FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automating an Open-Source Security Operations Center: Deploying, Updating and Scaling

César Alves Nogueira

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Ricardo Morla

July 29, 2022

Automating an Open-Source Security Operations Center: Deploying, Updating and Scaling

César Alves Nogueira

Mestrado em Engenharia Informática e Computação

July 29, 2022

Abstract

Security Operations Centers (SOCs) are getting more widely adopted in enterprises, but much time is wasted configuring, deploying, and maintaining these systems. Development Operations (DevOps) techniques should be able to solve this problem by providing the necessary automation, thus saving the engineers responsible for the SOC valuable time. Timeliness is of utmost importance in security as businesses face a constant risk of attack. We believe bringing DevOps to the SOC would enable security professionals to more easily maintain its components, provide patches faster, in a seamless and standard manner, and scale resources as necessary.

In this work, we propose an approach using DevOps techniques for managing a SOC. In particular, we developed a proof-of-concept Python module that allows to program various use cases of the SOC. The tested SOC components include Elasticsearch and ModSecurity running on Docker, for a reproducible and testable environment.

Specific implemented use cases include deployment, ugrading/downgrading (migrating existing data or deleting it) and scaling of each component. Expected difficulties going forward include the expansion of the module to include additional components as the diversity of available SOC components and each of their intricacies complicate the creation of a standard solution for all of them.

Further work can be developed on top of this module such as supporting other components of the SOC or adding functionality to the existing ones but the main takeaways of this work are the possible benefits and limitations of automating the SOC deployment process, whether using Python, Docker or any other technology that allows for the programming of the SOC.

Resumo

Os Centros de Operações de Segurança (SOCs) têm vindo a ser cada vez mais amplamente adotados nas empresas, contudo é gasto bastante tempo nos processos de configuração, *deployment* e manutenção destes sistemas. Operações de Desenvolvimento (DevOps) deverão ser capazes de resolver este problema ao fornecerem a automação necessária, permitindo aos engenheiros responsáveis pelo SOC um melhor aproveitamento do seu tempo. A eficiência temporal revela-se de extrema importância no que concerne a segurança, na medida em que os negócios enfrentam um risco constante de ataque. Ao trazer DevOps para o SOC, será possível permitir aos profissionais de segurança que mais facilmente façam a manutenção das suas componentes, forneçam *patches* mais rapidamente, com qualidade e de forma estandardizada, e ajustem os recursos de acordo com o necessário.

Neste trabalho, propomos uma abordagem utilizando técnicas de DevOps para gerir um SOC. Em particular, desenvolvemos um módulo em Python como prova de conceito que permite programar vários casos de uso para o SOC. Os componentes do SOC testados incluem o Elasticsearch e o ModSecurity a correr em Docker, resultando num ambiente reprodutível e testável.

Casos de uso específicos implementados incluem *deployment*, atualização/desatualização (migrando dados existentes ou apagando-os) e escalabilidade de cada componente. Dificuldades esperadas incluem a expansão do módulo para incluir componentes adicionais visto que a diversidade de componentes do SOC disponíveis e cada uma das suas complexidades complicam a criação de uma solução padrão para todos eles.

Trabalho posterior pode ser desenvolvido por cima deste módulo, como suportar outros componentes do SOC ou adicionar funcionalidade ao existente mas o mais importante a retirar deste trabalho são os possíveis benefícios e limitações vindos da automação do processo de *deployment* do SOC, quer seja utilizando Python, Docker ou qualquer outra tecnologia que permita a programação do SOC.

Acknowledgements

First of all, I would like to leave a big thank you to my supervisor, Ricardo Morla, for all the support throughout these last few months. Thank you for the help in developing this work and for always pushing me to do better. I would not have been able to do this without you.

I also want to thank my friends for all the fun times we've spent together that made this a little less difficult. I wish them a lot of success in their future endeavours and that we may keep having fun like always.

I thank my family, and in particular my parents and sister, for supporting me in this journey that was college and always believing in me.

Finally, I want to thank my girlfriend for always being there for me when I needed and when I didn't, and for being my closest friend.

César Nogueira

"Without the element of enjoyment, it is not worth trying to excel at anything."

Magnus Carlsen

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AV	Antivirus
CI/CD	Continuous Integration and Continuous Development
CTI	Cyber Threat Intelligence
EDR	Endpoint Detection and Response
FW	Firewall
HIDS	Host Intrusion Detection System
IDS	Intrusion Detection System
IOC	Indicator of Compromise
IPS	Intrusion Prevention System
IR	Incident Response
NOC	Network Operations Center
SDN	Software-Defined Network
SIEM	Security Information and Event Management
SME	Small and Medium-Sized Enterprises
SOC	Security Operations Center
VM	Virtual Machine
VPN	Virtual Private Network
WAF	Web Application Firewall

Chapter 1

Introduction

The first chapter of this dissertation includes a short introduction to the issue of Security Operations Centers (SOC) and their deployment. The context and motivation behind this dissertation are delineated and briefly summarized in a problem description, and then the objectives, contributions and document structure are presented.

1.1 Context and Motivation

Data breaches, malware infections, and cyberattacks are common occurrences for organizations. As technology evolves, cyberattacks get more sophisticated and present new threats to enterprises. Security Operations Centers (SOC) are complex structures that have the objective of enhancing an organization's security, such that many enterprises have adopted these kinds of systems as a way to increase protection against those kinds of attacks. However, deploying a SOC is still a very complex and manual process, requiring lots of planning. There is no standard solution for building a SOC, and the diversity of available SOC components makes it hard to create one.

In other sectors, DevOps has presented itself as a solution, as it has proven effective in deploying other types of diverse infrastructures in a standardized manner. This indicates the possibility of DevOps solving similar problems on the SOC.

1.2 Problem Description

There are various reasons behind why SOC have required such elaborate implementation processes. We may begin by addressing the challenge of the SOC having many different components, each one with its own specificities, adding to the need for every SOC to be carefully designed and manually deployed. Along with that, it becomes troublesome to keep track of any changes made to the SOC, for example, when updating or adding new components to the infrastructure. Together with the fact that one faces a lack of standardized SOC deployment testing, all of these constitute serious challenges to the standardization of the deployment process as a whole.

1.3 Objectives

The main goal behind this dissertation is to analyze the outcome of applying a DevOps approach to the SOC, better characterizing and evaluating the benefits that arise from this combination, as well as identifying which parts of the DevOps process constitute either boosts or bottlenecks to a quick, effortless SOC deployment. With this ambition in mind, a prototype SOC deployment will be developed and tested incrementally, using different components and DevOps techniques.

1.4 Contributions

The main contributions of this dissertation are:

- Implementation of a module for automatically deploying a SOC, capable of managing versions and scaling components, and expandable for other components and use cases.
- Analysis of the advantages of automating a SOC and possible improvements to current approach.

1.5 Document Structure

Apart from this introduction, this document is divided into the following chapters: Chapter 2, *Literature Review*, where relevant technology is explored regarding Security Operations Centers, DevOps and Infrastructure-as-Code; Chapter 3, *SOC Architecture and Deployment Requirements*, where a SOC architecture is presented and the purpose of each components explained along with requirements for its deployment; Chapter 4, *Development Methodology*, where a proof-of-concept developed module is discussed; Chapter 5, *Use Cases*, where developed use cases are presented and it's explained how other use cases can be added; Chapter 6, *Conclusions and Future Work*, where current work and results are summarized and possible future work paths are presented.

Chapter 2

Literature Review

This chapter first presents some background on Security Operations Centers (SOCs), how they can be modeled and their composition. It then focuses on DevOps for other types of infrastructure besides the SOC and, more specifically, Infrastructure-as-Code (IaC). Finally, it analyzes some uses of DevOps for the SOC and examples of other automated SOCs.

2.1 Security Operations Centers

The concept of a Security Operations Center appears in a context where cyberattacks that compromise an organization's assets are becoming ever more frequent. In order to face this growing threat, one needs to build a team dedicated to recognizing, adequately dealing with, and further avoiding such incidents. As stated by Vielberth, "The Security Operations Center represents an organizational aspect of a security strategy in an enterprise by joining processes, technologies, and people" [9].

In order to build a successful SOC, there are numerous factors that it should take into account, each with different levels of importance. One should start by examining what the SOC aims to protect: the organization's constituency – "a bounded set of users, sites, IT assets, networks, and organizations" [10]. To do so, one must understand in what ways said constituency could be a target for attacks with malicious intent and analyze what means one has available to defend it.

2.1.1 SOC Models

Depending on factors such as the purpose of the SOC and its geographic location, there are 5 different primary models as defined in [8]:

- Virtual SOC - SOC-as-a-Service or outsourced SOC
- Multifunction SOC/NOC - combination of in-house IT and security teams
- Hybrid SOC - combination of Virtual SOC and Dedicated SOC
- Dedicated SOC - in-house SOC

- Command SOC - multiple coordinated SOC in different locations

An organization's SOC may not necessarily fall into one of these but these – or slight variations of these – are the most common. Besides these main models, other types of SOCs with specific purposes exist such as SOCs for military operations or humanitarian causes which have distinct requirements from common SOCs and are often temporary.

2.1.2 SOC Components

To be able to protect its constituency, the SOC usually has security analysts responsible for operating it. These analysts need to be familiar with the constituency, in particular with how it's composed, how it can be attacked and how it can be defended.

Besides the monitoring and active defense tools in its constituency, the SOC is made up of other components at its core – commonly called the enclave of the SOC. One of the most important, the SIEM, offers the ability of aggregating data gathered from disparate sources in the constituency, performing correlation on said data, reporting and managing events, visualizing the data, and searching and analyzing it. Information about incidents detected by the SIEM can be fed to an Incident Response tool, another component of the enclave. When security incidents are identified, IR helps analysts act directly on the constituency in order to control their consequences, minimize damage to affected systems, speed up the recovery process and mitigate the exploited threat if possible, reducing the chances that such an event happens again and evolving the SOC to be better prepared for future incidents. Threat Intelligence tools – or Cyber Threat Intelligence – are responsible for storing, processing and acting on information about threats to the constituency. Many times also gathering information about known threats from public threat intel feeds, these tools collect IoCs in the constituency to automatically detect known intrusions with a high degree of confidence. Analysts can then directly take action on the constituency or send that information to the SIEM for further analysis.

2.2 Devops and Infrastructure-as-Code

Following on to the concept of DevOps, it can be defined as "a paradigm that aimed to reduce the disconnect between development and operations teams by promoting collaboration, communication, and integration between them" [7]. Although there's no globally accepted definition, in an attempt to clarify, it may be portrayed as a method for developing software that allows both teams to work closer together or even merge, sharing each other's know-how and thus reducing the friction between writing code and releasing the product.

In what comes to the notion of Infrastructure-as-Code, it can be part of the DevOps process, more specifically deployment automation. Simply, it allows to automate different kinds of infrastructure using code. Morris describes it as "an approach to infrastructure automation based on

practices from software development. It emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration. You make changes to code, then use automation to test and apply those changes to your systems" [6].

2.2.1 Major DevOps Tools Categories

As DevOps comprises many distinct areas, the process entails tools that serve numerous purposes, namely [5]:

- Knowledge sharing: Focus on collaboration and sharing between different people and departments. Documentation and task organizations tools such as Trello and Redmine are prevalent in this category.
- Source code management: Collaboration is also important in this category. Focus on versioning code and developing code together. Versioning tools such as Git and SVN are prevalent in this category.
- Build process: Focus on automation, analysis and testing. Build automation tools and testing frameworks such as Gradle and JUnit are prevalent n this category.
- Continuous integration: Focus on automation, reliable release process and CI/CD. Continuous integration tools such as Jenkins and Travis CI are prevalent in this category.
- Deployment automation: Focus on automation, reliable release process, continuous delivery. Infrastructure-as-Code, containerization and cloud platform tools such as Puppet and Heroku are prevalent in this category.
- Monitoring and logging: Focus on monitoring of aspects like performance and availability after running and logging of other metrics and alerts. Monitoring and log management tools such as Zabbix and Graylog are prevalent in this category.

  Various DevOps tools may have overlapping characteristics and functionalities. These categories help define each part in the DevOps process for an overall better delivery of software even if some of them may have similarities at times.

2.2.2 Infrastructure-as-Code Characterization

Infrastructure-as-Code may differ according to many factors such as the type of infrastructure resources it targets, approach, method and architecture.

There are 3 main types of infrastructure resources that IaC can target, as stated in [6]. These consist of compute, storage and networking resources.

When it comes to the approach, IaC can be either declarative or imperative. In a declarative approach, IaC describes what should be the target configuration, the appropriate state of the system, and it executes the necessary steps to reach that state. In an imperative approach, on the other

hand, IaC describes how the infrastructure is to be changed, the exact operations that need to be executed in the proper order to reach the desired conclusion.

IaC can use different methods, depending on how the servers should receive their configuration. If using a push method, the controlling server pushes the configuration to the destination system. If using a pull method, however, the server to be configured will pull its configuration from the controlling server.

IaC can accommodate different architectures: mutable or immutable. With a mutable architecture, the infrastructure has the ability to change. It's flexible but configuration drift may happen. With an immutable architecture, the infrastructure cannot be changed. It's costly to make changes as the infrastructure needs to be taken down and re-deployed in order for them to take effect.

2.2.3 Common Infrastructure-as-Code Tools

Various tools can be used for IaC, although some of the most common include:

- Docker
- Ansible
- Vagrant
- Kubernetes
- Chef
- Terraform
- Puppet

Some tools are more geared toward certain tasks or types of infrastructure so these can be experimented with in different ways for the SOC infrastructure. In this dissertation, Docker was used for containerizing components of the SOC. It uses virtualization at the level of the operating system to create isolated pieces of software that may interact with each other.

Ansible is configuration management tool used to orchestrate other target machines. It makes use of configuration modules, Playbooks, to define the desired state of the target.

Vagrant is a tool for creating portable virtual environments such as Docker containers and VMs using Provisioners and Providers. Provisioners are used to customize the environment (e.g. Puppet, Chef, Ansible) and Providers are used to create said environments (e.g. Docker, VirtualBox, Hyper-V).

Kubernetes is an orchestration tool used for automating the management of containers on a node cluster by scheduling and controlling various pods. Nodes are machines where containers can be deployed and have container runtimes (e.g. Docker) and pods consist of groups of containers running on the same node.

Chef is configuration management tool used to configure and maintain infrastructure with recipes. Recipes are descriptions of how the tool should configure the infrastructure and manage it (e.g. what should be installed or running on servers). The user can also group these recipes into *cookbooks* to simplify their management.

Terraform is a resource management tool for managing infrastructure resources using providers. There are different providers for different types of infrastructure resources and they integrate with Terraform to define their desired state.

Puppet is configuration management tool used to configure Windows and Unix-like systems. System state can be described in files, Puppet manifests, which are compiled into resources specific to the target system and then applied to it.

2.2.4 Examples of DevOps for Infrastructure

One of the most common uses of DevOps for infrastructure is Infrastructure-as-Code (IaC) which can range from simple scripting to containerization and templating. IaC is always used with the objective of automating infrastructure. This must happen because, especially in cloud environments, servers need to be configured and deployed on the fly in large scale and in environments of constant change. We believe these advantages can also be brought to the SOC, enabling easier deployments.

There are examples of DevOps being used to configure and deploy other types of infrastructure besides the SOC [6] [1] such as physical servers, VMs, containers or application hosting clusters. Besides the common use in deploying these compute resources, other types of infrastructure such as networks or storage can be managed with DevOps. One such example is Software-Defined Networks (SDNs) [3], where a network can be dynamically and programmatically configured in order to improve its performance and monitoring. Resources such as VPNs, proxies and gateways can be deployed using software-defined networking practices, making the process more streamlined and repeatable. Lastly, storage resources can also be managed using DevOps for data storage or object storage solutions and secrets management. An example of use is allocating storage to applications transparently and as needed instead of directly specifying which physical storage is to be allocated to each application.

2.2.5 DevOps for the SOC

Not much work exists yet related to the use of DevOps techniques in the SOC as SOCs themselves are a relatively novel concept in the security field. Although few, some similar SOCs have been built such as AlienVault[1], Apache Metron[2], Automated SOC [2] and OmniSOC[3]. Both Apache Metron and Automated SOC are open-source solutions, AlienVault is a commercial solution and

[1] AlienVault – https://cybersecurity.att.com
[2] Apache Metron – https://metron.apache.org
[3] OmniSOC – https://omnisoc.iu.edu

Table 2.1: Comparison Table *OmniSOC relies partly on commercial tools [2]

	Automated SOC	AlienVault	OmniSOC	Apache Metron
Automated Deployment	Y	Y	N	Y
Threat Intelligence	Y	Y	Y	N
Endpoint Detection	Y	Y	N	N
Network Detection	Y	Y	Y	N
SIEM	Y	Y	Y	Y
Reporting	Y	Y	Y	Y
Analysis Guidance	Y	N	N	N
Commercial Solution	N	Y	Y*	N

OmniSOC relies on some commercial tools. We believe the most relevant to this work is Automated SOC as it is the most complete in terms of functionality 2.1, it's completely open-source and aims to achieve a similar objective.

AlienVault, now called USM Anywhere[4], is a commercial solution provided by AT&T that aims to provide SOC functionalities to enterprises of all sizes. The cheapest package at the time of writting, called Essentials, is directed at small teams and starts at 1075$ per month which amounts to at least 12900$ per year. It can be hosted on-premise, in a remote location or in the cloud. A SIEM at its core, it integrates with modular components to extend security functionality according to the needs of the constituency, making for a quite complete SOC solution. Its open-source counterpart, OSSIM[5], however, lacks in functionality as it only provides a SIEM paired with a few basic security controls for a bare minimum in an enterprise looking for better security – a comparison between the tools can be found at `https://cybersecurity.att.com/products/ossim/compare`.

Apache Metron is a now retired project that evolved from Cisco's OpenSOC[6], an (also retired) integration of open-source tools that provides the capabilities of a simple SOC using a Hadoop[7]. Apache Metron also uses a Hadoop-based SIEM, which can be swapped for Elasticsearch, and is based on Apache projects to provide a SOC solution with a similar intent to the one presented in this dissertation. However, this project is not only very limited in functionality but also outdated as it is no longer supported.

Automated SOC is an open-source automated SOC targeted towards SMEs that have difficulty in establishing a cyber defense operation either due to the high costs or the complexity of deployment. It uses Elasticsearch as a SIEM, just like the solution presented in this dissertation. Aside from that, and as proof that claimed functionality is actually there, this SOC has been tested and verified by using attacks from the MITRE attack techniques[8] against it and checking if they were detected.

[4]USM Anywhere – https://cybersecurity.att.com/products/usm-anywhere
[5]OSSIM – https://cybersecurity.att.com/products/ossim
[6]OpenSOC – https://opensoc.github.io
[7]Hadoop – https://hadoop.apache.org
[8]MITRE Attack Techniques – https://attack.mitre.org/techniques/enterprise

OmniSOC is a SOC shared among multiple higher education institutions to monitor the networks of their campuses. It focuses on network detection and doesn't have any host-based tools which limits its protection to network-based attacks. This might make it difficult to find the source of malicious traffic. It also makes use of some commercial tools for assisting in mitigation, which raises costs of running this SOC.

2.3 Conclusions

SOCs are complex infrastructures but its use in organizations has seen an increase in the past few years, as a response to a growth in cyberattacks. Although DevOps has been in use for quite some time, its adoption and research have also recently expanded. The use of DevOps for the SOC is still a quite unfamiliar and unexplored area but the automation provided by DevOps tools can potentially benefit the SOC and the professionals responsible for maintaining it.

Chapter 3

SOC Architecture and Deployment Requirements

This chapter first discusses a possible architecture for the SOC and presents components that can be used to implement this architecture. Finally, it introduces the needed requirements for the deployment of such an infrastructure.

3.1 System Architecture

For a SOC to reliably protect its constituency, it should be capable of performing certain tasks. Some of these include host monitoring, network monitoring, asset management, threat intelligence collection, security information and event management, correlation, reporting, and incident response [2].

There's a variety of tools and technologies that can be used to build a SOC capable of performing the aforementioned tasks. These include commercial solutions as well as freely available software. In this work, we focus on free and open-source tools for this purpose.

As presented in 2.1, SOCs may have different configurations depending their purpose and location. In this section, we overview a configuration with components on the constituency and others outside – the enclave. Components that are part of the SOC enclave constitute mostly data aggregation, correlation and incident response whereas components in the constituency constitute mostly monitoring, data collection and active defense like firewalls.

3.1.1 Enclave

The enclave can be considered the core of the SOC. As shown in Figure 3.1, it performs tasks such as asset management, threat intelligence collection, security information and event management, correlation, reporting, and incident response. For this purpose it can make use of the following tools:

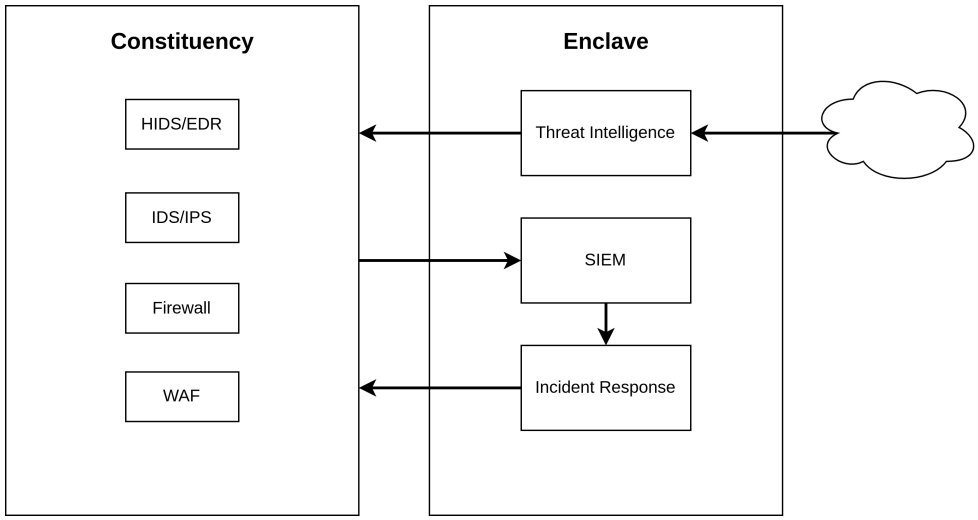

Figure 3.1: SOC Tool Architecture and Data Flow

1. Security Information and Event Management (SIEM): Where most of the SOC data is stored. The data is fed to it from the components running on the constituency. Data aggregation is one of its strong points.

2. Incident Response (IR): Interface for organizing and displaying alerts from other components. Analysts use this to analyze the information from alerts (source, reason for trigger, etc.) and decide on how to act.

3. Threat Intelligence, also known as Cyber Threat Intelligence (CTI): Interface for collecting Indicators of Compromise (IOCs). These can be visualized or even fed to other tools – e.g. SIEM – for further analysis and correlation.

3.1.2 Constituency

The constituency is what the SOC intends to protect. It is a "bounded set of users, sites, IT/OT assets, cloud assets, data, networks, and organizations" [4], that is all of the computers, servers, mobile devices, assets, employees, etc. that are part of one or several organizations (depending on what the SOC aims to protect). As shown in Figure 3.1, it performs tasks such as host monitoring, network monitoring and active defense. For this purpose it can make use of the following tools:

1. Firewall (FW): Establishes a barrier between the constituency network and the Internet. Controls and monitors incoming and outgoing traffic based on a set of rules. Can feed this information to the SIEM.

2. Web Application Firewall (WAF): Similar to a regular firewall, with a focus on HTTP traffic. Can be setup between a web application on the constituency and the Internet. Detected attacks to the application can be fed to the SIEM.

3. Antivirus (AV): Host-based software that prevents, detects and removes various types of malware. Information about detected malware can be fed to the SIEM.

4. Intrusion Detection System (IDS)/Intrusion Prevention System (IPS): Monitors network traffic on the constituency for known threats by comparing packets to a database of known attack signatures. IDS only monitors the packets while IPS may alter them to prevent attacks. Information about detected attacks can be fed to the SIEM.

5. Host Intrusion Detection System (HIDS)/Endpoint Detection and Response (EDR): Host-based software capable of monitoring a device and its network interfaces – similar to an IDS – for threats and possibily mitigate them. Information about detected threats can be fed to the SIEM.

3.2 Deployment Requirements

For deploying a SOC, an architecture that fits the constituency should be planned and tools that fit such architecture should be chosen. Hardware able to run the chosen tools should be available on both the enclave and the constituency. For the correct operating of the SOC, a secure communication channel should also be available between the constituency components and the enclave.

3.3 Conclusions

A SOC can be an extremely complex system that is hard create and get working correctly. There's a large diversity of components which can be used and that poses a challenge to those wanting build this kind of infrastructure. Add to this the trouble of tracking changes to the SOC, e.g. updating or adding new components, and the lack of standardized testing and a high barrier of entrance is created. SMEs are unable to keep up with these demands and thus are the ones most vulnerable to cyberattacks. This also poses a challenge to this work since automating such diverse components, each with its own specificities, in a standard manner and getting them to seamlessly communicate is no easy task. Being able to overcome the difficulty of automating this process may help smaller enterprises defend themselves against bad actors in a less costly manner.

Chapter 4

Development Methodology

This chapter presents how a Python module was developed to help automate the deployment of a SOC. It first presents the requirements for using the module and then explains how it can be used for each component.

4.1 Implementation Architecture

With the intent of automating the deployment of a SOC infrastructure similar to the one discussed in 3.1 and understanding the benefits of said automation, we developed a Python module, available at `https://git.fe.up.pt/cesarnogueira/soc-devops`. This module is capable of deploying two components of the SOC, one part of the SOC enclave – the SIEM Elastic Stack – and another in the constituency – the WAF ModSecurity.

The developed Python module was implemented using the Docker SDK for Python[1], so the components are running on docker containers. Using this module, security professionals can write scripts to automate some tasks such as the deployment of the components.

When the scripts are created, the machines that will run the components can fetch the code from the configuration server to then run it (although that step was not explored in this work). This means that hosts running the components have the following requirements:

1. Linux host OS;
2. public interface with open ports;
3. Docker installed;
4. Python 3 installed;
5. Docker SDK for Python installed;
6. permissions to run Python code;
7. permissions to run docker commands.

[1] Docker SDK for Python – https://docker-py.readthedocs.io

4.1.1 Elastic Stack

The module contains three functions to deal with Elastic Stack deploy_elk, remove_elk and add_elastic_node. Before using these functions, the user should take care to create the necessary directories and configurations needed to run Elastic Stack. This can be done using a bash script, setup.sh, that creates the directories and copies the configurations from the templates folder. The configurations should be altered to respect the environment.

After the configurations are correct, Elastic Stack is ready to be deployed. The function deploy_elk serves this purpose. It creates a network for the stack and runs three containers: Elasticsearch, Kibana and NGINX. The following parameters should be set to call this function:

- version: The version to use for the Elasticsearch and Kibana containers. There's no option to choose a different version for each to avoid compatibility issues.
- ngx_version: The version to use for the NGINX container. This does not have much impact on the other components since the container only serves as a reverse proxy for accessing Elasticsearch and Kibana.
- path: The path to the folder containing the configurations.
- port: The port exposed by NGINX to access the other components.

To add an Elastic node to the existing cluster, we can use the function add_elastic_node. The following parameters should be set to call this function:

- name: The name of the node. Can be any string the user desires.
- version: The version to use for the container. Should be the same as the other nodes to guarantee compatibility.
- path: The path to the folder containing the configurations.

To remove this stack, simply call the function remove_elk. The following parameters should be set to call this function:

- paths: The set of paths to be removed if the user wants to delete data and configurations. Can be left empty if the no path is to be removed.
- name: The name of the node to remove if removing a single Elastic node. Should not be set or should be set as 'None' if removing the deployed using the function 'deploy_elk'.

4.1.2 ModSecurity

The module contains two functions to deal with ModSecurity: deploy_modsec and remove_modsec. The function deploy_modsec can be used to create an instance of ModSecurity that sends logs to

Elasticsearch. It runs two containers: ModSecurity Core Rule Set and Filebeat. Filebeat is responsible for parsing the logs generated by ModSecurity and shipping them to Elasticsearch. Before using this function, the user can write additional rules on the files REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf and RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf if needed. The following parameters should be set to call this function:

- name: The name of the stack. Can be any string the user desires.
- backend: The location of the web application to be protected by ModSecurity.
- elastic: The location of the Elasticsearch server.
- version: The version to use for the ModSecurity container.
- fb_version:The version to use for the Filebeat container. Should be the same as the version of Elastic Stack to guarantee compatibility.
- path: The path to the folder containing the configurations.
- port: The port exposed by ModSecurity to access the protected web application.

To remove this stack, simply call the function remove_modsec. The following parameters should be set:

- name: The name of the stack to remove.
- paths: The set of paths to be removed if the user wants to delete data and configurations. Can be left empty if the no path is to be removed.

More than one instance of ModSecurity can be created, one for each of the web applications in the constituency. In this case, a different index name should be set on the configuration of Filebeat to help in distinguishing each instance in Kibana.

4.2 Conclusions

Being able to program the deployment of the SOC in this way helps in abstracting from problems such as compatibility with the host OS and installation/configuration of different tools and focus on customizing the tools for the environment – parametrization facilitates the configuration of each component. Besides that, the fact that scripts can be created for specific purposes makes it easy to reproduce the environment in case of a need for reconfiguring or rebuilding.

Chapter 5

Use Cases

This chapter showcases some of the benefits of using the module and how it can be used with code examples for each use case. Use cases include version management and scaling of both previously mentioned components: Elasticsearch and ModSecurity. Later, we explain how other use cases could be implemented from existing code and what modifications to the code would be need to implement them.

5.1 Use Case Overview

Once a SOC has been deployed using the deployment functions in the developed module, further functionality is available for managing the components. The two implemented use cases are related to management of component versions and scaling. In particular, it is possible to upgrade or downgrade Elastic Stack and ModSecurity while keeping previously existing data or deleting. It is also possible to add/remove both Elasticsearch nodes and ModSecurity instances. We deemed these use cases as some of the most useful to automate but the module can be expanded to accommodate additional use cases.

These use cases are presented alongside example code in small Python scripts that implement each of them. There are various ways to use the developed module with the same results such as directly calling the functions in a Python file – like a traditional script – or creating other wrapper functions with specific purposes to use in a more structured program – such is the versatility of a programming language like Python. The given examples are only guidelines on how to use the module out of the many possible ways it can be used.

5.2 Version Management

Version management in this context means being able to change the version of the Docker container(s) running each of the SOC components by either upgrading or downgrading them. Managing versions in a SOC is important for various reasons such as upgrading to get the latest security patches or downgrading for compatibility with other components and should be done with as little

downtime as possible. In the developed module, both upgrading and downgrading are done in a very similar manner – the user must only choose a higher or lower version respectively – so no emphasis will be given to the differences in their use in the following subsections.

5.2.1 Deleting Data

When changing a component's version, the user may want a *fresh start* by deleting previously existing data. This can be done when old data is of no more use or when data is incompatible between versions – making old data obsolete.

Below is an example of a script that downgrades Elastic Stack to version 7.10.1, deleting existing data. To implement this use case, the user simply needs to call the function that removes Elastic Stack, passing as a parameter a set of the locations where the data is located – '/home/soc/elk' in this case – and then call the function that deploys it again, defining as parameters the new desired version – 7.10.1 in this case – and the location of the new data – the same path '/home/soc/elk' in this case.

```
#!/usr/bin/python3

from soc_devops import *


def downgrade_elk():
    remove_elk({'/home/soc/elk'})
    deploy_elk('7.10.1', 'latest', '/home/soc/elk', 80)


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

Below is an example of a script that upgrades ModSecurity to version 'nginx', the latest available version that uses NGINX as the web server, deleting existing data. To implement this use case, similarly to the previous example, the user simply needs to call the function that removes ModSecurity, passing as a parameter a set of the locations where the data is located – '/home/soc/-modsecurity' in this case – and then call the function that deploys it again, defining as parameters the new desired version – 'nginx' in this case – and the location of the new data – the same path '/home/soc/modsecurity' in this case. Note that the version of the Filebeat container that runs together with ModSecurity can also be set as a parameter so the same procedure can be used when changing the version of Elastic Stack to ensure Filebeat is compatible with it.

```
#!/usr/bin/python3

from soc_devops import *
```

```
def upgrade_modsec():
    remove_modsec('crs', {'/home/soc/modsecurity'})
    deploy_modsec('crs', 'http://192.168.1.44:8000', '192.168.1.40',
        'nginx', '7.10.2', '/home/soc/modsecurity', 80)


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

5.2.2 Keeping Data

Contrary to the previous use cases, the user may want to keep data when changing a component's version. This can be done to avoid having to start from scratch after an update for example.

Below is an example of a script that downgrades Elastic Stack to version 7.10.1, keeping existing data. To implement this use case, the user simply needs to call the function that removes Elastic Stack and then call the function that deploys it again, defining as parameters the new desired version – 7.10.1 in this case – and the location of the old data – the path '/home/soc/elk' in this case.

```
#!/usr/bin/python3

from soc_devops import *


def downgrade_elk():
    remove_elk()
    deploy_elk('7.10.1', 'latest', '/home/soc/elk', 80)


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

Below is an example of a script that upgrades ModSecurity to version 'nginx', keeping existing data. To implement this use case, the user simply needs to call the function that removes ModSecurity and then call the function that deploys it again, defining as parameters the new desired version – 'nginx' in this case – and the location of the old data – the path '/home/soc/modsecurity' in this case.

```
#!/usr/bin/python3

from soc_devops import *


def upgrade_modsec():
```

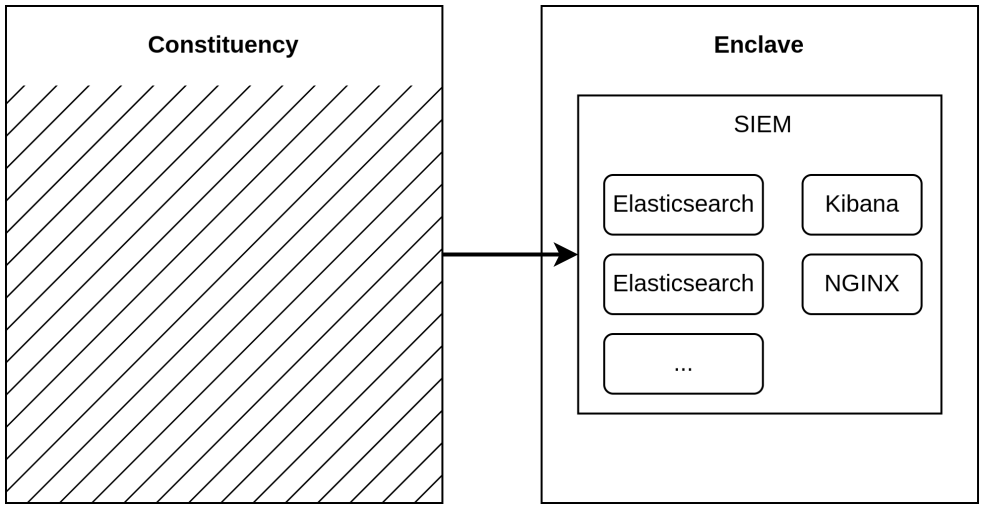

Figure 5.1: Scaling Elasticsearch

```
    remove_modsec('crs')
    deploy_modsec('crs', 'http://192.168.1.44:8000', '192.168.1.40',
        'nginx', '7.10.2', '/home/soc/modsecurity', 80)


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

5.3 Component Scaling

When the SOC has a larger constituency, just running a default configuration may not be enough. This is why scaling the SOC according to the needs of the constituency is important. A larger constituency may imply the need for more computing resources in the enclave to process the data, more data gathering in the constituency or both.

5.3.1 Adding Elasticsearch Nodes

To increase the capacity of Elasticsearch, the user may need more nodes in the cluster. This helps in the distribution of tasks such as data storage, searching and indexing. As seen in Figure 5.1, this only adds nodes to the Elasticsearch cluster, keeping both Kibana and NGINX as before.

Below is an example of a script that adds an Elasticsearch node to the existing Elasticsearch cluster. To implement this use case, the user simply needs to call the function that creates an additional node, defining as a parameter which cluster to join.

```
#!/usr/bin/python3

from soc_devops import *
```

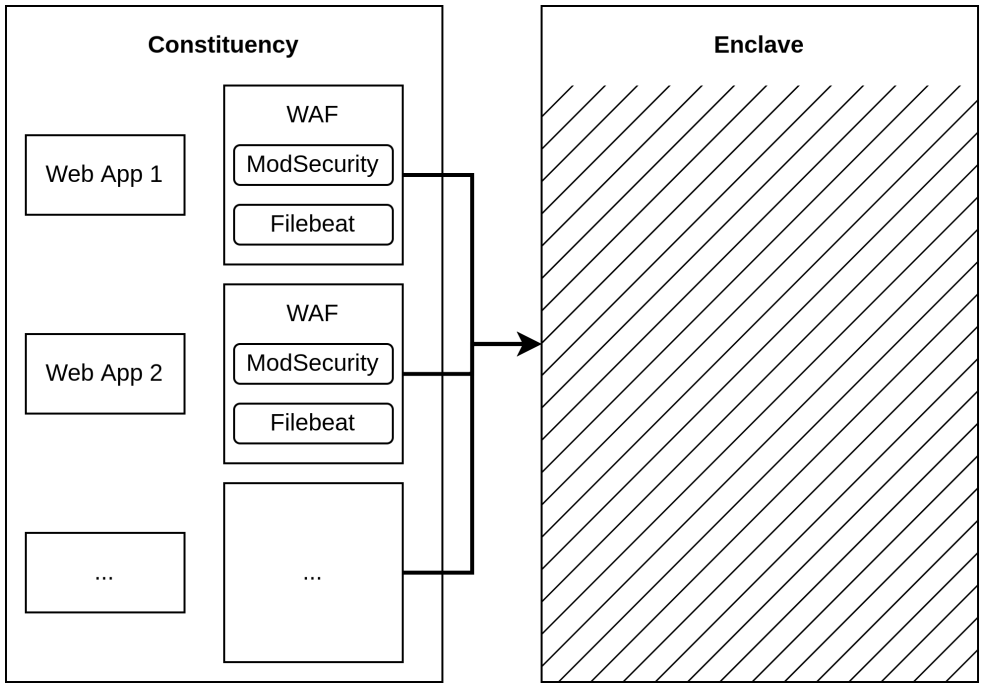

Figure 5.2: Scaling ModSecurity

```
if __name__ == "__main__":
    add_elastic_node('extra-node', '7.10.2', '/home/soc/elk')
```

5.3.2 Creating Additional ModSecurity Instances

If the constituency needs protection on more than one web application, the simplest solution is to deploy multiple ModSecurity instances. Deploying one or multiple instances works the same and they can all send their data to the same Elasticsearch cluster. When deploying more instances, they all contain a new set ModSecurity and Filebeat containers, one for each different web application that needs to be protected, as seen in Figure 5.2.

Below is an example of a script that creates an additional ModSecurity instance. To implement this use case, the user simply needs to call the function that deploys ModSecurity, defining as a parameter the location of the web application to be protected.

```
#!/usr/bin/python3

from soc_devops import *


if __name__ == "__main__":
```

```
deploy_modsec('crs', 'http://192.168.1.45:8000', '192.168.1.40',
    'nginx', '7.10.2', '/home/soc/modsecurity', 80)
```

5.4 Additional Use Cases

In order to create a more complete solution and accommodate more use cases, the developed module can be expanded. Adding other components should not need modifications on the existing ones but would most likely need to connect to the existing Elastic Stack. Creating functions to deploy and remove components can be done similarly to the existing ones, using Docker SDK. Other components may also be added to the SOC not using Docker, the SOC should still work as expected, but management would be more difficult, defeating the purpose of automating it. Other use cases related to existing components, such as changing configurations, would need changes to the code to parameterize more configurations or changing the configurations from the default to the desired ones directly in the code.

5.5 Conclusions

These use cases show some of the usefulness of automating the SOC besides just deployment. Changing versions comes down to two lines of code instead of possibly having to reinstall or build different versions of the same component on the host OS as well as with little downtime and scaling can be done on the fly and as needed. Expanding this to even more use cases would enable less time concerning with SOC management and more time spent responding to actual security incidents.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The work presented in the previous chapters of this dissertation was developed with the goal of automating some parts of the SOC, its configuration, deployment and later management, and understand how this automation affects the process of creating a SOC from scratch, specifically what advantages come with programming common tasks and parametrizing configurations.

In Chapter 3, we propose an architecture that encompasses the main requirements for a SOC. We can already see how complex this kind of system can be and how that poses a challenge to its automation.

In Chapter 4, we show how we were able to successfully automate the deployment of two of the SOC components by developing a Python module that utilizes Docker to run various tools. It is still not able to deploy a complete SOC but we can already see the benefits of this simplification in the deployment.

Some other common use cases in the management of the SOC were developed in the Python module, presented in Chapter 5, including version management and scaling of components. These allow for managing the SOC according to needs at any time and without having to change the infrastructure too much. This is also extendable to other common use cases to reduce time expended in maintenance of the SOC.

In the end, the developed module is not a complete solution nor intended to be the next standard for SOC automation. For developing a complete solution, possible paths are introduced in 6.2. It serves to show the possible benefits of one such solution, such as spending less time with configuration and maintenance of the SOC and focusing on utilizing it to more swiftly respond to security incidents.

6.2 Future Work

There is still much to be explored in automation of SOCs and a few different paths can be taken. Introducing some of the IaC tools presented in Chapter 2 in the automation process might facili-

tate the management in larger and constantly changing environments. Different architectures and purposes than the ones presented on Chapter 3 may still be explored such as emergency SOCs for military operations and what different benefits automation could bring to those. Some SOC components may benefit more from automation than others and that was not fully explored in this work. In addition, other DevOps practices can be investigated such as CI/CD in order to create a more complete DevOps pipeline for the SOC and understand what other advantages could be obtained from the application of these kind of practices in the SOC.

References

[1] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. DevOps: Introducing Infrastructure-as-Code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 497–498, May 2017.

[2] Anthony Cardarelli. Automated Deployment of a Security Operations Center. Master's thesis, University of Cincinnati, 2020.

[3] Paul Goransson, Chuck Black, and Timothy Culver. *Software Defined Networks, Second Edition: A Comprehensive Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2016.

[4] Kathryn Knerler, Ingrid Parker, and Carson Zimmerman. *11 Strategies of a World-Class Cybersecurity Operations Center*. MITRE Strategic Communications, March 2022.

[5] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A Survey of DevOps Concepts and Challenges. *ACM Computing Surveys*, 52(6):127:1–127:35, November 2019.

[6] Kief Morris. *Infrastructure as Code*. O'Reilly Media, Inc., USA, 2nd edition, December 2020.

[7] Roshan N. Rajapakse, Mansooreh Zahedi, M. Ali Babar, and Haifeng Shen. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141:106700, January 2022.

[8] Gorka Sadowski, Craig Lawson, Toby Bussa, Pete Shoard, Rajpreet Kaur, and Mitchell Schneider. Selecting the Right SOC Model for Your Organization, September 2018.

[9] Manfred Vielberth. Security Operations Center (SOC). In Sushil Jajodia, Pierangela Samarati, and Moti Yung, editors, *Encyclopedia of Cryptography, Security and Privacy*, pages 1–3. Springer, Berlin, Heidelberg, February 2021.

[10] Carson Zimmerman. *Ten Strategies of a World-Class Cybersecurity Operations Center*. MITRE Corporate Communications and Public Affairs, USA, 1st edition, October 2014.