university of groningen

faculty of science and engineering

University of Groningen

Securing Infrastructure as Code deployments with Bills of Materials Approaches

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Computing Science
at University of Groningen under the supervision of

Prof. Dr.	F. Turkmen	Computing Science, University of Groningen
Prof. Dr.	V. Andrikopoulos	Computing Science, University of Groningen
Dr.	P. Zuraniewski	ICT, Strategy and Policy, TNO

Aditya Ganesh (s4903382)

August 30, 2023

Contents

Abstract

Modern enterprise networks are becoming increasingly heterogeneous due to the proliferation of readily available open source technologies, which allow for new applications and infrastructure to be brought online in short spans of time. This presents an extremely large attack surface for software supply chain attacks which focus on compromising these commonly used technologies to exfiltrate data and gain control of private enterprise networks.

Monitoring such networks for vulnerabilities and weaknesses is nontrivial, often requiring significant effort from Security Operations teams. The concept of a Software Bill of Materials (SBOM) is frequently used to simplify this monitoring in the context of applications and containers. This thesis explores the feasibility of extending the SBOM concept to programmable network infrastructure in describing Infrastructure as Code (IaC) deployments.

More specifically, the thesis investigates whether a system that employs SBOMs augmented with information such as Common Vulnerability Enumeration (CVE) entries can be used as a representation of a given IaC deployment, and whether such a representation can be used for business impact assessment, i.e. allowing enterprises to make more informed security decisions, and for security orchestration, i.e. enabling existing remediation systems to enact security policies programmatically.

For this purpose, a system is designed and developed that creates a representation of an IaC by constructing a Graph Based Representation (GBR) of a given infrastructure deployment, and then constructing an SBOM. Using this system, it is found that SBOMs are able to represent a given IaC deployment, but their verbose, static nature precludes their application to business impact analysis or security orchestration.

To demonstrate the efficacy of the developed system, it is compared against two existing open source solutions that create an SBOM for an IaC deployment from the perspectives of functionality and execution time. It is found that the developed system performs better in both aspects.

1 Introduction

Modern software systems exhibit staggering complexity, most of which is abstracted by extensive open source technologies. It is now not only possible but routine to have an application that appears to be only a few lines of code deployed in a manner that makes it globally available using simple configuration files. Simultaneously, legacy applications, services, and devices persist, especially in enterprise networks. In this extremely heterogeneous technological landscape, it is not uncommon to see software being developed that makes use of a number of open source libraries, each possibly with their own dependencies, that will run on a combination of public cloud resources and specialized hardware, for example, a smart home solution whose operations involve hardware sensors, an IoT gateway, user devices, and a cloud-based back end, most of which are facilitated by the concept of Infrastructure as Code (IaC), network and information infrastructure deployed, operated, and maintained using configuration files akin to managing software.

A complex network such as this is increasingly referred to as a Software Supply Chain, borrowing a term from industrial supply chain management. In traditional operations, defects in upstream components percolate (either through negligence or malfeasance) downstream into the finished product, sometimes with catastrophic consequences. Consider the 2014 recall of General Motors automobiles[56]: Due to faulty standards and material choices, ignition switches used in GM cars were found to slip, deactivating the car's engine during run-time, which in turn deactivated safety features such as airbags. This fault was present in cars that were sold resulting in 124 deaths[60], and an eventual USD 2.6 billion paid in damages.

The software supply chain is not significantly different. Defects can take the form of vulnerabilities, weaknesses, and non-adherence to best practices. A complicating factor is the significance of external adversaries who can cause extreme financial and reputational damage. An extensive reliance on open source technology creates a broad surface for Software Supply Chain Attacks, where an upstream dependency of a particular software is compromised. The effectiveness of these attacks lies in their abuse of developer trust[51] in systems such as package management and build automation, both virtually essential to accelerating the deployment of any application. This is in turn exacerbated by the fact that some widely used packages are maintained by individual open source contributors[69] who may not be able to spend time in addressing vulnerabilities in their projects, and that many popular packages have malicious alternatives with names designed to trick developers into using them by having low Levenshtein distances to the target packages, a practice known as typosquatting [14]. The clandestine nature of such an attack renders the monitoring of vulnerabilities within an enterprise network a significant challenge, often requiring extensive manpower to analyze, monitor, and patch vulnerabilities as they are discovered within a system. Meanwhile, the cost associated with the breach of such software systems and networks continues on a steeply upward trajectory.

As a motivating example, in the 2020 cyber attack on the United States Federal Government, a cyber-espionage group was able to compromise a number of government and private enterprise organiza-

tions by compromising the Continuous Integration / Continuous Deployment (CI/CD) pipeline of Orion, a network monitoring platform by an upstream provider SolarWinds[57]. Similarly, through a backdoor introduced in Microsoft Exchange Server, over 20000 US governmental organizations were compromised. [45] IBM's[27] annual report into the cost of a data breach found that:

1. 1 in 6 breaches occurs due to a supply chain compromise.
2. The Mean Time to Discovery (MTTD) of a breach is 277 days, seen as an improvement from 287 days in 2021 .
3. 45% of respondents operated a hybrid model of cloud and on premises operations.
4. The average cost of a data breach is above USD 4 million.

An exploration of the current needs in the domain of enterprise security suggests that security teams routinely face challenges in:

1. Automating the processes of monitoring and response to cyber attacks [65].
2. The large number of technology stacks to be understood, monitored, and maintained [65].
3. Communicating the risks posed by these cyber attacks and the need to formulate a response to executive leadership[21] [5].

This combination of ecosystem complexity, pervasive use of software with multiply nested dependencies, and increased costs associated with a data breach provides an imperative to implement traceability in software ecosystems. That Clarke, Dorwin, and Nash [10] found open sourcing to not significantly increase the risk vulnerabilities in code suggests that closed source software is no more secure, and would too need to implement traceability. Traditional supply chains once again provide a template to address this in the form of a Bill of Materials, a structured document enumerating the components, assemblies, and other raw materials that constitute a given product, imparting transparency and traceability to it. Its analogue, the Software Bill of Materials (SBOM) is being set as a requirement by multiple governments. Koran et al.[31] and Muirí [46] assert that SBOMs can mitigate cybersecurity risks by:

1. Enhancing the identification of vulnerable systems and the root cause of incidents.
2. Reducing duplication of effort by standardizing formats across multiple sectors.
3. Identifying suspicious or counterfeit software components.
4. Collecting and communicating this information in such a manner can lower the cost, increase the reliability of, and increase our ability to trust our digital infrastructure.

SBOMs can now be generated for a given piece of software, a set of files, an operating system, or even a container image hosted in a repository such as Docker Hub, and there exist a number of widely used tools that can generate such SBOMs. There are also a handful of solutions, at least one of them open source, that perform the arduous task of tracking the Bills of Materials generated for an enterprise, and flagging outdated, vulnerable, or otherwise weak software. Yet, the potential of SBOMs as a concept in the context of cloud and hybrid environments might still be unrealized; vulnerabilities and weaknesses in dependencies would represent a subset of supply chain vulnerabilities for such a software deployment. Properties such as which ports of a computer running a given piece of software are open, or what services in a given network are open to other parts of the network would also constitute useful information to document. Further, the concept of an SBOM could potentially form a base for more than simple enumeration and documentation. Thus, it would be interesting to explore the concept of the SBOM in the context of cloud environments, specifically:

1. What useful information about cloud environments can be captured using existing SBOM formats?
2. What additional information is needed to make these truly representative of the deployment?

3. How can the SBOM be integrated to other systems such that it can achieve a greater measure of security for the deployment, if at all?

A survey of the existing landscape of SBOM generators demonstrated that While there are a number of existing solutions that create SBOM for container images, machine file systems, or similar objects, few tools generate SBOMs for cloud environments, and none attempt to use the generated SBOMs for any purpose other than reporting. It would thus be a novel concept to develop a system based on SBOMs that not only perform this reporting, but can also alleviate the challenges of automation, reporting, and heterogenity faced by security teams. This leads us to identify a set of research questions, stated formally in section 1.1, that will be explored through the course of this thesis.

The work presented is undertaken in the framework of the Automated Security Operations (ASOP)[62] project led by De Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek (TNO), continuing a theme [70] of research in the field of defence against attacks on Information and Communications Technology (ICT) networks. This project is undertaken by a consortium with the Communications Service Provider KPN, the Software-as-a-Service based Enterprise Architecture solution provider BizzDesign, the cloud infrastructure technology provider VMWare as members. The project undertaken consists of three phases, namely validate, operationalize, and scale up. One of the stated goals of the second phase, the creation of an abstraction layer for infrastructure, guided the research questions posed.

In chapter 5.1 it will be shown that a set of existing standards were evaluated against the purpose of creating such a system for an IaC deployment that could model multiple type of deployment, and were found to not support one or more key requirements. It would thus be a temptation from an academic perspective to devise a new standard that supports all identified requirements, and attempts to be future-ready. In their study on factors that contribute to the success of a standard Van de Kaa and De Vries [29] concluded that the commitment of the standard promoter, diversity of the standard's network of users, and support for the creation of complimentary goods (here, third party solutions) were common to all winning formats surveyed. A standard presented without these factors is likely to meet a fate demonstrated in figure 1.1. One such standard that should in fact have facilitated a significant portion of the work to be presented is discussed in section 5.1.2. It is thus preferable to leverage an existing standard that meets these criteria should the work presented in this thesis have any aspirations of being adopted by the wider community.

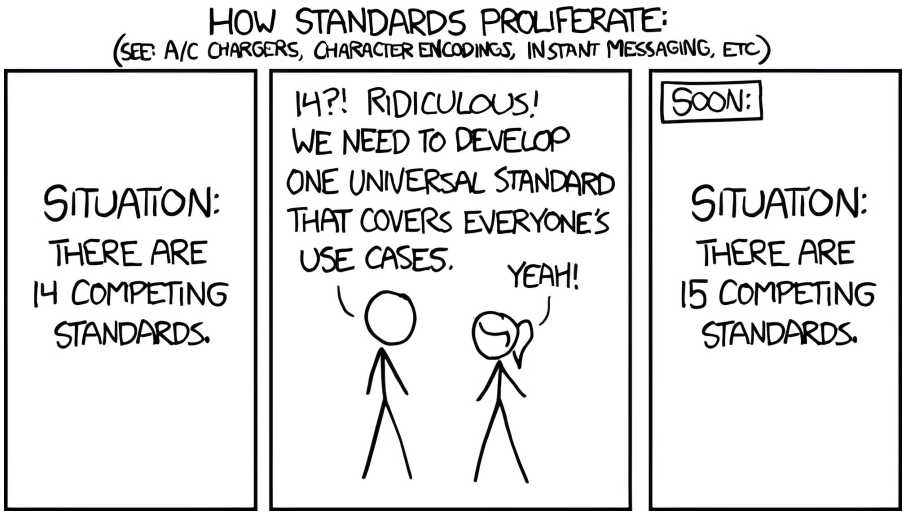

Figure 1.1: The eventual fate of a new standard[47]

This thesis shall first briefly explain some of the key concepts discussed throughout the material in chapter 2, along with a more detailed description of the standards that were chosen for the purpose of answering the research questions posed, and a selection of the tools used in practice. Chapter 3 will then provide the conceptual approach used in answering the research questions posed, and the considerations that needed to be taken in the process of answering them. The technical implementation of these concepts will then be presented in chapter 4, along with a framework for their evaluation, and the results thereof. Chapter 5 will then present a summary of related academic work, discuss the results and the relevance of the work presented. Finally, chapter 6 will summarize the conclusions drawn, and suggest possible avenues for future development.

1.1 Research Questions

The primary research questions may be stated as follow:

RQ1. Can Software Bill of Materials (SBOM) based techniques be used to describe Infrastructure as Code (IaC) deployments, providing information such as Common Vulnerability Enumeration (CVE) entries affecting a given component?

- (a) If not, is there some set of tools and techniques that can be used to impart similar traceability?

RQ2. If such a technique is possible, can it be used to integrate with the existing open-source enterprise architecture systems

- (a) Can such a technique be used to facilitate business impact assessment?

RQ3. If such a technique is possible, can it facilitate security orchestration by integrating with existing orchestration systems?

- (a) Can such a technique be used to enact security policy, or security instructions?

2 Background

This thesis borrows from three domains - cloud infrastructure, business modelling, and network security. In order to provide a greater understanding of the relevant concepts that will be used in chapters 3 and 4, a brief introduction to them is provided in this chapter.

Beginning with the concept of Infrastructure as Code (IaC), the workings of the platform selected for this thesis, Kubernetes (K8s) will be described. Then, the concept of a Software Bill of Materials (SBOM) will be introduced, along with this work's standard of choice CycloneDX (CDX). This combination is intended to provide sufficient understanding of the challenges to be addressed in answering RQ1.

A brief introduction to the concept of business modelling and Business Impact Assessment (BIA) will then be given, along with the standard of choice ArchiMate. These concepts will be used in addressing RQ2.

Finally, the concept of Integrated Active Cyber Defense (IACD) will be introduced, which gives us a framework to understand how autonomous cyber security may be implemented. Our chosen implementation OpenC2 (OC2) will then be discussed, giving us a means to answer RQ3. These concepts will then be put to practice in chapter 3.

2.1 Infrastructure as Code

Hütterman[26] describes the notion of infrastructure as including "every part of the solution that is not the developed software application itself", typically seen as the purview of System Administrators, involved in software operations rather than development. The term "IaC" would thus suggest handling such resources similar to the application code delivered by the development team. Practically, this translates to a requirement that infrastructure resources such as servers, network devices, operating systems etc. be provisioned by means of machine readable configuration files, typically deployed and maintained using concepts borrowed from Continuous Integration / Continuous Deployment (CI/CD) practices used in software development. The deployment of these resources may then be in the form of virtual machines and containerized instances, but may also include physical devices.

A number of open standards implement the concept of IaC at differing levels of abstraction. Salt, Ansible, and Puppet for example focus on event driven automation and configuration management.

Amazon Web Services, Microsoft Azure, and Google Cloud maintain AWS CloudFormation, Azure Resource Manager, and Deployment Manager respectively as IaC interfaces for their cloud environments. OpenStack's Heat templates provide similar functionality. HashiCorp's Terraform provides a more vendor-neutral approach to managing cloud computing environments. Pulumi, another such

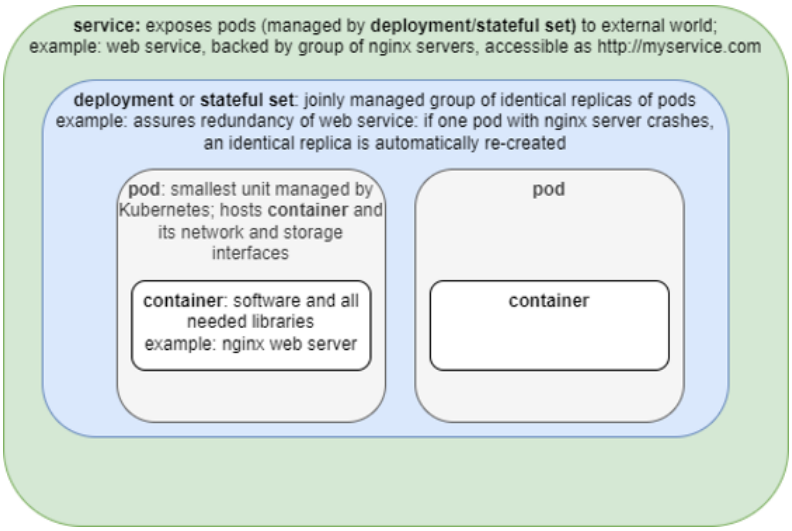

Figure 2.1: Kubernetes objects

standard, attemtps to address the same problems as Terraform. It, however, expresses itself as programme code instead of configuration files.

IaC concepts also extend to the realm of application deployment with tools like Kubernetes (K8s) and Docker Swarm, which use containerization to deploy applications on top of existing physical or cloud infrastructure. K8s has become the de-facto standard[9] of IaC for workload management due to its ability to simplify container-based application deployment, to the point that all major cloud providers such as AWS, Google Cloud, and Azure and Alibaba Cloud offer managed K8s instances.

2.2 Kubernetes

K8s defines itself [33] as an "open source system for automated deployment, scaling and management of containerized applications". It can be run either standalone on a single machine, or as a cluster of arbitrary size. Its architecture consists of one or more controller nodes in high availability mode managing a number of worker nodes which run a number of containerized applications or workloads. K8s implementations deploy an API server to interact with the controller over REpresentational State Transfer (REST), and a scheduler that handles workload allocation. K8s configurations are typically written and parsed in YAML files, colloquially termed "manifests".

This section uses the official K8s documentation[32] as primary reference.

2.2.1 Kubernetes Objects

The key K8s objects of relevance to the work of this thesis shall be presented in increasing order of abstraction. A pictorial reference is given in figure 2.1

A **container** runtime in the context of K8s is thus a standardized, isolated environment running on some host machine. These runtime environments are typically instantiated from pre-packaged container images. While container runtimes form the fundamental functional units in a K8s cluster, they are not deployed directly, as in the case of a Docker environment, but through a "pod".

K8s guides refer to **pods** as the smallest deployable units in K8s. These are logical groupings of containers, defined using a specified image, with a given number of replicas with shared network

resources, and possibly shared storage. In practice, pods are rarely instantiated and interacted with by themselves. Instead, one typically makes use of "workload resources", having defined behavioural characteristics, which aim to reduce pod micromanagement. The work presented in this thesis makes use of two such resources - Deployments and StatefulSets. Both run pods with identical application code. The key difference lies in the fact that pods in a StatefulSet have distinct identities, and therefore predictable names, while pods in a Deployment are meant to be completely interchangeable, running stateless applications, and therefore have no need for predictable naming.

The preferred method of exposing network applications within a cluster as well as to the outside world, **services** provide invariant, predictable, DNS resolved endpoints that are fulfilled by a set of pods that meet certain selection criteria described in section 2.2.2. This structure allows for pods exposing an application to be changed arbitrarily without impacting the means by which other applications in the cluster address it. In the case of intra-cluster communications, it is typically possible to derive a service endpoint as `<service-name>.<namespace>:<port>`.

For example, a service defined as given in listing 2.1 it is possible to address the service as `legacy-lab-svc.lab:8080`.

```
metadata:
  name: legacy-lab-svc
  namespace: lab
spec:
  ports:
  - name: legacy-lab-svc-port
    protocol: TCP
    port: 8080
```

Listing 2.1: Deriving a service endpoint

Namespaces serve as a means of conceptual partitioning of resources into "distinct, non intersecting collections" for ease of resource management. Resources within a namespace must have unique names, but names can be repeated across namespaces, and accessed using the combination of the namespace and resource name.

2.2.2 Labels and Selectors

Taking the form of key-value pairs, contained in Pods' metadata, Labels form an integral part of K8s orchestration. These labels are matched against Selectors, similar sets of key-value pairs used to identify and select resources in a non-unique manner. The official documentation[34] states the motivation for this system to be a means of imposing arbitrary organization onto the cluster without enacting fundamental semantic changes to the means of operation. Consider the example given in listing 2.2 describing a deployment. Here, the section `template` describes the pods that will be instantiated for specifically for the deployment. With multiple labels in place used to describe the pods, here `app = minio-pods`, and `track = stable`.

The `selector` is used by K8s to identify pods that will be used to create the deployment. The `matchLabels` here are necessarily a subset of the labels given in the `template`. Thus, any $\binom{2}{1}$ or $\binom{2}{2}$ of `app = minio-pods` and `app = minio-pods` will satisfy the deployment.

```
spec:
  selector:
    matchLabels:
      app: minio-pods
  template:
```

```
    metadata:
      labels:
        app:  minio-pods
        track: stable
```

Listing 2.2: Labels describing a deployment, with a pod template

In the example provided a single `matchLabel : app = minio-pods` is provided. Thus, if a new pod is declared as given in 2.3, this pod too will satisfy the selection criteria for the deployment, and therefore be included. On the other hand, if a service is defined, as in 2.4, only the pods defined in the `template` of 2.2 will satisfy the requirement, and thus be used. A similarly defined service 2.5 with an additional matchLabel not contained in either set of labels will not be fulfilled.

```
metadata:
    labels:
        app: minio-pods
        track: canary
```

Listing 2.3: Labels describing another pod, with an additional label

```
spec:
  selector:
    app: minio-pods
    track: stable
```

Listing 2.4: Match Labels required by a service

```
spec:
  selector:
    app: minio-pods
    track: stable
    version: 1
```

Listing 2.5: Unsatisfiable match labels

2.2.3 Network Fabric and Network Policy

While K8s is capable of orchestrating cluster operations, handling scaling and fault handling, it performs no traffic flow management. This is instead the purview of Container Network Interface (CNI) plugins such as Flannel, Cilium, and Calico, each with a unique set of features, and underlying technology (for example VXLAN vs eBPF). One such feature is the support of K8s' NetworkPolicy specification, integral to enabling cluster security. In their review of mainstream CNIs Qi et al. [52] found that most CNIs support this specification, with the exception of Flannel. Shamim et al. [55] determined that having a CNI in place with sensible, secure network policy is integral to cluster security. However, Minna et al. [41] highlight that careful, vigilant, monitoring is required as the CNI plugins run as privileged programmes on worker nodes, and a compromise to these processes could affect the entire network.

The typical structure of a network policy is given in 2.6. Affected resources are typically identified using the `namespace` of the policy, and the `matchLabels` given in the `podSelector`, evaluated as described in section 2.2.2. A given NetworkPolicy may contain any $\binom{2}{1}$ or $\binom{2}{2}$ `policyTypes` from `"Ingress"` and `"Egress"`.

If one of the above `policyTypes` is defined, it is typical to have a corresponding section contained within the NetworkPolicy's `spec`, containing an array of modular rules that explicitly allow certain

flows of traffic. An example of one such flow rule is presented in listing 2.7 intended to allow traffic originating from the resources having the label `app = legacy-lab-pods` situated in the `lab` namespace. It is also possible to combine these rules in parallel, by having them form separate array entries under the flow label `from`, resulting in traffic being allowed from the `lab` namespace `OR` from pods labelled `app = legacy-lab-pods`. Similarly it is possible to specifically allow traffic on certain ports and protocols, or certain IP blocks. Multiple network policies may be implemented that affect a given resource, the resultant behaviour being the logical OR of all rules.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: <policyname>
  namespace: <targetnamespace>
spec:
  podSelector:
    matchLabels:
      <targetlabels>
  policyTypes:
    - Ingress
  ingress:
    <ingressrules>
```

Listing 2.6: NetworkPolicy structure

```
ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          user: lab
      podSelector:
        matchLabels:
          app: legacy-lab-pods
```

Listing 2.7: NetworkPolicy structure

There exist a few special rules to note:

1. **(Default) Deny All** - Implemented by declaring a policyType and either omitting the corresponding section in the NetworkPolicy or declaring no rules within it. In conjunction with other rules, this ensures that only explicitly allowed communication paths exist within the network
2. **Allow all** - Implemented by using `"{}"` as the only rule for a given flow type. This precludes the enforcement of any other rules.

2.3 Software Supply Chain

In traditional operations a supply chain refers to a network of entities and processes that, through a number of intermediate steps, transforms raw materials into finished products and delivers them to the end user. Its analogue in software may be thought of as the set of components and processes that constitute a finished software product to the end consumer.

With this definition, Software Supply Chain Security may be thought of as an analogue to supply chain risk management in the traditional operations realm, the principal difference being what flows through the supply chain: products in the traditional, data or information in software. Thus, security in the software domain is primarily concerned with the confidentiality, integrity, and availability of

the information that flows through it, along with the risks posed by the disruption of this information flow [59]. Here, the work of Ellison et al. [15] is used as primary reference.

Ellison et al. discussed a set of risks that contributed to a security breach in a supply chain:

1. Poorly defined security requirements.
2. Coding and design defects or weaknesses.
3. Poor access control.
4. Insecure deployment configurations.
5. Operational changes in the use of a product in the field that introduce other security flaws.
6. Information mishandling or manhandling.

Two broad approaches were provided by Ellison et al. with the intention that they be used in a complementary manner :

1. Attack surface analysis.
2. Risk assessment through threat modelling.

The goal of the former is to create a comprehensive understanding of the existing system state from a defensive, bottom-up perspective including the software used as well as the dependencies that a given system or piece of software may have on other components within the system, thereby allowing for the creation of a vulnerability map.

Threat modelling on the other hand attempts to identify security risks within the system by taking a top-down adversarial approach, first considering the normal operation of the system, and then determining exploitable weaknesses or vulnerabilities within the system.

2.4 Software Bill of Materials

The role of a BOM in traditional operations is to serve as a structured record of the raw-materials, intermediate components, and assemblies that go into the creation of a product. Typically each of these entities is provided a unique identifier that allows bills of materials to be connected hierarchically in the form of a Directed Acyclic Graph (DAG).

DAGs as the name would suggest are directed graphs without cycles, i.e. it is not possible to trace a path from any given node back to itself, and no two nodes have bidirectional connections. DAGs have at least one vertex with zero in-degree (i.e. a root), and one vertex with zero out-degree (i.e. a leaf).[61]. DAGs differ from trees by virtue of the fact that a parent node in a DAG may also be a parent of one of its children's child nodes. Trees may thus be considered to be a subset of DAGs. Like a tree, a DAG allows for a path to be traced from root to leaf through an arbitrary set of branches within the DAG.

Such a structure provides a number of useful benefits, such as:

1. Improved resource planning.
2. Increased product traceability.
3. The ability to evaluate similarity between products.

The Bill of Materials concept translates well in the software domain[1], especially in its identification of vulnerable software components, underlying the importance of its adoption in modern software supply chains. From Ellison et al.'s mitigation strategies given in section 2.3, SBOMs may be considered to fall under the category of Attack Surface Analysis.

	Scan Type			
	Filesystem	Image	Run-time	Design-time
Snyk	✓	✓	X	X
Trivy	✓	✓	X	✓
Syft + Grype	✓	✓	X	X
Kubescape	X	X	✓	X
Aqua CSP	X	X	✓	X
Falco (Sysdig)	X	X	✓	X
sbom-operator	X	✓	X	✓
ksoclabs/KBOM	X	✓	✓	✓
kubernetes-sigs/bom	X	✓	X	✓

	Open Source	BOM Reporting	Drift Detection	Hybrid Environments
Snyk	✓	✓	X	X
Trivy	✓	✓	X	✓
Syft + Grype	✓	✓	X	✓
Kubescape	✓	X	X	X
Aqua CSP	X	X	✓	X
Falco (Sysdig)	✓	X	✓	X
sbom-operator	✓	✓	X	X
ksoclabs/KBOM	✓	✓	X	X
kubernetes-sigs/bom	✓	✓	X	X

Table 2.1: Comparison of Infrastructure Vulnerability Scanning Tools

As of 2023, three major standards exist. This chapter shall elaborate on CycloneDX in section 2.5. The other standards will now be discussed briefly. A survey of the existing tools supporting these standards shall be presented at the end of section 2.5.

First published by the ISO (IEC 19770)[4] in 2009, Software Identifier (SWID) is the first major standard to satisfy the conditions of an SBOM. SWID tags typically contain information such as a unique tag ID, the software name, vendor, version number, patch number, or similar, used to uniquely identify a given piece of software. Used in conjunction with the SWID tags of dependent software, it is possible to construct a bill of materials. However in isolation, SWID tags can only serve as identification.

The Software Package Data eXchange (SPDX), supported by the Linux Foundation was first drawn up in 2011 [17]. As of 2021, it has been accepted and published by the ISO (IEC 5962:2021). The original,stated purpose behind its inception was to "enable companies and organizations to share license and component information (metadata) for software packages and related content with the aim of facilitating license and other policy compliance." Seeing the utility of the standard in capturing more than simple license information, SPDX has since broadened this scope to vulnerability enumeration and redundancy reduction[18].

Several tools have been developed for the cloud infrastructure ecosystem to address the challenge of vulnerability scanning and enumeration. A selection of the most widely-used ones is presented along with a summary of their capabilities in table 2.1.

2.5 CycloneDX

Buttner and Martin [7] describe the standard as lightweight, open, and suitable for use in application security applications and supply chain component analysis. This is evidenced by the organization of the typical SBOM it generates, as shown in listing 2.8, where the constituents and vulnerabilities of a given object may be accessed readily. A number of Bill of Materials types and use cases are supported.

The CDX specification may be expressed in three primary schemas : JSON, XML, and Google's Protocol Buffer. The methods employed by this thesis restrict themselves to interacting with the standard in its JSON form.

Each SBOM must contain a unique serialNumber compliant to the UUID / RFC-4122 specification. Should a given object be scanned multiple times, even if the SBOMs are otherwise identical, the `serialNumbers` must differ. The version field is an integer tracking the number of scans performed on an object. The metadata component field contains three identifying subfields, namely:

1. type : The class of object described by the SBOM, e.g. application, container, library, or operating-system.
2. bom-ref : A static, unique identifier for a given SBOM. In contrast to the serialNumber, this identifier should persist for identical scans.
3. name : A short nominative of the object being described.

An object captured in CDX (typically an application) may have one or more services exposed by it, enumerated in the corresponding field. As services in a network represent abstractions by which parts of the system communicate with each other, it is informative to capture these. The expression of a service is given in listing 2.9. Each entry should contain a list of valid endpoints to address the service, and the nature of data flows through the service. Such flows are described using the format given in listing 2.11. The direction of traffic and intended source/destination may thus be captured.

Each object's description, expressed as a component, may list a number of subcomponents within it. This enables the construction of the Directed Acyclic Graph that a Bill of Materials format must create. Each referenced subcomponent must be described as in listing 2.10

```
{
    "bomFormat": "CycloneDX", // required
    "specVersion": "1.5", // required
    "serialNumber": "urn:uuid:<rfc-4122-compliant>",
    "version": 1,
    "metadata": {
        component:{
            "type": "<cdx-type>",
            "bom-ref": "<unique-static-id-pref-rfc-4122-compliant>",
            "name": "object-name"
        }
    }, // required
    "services": [...],
    "components": [...],
    "vulnerabilities":[...]
}
```

Listing 2.8: CycloneDX BOM structure

```
{
    "bom-ref": "<unique-static-id-pref-rfc-4122-compliant>",
```

```
    "name" :
    "properties": [...],
    "endpoints": [...],
    "data":[...]
}
```

Listing 2.9: CycloneDX Service Description

```
{
    "bom-ref": "<unique-static-id-pref-rfc-4122-compliant>",
    "name" :
    "type": "<cdx-type>",
    "properties": [...],
    "components": [...],
    "vulnerabilities":[...]
}
```

Listing 2.10: CycloneDX Component Description

```
{
    "flow" : ""<inbound/outbound/bi-directional/unknown>",
    "classification" : "<string : data-sensitivity/type>",
    "source" : "<bom-ref>",
    "destination" : "<bom-ref>"
}
```

Listing 2.11: CycloneDX Component Description

While SPDX and CycloneDX are both capable of generating SBOMs, and thus seen as competing standards, the intended purposes behind the two are significantly different. The primary goal of SPDX since its inception has been license discovery and compliance, with more recent versions adding support for a Bill of Materials description compliant with the description given by Muirí et al. [46] from the NTIA Framing Working Group. CycloneDX on the other hand was constructed with the goals of component description in mind. While the two standards are now roughly equivalent in terms of their ability to describe a given image or file system, the extensibility of CycloneDX suggests suitability for use in describing IaC systems in their entirety.

2.6 Business Impact Assessment

Part of the broader field of Business Continuity Planning, BIA refers to the process of quantifying the risk posed to an organization, or process, by an adverse event outside the nominal scope of operations. Doughty [13] suggests that, depending on the context of analysis, the impact being measured could be financial, i.e. losses incurred or operational, i.e. service outages, inability to meet quality standards etc. The information required for such analysis is typically a mixture of subjective and objective sources. The objective sources used are usually records of technology platforms, software and equipment used, and data stored, while the subjective information pertains to the criticality of the objective data captured.

Enterprise Architecture systems are typically used to simplify the process of collecting and representing the information collected. It aims to convert abstract concepts such as communication within the organization, operations flows, and organizational roles into objects that can be related with the data that flows through them, and support BIA by demonstrating the effects of events occurring in one part of the system on the remainder. Two open source standards exist, both published by the Open Group,

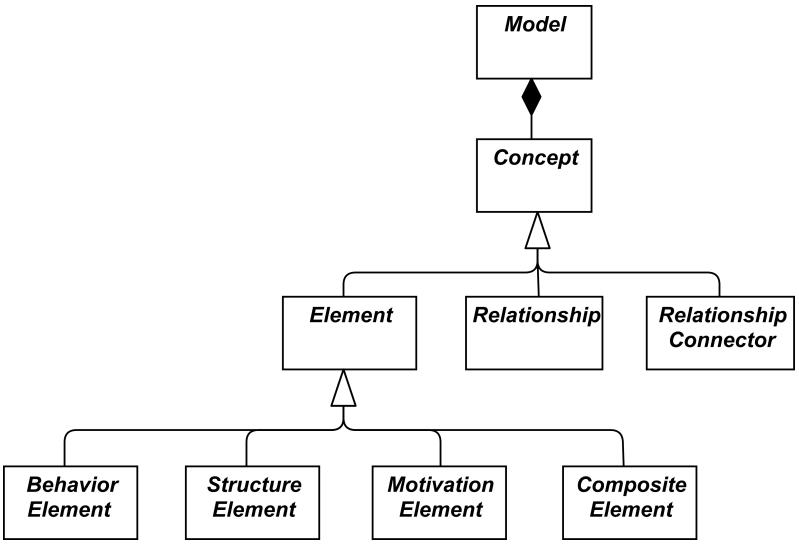

Figure 2.2: The Hierarchy of ArchiMate concepts[20]

namely The Open Group Architecture Framework (TOGAF) and ArchiMate, and are largely interoperable with most third party tools supporting both standards. Lankhorst et al. [35] discuss the need from an enterprise architecture perspective for an object-relation basis for architecture description, drawing similarities to standards such as the Universal Modelling Language (UML) and Business Process Modelling Notation (BPMN).

2.6.1 ArchiMate

ArchiMate aims to serve as a modelling language for both enterprise architecture as well as information exchange. This section uses the publications The Anatomy[35] and The Architecture[36] of the ArchiMate Language by Lankhorst et al. along with the ArchiMate 3.2 specification published by The Open Group[20] as primary references.

In the ArchiMate language, a model is considered to be a collection of concepts. Each concept could be an element, i.e. an object of some sort, or a relationship between elements. Elements can be of multiple types, behavioural, structural, motivational, or a composition of one or more sub-elements. This hierarchy is demonstrated in figure 2.2

Elements in ArchiMate belong to one of four (core framework) or six (full framework) layers, depending on their granularity and scope. Elements in the business layer, for example, pertain to the services, events, and objects pertaining to the operations of the enterprise, while technology layer elements describe the physical and logical entities that make up the infrastructure that supports the enterprise. These layers can be thought of as another hierarchy, where technology layer elements enable application layer elements, which further realize business processes. The elements of interest in the context of this thesis belong to the technology layer.

Aspects, on the other hand, describe the nature of a given element. Likening aspects to parts of sentence structure, Lankhorst et al. describe active structures as subjects or actors. Active structure elements represent objects whose purpose is to perform a certain action. Behaviour elements specify what these actions are. Meanwhile, passive structure elements represent the objects upon which the actions are performed. For example, a physical computer (an active structure element) may read a set of static files (passive structure elements) to deploy a web service (a behaviour element).

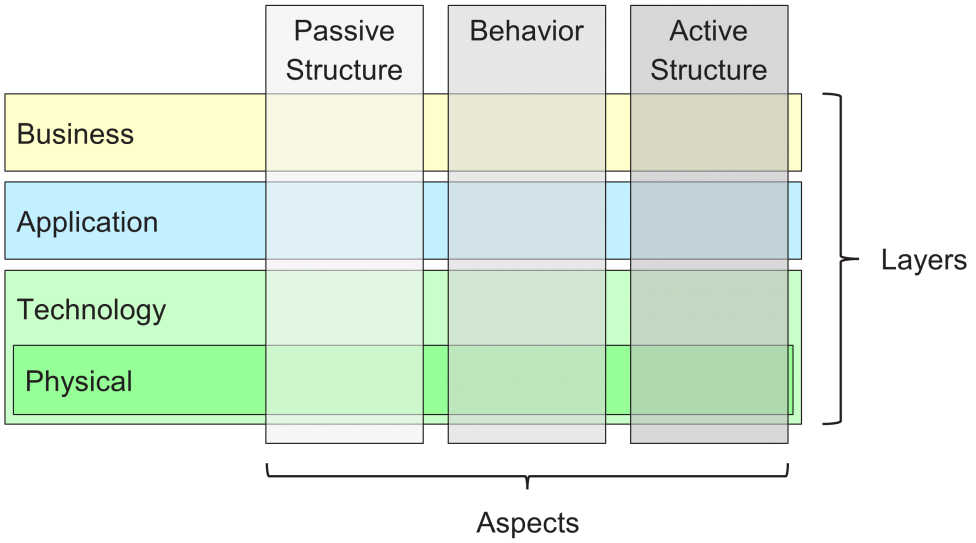

Figure 2.3: ArchiMate layers and aspects[20]

In order to facilitate interoperability between different implementations of the ArchiMate standard, the **OpenModel Exchange** format was devised, using XML as a base. It contains a set of key sections, namely `elements`, `relationships`, `propertyDefinitions`, and `views`. The structure of the file is given in listing 2.12, while the formats used for elements, relationships, and custom properties attached to elements are given in listings 2.13, 2.14, and 2.15 respectively.

```
<model xmlns="..." xmlns:xsi="..." xsi:schemaLocation="..." identifier="[Unique-ID]">
    <name xml:lang="en">[Model Name]</name>
    <elements>
        ...
    </elements>
    <relationships>
        ...
    </relationships>
    <propertyDefinitions>
        ...
    </propertyDefinitions>
    <views>
        ...
    </views>
</model>
```

Listing 2.12: OpenModel Exchange Structure

```
    <element identifier="[Unique-ID]" xsi:type="[Element-Type]">
      <name xml:lang="[ISO_639-1_Code]">[Element-Name]</name>
      <properties>
        <property propertyDefinitionRef="[Property-Identifier]">
          <value xml:lang="[ISO_639-1_Code]">[Value]/value>
        </property>
      </properties>
    </element>
```

Listing 2.13: OpenModel Element Format

```
<relationship   identifier="[Unique-ID]"
                source="[Source-ID]"
```

```
                target="[Target-ID]"
                xsi:type="[Relationship-Type]"/>
```

Listing 2.14: OpenModel Relationship Format

```
<propertyDefinition identifier="[Property-ID]" type="string">
  <name xml:lang="ISO_639-1_Code]">[Property-Name]</name>
</propertyDefinition>
```

Listing 2.15: OpenModel Property Format

2.7 Integrated Active Cyber Defense

Mavroedis[39] described the challenges faced in securing modern network and cloud infrastructures exemplified by the statistic that the average time to compromise a network is in the order of minutes, while the time to identification and containment of such a breach averages at 280 days. In order to bridge this gap, he identified the concept of Integrated Active Cyber Defense (IACD) introduced by the Johns Hopkins University Applied Physics Laboratory [12] as a conceptual framework to implement, requiring three principal capabilities:

1. Automation
2. Information sharing
3. Interoperability

Mavroedis' envisioned implementation of IACD comprises three principal components:

1. Structured Threat Information Exchange (STIX) [3]
2. Collaborative Automated Course of Action Operations (CACAO) playbooks [40]
3. OpenC2 (OC2) [39]

2.7.1 OpenC2

Mavroedis [39] describes the motivation behind its development to be the creation of a structured, data driven language that would enable automation of cybersecurity operations, thereby implementing the IACD framework, in an operations environment with heterogeneous, proprietary infrastructure. OpenC2 is a specification designed to achieve this by facilitating communication to, from, and between cyber defense systems present within a given infrastructure in a standardized manner. This section uses Mavroedis et al. and the OASIS OC2 Specification [49] as primary sources.

The standard OC2 implementation is a producer-consumer model as demonstrated in figure 2.4. The producer generates a command to a set of systems with a given set of instructions. The consumers of this command perform the requested action, and return the result or status to the producer in the form of a response. Each consumer may implement one or more actuators which interpret the received message into the intended actions.

OC2 makes use of the JSON format for its messages. Each message must conform to the format given in listing 2.16. A message can be one of two types: command or response. Each message allows for multiple recipients. The payload of the message is contained in the "content" field.

```
{
    "content_type": application/openc2, // Mandatory
    "msg_type": <command-or-response> // Mandatory
    "request_id": <RFC4122>, // Mandatory
```

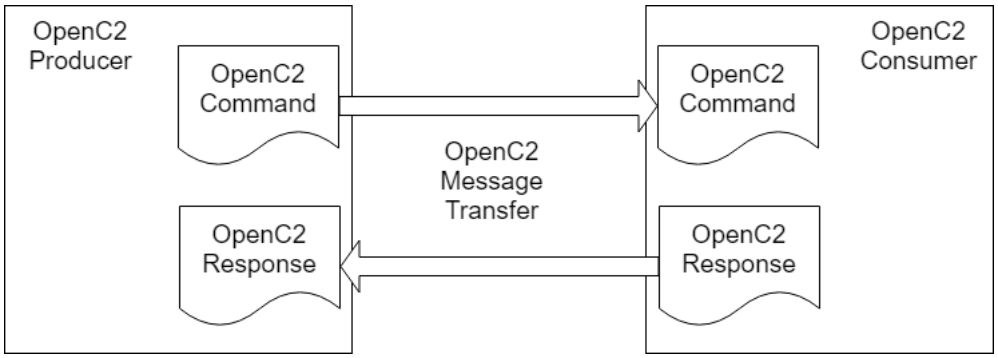

Figure 2.4: OC2 Operation

```
    "created": <unix-timestamp>,
    "from": <message-source>,
    "to": <message-dest(s)-str-or-list>,
    "content": <payload>
}
```

Listing 2.16: OpenC2 Message Format

The format of an OC2 command is given in listing 2.17. Each command must necessarily contain an `action` which takes one of 32 predefined values (for example `"scan"`,`"allow"`, or ,`"redirect"`). Actuators may implement a subset of the actions, on a defined set of targets. A valid consumer must implement a method to query it for supported features and actuator profiles. A single target must be provided in each message, having a single specified target type. Arguments may be provided in addition to the target depending on the actuator and the action to be performed. If multiple actuators are defined in a system, the actuator may be specified.

```
{
    "action": "<action>", // Mandatory
    "target": {...}, // Mandatory
    "args": {...},
    "actuator": {...}
}
```

Listing 2.17: OpenC2 Command Format

A valid response, as seen in listing 2.18 must at least contain a `status`, which implements a subset of HTTP response codes. A response may optionally contain a human readable status message and, if required by the issuing command, results of the query made.

```
{
    "status": <http-response-code>, // Mandatory
    "status_text" : "<human-readable-status-message>"
    "results": {...}
}
```

Listing 2.18: OpenC2 Response Format

3 Conceptual Approach

The inferences drawn from section 5.1.4 led to the conceptualization of a system that would generate an SBOM for a given cloud infrastructure, attempt to use it to create a representation of the infrastructure for Business Impact Assessment, and effect changes to the infrastructure given commands from a security orchestrator. This chapter catalogues the efforts to implement this, first discussing the key features required for an accurate representation of a cloud infrastructure environment, two approaches for SBOM creation given an IaC deployment expressed in Kubernetes, and highlighting which method is preferred based on the features provided by them. A number of considerations to be factored in the creation of this representation are also presented, along with brief descriptions of their operational significance. Then approaches to extend the preferred method to extend to a BIA system expressed in ArchiMate, and security orchestration using OpenC2 are detailed.

3.1 Capturing Infrastructure in CycloneDX

As described in section 5.1.1, current SBOM standards and tools tend to focus on the Bill of Materials for a given image, file-system, or application. Representing a given IaC configuration file, or a set thereof, is not currently supported. A method must therefore be implemented that creates such a representation. In order to do this for K8s, the following resource types are considered

1. Services
2. Deployments
3. StatefulSets
4. Containers

A logical dependency graph may then be created. It is noted here that

1. The pods denoted by Deployments and Statefulsets are logical groupings : once applied, a service mapped to such a pod truly depends on the containers that comprise the pod. However, these logical groupings are still useful to the model to be created due to the method employed by K8s to map requirements to resources, i.e. match labels.
2. A non-K8s top layer “Application” has been created. This allows for a simplified singular SBOM representation instead of a number of lower layer SBOMs that would subsequently need to be associated, while staying consistent with the notion of a web application consisting of a number of linked services.

3.1.1 Identifiers

A key requirement of the model to be created, i.e. traceability, mandates the use of unique identifiers for all components. Two approaches have been employed to obtain such identifiers, with the choice

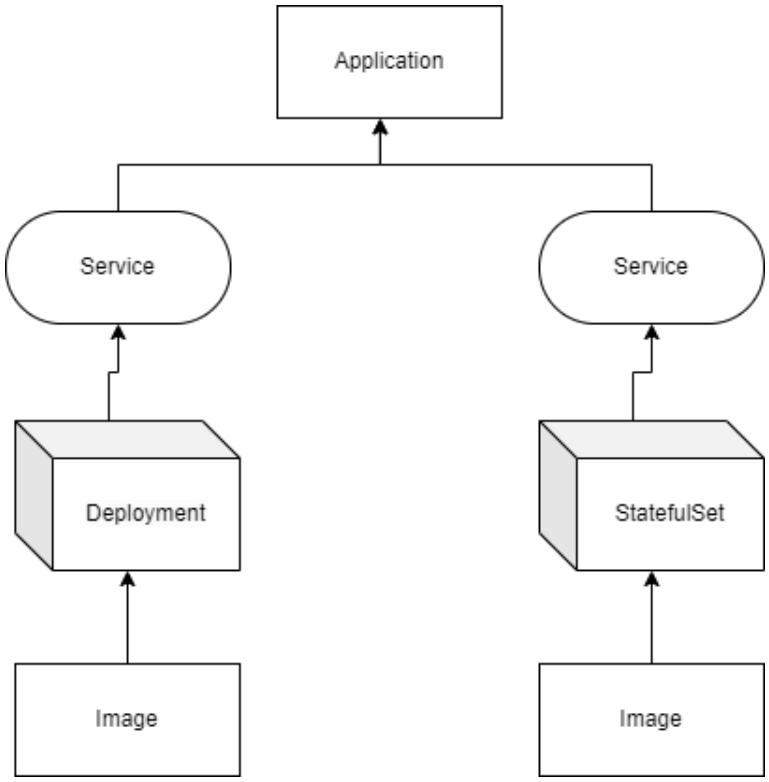

Figure 3.1: K8s Dependency Graph

of approach contingent on the state of the infrastructure:

1. Design Time : UUIDs complying with RFC 4122
2. Run Time : Use the UUIDs created for each object by K8s itself

3.1.2 Approaches to creating SBOM

Two approaches were also made for the creation of the SBOM itself, namely:

1. A naïve direct conversion
2. Creation of a GBR

The second approach materialized as a result of investigating RQ2 and RQ3. In order to evaluate the effectiveness of the two approaches in their ability to accurately represent the infrastructure, five principal features were selected to determine the most appropriate method:

1. Capturing the infrastructure used and services exposed
2. Hierarchical enumeration of network elements
3. Vulnerability enumeration
4. Traceability of fulfillment between services and the underlying infrastructure
5. Traceability of communication between elements in the network

3.1.3 Direct Conversion

A script was written that ingests static K8s manifests in YAML form and creates a bill of materials in the hierarchical structure presented in 3.1 . Each element (e.g. a Deployment) is converted into a Bill of Materials entry, assigned with a unique identifier, here termed `"bom-ref"`. This `"bom-ref"`

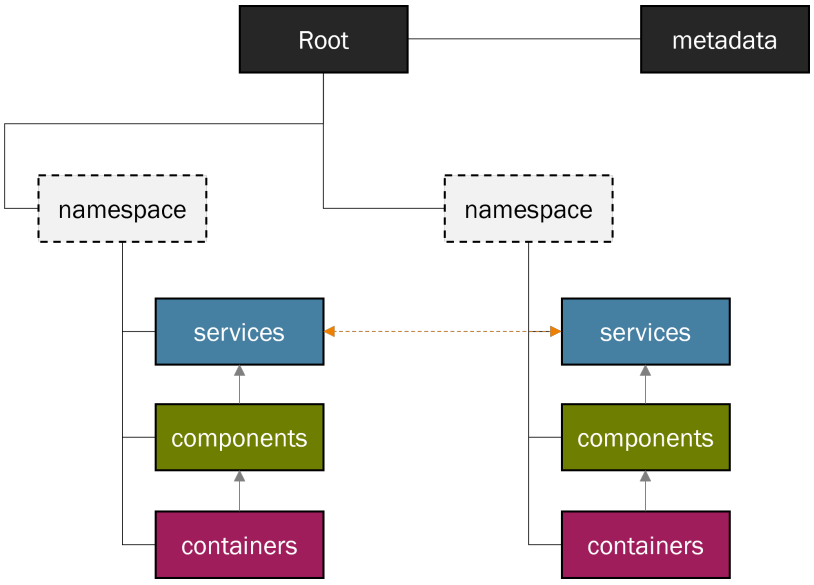

Figure 3.2: Graph based representation

is then used to trace dependencies and relationships. Manifests of `Deployments` and `StatefulSets` must necessarily contain an `"image"` tag, referring to a container image from a repository such as Docker Hub, to spawn the containers that form the pod. This tag is used to generate an SBOM for the image using a standard 3rd party generation tool. The SBOM created has a static `"bom-ref"` of its own, constant for a given container image. The image SBOMs are then referenced within the SBOM entries of the associated pods.

Here, a terminology change is introduced. In its specification CDX provides two principal classes of elements:

1. Services
2. Components

While the former represents a corresponding K8s concept exactly, the latter has no direct analogue. CDX requires components to be assigned a pre-defined `"type"`. In order to comply with the intent behind the enumerated types, StatefulSets and Deployments are classified as `"application"`, while the containers they are fulfilled by are labelled `"container"`. It is noted here that `"container"` is an overloaded type; the SBOMs created by 3rd party tools for container images will also carry the type `"container"`. It was found that of the five features identified in section 3.1.2 this method was capable of the first three, but not the remaining two, i.e. traceability in infrastructure fulfillment, or communication between elements.

3.1.4 Graph Based Representation

Retaining terminology from CDX an intermediate Graph Based Representation (GBR) was created, as given in figure 3.2

The graph was designed to enumerate all resources present within each namespace of the network such that each resource could be connected in the logical hierarchy described in section 3.1.3.

This representation was maintained in memory in order to interact with other modules of the implemented system. Relationships between upper layer and lower layer objects (for example, between a service and the pods exposing it) were represented using a pair of properties:

1. `realized-by` : Mapping an upper layer object to a lower layer object
2. `realizes` : Mapping a lower layer object to an upper layer object

In order to ensure vertical traceability, both objects in an upper layer - lower layer pair would bear the corresponding properties. An example of such a connection represented by the graph is as given in the Listing examples 3.1 and 3.2. As a beneficial side effect, this approach is compliant with the Essential Deployment Metamodel discussed in section 5.1.2.

```
{
    "legacy-lab-svc":{
        "name":"legacy-lab-svc",
        ...
        "realized-by":
          {
            "name":"legacy-lab-statefulset",
          }
    }
}
```

Listing 3.1: Example service with component linkage

```
{
  "legacy-lab-statefulset":{
        "name":"legacy-lab-statefulset",
        ...
        "realizes":
          {
            "name":"legacy-lab-svc",
          }
    }
}
```

Listing 3.2: Example component with service linkage

Enumeration of intranetwork communication

A standard manoeuvre post initial compromise in most cyber attacks is “Lateral Movement”, TA0008 in the MITRE ATT&CK framework [42]. This involves targeting systems within a network through exposed communication links to a compromised system. It is thus informative from a risk assessment perspective to know what communication paths are exposed within the infrastructure.

In section 2.2.3 it was discussed that in its default state a K8s cluster places no restrictions on intra- and inter-cluster communication, except for port closure. Such restrictions may instead be implemented by a suitable CNI. The assumed presence of a CNI that implements NetworkPolicy is believed to be a reasonable one as the alternative, akin to a physical network without a firewall, is anarchy. This was thus chosen as a base to represent communication flow within the cluster. A method was created to interpret K8s standard network policy manifests to :

1. Identify the objects within the cluster that would be affected by a given network policy
2. Identify the connections to other objects that have been explicitly allowed by the policy

By interpreting all network policies affecting a given cluster, it was possible to create a granular set of rules pertaining to each object, and enumerate them within the graph, under the key `"flowrules"`

```
{
    "legacy-lab-svc":{
```

```
        ...
        "flowrules":[
          {
            "name":"np.lab.legacy-lab-svc",
            "flow":"ingress",
            "rule":"allow",
            "components":[
              {
                "name":"minio-deployment",
              }
            ],
            "services":[
              {
                "name":"minio-svc",
              }
            ]
        }
        }
    }
}
```

Listing 3.3: Sample rules enumerated

However, while it was possible to enumerate these rules in the GBR, it was found that in the CDX 1.4 specification, there is no standard method by which to capture this information. The 1.5 specification contains a limited method to capture this, in the context of service communication as `"dataflow"` pertaining to a given service. The information that is captured is presented in the listing 3.4.

```
// Specifies the data flow.
message DataFlow {
  // Specifies the flow direction of the data.
  DataFlowDirection flow = 1;
  // Data classification tags data according to its type, sensitivity, and value
    if altered, stolen, or destroyed.
  string value = 2;
  // Name for the defined data
  optional string name = 3;
  // Short description of the data content and usage
  optional string description = 4;
  // The URI, URL, or SBOM-Link of the components or services the data came in
   from
  repeated string source = 5;
  // The URI, URL, or SBOM-Link of the components or services the data is sent
   to
  repeated string destination = 6;
}
```

Listing 3.4: CycloneDX 1.5 service dataflow description

3.1.5 Handling Design Time vs Run Time Differences

During the requirements gathering process for 3.1, it became apparent that the information pertaining to the images used may only be valid at the beginning of the application's deployment, and there may be significant drift as the application is used. Consider the following flow of events as an example:

1. A system is deployed with a postgres:buster image, with known vulnerabilities that have mitigations in place

2. In order to debug some runtime issues, an engineer wishes to examine container logs, and installs `neovim`
3. The version of `neovim` that ships with Debian Buster is 0.3.4-3, which is affected by `CVE-2019-12735`, a critical remote code execution vulnerability
4. This induces serious discrepancy between the original SBOM and the current running container

This scenario has also been discussed in the documentation of CDX, where the concept of an Operations Bill of Materials (OBOM), i.e. a bill of materials generated for runtime use, is used to track run-time containers. While the CDX standard does not suggest how an OBOM should be generated, most existing tools offer functionality that can be repurposed to perform this, namely file-system scanning. Using this, two methods have been found and demonstrated, namely:

1. Host file-system scanning
2. Remote SBOM generator execution

Both methods come with a set of caveats, explained in sections 3.1.5 and 3.1.5

Host File-system Scanning

Containers running on a given host in the linux system reside in well understood locations. For example, all docker containers reside as directories in `/var/lib/docker/overlay2` with machine-friendly (human-unfriendly) names. Through the `describe` function of the K8s API it is possible to map a running container to its directory and perform a scan of that directory using an SBOM generator of choice. Running such a scan, however, typically requires elevated privileges, and tends to place a higher load on host CPU and memory for the duration of the scan.

Remote SBOM generator execution

Most SBOM generator tools are compiled statically. This allows for simple installation as part of design time by creating a volume mount with the requisite binaries, and adding it to each resource. The K8s API may then be used to

1. Trigger SBOM generation through a shell call
2. Copy the resultant SBOM file to the host

This method makes an implicit assumption that the running containers are equipped with the tar binary required by K8s for its copy command. While the vast majority of images would have tar present, "distroless" images (preferred for their added security, minimal nature, and lack of an interactive shell [63]) may invalidate this assumption. This process does, however, add a component to the SBOM, i.e. the SBOM generator itself, with associated vulnerabilities.

3.1.6 Vulnerability Enumeration

In addition to generating a BOM for a given container or file-system as described in section 3.1.5, most SBOM generation tools also support the enumeration of vulnerabilities in CDX format. Using these, two methods of enumeration were developed for the GBR:

1. Capturing vulnerabilities in the associated container or image CDX SBOMs
2. Storing a list of vulnerabilities within the artifact graph, against the corresponding object

These methods serve different purposes and it is intended that they be used in conjunction. The former method serves as a record of the system, suitable for historical analysis, while the latter implements

K8s Object	ArchiMate Equivalent
Service	Technology Service
Pod	Technology Node
Container	System Software
Image	Technology Artifact
Namespace	Grouping

Table 3.1: ArchiMate equivalents of K8s objects

real-time query of the in-memory GBR.

3.1.7 Rescan Trigger

While periodic scans of running containers as part of OBOM tracking are useful for traceability, some changes of a more ephemeral nature might be missed if they fall between scan intervals. Extending the example from section 3.1.5, if the re-scan interval is 7 days, and the engineer installs `neovim` on day 2, realizes the added vulnerability on day 5 and removes it, there is a 3 day period of increased risk that will not appear on the OBOM history. In order to avoid this problem, an event driven approach is necessary in addition to periodic scans.

Such a system in K8s was implemented by making use of K8s internal audit logs. Disabled by default, audit logs may be configured to record API calls to `kubectl`, the main interface for managing a K8s cluster. Further, the detail at which logs are maintained may be varied based on numerous conditions. For example, API calls that may cause some alteration of the running services or containers, such as `CREATE`, `UPDATE`, `EXEC`, or `EXPOSE` coming from human administrators may be logged at a `"Request-Response"` level, allowing enhanced scrutiny of the cluster's resources.

After configuring the audit logs in such a manner, a scanner was created that continuously monitors the logs generated and triggers a re-scan of a given resource when a suspicious API call has been made. A complication that was observed, which indirectly demonstrated the validity of the technique, was that if the scanner used the remote execution technique, then the re-scan trigger would also appear in the logs, triggering yet another scan. This behaviour was mitigated by creating a whitelist of safe commands (the SBOM scanners), and performing a sub-string match of the API call with the whitelist.

3.1.8 Historical Record Maintenance

For record keeping purposes the module implementing the GBR was augmented with a `history` dictionary field, and a method `snapshot()` that would push the current state of the graph, with the timestamp at the time of scanning acting as key. This method could then be called at the time of re-scan, and the history could be presented and accessed by querying for `history`'s keys.

3.2 Composing an ArchiMate model from the GBR

From the ArchiMate 3.x specification it was determined by the author that all elements of the K8s infrastructure would reside in the Technology layer. Based on the descriptions provided by the 3.x standard, equivalents were selected for the concepts present in K8s as given in table 3.1

From Type	To Type	Relationship
Grouping	All other	Composition
Technology Artifact	System Software	Realization
System Software	Technology Node	Assignment
Technology Node	Technology Service	Realization
Technology Node	Technology Node	Flow
	Technology Service	
Technology Service	Technology Node	
	Technology Service	

Table 3.2: Relationship types employed

It may be noted here that no equivalent was found for the concept of a vulnerability. However, ArchiMate allows for the enrichment of objects with custom properties. This was therefore the chosen approach for vulnerability enumeration.

With these equivalents so selected, the types of ArchiMate relationships required to accurately represent the infrastructure were to be determined. Based on the specification, it was concluded that a subset of allowed relationships would be employed, listed in 3.2

The rationale for selection is as follows:

1. **Composition**: The objects within a group
2. **Realization**:
   (a) **Artifact to Software**: Creation of software entities from abstract technology artifacts
   (b) **Node to Service**: Operation of the abstract service by the node
3. **Assignment**:
   (a) **Software to Node**: Allocation of functional execution of the node's responsibilities by the system software
4. **Flow**: Transfer of information within the network

As the OpenModel Exchange is one of many possible ArchiMate formats the overall method of converting the GBR into ArchiMate representation was divided into two parts:

1. A module that receives the GBR as input and interprets its elements and relationships between them as ArchiMate concepts
2. A handler that receives these ArchiMate concepts and constructs an OpenModel file

3.2.1 ArchiMate interpreter

On receiving the GBR, the keys of the root, except `metadata` were used to reconstruct the namespaces present in the model. These would subsequently be used to create ArchiMate groups. The model was then scanned per namespace in a depth first manner to obtain all objects, i.e. services, components, and containers, present in the model. These objects were then passed to the handler to create in the appropriate format. For each object to be written, the dictionary pertaining to the object from the GBR was passed to the handler.

For a given object, if vulnerability information was found within the entry of a given object, the list of vulnerabilities was iterated over and an entry was created under a `"vulnerability"` property of the corresponding object.

Field	Description
identifier	The unique identifier of the relationship
source	The unique identifier of the source object
target	The unique identifier of the target object
type	The type of ArchiMate relationship to create
source_name	The source object's name
target_name	The target object's name
source_namespace	The source object's namespace
target_namespace	The target object's namespace

Table 3.3: Fields passed to the handler to create relationships

Once all objects were written, the model was walked a second time to write the relationships between objects. Two passes were necessitated by the eventuality that some handlers might require objects to exist before relationships can be made between them, which is discussed further in section 3.2.2. A dictionary with keys as given in table 3.3 was passed from the interpreter to the handler to create a given relationship. This dictionary contains a super-set of all information that could be used to create a given relationship.

3.2.2 OpenModel handler

The ArchiMate OpenModel Exchange format follows the structure given in listing 3.5. It is not required that all fields exist. For example, an exchange file without the views section is still considered valid. The open source programme Archi appears to impose an order restriction, however, requiring the existing sections to be ordered as `elements`, `relationships`, `propertyDefinitions`, and, `views`. Archi, further, does not accept UUID identifiers, instead mandating identifiers of the form `"id-<IDENTIFIER>"` where `<IDENTIFIER>` is a unique 32 or 64 bit hexadecimal.

The OpenModel handler was designed to create this structure on instantiation.

```
<model xmlns="...">
    <name>"..."</name>
    <elements>
        ...
    </elements>
    <relationships>
        ...
    </relationships>
    <propertyDefinitions>
        ...
    </propertyDefinitions>
    <views>
        ...
    </views>
</model>
```

Listing 3.5: OpenModel Exchange Structure

If vulnerability information was found in the GBR, the handler was instructed to create a corresponding property definition as shown in listing 3.6

```
<propertyDefinitions>
    <propertyDefinition identifier="propid-1" type="string">
```

```
      <name xml:lang="en">Vulnerability</name>
    </propertyDefinition>
</propertyDefinitions>
```

Listing 3.6: Vulnerability property definition

ArchiMate objects and relationships created by the interpreter were then appended to the relevant sections as demonstrated in 3.7 and 3.8

```
<elements>
    <element identifier="[some-element-identifier]" xsi:type="Grouping">
      <name xml:lang="en">lab</name>
    </element>
</elements>
```

Listing 3.7: Kubernetes objects in OpenModel Exchange

```
<relationships>

    <relationship identifier="[relationship-identifier]"
     source="[some-element-identifier]"
     target="[other-element-identifier]"
     xsi:type="Composition"/>

</relationships>
```

Listing 3.8: Kubernetes relationships in OpenModel Exchange

Vulnerabilities were then appended to the `properties` section of the relevant objects as shown in listing 3.9.

```
<element identifier="[some-element-identifier]" xsi:type="Node">
  <name xml:lang="en">workstation-statefulset</name>
  <properties>
        <property propertyDefinitionRef="propid-1">
          <value xml:lang="en">CVE-2022-3924</value>
        </property>
  </properties>
```

Listing 3.9: Vulnerability enumeration

The resultant XML generated by the handler was imported as an OpenModel Exchange file in Archi, demonstrated in figure 3.3. It is noted here that by default, no ordering is implemented by Archi; the elements of the graph were arranged manually.

3.2.3 Business Impact Functionality

As described in section 2.6, given an enterprise architecture model, business impact assessment typically evaluates the effect of changes in one part of a network on other parts. When viewed as a graph, this amounts to determining the presence of paths between affected objects, and other objects in the network. Thus business impact use cases were considered:

1. Describing an object, listing objects adjacent to it, and associated vulnerabilities
2. Determining objects affected by a given vulnerability

Using these methods, it would then be possible to get a comprehensive report of all objects that could adversely be affected by a given vulnerability by first querying for directly affected objects, then

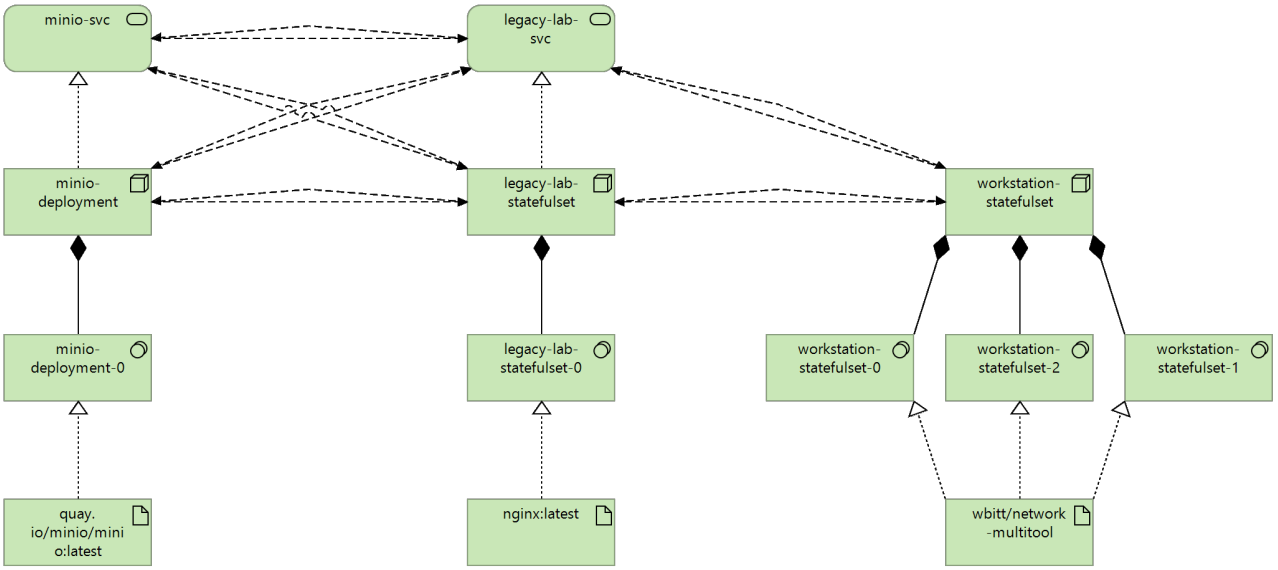

Figure 3.3: Output ArchiMate OpenModel

tracing upward fulfillment paths from those objects to the services or components they fulfil, and finally open communication paths between services.

A module was designed and developed that, given an OpenModel Exchange file, parsed the model and stored information pertaining to the identified use cases. Given an input OpenModel Exchange file, a query for adjacent objects would first find the queried element within the XML using its name. On finding the relevant element, the `element ID` was then extracted, queried for in the relations contained in the XML, and parsed based on the type of the relation given in table 3.2 into categories `"realizes"` and `"realized-by"` for layer positioning, and `"flow"` for communication paths.

Scanning for objects affected by a given vulnerability was performed by iterating over all elements contained in the XML, and further iterating through its list of properties to verify the presence of the input vulnerability. If present, details pertaining to the element in question were appended to a list and returned.

3.3 Implementing IACD using OpenC2

With the in-memory GBR implemented as described in section 3.1.4, and the architecture of OC2 deployments given in section 2.7.1 it was determined that the developed system could be integrated into OC2 by employing a two layer approach, with the top layer implementing an OC2 actuator in Yuuki, and the bottom layer composed of the GBR. Methods were implemented to create a minimum viable actuator, listed in 3.3.2.

Given that a significant set of operations in IACD pertain to controlling the flow of traffic to, from, and within the network, a method was designed to convert pairs of objects within the GBR into valid K8s network policy, thereby explicitly allowing connections between them. This method is described in section 3.3.1. Its extension to IP blocks, while feasible, was not performed in the scope of this implementation.

3.3.1 Dynamic Network Policy Creation

With the behaviour of K8s network policies explained in section 2.2.3, it was determined that default deny all is a necessary prerequisite to implementing programmatic network policy. Further, it was determined that in conjunction with deny all, the creation of atomic network policies explicitly allowing communication between selected components within the network for given traffic types would achieve the objective of allowing intended communication paths without side effects.

A method was designed and developed that creates such atomic network policies given a pair of objects and a traffic flow direction between them, within the GBR. This method would then obtain the relevant `matchLabels` for source and destination, and create a valid network policy. This policy would then be implemented by passing it to the kubectl wrapper's `apply` function to enact it in the infrastructure.

A point of consideration in the implementation of programmatic network policy is the policy name. While policy names should ideally themselves be programmatic in order to identify rules in O(1) time, K8s objects names must adhere to RFC 1123 with a maximum length of 63 characters, regardless of object type. It would thus be possible for a programmatic name pertaining to resources which have longer names to exceed the limit and therefore render the manifest invalid. Thus, it was instead decided to use UUIDs for the network policy names. This results in a lookup complexity of O(n), but guarantees the creation of a network policy for a given pair of objects.

3.3.2 OpenC2 Command Creation

A custom OC2 actuator profile titled `"kubernetes"` was created to interact with the GBR. The list of verbs implemented by this actuator as a proof of concept are given in table 3.4. It is noted here that this is one possible interpretation of the verbs listed by the OC2 standard; it is intended that these verbs be overloaded such that a command producer can perform a wide range of fine-grained actions in using comparatively terse expressions.

Verb	Target	Action
scan	Cluster	Capture the latest status of the cluster. Optionally capture vulnerabilities
query	Object	Return the artifact graph entry of the object
locate	Object or IPv4	Present the object in its logical hierarchy and network information
allow	Object pair	Create a network policy to allow communication between the given object pair for a given direction
deny	Object pair	Delete a network policy between the given object pair
create	Component	Add a resource to the given component
delete	Component	Delete a resource from the given component

Table 3.4: OpenC2 verbs implemented for the K8s actuator

4 Technical Implementation, Evaluation and Results

This chapter concerns itself with the technical aspects of implementing the conceptual approach described in chapter 3. A description of the test infrastructure created to verify the workings of the approach is provided first, along with the software libraries used in the approaches' construction, and their quirks that would need to be considered in the process of implementation. Then, a set of quantitative evaluation criteria were constructed, and the developed method was then compared with the two existing solution implementations `ksoclabs/KBOM` and `kubernetes-sigs/bom`. These existing solutions were also compared with the developed method on a feature basis. Finally, the results of these evaluations are provided

4.1 Technical Implementation

For the purpose of the experiment, a model Infrastructure Under Test (IUT) was created using K8s, given in figure 4.1, meant to simulate an environment with legacy, mission critical, hardware which interacts with some form of cloud storage, and is operated on through a set of computers denoted workstations. Here K8s was chosen for its ease of modelling heterogeneous network topology. The IUT could also be modelled using other IaC frameworks such as OpenStack or Terraform using the appropriate virtual network appliances.

A Virtual Machine (VM) was created with `Ubuntu 22.04` as the base image, with the K8s stack and `kubectl` installed to deploy and manage the infrastructure under test. Python was selected as implementation language for the methods to be created. A single node K8s cluster was instantiated, and code was developed that executes in the VM, thereby effectively running on the K8s control node.

While a standard library Python exists for interacting with K8s environments, it was found that the shell based `kubectl` tool provided greater range and flexibility of expression in querying resources and enacting policy. A wrapper was thus created in python around `kubectl`.

The GBR was constructed using the `dictionary` data structure in python, with K8s namespaces acting as first level keys. A class was constructed around this dictionary to construct the representation given a set of manifest files from the `kubectl` wrapper through a set of steps:

1. Construct the top level keys from the namespaces found in the manifests.
2. Sequentially populate Services, Deployments and StatefulSets, and then Pods as `services`, `components`, and `pods` respectively.
3. Link `services` to `components` based on `matchLabels`. Similarly link `components` to `pods`.
4. Create `flowrules` between elements based on available `NetworkPolicy` manifest files

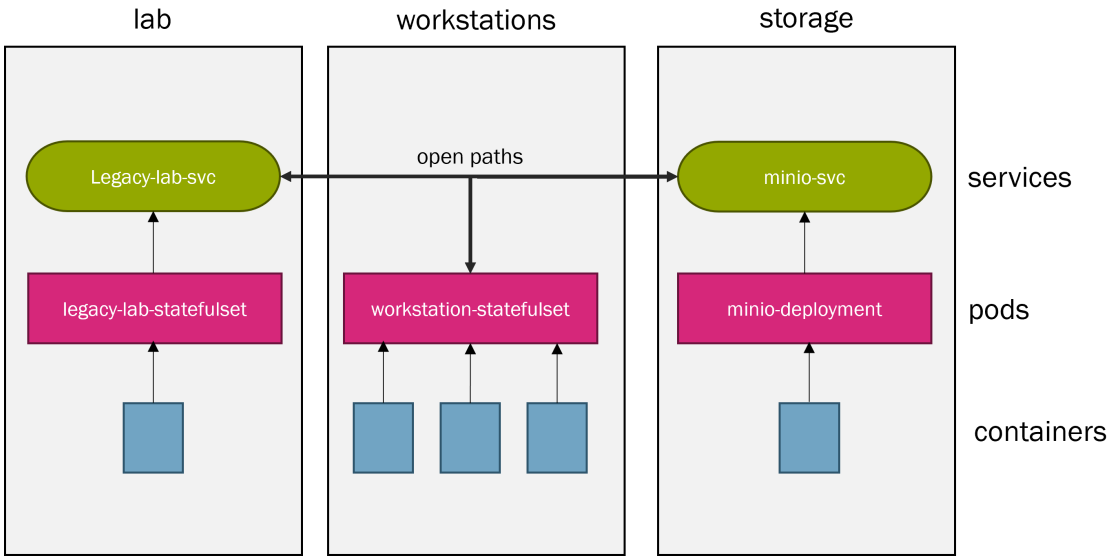

Figure 4.1: Infrastructure Under Test

This class was then extended with a set of functions such as `describe()` and `locate()`, that serve as queries to the GBR forming a base for OC2 interaction.

As discussed in section 2.4, a number of pre-existing tools are available for the generation of SBOMs given a container image, file-system, or file. The combination of Syft and Grype by Anchore has been selected for this purpose, but any third party tool that generates valid CDX SBOMs given a container image name or file system target in JSON format would be compatible. No standard library in Python exists for the creation of a new bill of materials in CDX format. However, CDX may be expressed in JSON, which may be readily constructed using Python dictionaries.

No standard library exists in Python for the creation of ArchiMate models. However, the ArchiMate Open Model Exchange is an XML file with a registered namespace. A library was created to generate ArchiMate models in this format using the standard `LXML` python library. The open source programme "Archi" was used to validate generated open model exchange files. Further, a python module was created to ingest generated ArchiMate models and maintain an XML representation in memory was extended with functions similar to those implemented in the GBR, so as to facilitate Business Impact Assessment.

While the OC2 language is typically expressed in JSON a standard library in Python exists, the OpenC2-Lycan library, to facilitate the construction of OC2 messages. OpenC2-Lycan itself is built upon the Stix2 library. It is noted here that since version 1.0.5 OpenC2-Lycan is incompatible with Stix2 versions higher than 1.3.0.

A consumer-specific Python library also exists, OpenC2 Yuuki, which allows for simplified instantiation of OC2 consumers along with customized actuator profiles. A caveat noted is that while the message bodies accepted by Yuuki are standard OC2, the messages expected by the consumer are not according to the standard implementation.

4.2 Evaluation

A suite of tests was developed to measure the execution time of various aspects of the system developed, as well as compare it with the identified existing solutions and `ksoclabs/KBOM` and

`kubernetes-sigs/bom`. Each test was performed 1000 times, and a 3 sigma treatment was performed for outlier removal. The evaluative experiments performed may be summarized as below:

1. Comparison of execution time between the existing solutions and developed method.
2. Decomposition of the execution time of the developed method to create an SBOM.
3. Measuring the effect of the number of running containers on the method's execution time.
4. Comparison of the execution times of API calls made to the GBR versus their equivalents in the developed ArchiMate model.
5. Comparison of the execution time of image name based SBOM generation versus container runtime generation.

The implemented method was also demonstrated to members of the ASOP consortium, along with a set of interested external parties as part of a larger discussion on efforts to automate security operations. The results obtained from feature and quantitative evaluation are presented in section 4.3. The findings from the consortium discussion are presented in section 5.3.

The time to generate an SBOM for the infrastructure under test was measured for the developed system against those of the existing solutions. For the developed system, the overall time in each iteration was computed as the sum of the times for the steps:

1. Obtaining all manifests of the IUT from `kubectl` via the wrappers created.
2. Constructing the GBR of the infrastructure.
3. Obtaining container SBOMs and constructing the final K8s SBOM.

The effect of increasing infrastructure complexity was measured by varying the number of running containers in the system, from 0 to 50 containers. The effect was measured on 2 parameters, namely the time taken to compose the graph, and the time taken to collect run-time manifests from K8s.

In order to determine the overhead of querying vulnerability information from generated ArchiMate models compared to the GBR, the execution time of `describe()` calls.

The overhead caused by triggering SBOM scans from within running containers was measured by comparing the execution time against obtaining the SBOM using the image name from the host. The number of vulnerabilities listed was then compared for one of the containers using both methods before and after installation of an extraneous package `vim`.

4.3 Results

A summary of the features implemented using the GBR in comparison with the two existing solutions selected, i.e. `ksoclabs/KBOM` and `kubernetes-sigs/bom`, is presented in table 4.1. It may be seen that `ksoclabs/KBOM` only performs the capture of K8s infrastructure in SBOM format, and does not attempt any hierarchical representation of the infrastructure, vulnerability enumeration, or communications tracing. Similarly, the functionality implemented by `kubernetes-sigs/bom` is exclusively vulnerability enumeration through the creation of an SPDX SBOM.

It may be seen from figure 4.2 and table 4.2 that the developed method takes on average 1.4 seconds less than the existing solution `ksoclabs/KBOM` and 5.5 seconds less than `kubernetes-sigs/bom`. While the execution time for the developed method demonstrates a significantly larger standard deviation than `kubernetes-sigs/bom`, its worst case performance is close to 1 second better.

Figure 4.3 shows the time spent by the method in the three steps discussed in section 4.2. It is observed that the creation of the graph based representation causes minimal overhead, in the order of

Feature	ksoclabs/KBOM	kubernetes-sigs/bom	Developed Method
Infrastructure capture	✓	X	✓
Hierarchical enumeration	X	X	✓
Vulnerability enumeration	X	✓	✓
Fulfillment traceability	X	X	✓
Communication traceability	X	X	✓

Table 4.1: Feature comparison between developed method and existing solutions

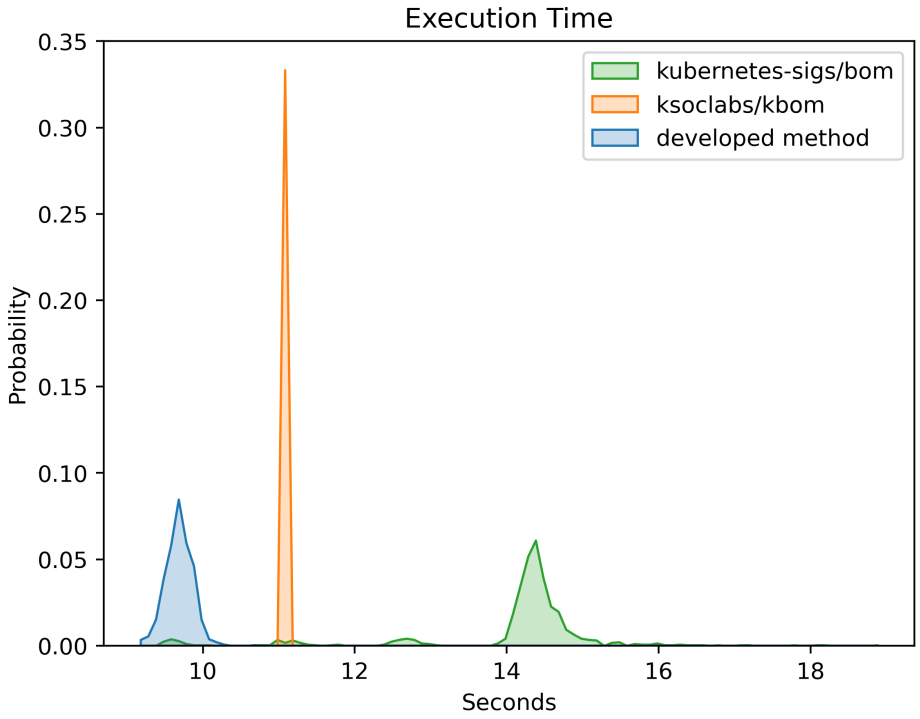

Figure 4.2: Comparison of execution time between developed method and existing solutions

	Mean	Standard Deviation
Method	9.682	0.175
ksoclabs/KBOM	11.073	0.017
kubernetes-sigs/bom	14.103	1.192

Table 4.2: Descriptive statistics of execution time

milliseconds. The single longest activity is the creation of the SBOM itself, with an average of 8.3 seconds per SBOM. Fetching run-time manifests also adds a measurable overhead of 1.25 seconds on average.

Comparing the histograms of execution time taken by the K8s SBOM generator and the set of calls made by it to the underlying image SBOM tool Trivy, it was found that there is a large overlap between the two, suggesting that the conversion of the graph based intermediate to CDX did not add significant overhead to the process of creating the final set of SBOMs. A Z test of the two distributions against a null hypothesis of the means being identical returned a p-value of 0.94, providing evidence to accept the same.

The response of the system to increased network resources was measured in the contexts of graph generation and interaction with the K8s API, as shown in figure 4.5. It may be seen that in both cases there is a reduction in execution time between 0 and 6-7 resources, and a roughly linear increase in execution time up to 30 resources. The reason for higher execution times for low resource counts is unknown. It may be noted that the worst case performance for graph generation is 5 milliseconds.

Figure 4.6 shows that container based scanning results in a 100% overhead compared to creating an SBOM using the image name of the container. Prior to installing the extraneous software both methods reported 75 vulnerabilities. Post installation the reported values diverged with the image scan remaining at 75, while the container scan reported 172 vulnerabilities, an increase of 98. In the interest of completeness, the vulnerability count comparison was also performed for the containers from the other pods, i.e. `legacy-lab-statefulset` and `minio-deployment`. Here it transpired that the vulnerabilities reported in the container scan were less than what was reported in the image. This is likely because of an incorrect set of scanners provided to the SBOM tool, and were thus invalid. For unmodified containers, the vulnerabilities listed in the file-system scan must match that of the image scan.

From figure 4.7 it may be observed that the both API calls exhibit Poissonian behaviour and typically take under 10 milliseconds per call in both cases. A significant difference is observed between the means of the two distributions, with the average execution time for API calls to the graph based representation being 3 milliseconds versus 3.5 milliseconds for the calls to the ArchiMate OpenModel representation.

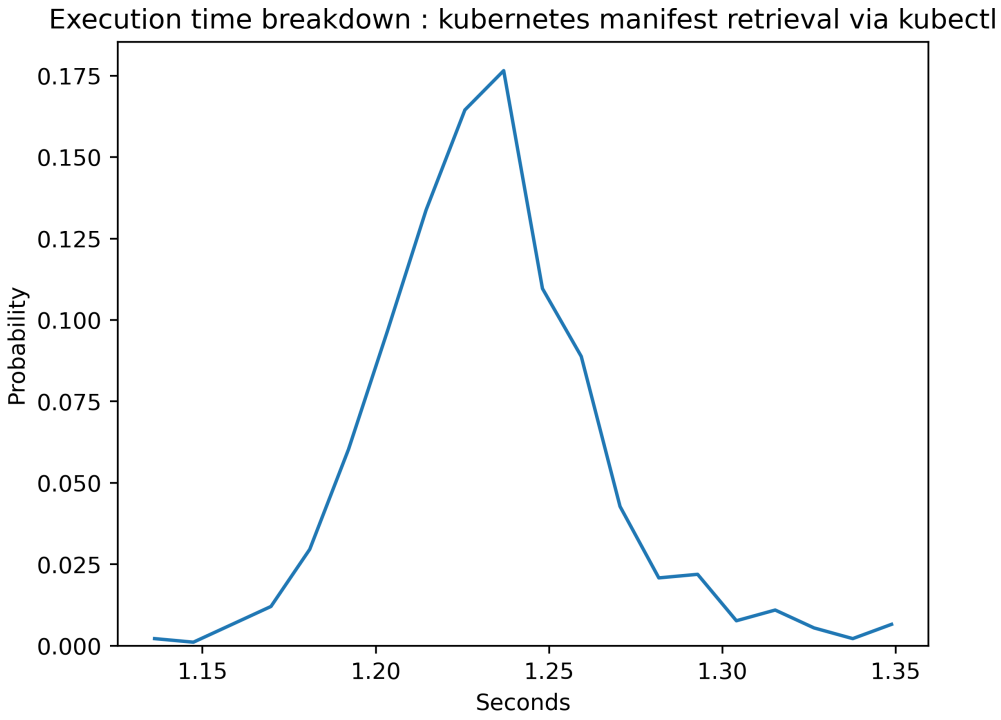

Figure 4.3: Time taken to fetch manifests from `kubectl`

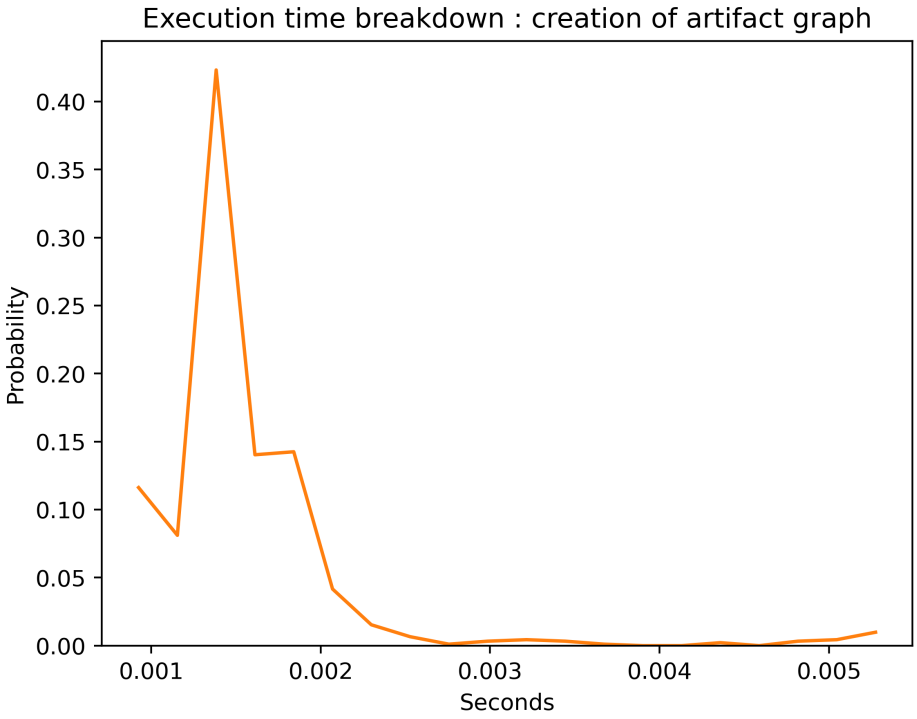

Figure 4.3: Time taken to construct the GBR

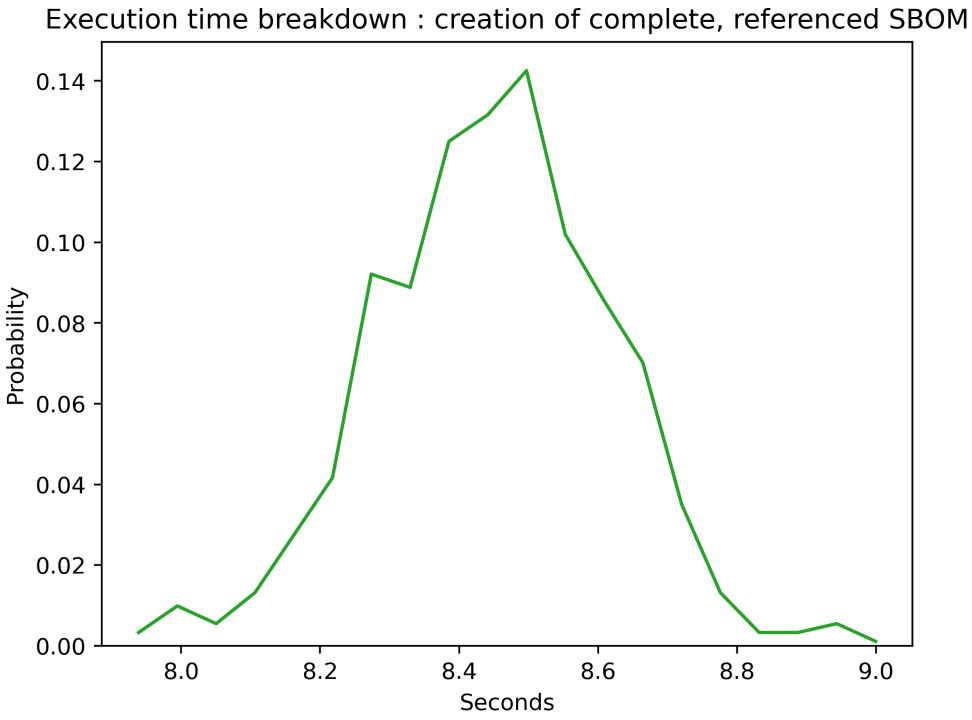

Figure 4.3: Time taken to construct the complete, referenced SBOM

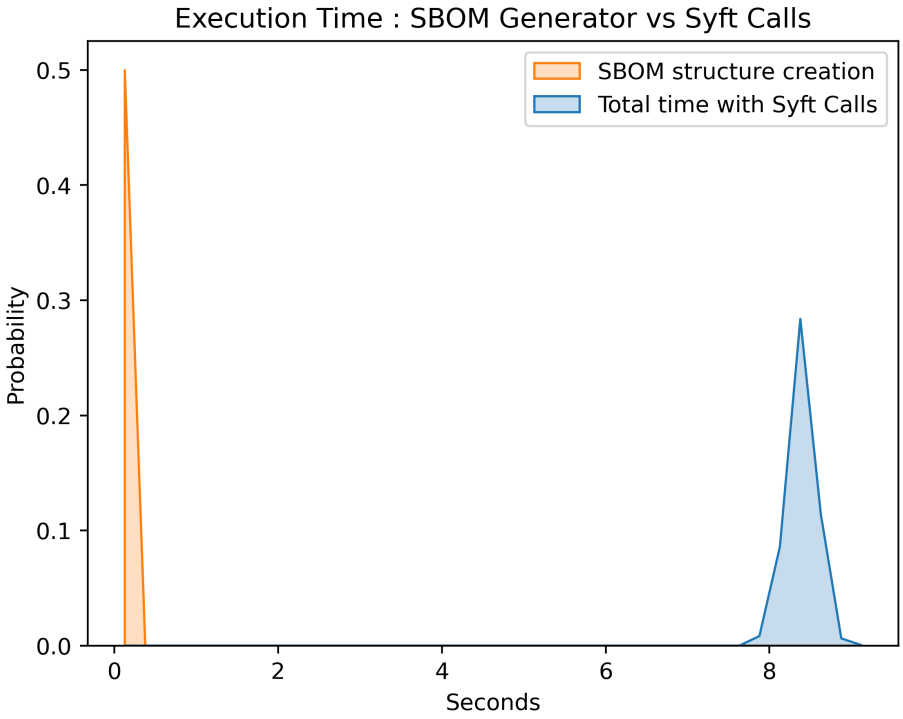

Figure 4.4: Comparison of SBOM generator vs. constituent syft calls

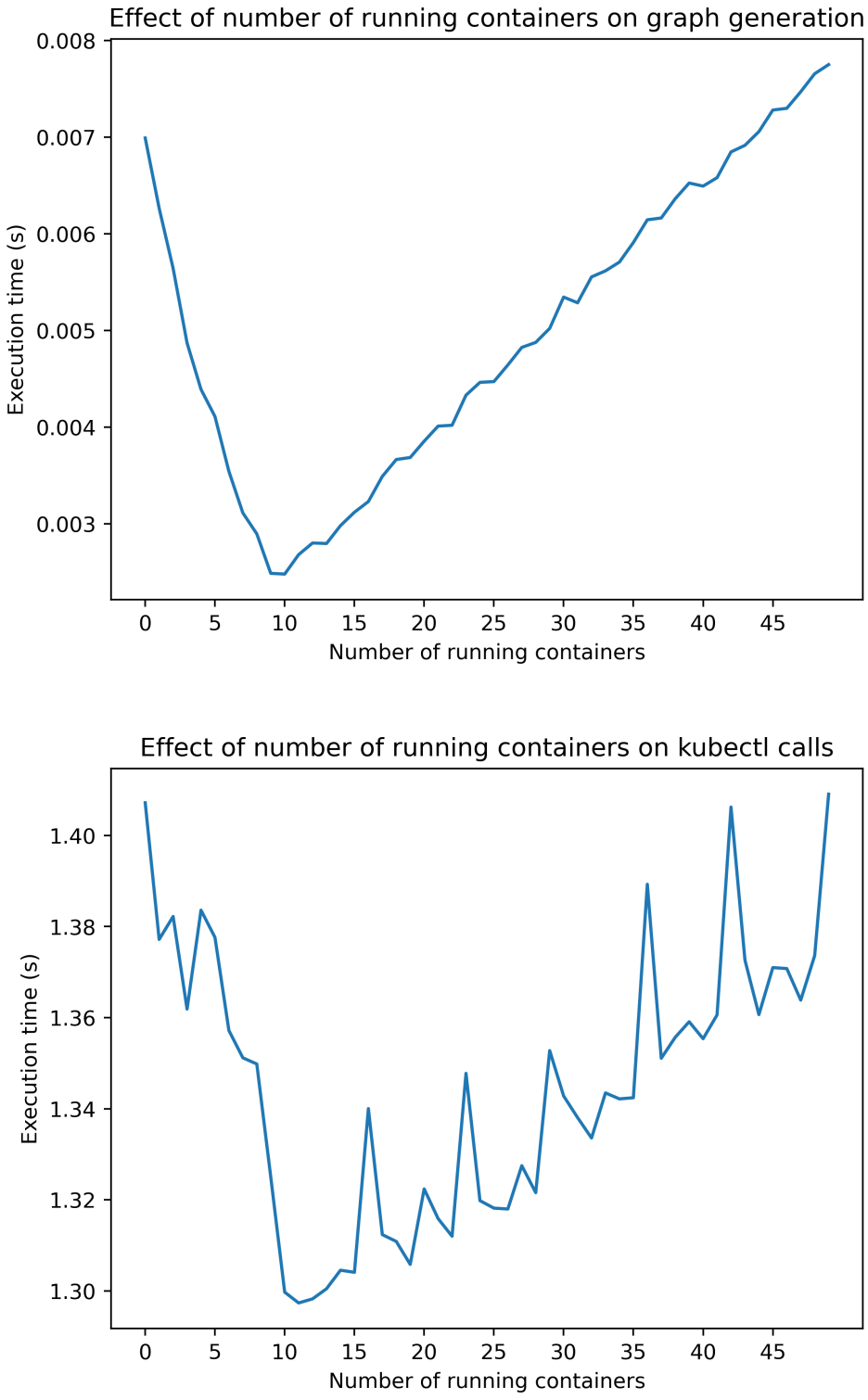

Figure 4.5: Effect of increasing network resources on execution time

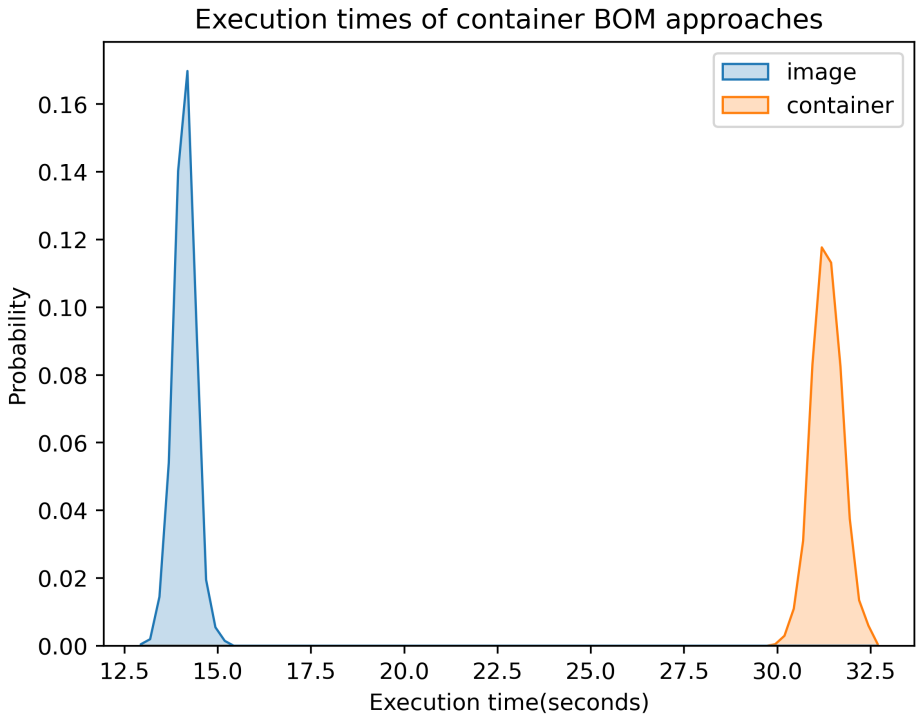

Figure 4.6: Comparison of execution time static image versus running container scanning

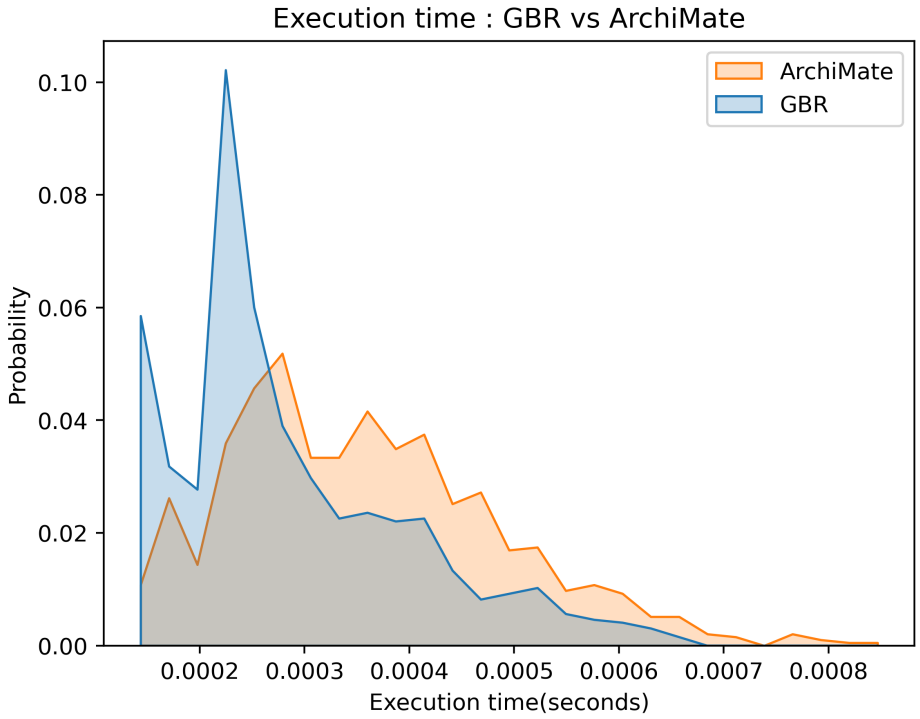

Figure 4.7: Comparison of execution time between Graph and ArchiMate API calls

5 Discussion

This chapter shall provide a brief overview in section 5.1 of the current academic work pertaining to the key concepts discussed. It will then provide a commentary on the results obtained from the experiments performed, and the implementation of the devised method. Findings from the discussion with the larger ASOP consortium will then be presented, followed by a set of identified threats to this work's validity.

5.1 Related Work

This section presents a cross section of current academic work in the fields of Software Bill of Materials, attempts at the creation of an Infrastructure Model for cloud environments, and the challenges faced in the domain of Software Supply Chain Security. Finally, it lists a set of inferences that may be drawn from the review of this work, that influence the conceptual approach presented in chapter 3.

5.1.1 Software Bill of Materials

Muirí [46], on behalf of the NTIA, published a set of requirements that any standard intending to serve as an SBOM would need to fulfil in order to be useful in:

1. Improving vulnerability identification.
2. Improving traceability.
3. Reducing software counterfeiting.
4. Increasing vendor trust.

Summarizing the list of requirements, a minimum viable SBOM must contain:

1. Baseline component information.
2. Component relationships.
3. Authentication.

The first two requirements were also listed by Camp and Andalibi[8] in their exploration of the minimal SBOM. Here, baseline component includes, but is not limited to, the component name, the supplier name, a unique identifier (preferably a version 4 or version 5 UUID / RFC 4122 identifier), a version string, and a cryptographic hash.

Stating that an SBOM does not, and should not exist in isolation, the component relationships serve to create a Directed Acyclic Graph of between the elements of a given piece of software, where each relationship asserts some form of composition between an upstream and a downstream component. Muirí categorized the possible relationships that a given software component may have with other components as:

1. Unknown - A default state where no relationship is known between the given component and any others.
2. Root - The given component has no upstream dependencies.
3. Partial - At least one upstream component exists, but the complete list of upstream dependencies is not known.
4. Known - The full set of upstream dependencies is known.

At the time of publication, Muirí commented that no single standard had been developed with the specific intention of addressing SBOM use cases. However, they identified two standards that met the required criteria, namely SWID, an XML-based standard adapted by NIST, and SPDX, backed by the Linux Foundation. Camp and Andalibi[8] identify a third standard,CycloneDX, backed by OWASP.

Camp et al., in their comment, also provided a set of risks that in their view would materialize upon widespread adoption of SBOMs . A summary of these risks has been provided below.

1. Current standards allow for self-attestation without verification of SBOMs by the organizations that deliver the software under description.
2. SBOMs are not made available in a standardized, accessible manner.
3. The hashing algorithms supported, both for the BOM as a whole as well as its individual components are not collision free. The insecure[67, 58] SHA-1 algorithm has been mandatory for component hashes in SPDX, while it is accepted as valid, but not required, in CycloneDX.

Some of these risks have had remediation efforts since the publication of Camp et al.'s paper. SigStore[48] proposed by Newman et al. and backed by the Linux Foundation and the Open Source Security Foundation aims to simplify the processes of signing and verification of open source software. SigStore's reference implementation, Cosign, allows for signing container images uploaded to Docker Hub, a major repository, with a single command. Signature verification is more involved, requiring the user to know the certificate publisher's identity and OIDC issuer in advance.

Despite these shortcomings, the concept of an SBOM has gained significant traction due to its utility. The Linux Foundation's 2022 report [22] on the state of SBOMs showed that 47% of the organizations it surveyed already employed SBOMs in practice. A helpful factor would have been Executive Order 14028 from the President of the United States of America[23] that made it a requirement for any enterprise that provides software solutions to the United States Federal Government to make available an SBOM for a given product. A similar proposal has been drafted by the European Commission[11]. While Xia et al. [74] found that most industry experts believe that a large percentage of open source projects do not have SBOMs it is conjectured that the situation will improve with two major political blocs mandating their generation.

5.1.2 Infrastructure Modelling and Representation

Due to the heterogenity of cloud infrastructures, in terms of architecture, scope, and function, a number of attempts have been made to devise some form of technology and vendor agnostic abstraction, usually with differing objectives. For example Martino et al. [38] devised the Object Design Ontology Layer (ODOL) and the Ontology Web Language (OWL) to express patterns in the composition of services in the cloud. Similarly, Oberle et al. [50] attempted to create a language for service description aimed at technology agnostic service modelling. Veselý et al. [64] evaluated a number of existing tools aimed at representation and modelling of virtual network infrastructure.

A number of open source standards have also been published that aim to allow for vendor agnostic specification of cloud artifacts, the most significant among them being OASIS TOSCA. The Open-

Stack Heat format is TOSCA compatible [24]. Platform specific standards also exist in the industry, such as AWS CloudFront, Terraform, and Cloudify. These will however not be explored in detail in the context of this thesis by virtue of their specificity being antithetical to the goal of a vendor agnostic cloud infrastructure representation. The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an XML based modelling language created with three objectives in mind, summarized by Brogi et al.[6] as:

1. Automated application deployment and management.
2. Application portability.
3. Interoperability and component reusability.

In order to simplify the process of modelling TOSCA based applicaitons Kopp et al. [30] developed Winery, a web based tool for the creation of TOSCA CSARs.

Saatkamp et al. [54] have demonstrated TOSCA's extensibility to the domain of threat modelling and subsequent control of virtual network resources and functions.

Wurster has made significant contributions toward adapting the TOSCA language for industry use. They had identified that, though studied extensively in academic circles, TOSCA sees little adoption in the industry [73].

Through a systematic analysis of the major cloud deployment models available in the industry they created the Essential Deployment Metamodel (EDMM) [71], containing the set of concepts that fulfil a complete semantic mapping to all declarative deployment models.

Using this metamodel they constructed TOSCA Light [72], a minimal TOSCA specification, with the aim of reconciling the disparity between the TOSCA standard's view of cloud computing and the industry's implementations. Finally, they created TOSCA Lightning [73], integrating Winery to create a tool that is capable of converting TOSCA Light representations into deployment specific artifacts. However, it appears that development of TOSCA Lightning has ceased as of 2021, suggesting that it may not have seen more mainstream adoption.

5.1.3 Software Supply Chain Security

A number of high profile cyber-attacks in recent history have underscored the difficulty in securing large scale software pipelines due to their large attack surface, the increased time to discovery for severe vulnerabilities and simultaneously their reduced time to exploitation.

The SolarWinds hack, raised in chapter 1 as a motivating example, was dissected by Willett [68]. Over the course of several months, beginning at least as early as October 2019, attackers infiltrated SolarWinds' systems, compromising their update pipeline in order to push a malicious version of the company's Orion IT network monitoring solution with valid signatures, infecting over 18,000 of the organization's clients. Willett posits that SolarWinds as an organization likely did not follow strong security practices, allowing for the hack to be possible in the first place. He also highlighted the relative rarity of large-scale state sponsored cyber-attacks, stating that the majority of attackers are significantly less sophisticated, and that adhering to standard security best practices significantly reduces the overall risk of a breach.

The 2017 Equifax hack is another often used example of a software supply chain attack. Wang and Johnson [66] present a timeline of the attack, highlighting the potentially extremely short time-to-exploit for vulnerabilities. On 8th March 2017 US-CERT announced the discovery of CVE-2017-5638 affecting Apache Struts 2. Attackers exploited this vulnerability 2 days later, performed lateral

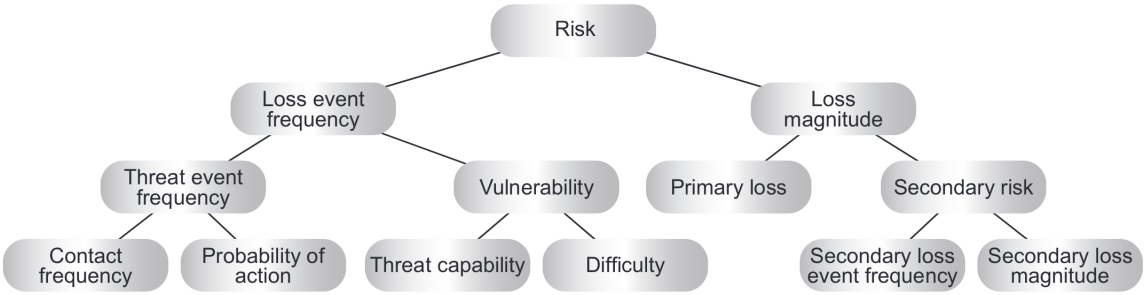

Figure 5.1: FAIR Risk Taxonomy [19]

movement tactics undetected, and had access to Personally Identifiable Information (PII) by 13th May. Equifax first detected unusual traffic on their network on 29th July, but by then 2.4 million American citizens had been affected.

Feutrill et al. [16] found that between 2017 and 2018 the median time to population for CVSS metrics rose from 1 day to 19 days, while the median time to exploit for a vulnerability during the same period shrank from 296 days to 6 days.

Reed, Miller, and Popick [53] performed a comprehensive analysis of supply chain attack patterns, codifying a number of standard attacks, and mapping them across the phases in the attack life-cycle. They also provided a set of countermeasures, along with their expected risk reductions. Among the countermeasures deemed to have significant impact are:

1. Maximize visibility into all supply chain tiers.
2. Restrict traffic on supply chain networks.
3. Establish pedigree/provenance across the supply chain.
4. Require (good) cryptographic techniques for authentication.
5. Closely monitor and regulate the software update process.

Miller's position as a chief engineer at the MITRE corporation at the time of publication likely aided the creation of the MITRE ATT&CK [42] and DEF&ND[43] frameworks, which present expanded sets of attack tactics, classified by goal, and similarly classified countermeasures.

With the intent of simplifying the process of determining and communicating information security risks Freund and Jones [19] devised the FAIR approach, a framework to arrive at temporally-bound probabilities of adverse events. In order to facilitate computing risk, what was thought to be difficult to meaningfully evaluate, they decomposed the notion of risk into "loss event frequency" and "loss magnitude". Successive decomposition led to them presenting a risk taxonomy as presented in figure 5.1, which was then used to formulate a framework to arrive at a quantitatively oriented risk mitigation strategy. Wang et al. proposed an improvement to the quantitative techniques used in the FAIR framework, using Bayesian Networks in combination with the Monte Carlo method suggested in the original implementation.

Hubbard and Seiersen [25] further codified this by proposing an an algorithm and model-centric approach to arrive at risk estimates, finding that machine driven techniques frequently arrive at more accurate estimates for risk than human expert opinion. They too proposed using Bayesian techniques to arrive at more accurate quantitative estimates for loss event frequency and loss estimates.

Band et al.[2] proposed a means of performing risk analysis using the ArchiMate language, and demonstrated the same using a case study.

Moore et al. [44] found that most private sector organizations are increasingly concerned about information security, and are correspondingly increasing their spending on security. They found that the key bottleneck in implementing security operations is no longer funds but skilled personnel to enact security policy. Among Chief Information Security Officers, the use of frameworks and tools to formulate decision making is relatively high, the key motivating factors being perceived risk reduction and compliance.

Boone [5] states in no uncertain terms that cyber-security must be a core part of organizational strategy at the executive level, citing exploding attack surface and omnipresent connectivity as motivating factors.

On the other hand, Haney and Lutters [21] find that cyber-security advocates must carefully formulate their communication in order to elicit action from executive leadership. They find that three complaints need to be overcome in order to generate decisive action: “it’s scary”, “it’s confusing”, and “it’s dull”, implying that:

1. The communication must be discerning in the risks portrayed, demonstrating what is at stake without inducing a sense of helplessness.
2. The risks, actions, and consequences should be explained without invoking too much technical detail.

Islam et al. [28] found that a lack of automation is still a significant barrier; routine activities constitute a large part of SOC engineer workloads. Due to the large number of differing infrastructure standards, uniform monitoring of network resources is a challenge. A lack of run-time modelling tools similarly limits team effectiveness. A lack of standardization between the interfaces of security tool vendors further cripples automated operations.

5.1.4 Inferences Drawn

A number of findings from the related work inform the direction of exploration shown in chapter 3. Section 5.1.2 leads us to conclude that:

1. While a number of infrastructure modelling standards may exist, TOSCA appears to be the most mature, open, vendor agnostic standard.
2. However, despite TOSCA Light and associated implementations, uptake by the industry is low.

The governmental pressure to adopt SBOMs discussed in 5.1.1 suggests that SBOMs may present a viable alternative, provided that they can implement Wurster’s EDMM.

From section 5.1.3, it may be inferred that

1. Basic security monitoring and hygiene significantly reduces attack risk and potential severity
2. Security tools have to focus on automation, standardization, and integration with modelling tools in order to reduce engineer load
3. In order to get executive buy-in, the potential impact of security incidents must be clearly communicated

This implies that even though SBOMs will likely see increased adoption, it may be beneficial to create a layer of abstraction over the raw data provided by them, and attempt to translate vulnerability and weakness information into business impact terms, suggesting a translation chain from network infrastructure to SBOM to enterprise architecture models such as ArchiMate. While this visibility is beneficial, it is most effective when paired with a means of automating response.

5.2 Comments on implementation and results

The work presented in this thesis demonstrates the following novel concepts:

1. An approach to represent cloud infrastructure defined in K8s in a hierarchical manner using Software Bill of Materials approaches, and existing SBOM tools.
2. A means of leveraging existing K8s network policy to enhance observability and impact assessment.
3. A means of representing K8s cloud infrastructure artifacts in the ArchiMate language to integrate with business impact assessment.
4. An OC2 adaptor for K8s infrastructure

Some aspects of the method implemented warrant deeper discussion, such as the choice of the Graph Based Representation, its performance against the chosen existing solutions, and operational findings through the course of implementation and experimentation.

5.2.1 Necessity of the graph based approach

The direct interpretation of K8s YAML files achieved limited success. It was found that while it was possible to construct a valid CDX SBOM that performed criteria 1 and 2 identified in section 3.1.2, i.e. capturing infrastructure and expressing it hierarchically, it was not possible to perform 3 and 4, as this would require context to be retained across files. Thus, direct conversion was deemed unsuitable.

While the graph based approach required two steps to compose a valid SBOM, as opposed to one step in the direct approach, it was found to satisfy all four evaluation criteria. As a result of this approach the process of creating the SBOM was decoupled from the process of scanning the cluster, leading to a modular system.

Further, it was found on comparing the output of the graph based approach against the two existing solutions that only the developed method composed a hierarchical SBOM, where services were linked to their fulfilling components, and these components were further linked to the images declared, and the containers they would eventually be composed of. In the view of the author this is a key feature of the SBOM format that allows for the representation of arbitrarily complex infrastructure in an efficient manner, lending itself well to expressing the logical hierarchy created by IaC deployments such as K8s from the services they expose to the running containers. This also helps stay true to the goal of any Bill of Materials format i.e. traceability from the raw materials to the finished product via a number of intermediate steps.

5.2.2 Comparison of the developed method against existing solutions

It may be seen from section 4.3 that the feature-set supported by the method developed is superior to both existing solutions, save for the ability to scan the complete cluster including control plane nodes, network fabric et al. While this is technically supported, it was disabled in code by creating a blacklist of control plane namespaces, thereby limiting graph creation to those namespaces in the Infrastructure Under Test. The feature disparity is due to the fact that both existing solutions were developed with the goal of BOM generation purely for reporting, and not for the goals identified in section 1.1.

The existing solution `ksoclabs/KBOM` demonstrates a very low variance in execution time, with a near deterministic performance. This is very likely attributable to the fact that it exclusively queries

the K8s API and represents the artifacts returned by it as components. Thus, while the image names used in the cluster may be seen within the BOM generated by this tool, no vulnerability or package information is provided for it. Further, it is not currently possible for third party SBOM solutions to parse the output to provide this information.

The other existing solution, `kubernetes-sigs/bom`, performs a similar reporting by default; it was only possible to obtain software package and vulnerability information by supplying the container image names to the tool instead of the cluster name. The high execution time and variance for this tool may possibly be attributed to the fact that the scanning implementation is ostensibly custom made. The developed method likely benefited greatly from reusing existing state of the art methods for retrieving the relevant SBOMs.

5.2.3 Performance of the graph based method

From the results in section 4.3 it may be inferred that the use of a graph based intermediate causes no significant overhead in the generation of the infrastructure's SBOM. Given that the graph creation time is in the order of milliseconds, it can be said that the technique's advantage discussed in section 5.2.1 comes at no significant cost. It must be noted though that this inference may only apply to the developed method which:

1. Makes extensive use of Python's `dictionary` data structure, which may prove more difficult to scale development in production environments
2. Runs entirely on the K8s control node, which may not be preferable in production environments.

The results presented in section 4.3 lead to a similar inference for the creation of the SBOM from the graph based representation, with an average execution time under 1 ms excluding the calls to the external tool in order to fetch image SBOMs. This suggests that in production, the cost of generating a new snapshot SBOM is insignificant, provided a database is maintained for the image and container SBOMs.

5.2.4 Scanning approaches

When correctly configured, the process of scanning running containers is likely to capture more accurate vulnerability information than simply generating a scan using the given image name. Still, the significant overhead, over 100%, seen for the former as latter might diminish its suitability to production environments. Based on the results obtained, it may ostensibly be prudent to instead minimize the mutability of the containers by:

1. Using scratch containers containing only the minimum required binaries for operation.
2. Have default deny all network policies in place to ensure that unregulated update operations cannot be performed using the shell.
3. Disable access to the containers' shells altogether.

Even so, immutability is not guaranteed. For example the containers of the Cilium network fabric allow access to shells. A proposed hybrid solution would be to enforce these policies, and have container based rescans only on access to the shell as described in section 3.1.7.

5.3 Findings from consortium discussion

The consensus obtained from the consortium discussion in section 4.2 are presented below,

1. The architecture of the developed method is good and serves as a suitable proof of concept for automated security, but would need further effort in order to reach maturity
2. It is important that a standard for representation, once made or selected , is committed to and promoted. The decision to rely on open source standards will be helpful in this regard
3. diversity and heterogenity of infrastructure present in production systems would complicate modelling efforts due to a lack of discoverability
4. This lack of discoverability may be mitigated through integration with existing tools, which requires greater involvement from major infrastructure vendors.
5. The integration with BIA systems is both warranted and useful. It would, however, place maturity requirements on enterprises that hope to adopt this manner of system that would need to be met by the enterprises themselves.
6. Without this maturity being present, it is unlikely that an organization would have the necessary information to convert the impact analysis performed in the techology layer into meaningful risk assessment at the business or enterprise levels.
7. The effect of vulnerabilities presented at increased levels of abstraction would be interesting from a security expert's perspective, especially the effect across multiple organizations

5.4 Threats to validity

While the methods developed in the process of answering the research questions outlined appear promising, a few caveats to be noted are presented as below.

5.4.1 Results of Performance Evaluation

All evaluation experiments were performed with a local K8s cluster and the developed method running on the control node. This combination is unrepresentative of IaC deployments used in practice, which can span entire countries. The results therefore present a far more optimistic scenario than might be seen in such real world conditions. Still, the obtained results suggest that the GBR and SBOM generation methods will not add significant overhead to existing workloads.

5.4.2 SBOM Integrity

Lin[37] et al. experimented with CDX and DependencyTrack to automate the process of licence compliance and vulnerability tracking. They state that while the system was able to track and analyze SBOMs in real time as expected, it was possible to for an attacker to alter an SBOM in order to avoid detection. At the time of writing, Lin's work was not available publicly, and thus it was not possible to evaluate the method by which such alterations were performed.

While the system described in 3 makes use of a few countermeasures to detect alteration in running containers, it does not perform such checks on the SBOMs themselves once they have been written to database. A more mature version of this implementation should extend the runtime audit principle to the SBOM database and restrict API privileges to ensure immutability.

It is still possible for a sophisticated attacker to compromise the system by creating a spurious SBOM generator, employing techniques similar to the software supply chain attacks discussed in order to prime targets by masking some critical vulnerabilities. While far fetched, it is not outside the realm of possibility given the increased prevalence of Advanced Persistent Threats. There is thus no substitute for constant vigilance.

5.4.3 SBOM as a representation

The goal of this thesis was to explore the possibility of SBOM creating a vendor and technology agnostic network representation. While this was technically achieved, it can be argued that the SBOM is not the representation in itself, rather a manifestation. The true representation, it could be argued, is the GBR created to represent the network and its features. The CDX SBOM generated would then simply be a translation of the graph into a standardized format.

5.4.4 Utility of ArchiMate models in Business Impact Analysis

In the scope of this thesis two methods were developed to help identify objects affected by a given vulnerability and enumerate neighbours of a given object. In conjunction, it is possible to trace a path of objects that could be adversely affected by a given event in an automated manner. However, this implementation is limited to the Technology layer. Unless the services and components created from the graph are linked to their counterparts in the Application and Business layers it will not be possible to perform such an analysis at these levels. Such linkage would typically need to be performed manually by an enterprise architect on consultation with the teams responsible for the business function and infrastructure.

Further, while these methods allow for identification of affected objects, they cannot provide any information pertaining to the risks posed. While querying the NIST CVE database for vulnerability score might provide partial information, the criticality of the resources a given vulnerability may affect is not machine-derivable. Similarly, the quantified potential impact of a given vulnerability, for example in days of lost revenue, or penalties from class action lawsuits, cannot be derived programmatically and need to be estimated by human experts. Thus, the methods developed are of limited utility, in that they can only facilitate human-driven business impact analysis, not replace it.

5.4.5 More cloud more problems

This thesis demonstrated the possibility of representing a network defined in K8s YAML in CDX. While it is widely adopted and most production loads would likely make use of it even if running on public cloud infrastructure such as AWS or Azure, it is far from the only standard. A cloud environment deployed using Terraform or Pulumi for example would have a significantly different structure. This was not explored in the context of this thesis, and further research would be required to determine the suitability of this method to other IaC tools. It is expected that while some representations such as Terraform or OpenStack Heat may be able to use the concepts directly, others such as Pulumi may require reimplementation or be limited to interacting with information provided by the API server.

6 Conclusions and Future Work

This section provides a set of concise answers to the research questions proposed in section 1.1, based on the work presented in this thesis. It shall then speculate on possible avenues for extending the work presented.

6.1 Conclusions

From the evaluation of the developed method and its demonstration to the focus group the following conclusions may be drawn against the research questions identified in section 1.1:

RQ1. Can Software Bill of Materials based techniques be used to describe Infrastructure as Code deployments, providing information such as Common Vulnerability Enumeration (CVE) entries affecting a given component?

Yes, by creating a graph based representation of an infrastructure as code deployment, a method to convert this graph into a Bill of Materials, and using existing SBOM enumeration tools it is possible to completely describe a given deployment including vulnerability information, communication flow, and service fulfillment.

RQ2. If such a technique is possible, can it be used to integrate with the existing open-source enterprise architecture systems?

No, using the SBOM generated, it is not possible to integrate this technique with existing enterprise architecture systems. It is however possible, and straightforward, to integrate the graph based representation to enterprise architecture systems.

(a) Can such a technique be used to facilitate business impact assessment?

Partially: it is possible to identify dependencies between components, and realization paths, to obtain the set of objects that would be affected by compromise of a given object in the deployment, but this information is binary in nature. Comprehensive Business Impact Assessment requires information regarding the business importance of a given asset along with risk information. While the latter can be obtained by querying the appropriate databases, e.g. MITRE, business importance cannot be inferred technologically, and must be assigned by the enterprise architect.

RQ3. If such a technique is possible, can it facilitate security orchestration by integrating with existing orchestration systems?

Similar to the answer to RQ2, **no**, using the SBOM generated, it is not possible to integrate this technique with existing orchestration systems. It is however straightforward to integrate the graph based representation.

(a) Can such a technique be used to enact security policy, or security instructions?

Yes, it has been demonstrated that allowed communication paths within the infrastructure may be altered programmatically via OpenC2 commands.

6.2 Future work

During the formulation of the research questions to be answered, it was envisioned that the work undertaken in this thesis form a stepping stone to further development towards programmable infrastructure security. A number of extensions of this work made themselves evident at the time of implementation, and have been summarized below.

6.2.1 Extension to other formats

In order to maximize the likelihood of the developed system being adopted in practice, it is essential that it be applicable to a wide variety of infrastructure formats, not tied down to a single format, i.e. K8s. This can be done by:

1. Surveying the structures of other infrastructure formats, and creating a common set of concepts to build a technology agnostic graphical representation.
2. Building adaptors to represent these infrastructures in the derived common graph format.
3. Creating OC2 actuators that translate commands issued to the graph into the requisite infrastructure specific calls.

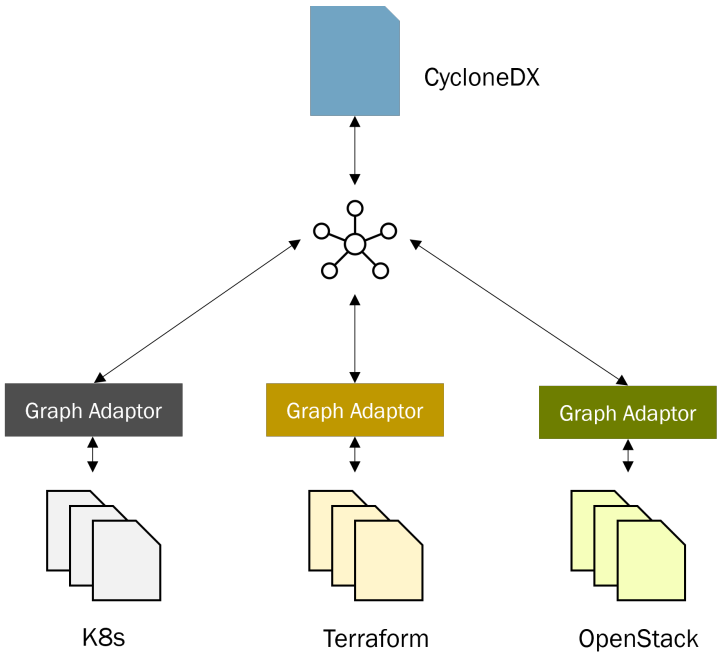

Figure 6.1: CycloneDX as a possible technology agnostic Infrastructure as Code representation

6.2.2 Using a Graph Database

Given the potential development challenges the implemented dictionary-based approach may pose, it would be interesting to compare the performance of this method with a more production-friendly

graph database method. While such a comparison was considered, it was concluded that reimplementation would not be feasible in the time remaining.

It is expected that while populating and querying such a model implemented in a graph database might cause some overhead, it would not significantly affect the overall execution time to generate a bill of materials given that the process that forms the bottleneck is the existing SBOM tool.

These steps are trivial to list, but would require significant effort in order to actually implement. The author hopes that the work presented in this thesis may simplify the process. A possible shortcut may be to create an adaptor for the Essential Deployment Metamodel, leveraging the work of Wurster[71].

6.2.3 Integrating other infrastructure security tools

Vulnerabilities represent only one aspect of infrastructure security. The developed graph based representation may be used to capture arbitrary information about the underlying infrastructure. One potential extension may be the integration of a secret scanning tool such as `trufflehog`. In the existing implementation, a `flags` field was created for each service and component in the cluster, where objects not affected by any network policy for a given flow type were marked as `"exposed_ingress"` or `"exposed_egress"`. It would be straightforward to create a module that would enrich the graph with flags for exposed access credentials based on the analysis from `trufflehog`. In such a manner it would be possible to maintain a single graph based representation that receives inputs from a number of such modules, each scanning for a specific weakness or antipattern.

6.2.4 Petitioning SBOM Standards bodies to adopt extensions

Through the course of implementation it became apparent that while CDX supports a number of the features desired to capture IaC deployments there is still scope for improvement. A number of properties that were created to impart traceability between objects within the infrastructure, for example `realizes` and `realized-by`, could be incorporated into the core specification. The list of component types may also be expanded to support the objects typically used in IaC deployments. For example, the component type is `"container"` both for a container image such as `nginx:latest`, as well as that of a running container deploying this image. While the CDX specification makes no distinction between these two concepts, in the domain of programmable infrastructure they differ in meaning, making the terminology confusing for all parties involved. Explicit types such as `"container_image"` and `"container_runtime"` would swiftly fix this problem.

It was also found that the main competitor format of CDX, SPDX, while ostensibly feature equivalent for traditional BOM use cases, lacks the extensibility inherent in the CDX format. The lack of a custom `"properties"` tag precluded its use for any application outside of what has already been defined in the specification. Further, the now published CDX 1.5 specification has in-built structures to handle the flow of data in a system, which was used in the developed method to determine adjacent systems. This disparity may eventually prove to be an existential threat to the standard, as CDX increases the variety of systems it can capture within a single standard. Petitioning for the inclusion of such features and future-proofing may help the standard remain relevant.

Bibliography

[1] Arushi Arora, Virginia Wright, and Christina Garman. "Strengthening the Security of Operational Technology: Understanding Contemporary Bill of Materials". en. In: *Journal of Critical Infrastructure Policy* 3.1 (June 2022). ISSN: 26933101. DOI: 10.18278/jcip.3.1.8. URL: https://www.jcip1.org/strengthening-the-security-of-operational-technology-understanding-contemporary-bill-of-materials.html (visited on 03/02/2023).

[2] Iver Band et al. "Modeling enterprise risk management and security with the ArchiMate language". In: (Jan. 2015).

[3] Sean Barnum. *Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX™)*. 2014.

[4] Henk Birkholz et al. *Concise Software Identification Tags*. Request for Comments RFC 9393. Num Pages: 61. Internet Engineering Task Force, June 2023. DOI: 10.17487/RFC9393. URL: https://datatracker.ietf.org/doc/rfc9393 (visited on 08/10/2023).

[5] Adam Boone. "Cyber-security must be a C-suite priority". en. In: *Computer Fraud & Security* 2017.2 (Feb. 2017), pp. 13–15. ISSN: 1361-3723. DOI: 10.1016/S1361-3723(17)30015-5. URL: https://www.sciencedirect.com/science/article/pii/S1361372317300155 (visited on 08/07/2023).

[6] Antonio Brogi, Jacopo Soldani, and PengWei Wang. "TOSCA in a Nutshell: Promises and Perspectives". en. In: *Service-Oriented and Cloud Computing*. Ed. by Massimo Villari, Wolf Zimmermann, and Kung-Kiu Lau. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 171–186. ISBN: 978-3-662-44879-3. DOI: 10.1007/978-3-662-44879-3_13.

[7] Drew Buttner and Bob Martin. "THE CYBERSECURITY BENEFITS OF LEVERAGING A SOFTWARE BILL OF MATERIALS". en. In: 22 ().

[8] L Jean Camp and Vafa Andalibi. "SBoM Vulnerability Assessment & Corresponding Requirements". en. In: ().

[9] Carmen Carrión. "Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges". In: *ACM Computing Surveys* 55.7 (Dec. 2022), 138:1–138:37. ISSN: 0360-0300. DOI: 10.1145/3539606. URL: https://doi.org/10.1145/3539606 (visited on 08/19/2023).

[10] Russel Clarke, David Dorwin, and Rob Nash. *Is open source software more secure?* 2009.

[11] European Commission. *Cyber Resilience Act — Shaping Europe's digital future*. en. Sept. 2022. URL: https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act (visited on 05/03/2023).

[12] B K Done et al. "Toward a Capability-Based Architecture for Cyberspace Defense". en. In: ().

[13] Ken Doughty. *Business Continuity Planning: Protecting Your Organization's Life*. CRC Press LLC, 2001.

[14] Ruian Duan et al. "Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages". en. In: *Proceedings 2021 Network and Distributed System Security Symposium*. Virtual: Internet Society, 2021. ISBN: 978-1-891562-66-2. DOI: 10.14722/ndss.2021.23055. URL: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_1B-1_23055_paper.pdf (visited on 02/10/2023).

[15] Robert J Ellison et al. "Evaluating and Mitigating Software Supply Chain Security Risks". en. In: (2010).

[16] Andrew Feutrill et al. "The Effect of Common Vulnerability Scoring System Metrics on Vulnerability Exploit Delay". In: *2018 Sixth International Symposium on Computing and Networking (CANDAR)*. ISSN: 2379-1896. Nov. 2018, pp. 1–10. DOI: 10.1109/CANDAR.2018.00009.

[17] Linux Foundation. "SPDX 1.0 Specification". en. In: (2011).

[18] Linux Foundation. *SPDX 2.3.0 Specification*. 2022. URL: https://spdx.github.io/spdx-spec/v2.3/ (visited on 05/03/2023).

[19] Jack Freund and Jack Jones. *Measuring and Managing Information Risk: A FAIR Approach*. USA: Butterworth-Heinemann, July 2014. ISBN: 978-0-12-799932-6.

[20] The Open Group. *ArchiMate® 3.2 Specification*. 2022. URL: https://pubs.opengroup.org/architecture/archimate3-doc/.

[21] Julie M. Haney and Wayne G. Lutters. ""It's {Scary...It's} {Confusing...It's} Dull": How Cybersecurity Advocates Overcome Negative Perceptions of Security". en. In: 2018, pp. 411–425. ISBN: 978-1-939133-10-6. URL: https://www.usenix.org/conference/soups2018/presentation/haney-perceptions (visited on 08/11/2023).

[22] Stephen Hendrick. "Software Bill of Materials (SBOM) and Cybersecurity Readiness". en. In: ().

[23] White House. *Executive Order on Improving the Nation's Cybersecurity*. en-US. May 2021. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/ (visited on 05/03/2023).

[24] Michael Howard. *Terraform – Automating Infrastructure as a Service*. arXiv:2205.10676 [cs]. May 2022. DOI: 10.48550/arXiv.2205.10676. URL: http://arxiv.org/abs/2205.10676 (visited on 08/19/2023).

[25] Douglas W. Hubbard and Richard Seiersen. *How to Measure Anything in Cybersecurity Risk*. en. Google-Books-ID: AwD0BgAAQBAJ. John Wiley & Sons, July 2016. ISBN: 978-1-119-08529-4.

[26] Michael Hüttermann. "Infrastructure as Code". en. In: *DevOps for Developers*. Ed. by Michael Hüttermann. Berkeley, CA: Apress, 2012, pp. 135–156. ISBN: 978-1-4302-4570-4. DOI: 10.1007/978-1-4302-4570-4_9. URL: https://doi.org/10.1007/978-1-4302-4570-4_9 (visited on 05/01/2023).

[27] IBM. *Cost of a data breach 2022*. en-us. Mar. 2023. URL: https://www.ibm.com/reports/data-breach (visited on 03/21/2023).

[28] Chadni Islam, M. Ali Babar, and Surya Nepal. "A Multi-Vocal Review of Security Orchestration". In: *ACM Computing Surveys* 52.2 (Mar. 2020). arXiv:2002.09190 [cs], pp. 1–45. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3305268. URL: http://arxiv.org/abs/2002.09190 (visited on 08/11/2023).

[29] Geerten van de Kaa and Henk J. de Vries. "Factors for winning format battles: A comparative case study". en. In: *Technological Forecasting and Social Change* 91 (Feb. 2015), pp. 222–235. ISSN: 0040-1625. DOI: 10.1016/j.techfore.2014.02.019. URL: https://www.sciencedirect.com/science/article/pii/S0040162514000833 (visited on 05/03/2023).

[30] Oliver Kopp et al. "Winery – A Modeling Tool for TOSCA-Based Cloud Applications". en. In: *Service-Oriented Computing*. Ed. by Samik Basu et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 700–704. ISBN: 978-3-642-45005-1. DOI: 10.1007/978-3-642-45005-1_64.

[31] Amelie Koran et al. "The Cases for Using the SBOMs We Build". en. In: ().

[32] Kubernetes. *Kubernetes Documentation*. 2022. URL: https://kubernetes.io/docs/home/.

[33] kubernetes.io. *Kubernetes - Production-Grade Container Orchestration*. en. URL: https://kubernetes.io/ (visited on 08/15/2023).

[34] *Labels and Selectors*. en. Section: docs. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/ (visited on 07/11/2023).

[35] M. M. Lankhorst, H. A. Proper, and H. Jonkers. "The Anatomy of the ArchiMate Language". en. In: *International Journal of Information System Modeling and Design (IJISMD)* 1.1 (2010). Publisher: IGI Global, pp. 1–32. ISSN: 1947-8186. DOI: 10.4018/jismd.2010092301. URL: https://www.igi-global.com/gateway/article/www.igi-global.com/gateway/article/40951 (visited on 02/16/2023).

[36] M. M. Lankhorst, H. A. Proper, and H. Jonkers. "The Architecture of the ArchiMate Language". en. In: *Enterprise, Business-Process and Information Systems Modeling*. Ed. by Terry Halpin et al. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 2009, pp. 367–380. ISBN: 978-3-642-01862-6. DOI: 10.1007/978-3-642-01862-6_30.

[37] Lu Lin. "Generating Software Bill of Material for Vulnerability Management and License Compliance". en. In: (Jan. 2023). Accepted: 2023-01-29T18:07:06Z. URL: https://aaltodoc.aalto.fi:443/handle/123456789/119386 (visited on 05/03/2023).

[38] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. "Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Vol. 2. Dec. 2013, pp. 182–187. DOI: 10.1109/CloudCom.2013.123.

[39] Vasileios Mavroeidis and Joe Brule. "A nonproprietary language for the command and control of cyber defenses – OpenC2". en. In: *Computers & Security* 97 (Oct. 2020), p. 101999. ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.101999. URL: https://www.sciencedirect.com/science/article/pii/S0167404820302728 (visited on 02/16/2023).

[40] Vasileios Mavroeidis and Mateusz Zych. *Cybersecurity Playbook Sharing with STIX 2.1*. arXiv:2203.04136 [cs]. Aug. 2022. DOI: 10.48550/arXiv.2203.04136. URL: http://arxiv.org/abs/2203.04136 (visited on 08/14/2023).

[41] Francesco Minna et al. “Understanding the Security Implications of Kubernetes Networking”. en. In: *IEEE Security & Privacy* 19.5 (Sept. 2021), pp. 46–56. ISSN: 1540-7993, 1558-4046. DOI: 10.1109/MSEC.2021.3094726. URL: https://ieeexplore.ieee.org/document/9497237/ (visited on 02/14/2023).

[42] Mitre. *MITRE ATT&CK®*. 2020. URL: https://attack.mitre.org/ (visited on 05/17/2023).

[43] *MITRE D3FEND Knowledge Graph*. en. 2022. URL: https://d3fend.mitre.org/ (visited on 08/19/2023).

[44] Tyler Moore, Scott Dynes, and Frederick R Chang. “Identifying How Firms Manage Cybersecurity Investment”. en. In: ().

[45] “More than 20,000 U.S. organizations compromised through Microsoft flaw”. en. In: *Reuters* (Mar. 2021). URL: https://www.reuters.com/article/us-usa-cyber-microsoft-idUSKBN2AX23U (visited on 08/16/2023).

[46] Éamonn Ó Muirí. “Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM)”. en. In: ().

[47] Randall Munroe. *How Standards Proliferate*. 2011. URL: https://xkcd.com/927/ (visited on 07/07/2023).

[48] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. “Sigstore: Software Signing for Everybody”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 2353–2367. ISBN: 978-1-4503-9450-5. DOI: 10.1145/3548606.3560596. URL: https://dl.acm.org/doi/10.1145/3548606.3560596 (visited on 05/03/2023).

[49] OASIS. *Open Command and Control (OpenC2) Language Specification Version 1.0*. 2019. URL: https://docs.oasis-open.org/openc2/oc2ls/v1.0/oc2ls-v1.0.html.

[50] Daniel Oberle et al. “A unified description language for human to automated services”. en. In: *Information Systems* 38.1 (Mar. 2013), pp. 155–181. ISSN: 0306-4379. DOI: 10.1016/j.is.2012.06.004. URL: https://www.sciencedirect.com/science/article/pii/S0306437912000968 (visited on 07/28/2023).

[51] Marc Ohm et al. “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks”. en. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Clémentine Maurice et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 23–43. ISBN: 978-3-030-52683-2. DOI: 10.1007/978-3-030-52683-2_2.

[52] Shixiong Qi, Sameer G. Kulkarni, and K. K. Ramakrishnan. “Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability”. In: *IEEE Transactions on Network and Service Management* 18.1 (Mar. 2021). Conference Name: IEEE Transactions on Network and Service Management, pp. 656–671. ISSN: 1932-4537. DOI: 10.1109/TNSM.2020.3047545.

[53] Melinda Reed, John F Miller, and Paul Popick. “Supply Chain Attack Patterns: Framework and Catalog”. en. In: ().

[54] Karoline Saatkamp et al. “Application Threat Modeling and Automated VNF Selection for Mitigation using TOSCA”. In: *2019 International Conference on Networked Systems (NetSys)*. Mar. 2019, pp. 1–6. DOI: 10.1109/NetSys.2019.8854524.

[55] Md Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices". In: Sept. 2020, pp. 58–64. DOI: 10.1109/SecDev45635.2020.00025.

[56] Adnan Shaout and Cassandra Dusute. "WHERE DID GENERAL MOTORS GO WRONG WITH THE IGNITION SWITCH RECALL?" en. In: *IIUM Engineering Journal* 15.2 (Nov. 2014). Number: 2. ISSN: 2289-7860. DOI: 10.31436/iiumej.v15i2.522. URL: https://journals.iium.edu.my/ejournal/index.php/iiumej/article/view/522 (visited on 05/19/2023).

[57] *SolarWinds: How Russian spies hacked the Justice, State, Treasury, Energy and Commerce Departments - CBS News*. en-US. July 2021. URL: https://www.cbsnews.com/news/solarwinds-hack-russia-cyberattack-60-minutes-2021-07-04/ (visited on 08/16/2023).

[58] Marc Stevens et al. "The First Collision for Full SHA-1". en. In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 570–596. ISBN: 978-3-319-63688-7. DOI: 10.1007/978-3-319-63688-7_19.

[59] Andrei Tchernykh et al. "Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability". en. In: *Journal of Computational Science* 36 (Sept. 2019), p. 100581. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2016.11.011. URL: https://www.sciencedirect.com/science/article/pii/S1877750316303878 (visited on 08/07/2023).

[60] *The Detroit News*. en. URL: http://www.detroitnews.com/story/business/autos/general-motors/2015/08/24/gm-ignition-fund-completes-review/32287697/ (visited on 05/19/2023).

[61] K Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms*. 1992.

[62] tno. *Automated Security for a secure digital economy — TNO*. en. 2023. URL: https://www.tno.nl/en/digital/digital-innovations/trusted-ict/automated-security-digital-economy/ (visited on 08/25/2023).

[63] Google Container Tools. *"Distroless" Container Images*. original-date: 2017-04-18T22:02:38Z. Aug. 2023. URL: https://github.com/GoogleContainerTools/distroless (visited on 08/17/2023).

[64] Peter Veselý, Vincent Karovič, and Vincent Karovič ml. "Tools for Modeling Exemplary Network Infrastructures". en. In: *Procedia Computer Science*. The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops 98 (Jan. 2016), pp. 174–181. ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.09.028. URL: https://www.sciencedirect.com/science/article/pii/S1877050916321573 (visited on 08/07/2023).

[65] Manfred Vielberth et al. "Security Operations Center: A Systematic Study and Open Challenges". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 227756–227779. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3045514.

[66] Ping Wang and Christopher Johnson. "CYBERSECURITY INCIDENT HANDLING: A CASE STUDY OF THE EQUIFAX DATA BREACH". en. In: *Issues In Information Systems* (2018). ISSN: 15297314. DOI: 10.48009/3_iis_2018_150-159. URL: https://iacis.org/iis/2018/3_iis_2018_150-159.pdf (visited on 02/21/2023).

[67] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding Collisions in the Full SHA-1". en. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 17–36. ISBN: 978-3-540-31870-5. DOI: 10.1007/11535218_2.

[68] Marcus Willett. "Lessons of the SolarWinds Hack". In: *Survival* 63.2 (Mar. 2021). Publisher: Routledge _eprint: https://doi.org/10.1080/00396338.2021.1906001, pp. 7–26. ISSN: 0039-6338. DOI: 10.1080/00396338.2021.1906001. URL: https://doi.org/10.1080/00396338.2021.1906001 (visited on 05/01/2023).

[69] Chris Williams and Editor in Chief. *How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript*. en. URL: https://www.theregister.com/2016/03/23/npm_left_pad_chaos/ (visited on 05/19/2023).

[70] Reinder Wolthuis and Frank Fransen. "SOCCRATES - Real-time threat, impact analysis and response automation for SOC/CSIRT operations". en. In: ().

[71] Michael Wurster et al. "The essential deployment metamodel: a systematic review of deployment automation technologies". en. In: *SICS Software-Intensive Cyber-Physical Systems* 35.1 (Aug. 2020), pp. 63–75. ISSN: 2524-8529. DOI: 10.1007/s00450-019-00412-x. URL: https://doi.org/10.1007/s00450-019-00412-x (visited on 08/14/2023).

[72] Michael Wurster et al. "TOSCA Light: Bridging the Gap between the TOSCA Specification and Production-ready Deployment Technologies:" en. In: *Proceedings of the 10th International Conference on Cloud Computing and Services Science*. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2020, pp. 216–226. ISBN: 978-989-758-424-4. DOI: 10.5220/0009794302160226. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0009794302160226 (visited on 08/14/2023).

[73] Michael Wurster et al. "TOSCA Lightning: An Integrated Toolchain for Transforming TOSCA Light into Production-Ready Deployment Technologies". en. In: *Advanced Information Systems Engineering*. Ed. by Nicolas Herbaut and Marcello La Rosa. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2020, pp. 138–146. ISBN: 978-3-030-58135-0. DOI: 10.1007/978-3-030-58135-0_12.

[74] Boming Xia et al. "An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead". In: (2023). URL: https://arxiv.org/abs/2301.05362 (visited on 03/02/2023).

Acronyms

ASOP Automated Security Operations. 4, 33, 40

BIA Business Impact Assessment. 6, 14, 19, 32, 47

BOM Bill of Materials. 11, 24, 41, 45, 46, 51

BPMN Business Process Modelling Notation. 15

CACAO Collaborative Automated Course of Action Operations. 17

CDX CycloneDX. 6, 12–14, 21, 23, 24, 32, 35, 41, 45, 47, 48, 50, 51

CI/CD Continuous Integration / Continuous Deployment. 3, 6

CNI Container Network Interface. 9, 22

CSAR Cloud Service ARchive. 42

CVE Common Vulnerability Enumeration. 1, 5, 48, 49

CVSS Common Vulnerability Scoring System. 43

DAG Directed Acyclic Graph. 11

EDMM Essential Deployment Metamodel. 42, 44

GBR Graph Based Representation. 1, 20, 21, 23–27, 29–33, 36, 45, 47, 48

IaC Infrastructure as Code. 1, 2, 4–7, 14, 19, 31, 45, 47, 48, 50, 51

IACD Integrated Active Cyber Defense. 6, 17, 29

ICT Information and Communications Technology. 4

IoT Internet of Things. 2

IUT Infrastructure Under Test. 31, 33

JSON JavaScript Object Notation. 13, 17, 32

K8s Kubernetes. 6–9, 19–22, 24, 25, 29–31, 33, 35, 45–48, 50

MTTD Mean Time to Discovery. 3

OBOM Operations Bill of Materials. 24, 25

OC2 OpenC2. 6, 17–19, 29, 30, 32, 45, 50

ODOL Object Design Ontology Layer. 41

OWL Ontology Web Language. 41

PII Personally Identifiable Information. 43

REST REpresentational State Transfer. 7

SBOM Software Bill of Materials. 1, 3–6, 11–14, 19–21, 24, 25, 32, 33, 35, 37, 40, 41, 44–49, 51

SPDX Software Package Data eXchange. 12, 14, 33

STIX Structured Threat Information Exchange. 17

SWID Software Identifier. 12

TNO De Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek. 4

TOGAF The Open Group Architecture Framework. 15

TOSCA Topology and Orchestration Specification for Cloud Applications. 41, 42

UML Universal Modelling Language. 15

UUID Universally Unique Identifier. 13, 20, 27, 30, 40

VM Virtual Machine. 31

XML eXtensible Markup Language. 13, 16, 28, 29, 32, 42

YAML YAML Ain't Markup Language. 7, 20, 45, 48