UNIVERSITÀ
DI TRENTO

Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer Science

Final dissertation

Configuration of threat protection systems for the enterprise

A data-driven real-world case study

Supervisor
Silvio Ranise

Student
Matteo Franzil

Academic year 2019/2020
Revision 2020071301

Acknowledgements

Fitting everyone's name here would be an impossible task. I strongly believe that no encounter and no relationship is random, and each person we meet in our life shapes it in his or her own way. I would therefore like to start by thanking everyone I met in these three years. Without them, I would be another Matteo right now.

Secondly, I would like to thank Trentino Digitale for the internship opportunity and all the support offered during the project. In particular, my deepest thanks go to Francesco Telch, Lorenzo Grosselli, Antonella Pellizzari, and all of the SOC.

Thirdly, I would like to thank all of my peers for shaping my life during my stay in Trento. From all the people I met in NEST, to all my university fellow-students, I would like to thank Francesca Annibaletti, my perennial food addiction companion, Lorenzo Masciullo, whose steps in cyber security were the ones I ended up following, and Paolo Baiguera, Simone Nascivera and Carlo Corradini, my university project teammates who endured my hot-headed temperament for three whole years.

Finally, I would like to thank my family. A very remarkable thank you goes to my father, my mother and my sister. They were instrumental in allowing me to study in Trento and giving me all the love and affection they could. My deepest gratitude goes to my uncle, a web developer and system administrator. Without him, I would be probably somewhere else studying god-knows-what. I will never be able thank him enough for suggesting me to direct my studies towards Computer Science, making me aware of a field I had never realized I would love.

Contents

Summary

Modern security systems nowadays are no longer limited to the basic, rule-based behavior that characterized them for well over a decade. Born as single-device firewalls capable of analyzing network traffic and making simple decisions based on source and destination IPs, in the last twenty years they evolved into complex and powerful beasts. They are now capable of thoroughly inspecting packets at hundreds of Gb/s, blocking threats, identifying patterns and providing an astounding level of protection even at higher levels of the internet protocol suite.

In enterprise, being equipped with an up-to-date, resilient, and proactive system is key to providing a complete protection to the whole internal network, the employees and all the equipment. Violations of confidentiality, integrity or availability must be avoided at all costs, as they may come with disastrous consequences such as a data breach or an irrecoverable information loss.

Often, the deployment of such a security system may prove to be hard, with legacy systems and compatibility issues slowing down the process. This dissertation aims to address this issue by providing a suitable methodology for the task. The methodology focuses on maintainability and modularity while providing a reasonable level of protection and avoiding performance setbacks.

The state of the art is initially analyzed, with security components available on the market being extensively studied. A data-driven, test-first approach is successively proposed, setting up a honeypot network mirroring the services available in production. Inbound and outbound traffic is then monitored and logged for an extended period. Such traffic is subsequently processed using dedicated software to gather statistically significant data and used to write efficient protection rules for each type of service. Finally, these rules are validated against an additional set of tests and finally deployed in production.

This approach was studied and adapted for Trentino Digitale's network, within an internship that lasted two months in early 2020. The company's *Security Operations Center* (SOC) oversaw the project and helped in the deployment of an ad-hoc testing network. A week's worth of data was collected and analyzed, with draft configurations being written down in the first phase of the work.

The COVID-19 pandemic in March 2020 brought the project to an abrupt interruption, initially delaying it. The internship was later resumed in remote working. Due to a severe lack of time, the deployment to production of the obtained configuration was ultimately halted. A different path was meanwhile conceived, partly demonstrating the efficacy of the methodology, but the full results of the work remain to be seen.

The dissertation is divided in four main chapters. Chapter 1 focuses on introducing fundamental notions and summarizes the important components of a security system. Chapter 2 further presents the state of the art for each of the components, additionally detailing each one's strengths and weaknesses. Chapter 3 presents a methodology for correctly configuring such components in an enterprise environment. Chapter 4 explores the application of the aforementioned methodology with a case study. Finally, the conclusions of the work are presented, outlining the results and suggesting future work on the subject.

1 Introduction

While *Intrusion Prevention System* (IPS) and *Intrusion Detection System* (IDS) technology has been around for at least thirty years, the actual necessity of strengthening networks with more than just a firewall only came up relatively recently. While in the early 90s traffic flowed unencrypted between hosts and web servers had an insignificant market share, rendering classic firewalls sufficient for defeating intrusion attempts, the following decades revolutionized the landscape. This forced both attackers and defenders to upgrade their tactics. The market prevalence of IPS solutions finally started ramping up in the mid-2000s, but only in the last few years it saw an integration with other network security solutions [19].

In a landscape of countless vulnerabilities, threats and exploit opportunities, enterprises are often a tasty treat for attackers, being usually targeted with the goal of seizing confidential data and ultimately for economical purposes. Often having started as small companies, middle-sized enterprises may inadvertently evolve into giant beasts with intricate networks, filled with legacy systems and outdated software. Especially in Italy, slow bureaucracy and strict policies worsen the situation, render deep interventions costly and time consuming, even more so in case of a successful breach to the network itself.

As an example, the median time between a successful network intrusion and the subsequent detection is more than two weeks [24], rendering proactive detection even more of a must. Even worse, nowadays the possible attack surfaces are countless and increase at an incredible pace, with threats incoming from both outside and inside networks. However, the literature agrees [19] [6] that the focus should be the protection of an internal network's fundamental security attributes[1], and its preservation from outside threats.

A modern enterprise must be therefore well equipped with a robust protection system, but also should be agile enough to react quickly to landscape changes, adapting its configuration to new threats without falling into cumbersome logistic black holes and mingled firewall policies. This dissertation aims to suggest a methodology that ticks all boxes, with the goal of writing a maintainable, light, and modular configuration for an enterprise network protection system.

1.1 Components

To provide a reasonable level of protection to a system and to supplement a firewall's basic protection, several network components have been commonly described in literature, with an IDS concept first appearing in a paper dating back to 1986 [1]. Nowadays, technology has evolved and diversified, with vendors offering various products specializing in in-depth protection of certain services or acting as an "all-in-one" solution.

From an enterprise standpoint, "the availability of a broad hardware range [...], a detection engine incorporating multiple technologies [...], IDS-like features to detect less critical threats [...] and centralized management and alerting capabilities across multiple devices" [25] are features considered critical for a security suite. Such features, especially in the last few years, have been included in a multi-purpose device called *Next-Generation Firewall* (NGFW). In even more recent times vendors have started to offer virtualized versions of NGFWs or single components, both as virtual machine files and in the cloud. Specialized hardware appliances, however, remain best suited for bigger companies with high-load networks and great amounts of traffic.

Digging deeper, we shall now analyze the single components that, from a software standpoint, will comprise our *protection suite* and are considered mission critical for the execution of the methodology. These are:

[1] Commonly called the "CIA Triad", it refers to Confidentiality, Integrity, and Availability, three attributes of the protected data at the core of information security [18]

- *Intrusion Prevention System* (**IPS**): a network device able to continuously monitor activity within a network, using patterns known as *signatures* or thoroughly inspecting the packet to detect unauthorized entry attempts. In case of a suspected attack, the device reacts accordingly by blocking the malevolent traffic and sending an alert [16];
- *Web Application Firewall* (**WAF**): a specialized device that augments the capabilities of a regular firewall by providing a deeper analysis of inbound and outbound traffic from and to a HTTP/HTTPS server, identifying advanced threats and malevolent content found in each packet's payload [17];
- *Denial of Service protection* (**DoS protection**): a traffic monitoring device that quantitatively analyzes the amount of traffic flowing between multiple sources and destinations within a single network interface, blocking or rate limiting when such traffic exceeds a specific threshold;
- *Transport Secure Layer termination* (**TLS termination**): often integrated as part of the aforementioned components or within a NGFW, it acts as a forward proxy endpoint for network connections secured with *Transport Layer Security* (TLS) such as HTTPS or SMTPS, terminating the encrypted session and enabling full inspection from the IPS or the WAF.

For the purpose of this dissertation, we will consider each of the aforementioned network protection features separately, since vendors may merge or split features, change naming schemes, and alter feature availability. Chapter 2 provides additional information on these components, including an overview on the current state of the art.

1.2 OSI stack

We conclude the introduction by recalling the conceptual model known as the *Open Systems Interconnections* (OSI) stack.

The OSI stack is an abstract model whose goal is to standardize the communication functions of network endpoints. This is obtained by partitioning the required components into various levels. Each level hosts one or more protocols and works unaware of the logic running the levels above and below it. In other words, Level N receives structured data from Level $N + 1$, implements its own logic, then send the newly wrapped data to Layer $N - 1$. This mechanism allows the complete encapsulation of each layer's logic, allowing its alteration and replacement without issues on bordering layers.

In real-world applications, the OSI stack has been only used as a blueprint, with no 1:1 implementation being available. It has however served as the underlying foundation of the internet protocol suite, commonly referred as the TCP/IP stack [29]. The TCP/IP stack features only 5 layers, with TCP/IP Layers 1 to 4 roughly mapping to their OSI equivalent and TCP/IP Layer 5 containing features of OSI Layers 5, 6 and 7. Nonetheless, this dissertation will stick to the OSI numbering usually used in literature. Table 1.1 briefly summarizes the structure of the OSI stack.

The components previously introduced may cover one or more levels of the stack, especially with enhanced market offerings whose feature set differs significantly from the standard one described in literature. Most advanced IPSs nowadays offer both standard inspection of the network and transport layers, and when coupled with a TLS termination system may additionally inspect the application layer payload. WAFs, on the other hand, are specifically built for analyzing HTTP/HTTPS payload and thus their scope is limited. Similarly, DoS protection systems usually maintain thresholds for Layer 3 and 4 traffic, as most Denial of Service attacks and aggressive port scans carry empty payloads. Finally, TLS termination proxies ideally work at Layer 6, where content encryption was initially conceived to be included, and typically work at TCP/IP Level 5 in real-world applications. Table 1.2 summarizes the typical working levels for all four components.

Level	Name	Description
7	Application	Provides applications with a high-level interface for accessing the communication environment. It includes protocols such as HTTP, FTP, SMTP, VNC, SMB, and more.
6	Presentation	Deals with the representation of transferred data, such as adapting a webpage to the aspect ratio of the endpoint device. Also manages encoding and encryption. In the TCP/IP stack, it is integrated in the application layer.
5	Session	Maintains sessions between recipients and structures the data exchange, enabling suspension, resumption and termination; additionally, deals with error recovery from Level 4. In the TCP/IP stack, it is integrated in the application layer.
4	Transport	Masks errors in Level 3, eventually segmenting/de-segmenting and reordering of packets. In TCP/IP, it includes protocols such as TCP or UDP, and manages the ports in the system.
3	Network	Provides the means of opening, maintaining and closing a connection between two endpoints. Also manages routing and congestion control. The IP protocol is tasked with these functions.
2	Data Link	Sets up the functional means for the transfer of data units between Level 3 entities, and masks Level 1 malfunctions. Also provides flow control. IEEE 802.11 WiFi and Ethernet are two protocols belonging to this layer.
1	Physical	Provides the physical means of establishing and closing a connection between entities, by transferring single binary units (usually bits). Additionally deals with encodings, connectors and voltage levels.

Table 1.1: The OSI stack.

Level	7	6	5	4	3	2	1	Notes
IPS	✓	—	—	✓	✓	—	—	L7 only with a TLS termination proxy.
WAF	✓	—	—	—	—	—	—	—
DoS prot.	—	—	—	✓	✓	—	—	—
TLS term.	—	✓	—	—	—	—	—	Works at Level 5 of the TCP/IP stack.

Table 1.2: Standard layers of operativity for the security components.

2 Related work

In this chapter, we provide an overview on the related work that is relevant to our proposal. Given the huge amount of research in the field and the wide variety of options available on the market, the work outlined in the following sections is not meant to be comprehensive.

2.1 Intrusion Prevention System

The first and foremost component of any network protection system is an *Intrusion Prevention System* (**IPS**). An IPS is a network device that supplements a regular network firewall's presence by inspecting traffic and making decisions based on the content of flowing packets.

IPSs were born in the late 2000s as an evolution of IDSs, sharing the same working concepts but differing on the response capabilities. An IDS would set off alerts, for example by sending them to a *Security and Event Information Manager* (SIEM). In contrast, an IPS takes further steps to neutralize the threat. Usually, either the packet is dropped, or the attacker is straight out prevented from accessing the entire network for a set amount of time.

From a high-level standpoint, IPS devices are usually divided into two different types: **host-based** and **network-based** [31]. The former is usually installed in an endpoint device and monitors only outbound and inbound traffic from that particular machine, the latter works as a standalone node and can be found as part of NGFWs and as an individual hardware appliance, being capable of monitoring an entire network. For an enterprise, a network-based IPS is usually the preferred choice in a cost-cutting and centralization perspective.

Within network-based IPSs, the literature usually further distinguishes two sub-categories of threat detection [6]:

- **Signature-based** detection: the IPS detects threats by comparing the single packet contents against a set of *signatures*, which match a known set of attack vectors.

- **Anomaly-based** detection: the IPS detects threats by analyzing the attack circumstances or the content of the packet itself, triggering if an anomalous behavior is detected.

Both solutions present benefits and drawbacks. Signature-based IPSs usually have a lesser impact on performance, although this is strictly dependent on the size of the database itself. Indeed, comparing a single packet against thousands and thousands of signatures can prove both a waste of time of resources. Moreover, signature-based detection is useless in case of zero-days threats[1] and variations of existing attacks that would fall outside of selected signatures. Nonetheless, signature-based detection has become the norm in modern IPS, with vendors offering authoritative signature databases. These databases are often updated by the hour once new threats are discovered.

On the other hand, anomaly-based detection may be worse performance-wise but usually addresses the issue of zero-day attacks. Early anomaly-based IDS used statistical baselines for detecting incoming threats, starting from the very first IDS concept in a 1986 paper [1]. Usually, the bandwidth, protocols, flags and more would be compared for identifying potentially illegitimate traffic. Newer anomaly-based IPSs have evolved, and when used together with a TLS termination proxy (section 2.4) are able to semantically analyze the content of the packet. With this approach, potential malicious activity such as buffer overflow attempts and malware injection are instantly detected. However, this may result in an even greater performance impact.

[1] A zero-day vulnerability is a vulnerability known only by the attacker exploiting it, and for which no mitigation solution exists on the market.

2.2 Web Application Firewall

While Intrusion Prevention Systems offer a suitably large amount of protection to all sorts of services, HTTP/HTTPS servers nowadays need special attention due to their ubiquity and intrinsic heterogeneity. With hundreds of different combinations of operating systems and application servers, each one with its own array of vulnerabilities, the risk of an unexpected intrusion is always behind the corner, especially with badly designed websites, outdated platforms, and insufficient network protection. *Web Application Firewall*s (**WAF**s) respond to this necessity by thoroughly examining all traffic directed and coming from web servers.

In contrast to IPSs, WAFs are a relatively recent creation, being first conceived in the late 1990s and early 2000s [2]. It was only a decade later that their usage became prevalent, coinciding with the web services boom and the corresponding surge in attacks directed at them.

From a technical standpoint, WAFs were built from the start with the goal of inspecting network traffic directly in Level 7. This is in contrast with a classic IPS's operating mode, usually limited to Levels 3 and 4. Each packet in transit within a network is therefore fully unpacked. If necessary, a TLS termination component (section 2.4) provides decryption. Finally, the payload is thoroughly inspected.

At the inspection level, WAFs may either use a *positive* or *negative security model*. The former defines a list of acceptable behavior, the latter rather defines what is not permitted. Usually, WAFs employ a combination of the two, by checking traffic against both models [17]. A combination of signature matching, semantic, and syntactic parsing is used to carry out this phase. Finally, WAFs can permit, block, or quarantine traffic, in a manner similar to IPSs.

The types of attacks detected by WAFs usually varies greatly between vendors, and an attempt at standardization was made in 2003 through the *Open Web Application Security Project* (OWASP) Top 10 List [15]. The list is updated annually and provides an overview of the most popular threats that WAFs must be able to detect and neutralize. Usually, attacks such as *Cross-Site Scripting* (XSS) and *SQL injection* (SQLi) are featured on the list, although the amount of attack surfaces is typically endless. Finally, WAFs sometimes include corollary detection capabilities, such as the detection of unencrypted credit card numbers within the payload, or the scan of attachments sent in `POST` requests.

While immensely powerful, WAFs do have drawbacks. Firstly, an overly aggressive signature engine may produce false positives, which may prove disastrous in case of traffic-heavy websites. Secondly, WAFs increase complexity within the IT infrastructure, potentially adding unwanted latency as the inspections increase. Finally, WAFs necessarily need to be sided with complementary components such as an IPS or a DoS prevention system, being unable to detect basic breach attempts such as a port scan or network-level attacks.

2.3 DoS prevention

A *Denial of Service* (**DoS**) attack is a type of cyber-attack where a network endpoint is flooded with an unmanageable amount of empty, superfluous requests. This usually results in a system overload and possibly a complete crash, disrupting legitimate requests and often requiring a full reboot. A *Distributed Denial of Service* (**DDoS**) attack builds upon the same philosophy. An array or *botnet* of infected network hosts, called *zombies*, are controlled remotely by an attacker. All zombies then proceed to target the same host with requests, increasing exponentially the effectiveness of the attack.

Nowadays, several solutions have been devised to address such types of attacks. Common defense mechanisms are divided in source-based, network-based and destination-based [33]. Source-based and network-based defense mechanisms attempt to stop DoS attacks at the source and before it reaches the target network. Within an enterprise, the focus therefore shifts on destination-based mechanisms. The ultimate goal is to stop the attack as far as possible from the intended destination, to avoid network congestions throughout the attack's path as much as possible [12].

Many destination-based techniques have been described in literature. Some of them include *history-based IP filtering*, where only packets belonging to a IP whitelist are allowed during an attack, and the usage of a *Management Information Base*, where statistical traffic patterns are recorded and regularly accessed to check whether a certain type of attack is ongoing [33]. While powerful, such a statistical approach that may prove ineffective in case of large-scale attacks that end up depleting most resources from the start.

The most common mitigation approach remains the usage of *network thresholds*. Thresholds pose a hard limit on the amount of inbound and outbound traffic within a network. When an appropriate threshold is selected, using parameters such as the network and host capacity, the probability of DoS attack is significantly reduced. While remarkably effective and performing, this method presents several drawbacks. The first is that setting a fixed threshold may negatively impact legitimate traffic during an attack. If a DDoS attack consumes 95% of the available bandwidth, the remaining 5% may prove insufficient for the regular traffic trying to access the host. The second is that thresholds may need to be continuously adjusted to cater to the changing capacities of the network and the hosts, possibly wasting human resources tasked with the management of these systems.

No DoS protection mechanism is 100% perfect. In fact, the literature suggests that intervention with network-based defense mechanism is far more effective in prevention of these attacks [33]. Either case, implementing threshold-based mechanisms within an enterprise can prove valuable if configured correctly.

Indeed, most modern NGFWs are nowadays enriched with threshold-based DoS prevention mechanisms. This functionality may be either provided within an IPS in the form of dedicated signatures, or as a standalone component. These components can inspect and catalogue traffic at Layer 3 (TCP or UDP) and Layer 4, registering source and destination IPs. Refined rules can be therefore created, encompassing generic rate limiting for inbound traffic to more elaborate thresholds for single services.

Finally, some high-end DoS protection devices also provide custom thresholds for single and multiple TCP/UDP ports, enabling detection of port scans, SYN and RST packet floods. Aggressive ones are usually detected, but stealthier ones (such as *Nmap*'s `T1` and `T2` modes) usually slip through and require the usage of a SIEM or a log manager to detect the threat.

2.4 TLS termination

While IPSs and WAFs are critical to the protection of a network, their firepower would be useless without a proper way of inspecting TLS-protected traffic. For example, an IPS inspecting an encrypted packet would be barely able to assess the contents of the header, and a WAF would not even touch the payload. Several approaches have emerged for **TLS decryption**, with many being described in literature [21].

Perhaps the most common technique is the usage of a TLS termination proxy or reverse proxy. A reverse proxy is an intermediary acting on behalf the backend server, serving requests from incoming clients. When configured with a copy of the protected service's TLS certificate, a reverse proxy allows full TLS decryption of incoming traffic. This unencrypted traffic can be later directed towards the internal network where other security components may be located.

While being a flexible tool, reverse proxies require IPSs or WAFs to be located after them, which may be infeasible in certain network setups. To address this issue, standalone tools performing decryption have emerged. Vendors such as Symantec and Gigamon provide dedicated appliances [21] for full TLS decryption, which can be arbitrarily positioned within a network.

Finally, some organizations may even require encrypted traffic to flow through even the internal network, further increasing security. A possible solution is the usage of NGFWs, whose vendors began integrating TLS termination solutions within their offerings. Thus, a NGFW can decrypt incoming traffic, direct it to other security components residing in the same device, then re-encrypt using the same certificate and dispatch it to the network.

While powerful, the aforementioned technique adds a significant overhead and may disrupt application flows which require a very low network latency. Additionally, NGFWs often offer options to dump decrypted traffic to another interface for later storage and inspection. This may pose an immeasurable privacy threat when used irresponsibly, for example as a means of data harvesting from customers [5].

3 Methodology

This chapter focuses on the definition of a practical, high-level methodology that shall be valid for defining the parameters of several network protection components as described in section 1.1.

The goal is not to start a new configuration from scratch, but rather to improve upon an existing one within a deployed enterprise network. To carry out this refinement, a data and test-driven honeypot approach is proposed. The configuration is iteratively updated in repeating steps, being thoroughly validated it against both automated and human-led tests, until a satisfactory balance between security and performance is found.

The strength of the methodology lies in its repeatability. The configuration is modular in nature, being split from the start into smaller, logically separated pieces depending on the destination and the protected service. Each piece can be then individually updated as the threat landscape evolves and the enterprise requires additional protection. Even in case of drastic network or policy changes, which may happen several times within the lifespan of an enterprise, the previous configuration can prove valuable as a starting block for the new changes.

What is described in the following sections is platform-independent, as popular solutions by vendors may differ by naming schemes, availability, IPS signature databases, and more. Moreover, the obtained configuration will be only valid for protecting public-facing services from outside threats. Other types of traffic patterns, such as intra-network traffic between subnets and outbound traffic rules, are not covered and may require a different approach to the problem.

3.1 Preliminaries

This methodology defines at a high-level the steps required for carrying out a structured definition and deployment of a network protection configuration. As an input, the methodology requires:

- An existing enterprise network of arbitrary dimensions, with communication carried out with protocols of the TCP/IP stack;
- A *Demilitarized Zone* or equivalent subnet, positioned somewhere within the network, with NAT or public IPs assigned allowing access from the outside;
- A set of network protection components matching the ones described in section 1.1;
- An existing configuration for the aforementioned device(s), either in plaintext form or in the vendor's proprietary format, such as a `.bak` backup file (*optional*).

The output will be an equivalent configuration file, which should be then ideally deployed in place of the existing one.

For brevity, the methodology will assume that all the security components reside in a single physical device additionally doubling as a "legacy" firewall with Level 3 filtering capabilities. A NGFW usually ticks all of these boxes, although for many enterprises this may not be the case, especially for networks with many appliances from different vendors. Connecting the devices in series and matching network coverage is therefore a viable alternative.

A set of core assumptions are then defined, which must stay valid throughout all phases of the methodology:

- **Immutable network**: we assume the deployment environment to be completely untouchable, only having access to current network patterns and configurations (e.g. `iptables`) while being unable to move or edit anything. This approach is common in enterprise, where networks are often convoluted and filled with legacy systems. Therefore, careful work is required to make significant modifications to the infrastructure.

- **Border firewall**: in enterprise, multiple entry points to WANs may be common, with several perimetric firewalls and multiple routes leading to DMZs and internal services. To ideally cover 100% of the inbound rules, we assume that the network protection device is positioned at the border of the network, with access to all inbound traffic directed towards the services to be protected;
- **Balanced configuration**: the obtained configuration should strike a compromise between security and performance. Often, system administrators enable indiscriminately all the protection features ignoring both the nature of the protected service and the potential performance impact, adding false positives, excessive packet buffering and processing time added by inspections. This may cause an increase in latency, a decrease in throughput, or worse downtimes and access issues to servers. A top-down approach is therefore preferable, where rules and signatures offered by the protection systems are carefully chosen and implemented step by step, while performance is monitored and issues are identified as soon as possible. The other side of the spectrum must be also avoided, as implementing default and basic protection rules is trivial and is usually insufficient for detecting advanced threats.
- **Compromised exterior**: we assume that traffic incoming from anything other than destinations defined in the firewall *white list* is insecure and may compromise the system. This is especially true in the later phases, when testing machines that may have been incorrectly configured could infect other parts of the network if not timely stopped.

We shall then distinguish three different phases of the process.

- **Data collection phase** (section 3.2): data about the current configuration and common threats is gathered for each service, using already-available information such as past attack data and previous configurations;
- **Test phase** (section 3.3): using a honeypot network, outsider threats are lured, captured and their content analyzed. All data is collected and processed for each type of service;
- **Parameter validation phase** (section 3.4): the data obtained in the previous phase is used for writing a configuration that is then tested against actual threats to validate its feasibility. This phase can be repeated an arbitrary amount of times until a satisfactory result is reached.

3.2 Data collection phase

The first phase consists in the enumeration of the services that will be protected. Under the assumption of an already established network, a good starting point can be the analysis of existing firewall policies, in order to obtain a comprehensive list of destinations within the DMZ. Most NGFW provide the option of tying the single firewall rule to one or more security policies: this option, if present, is ideal in order to avoid duplicate entries and facilitate the insertion of the configuration.

For each firewall rule, we check the number of destinations. If one covers more than a service (e.g. a multi-purpose server with HTTP and FTP services open, or a reverse proxy covering multiple appliances), then all the services shall be considered as a single "congregate", writing down all of them within the same row; else, we just take into account the single service. The latter case is usually optimal, easing the addition of ad-hoc signatures later without impacting performance or causing issues.

For each service, we take note of the operating system, the protocol(s) and service version(s). We also list past attacks that specifically targeted each service. Random port scans and attempts to use widely known exploits (such as WannaCry) shall not be counted for the purpose; however, if a particular service has been compromised in some way such as DoS or DDoS attacks, a data breach, or a vulnerability exploit, then we take note of the incident.

Finally, we evaluate the amount of traffic that the service handles. The baseline shall be measured in packets per second as an average over a large amount of time (possibly days).

Tables 3.1 and 3.2 the data that needs to be collected with random and non-exhaustive examples. The *Rule* column refers to a hypothetic firewall table row to avoid overcrowding with data.

Rule	Protocols	OS	Packets/second
11	HTTP, HTTPS (Apache)	Red Hat 8.2	300
20	FTP (vsftpd) SMBv1, SMBv3	Windows Server 2012 R2	50
34	SMTP, SMTPS (Postfix)	Ubuntu 18.04 LTS	20
50	HTTPS (nginx)	Ubuntu 18.04 LTS	1400

Table 3.1: First part of example data gathered for the validation phase, including protocols, OS and packets per second.

Rule	Past attacks
11	-
20	Constant EternalBlue exploit attempts by botnets over the last 2 years.
34	-
50	DDoS attack in November 2018. 150000 packets per second at peak. Buffer overflow (CVE-2019-7401)

Table 3.2: Second part of example data gathered for the validation phase, featuring descriptions of past attacks.

3.3 Test phase

In this phase, we shall gather additional data in order to get a full overview of the threats each service must be protected from, particularly if the initial data featured no past attacks and no particular vulnerability.

To avoid disruptions and downtimes for the production servers, a dedicated environment will need to be deployed to execute tests. First of all, a public IP bubble separated from the production one should be identified and reserved for the scope. This maximizes the overall security, minimizing unauthorized entry attempts in case of disasters and data breaches affecting the test machines.

Secondly, a set of fake machines mimicking the production servers shall be deployed within the network. Even in medium sized networks, making an exact copy of existing infrastructure may prove to be an impossible task. Fallback options can be then either the replication of a smaller subset of the network, or the creation of a honeypot network.

A honeypot is a specially deployed machine that closely mimics the content of a real server, baiting attackers with apparently legitimate content and inviting them to take over the system. Once broken into, they are usually no more than empty shells, although some elaborate honeypot mechanisms provide virtualization of real services with complete keylogging and monitoring of unaware attackers trying to breach the fake system.

An array of honeypots can be therefore deployed within a single subnet, possibly diversifying and customizing them in order to increase both similarity to existing infrastructure and attraction to potential attackers. Each honeypot should be equipped with its own public (or NAT) IP within the bubble, in order to be fully accessible from outside the enterprise network.

Between the newly created network and the public WAN, a full set of network protection components should be deployed. While a clone of the production equipment would be ideal, usually deploying lower-end versions should work fine as long as interoperability is kept and the feature set remains the same. Using appliances from different vendors is not recommended due to the infeasible amount of time that would be later required for porting over the configuration.

With everything set up, the network shall be exposed to the internet. Data should be then collected for a statistically relevant[1] amount of time by simply allowing it to be targeted by attackers. Important data to be collected include, for each firewall rule:

- Median inbound packets per second; more in-depth data can be collected by differentiating between TCP and UDP traffic or packet flags (e.g. `RST`, `SYN`);

[1] The amount of time for an experiment to be considered statistically relevant varies widely, with trial and error usually being the best way to approach time. Approximate times may be from a couple days to several weeks.

- Attack types, including the total amount of attempts and possibly the amount of successful breaches;
- Geographic provenience of attacks.

While this approach is usually successful for most services, sometimes services may be ignored by attackers if they are considered impossible to breach, while others may require a greater amount of data that automated tests cannot provide in any way. Usually, HTTP and HTTPS services fall into the latter category. In these cases, an additional round of ad-hoc tests conducted by human beings is required to integrate the previously obtained data.

Such tests may include the deployment of DoS attacks, silent and aggressive port scans, XSS and SQLi attacks, and more. Once a sufficient amount of data has been collected and processed, work must focus on the parameter validation phase.

3.4 Parameter validation phase

The goal of the parameter validation phase is to merge the data obtained in the previous phase with the current configuration, obtaining a set of parameters that may be applied to the production environment with no disruption.

If a configuration file is available from the production environment, then it can be used as a starting point. Else, work must be started from scratch using only data collected in the previous phase. For each component, we write a draft configuration, using the following guidelines:

- IPS: we shall include signatures for past attacks listed in section 4.4 and for the most popular attacks that were recorded in the test phase. Additionally, we include platform-specific signatures to provide a basic layer of security for that particular service: for example, for rule 11 in table 3.1, we add signatures for HTTP, HTTPS and Apache;
- WAF: usually a case by case approach works best, and the draft configuration can start from basic XSS and SQLi protection. The configuration can be then refined using the data collected beforehand. Additional levels of protection (e.g. other types of code injections, or sensitive data exposure) can be added, or HTTP header inspection can be enabled.
- DoS protection: thresholds should be inserted so that, depending on the data, legitimate traffic is allowed to go through while more aggressive port scans and DoS attempts are blocked. Usually, a threshold 1.3 to 1.5 times the median packet per second value is sufficient, both for generic thresholds and when more detailed data is available;
- TLS termination: if the service protected employs TLS-extended protocols such as SMTPS or HTTPS, TLS termination must be turned on in order to inspect any encrypted packet.

This draft configuration shall then be applied to the honeypot firewall, and a smaller subset of tests shall be run. These tests may include carrying out selected penetration tests for each service and the reconnection of the honeypot to the WAN. While new information about incoming attacks can prove valuable if collected, running more weeks' worth of automated tests can prove ineffective, since the goal of this phase is to test the effectiveness of the configuration rather than to collect data over a long period of time.

During this phase, the protection capabilities and performance should be closely monitored. TLS termination must be thoroughly verified, as attacks may slip through undetected in case of misconfigurations. If the newly enabled functionality disrupts the honeypot, blocks traffic or generates false positives, the issue must be identified and the configuration temporarily rolled back.

This final cycle may be repeated an arbitrary amount of times, until a satisfactory set of parameters is obtained. In accordance with the core assumptions, the configuration should be robust enough to withstand both common attacks and ad-hoc attacks that specifically targeted the service, while being flexible to integrations and having zero to minimal impact on the overall performance. *In medio stat virtus*: weak configurations barely different from default ones and exaggerated ones with

indiscriminately enabled features should be avoided at all costs, as they would defeat the purpose of this methodology.

Finally, the obtained configuration can be deployed to the production environment. This is a very delicate process and disasters of any kind may arise during this phase. In the worst case, clashes with existing policies and other firewall appliances may interfere with the traffic, causing downtimes and server-side errors. It is therefore fundamental be very precautious, possibly by planning downtimes and deploying little bits of the configuration at a time with regular backups. Each time the production configuration is updated, tests should be immediately conducted in order to ensure the functioning of the whole network. In case of failures, the configuration should be rolled back and returned to the test environment for a more in-depth analysis.

While the aforementioned method works, it can be slow especially for very intricated networks and long firewall rules. The deployment can be accelerated by avoiding downtimes and updating the configuration on the fly, although the strategy may prove catastrophic and is not recommended for large enterprises. Other mitigation strategies may include *A-B testing*, where the default network path with the old configuration is temporarily sided with another one featuring the new configuration. Load balancing can be then set up between the two paths, with constant testing and monitoring being introduced to verify the correctness of operations.

4 Case study

This chapter focuses on a case study implementing the methodology with the goal of verifying its effectiveness. This experiment was conducted during an internship in partnership with *Trentino Digitale SpA*. The whole project took place between February 2020 and April 2020 and lasted 250 hours.

Trentino Digitale is an enterprise formed in 2018 as a result of a merger, incorporating *Informatica Trentina SpA*, a service provider and software house, with *Trentino Network s.r.l.*, a telecommunications company administering the public network of the Trentino region. Over time, it became apparent that a radical rejuvenation and reorganization of the network was urgent.

Under strict scrutiny by the company's *Security Operations Center* (SOC), the MISP project was conceived just months before the merger and was a first step in this direction [13]. To further increase the overall security and to empower the SOC by prioritizing proactivity over reactivity, the IPS and WAF usage was to be bolstered. However, a structured approach was needed for the deployment of these technologies.

4.1 Project overview

As a result of the merger, several DMZs from Informatica Trentina came under management by Trentino Digitale. Being very heterogenous in terms of services provided, with various network paths and intermediate firewalls of different vendors filtering out traffic, a structured strategy was needed in order to increase network protection.

On the border of Trentino Digitale's own *Autonomous System* (AS), a pair of perimetric NGFWs was recently deployed, protecting the whole public administration network from outside threats. Their under-utilization was however apparent, with the IPS and the WAF's protection features only partially enabled in very few cases.

The goal of this project was therefore a total restructuration of the security configuration, maximizing the available protection for Trentino Digitale's services while minimizing the impact on performance. Modularity was also identified as a key aspect of the configuration, easing future interventions even after the end of the internship.

Following the methodology explained in chapter 3, a honeypot network was set up, mimicking selected service types using both open-source honeypot solutions and dedicated machines.

After over a week's worth of data collection, a draft configuration for all security components was written and then underwent through further testing. Popular penetration testing tools were used in conjunction with automated scripts and exposition to outside threats to provide a reasonable level of validation for the results.

With the project approaching its final phases, the COVID-19 pandemic forced a temporary suspension of the internship and the deployment to production was halted. The remaining part of the project was completed in *remote working*, although the final deployment had to be simulated due to a shift in priorities within the company. Nonetheless the results obtained can be considered meaningful for the purpose of this dissertation.

As a part of a non-disclosure agreement, networks, IP addresses, services and sensitive data described in this chapter are either fictional or edited out and do not reflect reality.

4.2 Preparatory study

The first part of the case study consisted in the analysis of Trentino Digitale's network, the identification of the services to be protected, and the firewall solutions used within the enterprise.

4.2.1 Network and services

The DMZ subject of the study had IPs located in the `172.1.0.0/16` subnet and was positioned after a pair of load-balanced perimetric firewalls and an unspecified amount of network nodes. This DMZ hosted both services offered to the internal network, whose traffic was not routed through the AS border, and services offered to the WAN.

Figure 4.1 summarizes an example of a route from the WAN to a server positioned within the DMZ.

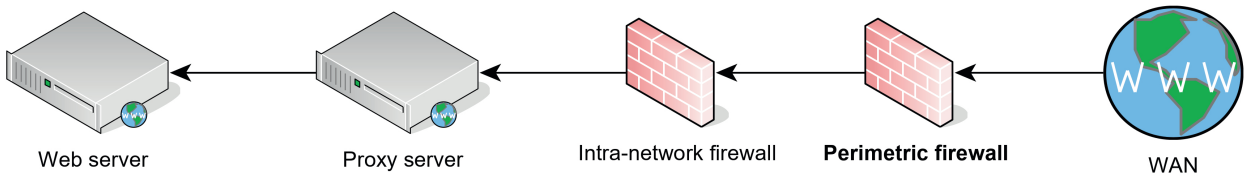

Figure 4.1: Example route from the WAN to a DMZ server.

Within the latter category, services ranged from simple Apache web servers, to server farms hosting multiple websites under a single Virtual IP, to `FTP` and `SMB` servers. Some of them were positioned behind an additional machine, usually a reverse proxy or a load balancer, that hid the true IP address of the server; some others were actually virtualized and shared hardware with other services while differing by their advertised IP address.

With strict policies in place, no modification (such as network configuration or routing) could be made to any service at all. Work first focused on the identification and enumeration of services, then shifted on the analysis of the current firewall setup.

4.2.2 Firewalls

Extensive work was put into the analysis of the firewall systems in use in Trentino Digitale, especially with regards to the capabilities offered by the *Fortinet* platform. As stated before, Trentino Digitale's AS was protected with a pair of two high-end FortiGate firewall appliances, clustered in an *Active-Active* configuration[1]. These firewalls were configured to monitor most network routes leading to the WAN, although they additionally managed selected intra-network traffic within the AS.

From a software standpoint, the firewalls were running a recent version of *FortiOS*. The documentation provided by Fortinet was studied, analyzing the capabilities of the system. Upon further inspection, the features needed for the case study were identified, matching them to the components described in section 1.1:

- **Intrusion prevention**: the system allowed to create custom IPS sensors that may either include an arbitrary amount of signatures, or "category filters" that would include groups of signatures based on their applicability (for example for an Apache server, or a Linux system, or for HTTP traffic and so on). Signatures would be automatically pulled from *FortiGuard*, Fortinet's central security database, and used a custom nomenclature system that will be used throughout the chapter;

- **Web Application Firewall**: the system included a limited, predetermined set of "sensors" (such as SQL Injection, Cross-Site Scripting, Trojans) along with checks regarding the semantics of the single HTTP request (e.g. Illegal HTTP Version, invalid Payload Length) that could be turned on and off within a single security profile;

- **DoS protection**: the system featured a dedicated panel for setting thresholds for each firewall rule, both for TCP, UDP and SCTP packets and depending on source and destinations;

- **SSL deep inspection**: an implementation of a TLS termination proxy, it enabled inspection of otherwise inaccessible TLS-encrypted traffic by other sensors such as IPS or WAF.

[1] An Active-Active configuration is a *High Availability* configuration that allows load balancing between two or more network nodes sharing the same configuration.

Another key feature was the possibility to switch between inspection in *flow mode* or in *proxy mode*. The former analyzed packets as they came through, while the latter introduced buffering to enable in-depth inspection and was strictly required by the WAF, along other security features. Downsides, however, included the introduction of latency and possible time-out errors for slow or out-of-date servers.

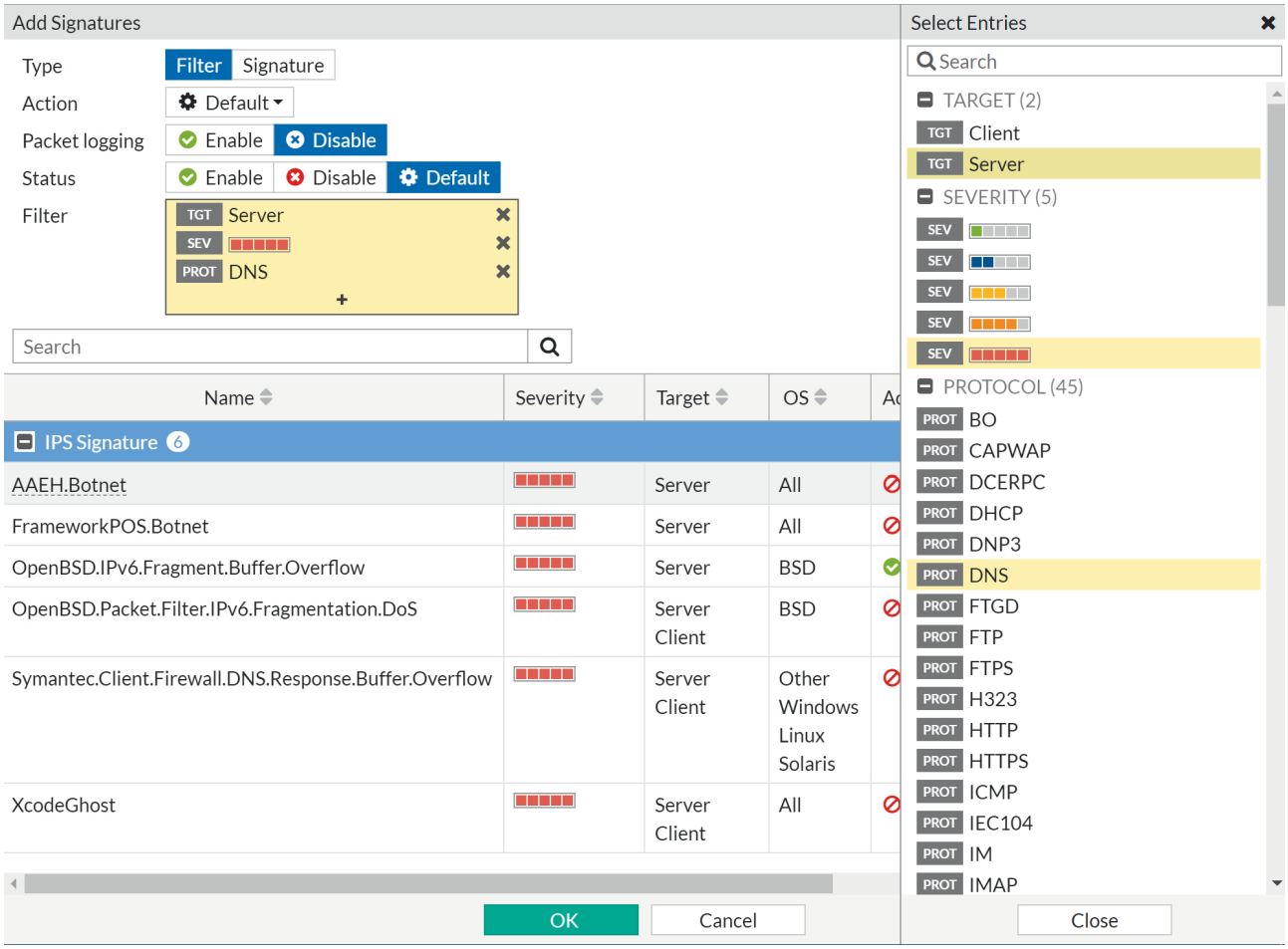

Figure 4.2: A screenshot from Fortinet's IPS, showing a sensor employing Server, DNS and Very High Risk signatures.

The production firewall configuration was thoroughly analyzed and copied down. While some firewall rules already had enabled some IPS features, most others had not and the goal was to standardize everything and re-apply rules in a consistent manner.

On the other hand, the configuration of the intermediate firewalls (provided by vendors different from Fortinet) was unknown and could not be edited in any way; the only information available was that no IPS or WAF had been enabled. Any modification within the perimetric firewalls could not then alter their performance or their output, especially with TLS termination enabled.

4.3 Environment setup

Particular attention was given to the creation of a reliable honeypot DMZ, in order to get credible, statistically relevant results in an environment that could both closely match the production one and appear desirable to attackers.

4.3.1 Honeypot machines

The whole honeypot network, to ensure both a high degree of scalability and security, was completely virtualized and special care was taken in choosing the correct service configuration.

An enterprise-grade server, used as a centralized virtualization environment, was built and put in place with the following configuration:

- **CPU**: 16 × Intel Xeon E5620 @ 2.40GHz
- **RAM**: 38 GB
- **Storage**: 4 × 200 GB, RAID 5
- **OS**: Proxmox VE 6.1

Work then focused on the generation of the *Virtual Machines* (VMs). Many open-source alternatives were found and tested: while most were discarded, the *Cowrie* [4] and *T-Pot* [30] machines made the cut. These two honeypots were supplemented by a vanilla installation of an *Apache* server and a *Splunk* [27] server acting as a log recorder and data collection tool.

Selected code excerpts, outlining each machine's configuration, can be found in appendix A.

Cowrie

Cowrie [4] is an open-source honeypot tool, using a customized Debian environment to emulate both SSH and Telnet services. Attackers trying to access ports `21` and `23` are tricked into having successfully penetrated the system. In reality, they are actually browsing a completely fake and virtualized Unix system. This system can be fully customized, for example by changing access credentials and by providing fake files to be downloaded within the system. Each login attempt, both failed and not, is recorded along with the command history of each session.

The Cowrie machine was primarily used as a username-password sniffer tool, in order to collect data about common credential attempts.

T-Pot

T-Pot [30] is another open-source honeypot tool. Using a custom Debian environment, it runs Docker versions of other open-source honeypot tools in a single box along with an ELK stack for managing collected data.

T-Pot was used primarily for hosting common vulnerable services such as `SMBv1`, `MySQL` and `VNC`, and was therefore the most targeted machine during all testing phases.

Apache server

A simple Ubuntu Server 18.04 installation was used as a base for installing a *LAMP* stack running fake versions of websites hosted in the real DMZ. The installation was further customized with an installation of DVWA [8], a web application used tool for testing web-based vulnerabilities.

The machine was initially used as an HTTP-only server and was all but ignored in the first phase. However, it was later upgraded and given a TLS certificate in the second phase.

Splunk

Finally, another fresh copy of Ubuntu Server 18.04 was used in conjunction with Splunk [27], a data collector and aggregator, with powerful query tools integrated.

Splunk was both used as a syslog collector and as a query tool for data incoming from the other honeypots, creating a centralized platform from which data could be obtained from.

4.3.2 Network configuration

The following step was to connect the virtualization environment to the WAN. A then-unused public IP range (pictured blue, `123.45.67.88/29`), completely separated from the enterprise network but within the AS, was used as an entry point and was connected to an entry-level *FortiGate 60E* [9], to be employed both as a perimetric firewall and as a traffic sniffer unit. The 60E, while bearing lower bandwidths, was geared with the same software features as the production ones.

All the VMs installed in the Proxmox virtualizer were placed in a dedicated subnet (pictured gray, `192.168.100.0/24`) and connected to the FortiGate.

Both the virtualizer and the FortiGate were then wired to the internal enterprise network (pictured orange, `10.2.0.0/24`), along with a PC used specifically for monitoring and running tests. This enterprise network was only used for management and could not communicate with anything but the firewall and the Proxmox management interface. To avoid resorting to the `VNC` interface provided by the Proxmox web portal for managing the VMs, a *Virtual Private Network* (VPN) was set up on the FortiGate, allowing access to the honeypot subnet.

Finally, basic firewall rules and NAT were set up, assigning public IPs to the two honeypot machines and the Apache web server within their subnet range (initially `123.45.67.90`, `123.45.67.91`, `123.45.67.92`) and allowing traffic to go through.

Figure 4.3 describes the resulting network with IP addresses for each machine installed.

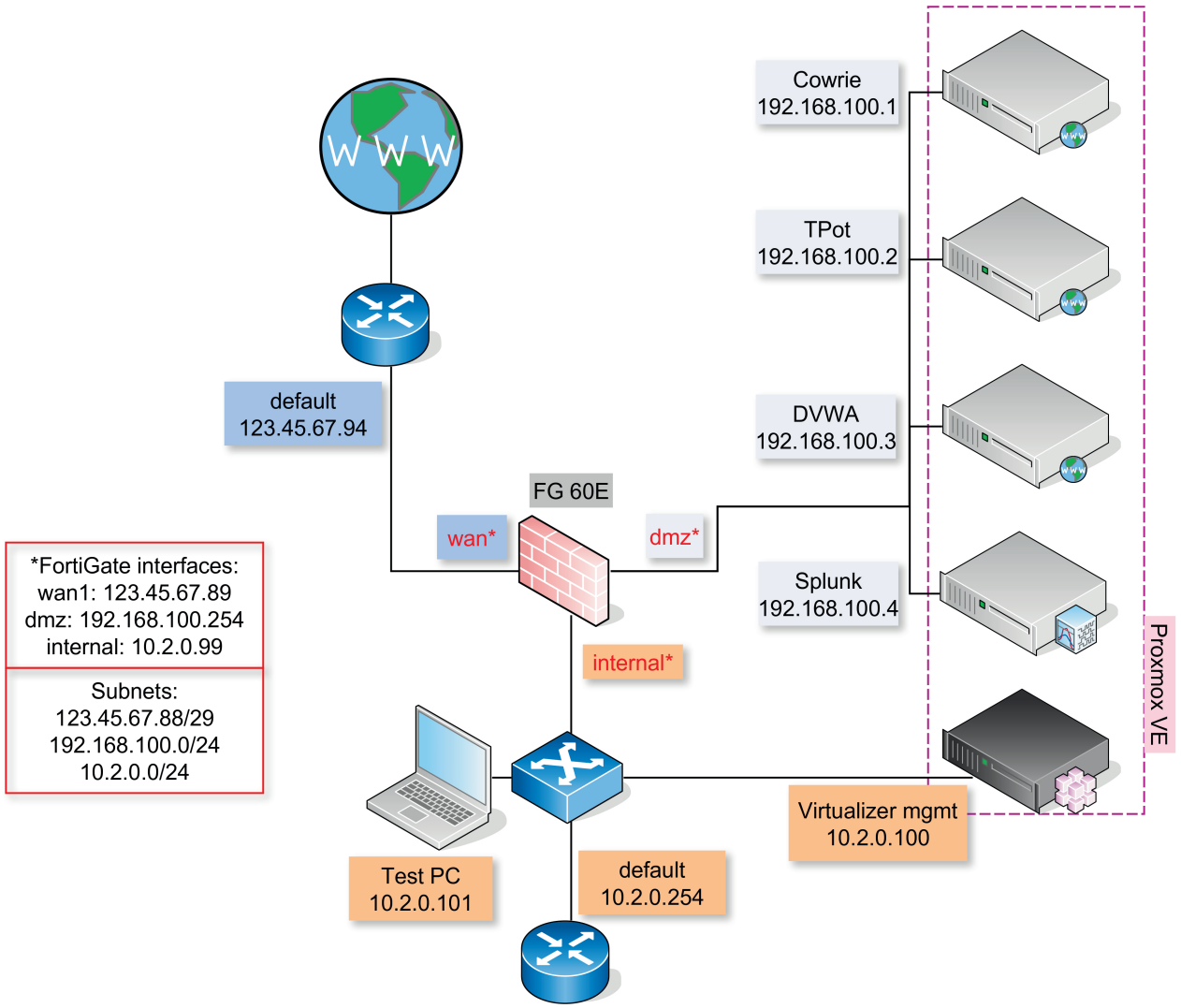

Figure 4.3: The honeypot network.

The final step was to enable log reporting from both the firewall and the machines to the Splunk one, finalizing the setup of the whole environment.

4.4 Data collection and first testing phase

The data collection phase lasted for a total of 158 hours and 45 minutes, and was divided into four sub-phases:

1. Phase **1**: the network was left exposed to the internet with no WAF protection and IPS used as an *Intrusion Detection System* (IDS);

2. Phase **2**: basic DoS protection was inserted and the network was targeted with aggressive port scans and Denial of Service attacks;

3. Phase **3**: the data gathered in Phase 1 and 2 was used for deploying an initial IPS and WAF profile and improving the DoS configuration precedingly added;

4. Phase **4**: additional data gathered in Phase 3 was used for refining the configuration.

4.4.1 Quantitative analysis

Table 4.2 shows the amount of inbound traffic recorded during the four phases as a median of packets per hour recorded during that phase. The data showed is a sum of all the incoming packets towards all three WAN-exposed machines, collected from the FortiGate's point of view. The duration of each phase is shown in table 4.1.

Phase	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**
Duration	119:00	2:45	19:30	17:30

Table 4.1: Duration of each phase in hours.

Phase	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**
Teardown traffic	655	111473	965	359
Allowed traffic	7468	22183	2918	840
Blocked traffic	0	1460	78	364
Total traffic	8122	135117	3961	1564

Table 4.2: Traffic from each phase in packets/hour.

The diagrams in figure 4.4 and figure 4.5 show the correlation between phases, both in relative and absolute form (with a logarithmic y-axis for better visualization). The colors represent three different categories of traffic:

- **Teardown** traffic: this category comprises `SYN`, `RST` and empty packets, such as those sent during a DoS or a port scan;

- **Allowed** traffic: legitimate traffic (that may or may not include a compromising payload);

- **Blocked** traffic: traffic flagged by the firewall as either malevolent (by the IPS or the WAF) or outside the rate limit (by the DoS protection).

Phase 1 traffic is characterized by a heterogenous and relatively high amount of traffic, comprising both legitimate and illegitimate attempts to access the network. The IPS sensors were enabled in monitoring mode during this phase, collecting initial insights on common exploits and attacks.

Phase 2, on the other hand, features an extremely high and artificial peak of traffic originating from hand-made Denial of Service attacks and *Nmap* port scans. The initial DoS protection profile, deployed only for avoiding network disruptions, was extremely indulging (with thresholds set in the thousands of packets per seconds) and struggled to limit the incoming attacks.

Starting from Phase 3, there was a sharp decline in the amount of traffic. With the IPS active dropping malevolent packets and releasing no response at all, most attackers tended to give up immediately after seeing their traffic cut short, while a small amount retried after several hours or days; in both cases, the total amount of packets processed fell one to two order of magnitude.

Finally, Phase 4 traffic suffered an even sharper decline, mainly due to the addition of aggressive IPS signatures for the T-Pot honeypot. In this phase, the amount of blocked traffic rose from 2% to 23%.

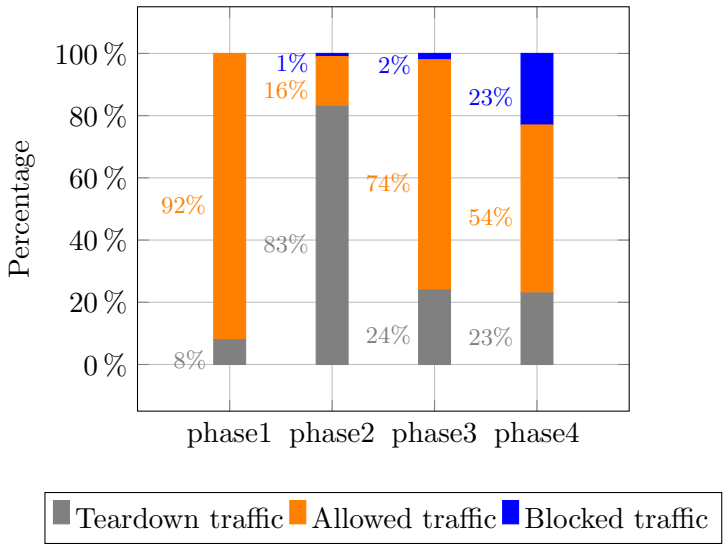

Figure 4.4: Normalized plot of traffic during the data collection phases.

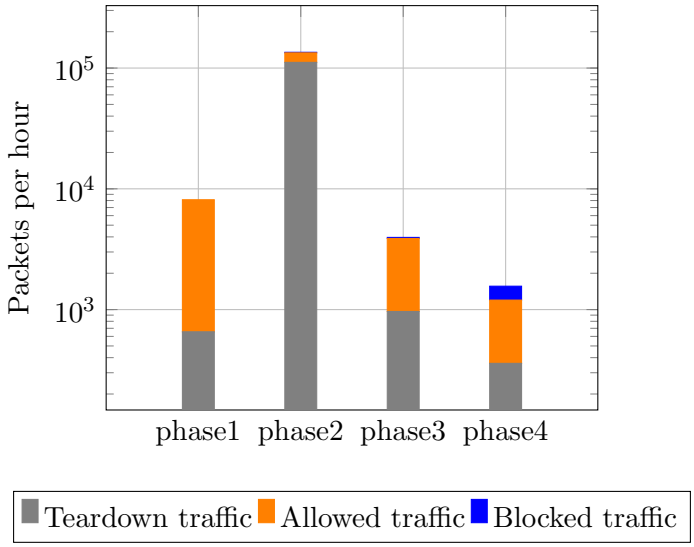

Figure 4.5: Semi-log plot of traffic during the data collection phases.

4.4.2 Qualitative analysis

During each of the four phases, full data for all traffic was recorded with special regards to source country, destination port and service. Extra flags were inserted by the 60E, highlighting packets that were inspected by security features such as the IPS or the WAF. In this case, additional information would be dumped to the Splunk machine, logging the type of the attack, the amount of retries, and more.

Firstly, over 80% of the incoming traffic was flagged as originating from either botnets or automated scripts retrying the same exploits over and over again. The list of the most common nations from where attacks originated is shown in figure 4.6).

Vietnam, India and Russian Federation were among the most reported countries with automated attacks. This comes as a no surprise, as these states are usually flagged as very dangerous by cyber security reports [23]. One notable exception from the list is the United States, falling short of the list with around 8000 attempts. Italy itself was even farther below, barely crossing the 5000 mark. This data aligns with the current distribution of botnets over the world, being heavily skewed towards "dangerous" countries [26].

Regarding targets, table 4.3 lists services (with the corresponding port) that were targeted the most during all phases. The services listed all use at least one TCP port. Services using two or more are marked with an asterisk $*$. Ports matching unknown services were aggregated into single entries,

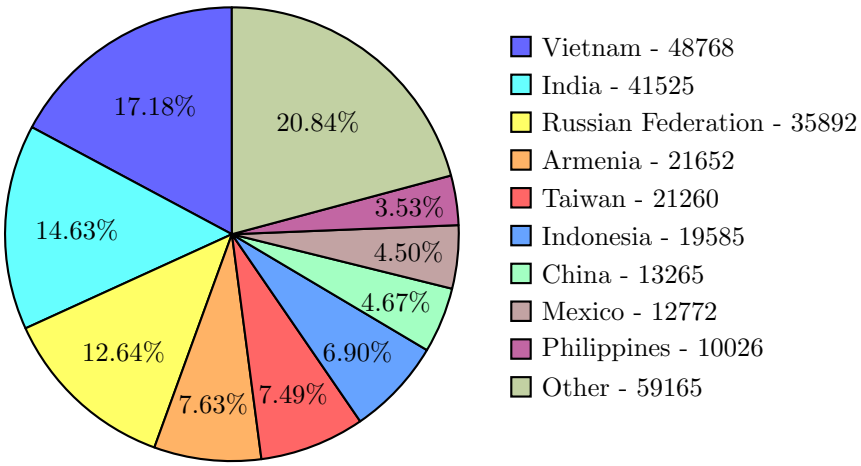

Figure 4.6: Pie chart of the most common attack sources with at least 10000 recorded attempts.

distinguishing them between well-known TCP ports, ephemeral TCP ports and UDP ports. The total packets column refers to the sum of packets that were flagged by the IPS or the WAF as malevolent, or by the DoS protection as over the threshold for that particular service.

Service	Port	Total packets
`MSSQL`	1433	434725
`SMBv1`	445	334411
Other TCP port (>1024)	-	155488
`RDP`	3911∗	113656
`VNC`	5900∗	27843
`Telnet`	23	22929
`SSH`	22	11798
Other TCP port (≤ 1023)	-	6946
`SMBv2`, `SMBv3`	445	5376
`HTTPS`	443∗	5207
`HTTP`	80∗	3554
`RTSP`	7070	3125
`SIP`	5060∗	2964
`VDOLIVE`	7000∗	2335
Other UDP port	-	2314
`SMTP`	25	1160

Table 4.3: Commonly targeted services with at least 1000 recorded attempts.

At the top of the table, the `MSSQL` and `SMBv1` services stably occupy the first two positions, receiving more than 67% of the total incoming traffic. This comes as no surprise, with the latter even being acknowledged as extremely vulnerable from Microsoft itself [20]. Other popular targets include the inherently unsafe `Telnet` and `HTTP`.

Surprisingly, the latter along with `HTTPS` were only modestly targeted in proportion to their enormous market share. This can be attributed to the inherent automated nature of attacks received. While botnets find it easy to overwhelm commonly vulnerable services, penetrating a well-protected web service usually requires human intervention both before and during the attack due to their heterogeneity. Indeed, most attack attempts directed at them were well-structured, not coming from botnets, and were usually cut short once detected by the WAF. The content of this type of attacks usually varied from simple SQL injection attempts to more tangled and elaborated XSS, often executed in batches. However, the attack variety, especially when compared with popular web application vulnerability lists such as the OWASP Top 10 [15], was found to be lacking.

Finally, some services such as Cowrie's SSH and T-Pot's SMTP actually virtualized a whole fake server and kept intruders entertained once breached, with their actions being logged. Inspecting the events, however, yielded little information apart from than logs of automated scripts and a couple commands before the attacker would quit the server. In contrast, others such as SMB.v1, RDP and VNC were no more than empty shells once contacted. Nonetheless, they spent the whole test phase being sieged just by keeping their TCP port open and advertising their presence. This confirms the theory that botnets usually prefer to focus their efforts on the easiest target available, with most of them being content just by gaining access to a system rather than diving deep into the machine.

Data was also collected on the incidence of common attacks. Table 4.4 presents popular attack attempts during the first three phases. This further confirm evidence that favorite attack surfaces remain those of insecure and out of date services, with attackers preferring to focus their efforts on using the same exploits over and over again rather than diversifying their efforts.

Attack	**Attempts**
Backdoor.DoublePulsar	255442
Android.ADB.Debug.Port.Remote.Access	25257
Telnet.Default.Credentials	16241
Mirai.Botnet	11190
SQL.Malicious.Statements	4318
Telnet.Remote.Root.Login	3769
Remote.CMD.Shell	600
MS.SQL.Server.Empty.Password	559
Sora.Botnet	522
UPnP.SSDP.M.Search.Anomaly	198
Telnet.Login.Brute.Force	182
MS.SMB.Server.SMB1.Trans2.Secondary.Handling.Code.Execution	164
MS.SMB.Server.Trans.Peeking.Data.Information.Disclosure	116

Table 4.4: Common attacks **during the first three phases**, with at least 100 recorded attempts.

The Backdoor.DoublePulsar attack precedingly described was by far the most popular, with 80% of breach attempts making use of this exploit. DoublePulsar is backdoor exploit often used as a vehicle for the WannaCry ransomware attack [32]). During these tests, a peculiarity was found: the exploit was run exactly 3980 times by 45 different attackers and botnets. Other popular automated exploits followed this pattern, such as the two MS.SMB.Server attacks. These two attacks exploit insufficient input validation on the server-side to execute malicious code. Other common attacks included botnet implanting attempts (e.g. Mirai.Botnet and Sora.Botnet) and credential brute forcing. The latter included the usage of a wide array of default credentials for both the Telnet and SSH services.

In order to get more meaningful data, between Phase 3 and Phase 4 some ports and services were shut down in order to better analyze the behavior of attackers. These include SMB.v1, MSSQL, RDP and VNC: the honeypot service was shut down and the corresponding port blocked by the firewall. While this is an optimal mitigation strategy, some network configurations requiring these services require a different approach. A good alternative can be the addition of each service's filter category to the IPS configuration, as in figure 4.2. Either way, outright blocking the most common threats is indispensable.

With trivial vulnerabilities being blocked in Phase 4, the honeypots collected valuable data over other lesser known attacks. Surprisingly, some attackers still dumbly retried popular attacks such as DoublePulsar over and over, even with the relative TCP port blocked by the firewall.

It was assumed that the public IPs used during the tests were flagged by bots and were still considered vulnerable. Thus, the tests were interrupted and retried several days later. This time, the amount of attacks towards closed ports decreased by 67%, with attackers refocusing their efforts on the services left open. Table 4.5 shows the most common attack attempts recorded in Phase 4.

The gathered data showed a wider variety of exploits being attempted. Almost all of them consisted in remote code execution attempts, usually as a result of poor coding on the application's side.

Attack	Attempts
VxWorks.WDB.Debug.Service.Version.Number.Scanner	1878
HTTP.URI.SQL.Injection	211
Netcore.Netis.Devices.Hardcoded.Password.Security.Bypass	137
PHP.Diescan	116
Joomla!.Core.Session.Remote.Code.Execution	105
vBulletin.Routestring.widgetConfig.Remote.Code.Execution	91
ThinkPHP.Controller.Parameter.Remote.Code.Execution	82
TCP.Split.Handshake	64
MS.Office.RTF.File.OLE.autolink.Code.Execution	60
Android.ADB.Debug.Port.Remote.Access	54
JAWS.DVR.CCTV.Shell.Unauthenticated.Command.Execution	41
NETGEAR.DGN1000.CGI.Unauthenticated.Remote.Code.Execution	36
ThinkPHP.Request.Method.Remote.Code.Execution	20
Java.Debug.Wire.Protocol.Insecure.Configuration	20
Goahead.Webserver.HTTP.Request.DOS	20

Table 4.5: Common attacks **during the fourth phase**, with at least 20 recorded attempts.

Unfortunately, most of them were pointless, due to the honeypots intentionally confusing attackers by posing as other operating systems during port scans. Nonetheless, some attack signatures were ultimately included in the configuration such as the `VxWorks` and `ThinkPHP` ones.

In conclusion, while the data gathered was useful, it was deemed somewhat generic and incomplete. This is true especially for web services, which overall received little attention during all four phases. Therefore, the data was integrated with manual, human-like attacks and a supplementary phase of data collection.

4.5 Second testing phase

The data driven approach using honeypots was successful for most types of services, but the obtained results were not considered satisfactory enough. Web services were lightly targeted, with the WAF being triggered an insignificant number of times during all four phases. Moreover, the reliability of the system with the newly created draft configuration was still to be tested.

To address these issues, a second round of tests was employed. Two different types of tests were executed during this round, one for web services and one for the others.

4.5.1 Web service testing

The first testing phase yielded a statistically irrelevant data set for web services, with both HTTP and HTTPS traffic being limited and attacks being extremely rare. The company's DMZ, however, was mostly composed of web servers essentially monopolizing the inbound traffic. The need for a more detailed round of tests was therefore evident.

The web service threat landscape is vast, with a great fragmentation of operating systems, application server solutions, programming language choices, and more. Launching an automated, large scale attack different from a DoS on various web servers is thus typically unfruitful, needing intervention from humans. From these considerations, an additional testing phase was planned, moving away from the data-driven techniques previously used in favor of a more structured, human-led approach.

Firstly, the DMZ was completely cut off from the WAN, allowing traffic only from a dedicated test machine's IP address. Secondly, a draft configuration for web servers was deployed:

- IPS: selected web-related signatures were enabled, including basic prevention for SQLi, XSS and directory traversal;
- WAF: initially enabled with both regular and extended protection for SQLi and XSS;

- DoS protection: thresholds were set using data from the production servers, roughly setting them to 1.3 times the daily peak traffic;
- TLS termination: enabled.

This configuration was deployed in protection of the Apache and T-Pot machines, with the Cowrie one being shut down. The Apache machine was reconfigured with a TLS certificate and optional HTTP to HTTPS redirection. Additionally, DVWA [8] was installed along with cloned versions of vulnerable, legacy and HTTP-only websites gathered from the production DMZ. On the other hand, T-Pot was set to manage only HTTP and HTTPS honeypots, which offered fake websites that changed every few hours from a rotation pool.

Work then focused on the creation of the tests themselves. They were further divided between **reliability tests** and **integration tests**.

Reliability tests consisted in the systematic execution of common vulnerability exploits using state of the art penetration testing tools, in order to verify the effectiveness of the configuration and eventually to check the necessity of modifications.

For testing SQL injection attacks, SQLMap [28] was initially used. SQLMap is an automated tool that targets a specific web page with a myriad of common SQL injection attempts while notifying the attacker straight from the command line in case of success. Further tests were conducted by hand, using example strings available online in threat lists [15]. Cross-Site Scripting tests followed roughly the same footprint, with ad-hoc JavaScript files being injected into the websites' forms.

The DVWA [8] installation proved ultimately invaluable during these tests. Usually used as an educational tool, it provides various tabs divided by attack (e.g. XSS, injection). In each tab, users could test their attacks without fears of destroying a real setup and having to recreate the machine.

Other tools used during the reliability tests include Metasploit [22], Nmap [14] and HULK [10]. Metasploit is a penetration testing framework automating the exploitation of vulnerabilities, widely used by security assessment teams. Nmap's usage was fundamentally unchanged and was used in conjunction with HULK for testing DoS protection thresholds.

The outcome of the tests was fairly positive, with the WAF reliably blocking all SQLi and XSS attacks. On the other hand, directory traversal attacks were blocked by the dedicated IPS signatures. Other types of attacks were instead blocked by more advanced WAF settings that were enabled step by step, such as *Generic Attacks* and *Bad Robot*. Finally, more advanced functions providing header-level inspection were tested but were not ultimately enabled because of excessive false positives being reported even during regular usage of the websites.

Integration tests, on the other hand, were conducted with the intention of verifying compatibility between each protected web service with the WAF, proxy mode of inspection, and TLS termination enabled. All of these features were selectively enabled and disabled each time, yielding consistent results but confirming previous fears that not all combinations would work correctly.

Indeed, enabling TLS termination and proxy mode inspection with any IPS configuration would trigger a bug, completely cutting off all traffic regardless of the source and triggering `RST` responses from the destination web server, no matter if they were legitimate or not. Due to lack of time, this bug was not able to be further investigated, but it was confirmed that past attempts of enabling the same configuration on the production firewall would yield the same result; moreover, the results would not differ even by changing the single firewall rule or the protected HTTP/HTTPS service. Full results of these integration tests are available at appendix B.

4.5.2 Non-web service testing

Tests for non-web services initially consisted in the change of honeypot IPs, with the attempt of resetting previous vulnerability flags. Surprisingly, even after two weeks and a full shutdown of the honeypots the botnets kept targeting the same IPs with the same common exploits. Secondly, a draft configuration was deployed:

- IPS: a policy was deployed featuring protection for attacks rated 3 out of 5 in Fortinet's vulnerability scale, then integrated for each service with standard filters for each type (e.g. Apache, HTTP, HTTPS, Linux) and key signatures collected in the first phase;

- WAF: disabled, as no web service was being tested;
- DoS protection: generic thresholds were set for TCP traffic and port scans, approximately 1.5 the amount of traffic recorded during the previous tests for each rule;
- TLS termination: disabled, as the totality of attacks encrypted with TLS was directed towards web services.

With the configuration in place, the system was reconnected to the WAN for an additional day. Unfortunately, no statistically significant data was obtained during this phase. However, the implemented signatures successfully deflected most attacks with no impact on performance. The DoS protection thresholds were tested with *Nmap*'s more aggressive scans, such as the `T5` which poses no limit on bandwidth usage [14]. This mode was used in conjunction with various types of scans offered by the tool, including SYN and TCP connection scans. The amount of traffic generated by these scans was more than enough to exceed the threshold and was rightfully blocked.

During most tests, the configuration received little change, with most edits falling into the removal of certain feature-braking signatures and the tweaking of DoS thresholds. Once all tests were completed, the configuration was deemed stable enough. It was therefore integrated with the web service one and marked as finalized.

Finally, a guideline document was created. This document featured an excerpt of the methodology used during the project and a summary of the all the results obtained. Instructions for restarting the data collection and testing phases were included, to be put in use by Trentino Digitale for future interventions on the network without restarting from scratch.

4.6 Deployment to production

Once the obtained configuration had been finalized and thoroughly tested, the remaining step was the deployment on the production environment and monitor the results.

Due to the COVID-19 pandemic, the work was however unexpectedly shut down and suspended for two weeks. During this time, the whole enterprise's workforce adopted smart working strategies for preventing the spread of the virus, and priorities shifted to tackling the emergency, such as the large-scale deployment of VPNs to workers.

As a consequence, a work of such magnitude requiring both time and effort was impossible to be carried over while remaining within the time constraints of the internship, so an alternative path was followed.

4.6.1 DNS redirection

To cope with the inability of editing the production firewall's configuration, a different strategy was conceived: the FortiGate 60E, used for the previous tests, would be repositioned and some selected DNS records would be updated to point to the 60E's position. Figure 4.7 shows an example of the procedure.

This configuration was applied to a few selected web servers, for the time strictly required for testing the availability, the reliability and stability of the system. While useful as a buffer solution, it could not in any way be compared to a full-scale deployment in the production firewall.

4.6.2 Deployment results

Once the DNS redirection had been set up, the configuration for HTTP and HTTPS servers, previously written down from the test environment, was re-imported with little to no modification and applied.

Unfortunately, the small amount of time available made available for testing meant that no statistically relevant analysis was able to be conducted; a test suite was therefore crafted following the footprint of reliability tests previously ran in subsection 4.5.1. This suite was integrated with snippets of past XSS and SQLi attacks, pulled from the company's SIEM and event logs. Additionally, strings vaguely resembling malevolent attacks were randomly inserted in the list to assess the capability of the system of avoiding *false positives*. Code snippet 4.1 features selected examples of attack strings used in the test suite, in the form of URL parameters.

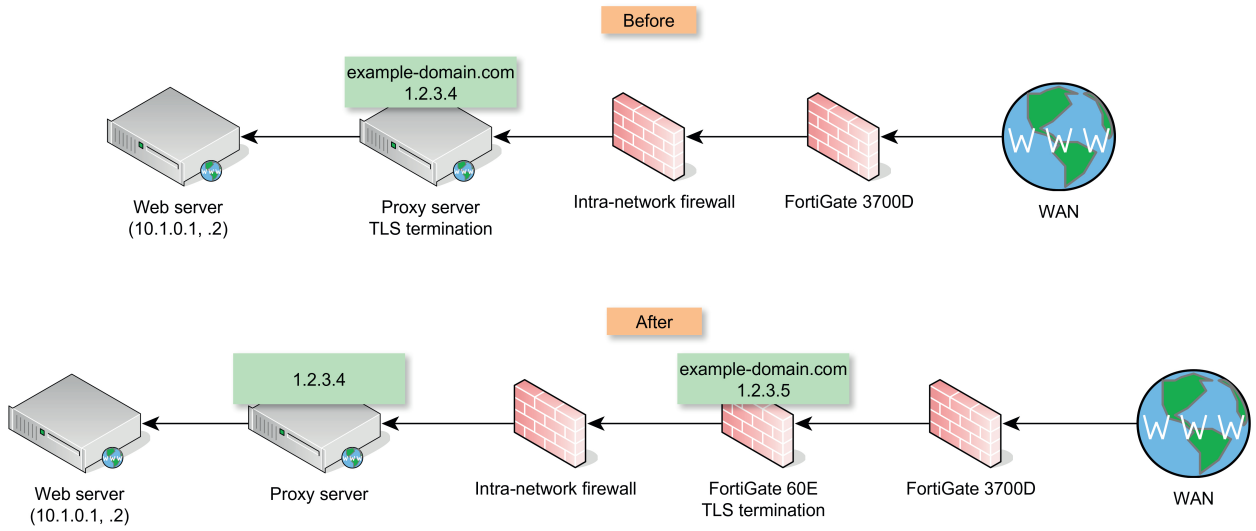

Figure 4.7: An example of DNS redirection employed for the final tests.

```
# Sample directory traversal attack.
?tags=../../../../../../../etc/passwd%00&groups=admin

# Sample XSS attack using JavaScript's alert function.
?tags=%22%3e%3cjavascript%3alert(String.fromCharCode(88%2c83%2c83))%3b%22%3e
&groups=admin

# Sample SQLi attack with URL encoding.
?SearchText=%20AND%205603%3DCONVERT%28INT%2C%28SELECT%20CHAR%28113%29%2BCHAR
%28122%29%2BCHAR%2898%29%2BCHAR%28113%29%2BCHAR%28113%29%2B%28SELECT%20%28CASE
[...]29%29%2BCHAR%28113%29%2B

#Sample SQLi attack with union statement.
?SearchText=%27%20UNION%20ALL%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL
%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--%20praG
```

Code 4.1: Sample attack strings used in the test suite

This test suite was deployed against the chosen web servers, with the 60E first employing the initial configuration and then the new one.

Firstly, the new IPS rules resulted roughly in a 35% increase in detected attacks over the previous configuration. The WAF successfully detected all relevant intrusion attempts when enabled, and no false positives were detected. The DoS protection thresholds were not tested.

Secondly, the performance of the system remained relatively unaffected. Even with the supplementary network routing required by the 60E's presence, measurements in terms of both load and latency showed little difference from before, with peaks of 200ms; it was estimated that by using the production firewall the extra latency would be cut at least in half.

The results of the limited deployment can be considered fairly positive. However, a full-scale deployment including all services in the DMZ and exposure to the WAN for several days remains mandatory. This allows a complete assessment of the capabilities and the limits of the final configuration.

5 Conclusion

This dissertation analyzed the state of art for threat protection systems and outlined a high-level methodology for deploying them in an enterprise environment while maximizing efficiency and security. The case study proposed a possible application of the steps required for generating a consistent, reliable and up-to-date solution for intricate situations, requiring little modification to the existing infrastructure and configuration.

In the first chapter, an overview of the landscape of threat protection systems for the enterprise was given. The terminology used throughout the dissertation was introduced, including details on the OSI stack and on components considered key to providing greater protection to a network: an IPS, a WAF, a DoS protection system and a TLS termination proxy.

The second chapter provided further information on such components, detailing their operation and the state of the art. Their capabilities were put into perspective, with each one's strengths and weaknesses being outlined.

In the third chapter, a methodology was crafted employing honeypots to collect data about incoming threats. The obtained data was subsequently employed for writing IPS and WAF policies. Built around the concept of lightness, it aimed to strike a balance between performance and protection while distinguishing between services. For each service type, a honeypot mimicking its likenesses was set up and used as a bait for attackers; data about both successful and unsuccessful breach attempts was then collected. Statistics and past attack data were used afterwards for defining a suitable configuration for all four components. This configuration was then progressively refined and rendered apt for the protection environment.

In the fourth chapter, a case study employing the aforementioned methodology was described. Born as an internship collaboration with Trentino Digitale (a public administration enterprise running the province's network), the work focused on the definition of IPS, WAF and DoS protection rules, their consequent validation and deployment on the company's DMZ. Data from the previous, incomplete configuration was mixed with valuable information incoming from a specially-crafted honeypot network, which left open for almost a week and exposed to the WAN. More testing phases and ad-hoc attacks followed suit, refining the obtained configuration even more.

However, the whole project was ultimately reshaped by the COVID-19 pandemic, which forced a temporary suspension of the internship and rendered the remaining time insufficient for carrying out the deployment to production suitably. An alternative approach was crafted, based on DNS redirection. The production equipment was not touched and the configuration was validated from the test environment itself. However, this approach was far from perfect and presented several weaknesses.

Firstly, the addition of an extra piece of hardware within the network clashed with the fundamental principles initially outlined, namely the immutability of the network. Even if a more high-end firewall appliance was to be installed in place of the testing 60E, spreading out rules throughout various firewalls would worsen maintainability and introduce additional risks in the form of rule clashing.

Secondly, the configuration was thoroughly tested only on HTTP/HTTPS services, leaving out highly vulnerable ones such as `SMBv1` and more prevalent ones such as `SSH`. Fully testing all the configurations obtained is key to avoid potential disasters during the final deployment: even if no errors were detected during the testing phase, the production environment is usually several orders of magnitude more chaotic and filled with threats, misuses and hundreds of potential points of failure.

The methodology described could be further explored by employing, for example, mirror images of actual services instead of honeypots that could further fool attackers and better reflect the actual nature of each service, or by completely automating the testing phase and avoiding time wasting on firewall maintenance. Of course, such an approach would require specific APIs by the vendor.

On the other hand, future work outlets for the case study should absolutely focus on the completion of the unfinished project. The first step will be the completion of the previously obtained configuration, followed by the carrying out of structured deployments to all of Trentino Digitale's network infrastructure. In case of significant delays within this process, the configuration will need to be further reviewed and integrated with fresh data coming from additional, up-to-date honeypot exposures.

Indeed, nowadays the cyber security landscape shifts at an unparalleled pace and new threats and vulnerabilities are discovered each day. This stands true especially for Trentino Digitale: as an AS owner and maintainer of a vast and diverse network, the threat of new and incoming attacks is always around the corner. Therefore, the configuration obtained should be maintained and updated often, both with the aforementioned methodology and with vendor-specific facilitations. Fortinet, for example, provides automatic category filter updates for its IPS system, enabling the automated addition of new signatures from FortiGuard's database directly into existing policies.

The testing honeypot network, meanwhile, remained in place even after the end of the internship with the firewall blocking all traffic and the hypervisor turned off. Surprisingly enough, the event logs show that even after several months the previously used IPs are still attracting random bots and attacks, at a sustained yet lower pace. This further summarizes the current landscape of cyber threats, nowadays mainly composed of botnets continuously scanning the whole internet and pinpointing targets for months and months once they have been tagged as vulnerable.

The aforementioned infrastructure could, in the future, hold an even greater potential. Possible opportunities could be either the relocation to a permanent location within the enterprise data center, or a total rebuilding from scratch incorporating more robust software and hardware solutions while maintaining the current skeleton. This could highly benefit the SOC's work by providing a more reliable and stable testing environment.

Long-term wise, the decommissioning of the intermediate firewalls, nowadays considered obsolete and redundant, can be fundamental within a vision comprising a centralized threat management platform and streamlining of the network for a quicker detection and disposal of incoming threats.

Bibliography

[1] *Proceedings of the 1986 IEEE Symposium on Security and Privacy.* IEEE Computer Society, April 1986. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/6234845/proceeding

[2] M. p.-n.-p. Alam, *Recent Developments in Computing and Its Applicationsn.* K International House, 2009.

[3] CertBot. (2020) Certbot documentation. Accessed 2020-03-11. [Online]. Available: https://certbot.eff.org/docs/

[4] Cowrie. (2020) Cowrie. GitHub. [Online]. Available: https://github.com/cowrie/cowrie

[5] X. de Carné de Carnavalet and M. Mannan, "Killed by proxy: Analyzing client-end tls interception software," 01 2016.

[6] C. Douligeris, *Network Security: Current Status and Future Directions*, July 2007.

[7] J. Ellingwood. (2016) How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 16.04. DigitalOcean. Accessed 2020-03-11. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-16-04

[8] Ethicalhack3r. (2020) DVWA. GitHub. Commit 720d1972e4d914c0df58d76b752b815fb99d3293. Accessed 2020-03-01. [Online]. Available: https://github.com/ethicalhack3r/DVWA

[9] Fortinet, "FortiGate 60E Series," Fortinet Inc., June 2020, model FGFWF-60E-DAT-R28-202005. Accessed 2020-06-14. [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiGate_FortiWiFi_60E_Series.pdf

[10] Grafov. (2020) Hulk. GitHub. [Online]. Available: https://github.com/grafov/hulk

[11] K. Juell and E. Heidi. (2018) How To Secure Apache with Let's Encrypt on Ubuntu 18.04. DigitalOcean. Accessed 2020-03-11. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-18-04

[12] R. Manisha and J. Nene, "A survey on latest dos attacks: Classificationand defense mechanisms," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, pp. 1847–1860, 2013.

[13] L. Masciullo, "Analisi di piattaforme per la gestione delle vulnerabilità: Uso del misp nella realtà del trentino," Bachelor's Degree thesis, Università degli Studi di Trento, July 2018.

[14] Nmap. (2020) Nmap. Accessed 2020-07-01. [Online]. Available: https://nmap.org/

[15] OWASP Foundation. (2020) OWASP Top 10. Accessed 2020-06-09. [Online]. Available: https://owasp.org/www-project-top-ten/

[16] A. Patel, Q. Qassim, and C. Wills, "Survey of intrusion detection and prevention systems," *Information Management and Computer Security*, vol. 18, October 2010.

[17] PCI Security Standards Council, *Information Supplement: Application Reviews and Web Application Firewalls Clarified*, 1st ed. PCI Security Standards Council, October 2008.

[18] C. Perrin. (2020) The cia triad. TechRepublic. Accessed 2020-06-18. [Online]. Available: https://www.techrepublic.com/blog/it-security/the-cia-triad/

[19] J. Pirc. (2017) The evolution of intrusion detection/prevention: Then, now and the future. SecureWorks. Accessed 2020-06-19. [Online]. Available: https://www.secureworks.com/blog/the-evolution-of-intrusion-detection-prevention

[20] N. Pyle. (2016) Stop using smb1. Microsoft, Inc. Accessed 2020-07-01. [Online]. Available: https://techcommunity.microsoft.com/t5/storage-at-microsoft/stop-using-smb1/ba-p/425858

[21] T. Radivilova, L. Kirichenko, D. Ageyev, M. Tawalbeh, and V. Bulakh, "Decrypting ssl/tls traffic for hidden threats detection," in *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2018, pp. 143–146.

[22] Rapid7. (2020) Metasploit framework. GitHub. Commit 2ab615fa431b235d82ff16905a9bfc040f891cf0. Accessed 2020-06-11. [Online]. Available: https://github.com/rapid7/metasploit-framework

[23] J. Robinson. (2020) Cyberwarfare statistics: A decade of geopolitical attacks. Privacy Affairs. Accessed 2020-07-01. [Online]. Available: https://www.privacyaffairs.com/geopolitical-attacks/

[24] M. Sher-Jan. (2017) From incident to discovery to breach notification: Average time frames. iApp. Accessed 2020-06-18. [Online]. Available: https://iapp.org/news/a/from-incident-to-discovery-to-breach-notification-average-timeframes/

[25] J. Snyder, "Evaluating enterprise ips: Seven key requirements," *Opuse One*, October 2012.

[26] Spamhaus. (2020) The world's worst botnet countries. Spamhaus. Accessed 2020-07-01. [Online]. Available: https://www.spamhaus.org/statistics/botnet-cc/

[27] Splunk Inc. (2020) Splunk. Accessed 2020-03-01. [Online]. Available: https://www.splunk.com/

[28] SQLMap Project. (2020) SQLMap. GitHub. Commit d0be782ece7f8e6ac54eb0dfc52a5499f4c35908. Accessed 2020-06-11. [Online]. Available: https://github.com/sqlmapproject/sqlmap

[29] A. S. Tanenbaum and D. Wetherall, *Computer Networks, 5th Edition*. Pearson, 2011. [Online]. Available: http://www.worldcat.org/oclc/698581231

[30] Telekom Security Development. (2020) T-Pot. GitHub. Commit f2abb1d1bdeafc702bb2f9590c1922532a1bf619. Accessed 2020-02-19. [Online]. Available: https://github.com/dtag-dev-sec/tpotce

[31] J. R. Vacca, *Computer and Information Security Handbook, Second Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[32] Wikipedia contributors. (2020) Wannacry ransomware attack. Wikipedia. Version 944184896. Accessed 2020-03-10. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

[33] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

Attachment A Honeypot configuration

This appendix provides code used during the initial configuration of the honeypot machines. All code sections have been reduced and rearranged, and are not to be intended as paste-ready.

A.1 Network configuration

All virtual machine shared the same network configuration mechanism, used via a netplan `config.yaml` file injected into the home folder at installation time. Each machine was assigned an IP in the `192.168.100.x/24` subnet, with DNS servers being set to the `8.8.8.8` DNS servers, maintained by Google. The choice was entirely arbitrary: eventual substitutions with local or other authoritative ones should come without consequences.

A.2 TLS certificate generation

For the Apache machine, two different TLS certificates were used: a self-signed one, and one obtained from *Let's Encrypt*, a free Certificate Authority used for automating the provisioning of TLS certificates.

The following code assumes that generation and configuration is made on an Apache machine. The complete documentation and code can be found on the DigitalOcean website [7] [11] and in the CertBot one [3].

We start by using an OpenSSL command for generating a self-signed certificate:

```
bash$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
      -keyout /etc/ssl/private/cert.key
      -out /etc/ssl/certs/cert.crt
```

Code A.1: Code for generating a self-signed certificate.

For a Let's Encrypt one, instead, Certbot must be downloaded and the following code must be executed:

```
bash$ sudo add-apt-repository ppa:certbot/certbot
bash$ sudo apt install python-certbot-apache

bash$ certbot certonly --manual --preferred-challenges dns
```

Code A.2: Let's Encrypt code for generating a certificate.

At this point, a domain for the TLS certificate will be requested and to confirm, a `TXT` record will have to be inserted in the authoritative DNS server for that domain.

```
_acme-challenge.dvwalab.sampledomain.it. 300 IN TXT "abcde...1234"
```

Code A.3: Sample string to be inserted in the DNS authoritative server.

Two `.pem` files will be obtained, which can either be used as they are or converted in the corresponding `.key` e `.crt` extensions with OpenSSL:

```
bash$ openssl x509 -outform der -in cert.pem -out /etc/ssl/certs/cert.crt
bash$ openssl rsa -in privkey.pem -out /etc/ssl/private/cert.key
```

Code A.4: OpenSSL code for converting certificates.

Finally, we configure the Apache server using the freshly obtained certificate. The code is available on the DigitalOcean website [7].

A.3 Software configuration

A.3.1 Cowrie

In addition to the following code snippets, the following steps must be made:

- If an OpenSSH server (highly suggested) or Telnet server is present in the system, it must be moved to a different, possibly ephemeral and high port, both to let the honeypot work on port `22` and `23` and to avoid accidental intrusions by attackers scanning other ports;
- The `cowrie` process must be authorized to listen on TCP ports `22` e `23`;
- The `etc/cowrie.cfg` e `etc/userdb.txt` files must be edited to suit the local network configuration: in the case study, Splunk log collection along with Telnet and SSH connections were enabled for the first file, while a list of common weak credentials pair was inserted in the second.

The complete documentation is available in the GitHub repository [4].

```
bash$ sudo apt-get install git python-virtualenv libssl-dev libffi-dev
bash$ sudo apt-get install build-essential libpython3-dev
bash$ sudo apt-get install python3-minimal authbind
bash$ sudo adduser --disabled-password cowrie

bash$ sudo su - cowrie # A different user is created
bash$ git clone http://github.com/cowrie/cowrie
bash$ cd cowrie
bash$ virtualenv --python=python3 cowrie-env

bash$ source cowrie-env/bin/activate
bash$ pip install --upgrade pip
bash$ pip install --upgrade -r requirements.txt

bash$ cp etc/cowrie.cfg.dist etc/cowrie.cfg
bash$ cp etc/userdb.example etc/userdb.txt

# System start, must be logged in as user cowrie and within the home folder
bash$ cowrie/bin/cowrie start
```

Code A.5: Full code for setting up the Cowrie machine.

A.3.2 DVWA

The complete documentation is available in the GitHub repository [8].

```
bash$ sudo apt-get -y install apache2 mysql-server php
bash$ sudo apt-get -y install php-mysqli php-gd libapache2-mod-php
bash$ git clone https://github.com/ethicalhack3r/DVWA

bash$ cd DVWA/config/
bash$ cp config.inc.php.dist config.inc.php
bash$ vim config.inc.php

# The following lines must be edited:
vim> $_DVWA['db_user'] = 'dvwa';
vim> $_DVWA['db_password'] = 'SuperSecretPassword99';
vim> $_DVWA['db_database'] = 'dvwa';

bash$ sudo mv DVWA /var/www/html/dvwa
bash$ sudo -i
bash$ mysql -D dvwa

# The following commands must be sent:
mysql> create database dvwa;
mysql> grant all on dvwa.* to dvwa@localhost
mysql> identified by 'SuperSecretPassword99';
mysql> flush privileges;

bash$ systemctl start mysql
```

Code A.6: Full code for setting up the DVWA machine.

A.3.3 T-Pot

The T-Pot machine, having been installed from a pre-made customized ISO, did not need particular modifications during setup and was deployed as is.

Attachment B Integration test results

This appendix summarizes results for integration tests, carried out in the second testing phase (section 4.5) of the case study.

All tests were run in an isolated environment with no interfering traffic and packet logging enabled. Initial IPS and WAF configurations deployed were obtained and refined in the previous testing phases. The IPS one featured basic protection with 3-star signatures along with `HTTP.Request.URI.Directory.Traversal` for directory traversal, `HTTP.URI.SQL.Injection` for SQL injection, among others. Some types of signatures, such as the former, appeared to be somewhat overlapping with existing WAF functionality and were superseded in case of conflicts.

It must be remembered that WAF could be enabled only if proxy mode inspection was too.

B.1 HTTP traffic

With all traffic being unencrypted, deep inspection was not required and enabling it would have yielded no effect.

IPS	Proxy	WAF	Result
×	×	—	No malevolent traffic is detected.
×	✓	×	No malevolent traffic is detected.
×	✓	✓	SQLi and XSS attacks are detected by the WAF. Navigation is otherwise regular.
✓	×	—	All IPS attacks are detected, including some basic SQLi attacks covered by signatures.
✓	✓	×	All IPS attacks are detected, including some basic SQLi attacks covered by signatures.
✓	✓	✓	Directory traversal attacks and others are detected by the IPS; SQLi and XSS attacks are detected by the WAF. Navigation is otherwise regular.

B.2 HTTPS traffic without deep inspection

Traffic was tested with both HTTPS and HTTP traffic enabled, with and without redirection, showing no difference with prior tests.

IPS	Proxy	WAF	Result
×	×	—	No malevolent traffic is detected.
×	✓	×	No malevolent traffic is detected.
×	✓	✓	No malevolent traffic is detected.
✓	×	—	No malevolent traffic is detected.
✓	✓	×	**The protected HTTPS service refuses any connection attempt by replying with `RST` packets. HTTP traffic remains unaffected.**
✓	✓	✓	No malevolent traffic is detected.

B.3 HTTPS traffic with deep inspection

It must be remembered that deep inspection works only if at least a security profile is enabled, such as IPS or WAF, that justifies the interruption of the TLS layer. Otherwise, deep inspection will automatically disable itself.

IPS	Proxy	WAF	Result
×	×	—	No malevolent traffic is detected.
×	✓	×	No malevolent traffic is detected.
×	✓	✓	SQLi and XSS attacks are detected by the WAF. Navigation is otherwise regular.
✓	×	—	All IPS attacks are detected, including some basic SQLi attacks covered by signatures.
✓	✓	×	**The protected HTTPS service refuses any connection attempt by replying with `RST` packets**. **HTTP traffic remains unaffected**. If the payload contains IPS-flagged traffic, it is still blocked before it even reaches the web server.
✓	✓	✓	Directory traversal attacks and others are detected by the IPS; SQLi and XSS attacks are detected by the WAF. Navigation is otherwise regular.