

- boto3 =1.20.48
- botocore =1.23.48
- neobolt =1.7.17
- neo4j =4.4.1

- pymongo =4.0.2
- netaddr =0.8.0
- pyTenable =1.4.3
- parliament =1.5.2

Configuration

The user must supply a few arguments to the application: First the path to the configuration file. Second, the stages that the user wants to run. The configuration template with all possible settings can be found on the enclosed flash drive. Currently supported stages are:

- aws
- aws-inspector2
- aws-access-analyzer
- tenable-io
- parliament
- traversal

Usage

In usage example C.1, the application will retrieve all the data from AWS. Example C.2 retrieves all findings and vulnerabilities from Tenable.io, AWS Inspector2, AWS Access Analyzer and Parliament. And finally, in example C.3, it runs the traversal on the data. The configuration file template can be found on the attached media in the `clonem` folder under the name `config_template.json`.

```
python3 cli.py --aws \
               --config "config.json"
```

Listing C.1: Application usage example

```
python3 cli.py --tenable-io \
               --aws-inspector2 \
               --aws-access-analyzer \
               --parliament \
               --config "config.json"
```

Listing C.2: Application usage example

```
python3 cli.py --traversal \
               --config "config.json"
```

Listing C.3: Application usage example

Appendix D

Testing setup

If not stated otherwise, all configuration files mentioned in this section can be found on the attached flash drive in the folder `test_setup`.

For testing, MongoDB was running on a testing machine in a Docker. Configuration of the container can be found under the name `docker-compose.yml`. The Neo4j database was running within the Neo4j Desktop application. Configuration of the database can be found in the file `neo4j.conf`.

Folder `nessus` contains configurations of remote Nessus scan and Nessus agent scan. Folder `terraform_templates` contains all Terraform files used to deploy the infrastructure and install shell scripts to configure machines.

Each test run has its own folder containing Terraform files for NACLs, Security groups and NICs, configuration JSON for the application, and exported JSON and PNG from Neo4j Browser containing the results. Unfortunately, the resulting JSON file cannot be imported back to the Neo4j Browser or Arrows.app.

Appendix E

Contents of enclosed flash drive

- `readme.txt` ..... the file with flash drive contents description
- `clonem` ..... the directory of source codes
  - `src` ..... implementation sources
  - `docs` ..... documentation
  - `wiki` ..... wiki
  - `requirements.txt` ..... python required packages
- `test_setup` ..... configuration files for testing
  - `terraform_templates` ..... terraform templates
  - `nessus` ..... Nessus scan setup
  - `test01` ..... test 01 setup and results
  - `test02` ..... test 02 setup and results
  - `test03` ..... test 03 setup and results
- `graph` ..... graph data model
- `thesis` ..... the thesis text directory
  - `src` ..... the directory of LaTeX source codes of the thesis
  - `thesis.pdf` ..... the thesis text in PDF format