POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Towards intelligence driven automated incident response

Supervisors
Prof. Cataldo Basile
Dr. Leonardo Regano

Candidate
Francesco Settanni

ACADEMIC YEAR 2021-2022

Summary

Nowadays, an ever-increasing number of cybersecurity threats is looming over organizations all over the world. Some of them have recently digitalized many of their assets, leading to an increased attack surface. This leaves the door open to threat actors since solid management of cybersecurity issues is often missing, especially in SMEs.

Incident response teams dedicated to handling incidents are running low on personnel, and even when this is not the case, they get overwhelmed with new alerts, meaning they cannot keep pace with new threats and adversaries. This is pushing an automation effort aimed at easing the burden of repetitive tasks on SOC teams. Many companies are proposing their own solution, and these often take the form of integrated frameworks used to handle all aspects of security in an automated way, but all lacking interoperability.

Another aspect reinforcing the need for automated security management is the integration of threat intelligence feeds into the defense stage. Feeds are used to share information about new threats, vulnerabilities, or incidents that have affected organizations. Feeds can improve the defense stage by enriching threat knowledge in an automated way, thus decreasing incident response times and improving the defensive stance.

The knowledge-sharing effort, though, is still mainly limited to indicators of compromise and secondary information such as those regarding the identity, the behavior, or the campaign history of a threat actor. What is still lacking is the coverage of actionable courses of action, that is, methods and approaches that can be taken in response to the given threat or incident.

In this thesis, a comprehensive approach to automation, from knowledge sharing to incident response, is covered. This effort takes the form of a standard-based framework that makes full use of emerging standards in the field. With this framework, automatable actions are defined by analysts as recipes in a simplified high-level security policy language.

Courses of action together with threat details can be shared across teams or exchanged machine to machine by means of threat feeds, enclosed in intelligence reports. Organizations can deploy that course of action in a different operational environment, adapting it to their unique incident situation but still taking advantage of the shared intelligence. The STIX language is used to gather all relevant information about incident handling and generate a report. The amount of information shared is bound to their confidentiality. Interoperability across different operational environments is guaranteed by wrapping recipe courses of action in a machine-parsable structured language that can embed in it all required deployment variables.

A proof of concept will be shown, consisting of an emulated software network landscape for which security alerts will be received. An interpreter will work as a translator of high-level policy language to the low-level commands to be applied to the network environment, thus enforcing a given course of action. At the end of the remediation, a report will be produced containing threat and incident response details. In the evaluation phase, it is shown how the system evolves its state in response to a new alert, optimizing service operations and resources.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Introduction

Recent developments in cybersecurity are posing major new challenges to organizations that rely on digital infrastructures. The threat landscape has evolved over time, and today features far more sophisticated and complex threats than were previously seen. At the same time, organizations are quickly transitioning to full-fledged distributed environments, driven by the advent of cloud computing. While offering many advantages, this also leads to a sudden expansion of the attack surface in their infrastructures.

In this scenario, the presence of highly skilled security operators handling all the various security aspects within organizations is imperative, but the reality tells a different story. In fact, the industry is plagued by a persistent personnel shortage which hinders the efforts aimed at improving and advancing all aspects related to the management of cybersecurity. This is coupled with the current threat landscape showing that the number of new threats in the wild is increasing as never before, thus putting teams dedicated to security management in harsh conditions. This trend is also driven by the phenomenon of commoditized exploits, which expands the base of malicious actors by providing easy access to complex techniques and tools used to perform highly targeted attacks.

The need for automation in all IT security management is clearly underscored from these premises. Many of the operations performed every day in the security field, such as alert detection, triage, and incident response can be synthesized as a set of procedures, each composed of different steps. Often these processes consist, in fact, of repetitive actions that can be easily automated, thus freeing security operators from specific duties while at the same time guaranteeing human supervision for those operations in which liability is required.

Furthermore, the new threat landscape calls for a rebalancing in the general approach to guarantee security and deter attacks. In fact, it is increasingly common for certain highly stealthy attacks to be detected only far on in their exploitation process, when the malicious actor has penetrated inside the infrastructure and possibly has already reached his objectives. Supply chains attacks, zero-day vulnerabilities, and botnets are instances of these new threats, and they are well known for their capability to affect even the most regarded organizations from a security stance perspective.

This is due to the fact that the main focus up until now has been towards securing the internal infrastructure, for instance, keeping the systems patched and increasing the security awareness, with an inward leaned approach. Despite this being a prerequisite for a good security posture, the complexity and sophistication of new attacks and new menaces such as state actors with potentially unlimited capabilities and resources, imply that a new preemptive defensive security stance is undertaken. This new approach should be based on an outward leaned capability, under which the behaviors and techniques of malicious campaigns and threats are constantly monitored and used as a building block for the establishment of security prevention and detection measures, thus adopting a proactive stance that allows to deploy in advance the necessary measures.

In order to establish a comprehensive global situational awareness, though, it is essential to have access to a vast quantity of knowledge so as to gain a full view of the threat landscape.

To this end, it is gaining ever more traction in the field of cybersecurity the practice of threat intelligence sharing. Multiple initiatives have sprung up in this context, with many of them being supported by national actors and others offering both commercial and open solutions. These initiatives are used as sharing hubs of threat intelligence knowledge, which, if integrated into organizations' security pipelines, can transform the security handling from a reactive approach to a proactive one.

To effectively use all the shared knowledge that enriches the threat landscape awareness and the adversaries' behaviors, it is essential to adopt automation along the entire security pipeline, since the massive amount of information would be impossible to handle manually by security operators alone. Nowadays, shared threat intelligence is limited to raw artifacts that fall short of the stated potential advantages, such as IP addresses or hashes, often shared without any context. It would be helpful, instead, if shared intelligence also embraced more high-level concepts, such as the behaviors of a given threat seen in the wild or the procedure adopted to remediate an incident caused by it. This puts organizations in a forward position since they could tap into those already validated knowledge to adopt the necessary measures in incident response situations, thus drastically reducing response times.

This thesis aims to propose a new framework architecture for cyber security management, in particular concerning the incident response domain, by adopting those approaches and methodologies described earlier. The architecture makes full use of automation capabilities, leveraging on concepts and approaches in the network field, such as virtualization and software networks. The proposed model leverages the STIX format, which is the defacto standard in cyber threat intelligence for what concerns intelligence sharing, and proposes the use of security playbooks for the automation and documentation of incident response activities.

In particular, it is proposed a new high-level playbook descriptor format, called Recipe, which is used to express security procedures in a fashion resembling natural languages, thus allowing security operators to quickly inspect and possibly modify them, while at the same time guaranteeing nontechnical users an insight into them. Once playbooks are validated, they can be enclosed into STIX reports and then shared across organizations. To this end, the use of an emerging format for playbooks description is also proposed, that is, the CACAO format, an emerging standard in the industry that provides a way to structure playbook information in a consistent way, thus guaranteeing machine readability and cross compatibility.

This thesis work is focused on incident response-related automation tasks in the context of network infrastructures. In this context, the operational environment is represented by a network infrastructure, and in particular, the architecture makes use of a service graph abstraction that represents the network substrate. Security related features are thus enforced by network service functions deployed onto the service graph, according to the NFV paradigm. These act as security controls. An interpreter is used to deploy playbooks by converting high-level security actions into low-level configurations enforced by means of security controls. Moreover, security actions may require the modification of the network topology, and in this case, the service graph provides an interface with which these changes can be applied during the course of the playbook deployment.

A proof of concept is implemented adhering to the proposed architecture and is tested against a botnet scenario, that is a particular security threat common nowadays.

Thesis outline

The remainder of this thesis is organized as follows:

- Chapter 2: introduces the topics and concepts used throughout this dissertation and gives an overview of some related research works, together with state of the art in the field;
- Chapter 3: introduces the goal of this thesis work and its use case;
- Chapter 4: briefly discusses how the solution was designed;

- Chapter 5: covers the details regarding the proof of concept implementation of the architecture proposed in the previous chapter;
- Chapter 6: discusses the results of some simulations performed targeting the implementation presented in the previous chapter;
- Chapter 7: concludes the thesis and briefly mentions some of the possible avenues that can be explored to expand and improve this work.

Chapter 2

Background

This chapter comes as an overview on what are the current established and emerging technologies in the field of networking, and the related new aspects to take into consideration, especially regarding security [1]. Moreover some of the concepts related to security automation are treated, so as to give a background knowledge useful throughout the rest of this thesis.

2.1 SDN

Companies have a growing need for networking solutions that can be as scalable as possible. This is due mainly to the increasing reliance on cloud computing, driven by cost reduction prospects.

These changes have fueled cloud service providers' needs for more flexibility in network architectures, meaning the possibility to dynamically allocate network resources and also dynamically manage them.
In fact, the issue within the current landscape is that the capabilities of network devices and the network infrastructure, in general, are constrained by how its hardware was engineered by its vendor, thus inhibiting a scenario in which those elements are frequently reconfigured, both in location and functionality. For instance, the support for new network protocols would be cumbersome.

The integration of the software network architecture is an enabling factor for security automation, and its principles will be present throughout the work of this thesis [2].

2.1.1 Architecture

For this reason, a new architecture has been proposed and is at the moment being adopted or in the process of being adopted by many providers. Moreover, many investments in the research world are being poured into it, given its appealing features.
This new architecture is referred to as *Software Defined Networking (SDN)* and revolves around the idea of having two separate operational planes for every network device, one for control logic and one for the actual data forwarding. So each device can be viewed as working on two separate levels, the data plane and the control plane.
These two-level work in tandem to offer services to the application layer, the one which really makes use of the network capabilities. A general schema representing this architecture is shown in Figure 2.1

This new approach enables a programmable network architecture, in which configuration changes are done via software without necessarily modifying the actual network topology.
This new approach targets many issues encountered with other network architectures, thus accelerating the adoption of decentralized network applications and their management.
As far as security is concerned, SDN can enable quick response to security alerts via actions on

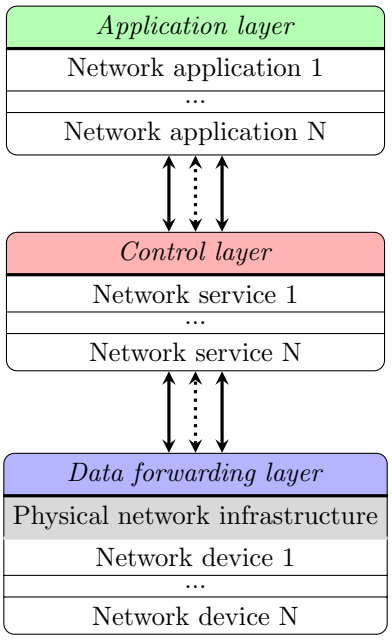


Figure 2.1. SDN Architecture overlay

network controls and facilitates the detection of those issues by allowing a global view of network traffic patterns.

A brief summary of the main advantages when using the SDN approach:

- Simplified network management: the SDN paradigm can heavily simplify network management processes, leveraging on its software abstraction layer, on which it is possible to build user-friendly management interfaces that provide a global sight of the network;
- Scalability: with a software network, it becomes easier to deploy new network resources or move them, avoiding the cumbersome process of moving physical network equipment from one area to another in order to meet network topologies changes;
- Cost reductions: software networks do not require specific equipment sold by certain vendors that must meet compatibility requirements. In fact, the management layer is software-based and independent from the physical network device.
  The traditional vendor lock-in issue is thus absent, giving freedom of choice to providers that can put costs as the first criteria in the hardware selection process;
- Centralized control: being software-based, it is possible to unify the control logic of the whole network, for example, by providing APIs that abstract the underlying network, whereas before, it was necessary to tap into each hardware device to modify its configuration.
  Moreover, APIs provide ways to orchestrate and automate network management;
- Standard-based: standards such as OpenFlow are gaining a foothold in the field, thus minimizing transition complexity.
  Moreover, the same can't be said for traditional network solutions where protocols and software connectors are proprietary.

2.2 NFV

Another innovation in the networking field worth mentioning is what goes by the name of *Network Function Virtualization (NFV)*. Its wide adoption, as with other new paradigms in the field, some of them subjects of this chapter, is driven by the move to cloud-centric solutions.

In particular, its employment results are useful for cloud providers to meet quickly evolving requirements from clients since it eases the deployment of new network services.

As is the case with software networks, this paradigm helps in the employment of automated solutions in the cybersecurity field [3].

2.2.1 Architecture overview

In a classic network environment, each functionality is expressed by a particular hardware device, and this implies that changing the way something works in those devices' functioning, or simply their position in the network topology, would require extended timeouts that could disrupt service operations.

The NFV approach consists of a transformation effort of those hardware devices that implement a particular network service into a software component to be deployed in a compatible environment.

At the technical level, it leverages new advances in virtualization technologies that reduce the performance penalties of virtualizing software that require a lot of computing power. The virtualization technologies used for this new architecture are both at the operating system level, leveraging dedicated hypervisors, or via containerization technologies, such as Docker.

For instance, a firewall or an intrusion detection system, which are typically deployed as hardware devices to be put in the middle of a certain network path, would be instead easily deployed as a new program to be run on a particular virtual environment.
This may be done in many ways, such as by launching a new VM with that capability or even more simply by spawning a new container with that capability, provided that the NFV environment supports both of them.
These software functionalities are called *Virtual Network Functions (VNF)*.

Main advantages encountered when adopting the NFV architecture:

- Speed up network management: network management can be done via convenient software solutions, in contrast with conventional network architectures, in which often it is required to physically connect to some equipment to manage or configure it;
- Cost reduction: network maintenance is greatly reduced since operations are done on software instead of on network equipment onsite. Moreover, NFV infrastructure can lay on white label hardware equipment, which is notoriously cheaper than hardware equipment;
- Flexibility: the NFV architecture allows the reconfiguration of network services almost in real-time, thus delivering unlimited optimization capabilities;
- Scalability: thanks to the flexibility it provides, the NFV paradigm allows the dynamic allocation and deployment of new resources and services depending on clients' demands and other parameters, such as resource usage optimization.

The European Telecommunications Standards Institute (ETSI) coordinated with a telecommunication vendor and proposed a standard architecture for the NFV paradigm, which nowadays is becoming one of the most used in the field.

2.2.2 Architecture standardization

The standard architecture of NFV is composed of three different components:

- Network functions virtualization infrastructure (NFVI): it represents hardware devices on which the virtualization software runs and on which virtual functions are deployed.
- Virtual Network Functions (VNFs): the network services to be deployed on the virtualization layer. Each one represents a network device in traditional network paradigms.

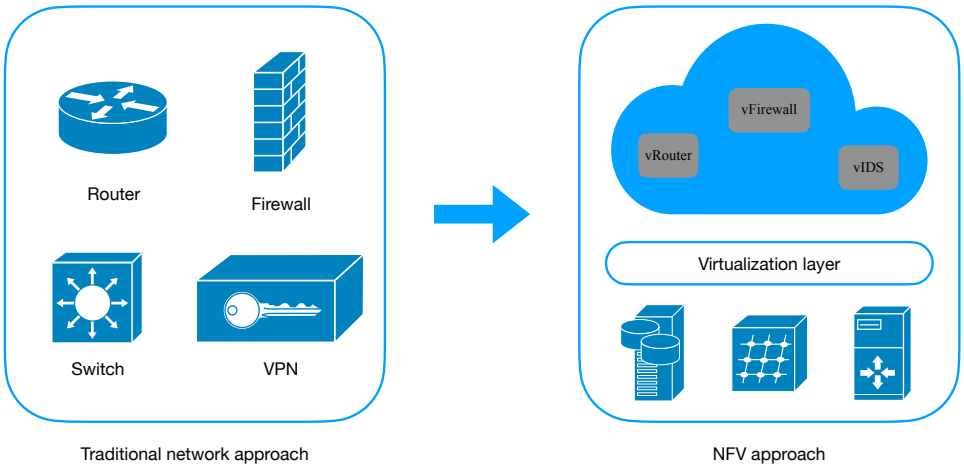


Figure 2.2. Transitioning to NFV architecture evaluation.

- Management, automation, and network orchestration (MANO): it is the framework that manages both the deployment of new VNFs and the resource allocation on the virtualization layer.

2.2.3 Service Function Chaining

The new NFV and SDN architectures, when combined, can provide an extremely flexible network landscape in perfect alignment with what cloud providers need, with the only requirement is having a network infrastructure supporting new protocols and architectures, but this is not an issue since the comprehensive offering of standard off-the-shelf equipment.

These new efforts have led to a new type of service deployment, the so-called *Service Function Chaining (SFC)*. SFC can be viewed as the application of the separation of concerns paradigm to VNFs, that is, assigning certain network functionalities to different VNFs, and chaining them whenever a combination of the different functionalities is needed on a certain network path.
This comes slightly in contrast to the conventional networking approach in which a single hardware device or VM provides all the capabilities required.

Given the high flexibility and modularity of this new network services architecture, it is clear that various optimizations can be applied for the optimal composition[4, 5] of a chain of functions that can deliver the best performance.
More importantly, the positioning of a VNF into the chain can be dictated by specific requirements, so particular care must be put into the design of certain chains.

2.2.4 Security Service Functions

A concrete usage scenario of SFC regards the enforcement of security functionalities.
Security Service Functions are a particular type of VNFs dedicated to enforcing security policies, such as blocking connections, detecting malicious traffic and network intrusions, mitigating *Distributed Denial of Service (DDoS)*, applying *Deep Packet Inspection (DPI)*, and so on.
Security Service Functions can carry out a wide range of different security tasks. Their capabilities can be stateful or not. For instance, some are just meant to forward traffic after having applied some type of control logic on it, but others instead can even maintain the state of the connections,

thus enabling more powerful security functionalities. Chaining different Security Service Functions corresponds to deploying different security functionalities to that network segment and, by doing so, guaranteeing depth of defense.

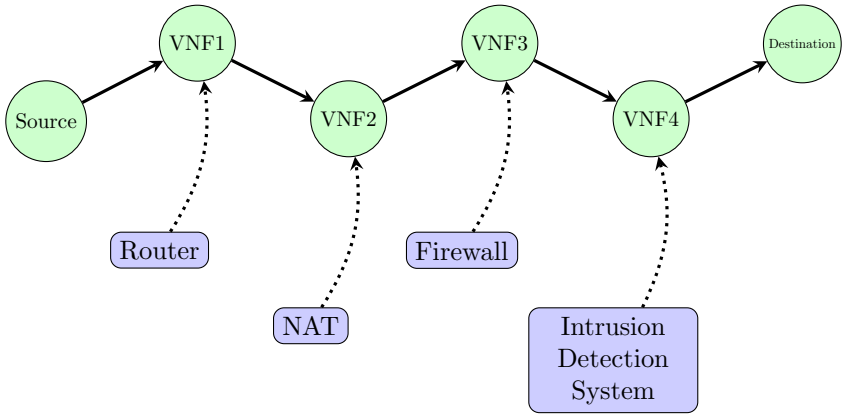


Figure 2.3. Service Function Chaining example

2.3 Security aspects

This section will give an overview of some of the aspects concerning security in modern networks architectures, such as those depicted throughout this chapter. A brief summary of basic procedures, or actions, that can be taken in response to security issues in network environments, such as those exploiting SDN features[6] will be presented towards the end of this section.

2.3.1 Software networks security pitfalls

The advent of new paradigms such as NFV and SDN are providing ever more value to cloud computing but also come with several potential security pitfalls if appropriate countermeasures aren't taken. In fact, those power features that enable new functionalities can also become single points of failure when exploited. Here is presented a summary of some of the main attacks mountable on network environment adopting SDN or NFV paradigms.

According to Reynaud et al. [1], we can mainly categorize attacks on software environments as those conducted on virtualization systems, those conducted on controllers, and other generic attacks that can be pursued on all networks, regardless of them being traditional or software.

- Denial of Service (DoS) attacks are the most common in the internet world, and as such, they also target cloud environments and, in particular, SDN and NFV environments.
  The impact of this type of attack can be more significant in virtualized environments in which, by its nature, more assets convey in the exact physical/network location.
- Software issues also pose potential security issues for network functions since they often lack extensive review and audit.
  This is certainly also true for traditional network equipment, but the nature of the approach pushing for rapid deployment may become counterproductive.
- Network functions are hosted on a layer of virtualization. This layer can be of different types, ranging from a full-fledged hypervisor to simple VMs or container platforms.
  They have in common the fact that also this layer can pose security risks, and it has been demonstrated by multiple pieces of research that vulnerabilities in this level can have strong consequences on service operability.
  In particular, in the cloud computing world, it may give access to resources not ordinarily accessible in the environment in which a function is hosted.

- Intra plane communications, that is, the channels with which different components or layers of software components talk with each other are usually dealt with application-level network protocols. Thus, particular care must be taken to their security properties since a compromise of these can compromise the entire apparatus.
- Centralization is sometimes seen as an important feature to have when dealing with network infrastructure and cloud infrastructure. However, given its far-reaching control of the infrastructure can become a big point of failure.
  Even in this case, vulnerabilities have shown up, making the risks more concrete. It goes without saying that strong protective measures must be taken to preserve controllers nodes, such as those in SDN environments.

Other sophisticated attacks against software environments do exist, and given the broad reliance on these new technologies, it is expected for these only to grow with time.

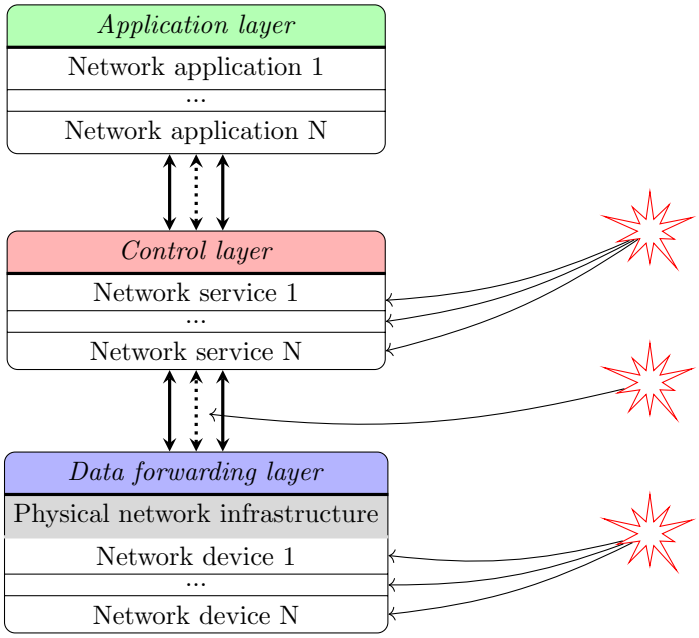


Figure 2.4. Targetable components in a software network architecture

2.3.2 Low level security actions

Some of the low-level actions that can be taken in response to security issues in a software network environment are here listed:

- *Traffic mirroring*: traffic monitoring is a low impact measure that nonetheless can provide a sufficient level of security assurance in those situations in which suspect activities were detected, for example, suspicious heavy CPU usage on certain hosts. In an SDN environment, traffic monitoring can easily be deployed by mirroring desired flows, in particular those originating from potentially compromised hosts, to network collectors that can then analyze in detail its contents.
- *Isolate host*: the most direct way to respond to a security breach or issue on a host of the network is to isolate it, that is, block all connections that have that node on the path. This is done by adding a rule to the SDN controller governing network flows to make it drop all related traffic.
- *Isolate switch*: SDN switch manage traffic between multiple nodes of the network. By isolating a switch, multiple network paths may get disrupted if no other redundancies in the network are available. So this measure heavily impacts the service operation.

- *Block links*: this measure is more selective than isolation measures previously presented since it doesn't block all connections between nodes but only certain ones. This measure thus requires a form of network intelligence capable of identifying those connections that are causing a particular issue.

2.4 Incident handling

The security landscape has seen major changes recently, and as threats grow in sophistication and number of new ways to manage security are being studied. The ultimate objective in all arguments concerning security is to avoid incidents impacting valuable assets or having general harmful outcomes.

According to the NIST, a security incident can be described as:

> *"a violation or imminent threat of violation of computer security policies, acceptable use policies, or standard security practices"*

Though a threat ultimately culminates with an incident affecting resources, services, and other assets, the steps and methodologies to be followed when handling it are not to be considered limited to the incident response. Instead, good incident management requires that multiple actions be regularly taken well before and well after the incident occurred. This action and procedure can be grouped in different stages.

Incident handling tasks are to be dealt with structured operations for all stages. Everything must have been established well before the actual nefarious event. Organizations must structure themselves so as to create appropriate teams and facilities that will have to handle in a seamless way all stages, from prevention to post-incident activities.

According to the NIST [7], incident handling can be viewed as a continuous loop of operations that take place before, during, and after an incident has taken place. These operations can be grouped in four different stages, each one targeting different macro tasks, following temporal criteria, with the moment in which the incident has been detected and a containment strategy is being decided as a central reference point.

It is clear that different stages may be covered by different teams, and in particular, it is not required, nor expected, for certain stages' tasks to be appointed to an incident response team. For example, it is instead common to leave the preparation stage to other teams, for instance, those dedicated to penetration testing and other similar activities. Human resources are a critical component in the incident handling life cycle. In fact, teams dedicated to different tasks must have the required skills and capabilities needed for their tasks. Moreover, each team must be provided with the appropriate tools that shall assist them in their job.

The incident handling life cycle stages are here summarized, and for each one, it is given a brief description of the main activities to be taken in it and the recommendations to handle it well. Moreover, in figure 2.5 it is represented a schema of the incident handling life cycle with its stages.

Following this methodology, i.e. the incident handling life cycle, permits organizations to perform incident response activities in a timely manner thanks to a predefined set of playbooks, thus reducing costs deriving from the incident and disruption to service operations.
Moreover, the constant evidence gathering effort guarantees a continuous improvement of those playbooks and of the procedures to be applied in all stages of incident handling. This effort may also be beneficial in the compliance direction since law regulations regardless require extensive documentation of incidents.

2.4.1 Stages of the incident handling life cycle

This part of the chapter shortly coveres each stage of the incident handling loop.

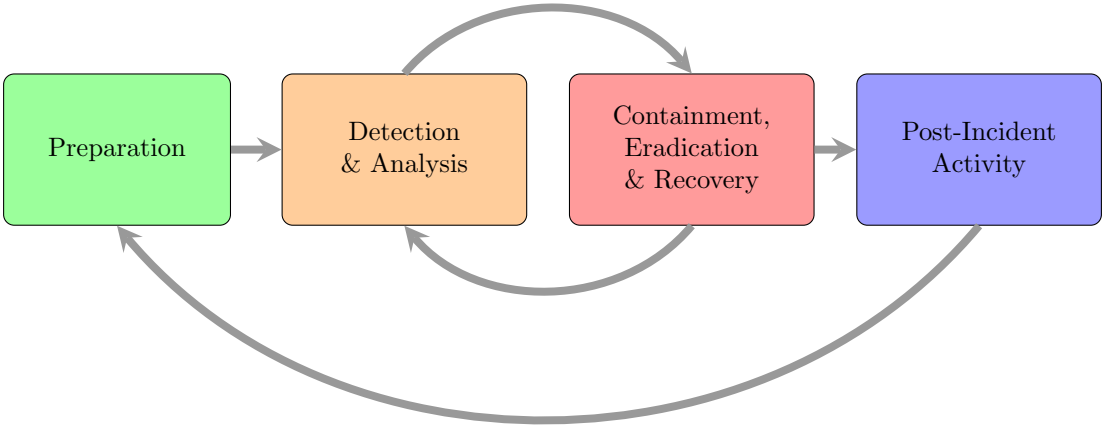


Figure 2.5. Incident handling stages according to NIST

Preparation

The preparation stage is perhaps the most important of all because it lays the foundation for the effective execution of all the others. This stage encompasses both the set of measures and methods that can assist whenever an incident occurs or has supposedly occurred and what it is due to be done to prevent it.

Given this, it is evident that much of the actions of this stage are not to be regarded as incident response activity. Nonetheless, it is correct to include them since it is part of the management life cycle.

From the human resources point of view, the roles must be clearly defined, and each task must have been assigned to its team. The tools and frameworks for handling an incident must be provided to the corresponding teams that must follow thorough training for their role.

Also, high-level aspects must be considered, such as the establishment of policies and guidelines, both for internal usage and law compliance. A road map must be outlined to define precise steps to follow when an incident occurs, thus forming an Incident Response Plan. The prevention part must include regular assessments and all the general aspects of it.

Detection and Analysis

This stage includes all those operations that revolve around detection, analysis, and reporting.

The detection phase, often automatized, must, however, always have human assistance to avoid false positives that may impact the uptime of the service. This phase can and should make as much as possible use of intelligence feeds so as to expedite the detection of new threats by means of sharing indicators of compromise.

The analysis phase is meant to take all possible shreds of evidence of the incident and document it by means of the tools specialized in this, infrastructure logs, and so on.

With the previously collected information, the incident analysis must be reported for a follow-up to the teams dedicated to the next stage. Eventually, also higher-level departments should be notified in case the incident may have a significant impact.

However, prior to it, incoming alerts must be prioritized. This is a direct consequence of the massive number of alerts incoming continuously, so skipping this step can lead to incident response teams being clogged by alerts and missing the most important ones.

Containment, Eradication, and Recovery

This stage is composed of three different phases and together represent a phased incident response approach. The containment phase basically involves a short-term strategy to mitigate damages while a long-term strategy is laid down to eradicate the threat effectively.
Each type of incident requires a tailored containment strategy since sometimes a not suitable one may inflict more damage than if no containment is done.

The eradication phase eliminates the threat but does not restore service to normal operational status. A complete return to normal activity is performed within the recovery phase. Recovery must be thoroughly tested and validated, leading to recovery certification.
All phases must be accompanied by an extensive production of evidence and logs so as to be able to reconstruct the whole attack history together with the incident response. Evidence gathering may also be helpful, or even necessary, for compliance with laws and regulations.

Post-Incident Activity

This is the last stage of one cycle of the incident loop. At this point, all previously gathered knowledge about the incident, up to the threat and which was the reaction to it, are reviewed and modeled in a structured way to produce a report.
This review will result in the perfection of already established procedures and methods that make up all stages of the incident handling life cycle.
For example, it may come up that some assets' protections must be enhanced or additional preventive measures must be taken globally.

2.4.2 Cyber Threat Intelligence

Here a brief introduction to the subject of Cyber Threat Intelligence is given.

Overview

All gathered knowledge is stored for future usage inside the organization but can be of further use if shared across organizational boundaries.
The sharing effort should permeate in all stages of incident handling, thus even before the incident has been completely recovered.
Sharing intelligence can happen with different techniques, each offering different capabilities. According to the NIST, currently, the most used approach to shared intelligence is entirely manual. This implies that intelligence is shared between organizations and teams by links that are created not specifically for this scope.
This has many side effects, such as being impossible to share intelligence in a structured and coherent way across time and also lacking consistent time frames for when intelligence is actually shared.
This lack of formalization makes it impossible to really create value in what is shared. Examples of intelligence sharing that follows this style include email sharing, but also articles on companies' blogs, etc.
The preferred way to handle intelligence sharing is instead one in which the information processing and then its sharing is done in an automated way.
One issue with automated intelligence sharing is the lack of standards for information exchange. Another aspect that must be considered is the fact that an organization willing to automatize the process of intelligence sharing must have previously defined a framework that can be used to structure information and decide what should and should not be shared. In the end, human interaction may still be needed, especially in those situations in which some information to be shared is actually to be kept private. For this reason, some security considerations about the granularity of shared information must be made.

The current landscape of cyber intelligence sharing is growing every day, both in size and quality of the intelligence, which is an important parameter to look out for.

In particular, standards are emerging and are being adopted more and more by companies and organizations. Various initiatives are contributing to the integration of shared intelligence inside organizations' security practices. Some of these initiatives include the STIX standard.

ISACs

ISAC stands for Information Sharing and Analysis Centers, and as per the definition given by Sholihah et al. [8] are non-profit organizations that provide services to collect information on cyber threats, which can later be used for information sharing between the public and private sectors.
ISACs play a fundamental role in the Cyber Threat Information landscape as they allow sharing of knowledge that can be used to tackle the various stages of the incident handling life cycle.

Many initiatives are emerging in this context, thus providing ever more diverse knowledge, often of high quality. All this information can have a substantial impact on the security posture of organizations since it puts these in a position of advantage over the malicious actors that now more than ever are leveraging on the threat-as-a-service model.

The MISP Threat Sharing platform

The sharing effort leverages various initiatives for its functioning, one of these is the MISP platform.
First named Malware Information Sharing Platform, then MISP Threat Sharing, it is one of the main initiatives in the field of cybersecurity for what concerns sharing knowledge about malware and threats.
It is gaining wide support from public entities and the private sector.
Its sharing activity follows a hub and spoke model 2.6, in which the MISP platform works as a hub, gathering all collected knowledge, including from partners that chose to participate in the sharing process.
In this model, participants are either producers, producers, and consumers, or consumers only of threat intelligence. This enables various levels of collaboration. Organizations that choose to act as producers of cyber threat intelligence must first and foremost decide what type of intelligence to share with the communities and in what detail. This is mostly dependent on the confidentiality of the information and may be efficiently managed by means of solutions like the TLP (Traffic Light Protocol).
The MISP platform allows third parties to participate in the sharing effort by supporting different protocols such as TAXII.
A CTI consumer can instead leverage the export functionality of the MISP platform, which allows exporting knowledge contained in the MISP database into STIX structured data to make concrete use of it.

2.4.3 Indicators Of Compromise

One of the ways Cyber Threat Intelligence helps in fighting and preventing cyber incidents is by the sharing of IoCs.
Indicators of Compromise (IoC) are pieces of forensic data observed in an operational environment, such as network traffic or system logs, that is likely related to malicious activity, for instance, an intrusion into the systems.
Examples of IoCs can be malware hashes that can be detected in the file system, IP addresses that can be related to malicious traffic, such as exfiltration, that can be detected on network logs, etc.
IoCs are produced as a result of threat analysis, done by security analysts devoted to this particular effort or directly by incident responders as part of the documentation effort needed in the stage of incident response.
IoCs are used in threat hunting inside the infrastructure perimeter to establish the capability of collecting and correlating IoC, for example, when human analysts overwatch system and network

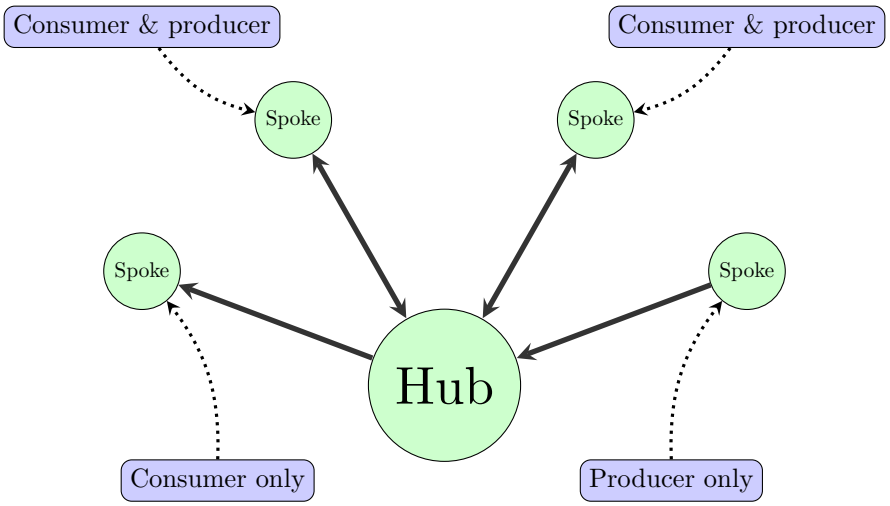


Figure 2.6. Knowledge sharing based on the hub and spoke model

logs or by adding them to intrusion detection and monitoring tools that passively block intrusion attempts.
Often indicators are looked together so as to improve the detection capability of sophisticated attacks, as could be an APT, in their early stages.
The continuous real-time monitoring of previously detected IoCs and IoC shared by other organizations or entities helps strengthen the security posture and enables more reactive response activities to potential incidents.

A typical incident handling workflow in environments that adopt the Cyber Threat Intelligence sharing schema and that make use of indicators of compromise follows.

1. Organization A detects suspicious network traffic in its network logs, and after some analysis, security analysts correlate it to one host showing signs of compromise. Incident response operations take place, and alongside them, various IoCs related to the threat are generated. These IoCs take the form of some IP addresses, together with connections details and other system information;
2. Organization A, after having recovered from the attack, generates a report of the incident containing the IoCs and shares it with an ISAC;
3. After having reviewed and certified the report, the ISAC, in turn, shares it with its partner organizations;
4. Organization B, that partners with the ISAC, receives the report containing the IoCs generated by Organization A;
5. Organization A after reviewing the report integrates the IoCs into its threat hunting and incident handling workflow and immediately detects traffic that corresponds to the recently added indicators of compromise;
6. Organization A promptly adopts dedicated security measures and investigates its systems looking for other signs of compromise so as to avoid further damages. Its incident response teams later discovered that the attack was in an early stage, and if actions weren't taken, it would have compromised in-depth other organizations' assets.

2.4.4 Security automation

In the previous paragraphs, it has been presented a standardized way of approaching cybersecurity and incident handling in particular.

Other aspects must be considered when establishing a solid strategy for incident handling. In the past few years, the sophistication of attacks has been increasing, with the entrance of new threat actors, such as state actors, and the birth of new types of threats, such as Advanced Persistent Threats (APT), that pose major risks to organizations.
Moreover, the commodification of exploits and vulnerabilities is shrinking the time from when a new vulnerability is found to when it is exploited, and at the same time, the attack surface is continuously expanding.

The cybersecurity scenario has evolved over time to meet new landscape requirements, and some of the efforts in this direction include the adoption of Security Information and Event Management systems (SIEM), tools that assist in managing the Detection & Analysis stage, thus decreasing the detection time of compromises, and the employment of Security Operations Centers (SOC), that are teams dedicated to different stages of the incident handling life cycle.

Many of the stages forming the structured strategy proposed by NIST have in common the fact that a lack of specialized personnel may hinder the effectiveness of their operations. It is agreed in the academic world [9] it is of utter importance to establish automation strategies for all stages of the incident handling life cycle, and the NIST guidelines [7] make it explicit. This is motivated by wide evidence on the ground that integrating automation may not only help with personnel shortage but also significantly decrease incident handling life cycle duration 2.7 and, as a consequence costs 2.8.

Automation efforts must build upon already established capabilities, such as those of alert triage, but additional steps may be taken to increase its impact on incident handling, and one concrete approach that is emerging is the integration of automation in the process of intelligence sharing, that will be covered later on in this thesis. Besides many other advancements are unlocked by embracing automation capabilities, Nespoli et al. [10] have presented an in-depth study on the current security landscape, in which they corroborate the urgent need for response strategies that leverage security automation.

Many operations that are low priority, meaning that they cannot impact the service operability, are the perfect target for automation efforts.
Here are some use cases related to incident handling where automation adoption could improve the security posture.

1. Identification of unusual network patterns: repetitive identification tasks that are often done manually can be easily automated. For example, trends related to network traffic;

2. Integration of Indicator Of Compromise: the integration of IoC is usually done manually by security analysts. These can include IP addresses, hashes related to malware, etc.;

3. Incident response: often, minor incidents or threats can be dealt with by applying standardized actions that do not require human intervention since they do not impact service operability. For example, temporarily filtering traffic from certain IPs contained in recently received IoCs or that are showing DoS behaviors.

Let's take into consideration the scenario depicted previously regarding the sharing of an indicator of compromise to show how intelligence sharing operations are handled traditionally, clarifying the pitfalls and inefficiencies, and after that, the automated approach will be presented.
In step 2, Organization A shares the incident report with the ISAC, but no predefined process was established before, so this operation probably would take a lot of time and would probably be conducted by means of e-mails, and in any case, not in machine-readable formats
In step 3, the ISAC receives the report, but now it will take time to interpret, review, and model the received data so as to make it compliant with other organizations sharing protocols.
In step 5, once Organization B has received the redacted report from the ISAC, it has to integrate it into its tools and workflows. However, since the report does not come in any machine-readable format, further work is required to translate it into actionable knowledge, and in particular, to extract the IoCs present in it.

Threat feeds

Threat feeds consist of a continuous stream of new data that is integrated into an organization's security apparatus so as to communicate the latest information about the threat landscape, allowing to predict and prevent an external threat before they even cross an organization's perimeter. A threat feed contains context information about a threat, such as timestamps, indicators of compromise, also called artifacts, such as IP addresses and malware hashes, and so on.

The adoption of threat feeds alongside threat sharing aims at overcoming major pitfalls that are present in threat intelligence sharing processes. In other terms, automation tackles the logistics issues of intelligence sharing by transforming manual operations into automated ones.
Taking into consideration the IoCs sharing scenario discussed above, various differences can be highlighted in those steps.
First, by using threat feeds, the usage of inappropriate communication channels is avoided. In fact, once IoCs are collected, it is sufficient to pass them to the threat feed endpoint. At the other end of the channel, Organization B, which had previously subscribed to the same feed, will consume the messages received in which the IoCs are present.

Threat feeds are therefore capitalized on by integrating them into SIEM platforms so as to make that information actionable, for example, by comparing it to internal telemetry. Nowadays, the security automation scenario is evolving towards integrated platforms, analogous to those used for the Detection & Analysis stage, namely SIEMs, but with the addition of additional capabilities, specifically aiming at managing in a proactive way the response stage.

Security Orchestration, Automation, and Response is the new paradigm for incident handling that these platforms are following. Often SOAR solutions aim to integrate SIEM functionalities so as to offer an overarching framework for cybersecurity incident handling. SOAR platforms integrate all security controls in a single platform endpoint together with other security tools and sources of information so as to enable security automation.

Later on in this chapter the automation subject will be covered in more detail, especially for what concerns standard formats and languages related to incident response scenarios.

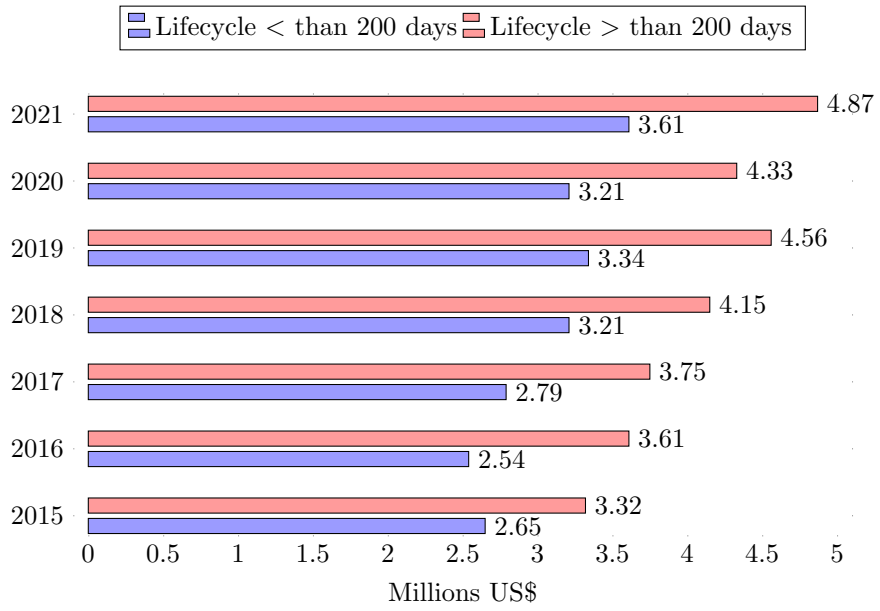


Figure 2.7. Average total cost of a data breach based on average data breach life cycle. Source:[1]

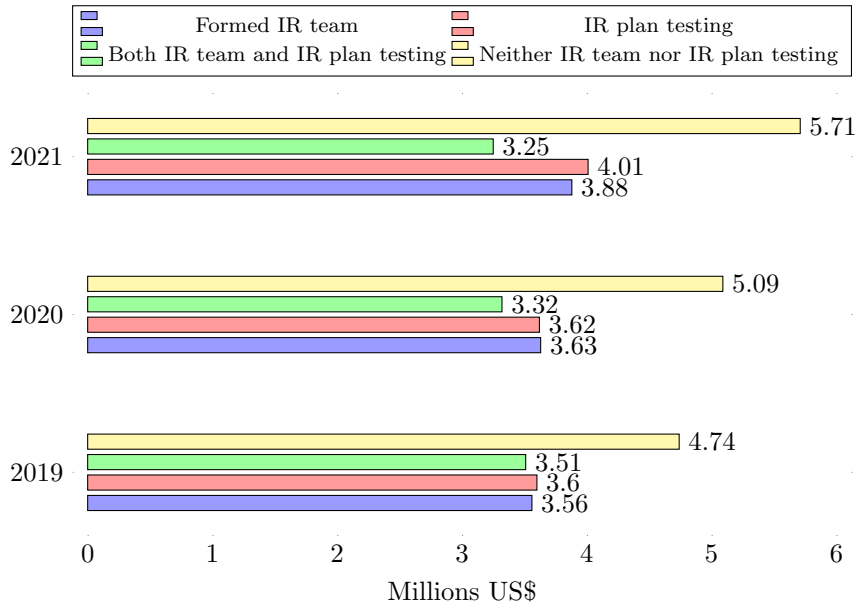


Figure 2.8. Average total cost of a data breach with incident response (IR) team and IR plan testing. Source:[3]

[1] https://www.ibm.com/downloads/cas/OJDVQGRY

[3] https://www.ibm.com/downloads/cas/OJDVQGRY

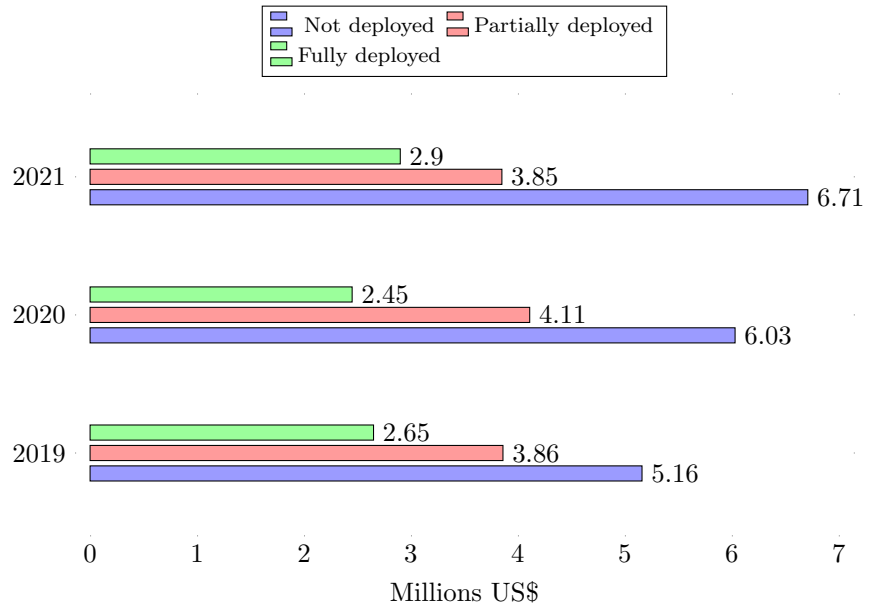


Figure 2.9. Average cost of a data breach by security automation deployment level. Source:[4]

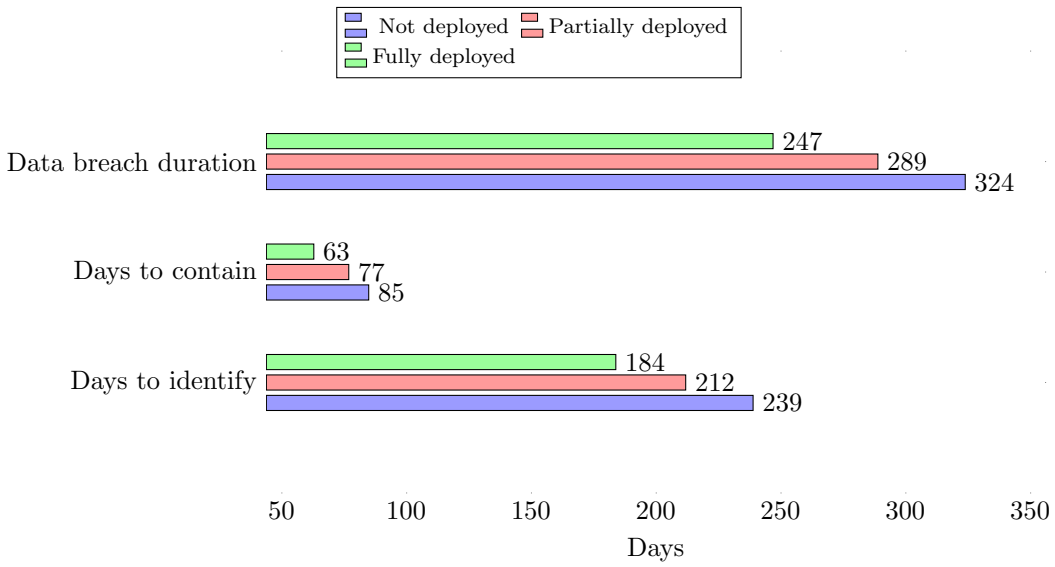


Figure 2.10. Average time to identify and contain a data breach by level of security automation. Source:[6]

[4] https://www.ibm.com/downloads/cas/OJDVQGRY

[6] https://www.ibm.com/downloads/cas/OJDVQGRY

2.4.5 STIX language

The STIX language, acronym of Structured Threat Information eXpression, is a standardized structured language, using JSON as format, that aims to improve different aspects of cybersecurity threat intelligence sharing and usage. The STIX initiative is maintained and brought forward by the OASIS consortium, and it follows an open development approach, so companies and other organizations may help drive its adoption and help develop new functionalities. The last version 2.1 has been designed to improve interoperability and usage across organizations [11]. The STIX language is paired with the TAXII transport format, which is an application layer protocol designed exclusively to support different logistics approaches to cyber threat intelligence. However, STIX provides extreme flexibility even in how the information is shared, thus not enforcing the use of its exchange format, leaving the ultimate choice to the organization. Some of the main characteristics of the STIX language are here summarized:

- *Flexibility*: the language permits the usage of only the set of components that are strictly required for the particular use case for which it is being used.
- *Extensibility*: given the high mutability of the cyber security landscape, it has been designed to be highly extensible so as to give the possibility to personalize its usage inside an organization. This feature will be particularly useful for the implementation of this thesis.
- *Automatability*: thanks to the consistent architecture and how the content is organized, it is easy to automate processes that digest STIX content, all without human intervention.
- *Readability*: In spite of the machine-friendly format and structure, even the human readability is taken care of. In particular, a graph representation of the contents o a STIX report may be very appreciated by human analysts.

Various use cases concerning cyber security management have been supported since its first iteration, covering all stages of the incident handling life cycle. Here comes a summary of the main ones according to their relative stage:

- Cyber Threat Analysis: the second stage of the incident handling life cycle comprise activities and tasks meant to gain a comprehensive understanding of threats. This ranges from their characterization to studying their behaviors and producing relevant courses of action for related response management;
- Cyber Threat Modeling: another effort in the context of threat management regards constructing a model for a certain threat, potentially making use of existing threat models and taxonomies. This culminates with the selection of indicator patterns that are significant for the detection stage, while at the same time, threat models help in building an effective incident response strategy;
- Cyber Threat Response Management: remaining stages fall under the more general threat response management and span from general preventive measures that are taken in order to prevent threats and avoid related incidents to appropriate courses of action that are implemented after a thorough investigation and evaluation of the attack.

Structured threat information can help in all those tasks involved in these activities.

General Architecture

From a high-level perspective, the STIX language can be viewed as a connected graph where nodes can be of two types, STIX Domain Objects (SDO) or STIX Cyber-Observables (SCO), while edges represent STIX Relationships (SRO) between nodes. Actually, relationships between nodes, that is, edges, can also be defined by embedded relationships. These are defined inside an SDO without the need to define a new SRO object. It follows a summary of the high-level view of objects with a brief description of what they represent.

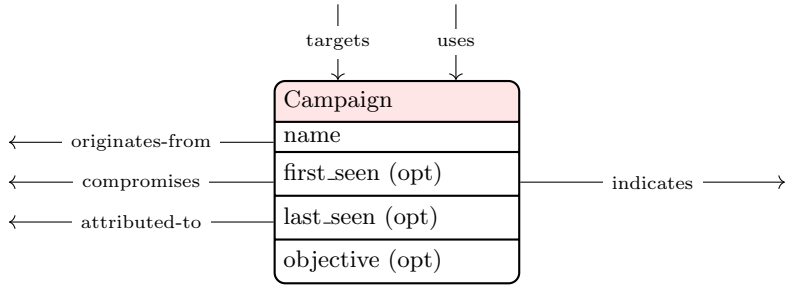
- *STIX Domain Objects (SDOs)*: describe core concepts of cyber threat intelligence, such as ad attack pattern, an indicator of compromise, a vulnerability, and so on;
- *STIX Cyber-observable Objects (SCOs)*: these objects are used to document various parameters and metrics concerning the state of certain resources, which can be network resources, system resources, and so on. They can be associated with an SDO to enhance their context. For instance, an Observed Data object may be attached to an Indicator object, meaning that the indicator was sighted. In this case, attaching details regarding what was sighted is done by referring to various SCOs from the Observed Data object;
- *STIX Relationship Objects (SROs)*: they represent edges in the STIX graph model, that is, how objects are related to each other. For example, it can represent the fact that a vulnerability has been exploited by a certain threat actor, and this relationship between vulnerability and threat actor would be represented with an SRO object;

The objects discussed up until now represent the core of the STIX language. However, other objects exist, called STIX Meta Objects (SMO). These will not be covered here since they do not add any content but are instead used to extend the language.
Apart from these, a STIX Bundle Object exists, which is simply a wrapper for all the contents defined within a CTI report, that is composed of the previously mentioned objects.

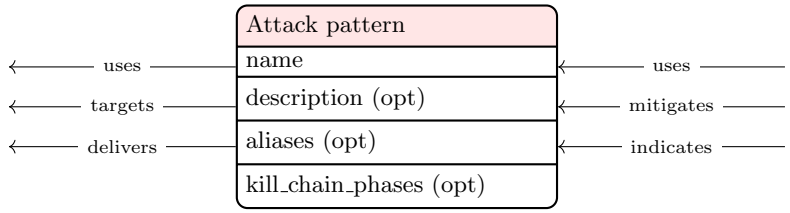
STIX Domain Objects

Next, in this section, a summary of the main STIX Domain Objects will be presented. For each one, the main relationships that the object may have with others are shown as arrows connected to the node. The direction of the arrow indicates if the relationship targets other objects or the object is the target of the relationship. In the latter case, the arrow is pointed to the node.
Also, for each node, some of the properties that can be set for each node are shown.

- *Campaign*: the campaign object represents a grouping of evidence and other information regarding various incidents that have taken place and have different aspects in common, such as the threat actor, the vulnerabilities exploited, etc.

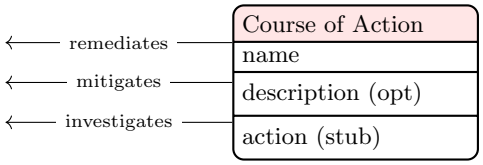


- *Attack pattern*: the attack pattern object is used to specify a type of attack, for example, a type of malware, and is often used to refer to maintained taxonomies such as CAPEC.

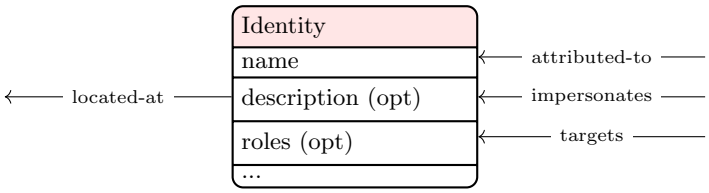


- *Course Of Action*: the course of action object is used to describe a set of actions that have to be taken or have been taken to respond to an incident. However, the object is

just a placeholder since its structure is subject to a future specification. For this thesis, an extension of the STIX language has been used to carry course of action information.

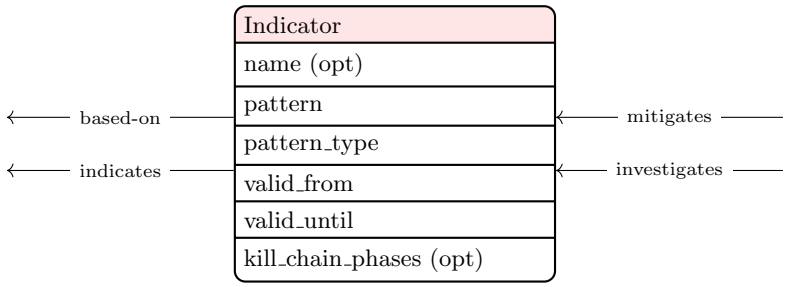


- *Identity*: the identity object is used to represent people, organizations, or other entities. This object is used to reference the owner of some assets or a threat actor's supposed or confirmed identity.

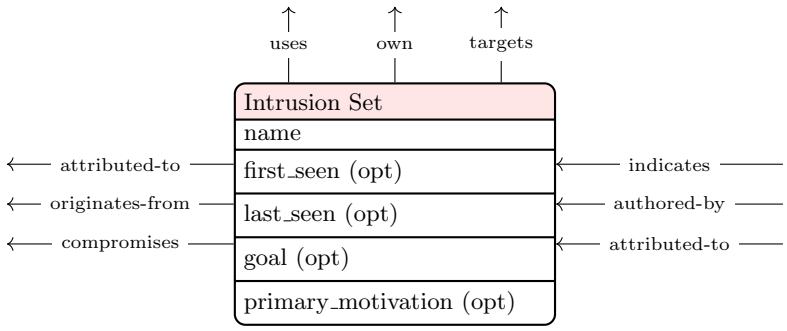


- *Incident*: the incident object is meant to describe all details regarding an incident but is currently only a placeholder and is suggested to extend the language for its usage.

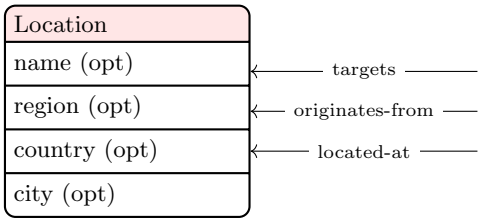
- *Indicator*: the indicator object represents the overly common indicator of compromise belonging to the cyber intelligence world. In STIX, when coupled with a pattern, delivers an automated way of detecting threats according to the given indicators.



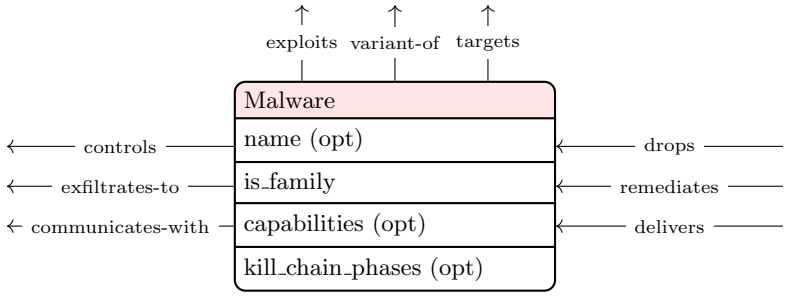
- *Intrusion Set*: the intrusion set object represents a catalog of the techniques used by the threat actor to bring forward its malicious activity. In the context of a campaign, the intrusion set represents all techniques used throughout the entire span of the campaign.



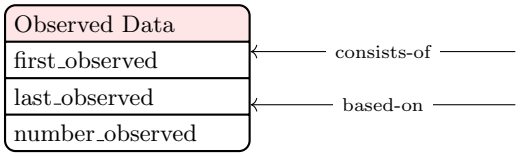
- *Location*: this object describes a location to be intended as a physical location. This may be used, for example, to add a geographical reference to where an incident occurred or to the supposed country of origin of a state threat actor.



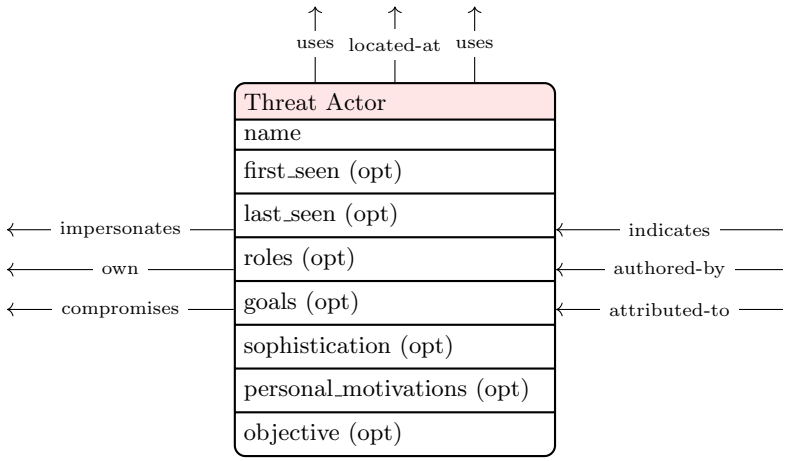
- *Malware*: this object represents a malware instance and all details regarding it. It belongs to the Tactics, Techniques, and Procedures (TTP) taxonomy. However, it should be used as an indicator object. That is, the details concerning how to detect an instance of malware should not be contained in this object but instead in an indicator object.



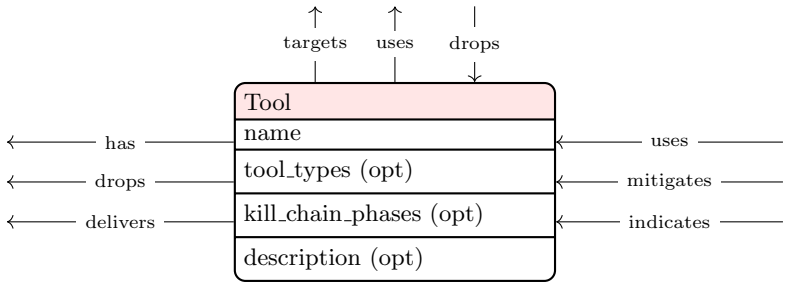
- *Observed Data*: this object is meant to represent collected data regarding assets of security relevance. Relevant data may be, for example, traffic data, connections, or files. This object finds its usage, for example, when related to an indicator object, to specify that such indicator was detected.



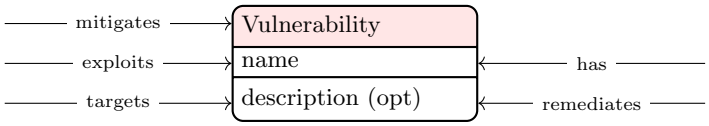
- *Report*: the report object is used as a grouping that brings together related intelligence objects representing different data. This grouping is useful for sharing purposes since it provides a comprehensive cyber threat story. For example, after an incident, a report can be produced containing the details of the attack, together with the remediation course of action and various indicators.
- *Threat Actor*: this object represents individuals, groups, or organizations supposed to be the subject of certain adversary operations. It may be related to a certain Intrusion Set and may be the actor behind a malicious Campaign.



- *Tool*: this object contains details regarding particular tools that can be used for the enforcement of a specific course of action or to detect indicators.



- *Vulnerability*: this object represents a vulnerability in some of the assets belonging to an infrastructure.



A real life application of the STIX language

In figure 2.11 it is shown a graph representation of an intelligence report regarding a botnet campaign. This report, albeit basic, since it is suited for media dissemination purposes, shows how the STIX language can be used to put together different intelligence information regarding a threat or incident in a cohesive and human-readable way.

This report covers the intelligence discoveries about a botnet. The botnet behaviors have been analyzed, from how it infects victim hosts to how the victim hosts call back to the command & control server. From this evidence, it has been deduced that the model used by the threat actor is of commoditizing its activities. So the targets are not related to each other.

The report, as far as threat detection and prevention is concerned, contains four indicators related to the malware object, each one with its own STIX pattern.
This pattern may be useful, for example, when integrated into a threat detection tool.
Each indicator covers different phases of the incident, from the initial infection to the actual relay back to the command and control server.
This categorization of an indicator by phases is provided by the kill chain property of the indicator object. The kill chain is a threat model that is gaining momentum in the cyber security defense world and will be covered in a different section of this chapter. Reports meant to be used across cybersecurity branches of organizations for threat defense and incident handling operations usually contain many other objects and as many details as possible about the threat.

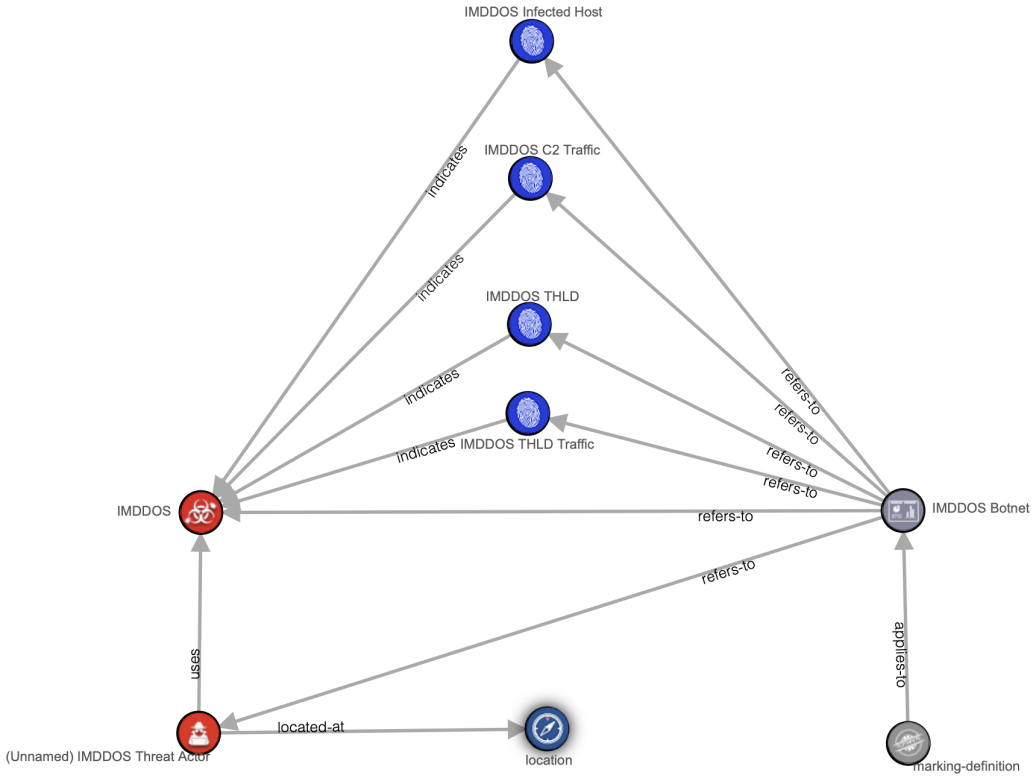


Figure 2.11. IMDDOS Botnet Report. Source:[7]

STIX Cyber-Observable Objects

As previously said, these objects are used to represent details about observables. These can, for example, include system-level parameters such as registry keys, network traffic patterns. Here it is reported one of the most commonly used Observable that is the Network traffic one:

Network Traffic
src_ref (opt)
dst_ref (opt)
src_port (opt)
dst_port (opt)
protocols (opt)
ipfix (opt)
...

STIX Patterning

Indicator objects adopt a specialized pattern language for describing matching rules that can later be used to detect when the activity corresponding to that indicator has been effectively seen. To handle this job, a custom patterning language is used. The language is thought to assist in the automated detection of activity related to an indicator.

[7] https://www.coresecurity.com/core-labs/publications/imddos-botnet-discovery-and-analysis

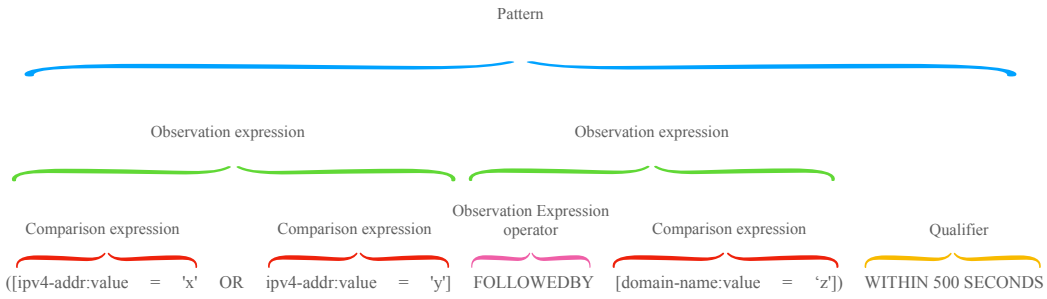


Figure 2.12. Example of a STIX Pattern

The figure 2.12 shows a STIX pattern with highlighted its grammar components. This pattern, when inserted into an indicator object, provides a way to detect that traffic pattern in collected traffic analytics.
Semantically the pattern is composed of three comparison expressions that have to match with the observed data and regard IP protocol information and DNS protocol information.
An observation expression operator is used to add a constraint to the clause. That is, the pattern is matched only when the first two comparisons match and within the same time frame specified by the qualifier WITHIN the third comparison expression is also matched.

2.5 Incident response automation

As previously discussed critical aspect of all platforms offering automation solutions, and in general of automation efforts in incident response, is a lack of standardization.
As has been previously discussed, intelligence sharing is a pillar of incident handling capabilities. In fact, shared information may be able to not only anticipate incoming threats but also help prevent them or speed up later stages, such as containment, eradication, or healing. Thus to get the most out of incident handling automation, proper standards and formats for the representation and exchange of this information are required.

In order to support security automation, cyber threat intelligence, and in particular, the incident handling part, should be easily machine parsable while at the same time maintaining human readability so as to support analysts and incident responders.

In the literature, languages for managing security related procedures and policies do exist, especially related to the management of network and application security. Valenza and Lioy proposed a user-oriented language meant to be used for the specification of high-level security policies [12].
Their proposal aims to facilitate the security management of personal devices or of various equipment in SOHO environments, with the assumption that the end user does not have the required expertise to handle low-level security policies.
This approach is based on three abstraction layers, an *High-level Security Policy Language (HSPL)*, a *Medium level Security Policy Language (MSPL)* and the low-level configuration meant to be modified.

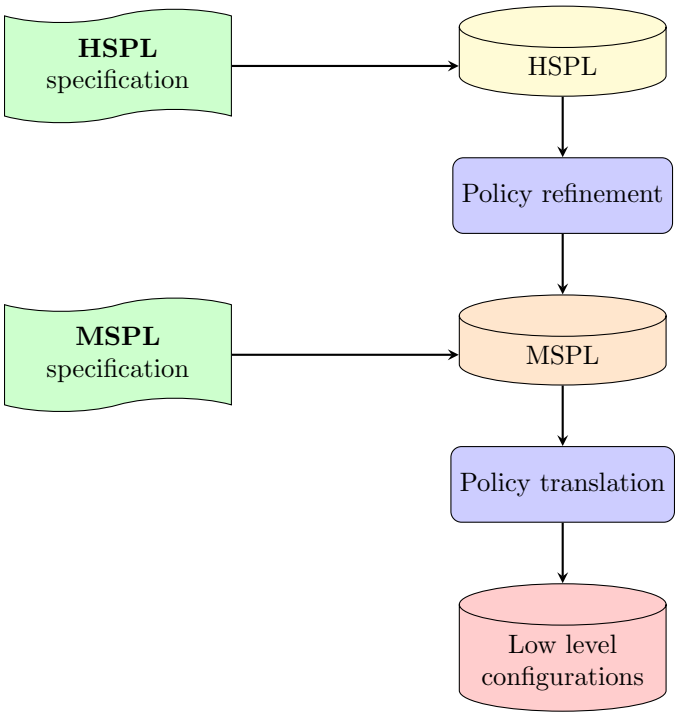


Figure 2.13. Security policies languages

The distinction between two languages, a high-level one and a medium-level one, is given to offer two levels of granularity to the end-user. So as to offer a simplified version to a user who doesn't have the sufficient skills to handle complex low-level configurations and a more powerful but complex way of expressing system policies by means of the medium level language.
The MSPL language has been designed as an abstraction of low-level commands belonging to different security controls. Thus the same MSPL statement can be converted to the same low-level configuration in different operational environments, even when security controls of different vendors, not compatible with each other, are present.

The process with which HSPL statements are converted into low-level configurations requires two passages. The first one is a refinement process that takes as input HSPL statements and yields MSPL statements. The second one is a translation process that takes as input MSPL statements and yields low-level configurations to be applied to the operational environment 2.13.
In case security statements are directly expressed in MSPL, skipping the HSPL, only the translation process will have to take place.

High Level Language overview

The basic element of the language is statements. Each statement expresses the action of someone that interacts with a security asset and adds a new security policy to it.
Security assets are any kind of network device that may be programmable but can also include other assets, with the only requirement being having an interface with which their behavior can be programmed.
Various instances of these types of assets may be software, perhaps controlling network settings, operating system interfaces, etc.

Follows the semantic of a HSPL statement:

[**subject**] **action object** [(field type,val)...(field type,val)]

- *subject*: the "subject" element represents the user identity that is enforcing this policy;

- *action*: the "action element is used to specify the type of operation that will be enforced on the object entity. In the case of endpoint security, some of the examples are the authorize access, limit speed, etc.;

- *object*: the "object" element represents the entity that executes the action, or that is the target of the action. For example, an object may be DNS traffic, and in this case, an action can express both a constraint to the traffic, such as block queries to a certain domain.

This element may come with additional parameters that are tailed to it. These parameters can add context to the object so as to delineate the scope of the action.

This language, though, as can be seen from the previous brief description, was not designed for security automation and the description of courses of action. In fact, it lacks fundamental features such as the possibility to express conditional logic. Overall this effort was limited to the management of security policies in situations far apart from the typical incident response one.
In spite of this and other drawbacks, it demonstrates one of the first attempts at building languages for security automation and management.

2.5.1 Security playbooks

Incident handling, as previously discussed, requires the coordinated effort of different teams, using different tools, and working on different operations, ranging from threat detection and auditing to incident recovery and documentation.
Overall those operations may be described as a set of actions that have to be executed procedurally, possibly with some form of conditional logic in it.
These security processes involve various tools or human capabilities and are usually documented in various formats, such as business-related ones.

Playbooks are an effective method to represent those types of information, following a workflow model in which actions and operations to be executed are represented by the steps in the workflow. In a playbook, different procedures can be represented by linear steps or more complex patterns that follow some sort of conditional logic. This characteristic helps to establish automation in the security operations pipeline, such as in incident handling and response, by providing a playbook for each common procedure, such as notification tasks, containment, remediation, and so on. For each operation, different Playbooks should be maintained, each one tailored to a class of threat, incident, or tasks, and simultaneously they can get validated by carefully analyzing the impact they would have on the target systems and the success factor. Playbooks are then regularly updated and perfected, making use of the knowledge acquired within the organization or through threat intelligence sharing. Their usage is an essential requirement in the transition to security automation in that it can provide a means to orchestrate security operations across all tiers of the security personnel.

The STIX language provides a dedicated object for specifying these types of security processes, with the Course Of Action object that is meant to be used for documenting security processes. In particular, the STIX specification states that it should be used to describe certain incident handling operations, such as investigation, mitigation, or remediation.
The STIX standard specifications, however, stop here and do not specify how these playbooks should be structured nor what their format should be, basically leaving the object as a stub for future extensions.

Various standardized approaches have been proposed to this end, with some limited to a basic network management scope and others more suitable for the description of workflows from a high-level business perspective. A typical example of a high-level playbook for the response to the detection of malware on a host is here described. It contains both high-level manual operations, such as notifying another person of a particular event, and low-level ones, such as rebooting a machine. Given the nature of the different steps composing a security pipeline, in this case, described through the playbook abstraction, it is the norm to assign them to teams dedicated to different tasks and of different tiers in the organization structure.

This playbook instance, in particular, is meant to be executed in response to a fictitious malware that infects hosts and is capable of persisting upon reboots if specific clean-up procedures aren't followed.

1. Open a ticket with level 2 priority;
2. Quarantine the infected host to sandbox VLAN;
3. Call level one desktop support;
4. Delete registry keys and triggers related to the malware on the system;
5. Reboot the system into safe mode;
6. Kill process named sgjbn.exe and then the one named ssyso.exe;
7. Delete all temporary files on the system;
8. Delete all compromised files on the system;
9. Delete other related registry keys on the system;
10. Reboot the system into safe mode;
11. Verify that malicious processes do not restart after the cleanup;
12. If the cleanup steps have not worked, escalate to higher levels, eventually up to the management level;
13. Patch defensive systems and scan the system with updated signatures;
14. Reboot the system to normal mode;
15. Update the ticket;
16. Move the host from quarantine VLAN to the original position to restore service operations.

CACAO Security Playbook

CACAO stands for Collaborative Automated Course of Action Operations for Cyber Security and is the product of a joint effort of various actors in the Cybersecurity field, both governmental and private, spearheaded by the OASIS organization, to define a structured language for writing machine parsable playbooks [13].
This effort was motivated by the ever growing need of automating processes concerning cyber defense operations, and for this reason, CACAO has been proposed as a standards-based and machine-readable solution for documenting and describing steps needed to prevent, mitigate, remediate, and monitor responses to threats, attacks, or incidents.

Follows a summary of the characteristics of the CACAO language:

- It can express multiple actions, thus yielding a sequence of actionable steps;
- Steps can be modeled by means of decision logic, either temporal or conditional, allowing to form structured playbooks;
- Steps are uniquely identifiable by means of unique identifiers;
- Playbooks support versioning and targeting, thus conferring a chronological history of changes applied;
- It allows the specification of the scope of the playbook, therefore providing a wide range of use cases;
- Provides support for auditing and reporting compliance requirements. So each action should provide information regarding the context that led to its execution;

- Supports security schemes such as digital signatures for signing and certification of a given Course of action and other security aspects such as transport integrity and authentication;
- A given Course of Action expressed in the CACAO language can be executed in a different operational environment.

The standard divides Playbook into two categories depending on the scope and actionability they aim to achieve:

- Executable Playbooks: there are meant to be executable from a recipient organization upon sharing, without needing further information integration or adaptation to the host organization's security operational environment. For a Playbook to be part of this category, it is required that the commands and actions used throughout it are standardized;
- Template Playbooks: on the other hand, Template Playbooks serve more as a documenting scope, that is, they provide a way to inform the host organization about the steps that have been taken or should be taken, and it can be up to the recipient organization to adapt the Playbook to their operational environment and security infrastructure to make it deployable, possibly after modifying and integrating it. For instance, it may be required to adapt some commands to the specific security controls of an organization. But it is also possible that Playbook actions are expressed in a high-level language that should be substituted with actual commands of the recipient organization before deploying it.

Moreover, the specification effort took into consideration different types of orchestration use cases in which security playbooks are most relevant and has come up with seven playbook categories:

- *Notification Playbook*: it is aimed at cyber threat intelligence sharing tasks and internal notifications. For example, what steps to follow to escalate security incidents to higher tiers.
- *Detection Playbook*: it describes actions that are taken as part of internal threat hunting procedures.
- *Investigation Playbook*: it embraces those procedures that are conducted as soon as an incident occurs or security-related issues are detected.
- *Prevention Playbook*: it describes actions and workflows that are meant to be taken to prevent incidents from happening and, in general, to increase the security posture.
- *Mitigation Playbook*: it is used to build workflows that are meant to mitigate the impact of security incidents, thus limiting the damage or blocking further damages.
- *Remediation Playbook*: it is used to build workflows that are meant to remediate security incidents. These are usually run after mitigation steps have already been taken. After their deployment, the affected systems and the infrastructure, in general, are restored to pre-incident status.
- *Attack Playbook*: this type of playbook represents penetration test procedures that are conducted in order to assess the security resilience of the infrastructure and validate the response strategies available.

A CACAO Security Playbook consists of a JSON file made up of various properties that are wrapped around an object of type "playbook" or "playbook-template". Atomic security operations, such as remediation, or prevention activities, are bundled in an object of type "workflow", in which each dictionary entry is a step object that represents an action. Steps are pieced together by means of unique identifiers and special step objects that enable various processing modes, such as conditional logic or parallel processing, together composing the Playbook workflow.

Knowledge to be used throughout the Playbook execution is maintained in memory through "variable" objects in a dedicated dictionary that can be present both at the Playbook level, and

in this case, the variables are accessible by all objects in the playbook or inside a step object, and in this case, variables can only be used in objects that are referenced by the object in which they were declared. In other words, variables can have different scopes, depending on where they are declared.

Mavroeidis et al. [14] have proposed a structured template that can be used to integrate security playbooks into structured and shareable threat intelligence. In their work, the CACAO language was taken as a reference, thus guaranteeing a perfect mapping between the CACAO language specification and their proposed template.
Their template proposal aims to provide interoperability between different security playbook standards by means of a unifying MISP playbook specification, thus supporting the integration of playbooks into knowledge management systems. Despite it being a recently designed language and lacking yet broad support in the cyber defense field, the CACAO Security Playbook standard has great potential to fill in the gaps that are present in STIX for what concerns the Course of Action object and is the best solution at the moment among the various playbook standard specifications as it manages to cover the most important use cases in the security response environment [15].

2.5.2 Security controls

In this part of the chapter a brief overview of the concept of security controls is given.

Introduction

To employ effective automated security practices, it is necessary to have tools available that are used to guarantee security parameters, and these must be able to be dynamically located in the operational environment in which one works. At this end, Security Controls come into play.

Security controls can be described as safeguards prescribed to protect organizations' assets, which can include physical property, computer systems, data, and so on. They can take the form of devices, policies, procedures, or any material action that can be used to mitigate, detect, or react to risks related to the assets.

Security controls can be classified by several parameters since they are diverse in their functionalities and in their protection context. For example, they can be specialized for enforcing security at the hardware level or at the software level, but are also involved in other aspects exist, such as the physical one, in which some security guarantees are required. An example of these can be surveillance cameras or locks, which in any case, are not a topic of this thesis work.

A common approach finalized at increasing the security posture leverages security controls by combining them, thus creating various layers of protection. With this strategy, even if one level fails, security constraints are not compromised, and for this reason, the strategy takes the name of defense-in-depth. An example of this approach has been treated in the NFV section of this thesis, with the discussion regarding Security Function Chaining.

Classification

Follows a high-level classification of security controls:

- *Physical Security Controls*: these categories of security controls enforce security in the real world. This category includes guaranteeing limited access to restricted locations, perimeter security, and other security measures related to non-cyber incidents, for instance, fire suppression systems, and so on.
- *Technical Security Controls*: this category refers to those security measures that can be taken at the software or hardware level. This includes all typical network security measures, such as firewalls, antivirus, but also encryption, strong authentication, and so on.

- *Administrative Security Controls*: this type of security control include policies, best practices, and procedure that are taken at the administrative level in an organization and often involve business practices more than technical ones. For example fall, under this category, two-factor authentication, the zero-trust approach, good password practices, and so on.

For the purpose of this thesis, only the second category is treated here. The following paragraphs present a specific type of security control, that is, firewall-like devices.

2.5.3 Firewalls

Firewalls are a fundamental network element for the management of network traffic and serve many purposes as security controls in software networks. Since the rise of computer networks as a means for interconnecting computer systems, the need for a way to control what flows in those links interconnecting network nodes has grown more and more important.
The term firewall nowadays is commonly used to refer to various types of network elements that can control network traffic. A common way to categorize them is by looking at which layer of the OSI stack they work. That is how deep in the datagram they can inspect.
From these criteria, they can be categorized into two types that are application-level firewalls and transport-level firewalls, depending on which level of the OSI stack they work at.

Application level firewall

It is also called level 7 firewall, and it can inspect network traffic up to the application layer. An instance of this firewall is web application firewalls, known as WAF. These are highly specialized types of application-level firewalls that are usually put in front of an exposed web application so as to control and filter the HTTP traffic flowing through it in both directions.
With this type of firewall, it is possible to filter traffic that may result in the exploitation of a known vulnerability on the web application, such as an SQL injection, rather than blocking all traffic on a certain port.
On the other hand, the high specialization requires a protocol-specific logic for each type of flow it has to handle.

Packet filtering firewall

It is also called level 4 firewall, and it can inspect network traffic up to the network layer of the OSI stack. Network-level firewalls make decisions based on fewer variables than application-level ones and can thus achieve much greater throughput. In fact, they discard or allow a packet based on source and destination port and IPs without being aware of the content within those packets. They work regardless of the protocols used by upper layers of the OSI stacks, without any protocol-specific logic implementation.
Regarding their capabilities, layer 4 firewalls can be used for several filtering scenarios since they are not limited to a single protocol.

Another distinction that can be made between firewalls is made according to their capability to store and maintain information about the flows passing through them. According to this characteristic, they can be divided into stateless and stateful ones.

Stateless firewalls

Stateless firewalls do not maintain the context of the streams it processes, so each new packet is treated singularly according to a predefined set of rules. These rules are matched against each packet, and depending on the result, the packet can be discarded or allowed. Discarded packets will never reach their intended destination, while the others will be let pass through.
The filtering decision is thus based on the packet headers, such as IP protocol source and destination addresses.

Filtering rules table							
N	Priority	Source IP address	Destination IP address	Source Port	Destination Port	Protocol	Action
1	1	10.1.10.1	34.100.100.1	*	22	TCP	DENY
2	3	10.1.10.3	43.0.4.5	*	443	UDP	DENY
3	6	10.1.10.1	8.8.8.8	*	22	TCP	ALLOW
4	5	10.1.10.2	1.1.1.1	*	22	TCP	ALLOW
5	6	8.8.8.8	10.1.10.1	*	22	TCP	ALLOW
6	7	1.1.1.1	10.1.10.2	*	22	TCP	ALLOW
...	...	...	...	...	...	...	...
default	inf	*	*	*	*	*	DENY

Table 2.1. Example of a filtering rules table

The main advantage of stateless firewalls is their unmatched performance when processing high throughput flows.
Stateless firewalls do not maintain the context of the streams it processes, so each new packet is treated.

Stateful firewalls

Stateful firewalls, on the other hand, do maintain the state of flowing traffic, so they keep track of all connections. For this reason, they are a powerful instrument for the detection of malicious traffic since they can have a form of intelligent decision system that increases in capability the more information is kept in memory.
This capability permits the monitoring and detection of traffic patterns and flows, not only the singular atomic element of the traffic as in stateless ones.

2.5.4 iptables

iptables is the standard packet filter shipped with Linux distributions, and its processing capabilities are native, integrated as modules into the kernel.
The filtering logic of iptables has been built around rules, and the decision logic has been divided into three distinct software abstraction layers.
These three layers are tables, chains, and rules. The first two layers are basically ways of grouping rules but on different criteria.
Tables are the highest level abstraction and are used to classify rules according to the type of action that will be triggered when a certain rule matches the packet information. For instance, the FILTER table groups all rules that, when matched, trigger an action of filtering type. This is the most commonly used table.
Chains instead group rules based on when they have to be evaluated, so for example, all rules that should be evaluated when the packet flows into a firewall are grouped into to INPUT chain.

Rules are instead simple statements used to program the behavior of the firewall and are the basis of its operation.
Each rule can be described as an element with two portions, the first is the matching one, which is the criteria that must be followed in order to evaluate whether to execute the target, with the target being the second portion of the rule. Targets represent actions to be taken, an example of them being whether to allow or discard a certain packet.
The matching portion of the rule, expressing the criteria, can take into consideration different parameters such as source and destination addresses, source and destination ports, protocols, interfaces, and since iptables has stateful capabilities, also how certain packets relate to others. For example, whether a packet belongs to the same connection.

Follows the list of most used built-in tables:

- *NAT*: it contains rules defining the routing behavior of the firewall when used to implement network address translation, so they manipulate the addresses at the IP level.
- *Filter*: it contains rules that are used to program the firewall in what concerns the blocking behavior, that is, when to let a packet pass and when not.
- *Mangle*: alike the NAT table is used to modify the header IP packets, though for different purposes, such as adjusting the time to live or making them processable by other network devices.

Follows the list of most used built-in chains:

- *PREROUTING*: it stores rules that must be evaluated before the packet routing has been determined. So as soon as the packet enters the local network interface. This is used for NAT operations.
- *INPUT*: it stores rules that must be evaluated when packet routing determines it is directed to the local network interface.
- *FORWARD*: it stores rules that must be evaluated when the packet routing determines that the packet is coming from another host with an outgoing network destination. This is used when the firewall operates on a host also operating as a router.
- *OUTPUT*: it stores rules that must be evaluated when packet routing determines the packet is originating from the local network interface with an outgoing network destination.
- *POSTROUTING*: it stores rules that must be evaluated after the packet routing has been determined before leaving the network interface.

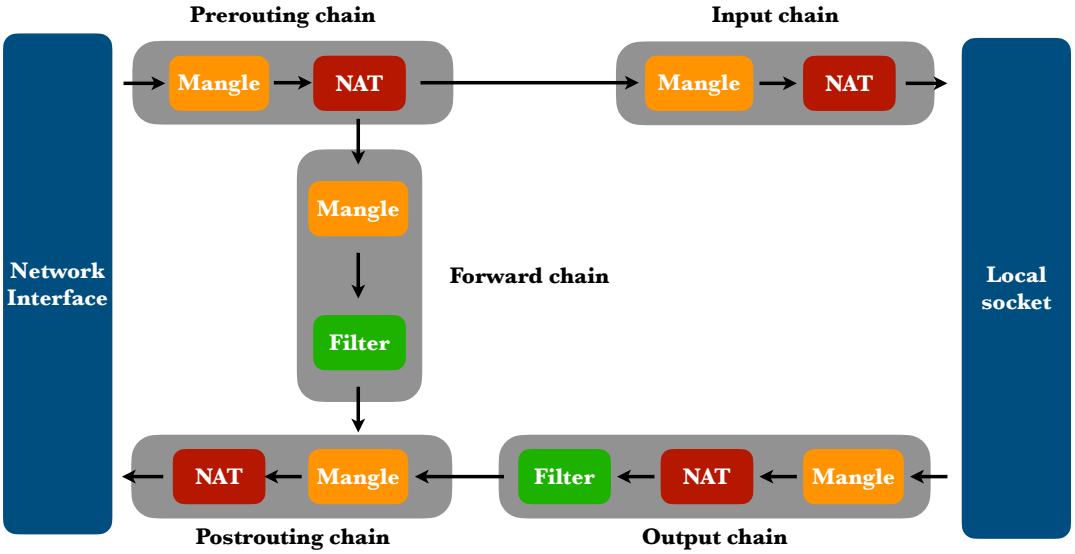


Figure 2.14. Flow of iptables rule evaluation.

For each chain, a default policy is defined, that is, a rule that matches in any case to a given packet. Its job is to be a fallback rule so as to avoid undefined behavior. This is useful when a packet's header doesn't match any other rule listed in the table.

As previously said, filtering capabilities are based on a different grouping of rules. These come with default behaviors, that is, different built-in tables and chains exist out of the box, but other

ones can be added by the user, thus guaranteeing certain extensibility.
Granular rules can be added to the firewall via human inputs, thanks to the user-space command-line utility, or via software, giving the possibility to couple filtering capabilities of iptables with the capabilities of other specialized programs, for example, those doing traffic analysis.

Command examples

Here a set of examples of iptables commands is shown:

Blocks all TCP traffic coming from the specific IP address.

```
iptables -A INPUT -p tcp -s xxx.xxx.xxx.xxx -j DROP
```

Blocks outgoing connections of protocol type equal to TCP with the specified destination IP and destionation port.

```
iptables -A OUTPUT -p tcp --dport xxx -d xxx.xxx.xxx.xxx -j DROP
```

Allows traffic incoming on TCP port 22 that corresponds to connections in the status ESTABILISHED or NEW. Corresponds to allowing SSH connections.

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW,ESTABLISHED
    -j ACCEPT
```

Set connection limits to prevent denial of service. In particular sets, the maximum number of connections per minute to 5 and specifies that this limit must be enforced only when the limit-burst connection number is reached.

```
iptables -A INPUT -p tcp --dport 22 -m limit --limit 5/minute --limit-burst
    50 -j ACCEPT
```

2.6 Threat landscape

In this section, a brief overview of the current threat landscape, with special consideration for those environments that follow the approaches previously discussed, is treated, specifically APT threats and Botnet-related activities. In the end, the threat modeling aspect is discussed.

2.6.1 Advanced Persistent Threats (APT)

Advanced Persistent Threats take the name from the behaviors seen in the wild in certain attacks, usually targeting corporations or state entities. This type of threat requires highly specialized capabilities since they usually use covert techniques so as to remain undetected for as long as possible, for instance, motivated by espionage needs. The attackers usually choose APT threats' targets carefully, and for each one, dedicated techniques, tools, vulnerabilities, and exploits are crafted. Thus it is also difficult to detect them since common prevention and detection techniques are usually fine-tuned for common low effort threats that target several victims at once, leveraging the numbers. For this reason, malicious actors behind these attacks can be traced back almost always to intelligence operations of rogue states or highly rewarding commercial espionage. The persistent nature of these threats means that the attacker minimizes his footprints in order to preserve his presence on the target infrastructure, and this is done by employing techniques such as pivoting and lateral movement. The objectives of an APT attack are usually achieved long after the exploit has occurred, following a brief overview of the steps describing the main steps involved in an APT-like event.

- *Gain access to the infrastructure*: the attacker manages to gain a foothold in the victim infrastructure, for example, by means of an infected email containing a malware, which then infects a host by exploiting a vulnerability. This step is preceded by a long preparation process that involves an accurate study of the victim infrastructure and the crafting of a dedicated attack technique;

- *Establish a control channel to hold control*: the main objective of the attacker at this stage is to establish a control channel and maintain it. The control channel is particularly delicate to handle since it could be easily detected if particular care is not taken. For this reason, a multitude of advanced techniques are used by the most resource full attackers that aim to masquerade the control channel as normal traffic;
- *Penetrate in-depth in the infrastructure*: this stage involves all the operations that can help with establishing control over more hosts in the affected network;
- *Objectives met, hold presence*: once the attacker has met his objectives, it usually preserves his presence if he deems it useful, otherwise, it will take care of erasing all traces of his presence.

2.6.2 Botnets

Botnets are networks composed of interconnected hosts infected by malware. This malware allows the malicious actor to take control of hosts and command them by means of a so-called command and control channel that communicates with a master host.
Botnets are characterized by a huge number of hosts infected, thus allowing the conduction of mass campaigns like DDoS. Various typologies have been seen in the wild, and nowadays, more resilient versions are common, in which the master host is decentralized, which means that the role is shared by many hosts, following a peer-to-peer network model.

A simplified phased representation of operations related to botnet activities follows:

- *Preparation*: the first phase involves devising a way to penetrate into many hosts so as to gain hold of as many hosts as possible. This is typically done by taking advantage of known vulnerabilities and targeting unpatched devices, for example, home devices such as an old router or smart home devices.
- *Contamination*: the attack is conducted by means of malware delivery methods, and once inside the victims' hosts, a C&C channel is opened. This phase is repeated indefinitely by infected hosts that try to infect other hosts.
- *Action*: once a sufficient number of hosts is reached, the bot master, that is, the malicious actor behind the botnet campaign, can finally leverage his bots to conduct the desired mass attacks. Mass campaigns often involve DDoS attacks or sending spam, but recently, crypto mining has also been seen. Once the objective is reached, the botnet can be dismantled or can continue to operate, potentially growing by continuously infecting other hosts.

2.6.3 Threat modeling

The current threat landscape shows an increase in the complexity and sophistication of new threats, often coupled with an intrinsic multi-stage behavior such as the one discussed previously for the APT threat typology. The multi-stage attribute is typically characterized by extended attack time frames, such that between the actual exploit and the eventual detection, the attacker has the possibility to conduct lateral movements, thus increasing potential damages.

At the same time, the stealthiness of these types of attacks implies that often the victim can act in defense only after the exploit. This implies that traditional preventive measures, typically passive, are no more sufficient to guarantee protection.

For this reason, proactive measures should be taken in the stages preceding the actual exploit, in contrast to the traditional approach of incident response.
Whereas the traditional approach is focused on internal security, such as vulnerability management, a better approach for dealing with these new threats should also focus on the adversary, including their TTP, therefore gaining a comprehensive view of the threat landscape that allows acting in advance, that is in the first stages of the attack.

Knowing the adversary thus is the pillar for a strong security posture, and shared intelligence is one of the key enablers of this approach, since one attack cannot suffice to build a high-quality picture of the threat with all its peculiarity. However, various iterations easily shared by third parties willing to cooperate definitely lead to it. Of course, the amount of information needed in this effort is of utter importance but certainly poses a challenge for security departments, thus motivating the need to embrace automation throughout security pipelines, which can support the handling of threat intelligence coming from different sources.

Kill chain

The previously discussed approach, however, requires models capable of representing those complex threats so as to come up with actionable knowledge. One of the leading models in this field is the kill chain, which was born in the military sphere and then adopted also in the cyber security sphere. This is nowadays used as a way to construct and structure an attacker's behavior in a chronological way. In the same way, the defensive posture is built around the concept of breaking it, since even without only one ring, that is, by stopping one of the kill chain stages, the attack fails. In the cyber security field, the first to propose it was Lockheed Martin with the Cyber Kill Chain. Their proposal is built around a kill chain consisting of seven stages that were specifically designed to create a representation model for APT style threats [16].

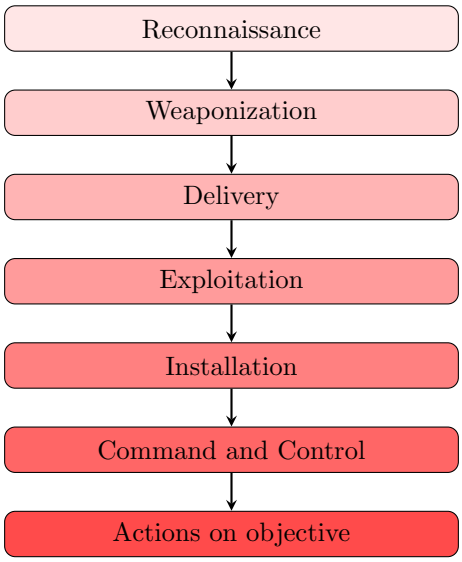


Figure 2.15. Cyber Kill Chain by Lockheed Martin

Nowadays various enhancement proposals have been made, and one of the most interesting is the Unified Kill Chain proposed by Paul Pauls [17]. In contrast to the Cyber Kill Chain, his proposal aims at covering the full spectrum of the possible stages of an attack, thus enabling the modeling of even the most sophisticated threats.

As far as usage goes, the STIX standard supports kill chain threat models since its first iteration, and in the latest versions allows to define a customized kill chain for usage inside the format. Various objects in the standard, in fact, can be labeled as belonging to a certain stage, in perfect compliance with further usage requirements. One example of how the kill chain can support enhanced cyber defensive posture is given by Wilkenss et al. [18] which leverage the Unified Kill Chain to construct a state machine based attack detection system, aimed at effectively reducing the number of alerts generated while delivering a consistent accuracy in the detection of APT alike threats.

Chapter 3

Problem statement

3.1 Introduction

In this chapter, it will be defined the subject of this thesis and what issues it aims to address.
In particular, a brief overview of the requirements that had to be fulfilled is illustrated in the next paragraph.
The second paragraph will detail the personal contribution toward fulfilling all requirements.
The third paragraph briefly describes the use case for the proposed solution, together with the proposed evaluation assessments and benchmarks.

3.2 Requirements

The scope of this thesis is to propose a new approach to incident handling that can embrace automation and intelligence sharing concepts that together aim at increasing the efficiency and efficacy of incident handling operations. This will take the form of a framework for the automation driven management of incident response.
In particular, the focus of this thesis will be on the incident response from the perspective of a virtualized network architecture. The threat detection and analysis stages of the incident handling life cycle, together with their peculiarities, will not be covered in this thesis but can be the subject of future works.

Follows a brief summary of the requirements put in place when developing this framework.

1. The first requirement corresponds to finding or developing a language that is expressive enough for the composition of security playbooks but, at the same time, allows for them to be shared across parties so as to enable the concept of shared actionable intelligence;

2. Another requirement is to be found in the support for shared intelligence. As previously discussed, the intelligence-sharing effort is one of the pillars of effective modern security solutions.

3. Intelligence sharing must provide the support for playbooks so as to enable a unified automated security posture across organizational boundaries;

4. Security operations, which are represented by security playbooks, must be able to be executed in an automated way, taking into consideration service availability and the operational status of systems;

5. Security operations, which are represented by security playbooks, must be able to be quickly deployed in response so as to pose an advantage compared to the traditional manual operated approach.

The language for security playbooks must meet the following requirements:

1. Abstraction: it should be possible to express security actions in various levels of abstraction. That is, it should be possible to have a statement expressing a security policy that is interpreted and enforced in the same way, independently of the security control that effectively enforces it at runtime;
2. Extensibility: it should be easy to expand it by adding the support for new capabilities that can also be related to new security controls;

3.3 Contribution overview

In this section, it is presented the personal contribution that this thesis aims to deliver. A new architecture for incident response management is proposed, leveraging previously discussed methodologies and innovations.
In particular, a proof of concept is built upon the proposed architecture in the form of a proof of concept framework. Later on in this section, a brief overview of the framework is given, together with its technological aspects, and in the end, its main strengths and limitation are discussed.

The main efforts in the development of this work are here listed:

- Standards evaluation: a significant part of the work on this thesis was related to the evaluation of different standards that are used or are being developed in the cybersecurity field, in particular, those regarding threat intelligence and incident response;
- Languages development: another part that required significant work involved the development of a new high-level language to be used for the description of security playbooks, aimed at incident response, together with an interpreter that bridges the language logic with the underlying network landscape. Moreover, another structured playbook format, CACAO, has been used throughout this work, and although it being standardized by the OASIS organization, it lacks a software implementation, so this was also developed as an interface for the high-level language developed in this thesis work;
- Security orchestration: piecing together all the developed components required to build a logic that would manage the entire security pipeline, from threat reports reception to the deployment of new security features;
- Remediation techniques: the last part revolved around searching in the literature for different incident response techniques and security strategies suitable in software network scenarios.

3.3.1 Framework overview

The framework is built modularly by various components. Of these, one is used to build a virtual representation of a software network topology, called landscape from here on, in which various security operations and policies are enforced. This helps in the optimization phase of the security deployment.
The network landscape is represented by a graph in which nodes represent virtual network devices or hosts, and the edges represent links between them.
The graph is constructed using the Python binding of the *igraph* library, which can deliver good computing performance while offering many features useful for network topology manipulation.

The framework can deploy different courses of action in response to a notification or new risk detected based on a series of parameters. These courses of action are expressed in the form of Security playbooks, that is, a list of procedures and policies to be executed, with some logic in it. For example, a given playbook is selected on the basis of the availability of certain security controls that are required to counteract a given threat. These checks and strategy optimizations

are done by means of a logic component, in which all known parameters regarding a given threat and other knowledge data come together. These data are then used in the selection of a suitable playbook and to deploy the selected playbook by adding context to it in the form of constraints and variables.

Security playbooks used throughout this framework have been designed to be represented in two formats, one of which is proposed for the first time in this thesis and is aimed at offering analysts and incident responders a way to express security procedures in a high-level way while at the same time remaining still readable by non-expert users. This allows to strip out unnecessary technical details regarding a particular deployment from its representation so as to keep the focus on the actual high-level pipeline expressed in it. Security playbooks expressed in this format have been named Recipes.

The other format chosen to represent security playbooks is the CACAO standard, which has been chosen in particular for the highly detailed specification that makes it a perfect match in the context of shared actionable intelligence. The two formats used have been made interoperable by means of an interpreter that can translate from one to the other. CACAO playbooks are easily embeddable into the final intelligence reports that are generated when an automated response operation is concluded, after the deployment of certain playbooks, thus conveying all information needed for their actionability into a STIX bundle.

Both playbooks formats express security features that are to be deployed on the landscape by means of security controls or other capabilities. In particular, it is possible to modify the network landscape layout, for example, in order to move a host to a quarantine area by means of a network landscape capability.

The STIX format has been chosen given its growing use in the cyber intelligence community and its support from major platforms in the threat intelligence sharing field, making it the defacto standard in the field. Besides, it allows bundling all information regarding how a particular threat has been handled in one single structure file. To this end, dedicated producers and consumers of these intelligence objects have been designed, together with various software components to represent and handle STIX Pattern expression, used as indicator patterns.

Here the proposed architecture's strengths are summarized:

1. Response speed: the removal of human intervention together with the direct ingestion of threat information can enable quicker threat response times, thus reducing the damage potential of security incidents;

2. Avoid human intervention: thanks to the holistic approach to incident response, the mitigation/remediation processes can be triggered as soon as a new threat is detected and a new alert is received without human intervention. This partially targets the qualified personnel shortage by relieving them of automated tasks;

3. Shared intelligence: third parties can leverage the collected artifacts to increase their security posture by updating their defensive systems. This puts them in an advanced position as far as attacks are concerned;

4. Actionable intelligence: all remediation steps that have been taken in response to an incident are documented in the final STIX report as a CACAO playbook, thus being machine-parsable. This allows third parties to directly deploy that course of action in case they also detect the same threat on their systems.

Proposed architecture limitations:

1. Scope: as previously stated, this proof of concept is currently limited ad handling network-related operations of incident response, so it can manage security issues that originate from a different architectural level;

2. Remediation techniques: despite having previously stated that the automation effort should start by automating those actions that do not require human intervention because they are low priority and as such would have limited to zero impact on service operability, it is still true that more sophisticated playbooks can be implemented. In those cases, a human can be supervising the automated processes so as to have the required guarantee regarding service operability;

3. Statefulness: during the development of the proof of concept framework, some choices had to be made regarding the complexity of the remediation strategies. One of these concerns the ability of the system to react in certain ways based on accumulated knowledge about the operational environment and on previously seen threats behaviors.
   This would considerably enhance the framework's capabilities by conferring a basic level of intelligence. Currently, these types of capabilities have been left out for future expansions.

4. Extensibility: connectors able to orchestrate common SDN controllers are missing. So one would have to use a custom-made API to interface with those. Since the nature of the proof of concept, though, this doesn't limit the potentiality of the framework.

3.4 Use case definition

The proposed architecture aims at providing a security framework that leverages a threat intelligence pipeline, both automated and human-driven, to automate common security operations by means of security playbooks. In particular, incident response is the main use case of this framework. Through this architecture, common repetitive manual operations regarding incident prevention, mitigation, and remediation are easily automated without a human in the loop.
As far as threat intelligence is concerned, it shows how integrating shared CTI into an automated incident response pipeline can drastically decrease the time between the reception of new indicators and the enforcement of new security policies.
It is possible to envisage different situations in which this framework would enhance security operations. In fact, it has been designed to support different types of mitigation/remediation strategies, as has been previously said, with a focus on network-related ones.
This allows testing of particular scenarios of network intrusions, together with the resulting incident response, each one with its own peculiarities.
Thus for the testing phase, a network topology representing a medium-sized company infrastructure has been designed.

Various types of threats were envisaged as a testing bench for this proof of concept. Out of the many, a botnet scenario has been deemed the most appropriate to evaluate in order to show off how the remediation phase is carried out while optimizing the resource usage, for example by minimizing the placement of certain virtual network devices, without any human oversight, thus aiming at impact reduction on service operability.
Given this choice, the supported security playbooks that have been thought of as possible remediation strategies to be applied to the operational environment mirror this scenario.

This scenario is considered by proposing a set of prevalidated playbooks, each targeted at a different stage of a botnet threat scenario and its corresponding security tasks that are to be performed to handle it. Since the threat functioning is guaranteed by persistent access to the master, cutting off this channel of communication is one of the proposed automated remediation techniques, together with monitoring ones. These tasks are performed by means of a firewall security control, which enforces a set of blocking rules minimizing the impact on the network infrastructure.

Chapter 4

Design overview

This chapter is organized as follows. In the first place, a high-level perspective of the proposed framework architecture is presented, making use of a component subdivision. After that, an overview of each component, be it a software module or abstract component of the framework is covered.
This categorization will not necessarily follow a structural criteria at the software level, meaning that components can also exist in the form of methods declared in a multi-scoped class, so at a different abstraction level. So various implementations of the proposed architecture may exist, depending on the various software design patterns used.
In the remaining paragraphs of this chapter, the workflow is covered. That is how those components that are covered in the rest of the chapter work together.

4.1 High level design

The proposed architecture is composed of various components, which can be viewed as belonging to different layers, each one grouping a certain functionality. A reference schema of it is shown in figure 4.1. The *decision logic* component is an abstract orchestrator layer that works as an interface for other components and coordinates the logic behind a particular response. In order to operate, it receives inputs from repositories, threat intelligence consumers, and the control layer. It thus coordinates the whole incident handling life cycle, in an automated way, by selecting a particular course of action in the form of a Playbook, that is taken from the available ones.

Another grouping is represented by the *persistence layer*, which is composed of three repositories. Some of those repositories are interconnected since some entities in them reference each other. Apart from that, they are directly connected not only to the decision logic, which leverages them in the decision phase, but also to the threat intelligence consumer since not all threat intelligence notifications require a new response by the decision logic.

The *threat intelligence layer* deals with the management of new intelligence information and notifications. In particular, it is composed of two components, one producer and one consumer. The consumer receives inputs from the outside, and after a parsing phase, it delivers them to the decision logic or directly to the dedicated repository.

Another component is the *Playbook interpreter*, which at an abstract level, sits in between the decision logic and the control layer, and its task is of deploying the selected Playbook that it has received by means of the control layer. This component deeply interfaces with the network landscape by means of capability modules that permit the enforcement of security features on the landscape.

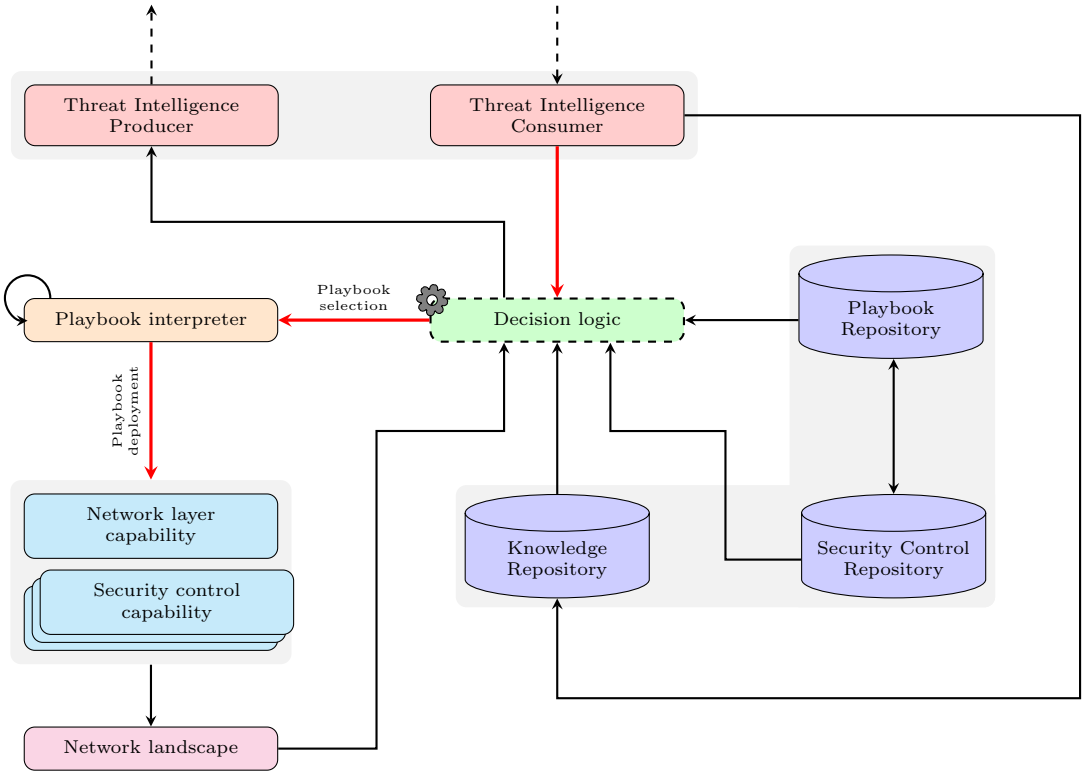


Figure 4.1. High level architecture overview

4.2 Persistence Layer

Repositories contain data that is used throughout the architecture security pipeline.
The role of this data goes from details regarding prioritization of incident response strategies, through data regarding known threats and their related suggested courses of action, to security controls parameters that are used to specify how to enforce certain security policies.
Repositories can hold data persistently, meaning that loaded data are kept between multiple playbook deployments and are deleted only in those circumstances in which the data comes with an expiry date, as is the case with certain indicators of compromise.
Adding new data to repositories can happen in two different ways. One is asynchronous, and the other synchronous. This distinction is justified by the typology of the information source.
Potential sources of data for a repository can be represented, for example, by the manual integration of threat intelligence into the platform that is made by security analysts, in this case of asynchronous type, or by threat intelligence data that is automatically ingested by means of threat feeds' subscriptions, in this case of synchronous type.
The structure of these repositories and their technical details will be treated later on in the chapter 5, together with a comprehensive view of their usage inside the framework.

4.2.1 Knowledge Repository

The threat intelligence repository is where the knowledge related to threats, attacks, and other ancillary information, such as their behavior, potentially following a TTP standard, is stored. Threat knowledge can be coming from internal threat hunting activities, but also, and especially, from cyber threat intelligence sharing. In the latter case, the repository is continuously filled up by means of threat feeds. The knowledge contained in the repository can be structured in ways that make it easy for the decision logic to infer behaviors and response strategies. For example, if knowledge about a threat follows a kill chain pattern, the response logic can select a more specific response, thanks to better situational awareness. Information for each threat regards, for

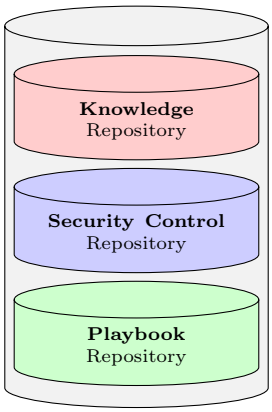


Figure 4.2. Repository architecture

example, the assessed impact that it would have on the organizations' infrastructures, but also the recommended response that should be given in the eventuality it is sighted on the network. These are not redundant information since Playbooks are an approved set of procedures that are audited to be compliant with the organization's security practices and apparatus. On the other hand, threat intelligence reports retained in the Knowledge Repository may contain Playbooks related to the threat. However, these cannot be directly applied to the recipient organization because they should foremost undergo various verification processes with which they are adapted to the organization's apparatus. At that point, they are added to the Playbook Repository.

4.2.2 Playbook Repository

The Playbook Repository stores all the available Security Playbooks that can be deployed in response to various events. Playbooks can be stored in different formats, that is, as CACAO Playbooks or Recipes. In any case, each one of them comes with a set of requirements for its deployment, one of them being the list of security capabilities required.

4.2.3 Security Control Repository

The Security Control Repository is used to store for each security control available its configuration requirements together with the corresponding security capabilities that it can enforce.
This repository is not just a data store. In fact, it also stores a reference to the various security control functions that are used directly in the deployment stage to enforce a given security feature.

4.3 Threat Intelligence Layer

This layer is composed of two threat intelligence workers, a consumer and a producer. Here these two components will be treated.

4.3.1 Threat Intelligence Consumer

This component ingests various forms of cyber intelligence, such as those received from internal threat hunting tasks or from soc analysts during incident investigations. Apart from this, the repository can also be integrated asynchronously whenever it is deemed appropriate, for example, when new evidence is gathered in regard to a specific behavior seen on the network.
As part of a reactive framework, the intelligence consumer is able to treat real-time threat alerts, as can be, for example, those originating from a SIEM system. Moreover, another source of intelligence is that of threat feeds. In particular, thanks to the support of the STIX format, it

can handle STIX reports in the form of STIX bundles. These reports contain various indicators of compromise that are directly added to the Knowledge Repository. More complex STIX representations, such as those containing the behavior of adversaries throughout certain threats campaigns, must be first agreed upon in order for their representation to be supported by the Knowledge Repository. Regardless of the threat intelligence typology, in fact, all information undergoes a normalization process at the end of which actionable knowledge is converted to a common internal format. In figure 4.3, it is depicted a simple schema representing the workflow of the threat intelligence producer.

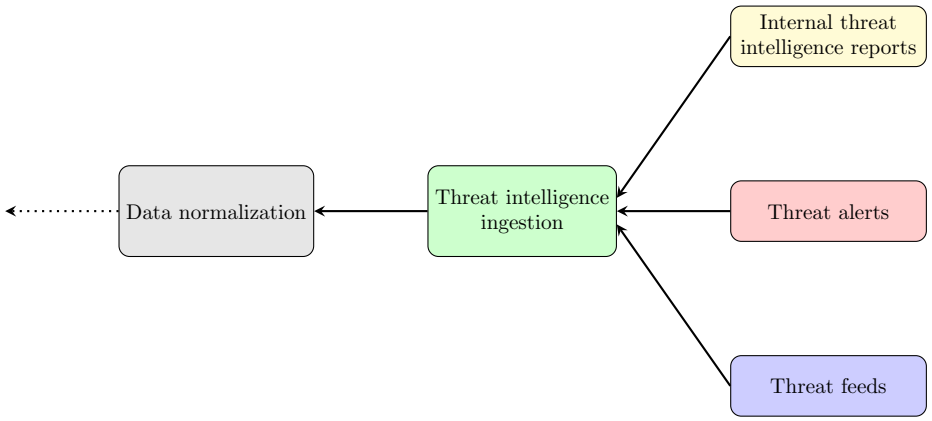


Figure 4.3. Threat intelligence consumption schema

4.3.2 Threat Intelligence Producer

As opposed to the threat intelligence consumer, this component deals with creating a final report containing actionable intelligence regarding the incident, risk, or threat that was remediated, mitigated, or handled with. Those reports can be internally scoped since they will be useful in the future to handle a similar situation or can be shared with partner organizations or ISACs. In this case, particular care should be posed in the selection of which information to share.
The final report includes, for example, the Playbook, which was deployed in the deployment stage, together with all the required parameters. In figure 4.4, it is depicted a simple schema representing the workflow of the threat intelligence producer.

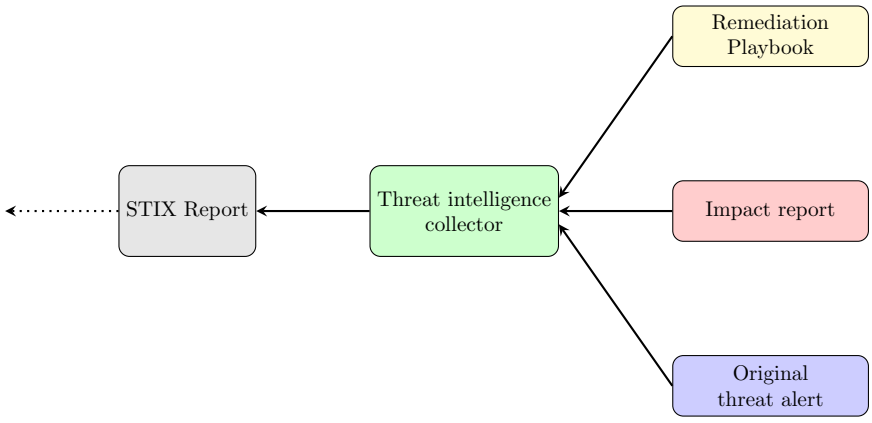


Figure 4.4. Threat intelligence producer schema

4.4 Network Landscape

As previously said, the network landscape is used to represent the network topology with a graph model in which nodes represent virtual devices while edges represent the interconnections between them. Some of the devices that can be represented in the landscape are, for example, switches and other networking equipment. A graphical representation of this graph can be obtained thanks to the export functionality of the igraph library. Moreover, it can export the graph model into various data formats, thus enabling further external analysis functionalities.

The network landscape corresponds to the service graph component that is used within NFV orchestration architectures. In software networks, functionalities are provided via dedicated virtual network functions instead of hardware devices as in traditional networks. The service graph, therefore, is used to model the deployed VNFs topology that sits on the underlying physical network substrate. For this reason, various techniques exist to map the service graph to the network topologies, and in this step, various optimizations are done in order to allocate resources better. It is, though, outside the scope of this thesis to define these methods, and the interactions between the service graph and the underlying network are simplified by means of the Control Layer abstraction.

4.5 Control Layer

The control layer is composed of the various security capabilities that are available. Capabilities represent a tool, security control, or human operator that can be used to perform a security related operation. Capabilities include, for example, security controls, which are used to enforce a security feature on the underlying network, for instance, adding a new filtering policy to a network forwarding path or protecting a node with an IDS. But they can also include, if present, an NVF MANO orchestrator with which it is possible to orchestrate VNFs deployment and management operations. In general, therefore, security capabilities are used to interface the automated response system with the operational environment so as to apply changes to it or gather further information from it.

4.5.1 Network Layer Capability

This component is highly dependent on the underlying network implementation, and in particular, on the SDN controller that is used. Nowadays, the OpenFlow protocol is heavily used in the controller mechanism of software networks; however, for the sake of simplicity, and since it would not add value to the framework PoC at the current prototype stage in this architecture, it works as a simple interface to the network landscape, meaning that topology changes are directly applied to the landscape without being pushed on the actual underlying network layer.

4.5.2 Security Controls Capabilities

These capabilities are used to enforce security functionalities. Each capability comes with a policy translator function that is used to configure the security control according to the high-level rule that is expressed by means of high-level actions in playbooks. When a playbook is deployed, each high-level action represents a policy that must be translated into the corresponding low-level policy in the language specific to the security control that is assigned to enforce that type of policy. The mapping between a certain security capability, intended as a security feature, and a given security control is performed by the Decision Logic before playbooks are run. It is, in fact, done prior to the effective deployment of the playbook, and it is part of the pre deployment operations that will be summarized later on below.

As far as security controls are concerned, they can be viewed as two components, one is the input receiver, which deals with the reception of new configuration requests coming from the playbook instruction, and the other is the actual actuator, that act on the underlying network

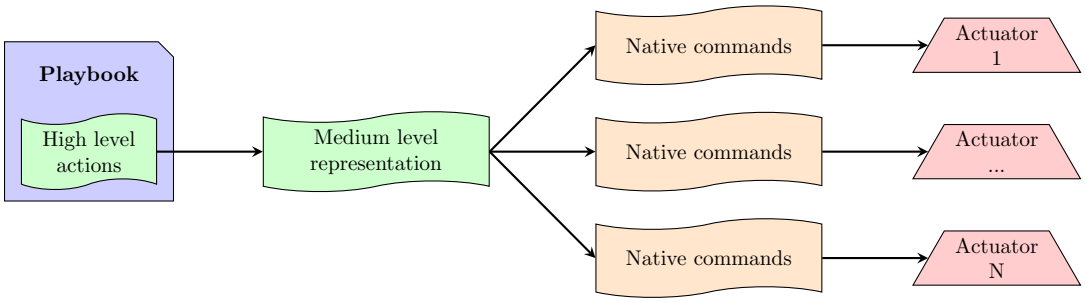


Figure 4.5. From high level security actions to low level configurations

layer, and consequently may lead to network landscape modification. As has been said before, a mapping mechanism between capability and the actual security control enforcing it is required. One of the reasons for it is that multiple security controls may be available to enforce a security feature.

Even if multiple security controls are available, each one requiring a customized configuration and input commands, the high-level language does not allow different action commands for the same security feature. For this reason, a medium-level representation of playbook actions is used. This abstracts the high-level action from the actual configuration to give as input to the actuator of the security control. In order to do this, a mapping function is required for each security feature, thus providing a common representation for each action related to a given security feature. Once this medium-level representation is generated, a policy translation function will be called to generate the actual low-level configuration that will be passed to the security control as input, and by turn, it will apply the new configuration to the actuator. This policy translation function is thus customized for each different security control 4.5.

Once all policies are translated at the end of a playbook deployment, they are passed to the network landscape as a set of commands or policies depending on security control input mechanisms. If the security control is represented by a node in the network landscape, then the command will be appended to the node's set of enforced rules. In a production implementation, these commands and translated policies are meant to be directly passed to security control connectors, SDN controllers, and NFV MANO orchestrators.

4.6 Core Layer

This abstract layer comprehends various logical layers that work together to carry the operational stages, eventually leading to the deployment of security features onto the operational environment. This part represents the automation engine of the framework.

4.6.1 Decision Logic

The decision logic represents all those tasks carried out in the early stage of security deployment, for example, the Playbook selection logic. The Playbook selection process is executed by first gathering all relevant information regarding the new threat or risk notification.

This information is then used to search the Playbook Repository for a compatible and suitable Playbook. This first step yields a set of Playbooks, and for each one, an Enforceability check is conducted, eventually leading to the selection of the Playbook to be interpreted.

Once the Playbook to be deployed is selected, it undergoes an enrichment process, in which the required deployment parameters are loaded into the Playbook interpreter dedicated memory storage or directly embedded into the Playbook data structure if deemed more appropriate.

4.6.2 Playbook Interpreter

As previously said, security playbooks are expressed in a high-level language as Recipe or alternatively in the equivalent CACAO format. For their deployment, an interpreter is required. Given the diverse nature of the two formats, the two languages require two separated interpreters, even though, given the interoperability between them, they share some of the logic. The two interpreters have been built to be lightweight, this is motivated by the fact that the languages to be interpreted do not present a complex grammar, nor its usage requires a multitude of constructs. In fact, the languages basically support Turing completeness, that is, conditional logic, and do not go further.

The Recipe language uses a natural language grammar but with fixed constructs, thus limiting the language complexity. Solutions such as ANTLR were discarded since many of their features would not have been used and would have added unnecessary complexity to the framework. Therefore, as far as Recipe language is concerned, it has been chosen a design in which the lexing and parsing parts are integrated into one component that scans each line, and after having tokenized it, it passes it to the respective function, together with the deployment parameters. In contrast, the CACAO format does not require a lexing process since the language uses a highly structured format to convey workflows and actions.

4.7 Workflow

This section covers the workflow of two of the main stages of the framework architecture. These stages concern the security pipeline of a security Playbook, from the preparation phase to the deployment phase.
An automated security playbook pipeline starts automatically whenever notification of a new threat or new risk is received. Moreover, it can be triggered manually by security operators to carry out a security task in an automated or semi-automated way by selecting a security playbook manually to be deployed. In this case, the preparation stage involves only the enforceability check and the enrichment process since the selection task is done manually by the analyst who has already evaluated the available playbooks and chooses what it deems better for the task to implement.

4.7.1 Playbook preparation stage

The stage is depicted in figure 4.6, and its function is early on in the playbook pipeline, and its role is of readying the Playbook for its execution by the Playbook interpreter. Moreover, if the start of the security playbook pipeline was triggered by the reception of a threat report, that is, in response to a new notification, it will also include the selection phase. This is needed since the process is automated based on the available information in the Knowledge repository, so the best suitable Playbook to deploy as a response must be chosen from a set of playbooks.

The selection stage searches in the Playbook repository for all the Playbooks that are compatible with the response to the specific type of notification. For this operation, various details of the received report are analyzed, and for this reason, the format should follow a consistent structure. The Knowledge Repository is queried in this process since it stores all relevant information regarding past incident response activities and updated threat intelligence that is received by means of threat feeds. Moreover, the Knowledge repository stores a set of triggers, or associations, between threat classes and playbooks, or alternatively between observable data and playbooks. These associations are later used in the decision logic to better identify response strategies.

Once all compatible Playbooks are found, another step is conducted that leads to the selection of only one Playbook, which is the one that will be deployed in the next stage. This step consists of a series of checks that are made in order to decide if a playbook can be run with the available security capabilities and if the available information is sufficient to run that playbook.

The last step that is part of this stage involves the enrichment of the selected Playbook. With this process, the set of relevant parameters needed for the Playbook deployment is loaded into the

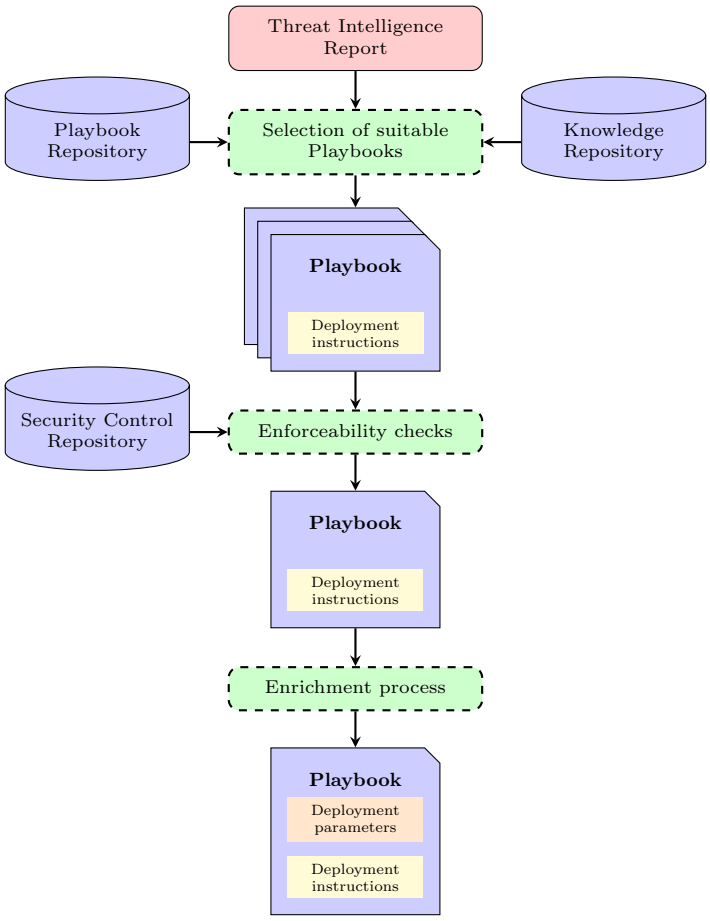


Figure 4.6. Playbook preparation stage

interpreter memory store dedicated to handling this data, or as previously said, directly embedded into it, depending on the Playbook format.

At the end of these preparation steps, the Playbook is ready to be run, the role of which is appointed to the Playbook interpreter.

4.7.2 Playbook deployment stage

Once the playbook has been enriched with the required data, it is passed to the playbook interpreter for its deployment. The deployment of a Playbook is performed with the sequential execution of each instruction of the Playbook, except for those steps in which conditional logic or iterations are present. The next chapter will delve into the details of how the interpreter works. A schema representing this stage is depicted in figure 4.7.

Each instruction can express low-level operations, such as the request for a security feature or an operation that yields a result that is useful during the course of the playbook's execution. These instructions may therefore query some characteristic of the network landscape or reference knowledge data to better model the response strategy. Moreover, they can express high-level operations such as those that are meant to be performed manually by a human operator. In certain situations, a human operator input may be required to perform some actions or to give the final authorization for the playbook to be run, especially in case the operations carried out may disrupt services.

Each low-level instruction may involve a Capability for its functioning. For example, those instructions that query the network topology require the presence of a network landscape capability in the framework. On the other hand, an instruction that expresses a restrictive rule, such

as a new filtering rule on a certain payload, requires the presence of a security control capability that is of the same filtering type as the rule that is to be enforced. The framework uses an NFV like approach, therefore, the effective enforcement of the rule may be equal to adding a new rule to an existing filtering device, or adding a new virtual device directly in a suitable position, so as to filter the desired flows.

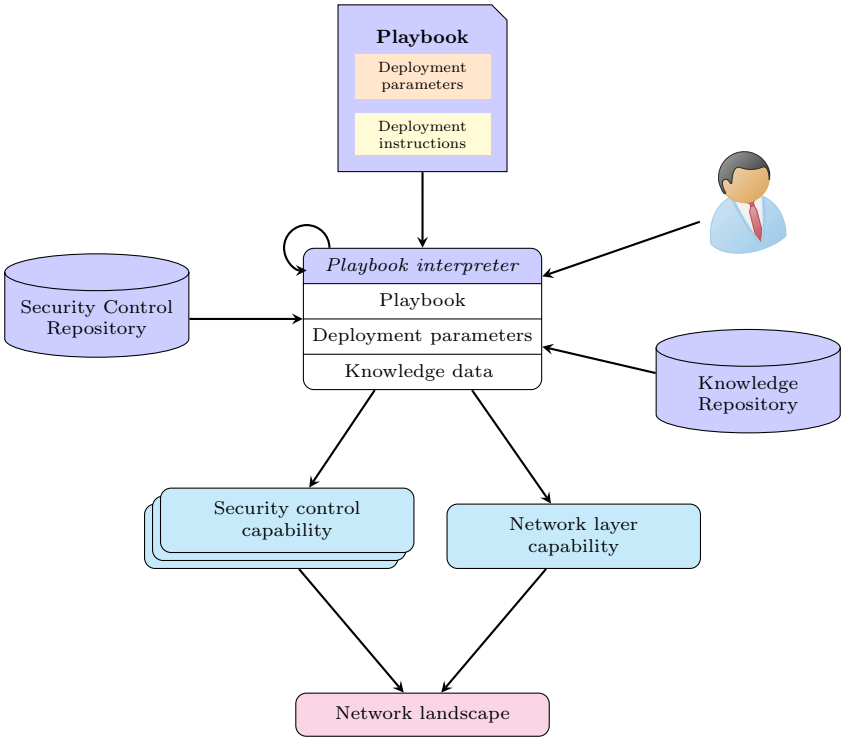


Figure 4.7. Playbook deployment stage

Chapter 5

Implementation

In this chapter, the details regarding the proof of concept framework developed will be covered. The framework architecture is based on the one discussed in the previous chapter and will be focused on the automated response to incidents by means of mitigation or remediation strategies packaged into Security Playbooks.

5.1 Introduction

A set of Playbooks is presented that synthesize response strategies related to network security policies enforcement to be used in various threat or incident events. Moreover, it will be shown a concrete instance of incident response that leverages this framework. Precisely it will be shown the workflow that is implemented by the framework, from threat alert ingestion to the deployment of a remediation through a Playbook, as a reaction to the detection of a Command and Control channel on a host belonging to the network infrastructure, that is attributed to an adversary conducting a Botnet campaign.

For the technical details regarding the framework and its usage, in the Appendix A the user manual can be found, while the developer manual can be found in Appendix B.

The chapter is organized as follows. In the first section, a set of Playbooks ready to be deployed for handling low level network related incident response activities is shown. They will be shown both in the Recipe format and in the CACAO format, and for each one, their details will be treated. In the second section, some more details about the Repository usage are discussed with the Network Landscape abstraction. In the third section, it is briefly presented how the Network Landscape abstraction implementation together with its graph representation. In the fourth section, it is discussed the implementation of the Threat Intelligence consumers and producers leveraging on the STIX format. Finally, in the last two sections respectively, a more in-depth view of the Security Capabilities component is presented, and how the various processes are handled via the Decision Logic abstraction is discussed.

5.2 Playbooks

In this section, it is reported a comprehensive view of the Playbook formats used throughout this thesis work.

5.2.1 Introduction

The Playbooks that are shown in this section are meant to orchestrate various network related operations that may be executed in a heterogeneous network environment, in which various devices are used, possibly from different vendors, running on different software stacks, for example, in

order to adopt an in-depth security strategy. Since these actions can modify those systems' behaviors or the network topology in which they lay, an interface layer that abstracts their proprietary controller interfaces is needed. In production systems that implement SDN and NFV architecture paradigms, this is done assuming the presence of SDN controllers and MANO endpoints, but either their implementation or their usage is out of the scope of this thesis. As previously said, in this proof of concept, Playbook actions will be tailored for documenting actions in the form of procedures related to the mitigation and remediation of network related security issues. While manual input is possible to be expressed, it is out of the scope of this thesis to define Playbooks that include human supervising.

5.2.2 Recipe Playbook format

The Recipe format for representing security Playbooks is based on a domain-specific language adopting a semantic scheme in which statements express actions to be run by means of a keyword that specifies the statement context. Based on the statement context, other parameters can be passed to the statement, with some being optional and others mandatory. The syntax used in this domain-specific language is minimal because it wants to offer a way to express playbooks that is as close as possible to a natural language without losing consistency and while still being unambiguous. This is a valuable feature in order to provide an easily readable and fast mechanism to document incident response measures that have been taken, possibly accelerating meetings used to brief higher departments with high-level views on the measures undertaken.

Parameters of a recipe, that is, variables referenceable inside it, can be specified inside the playbook using an assignment operator, called runtime parameters, or alternatively in a dedicated memory store that is declared before running the playbook, that can be referenced everywhere in the playbook, called global parameters. The latter is akin to global variables in common programming languages, and their memory store is thus called global scope. Of course, parameters can be modified during the execution of a playbook, even the global ones, and in this case, the memory store values are overwritten.

Actions, that is statements, which yield values, implicitly define new variables. The names of those new variables are dependent on the action, but the action name and scope are hints to the name of the yielded variables. For example, an action that is used to find out if a certain network device is present in a network path will yield a variable named "found", of the Boolean type. Variables do not have a type, but they can be inferred from the name and possibly from the context to which they belong. Each action has a corresponding function that is where the computations are done by interfacing with certain capabilities. The previous example regarding a function used to gather information regarding the network topology interfaces with the network layer by means of the network landscape. As far as extensibility goes, new actions can be declared and used inside Recipe playbooks, with the only requirement being the definition of corresponding functions that implement the needed behavior.

Moreover, basic construct statements used to implement branching logic are provided, in particular for dealing with iteration logic and conditional logic. These are called flow statements and are particular instances of action statements. The iteration construct adopts the functional paradigm, thus, it is implemented as a "for each" construct rather than the typical C style "for" construct. Flow statements are used to set up a new scope, in which a new branch of execution is launched, starting from the first statement of the block defined "inside" the flow scope. This procedure is similar to the traditional function call in ordinary programming languages. Depending on the construct, iteration, or conditional logic, the flow of execution will process the given sequence of instructions multiple times, in the case of iteration constructs, or only one workflow branch of the two, in the case of conditional logic construct. Flow constructs, like actions, can also yield variables, in fact, this is the method with which the flow logic is implemented. In practice, flow instructions yield variables that can be referenced only inside their scope, similarly to how functions' arguments can be referenced only inside their scope in ordinary programming languages. Emitted variables in flow logic constructs are used, for example, in the iteration construct, where at each cycle, a variable named "iteration_element" is redefined, and in it, the current iteration element is stored. This can be referenced by instructions contained in the scope

of the iteration block and also in any other nested scope that can be created by nesting flow statements.

Recipe interpreter

Nesting multiple blocks is in fact possible, and it is technically implemented via the recursive instantiation of new interpreter instances, very much like function stacks in programming languages. Once the inner block execution is completed, the interpreter "stack" unrolls, returning control to the upper-level interpreter instance. In this process, each interpreter has access to its own reserved memory store, in which data related to variables is maintained, together with other data functional to the interpreter operativity. The memory store contains, for example, a reference to the upper-level interpreter instance if there is one, and this is used to access upper-level scope variables, thus, memory is not cloned across instances. Moreover, the first instance, called global instance, is the one whose memory store contains data that has been gathered from the Knowledge Repository in the preparation stage, which precedes the deployment stage of Playbooks.

Recipes are run by means of a dedicated interpreter. The interpreter sweeps through the Recipe textual representation, and each statement invokes a call to the corresponding function or language construct, but before this, the statement is tokenized so as to obtain a structured representation of the instruction that comprehends string literals and variable references. Once this structure is filled with all constraints, the corresponding low-level function that contains the actual implementation is invoked. The functions leverage Security Capabilities that have been implemented to execute the required operation. An indicative example is a statement like "add_firewall" that, when interpreted, calls its corresponding function that, in turn, will update the Network Landscape according to the requirements expressed in the statement, that is, adds a firewall node to it in the specified position.

As far as human supervising goes, when an input is requested, be it for final authorization to deploy the Playbook in case it has a high impact on service operations or for integration of the knowledge used throughout the Playbook execution, the interpreter is paused until the input is received. Eventually, the input is persisted in the memory store of the global interpreter instance, which is the first instance that is spawned at the start of the Playbook parsing. The input is thus accessible through a global variable called "input".

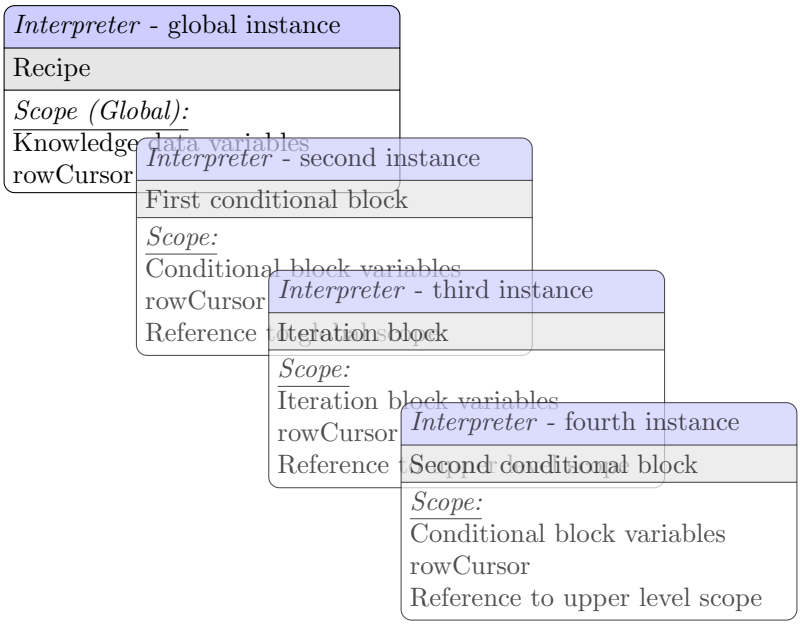


Figure 5.1. Playbook interpreter call stack

Recipe language overview

As said before, the Recipe expression language aims at providing a way to write security playbooks that can abstract the underlying technical implementation of a certain security operation, thus increasing the overall readability while being unambiguous. Atomic actions are expressed by means of instructions, that is, the statements of the Recipe, and the semantic is inspired from works related to high-level policy languages for network related concepts [19] [12].

A Recipe instruction semantic structure is defined as follows:

[**Action identifier**] [(arg name, arg value)...(arg name, arg value)]

In which:

- Action identifier: this keyword is an unambiguous name assigned to the particular operations and is positioned at the beginning of every instruction of a Recipe.
- Argument name: it represents a keyword or multiple ones separated by spaces that are used to enrich the statement context with additional details.
- Argument value: it instead represents the actual value of the argument or a variable reference. The value can be embedded inline only if it is of the string type and is used mainly to specify capabilities. Instead, for more structured values, they should be declared outside of the Recipe, directly into the memory store, or inside the Knowledge Repository. This is done to avoid jamming the Playbook lightweight syntax with heavy declaration blocks.

Arguments in statements are action-scoped, meaning that depending on the action to which they are added, they assume different meanings and are used to specify different action-related constraints, much like words can have different meanings in natural languages depending on the context in which they are used. It is up to the organization or team that makes use of Recipe playbooks to define a grammar that is consistent across context-related Recipes. At the same time, it is always good to follow some general canons, and in this regard, some illustrative examples will be shown below.

Recipe instructions act as an interface between the security operator who designs the procedure and the real low-level security configuration to be applied to the infrastructure object of the security treatment. Nevertheless, those instructions are not directly applicable to security controls that since the last ones need proprietary languages and configurations. For this reason, each instruction leverages an implementation function in which a translation process happens by means of the available Security Capabilities. Each high-level statement is eventually converted to the specific Security Control low-level configuration language that is at the end transmitted to the node in which the rule is poised to be applied to enforce the security feature.

Next, in this section, some examples of Recipe instructions are shown, and after that, some prevalidated Recipe playbooks representing remediation steps relevant to the use case that will be treated later are shown. In all cases, the syntax is highlighted in different colors to show the different semantic elements, in particular:

- Black: syntax belonging to basic language constructs;
- Brown: all variable names that are loaded in the memory store before the Recipe is run;
- Blue: all actions keywords;
- Orange: all variables that are defined at run time, be it for flow constructs or those that are yielded by other actions.

Recipe expressions examples

The following expression synthesizes the action of searching into a network path, consisting of various nodes, to ascertain whether a node of a specific typology and with the required capabilities exist, with these last details declared as arguments in the instruction. The statement is also self-explanatory.

```
find_node of type 'firewall' in network_path with 'level_7_filtering'
```

This first expression grammar consists of the following tokens, reported in order of appearance:

- find_node: this is the keyword that identifies the action associated to the statement;
- of type: is the first argument specifier, and is used to declare which typology of network node it is to be searched for;
- firewall: is a string representing "of type" argument value;
- in: is the second argument specifier, and is used to reference the variable containing the network path in which the search operation must be performed;
- network_path: is the name of the variable related to the "in" argument specifier. The value of the variable referenced is a list of node names, or IP addresses, that should be the addresses of the nodes representing the path;
- with: is the third argument specifier, and is used to specify the capability that the node must have;
- level_7_filtering: is a string representing the "with" argument value.

The following expression synthesizes the action of adding a new firewall device in the specified network path and position inside it. It can be noted as the "in" argument specifier here performs the same function as in the previous action, that is, it is used to indicate the network path on which the action acts. When the interpreter executes the action, the corresponding implementation function is invoked. The implementation function, or connector function, will interface with the network landscape and apply changes to the network topology, following a programmatic procedure involving graph operations that are reported in the algorithm 1 as pseudo-code. As said before, of course, in a real world scenario, a MANO will also be involved in the operation, so as to directly apply changes to the underlying network topology.

```
add_firewall behind impacted_host_ip in network_path with 'level_7_filtering'
```

This second expression grammar consists of the following tokens, reported in order of appearance:

- add_firewall: this is the keyword that identifies the action associated to the statement;
- behind: is the first argument specifier, and is used to specify the position in which the new device is to be put along the specified network path;
- impacted_host_ip: is the name of the variable related to the "behind" argument specifier. The value of the variable referenced is an identifier, or IP address, of the node behind which the new firewall node is to placed;
- in: is the second argument specifier, and is used to reference the variable containing the network path in which the new firewall device is to be added;
- network_path: is the name of the variable related to the "in" argument specifier. The value of the variable referenced is a list of node names, or IP addresses, representing the network path in which the firewall device is to be added;
- with: is the third argument specifier, and is used to specify the capability that the node must have;
- level_7_filtering: is a string representing the "with" argument value.

Algorithm 1 Landscape topology operations launched by the add_firewall instruction

$trafficPath \leftarrow Argument1$ ▷ the network path in which a new firewall ought to be inserted
$capabilities \leftarrow Argument2$ ▷ capabilities to be assigned to the new node
$node1 \leftarrow Argument3$ ▷ the node to protect
$networkGraph \leftarrow Argument4$ ▷ the graph representing the network topology
$capabilities \leftarrow Argument5$ ▷ the requested capability of the device to be added

for all $iterationIndex$, $iterationNode$ in $trafficPath$ **do**
  if $iterationNode = node1$ **then**
    $node2 \leftarrow trafficPath[iterationIndex+1]$
    break
  end if
end for

$networkGraph$.deleteEdgeBetween($node1$, $node2$)
$newNode \leftarrow NetworkGraph$.addNode(type="firewall", capabilities=$capabilities$)
$networkGraph$.addEdgeBetween($node1$, $newNode$)
$networkGraph$.addEdgeBetween($newNode$, $node2$)

Prevalidated Recipes

A set of Recipe remediation playbooks prevalidated to be used in a Botnet scenario are shown below, with some of which making use of previously discussed actions.

The following Recipe 5.1, performs a series of operations meant to deploy a new set of restrictive rules related to level 7 protocols. A capable devices is needed in the network topology to enforce the rules though, so the landscape is first queried by means of the "list_paths" and "find_node" actions, and based on the result of those actions the rules are applied, possibly by adding a new filtering device capable of enforcing them on each path interested.

Listing 5.1. Level 7 filtering Recipe

```
1  list_paths from impacted_host_ip to 'attacker'
2  iterate_on path_list
3     find_node of type 'firewall' in iteration_element with
          'level_7_filtering'
4     if not found
5        add_firewall behind impacted_host_ip in iteration_element
             with 'level_7_filtering'
6        add_filtering_rules rules_level_7 to new_node
7     else
8        add_filtering_rules rules_level_7 to found_node
9     endif
10 enditeration
```

The following Recipe 5.2 is equivalent to Recipe 5.1, but it deals with level 4 filtering instead of level 7.

Listing 5.2. Level 4 filtering Recipe

```
1  list_paths from impacted_host_ip to 'attacker'
2  iterate_on path_list
3     find_node of type 'firewall' in iteration_element with
          'level_4_filtering'
4     if not found
5        add_firewall behind impacted_host_ip in iteration_element
             with 'level_4_filtering'
6        add_filtering_rules rules_level_4 to new_node
7     else
```

```
8         add_filtering_rules rules_level_4 to found_node
9     endif
10 enditeration
```

The following Recipe 5.3 iterates on a set of given domains, and for each one adds a new policy blocking them at the level of the DNS server available in the network.

Listing 5.3. DNS blocking Recipe

```
1 iterate_on domains
2       add_dns_policy for iteration_element of type
            'block_all_queries'
3 enditeration
```

The following Recipe 5.4 iterates on a set of nodes impacted by a security incident, and moves each one to a reconfiguration network, in which the service operations are limited. This is useful in when recovering from a long lasting incident remediation.

Listing 5.4. Reconfiguration Recipe

```
1 iterate_on impacted_nodes
2     move iteration_element to 'reconfiguration_net'
3 enditeration
```

The following Recipe 5.5 iterates on a set of nodes impacted by a security incident, and for each one spawn a new honeypot host with a given "vulnerability" bait. In the end a mirror VLAN is created, working as a honeypot VLAN.

Listing 5.5. Honeypot Recipe

```
1 iterate_on impacted_nodes
2     add_honeypot with 'apache_vulnerability'
3 enditeration
```

The following Recipe 5.6 iterates on a set of nodes impacted by a security incident, and moves each one to an isolation network, in which only connections by security operators and security tools are allowed.

Listing 5.6. Servicing Recipe

```
1 iterate_on impacted_nodes
2     isolate iteration_element
3 enditeration
```

The following Recipe 5.7 iterates on a set of nodes impacted by a security incident, and simply turns them off. This Recipe has the highest impact on the service.

Listing 5.7. Isolation Recipe

```
1 iterate_on impacted_nodes
2     shutdown iteration_element
3 enditeration
```

5.2.3 CACAO Security Playbook format

Up until now, it has been discussed that the Recipe format. Nevertheless, Recipe Playbooks can be easily translated into CACAO Playbooks that enable more sophisticated machine parsing and handling capabilities thanks to its JSON structure. These characteristics become particularly important for Playbooks meant to be shared and manipulated by third parties or used within automated security pipelines.

As far as the interpreter goes, it shares many similarities with the one that handles Recipe Playbook. In fact, they are built along with the same architecture, except they are designed

to handle different playbook structures. CACAO Playbooks organize the workflow around step objects inside a dedicated dictionary entry of the playbook. Apart from the different structures, though, the execution process is equivalent, as is for the knowledge and variable management.

Within CACAO Playbooks, the equivalent of the Recipe global variables, which come from the knowledge repository and are stored inside the global scope of the first interpreter instance, is the "playbook_variables" dictionary entry, which is a designated entry in which variables that can be referenced throughout the playbook are declared.

For each step of the playbook workflow, two entries can be specified, "in_args" and "out_args". These two entries are used to specify the variables that are passed as input arguments to the target for the execution of the command corresponding to that step or that are yielded by the target after the execution of the command corresponding to that step. The target is the system, security control, tool, or human operator whose task is to execute the step's command. Within the framework architecture proposed in this thesis, targets correspond to the Security Capabilities that are supported in the operational environment. It should be noted that the target differs from the Interpreter. In fact, the interpreter task is to just execute the playbook, but it's not involved with the actual deployment of security actions 5.2.

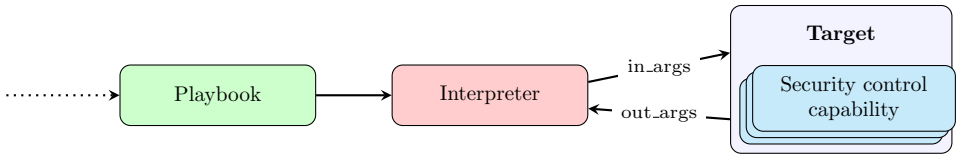


Figure 5.2. CACAO Playbook action schema

As for the commands used to represent actions, it has been chosen to adopt the same high level language used for Recipe playbooks instructions, thus guaranteeing full compatibility between the two interpreters, and most importantly making possible to use the same connector functions that implement the actual Security Capability configuration.

It is here reported an example of a Playbook in the CACAO format. It embodies the same workflow of the filtering Recipe Playbooks that have been shown before.

Listing 5.8. Filtering playbook in the CACAO format

```
{
   "type": "playbook",
   "spec_version": "1.1",
   "id": "playbook--uuid0",
   "name": "Level 7 filtering",
   "description": "This playbook will enforce a filtering policy leveraging
      on a firewall device, possibly adding it if not available",
   "playbook_types": ["remediation", "mitigation"],
   "created_by": "identity--uuid50",
   "created": "2022-02-24T05:00:00.123456Z",
   "modified": "2022-02-28T05:00:00.123456Z",
   "revoked": false,
   "valid_from": "2022-02-24T05:00:00.123456Z",
   "valid_until": "2023-02-24T05:00:00.123456Z",
   "derived_from": [playbook-uuid99],
   "priority": 16,
   "severity": 35,
   "impact": 4,
   "labels": ["malware", "filtering"],
   "targets": { },
   "extension_definitions": {
      "extension-definition--uuid2": {
         "type": null,
         "name": "iterate_on extension",
         "description": null,
```

```
25          "created_by": null,
26          "schema": null,
27          "version": null
28       }
29    },
30    "features": {
31       "if_logic": true,
32       "while-logic": true,
33       "extensions": true
34    },
35    "playbook_variables": {
36       "$$impacted_host_ip$$": {
37          "type": "ipv4-addr",
38          "description": "The IP address of the impacted node",
39          "value": "10.1.10.2",
40          "constant": false
41       },
42       "$$path_list$$": {
43          "type": "dictionary",
44          "description": "The list of traffic paths between the attacker and
                the impacted node",
45          "value": {"path_list": ["path1", "path2", ...]}
46          "constant": false
47       },
48       "$$rules$$": {
49          "type": "string,
50          "description": "The filtering rules to be enforced to protect the
                impacted node",
51          "value": "rule to be applied",
52          "constant": false
53       },
54    },
55    "workflow_start": "step--uuid1",
56    "workflow": {
57       "step--uuid1": {
58          "type": "start",
59          "on_completion": "step--uuid3"
60       },
61       "step--uuid3": {
62          "type": "single",
63          "commands": [
64             {
65                "type": "manual",
66                "command": "list_paths from $$impacted_host_ip$$ to
                      $$attacker$$"
67             }
68          ],
69          "on_completion": "step--uuid4",
70          "in_args": [
71             "$$impacted_host_ip$$",
72             "$$attacker$$"
73          ],
74          "out_args": [
75             "$$path_list$$"
76          ]
77       },
78       "step--uuid4": {
79          "type": "while-condition",
80          "condition": "$$iterationIndex$$ < $$path_list.length$$",
81          "on_true": [
82             "step--uuid5"
83          ],
84          "on_false": [
```

```
85              "step--uuid10"
86           ],
87           "step_extensions": {
88              "extension-definition--uuid2": {
89                 "type_of_while": "iterate_on",
90                 "info": "Interpreter will treat this while as a for each
                       statement",
91                 "iterate_on": "$$path_list$$"
92              }
93           },
94           "step_variables": {
95              "iterationIndex": {
96                 "type": "integer",
97                 "description": "Iteration index",
98                 "value": 0,
99                 "constant": false
100             }
101          },
102          "in_args": [
103             "$$path_list$$"
104          ]
105       },
106       "step--uuid5": {
107          "type": "single",
108          "commands": [
109             {
110                "type": "manual",
111                "command": "find_node of type firewall in
                       $$iteration_element$$"
112             }
113          ],
114          "on_completion": "step--uuid6",
115          "in_args": [
116             "$$iteration_element$$"
117          ],
118          "out_args": [
119             "$$found$$"
120          ]
121       },
122       "step--uuid6": {
123          "type": "if-condition",
124          "condition": "$$found$$:value = true",
125          "on_true": [
126             "step--uuid7"
127          ],
128          "on_false": [
129             "step--uuid9"
130          ]
131       },
132       "step--uuid7": {
133          "type": "single",
134          "commands": [
135             {
136                "type": "manual",
137                "command": "add_firewall behind $$impacted_host_ip$$ in
                       $$iteration_element$$ with"
138             }
139          ],
140          "on_completion": "step--uuid8",
141          "in_args": [
142             "$$impacted_host_ip$$",
143             "$$iteration_element$$"
144          ],
```

```
145             "out_args": [
146                 "$$new_node$$"
147             ]
148         },
149         "step--uuid8": {
150             "type": "single",
151             "commands": [
152                 {
153                     "type": "manual",
154                     "command": "add_filtering_rules $$rules$$ to $$new_node$$"
155                 }
156             ],
157             "in_args": [
158                 "$$new_node$$",
159                 "$$rules$$"
160             ],
161             "out_args": []
162         },
163         "step--uuid9": {
164             "type": "single",
165             "commands": [
166                 {
167                     "type": "manual",
168                     "command": "add_filtering_rules $$rules$$ to $$found_node$$"
169                 }
170             ],
171              "in_args": [
172                 "$$rules$$",
173                 "$$found_node$$"
174             ],
175             "out_args": []
176         },
177         "step--uuid10": {
178             "type": "end"
179         }
180     }
181     ...
182 }
```

5.3 Repositories

Given the prototype nature of the framework, it has been chosen to make use of simple JSON representations for the repositories. This allows the potential integration of other stateful features by means of lightweight NoSQL databases in the future.

The Knowledge repository plays a role of primary importance in the automated response to security issues, since the decision logic has the role of selecting the most appropriate playbook in response to a new threat intelligence report. The selection process makes use of many categorizations of threats, and it may also use threat modeling techniques in order to better identify the threat behavior. For example, by means of kill chain models or the MITRE ATT&CK knowledge base. For the proof of concept, it has been chosen to classify threats based on a generic categorization of the threat and possibly the particular strain of it. For each threat or strain, a set of related playbooks is maintained, each one specific for a particular task, such as investigation, mitigation, or remediation. Moreover, each playbook specifies the efficacy of the playbook in achieving the expected objective, and the impact it would have on the operational environment. These are all parameters that the Decision Logic component may evaluate in the playbook selection process.

The Playbook repository simply stores all available playbooks, be it in the Recipe or in the CACAO format, and for each one, it lists the security capabilities that are required in order to execute all its actions.

The Security Control repository is where operational details related to the available security controls are stored. In particular, for each security control, it is specified what security capabilities or features it can support, for example, a web application filter would have a level 7 filtering capability. Moreover, for each security control, a list of arguments is specified, these correspond to the variables that must be passed to the playbook action that makes use of a certain security control. For example, in the previously discussed CACAO interpreter, various "in_args" are passed to the target. Together with that, it is also specified which arguments are optional, and what constraints must be respected for their values.

5.4 Network Landscape

The network landscape represents the service graph abstraction that has been mentioned before. In this thesis, it has been chosen to use the igraph library in the Python language, motivated by its high performance even with heavily populated graphs.

A simple network graph topology has been built as a testing ground for the deployment of network related remediation Playbooks. The network is made up of various subnets, each one with a different role. In particular, the honeynet is used as a mirror network in which hosts that act as honeypots are placed. The other two special-purpose subnets are the reconfiguration subnet and the DMZ subnet. The reconfiguration subnet is where hosts that have to be reconfigured are placed, and for this reason, the outgoing and incoming connectivity is limited.

In order to effectively reproduce various network related playbooks, each node has been assigned some attributes, which are listed below:

- String identifier: it is used as a unique identifier in graph operations since it is easier to handle;
- IP address: represents the IP address that a certain node has in the deployed network.
- Subnet mask: as the IP address represents details about its IP layer connectivity;
- Node type: it is used to distinguish the various node typologies;
- Capability related ones: each node that represents a network service can have various attributes that describe its capabilities, for example, a firewall could be assigned a certain filtering capability.

Moreover, a set of constraints may be applied to service graph elements. These constraints are requirements regarding the nodes and the links between them, that must be respected in the underlying network when the various services and virtual devices are deployed on the physical network infrastructure. Even though these are not referenced and used by playbooks to determine the workflow, nor in the playbook selection stage, they are here briefly discussed. These constraints may, for example, be related to the computing capabilities that a node requires from the physical host on which it is deployed, such as the number of cores and memory, or storage requirements. Other constraints may concern link requirements, which are the properties that the physical connection connecting the hosts on which two nodes are deployed must respect. Some examples of these constraints are the average latency, the maximum latency, the average bandwidth, and so on.

In figure 5.3 it is depicted a simplified service graph topology that has been used in the evaluation chapter as a testing ground for the various security playbooks that embody incident response procedures aimed at protecting from botnet related activities. The graph follows the architecture that has been discussed previously, with various dedicated subnets.

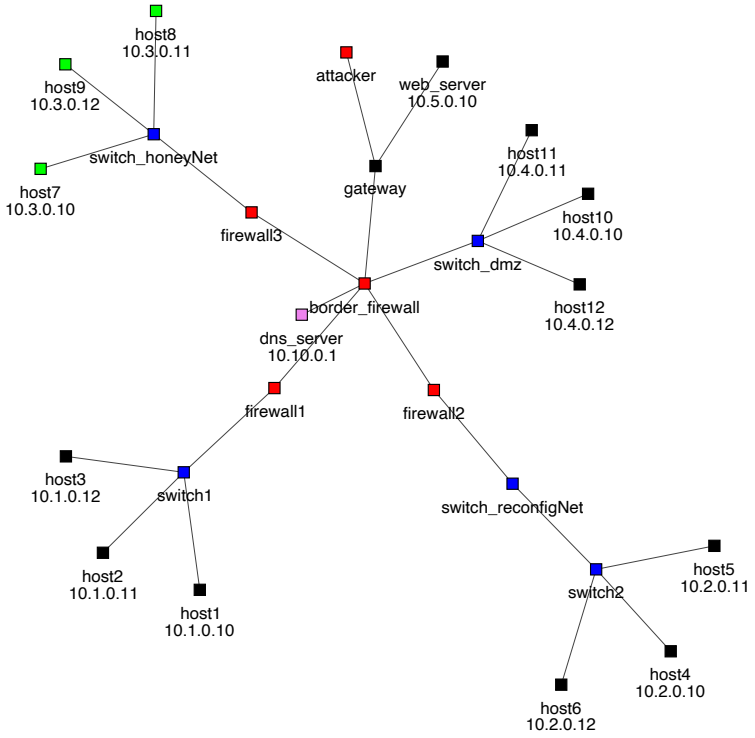


Figure 5.3. Service Graph

5.5 Threat Intelligence

The Threat Intelligence component of the architecture has been implemented through two dedicated methods that handle the ingestion of new threat intelligence reports in the STIX format and produce new ones at the end of the execution cycle. In the context of the use case chosen for the proof of concept, that is botnet related malicious activity, a threat intelligence report schema has been devised 5.4. This schema represents all the evidence about the security incident by means of the standard STIX objects, each one containing a piece of valuable threat intelligence, that will be later normalized and persisted in the Knowledge Repository. The main ones are discussed here.

The Malware SDO contains all relevant information about the malware that has penetrated into the environment, and can also be used to indicate its strain and characteristics of it. The Sighting SDO instead groups together all the evidence that has been sighted in the operational environment, each one represented by an Observable object. The Sighting object, in turn, references an Indicator object, this relationship is used to represent the fact that the Observed Data seen on the infrastructure confirms the presence of the Malware in question, which in turn is connected to the Indicator object, with a “indicates” relationship. The Indicator object embeds in it a STIX Pattern that can be used inside detection software or detection playbooks to identify the threat that is documented in this threat intelligence report. This helps increase the detection speed of threats. On the other hand, the same data is presented in a more structured way in the observable objects, which are grouped together via the Observed Data object. Moreover, the Malware object and the Indicator object present a kill chain attribute that contributes to a fine-grained Playbook selection process. In this particular use case scenario, the Indicator is assigned to the command and control stage of the Lockheed Martin Cyber Kill Chain, while the Malware object is to the

exploit stage. These data can be of primary importance in more sophisticated threats, but even in this simple use case can still contribute to an increased level of detail with which it is possible to categorize data into the Knowledge Repository.

When a new Threat Intelligence Report is received, data contained in it are normalized and registered into the Knowledge Repository. In contrast, after the incident response tasks have been completed, a new STIX Report is generated, equivalent to the first one, but with the addition of additional data that may have been gathered during the response process, and a new Course Of Action object, containing the Playbook in either the CACAO format or Recipe format. In figure 5.5, it is shown the report produced at the end of the incident handling life cycle in the scenario of the previously defined use case.

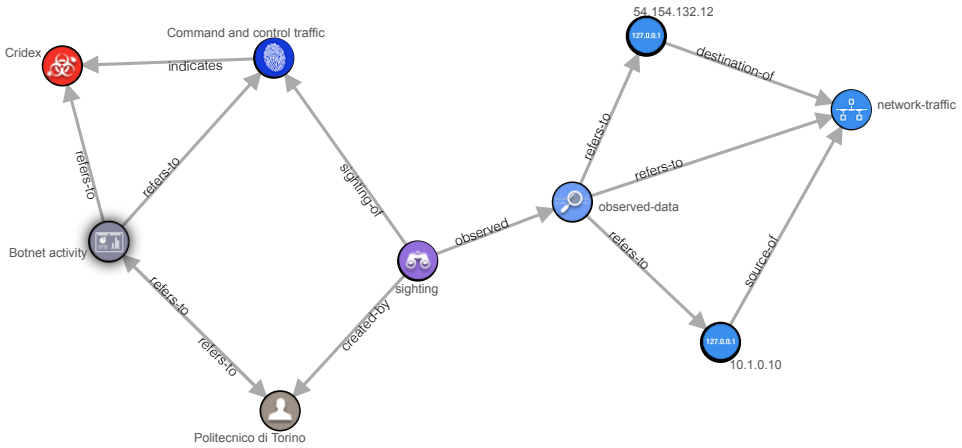


Figure 5.4. Graph representation of the STIX Report consumed

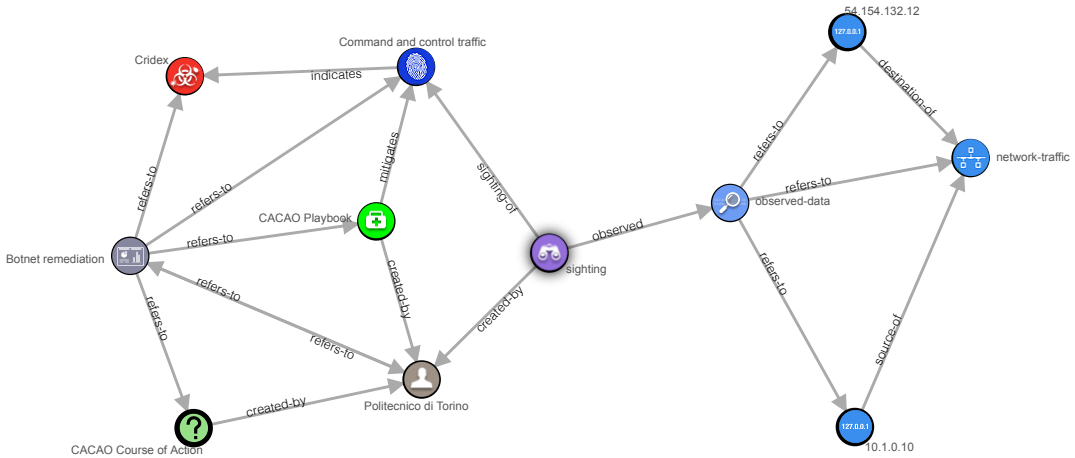


Figure 5.5. Graph representation of the STIX Report produced

Listing 5.9. STIX Threat Intelligent Report JSON

```
{
   "type": "bundle",
   "id": "bundle--29addfa2-2706-465f-8102-d43797f72356",
   "objects": [
      {
         "type": "relationship",
         "spec_version": "2.1",
         "id": "relationship--e8c107fe-92cb-407e-9912-5cd4a8b3aa19",
         "created_by_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
         "created": "2022-02-23T08:48:50.037427Z",
         "modified": "2022-02-23T08:48:50.037427Z",
         "relationship_type": "mitigates",
         "source_ref":
             "course-of-action--5b57c5b2-2f6b-45ff-975c-5c543a36add0",
         "target_ref": "indicator--df7adfb1-9668-4fe9-92cb-42e106711205"
      },
      {
         "type": "relationship",
         "spec_version": "2.1",
         "id": "relationship--e96ca28d-5b4a-4991-900a-b77a6147cdff",
         "created_by_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
         "created": "2022-02-23T08:48:50.037526Z",
         "modified": "2022-02-23T08:48:50.037526Z",
         "relationship_type": "refers-to",
         "source_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
         "target_ref": "report--92114365-8a29-4c35-9444-6a6cc286caea"
      },
      {
         "type": "relationship",
         "spec_version": "2.1",
         "id": "relationship--8149a4d4-4f65-49d8-b267-d1d12baff97f",
         "created_by_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
         "created": "2022-02-23T08:48:50.037618Z",
         "modified": "2022-02-23T08:48:50.037618Z",
         "relationship_type": "indicates",
         "source_ref": "indicator--df7adfb1-9668-4fe9-92cb-42e106711205",
         "target_ref": "malware--97011e79-90bf-4738-b0cb-5e0ef0554ce6"
      },
      {
         "type": "relationship",
         "spec_version": "2.1",
         "id": "relationship--ae5bbc1b-a695-4939-bf14-4763e26d5d00",
         "created_by_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
         "created": "2022-02-23T08:48:50.037707Z",
         "modified": "2022-02-23T08:48:50.037707Z",
         "relationship_type": "refers-to",
         "source_ref": "report--92114365-8a29-4c35-9444-6a6cc286caea",
         "target_ref": "malware--97011e79-90bf-4738-b0cb-5e0ef0554ce6"
      },
      {
         "type": "report",
         "spec_version": "2.1",
         "id": "report--92114365-8a29-4c35-9444-6a6cc286caea",
         "created": "2022-02-23T08:48:50.037273Z",
         "modified": "2022-02-23T08:48:50.037273Z",
         "name": "Botnet activity",
         "published": "2022-02-10T12:34:56Z",
         "object_refs": [
            "indicator--df7adfb1-9668-4fe9-92cb-42e106711205",
            "course-of-action--5b57c5b2-2f6b-45ff-975c-5c543a36add0",
            "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
            "extension-definition--f57fac2f-7a51-4d5b-854a-c0a862a7aaec"
```

```
            ]
        },
        {
            "type": "indicator",
            "spec_version": "2.1",
            "id": "indicator--df7adfb1-9668-4fe9-92cb-42e106711205",
            "created": "2022-02-23T08:48:50.028778Z",
            "modified": "2022-02-23T08:48:50.028778Z",
            "name": "Command and control traffic",
            "description": "This traffic indicates the source host is trying
                to reach to his command and control server",
            "indicator_types": [
                "malicious-activity"
            ],
            "pattern": "[network-traffic:dst_ref.type = 'ipv4-addr' AND
                network-traffic:dst_ref.value = '54.154.132.12' AND
                network-traffic:dst_port.value = '22']",
            "pattern_type": "stix",
            "pattern_version": "2.1",
            "valid_from": "2020-02-01T12:34:56Z",
            "kill_chain_phases": [
                {
                    "kill_chain_name": "lockheed-martin-cyber-kill-chain",
                    "phase_name": "Command and Control"
                }
            ],
            "labels": [
                "malicious-activity"
            ]
        },
        {
            "type": "identity",
            "spec_version": "2.1",
            "id": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
            "created": "2022-02-23T08:48:50.02564Z",
            "modified": "2022-02-23T08:48:50.02564Z",
            "name": "Politecnico di Torino",
            "identity_class": "organization"
        },
        {
            "type": "sighting",
            "spec_version": "2.1",
            "id": "sighting--665f8ded-84fb-474f-b7cd-eba5eb3de684",
            "created_by_ref": "identity--3826e082-a0c4-46e6-bbed-586ba2a337ba",
            "created": "2022-02-23T08:48:50.037839Z",
            "modified": "2022-02-23T08:48:50.037839Z",
            "count": 1,
            "sighting_of_ref":
                "indicator--df7adfb1-9668-4fe9-92cb-42e106711205",
            "observed_data_refs": [
                "observed-data--b2aa34d6-1780-4f1c-b03b-6c200623d16d"
            ]
        },
        {
            "type": "observed-data",
            "spec_version": "2.1",
            "id": "observed-data--b2aa34d6-1780-4f1c-b03b-6c200623d16d",
            "created": "2022-02-23T08:48:50.028153Z",
            "modified": "2022-02-23T08:48:50.028153Z",
            "first_observed": "2020-02-01T12:34:55Z",
            "last_observed": "2020-02-01T12:34:57Z",
            "number_observed": 1,
            "object_refs": [
```

```
                "ipv4-addr--6b8eb809-9746-5868-aee6-42c4a4b34228",
                "ipv4-addr--c55a4ce3-0755-5734-86dd-6f4e19f7eba1",
                "network-traffic--b2160560-4f6b-561c-ad8b-c51f67bbbefb"
            ]
        },
        {
            "type": "ipv4-addr",
            "spec_version": "2.1",
            "id": "ipv4-addr--6b8eb809-9746-5868-aee6-42c4a4b34228",
            "value": "54.154.132.12"
        },
        {
            "type": "ipv4-addr",
            "spec_version": "2.1",
            "id": "ipv4-addr--c55a4ce3-0755-5734-86dd-6f4e19f7eba1",
            "value": "10.1.0.10"
        },
        {
            "type": "network-traffic",
            "spec_version": "2.1",
            "id": "network-traffic--b2160560-4f6b-561c-ad8b-c51f67bbbefb",
            "src_ref": "ipv4-addr--c55a4ce3-0755-5734-86dd-6f4e19f7eba1",
            "dst_ref": "ipv4-addr--6b8eb809-9746-5868-aee6-42c4a4b34228",
            "dst_port": 22,
            "protocols": [
                "tcp"
            ]
        },
        {
            "type": "malware",
            "spec_version": "2.1",
            "id": "malware--97011e79-90bf-4738-b0cb-5e0ef0554ce6",
            "created": "2022-02-23T08:48:50.028641Z",
            "modified": "2022-02-23T08:48:50.028641Z",
            "name": "Cridex",
            "is_family": false,
            "kill_chain_phases": [
                {
                    "kill_chain_name": "lockheed-martin-cyber-kill-chain",
                    "phase_name": "Exploit"
                }
            ]
        }
    ]
}
```


5.6 Capabilities

Capabilities act as actuators of the actions that make up playbooks. Given the different levels of abstraction between, it is clear that the language used to express actions inside playbooks cannot be understood by the actual entities, such as security controls, that must enforce the policy derived from that action. For this reason, it is necessary to create a mapping mechanism allowing thhe translation of the action expressed in the playbook's high-level language to an equivalent command expressed in a format that can be interpreted correctly by the actuator. The actuator may be whatever device/tool that is used to perform a security relevant operation, and each one may have a different configuration language, since they are usually proprietary devices, such as firewalls, intrusion detection systems, SIEMs, and so on.

In the context of the use case explored in this thesis, the usage of a filtering capability was required. In playbooks that have been previously shown, this type of functionality is leveraged by

the add_filtering_rule action. At this end, it has been chosen to make use of a Medium Security Policy Language (MSPL) that acts as a bridge between the two different languages, the high-level one used to state the add_filtering_rule expression and the low-level one used to effectively configure the actuator.

5.6.1 iptables

For the enforcement of filtering rules, the actuator that has been chosen to deploy is iptables. In order to translate filtering actions contained inside playbooks into iptables commands, a translation mechanism that leverages a medium-level language has been used, as discussed in the previous chapter. In listing 5.10 it is shown how an actual instance of the medium level representation of the add_filtering_rule high-level action looks like. This medium-level language representation consists of an XML configuration file, that is eventually converted to the iptables native commands by means of a low-level policy translation tool written in the Java language that runs externally 5.6. The XML medium-level representation of the security feature is generated by means of a mapping function. These functions are present for each action to security control couple. The actual mapper tool that converts from the XML medium-level representation to the actual low-level iptables commands is part of another thesis work led by Aurelio Cirella. His thesis, titled "An abstract model of NSF capabilities for the automated security management in Software Networks", is discussed in detail how the refinement process works.

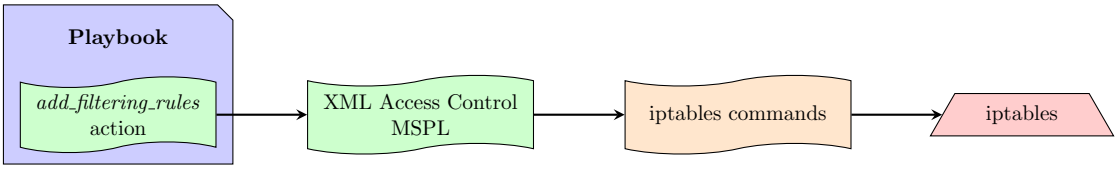


Figure 5.6. From playbook action to iptables configuration

Listing 5.10. XML Access Control MSPL

```
1  <policy nsfName="IpTables" targetRuleSet="INPUT"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="language_ipTables.xsd">
2        <!-- iptables -A FORWARD -p TCP -s 10.1.0.10/32 -d 2.2.2.2/32 -j DROP
             -->
3        <rule id="0">
4              <externalData type="priority">1</externalData>
5        <appendRuleActionCapability>
6                    <chain>FORWARD</chain>
7              </appendRuleActionCapability>
8              <ipProtocolTypeConditionCapability operator="exactMatch">
9                    <capabilityValue>
10                         <exactMatch>TCP</exactMatch>
11                   </capabilityValue>
12             </ipProtocolTypeConditionCapability>
13             <destinationPortConditionCapability operator="exactMatch">
14                   <capabilityValue>
15                         <exactMatch>80</exactMatch>
16                   </capabilityValue>
17             </destinationPortConditionCapability>
18             <ipSourceAddressConditionCapability operator="rangeCIDR">
19                   <capabilityIpValue>
20                         <rangeCIDR>
21                               <address>10.1.0.10</address>
22                               <maskCIDR>32</maskCIDR>
23                         </rangeCIDR>
24                   </capabilityIpValue>
```

```
25              </ipSourceAddressConditionCapability>
26              <ipDestinationAddressConditionCapability operator="rangeCIDR">
27                    <capabilityIpValue>
28                          <rangeCIDR>
29                                <address>2.2.2.2</address>
30                                <maskCIDR>32</maskCIDR>
31                          </rangeCIDR>
32                    </capabilityIpValue>
33              </ipDestinationAddressConditionCapability>
34              <rejectActionCapability/>
35        </rule>
36
37  </policy>
```

5.7 Decision Logic

The Decision Logic component is an abstract component encompassing the spectrum of tasks preceding the actual deployment of a Playbook, for which the Playbook Interpreter is designated.

The playbook selection process is one of those tasks that fall into it. During this process, various variables are to be taken into consideration. In fact, the effectiveness of the response to a given threat is also related to the quality and quantity of the data gathered as evidence of the incident. Much of this data is provided along with the threat intelligence report passed to the threat intelligence consumer, in which the security incident is detailed in-depth, both from an attack point of view and from an impact point of view. The threat intelligence gathering is an external task that can be both carried on by security operators via manual operations and by means of automated tools. In any case, the information related to threats or incidents provided in it is categorized by means of the previously mentioned kill chain model, thus facilitating the subsequent correlation. The report is accompanied by classification labels for the alert, that is, an indication of what is the threat that has been detected.

Moreover, the data provided in the threat report is coupled with what is stored in the Knowledge repository, which can, for instance, contain data related to previous occurrences of the same issue. For this reason, it is important to comply with the last stage of the NIST incident handling life cycle, that is, it should be necessary to document as much as possible all the steps taken in response to precious security related events so as to form a continuous intelligence enrichment. Within this architecture, this task is accomplished by means of the threat intelligence producer, which at the end of the playbook deployment yields a new report covering all the life cycle. Eventually, all this data is correlated in order to select the most appropriate Playbook capable of responding to this particular instance of the security incident.

An example of this process is represented by the addition of some indicators that are known to be connected to a certain threat to the information used to remediate an event. Whenever a strange behavior is detected in the network, for instance, an internal threat report is sent to the threat intelligence consumer, containing the raw artifacts representing the behavior together with the suspected threat label. Apart from this, though, no other information is known about the potential menace. For this reason, the label is compared with what is present in the threat repository, and if a match exists, then the threat report is enriched with the data contained in it. This data possibly adds new raw artifacts, such as IP addresses or malware hashes, that are related to the detected threat.

The playbooks contained in the repository are filtered by means of the labels provided in the report and the information available. This step leads to a reduced number of playbooks that ultimately leads to the selected one after the enforceability checks are performed. The enforceability checks guarantee that if no security control is available for a certain capability used in the playbook, then it is excluded from the potential responses. It must be noted that playbook associations with threat labels are pre-validated and continuously updated by security operators so that each playbook entails the presence of certain labels for its deployment. This avoids the risk of applying playbooks that are not compatible with a given threat notification.

Another task that is carried out just before the playbook deployment concerns the mapping between capabilities and security controls. In fact, each capability that is used throughout the chosen playbook may be enforceable by more than one security control, but only one must be chosen, preferentially one that is already available in the service graph, so as to optimize the network services resources deployment.

Once all relevant data is gathered, the playbook has been selected, and all the entailed checks have been performed, its enrichment is carried out, eventually becoming runnable by the interpreter as soon as all its input arguments are supplied.

Chapter 6

Evaluation and testing

In order to evaluate the performance on the proof of concept of the proposed architecture a series of tests have been conducted. These aim at testing various aspects, or component, of the proposed architecture implementation, along the whole workflow, from the reception of threat intelligence data to the automated response through the deployment of suitable playbooks.

6.1 Testbed and software used

All tests performed have been carried on a single machine with the following characteristics:

- Operating system: macOS Monterey;
- System architecture: ARM64 (Apple Silicon);
- CPU: M1 Pro (six 3.2 GHz cores + two 2.1 GHz cores);
- RAM: 16 GB UMA.

The following is the list of software and tools used for testings operations:

- Python 3.8;
- Python *timeit* library;
- htop.

6.2 Overview

For the quantitative evaluation of the proof of concept performance, it has been chosen to perform tests both taking into consideration one component at a time, and the whole workflow. The performed tests mainly gauge the execution time of the different processes which take place at different stages and in different components of the framework. In order to measure the execution time of processes, as previously said, it has been used the *timeit* Python library. The *timeit* library eases the testing process by taking on the task of repeating a certain test multiple times, without the need to manually repeat the test in order to achieve higher accuracy. The number of times a certain test is executed can be specified as an argument to be passed to the *repeat* function of the *timeit* library. In order to increase the accuracy of the tests, the result of them corresponds to the minimum value taken from the results of the repeated tests. This allows to minimize the noise effect of other processes running on the system taking over the processor.

As far as processor usage goes, it has been chosen to use the htop tool, which is a lightweight command-line tool that enables to inspect system resource usage, from RAM to CPU, with a

level of detail up to the single process. This allows to accurately tell whether the system usage is to be attributed to a certain process or to other background processes that have woken up for some asynchronous management task. In order to effectively measure the CPU usage during a certain test, it comes into help the previously mentioned repeat functionality of the *timeit* library. Thanks to it, in fact, it is possible to measure the average CPU usage along with a test that usually takes a few seconds. With this scheme, it is easy to measure the average CPU usage. It would be, in fact, difficult otherwise if it wasn't repeated multiple times since a single execution usually lasts a fraction of a second.

The components and workflow tested involve the threat intelligence processes, both incoming and outgoing, and the playbook deployment process, that is, the actual execution of playbook instructions.

6.3 Evaluation of the Threat Intelligence Consumer

This testing phase involves a series of measurements related to threat intelligence consumption operations in their entirety, that is, from loading in memory the threat intelligence report to the data normalization that occurs in order to make intelligence knowledge actionable, that is, interpretable by the interpreter and compatible in regards to how the knowledge repository is constructed and organized. Both these tests have shown a CPU usage that is not significant, around 10-20%, thus indicating low computational requirements for these tasks.

6.3.1 Latency

The tests performed are of two types. The first is a latency test conducted to validate the time needed to process a simple threat intelligence report, from the moment it is loaded in memory to the moment new data is persisted in the repository. The results are shown in figure 6.1, in which a series of iterations of the same process has led to a frequency distribution where intervals on the x-axis indicate the time needed to perform the data ingestion. From the figure, it is clear that the time needed remains consistently under the 6 ms value. The threat intelligence report used to perform this test consisted of a STIX Report equivalent to the one shown in figure 5.4, with one single indicator connected to some observed data objects. The number of tests performed to develop the frequency distribution data is around one hundred.

6.3.2 Scalability

Another test that has been performed regards the scalability of the threat intelligence ingestion process, which aims to measure how the system reaction times scale when giving as input bigger reports, which contain many more indicators and knowledge data in general. The actual test is based on the same report used for the latency tests but increasing the number of indicators objects. Each indicator object presents a relationship with an observed data object. The figure 6.2 represents the results of the testing done, which involved the ingestion of different reports within which the number of indicators ranged from 500 to 8000. From the plot, it can be observed that with the linear increase in the number of indicators, the time needed to process the report increases linearly, as the dashed orange tendency line confirms. Moreover, working on the data provided in the previous test and comparing them with this test, it can be observed an apparent inconsistency between the two, since the average time needed to compute just one indicator computed from the scalability test doesn't match the results of the latency test. The explanation of this behavior lays in the fact that regardless of the report size, a certain amount of time is required to bring the report to memory.

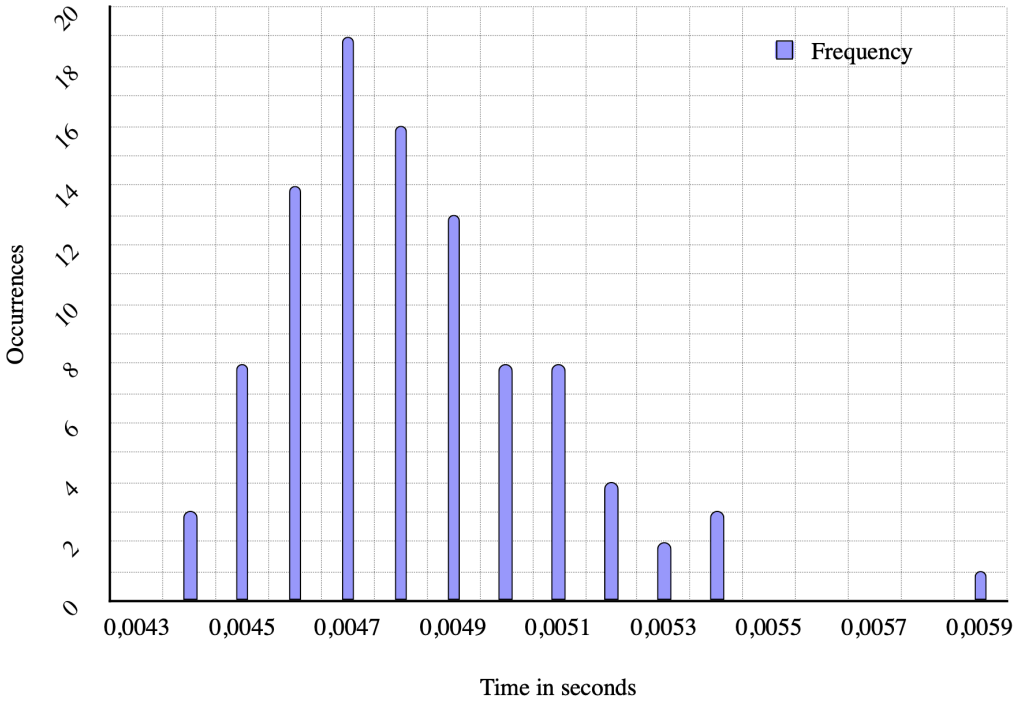


Figure 6.1. Frequency distribution of the consumer processing time

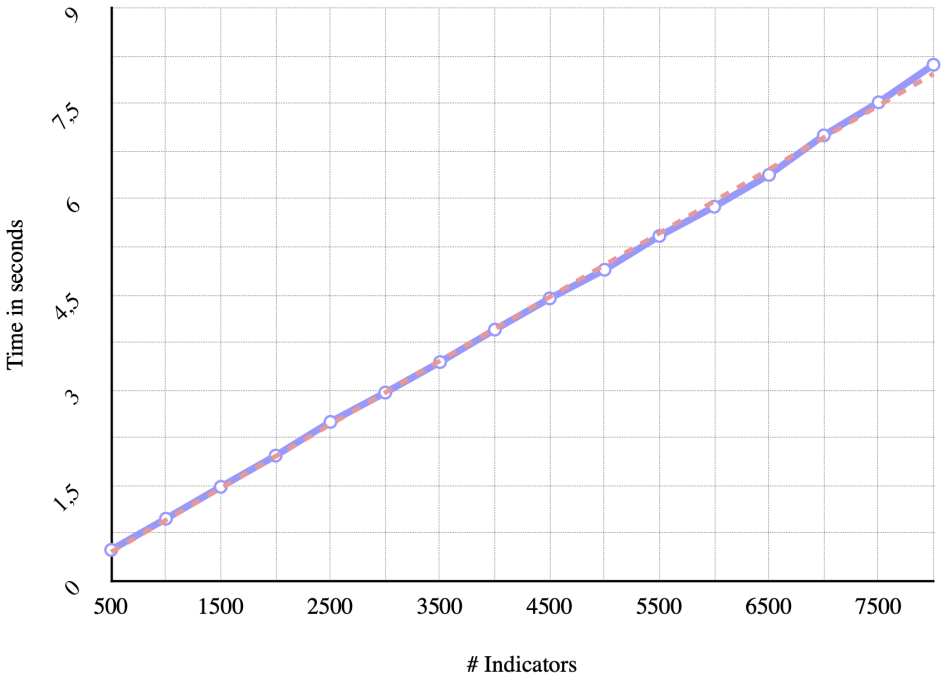


Figure 6.2. Consumer processing time by number of IoCs within the threat report

6.4 Evaluation of the Threat Intelligence Producer

This testing phase involves a series of measurements related to operations carried out by the threat intelligence producer. In particular the production of a STIX report holding in it all relevant information regarding the automated incident response adopted, together with ancillary information, such as the original intelligence report. This also include the generation of a playbook containing all relevant information for its execution by third parties, thus making it actionable out of the box. As for the Threat Intelligence Consumer tests concerning the Producer have also indicate a low computational footprint, with a CPU usage that stayed consistently under the 20% threshold.

6.4.1 Latency

Testing the threat intelligence producer involved the measurement of latency in a manner that is equivalent to the one discussed previously, in the context of the threat intelligence consumer testing. The results are shown in a similar frequency distribution plot, figure 2.8, and it indicates an average time of around 23 ms to generate the final report. The report generated originates from the handling of an incident response task in which a single indicator was present in the original threat report. So the final threat intelligence report is equivalent to the one shown in figure 5.5. As in the evaluation of the threat intelligence producer, the number of iterations used to produce statistically significant data about the latency aspect is around 100.

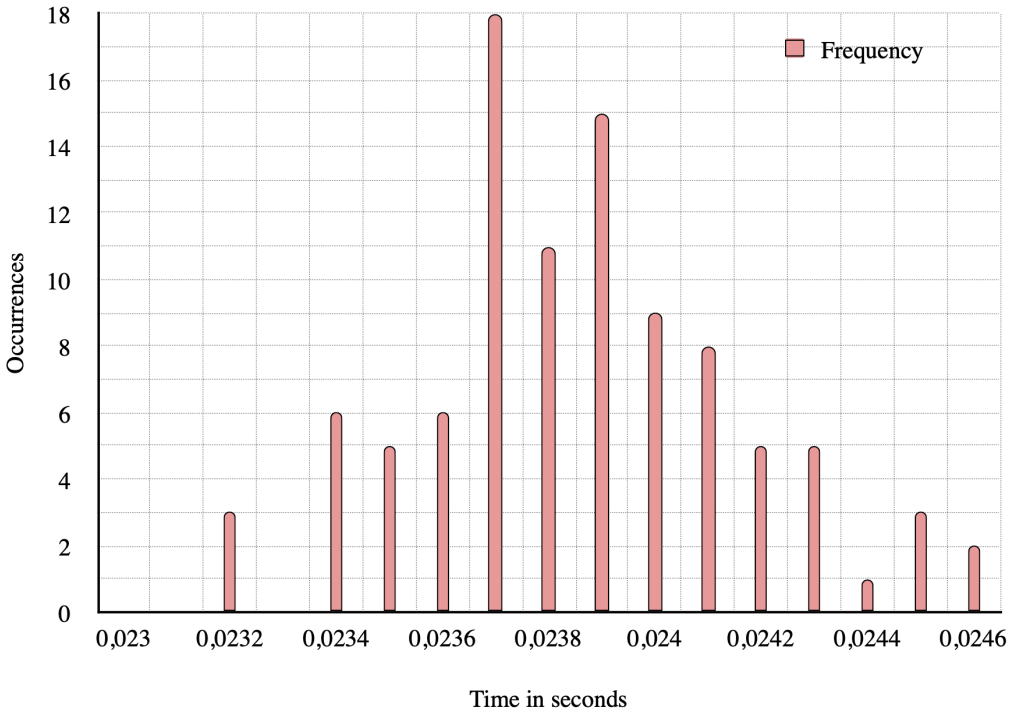


Figure 6.3. Frequency distribution of the producer lead time

6.5 Evaluation of the playbook deployment stage

In regard to the reactive component, that is, the actual playbook deployment, with which security actions are enforced by means of security controls, and the network landscape is potentially modified, another batch of tests have been performed. This series of tests have been conducted with the same methodologies as the ones discussed so far, also taking into consideration the same aspects, that is, latency and scalability.

As far as playbooks are concerned, two of the previously proposed playbooks have been used as references along with these tests. The first is the one used to assign new filtering rules to impacted nodes, and the second is the one that takes care of moving impacted nodes into another subnet, in particular into a reconfiguration network. The choice has fallen on these two because they manage to cover various aspects that are relevant in the context of performance tests. These are the computation of graph properties and paths, used in the selection of forwarding paths and in the repositioning of nodes along with the service graph, that is used in order to move hosts from a subnet to another and also the translation of a certain filtering statement to the corresponding low-level command, for which mapping the corresponding translator functions are used. The operations related to graph algorithms may be CPU intensive, especially with the increase in the size of the service graph, since many graph algorithms present an exponential complexity.

6.5.1 Scalability

The "add_filtering_rules" playbook has been evaluated by means of a scalability test in which the service graph size is maintained constant at the number of 1000 nodes, while the number of forwarding paths is increased linearly from 5 to 100. The playbook is deployed in response to a command and control channel detected in the operational environment, in particular from a single host to an external attacker. The number of forwarding paths represents the list of network paths that potentially allow the impacted host to reach the attacker node and thus require that a filtering rule be applied in order to avoid the establishment of a command and control connection.

The scalability test conducted led to the results as shown in figure 6.3. The plot shows a trend that is nonlinear but still maintains execution times that are consistently below the 1 second threshold, even for the higher intensive computations.

The second evaluation test that has been conducted, which sees the usage of the "reconfiguration" playbook, aims to evaluate the performance obtained when the service graph size grows bigger in the number of nodes. At this end, the playbook has been deployed to move 10 hosts to the reconfiguration subnet in a series of iterations in which the service graph size ranges from 1000 to 10000 nodes. The measurement results are shown in figure 6.4, and from that, it is clear that the increase in the execution time is linear.

During both the tests the CPU usage never went above 30%.

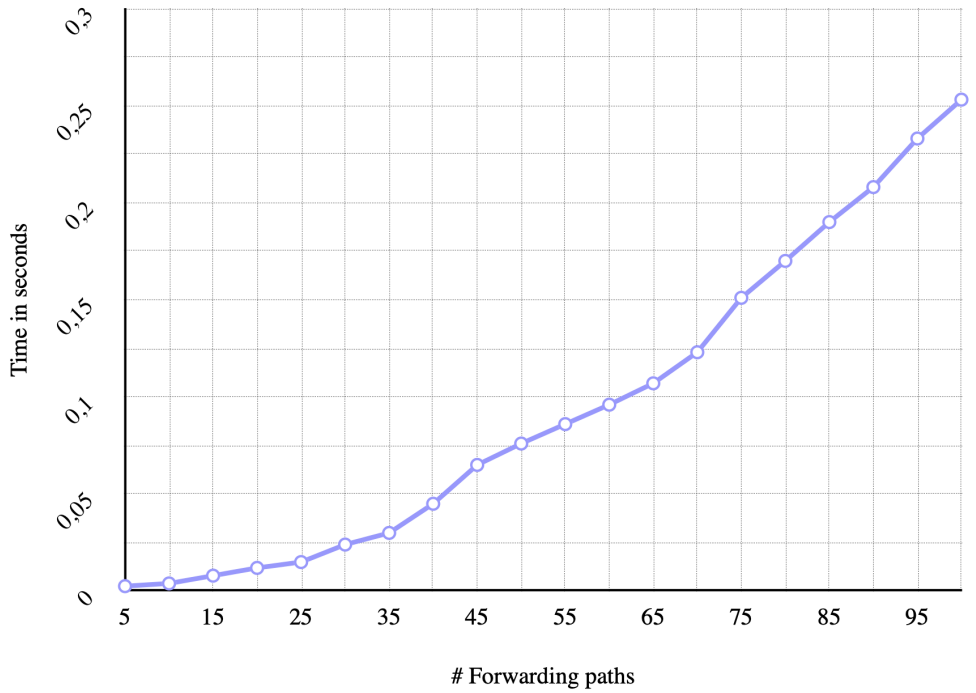


Figure 6.4. Response time by number of forwarding paths

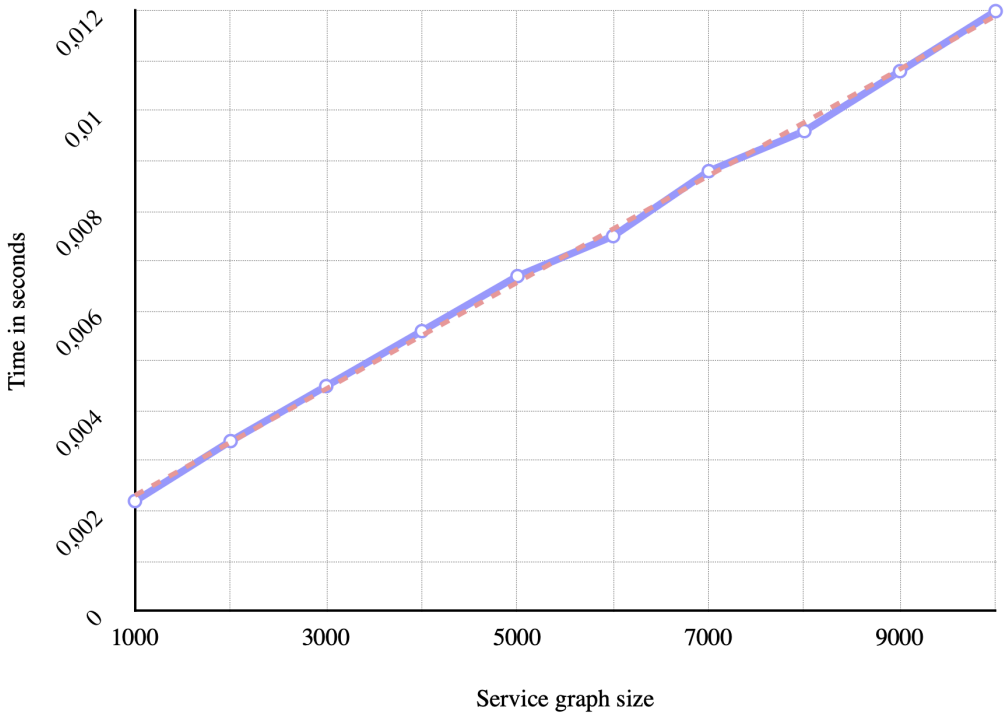


Figure 6.5. Response time by service graph size

6.6 Evaluation of the incident response life cycle

The whole workflow, from intelligence ingestion to final report generation passing by the actual incident response enforcement, is the object of testing in this section. In particular, the point of view that this test aims to make emerge is how the whole automated response, in particular to the use case proposed in this work, that is, a botnet scenario, can be efficient in contrast to the corresponding manual operations that would be needed in order to perform the same tasks.

In order to test the whole workflow, the same playbook used in the previous section to evaluate the scalability at the increase of the service graph size has been used, that is the one used to move a set of hosts to a reconfiguration subnet. As in some of the other tests performed before, the time needed to execute the whole process has been analyzed. By iterating the process multiple times, around 100 times, a frequency distribution has been produced, and its graphical representation is shown in figure 6.6. From this figure, it is easy to observe that the time needed for the reaction to an alert in the form of a STIX report stays below the 50 ms threshold, this while in the scenario mentioned above. During this test, the CPU usage stayed consistently around 30%. From the measurements carried out for this test, it has been excluded the time needed for the iptables translator tool to produce the low-level commands. This is due to the fact that its usage would have required a different testbed since the tool runs in a JVM. In any case, the performance of the translators is highly variable and depends on its actual implementation while being independent of how the framework architecture presented in this work is implemented. Nevertheless, other similar tests were conducted with a different testbed, a Docker container, in which the iptables translator performance indicated a latency of around 500 ms per rule.

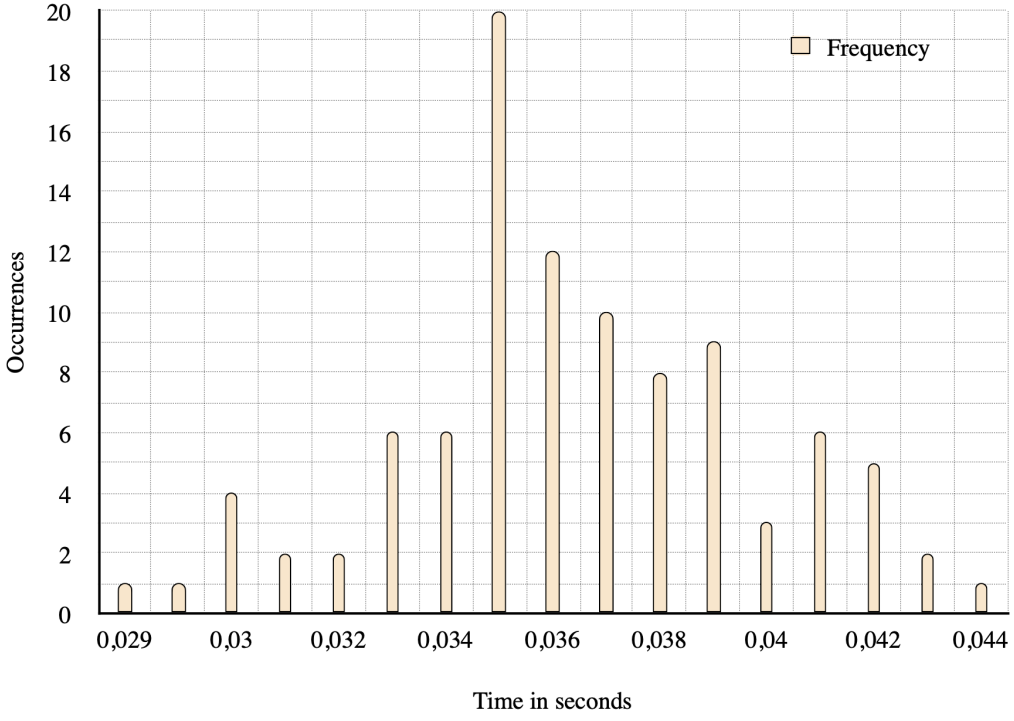


Figure 6.6. Frequency distribution of the entire incident handling life cycle

Chapter 7

Conclusions and future works

The cyber threat landscape is growing in sophistication and size, and organizations are quickly gaining awareness of the risks posed by it. Nevertheless, they fall short of changing their security posture in order to conform to the new reality. One of the main reasons for this is the lack of skilled personnel and the adoption of traditional approaches to security management, which are often obsolete in front of the scenario.

This thesis work is part of a larger research project that covers many of those aspects that must be taken into consideration when developing new solutions in the cyber security field. The scope of this thesis is focused on how to handle in an automated way some of those tasks related to security management with a particular focus on incident response related activities. Moreover, the importance of shared actionable threat intelligence is a central point of the work, and apart from the research perspective, a proof of concept has been implemented to demonstrate the applicability of this architecture, making use of the proposed methodologies.

The architecture proposed in this thesis combines the usage of shared threat intelligence with an automated response. At this end, it is proposed the use of the *Playbook* abstraction as a way to document and synthesize automated response procedures and the use of the STIX format to support those automated operations by enriching them with the knowledge contained within threat intelligence reports.

Even though many proposals have been made in the research concerning the use of threat intelligence throughout the security pipeline, no previous work tackled it in a consistent way that embraces the dissemination aspect together with the actionability of the knowledge, in particular by means of automated mechanisms. In fact, although the *Playbook* abstraction is a well known concept in the literature as well as in the cybersecurity industry, few works have been conducted on their integration in threat intelligence driven solutions.

On this basis, this work proposes two formats for playbooks description, with one, called Recipe format, being presented for the first time in this thesis and is the fruit of research made on the subject, while the other is an emerging standard from the OASIS organization, called CACAO. They are leveraged to enable the automation of security processes, and while the first mainly targets the human consumption issue, the latter is focused on being machine-readable and so the perfect match for integration within the STIX format. The Recipe format consists of a wrapper for a high level language used to express security related actions. This language has also been used coupled with CACAO playbooks, as a means to state high level actions.

As part of this thesis work, an engine has been built to interpret playbooks so as to apply the actions which are represented by it, which together form an automated response. As far as the actual responses go, a particular use case was taken into consideration throughout this work essay, that is, a botnet threat scenario. A set of playbooks is presented to handle some of the activities that can be automated in that context.

The proposed architecture makes use of the engine to deploy a set of responses that are mapped to a particular threat or task related to handling security in the context of that threat. The whole response automation is focused on handling a particular type of activity that is network related.

In light of this, the architecture leverages a service graph abstraction layer on which all security actions are applied.

Towards the end of this thesis, the proposed architecture is evaluated by means of a set of tests that aim to emulate real scenarios' conditions in which it would operate. Despite the prototype nature of the implementation developed for this thesis work, the tests demonstrated good scalability of the system, thus opening the doors to commercial usage scenarios for frameworks adopting this architecture.

Overall this thesis work provides a fresh outlook of the emerging approaches that enable to improve organizations' security posture. Nonetheless, a series of improvements can be made, and in the next section, a brief overview of the critical points is presented.

Future work

Various improvements can be made to the architecture, with some regarding the high level mechanism and workflow and others less significant being just helpful addition.

An example of an interesting feature that could ease the deployment of remediation playbooks in an automated fashion is leveraging on the STIX Patterning language by integrating response actions into the structure of the pattern so as to bind an indicator of compromise to its remediation or mitigation. Overall the idea is of inserting custom behavior in the semantic elements of a pattern. That is, when a STIX pattern is shared in STIX reports, the consumer is able to automatically enforce a protective measure in response to the indicator contained in it, leveraging only on the pattern without no other structured information. The STIX Python library enables the development of these functionalities without too much effort, thanks to its extensibility support.

From an higher-level perspective instead, it can be highlighted that some aspects have been left out of this work. This was done intentionally, since they would have gone out of the scope of this work, but here a brief summary of them is presented.

In this thesis, it has been mentioned the usage of kill chains for the modeling of sophisticated threats, and in a certain measure, kill chain models are used in the Playbook selection phase since the STIX format supports them. Despite this, in the framework, their usage is limited to the selection of a specific class of observed data based on the kill chain stage. Instead, as has been proposed in the literature, a good addition would be the usage of kill chain state machines, thus adding an additional layer of intelligence that can allow determining with higher confidence the best automated response to apply to a given threat.

Another aspect that can be an object of improvement regards the way in which high level security actions are enforced by means of security controls. In the current implementation for each security control, it is needed to prepare a translator function that translates the high level activities to the actual low level command or policy that will be enforced by means of the security control actuator. Alternatively, as has been the case with the action for applying new filtering rules, a mapping function can be used. This first converts the high level expression to an intermediate representation, which in the case of the add_filtering_rule was written in the XML language, and then a translator will convert that intermediate representation to the security control's own language. These approaches all require considerable efforts to handle the compatibility issues, so a better solution would be appropriate. To this end, the OASIS organization maintains a standard to handle these situations. The standard is known as OpenC2 and allows to express high level security-related actions in a structured way in a similar manner to how the CACAO format is used to structure playbooks. The OpenC2 format could be adopted as an intermediate representation for high level actions expressed in the Recipe format, and since the standard is being supported by various vendors, it would not necessitate writing translator functions for each security control.

Appendix A

User manual

In this chapter some information regarding how to run the proof of concept implementation discussed in the chapter 5 is provided. For ease of configuration it is advised to deploy it on a Docker container. In any case it has been tested also on traditional hosts, such as macOS and Linux, running natively.

A.1 Docker deployment

Docker is the advised way to deploy the proof of concept if it is not needed to make heavy development and changes to the source code. In that case it is preferred to install it natively on the work machine since it would speed up the development process. In all the other cases the Docker avenue is highly recommended. In order follow this method it is first necessary to install the Docker backend. Depending on the operating system, the install procedure may differ, and for this reason here it is not treated.

In order to deploy it on Docker the following Dockerfile must be used:

```
FROM openjdk:11
WORKDIR $HOME/
COPY . .
RUN apt update && apt install -y
RUN apt install python3-pip -y

ADD scripts /usr/src/scripts
WORKDIR /usr/src/scripts
RUN python3 -m pip install --upgrade pip
RUN python3 -m pip install wheel

RUN python3 -m pip install manimce
RUN python3 -m pip install -r requirements.txt
ENV PYTHONUNBUFFERED 1
ENTRYPOINT python3 /usr/src/scripts/remediation.py
```

Follows a step by step guide on how to deploy a container with the proof of concept framework:

1. Create a folder named *"remediation-framework"* on the host machine;
2. Copy the Dockerfile in the folder just created;
3. Copy the folder containing the source code into the same folder;
4. Open a terminal session and set the current directory to the folder created in step 1, named *"remediation-framework"*;

5. Run the following command *"docker build -t remediation ."* to create the Docker image;
6. Run the following command in order to spawn a new container and run it *"docker run -it remediation"*.

A.2 Native deployment

In order install the framework natively on the host it is necessary to install all the required libraries together with the Python runtime and Java runtime. The Java runtime is only used to execute the iptables translator, thus in case this is not needed its installation can be avoided.

The Python version preferred in order to avoid compatibility issues is the 3.8, but also 3.9 and 3.10 do not present any issue as of now. In general any new version that retains strict compatibility with version 3.8 does not pose problems. On the other hand, any version prior to the 3.8 is not guaranteed to run the framework.

The Python and Java runtime can be installed in various ways, it is left to the reader the choice. On macOS it is highly recommended to install the latest Python runtime via the *brew* packet manager, while it is instead discouraged the use of the Python instance baked in by default into the OS. Moreover in order to avoid cross library inconsistencies or conflicts it is highly advised to use Python virtual environment "venv", thus creating isolated environments, each one with its own libraries. This also eases the development process since it is possible to test out different versions of the libraries without entering in conflict with the others.

Here follows the list of libraries that ought to be installed. It may possible that some of these libraries are already installed or have been installed previously, but in any case it is recommended to re install it, since they would possibly be updated:

- *"nltk"*: this library handles natural language processing, and is used in playbook parsing operations;
- *"stix2"*: it is used to handle STIX data strcutures;
- *"igraph"*: it is the graph library with which the service graph is represented;
- *"matplotlib"*: this library is used to plot the service graph inside a matplot graphic environment;
- *"pycairo"*: this library allows to export the service graph to a file, in the chosen format.

In order to install these libraries it is necessary to install the *pip* package installer that eases the Python libraries management. For each of the previously listed libraries the following command should be run:

```
pip install library_name
```

Appendix B

Developer manual

This chapter focuses on the implementation details of the framework, and indicates the general guidelines to follow while making changes to the code, in order to extend or improve it.

B.1 Project structure

This is the list of Python modules:

- **cacao.py**: this module contains all the logic and data structures used to handle CACAO Security Playbooks, and is inspired to the official STIX Python library;
- **cacaoRecipes.py**: this module handles the logic behind the CACAO interpreter and Recipe converter;
- **iptables_translator.py**: this module is an interface to the actual iptables translator written in the Java language. Its task is of producing a medium level representation of the high level filtering rule;
- **manoAPI.py**: this module is a stub, and works as a connector between the service graph and the actual network substrate;
- **remediation.py**: this module covers various functionalities. First and foremost it contains the implementation of the Recipe interpreter. Moreover contains the logic behind the threat intelligence producer and consumer, and the logic behind the selection of a specific playbook;
- **SecurityControlFunctions.py**: this modules handles the mapping between security controls functions and their corresponding medium level translator function;
- **serviceGraph.py**: this module contains the data structure and the function used to implement the service graph abstraction, leveraging the *igraph* library;
- **STIX_Pattern_To_Dict_Translation.py**: this module is optional, it can be used to create a dictionary representation of a STIX pattern. In it a series of classes are declared extending the ones of the official STIX pattern visitor. It is very helpful when handling pattern containing multiple or nested observations.

B.2 Security controls configuration and extension

The framework can be extended by adding new security controls, which will be later usable within playbooks with the corresponding high level security action. In order to do this, two things are needed. First the mapping should be declared inside the *SecurityContrloFunctions.py* module. This allows the interpreter to invoke the corresponding *medium level translator function* when

the high level action is parsed. This mapping is done through a simple Python dictionary that will reference the function implemented in its own dedicated module. By turn, the function will take care of converting the high level action into the medium level configuration and then from this to the actual low level configuration.

The function will convert the action from medium level to low level by means of the *low level translator function*, that is specific of the given security control actuator. This function can be also declared in another module, and will be referenced directly inside the *medium level translator function*.

B.3 Repositories configuration

The three repositories are the object of this section. Each of them is meant to be extended, either manually, or by means of automated tools, and for this reason a basic specification regarding their structure is here given.

B.3.1 Playbook repository

This repository stores playbooks with other relevant information for their deployment. Its structure, along with its entries, is here reported. The root consists of the following entry.

- *key*: string, name of the playbook;
- *value*: dictionary, playbook details.

The *value* entry is a dictionary with three entries:

Description entry

The description entry contains a streamlined phrase that indicates what the playbook does, and what assets it impacts.

- *key*: string, “description”;
- *value*: string, playbook description.

requiredCapabilities entry

The requiredCapabilities entry contains a list of capabilities as strings. These capabilities are required to be available in the operational environment for the enforcement of the playbook policies.

- *key*: string, “requiredCapabilities”;
- *value*: list, [capability1: string, capability2: string, ...].

value entry

The value entry contains the raw playbook expressed in the high level policy language.

- *key*: string, “value”;
- *value*: string, the playbook.

B.3.2 Security Controls repository

It is a JSON dictionary with a single object at the top used to store a collection of security controls available as policy enforcers in the operational environment.

The structure of the dictionary, with its entries, is the following:

- *key*: string, name of the security control;
- *value*: object, details about the security control.

Each value object comes with four entries:

Capabilities entry

The Capabilities entry contains the list of security capabilities the security control can enforce policies for.

- *key*: string , "capabilities";
- *value*: array, [capability1: string, capability2: string, ...].

args entry

The args entry contains the list of arguments that MUST be passed to the security control function policy enforcing. Other details concerning the arguments are covered in other entries. They MUST follow the required format or value laid down in the args_details entry of the security control in consideration.

- *key*: string, "args";
- *value*: array, [arg1: string, arg2: string, ...].

optional_args entry

The *optional_args* entry contains the list of arguments that MAY be passed to the security control function. These arguments are OPTIONAL but, if they are used, they MUST follow the required format or value laid down in the *args_details* entry of the security control in consideration.

- *key*: string, "optional_args";
- *value*: string, the playbook.

args_details entry

The *args_details* entry is used to details specific requirements that a security control argument MUST comply with.

- *key*: string, "args_details";
- *value*: object, requirements object.

Each argument entry of its collection MUST follow this format:

- *key*: string, name of the argument;

- *value*: object, requirements for this argument.

The requirements object MUST follow this format:

- *key*: string, name of the requirement;
- *value*: string/object/array, it contains a requirement.

To specify that an argument value MUST be chosen from a predetermined set of values, the "options" requirement for that argument SHALL be specified. The value of the "options" entry MUST be an array of string values.

B.3.3 Threat repository

It is a JSON dictionary with a single object at the top used to store a collection of threats along with details about them.

The structure is the following. At the top level of the JSON lays the Threats entry, this contains all other components, structured on nested levels. Every object MAY be commented using the *_comment* entry. This will be ignored by the remediation framework.

Dictionary of threat types

Each entry of this dictionary represents a threat type, for example malware or unauthorized_access. This is the first level of abstraction.

Dictionary of threat families

This is composed of multiple threat families, and for each one of them an entry is present. This is the second level of abstraction. The key of those entry is the threat family name.

Threat family object

Each threat family is structured on two entries:

- *key*: string, rules;
- *value*: array, list of rule objects.

- *key*: string, playbooks;
- *value*: array, list of playbook objects.

Rule object

Represents filtering rules to be enforced by a firewall, and is structured on two entries:

- *key*: string, level;
- *value*: number, OSI layer.

The value MUST be between 7 or 4. 7 specifies an application level rule, and 4 a network layer rule.

- key: string, proto;

- value: string, protocol targeted by the filtering rule.

If the specified level of the rule is 7 then the payload entry MUST be present.

- *key*: string, payload;
- *value*: string, base64 encoded payload.

If the specified level of the rule is 4 then the c2serversIP entry MUST be present.

- *key*: string, c2serversIP;
- *value*: string, IP address subject of the rule.

Playbook object

The playbook object stores the playbook name, and information about its impact on the operational environment. Moreover the priority of the playbook is specified. Its entries are here listed:

- *key*: string, playbookName;
- *value*: number, the name of the playbook.

The name of the playbook MUST be the same of the one used to declare the playbook string in the Remediator.

- *key*: string, priority.
- *value*: number, priority of the playbook.

The *value* MUST be between 1 and 10. The priority is used during the remediation process, to determine which course of action is more suited in response to this family of threat.

- *key*: string, impact;
- *value*: string, impact of the playbook.

The impact determines how the deployment of this course of action impacts the SLA of the operational environment.

Bibliography

[1] F. Reynaud, F.-X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against network functions virtualization and software-defined networking: State-of-the-art", 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 471–476, DOI 10.1109/NETSOFT.2016.7502487

[2] A. F. Murillo Piedrahita, V. Gaur, J. Giraldo, A. A. Cárdenas, and S. J. Rueda, "Leveraging software-defined networking for incident response in industrial control systems", IEEE Software, vol. 35, no. 1, 2018, pp. 44–50, DOI 10.1109/MS.2017.4541054

[3] A. Hermosilla, A. M. Zarca, J. B. Bernabe, J. Ortiz, and A. Skarmeta, "Security orchestration and enforcement in nfv/sdn-aware uav deployments", IEEE Access, vol. 8, 2020, pp. 131779–131795, DOI 10.1109/ACCESS.2020.3010209

[4] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining", Journal of Network and Computer Applications, vol. 75, 2016, pp. 138–155, DOI https://doi.org/10.1016/j.jnca.2016.09.001

[5] L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A model for the analysis of security policies in service function chains", 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–6, DOI 10.1109/NETSOFT.2017.8004230

[6] F. Patzer, A. P. Meshram, and M. Hess, "Automated incident response for industrial control systems leveraging software-defined networking", ICISSP, 2019

[7] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide", tech. rep., NIST, August 2012

[8] I. M. Sholihah, H. Setiawan, and O. G. Nabila, "Design and development of information sharing and analysis center (isac) as an information sharing platform", 2021 Sixth International Conference on Informatics and Computing (ICIC), 2021, pp. 1–6, DOI 10.1109/ICIC54025.2021.9632989

[9] O. C. Briliyant, N. P. Tirsa, and M. A. Hasditama, "Towards an automated dissemination process of cyber threat intelligence data using stix", 2021 6th International Workshop on Big Data and Information Security (IWBIS), 2021, pp. 109–114, DOI 10.1109/IWBIS53353.2021.9631850

[10] P. Nespoli, D. Papamartzivanos, F. Gómez Mármol, and G. Kambourakis, "Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks", IEEE Communications Surveys Tutorials, vol. 20, no. 2, 2018, pp. 1361–1396, DOI 10.1109/COMST.2017.2781126

[11] OASIS, "STIX Version 2.1. Edited by Bret Jordan, Rich Piazza, and Trey Darley. 10 June 2021. OASIS standard." https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html. Latest stage: https://docs.oasis-open.org/cti/stix/v2.1/stix-v2.1.html

[12] F. Valenza and A. Lioy, "User-oriented network security policy specification", J. Internet Serv. Inf. Secur., vol. 8, 2018, pp. 33–47

[13] OASIS, "CACAO Security Playbooks Version 1.0. Edited by Bret Jordan and Allan Thomson. 23 June 2021. OASIS Committee Specification 02." https://docs.oasis-open.org/cacao/security-playbooks/v1.0/cs02/security-playbooks-v1.0-cs02.html. Latest stage: https://docs.oasis-open.org/cacao/security-playbooks/v1.0/security-playbooks-v1.0.html

[14] V. Mavroeidis, P. Eis, M. Zadnik, M. Caselli, and B. Jordan, "On the integration of course of action playbooks into shareable cyber threat intelligence", 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 2104–2108, DOI 10.1109/BigData52589.2021.9671893

[15] D. Schlette, M. Caselli, and G. Pernul, "A comparative study on cyber threat intelligence: The security incident response perspective", IEEE Communications Surveys Tutorials, vol. 23, no. 4, 2021, pp. 2525–2556, DOI 10.1109/COMST.2021.3117338

[16] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains", 2010

[17] P. Pols, "The unified kill chain." https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf

[18] F. Wilkens, F. Ortmann, S. Haas, M. Vallentin, and M. Fischer, "Multi-stage attack detection via kill chain state machines", Proceedings of the 3rd Workshop on Cyber-Security Arms Race, New York, NY, USA, 2021, p. 13–24, DOI 10.1145/3474374.3486918

[19] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. Pastor Perales, "Adding support for automatic enforcement of security policies in nfv networks", IEEE/ACM Transactions on Networking, vol. 27, no. 2, 2019, pp. 707–720, DOI 10.1109/TNET.2019.2895278