Titelblad



**Gegevensblad**

Studiegebied Industriële Wetenschappen en Technologie

|  |
| --- |
| **Stagiair**  Matthias Maes  **Opleiding**  Elektronica – ICT afstudeerrichting ICT  **Academiejaar**  2016/2017  **Stageperiode**  06/02/2017 - 12/05/2017  **Stagebegeleider**  Roel Van Steenberghe  **Stageplaats**  EY  De Kleetlaan 2 bus  1831 Diegem  België  **Mentor(en)**  Arvid Vermote, Eric Lembregts |

Summary

# Foreword

This thesis will form the conclusion to my bachelor degree of Electronics and ICT at Odisee Gent. During my education at Odisee, I have gained special interest in IT infrastructure and cybersecurity. After obtaining the “Certified Ethical Hacker” certificate (late 2016), the choice for a cybersecurity related subject seemed obvious.

I would like to thank EY and especially the cybersecurity team for granting me this opportunity. Working alongside these experienced and helpful colleagues created an educational, dynamic and motivating working environment. Special thanks to Arvid Vermote and Eric Lembregts (external mentors) for guiding me through my internship.

I would also like to thank the cybersecurity lecturers at Odisee for preparing me as best as possible for this internship. My thanks go out to Roel Van Steenberghe in particular for being my internal mentor but also for the assistance in obtaining the CEH certification that lead to this internship.

Finally, I would like to thank my family and friends for the support during this internship.

**Arvid Vermote**

* External mentor
* Manager
* Cybersecurity and privacy (EY)

**Eric Lembregts**

* External mentor
* Senior
* Cybersecurity and privacy (EY)

**Roel Van Steenberghe**

* Internal mentor
* Lecturer (Odisee)

# Table of contents

[Foreword 4](#_Toc481223015)

[Table of contents 5](#_Toc481223016)

[Glossary 7](#_Toc481223017)

[1 Presentation of the company 8](#_Toc481223018)

[1.1 History 8](#_Toc481223019)

[1.2 Structure 9](#_Toc481223020)

[1.2.1 Geographical 9](#_Toc481223021)

[1.2.2 Organizational 9](#_Toc481223022)

[1.2.3 Client sector 10](#_Toc481223023)

[1.2.4 Summary 10](#_Toc481223024)

[1.3 Big four 11](#_Toc481223025)

[1.4 Students 11](#_Toc481223026)

[2 Internship and thesis 12](#_Toc481223027)

[3 Action Plan 14](#_Toc481223028)

[4 Preliminary study 15](#_Toc481223029)

[4.1 Introduction to threats 15](#_Toc481223030)

[4.1.1 What is a threat 15](#_Toc481223031)

[4.1.2 Threat management 15](#_Toc481223032)

[4.1.3 Threat modeling 16](#_Toc481223033)

[4.2 Detecting threats 18](#_Toc481223034)

[4.2.1 Reputation-based 18](#_Toc481223035)

[4.2.2 Signature-based 18](#_Toc481223036)

[4.2.3 User behavior-based 19](#_Toc481223037)

[4.2.4 Heuristic-based 19](#_Toc481223038)

[4.3 Counteracting threats 20](#_Toc481223039)

[4.3.1 Firewall 20](#_Toc481223040)

[4.3.2 Intrusion detection system 21](#_Toc481223041)

[4.3.3 Intrusion prevention system 21](#_Toc481223042)

[4.3.4 Web Application Firewall 21](#_Toc481223043)

[4.4 Knowing the enemy 25](#_Toc481223044)

[4.4.1 OWASP 25](#_Toc481223045)

[4.4.2 Web attacks in numbers 26](#_Toc481223046)

[4.5 Proof of concept 27](#_Toc481223047)

[4.5.1 Python 27](#_Toc481223048)

[4.5.2 Logging web servers 28](#_Toc481223049)

[5 Practical execution 30](#_Toc481223050)

[5.1 Introduction 30](#_Toc481223051)

[5.2 General review 30](#_Toc481223052)

[5.2.1 Module: UNIFIER 31](#_Toc481223053)

[5.2.2 Module: PROFILER 32](#_Toc481223054)

[5.2.3 Module: FIREWALL 33](#_Toc481223055)

[5.3 Determining metrics 34](#_Toc481223056)

[5.3.1 Types of metric values 34](#_Toc481223057)

[5.3.2 Metric: TOTALCONNECTIONS 34](#_Toc481223058)

[5.3.3 Metric: LOCATION, 34](#_Toc481223059)

[5.3.4 Metric: DAY 34](#_Toc481223060)

[5.3.5 Metric: TIME, AGENT, STATUS, METHOD, REQUEST, LOGIN 35](#_Toc481223061)

[5.3.6 Metric: TIMELINE 35](#_Toc481223062)

[5.3.7 Metric: TIMESPENT, SIZE 35](#_Toc481223063)

[5.3.8 Metric: PARAM 35](#_Toc481223064)

[5.4 Detecting anomalies 36](#_Toc481223065)

[5.4.1 Severity 36](#_Toc481223066)

[5.4.2 Examples 36](#_Toc481223067)

[5.5 Code review 38](#_Toc481223068)

[5.5.1 Code: UNIFIER 38](#_Toc481223069)

[5.5.2 Code: PROFILERS 41](#_Toc481223070)

[5.5.3 Code: FIREWALL 48](#_Toc481223071)

[Conclusion 55](#_Toc481223072)

[Thesis 55](#_Toc481223073)

[Achievements 55](#_Toc481223074)

[Internship 55](#_Toc481223075)

[Working at EY 55](#_Toc481223076)

[Communication mentors 55](#_Toc481223077)

[References 56](#_Toc481223078)

# Glossary

**WAF** Web Application Firewall

**IDS** Intrusion Detection System

**IPS** Intrusion Prevention System

**SERT** Security Engineering Research Team

# http://www.ey.com/ecimages/EY.gifPresentation of the company

Figure 1: Logo EY (2017) [54]

EY (Ernst & Young) is a worldwide known accounting firm and is proud member of the “Big four”. EY is spread around the world with more than 700 offices in more than 150 countries offering jobs to more than 200.000 employees. The Belgian offices are located in **Diegem** (office where the internship took place), Ghent, Antwerp, and Bruges for a total of twelve different locations in Belgium. The 1335 Belgian employees are spread between these locations. [1]

## History

Throughout the years, EY was formed by merging with other organizations several times. The oldest originating partnership was founded in 1849 in England. In 1989, the fourth largest accountancy firm (Ernst & Whinney) merged with the fifth largest (Arthur Young) to create Ernst & Young. [2]



Figure 2: Logo Arthur Young [55]

Figure 3: Logo Ernst & Whinney [56]

At one point in time there were even plans to merge with KPMG (current member of the big four), but those plans were abandoned due to the problems with merging two very diverse companies and cultures.

Today EY is still looking for opportunities to further expand and improve the quality of its services. One of the fastest expanding sectors is the cybersecurity, especially with the recent announcement of the General Data Protection Regulation (GDPR)[[1]](#footnote-1) [3]. Organizations all over the EU are rushing to meet these regulations and ask companies like EY to guide them in this process.

## Structure

### Geographical

EY is geographically organized into four areas. Each of these areas are divided into multiple local areas. [2]

* **EMEIA (Europe, Middle East, India and Africa)**
  + Africa

Figure 4: EY’s company profile (EMEIA) [57]

* + **BeNe (Belgium and The Netherlands)**
  + CIS
  + CSE
  + FraMaLux
  + GSA
  + India
  + Ireland
  + Mediterranean
  + MENA
  + Nordics
  + UK
* Americas
* Asia-Pacific
* Japan

### Organizational

EY offers multiple services to companies worldwide. These services are grouped into four major service lines:

* **Tax**: provide information about (global) tax infrastructure.
* **Transaction advisory services**: provides information regarding raising, investing, preserving and optimizing the organizations capital.
* **Assurance**: provides general financial information.
* **Advisory**: provide clients with information regarding risk management and performance improvement.

The advisory service line facilitates the cybersecurity service in which this internship took place. The cybersecurity service is split into multiple services but members of the cybersecurity team can take assignments in any of these services.

Other services provided by the advisory branch are:

* IT Transformation
* Data Analytics
* Finance
* Supply Chain
* Customer & Strategy
* People Advisory Services
* Risk Management
* Internal Audit & Controls
* Risk Transformation
* **Cybersecurity**
  + Cyber Program Management
  + Cyber Threat Management
  + Identity & Access Management
  + Data Protection & Privacy
  + Business Resilience
  + Business Continuity
  + Incident Response (IR)
* IT Assurance

### Client sector

EY has clients from very divers sectors but the financial sector covers the most part of the clients.

The above-mentioned structure exists in duplicate. Once for clients in the financial sector (FSO) and once for clients of non-financial sectors (NON-FSO). This internship took place in the NON-FSO branch of EY.

### Summary

To get a better perspective where the internship took place within the complex organizational structure of EY the following flowchart was created.

## Big four

This is a group of the world largest professional service networks. PwC, DeLoitte and KPMG accompany EY in this group. The group originally consisted of eight members but the group was reduced to four members after several merges and scandals. [4]

EY differs itself from the other members of the big four by being the most globally managed firm of the four. EY has defined a set of policies to assure the consistency of services globally. This globalization is clearly visible in the daily working of the company. Desks are shared between colleagues of various nationalities; all employees get the chance to work together with colleagues from all around the world, which results in a very dynamic, coherent and educational work experience.

## Students

EY invests a lot into the potential of students, throughout the year there are numerous internships offered in all the different service lines. This is why EY is in the top of several rankings like: “best places to launch a career”, “Ideal employers”, “best place to work” etc.

# Internship and thesis

Today more and more services are offered as a web application and no longer as a software distribution (that has to be installed and configured by the user). Web applications are flexible, easier to develop, independent from the operating system of the user and can be accessed from everywhere. However, they have one major pitfall; they are very susceptible to cyberattacks. Attackers can perform a devastating attack (from everywhere) on a web application.

This is where the web application firewall comes in. This device can protect (multiple) web applications from attacks. It does this by recognizing specific patterns in requests that could be part of an attack. The firewall prevents these malicious requests from reaching the web application. For example: requests that contain JavaScript code in the query string can be blocked to prevent XSS, requests that contain the MySQL ‘UNION’ statement can be blocked to prevent SQL-Injection. In theory, this should mean that a web application that does not uses prepared statements is still protected from SQL-injection by the firewall.

The web application firewall looks like perfect solution for preventing web-based cyberattacks but it has its downsides. The biggest flaw is that they are difficult to configure in order to function correctly. What if a certain application requires JavaScript code in the query string or the MySQL UNION statement? These are normally blocked by the web application firewall and thus will prevent the web application from functioning correctly. This frustration among developers can lead to full out disabling the web application firewall making all the web applications vulnerable again. The cybersecurity team of EY has encountered this situation multiple times.

In order to eliminate the complex process of configuring the static rules, a web application firewall based on anomaly detection can be deployed. This type of firewall does not need any static configuration but will “learn” what type of requests are safe/legitimate and which are malicious. It does this by first observing legitimate requests (profiling of the application) and then later comparing incoming requests with the profile of legitimate requests to determine if the requests can be marked as legitimate. Malicious requests can be blocked because they show certain differences with the profiled (legitimate) requests. This kind of mindset will not only adapt itself when the web application changes but should also able to detect zero-day attacks.

During this internship, an application profiler and anomaly detection engine will be developed in order to proof the power of this type of web application firewall. EY would like to gather information about this kind of firewalls before organizations start to deploy these.

**Research question:** Is it possible to detect web attacks using an anomaly detection engine?

The following flowchart will illustrate the working of the proof of concept.



Figure 5: Flowchart WAF

# Action Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACTIEPLAN** | | | | |
| **Student**: Matthias Maes **Groep**: 3ICT3 | | | | |
| **Stageplaats**: EY Belgium | | | | |
| **Stageleider** **(interne promotor):** Roel Van Steenberghe | | | | |
| **Stagementor** **(externe promotor):** Arvid Vermote / Eric Lembregts | | | | |
| **Stap** | **Inhoud** | **Streef datum** | **Werkelijke**  **datum** | **Opvolging** |
| **1** | **Study of anomaly based detection engines, web application firewalls…** | **24/02** |  |  |
| **2** | **Development of profiler (process logs into profile file)** | **20/03** |  |  |
| **3** | **Development of detection engine** | **10/04** |  |  |
| **4** | **Development of real-time detection engine** | **08/05** |  |  |

# Preliminary study

## Introduction to threats

### What is a threat

A threat is considered everything capable of causing harm to a system or service. This can be the disclosure or modification of data, denial of service… Any action or event that deviates from the planned function of a system. Threats come in different shapes and sizes; it can go from a small malfunction of a program to an incident where the client database of a large organization is leaked. Because the amount of types of threats is so vast, a simple classification is needed. [5]

* **Deliberate:** These threats are actively trying to exploit a certain vulnerability in a system to disrupt its service. At any given moment in time, there are thousands of attacks ongoing worldwide. These attacks can vary from viruses written to disrupt a system, organized criminals that try to make a profit by stealing and selling data, spies that try to steal data from other companies or even other governments etc.
* **Accidental:** not every threat is the work of a criminal organization; a threat can also be an accident. These kind of threats can be caused by a malfunction of the system, misconfiguration of a system, power loss or accidental deletion and so on.
  + Environmental threats (floods, earthquake, lightning, tornado…) are also part of the accidental threats.

It is the goal of a cybersecurity team to counteract as much of these threats as possible. Accidental threats are most of the time countered using hardware resources. For example the build of a floodwall around a datacenter, the installation of an uninterruptible power supply to prevent accidental power loss, reinforcements against earthquakes… These countermeasures and type of threats go beyond the scope of this thesis so will no longer be considered.

The focus of this thesis will be at minimizing the deliberate threats. These threats are (most of the time) countered using software resources like firewalls, anti-virus, intrusion detection systems which will be discussed later on.

### Threat management

The cybersecurity team is constantly trying to improve the defenses of an organization. This process is the most effective knowing what threats are imminent. Without this knowledge, it would be like fighting an unknown enemy. The process of analyzing, researching and quantifying possible threats is called **threat management**. This process can have an active (actively researching possible attacks) or passive (analyzing attacks after they occurred) character.

Figure 5: [58]

Figure 6: Threat management [58]

#### Passive threat management: incident research

Passive threat management only kicks in when an incident or warning occurred on existent security infrastructure (firewalls, intrusion detection systems…). When an incident occurred the cybersecurity, team will assess and review the situation to prevent the incident from reoccurring.

#### Active threat management: threat hunting

Actively searching, detecting and isolating possible threats that can evade current security solutions. This is an iterative process because the pool of possible threats is constantly changing. This process needs to be executed continuously in order to be effective.

During threat hunting vast amount of network data must be analyzed. It would be quite impossible to analyze all this data manually so to automate a part of the analysis threat hunting software can be used. This software specializes in behavioral analytics.

* DomainTools: investigation DNS related cybercrime
* Exabeam, Sqrrl: User and Entity Behavior Analytics (UEBA)

Threat hunting seems to be very effective according to an investigation by the SANS institute. They reported (in 2016): 74% reduce in attack surface, 59% faster and more accurate threat responses, 52% more threats were discovered thanks to threat hunting.

### Threat modeling

Organizations participate in the threat modeling process to identify possible threats. During this process, the organization (the party that tries to improve their defenses) looks at their most valuable assets from the attacker’s point of view. This is the fastest way to find any possible loopholes in the organization defenses. Every threat model is different but they are all based on the same principles. [6] [7] [8] [9] [10] [11]

The basic threat model consists of four steps:

* Define objectives and scopes: without a set of business objectives, security criteria the threat model will lack foundation and is likely to be ineffective.
* Decompose the system: a set of possible targets.
* Identify threats: enumeration of the possible threats against each possible target.
* Prioritize the threats: possible threats are sorted from most harmful/likely to the most harmless/unlikely.

#### STRIDE

STRIDE is the successor of the DREAD threat model (DREAD is classified as outdated and will not be further discussed for this reason). This threat model divides threats into six categories. [12] [13]

* **Spoofing of user identity:** a process where a person or program presents itself as another one. This can be a malicious program hiding or acting as a legitimate program (Trojan), an attackers pretending to be someone else (social engineering), spoofing of IP address.
* **Tampering:** this kind of threats specialize in altering data in such a way to make it harmful to a system or the users of a system.
* **Repudiation:** a repudiation attack mainly targets the logging mechanism of a system. This means that the attacker has altered the system in a way that his malicious activities will not (or incorrectly) be logged. After a successful repudiation attack, none of the attacks can be linked with a certain attacker afterwards. Which makes counteracting the “hidden” attack and proving criminal activity next to impossible. [14]
* **Information disclosure:** this threat involves the exposure of information to individuals who are not supposed to have access to it. This can be intercepted network traffic (man in the middle), changing access permissions on files and directories. This type of threats are commonly used by whistleblowers to reveal unethical, malicious, fraudulent activities of a certain organization or government.
* **Denial of service:** one of the most popular, easiest and effective threats. These threats do not affect data in any way but it will deny any access to it. Denial of service attacks are widely used in attacks against web servers to prevent anyone accessing a certain website or web service. This threat is so popular because is quite difficult to defend against and even large organization can become a victim. GitHub, DynDNS, SpamHouse and the BBC have all been a victim of DDOS attacks.
* **Elevation of privilege:** in this type of threat an unprivileged user, gains privileged access to data or systems. With this privileged access, the attacker is capable of compromising or destroying the entire system. Most of the previous threats lead up to this attack (all defenses have been penetrated and the attackers becomes part of the trusted system itself). A few examples are: jail breaking Apple products, rooting Android products, dirty cow attack… [15] [16]

#### PASTA

The next threat model is fairly new but gaining popularity fast. The reason it is gaining this much popularity in such short amount of time is that it not only takes business objectives and technical requirements in to consideration but also business impact analysis and compliance requirements. This gives the modal a more dynamic approach. PASTA is the abbreviation for “Process for Attack Simulation and Threat Analysis”

Following are the seven steps in threat modeling according to the pasta model.

* **Define business and security objectives:** Capture requirements for the analysis and management of web based risks.
* **Define technical scope:** Defining the scope of technical assets/components for which threat enumeration will ensue.
* **Application decomposition:** Identify the application controls that protect high-risk web transactions.
* **Threat analysis:** Identifying and extracting threat information from sources of intelligence to learn about threat-attack scenarios used by web focused attack agents.
* **Vulnerability and weakness analysis:** Analyzing the weaknesses and vulnerabilities of web application security controls.
* **Attack modeling:** Attacks/Exploits enumeration and modeling.
* **Risk & impact analysis:** Impact Analysis, Residual Risk, and Countermeasure Development.

#### TRIKE

This threat model shows some similarities with the STRIDE model but the big difference is that it uses a risk-based approach. This model show great potential but is considered to be still in the experimental phase because of its difficulty to implement and poor documentation. However, since TRIKE is an open source project it is still undergoing some big changes and is starting to show some big potential. This model contains three distinct models: implementation, threat and risk.

Another big difference with other models is that TRIKE puts a great emphasis on communication between the organization and the stakeholders regarding threat management. The communication factor is something other models seem to neglect.

## Detecting threats

Before a certain threat can be blocked, it must be identified as a threat. There are several techniques for identifying threats. These principles are used by most security infrastructure (firewall, IPS, antivirus)

### Reputation-based

The most basic filtering is done reputational. Reputation-based filtering uses lists linking certain payloads or IP addresses to a reputation index. This is a representation of how trustworthy a payload (any data that is exchanged: code, files, e-mail…) or IP address is. [17] [18] [19] [20]

These scores can be determined locally or globally.

#### Local reputation

When a firewall successfully blocked an attack the source of the attack, being a file, IP address… will get a “bad reputation”. These reputations are stored in lists locally on the firewall. If a particular source with a bad reputation tries to connect in the future, the firewall will be extra careful handling this source. Every time the firewall detects malicious activity, the source will get even worse until the point where the firewall immediately blocks any connections or interactions with that source.

#### Global reputation

The lists of local scores can be shared to other firewalls over the network (other firewalls in the enterprise or ideally to firewalls all over the world). This makes that a source linked to malicious activity on a firewall will already get a bad reputation on another firewall on the other side of the world. The malicious source will thus be blocked on the firewall even before it ever connected to that particular firewall.

This is especially useful when blocking IP addresses. Addresses linked to botnets, tor nodes, known spammers and hackers, anonymous proxies are publicly available so firewalls can be extra careful handling traffic that originates from these sources. IP addresses can also be linked to a certain country with a certain reputation. For example, IP addresses from North Korea are more suspicious than an IP address from France.

### Signature-based

During signature-based detection, hashes of known malicious payloads are used to determine if a given payload is malicious. This offers a quick analysis of large files (the hashes have a fixed length independent of the file size), but is easily evaded using obfuscation or polymorphic code and does not protect against zero-day vulnerabilities.

At this point of time signature-based, detection is the most popular especially among antivirus software. These programs use immense databases containing signatures of malicious payloads. This technique is so popular because of its simplicity. On the downside, this is also the easiest technique to evade because even the smallest change in the code of a malware file will make it undetectable by this technique. [21]

### User behavior-based

User behavior analytics is the new big thing in the cybersecurity world. This technique will analyze human behavior and try to detect possible anomalies in this behavior. An anomaly is any behavior that deviates from the normal, expected behavior. This can be anything from excessive requests, larger payloads, unusual connection origin, strange input… User based analytics is so versatile it can even detect threats that look perfectly fine for any other device in the security infrastructure. [22] [23]

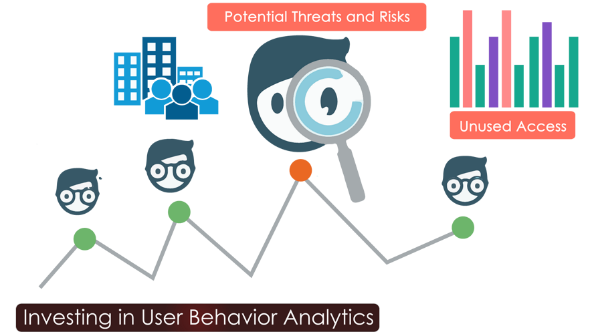
For example: Bob works in a local construction firm, he is in charge of paying all the suppliers and has full access to the firms bank account. Bob normally makes five transactions a day with an average value of €500. This is considered Bobs normal/expected behavior. Something would be not right if Bob suddenly makes a transaction worth €50.000. At this moment, the user behavior algorithms kick in, the transactions does not need to be blocked immediately (because it could still be possible this is a legitimate transaction for a big order) but the management will be notified immediately notified of this suspicious behavior. As it turns out Bob was about to leave the firm and tried to steal large amounts of money from the company. This kind of thread could only be detected by user behavior analytics because a firewall or intrusion prevention system does not make any difference between a transaction worth €50 or €50.000.

Figure 7: User Behavior Analytics [60]

### Heuristic-based

When a payload is not linked to a certain reputation index or signature, it is a total guess if the payload is malicious. The only way to find out if the payload is malicious is to execute it and analyze the effects on the system executing the payload. This is called sandboxing because the possible malicious file is executed in a test environment so any adversary effect will not have any impact on critical systems.

It can take a long time to complete the entire test but the results will be definitive. Because the test take a lot of time and resources, a deliberate decision has to be made which files need to be sandboxed. Typically, an .exe file carries more risk than a .png file.

## Counteracting threats

Organizations will try to defend themselves against threats using the above-mentioned techniques. These techniques will be implemented by cybersecurity infrastructure also sometimes referred to as the cyber defenses. These devices are continuously under development to keep up with the ever-changing pool of threats. In this chapter the four most used devices will be discussed, it is important to keep in mind that every device is specialized in counteracting a certain type of threat. The following is not a comparison of cyber defenses, rather an enumeration of the possible devices that can be used to reinforce the organizations defenses.

### Firewall

For counteracting the most basic and the more general threats, a firewall is used. This device filters the incoming and outgoing network traffic. Firewalls are implemented on borders of the network; this can be between internal networks but also be between the internal network and the external network. These devices categorize passing traffic using a set of rules; these rules are configured by the network administrator. The strength of the firewall depends on the strength of the rules so it is the responsibility of the network administrator to create the correct rules in order to maintain a safe environment. Throughout the years several generations of firewalls were developed. They started out quite simple but every time a new technology emerged, the firewalls had to change to maintain the level of security they provided before.

Figure 8: Firewall [61]

#### First generation: packet filters

In this quite basic system packets were filtered individually (no difference if the packet would be part of a stream or not). Every packet was filtered based on IP address, protocol and port number. The network administrator could specify which protocols were allowed using rules. If the network administrator decided to disallow SSH traffic, the firewall would block all traffic on port 22.

This generation became somewhat obsolete once other ports than the well-known ports (0 – 1023) were used and when threats over trusted protocols started appearing.

#### Second generation: Stateful filters

The following generation also functioned on the fourth layer of the OSI model (transport). Making it able of filtering traffic based on the state of the network connection. This means that the first packet of a stream will be checked more thoroughly. Once this packet is considered safe the rest of the connection stream will be inspected far less. This means connections could be inspected much quicker than before (for example not every packet of a large file transfer has to be inspected as thoroughly). [24]

#### Third generation: Application layers

This generation of firewalls inspects packets up to the application layer; this means that from now on traffic could be filtered based on the content. This was necessary because threads started appearing over normally trusted protocols (HTTP). These firewalls are capable of only filtering out the packets with malicious content so the protocol is still available for legitimate use.

Network connections to malicious websites or the transfer of virus-infected files could now be filtered out by the firewall. Until the birth of TLS/SSL, these encryption protocols were designed to encrypt network traffic in order to protect it from being monitored by attackers. This is one of the best security measurements but it has one big flaw. The content of the packets is encrypted so firewalls are no longer able to filter packets based on their content. Firewalls needed a new feature and this is where “Next-Generation Firewalls” come in. This type of firewall is able to decrypt network traffic to determine whether it is compliant to the rules. [25]

This was the latest big development in firewall technology.

### Intrusion detection system

These devices specializes in detecting possible network intrusions. They mostly use signature and anomaly-based detection techniques. When a possible intrusion is detected, the administrator will be notified (note that this system cannot undertake any action in preventing the intrusion it can only detect the intrusion). It is usually used to either strengthen the defenses in the future or gather evidence against a certain attacker. This system is a complete passive system that monitors the traffic and reports on possible threats. These devices are most of the time implemented offline, on a span port. This kind of ports duplicate all their traffic to another port, in this way the attacker will have no clue that he is being monitored. [26]

### Intrusion prevention system

An IPS is the active variant of an IDS. This typically sits right behind the firewall (inline) and can undertake certain actions to prevent intrusion. When an intrusion is detected there are several actions an IPS can undertake such as: reset, drop, block… [27] [28]

The IDS/IPS are not perfect as they can still be evaded in several ways:

* Fragmentation: by splitting the attack in an excessive amount of packets so that the IDS/IPS cannot reassemble the stream in time to detect the underlying attack.
* Encrypting and tunneling: an attacker can encrypt the packets and send them over a secure tunnel so the IDS/IPS cannot interpret the content of the packets.
* Insertion: the attacker inserts meaningless data in the stream that does not influence the payload (malicious) function.

### Web Application Firewall

When all previous defenses have failed at stopping the attack there is the last line of defense, the web application firewall.

#### Description

A web application firewall is an application, server plug-in or cloud based service that filters, monitors, blocks HTTP traffic to and from a web application. The WAF works at the application layer to prevent application layer attacks (SQL-Injection, XSS, session hijacking…), whereas the IDS/IPS works at layer 3 of the OSI model (mostly counteract network breaching attacks). [29] [30] [31] [32] [33] [34]

#### History

Web application firewalls were developed in the nineties in order to fill in the loopholes in the security measurements that were available at that time. These firewalls were only able to prevent attacks that used uncommon protocols/ports. Application layer attacks are mostly performed over the HTTP protocol. The problem with a stateless firewall is that this kind of firewall does not make any difference between an HTTP packet which will result in a SQL-injection or a packet that makes a legitimate request to the database. This is where the web application firewall comes in.

When the PCI Security Standard Protocol was formed and published, the PCI DSS (Payment Card Industry Data Security Standard) web application firewalls really gained attention because this standard mandates the use of a web application firewall (or an extensive code review). [35]

#### Working

The web application firewall works like an advanced IPS. The big difference between those two is that a WAF typically sits right in front of a web application (as a proxy). This means that it just has to analyze the traffic destined for that web application. A normal IPS has to analyze all the network traffic so it has only time to do a basic analysis. A WAF only has to protect a certain application so it can really specialize in the threats facing that type of application.

#### Implementation

A web application firewall will always be deployed right in front of the application it has to defend (web server, database) but there is still the choice between a hardware, software or cloud implementation.

##### Software

Implementing a software based web application filter is the most preferred option for small businesses. Software products are easier to install, update and maintain. It is also cheaper (most of the time) but it can lack some functionality or performance (performance depends on host performance).

There are even some open source web application firewalls available. Following is a comparison of the five most popular open source WAFs. [36]

###### ModSecurity

This is one of the most popular web application firewalls. It filters (allow/block traffic), monitors and logs based on “Sec Rules” (these rules are based on the core rule set of OWASP) in real-time. The engine can function as reverse proxy (a separate security layer, between external connection and web server) or can be embedded (apache module, the best option to implement in already existing architecture) within the server itself. ModSecurity uses several techniques (virtual patching, IP reputation, URL encoding) to prevent application layer attacks like XSS, Trojans, information leakage… [37]

###### IronBee

IronBee is a security framework to build your own web application firewall. This is not a finished product rather a framework to create your own tailored web application firewall. The main goals are to be minimalistic, provide simple API, modularity and allow interaction with external systems. Rules are written in LUA and come in three different types: basic matching rules, stream matching rules and external rules. [38]

###### NAXSI

This is a quite basic but very effective working web application firewall, it specializes in counteracting XSS and SQL-Injection and it only filters PUT and GET packets. The firewall uses the implicit deny all policy, so adding rules will add exceptions that are accepted. NAXSI is only available for nginx (NAXSI stands for Nginx against XSS and SQL injection).

It distinguishes itself from the other firewalls by not relying on signatures but rather the behavior of the payload. It also comes with its built in tool to generate your own custom rules. [39]

###### WebKnight

WebKnight is designed for Microsoft IIS, this web application firewall also provides protection against brute-force attack (this is the only web application firewall in this list that provides any protection against this kind of attack). [29]

###### Shadow Daemon

The last firewall aims to be the easiest of use. It still offers protection against the common attacks like the above firewalls but also against backdoor access.

##### Hardware

It is also possible to implement the web application firewall as an appliance (hardware component). This is the preferred option for larger organizations because appliance products tend to have a better performance to handle larger amounts of traffic. The performance of an appliance WAF is measured in throughput (volume of data that can pass through the device, this is important because all the traffic must go through the WAF before reaching the requested web service. This is the **network performance** of the web application firewall). Transactions per second (Amount of HTTP and SSL transaction that can be handled, this is the **processing performance** of the web application firewall). [40]

###### Barracuda

Probably the best-known web application firewall appliance is the one from Barracuda. At this moment, they sell five different models. Ranging from 25Mpbs to 4 Gpbs throughput, up to 50.000 SSL transactions with the price ranging from €6000 to €45000. They support HTTP(S), FTP(S), XML, IPv4 and IPv6. Barracuda assures protection against SQL-Injection, XSS, Cookie/form tampering and DDoS.

The barracuda WAFs also have some features to prevent (counteract before they occur) some attacks like website cloaking (Strips identifying banners and version numbers from web server software to defeat server fingerprinting attacks), data theft protection (deep inspects all server responses to prevent leakage of sensitive information using provided default patterns) and much more. [41] [42]

###### Netscaler

Another popular appliance is the Netscaler from Citrix. They sell models that have a throughput up to 44Gbps and can process 60000 SSL transactions per seconds. The big difference with previous mentioned appliances is that the Netscaler is modular (hence the name). The base models can be upgraded to improve their performance. [43]

###### SecureSphere

The last appliance that will be discussed is the one from Imperva, the SecureSphere. They share about the same performance as the web application firewalls from barracuda but are more oriented towards cloud integration. [44]

##### Cloud

If appliance performance is needed but installing one is not an option (cost, experience of staff…) a cloud-based solution can be interesting. Web traffic is diverted to specialized services where the traffic is analyzed before being redirected to the actual web application. An extra advantage is that only legitimate traffic is forward to the application, which means that the web application will not be flooded with malicious traffic (which is especially useful when the network performance of the application is limited). The three most popular services are Incapsula, CloudFlare, SUCURI. They will not be individually discussed because they all offer about the same features (DDoS, SQL-Injection, XSS, SPAM, login protection). [45]

## Knowing the enemy

Fighting an unknown enemy can be though so it can be interesting to look at today most imminent threats. The focus will be on the web application attack because this is the focus of this thesis.

### OWASP

#### Introduction to OWASP

The Open Web Application Security Group is an online (non-profit) organization that enables other organizations to conceive, develop and maintain applications that can be trusted. Everyone can post articles about newly discovered attacks or ways to mitigate them. OWASP is focused on web applications and is best known for their top ten Application Security Risks. This top ten is made up by data submitted by organizations word-wide. The most recent one was released in 2013 but later in 2017, a new top ten will be released. [46]

Figure 9: Logo OWASP [59]

#### Top Ten

1. **Injection:** this attack occurs when potential malicious data (user input) is used as part of a command or query. The attacker can use this to execute undesirable commands that can tamper, destroy or reveal data. These attacks are easy to execute and can have devastating results.
2. **Broken authentication and session management:** Incorrect implementation of functions related to authentication and session management. This can allow attackers to compromise passwords, keys or session tokens. (authentication tokens or session id in URL)
3. **Cross-site scripting (XSS):** This flaw allows attackers to execute scripts in the victim browser, which can lead to defaced websites, unwanted redirection… This attack shares some characteristics with injection attacks the big difference is that XSS is client side and injection attacks are server side.
4. **Insecure direct object references:** When an object reference is exposed (in the URL for example), the attacker can tamper this reference to access unauthorized data.
5. **Security misconfiguration:** Many applications can be very secure but are misconfigured or outdated which makes them vulnerable. This can be because the developers are misinformed, or the security measurements make the application inconvenient.
6. **Sensitive data exposure:** Sensitive data like credit cards, authentication credentials should be extra-protected (encryption) otherwise an attacker can have easy access to this data.
7. **Missing function level access control:** Attackers can forge request in order to get access to functionality without proper authentication.
8. **Cross-Site Request Forgery (CSRF):** Force a user to send forged HTTP requests. This can be a request for authentication token, cookies… The vulnerable application will think that the requests are legitimate.
9. **Using components with known vulnerabilities:** Some libraries, modules or frameworks have known vulnerabilities and are run with full privileges most of the time. When this software is used, their vulnerabilities can be exploited to compromise the server or gain access to data.
10. **Invalidated redirects and forwards:** Users are frequently forwarded and redirected to other pages, when these are not validated an attacker can use this to forward users to malicious websites.

### Web attacks in numbers

#### Attack motivations

What are the motives behind all these attacks? Are all attacks executed by federal spies to gather intel from foreign governments? Are all attacks part of a large-scale cyberwar operations? In February 2017 only 5.3% of the attacks are linked to cyber warfare and only 22.4% are linked to espionage (not only by certain governments but also between competing organizations). The major motive behind cyberattacks is just plain simple cybercrime. This can be individuals or small groups that try to make a profit by performing illegal actions (just like any other form of crime). Attackers try to make a profit by demanding a ransom for preventing data leakage, stopping a DDOS, decryption of data etc. The rest of the cyberattacks are linked to hacktivism, these are attacks performed by organizations like Anonymous. [47]

#### Attack targets

In the early days of cybercrime, 50% of the targets were large enterprises (> 2500 employees). This trend is radically changing. As of today 43% of the targets are small businesses (< 250 employees), and only 35% are large enterprises. These small businesses are less experienced with cybersecurity and incident response so they are easy targets. [48]

#### Attack types

In recent years, the web-based attacks have gained in popularity. This has multiple reasons, web applications are reachable from everywhere so they do not require on premise access, web based attacks can be easy to execute but can have devastating results never the less.

Figure 10: Chart cyber-attack types [47]

The SERT (Security Engineering Research Team) has determined that in 2016, 57% of all cyberattacks were web application attacks. These are attacks target at web application using techniques like SQLi, XSS, DDOS… The malware category does not (directly) attacks web services but is spread over the web (drive-by downloads, e-mail attachment) and is responsible for 19% of the attacks. Another big issue are the “unknown” attacks these are attacks where a system breach or data loss has been detected but it is still not clear how the attacker managed to execute the attack. [49]

## Proof of concept

### Python

The proof of concept firewall needs to be scripted in a language that supports multiple operating systems, heavy workloads and must have the ability to interact with network packets.

Python was launched early nineties by Guido van Rossum; the language was originally designed to be used by mathematicians and was based on BASIC. Python is specialized in processing large amounts of data and execute complex calculations.

#### Indentation

The big difference with other languages is that structure is determined by indentation. For example when a for loop is created in Java the loop statements are encapsulated with curly brackets, in Python the statements are indented instead of using curly brackets.

1. // For-loop in Java
2. **for**(**int** i=1; i<11; i++){
3. System.out.println(i);
4. }
5. # For-loop in Python
6. **for** x **in** xrange(1,10):
7. **print**(x)

#### Duck-typing

Python also implement duck typing of variables. This means that the type of a variable does not have to be defined. This makes for quick coding but can make debugging a bit more complex.

#### Extra features

Garbage collection and operating system independent runtime environment are also features of Python but the most loved characteristic is the gigantic package library. The packages can be installed easily using PIP. The use of libraries can save the developer hours of coding.

#### Different versions

Python comes in two versions: Python3.x and Python2.x. Although they appear to be very similar, they are not compatible. The organization behind python tries to push the community over to the 3.x version but many developers prefer to stay at the 2.x version for reasons that fall outside the scope of this research. Python can also be compiled into a standalone executable, which even eliminates the requirement for an interpreter. [52]

#### Different implementations

Python has different implementations; the most popular implementation is CPython. This implementation compiles Python code into byte code (.pyc files). This is the original implementation of python and is called CPython in order to prevent confusion with other implementations. The alternatives to CPython are Jython (Java), IronPython (C#), PyPy (RPython).

#### Conclusion

The following is an overview with the main arguments why Python will be used to create this proof of concept.

* Designed to handle big loads an complex calculations
* Operating system independent
* Ability to interact with network packets
* Very popular language in the cyber security community
* Preferred language by external mentors

### Logging web servers

Logs can give feedback about the activity and performance of the web server; they are generally split into access logs and error logs. Access logs record all the requests that are made by clients to the web server; this will be the most interesting log for the profiling engine. Each line in this log contains information about a request made by a client (IP address). Error logs record all the errors thrown by the web server and can be interesting for the administrator but are not that interesting for the profiler. The following are log entries examples, one from an access log and one from an error log.

*Example access.log entry:*

213.211.143.24 - - [14/Dec/2011:21:21:40 +0100] "GET /misc/favicon.ico HTTP/1.1" 200 5727 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.121 Safari/535.2"

*Example error.log entry:*

[Wed Dec 14 21:36:54 2011] [error] [client 213.211.143.24] File does not exist: /home/catapa/public/nl, referer: http://test.catapa.be/index.php

#### Log format

Access logs have two standard formats the common and the combined format but these can be altered to log other data. Logs formats are defined in the configuration file of the web server if the default log formats are used those lines would look like this:

**LogFormat** "%h %l %u %t \"%r\" %>s %b" common

**LogFormat** "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined

Each combination of a percent symbol and character will log certain data of the requests; following is a short summary of the used formats:

|  |  |
| --- | --- |
| Format String | Description |
| %h | IP address of client |
| %l | Remote log name |
| %u | Remote user |
| %t | Time the request was received |
| \”%r\” | First line of request (between quotes) |
| %>s | Status of the request |
| %b | Size of response in bytes |
| \”%{Referer}i\” | Logs referrer of request (between quotes) |
| \”%{User-agent}i\” | Logs user agent client (between quotes) |

#### Interesting values

The referrer and user-agent are the two most interesting values and deserve some extra attention.

* The user-agent: this is the software used to access the web application. When fetching new mail this will be the mailing program that is being used, when browsing the web this will the web browser that is being used etc.
* The referrer: this is the url associated with the request. A web page rarely consists of one HTTP request but rather one for images, one for style sheets etc. For example, surfing to index.html will result into three HTTP request: /index.html, /img/logo.gif and /css/styles.css. The referrer for those three resource requests will be index.html. The referrer can be used to group all de resource request originating from the same url.

#### Difference in logs between web servers

There are several web servers available; the most popular are Apache and Nginx. Luckily, they both use the same format for their access logs so the same code can be used for both web servers.

Another popular web server is IIS from Microsoft, this uses a complete different log format so a different script will be needed to profile IIS logs. [53]

# Practical execution

## Introduction

This chapter will cover the documentation concerning the development process of the proof of concept. The documentation will be structured according to the logical structure of the final release of the engine. This will not be the same order in which the engine was developed.

The documentation will start with a review of the overall structure and working of the engine. After the general overview, the modules will be reviewed in detail.

## General review

The following flowchart illustrates the general structure of the engine. The engine is intentionally split into multiple modules. This makes for a more organized, modular and performant environment. If a certain module does not fit the needs of a particular end-user, that specific module can be altered without disturbing the functioning of the other modules. For example if a user uses an alternate logging format, only the UNIFIER module will have to be altered.

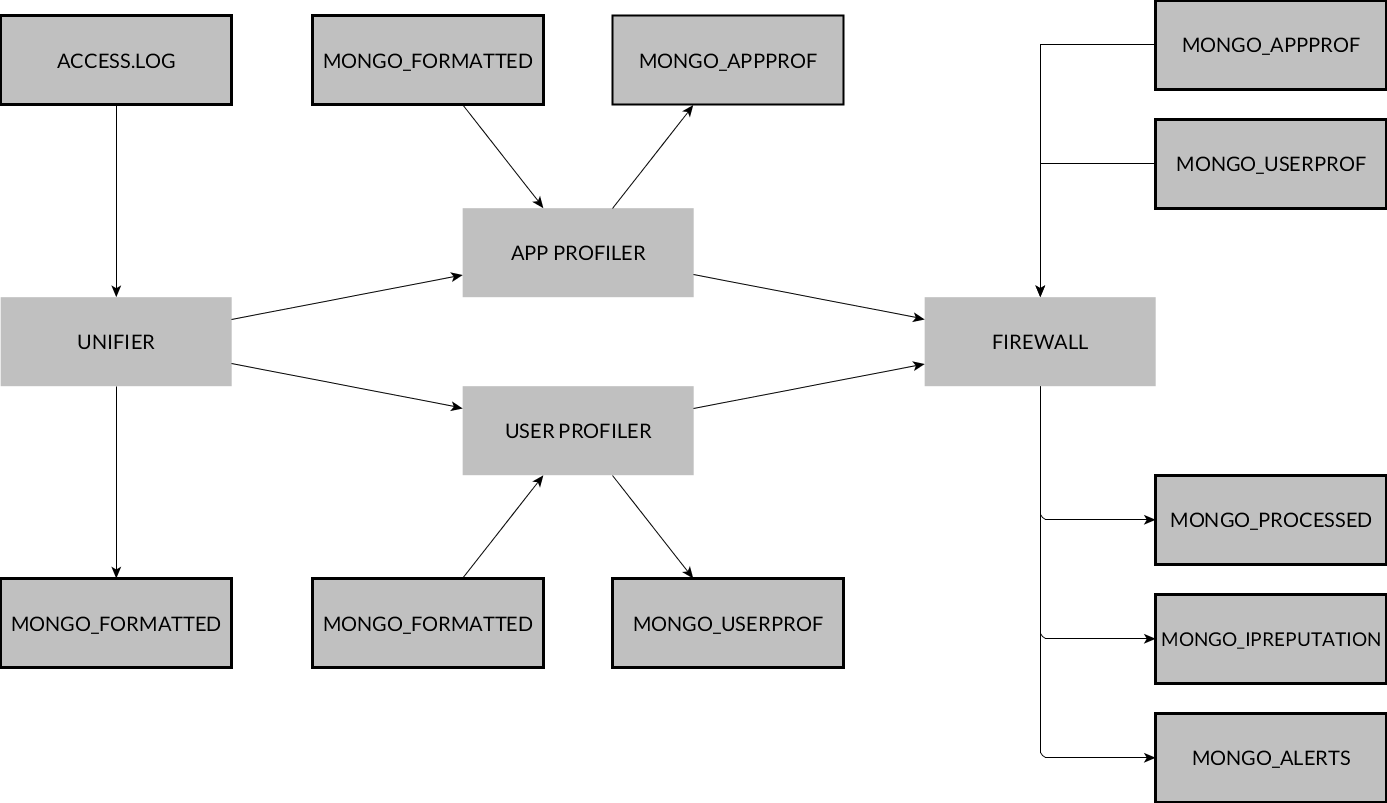


Figure 11: General overview engine

### Module: UNIFIER

The unifier module solely reads the lines in the access.log file and stores them in a MongoDB. The data is stored in the database conform a standardized format. Different log formatting will thus result in the same data structure in the database. The following illustrates the generalized data structure that is used to store the data in the database.

The index field is used to maintain the correct chronological order after processing the access.log multithreaded and to support sorting of the requests. At this point, there is no further interpretation of the data; the line in from the access.log was split into different fields. The requestURL and method are derived from the first line of the request. The other field are directly linked to a parameter in the log file (ex: ip -> %a, size -> %b, code -> %>s…).

The unifier currently supports the combined log format from apache and nginx. The unifier can be altered to support other log formats but in order to support all metrics the log format must include following parameters: user IP, HTTP status, HTTP method, size of the response, timestamp (date and time), first line of the request and the referrer URL.

The unifier reads the access.log line by line and stores them temporarily in a list (stored in RAM). This list is then passed to multiple workers to be processed. This ensures faster processing and limits memory usage.

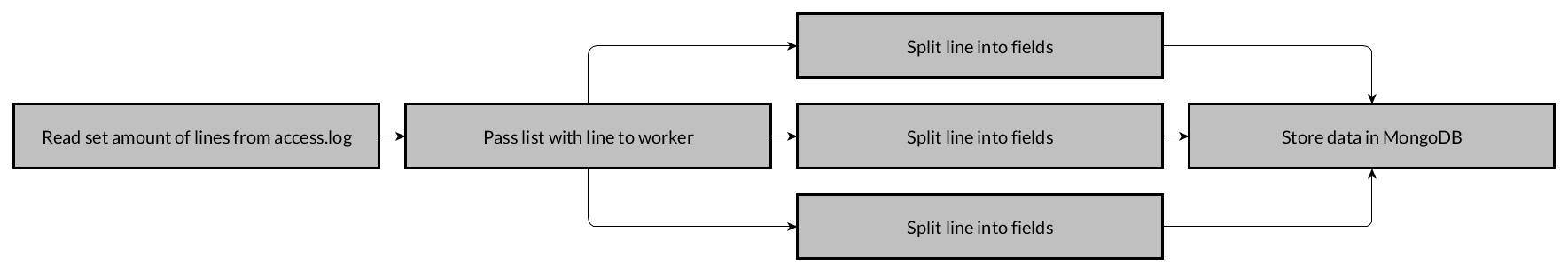


Figure 12: Working unifier.py in detail

### Module: PROFILER

The profiling modules of the engine will process the access.log data (stored in MongoDB by the unifier) into a profile. This will generate a profile of the traffic recorded during a baseline period. This profile typifies the traffic that is considered “safe” and will be used to detect anomalies in new connections.

Two type of profiles can be created, an app or user profile. They both implement the same metrics but use other keys. The app profile uses the requested URL as index (server side profiling). This means that every metric is URL-based for example metric\_agent will keep a record which user agents are used to connect to a certain URL of the web application. The other profile is the user profile; this profile uses the client IP as index (client side profiling). Every metric is IP-based for example metric\_agent will keep a record which user agents are used by a certain user.

The following diagram represents the structure of the profiles. As mentioned before, both profilers generate a diagram conform to the same data structure. The main difference between the user profile and the app profile is that that for the \_id field will contain an IP address or URL respectively. The metrics will be discussed in chapter 5.3

This module designed in such a way that they both use the same algorithms for determining the metrics (only the data supplied to the algorithm is different). When a new metric is implemented it only needs to be done once and not for both profilers separately. All these common function can be found in helper.py and will be reviewed in chapter 5.5.

The profiler will also process the input multithreaded.

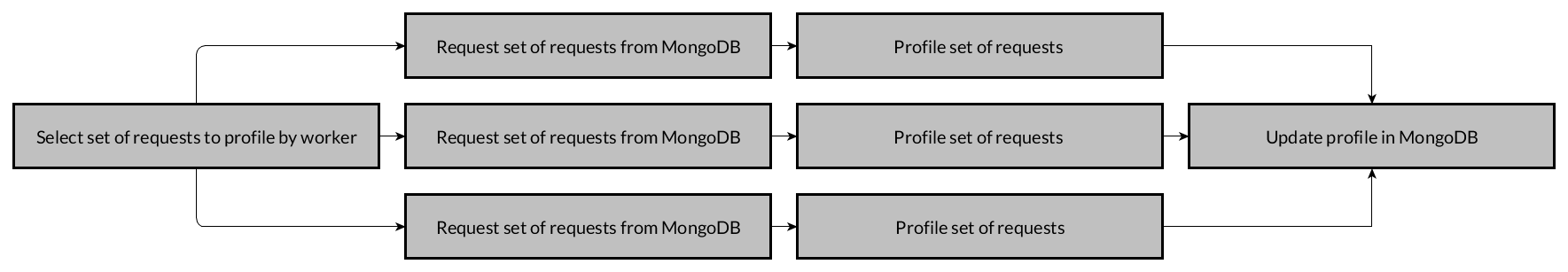


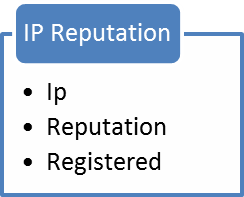
Figure 13: working of profiler in detail

### Module: FIREWALL

This module provides the actual firewall function of the engine. When enabled this module will use a file pointer on the access.log to track new connections. When a new line is added by the logging service of the webserver, it will be processed immediately. The firewall will construct a new profile of the new requests. This new profile is compared to the profile constructed during the baseline period in order to detect anomalies in the new requests.

When anomaly is detected, the incident will be reported. The incident is stored in a MongoDB along with its relevant data (timestamp, severity etc.) and the reputation of the client IP will be adjusted (according to the severity of the incident).

Once the reputation of the client IP exceeds a certain value, a firewall rule will be created in order to prevent any access to the web service by that client. After a certain period, the firewall rule will be removed and the client can access the web service again (until new anomalies are detected).



**IP Reputation DB:** this database keeps track of the reputation linked to a client IP. The field “Registered” keeps track if the reputation already caused the creation of a firewall rule or not.

**Firewall Messages DB:** this database keeps track of the reported incidents. The field “Details” contains which value caused the incident (example: normal connections/hour: 75, current connections/hour: 100, detail: +25)

## Determining metrics

Profiles are collections of several metrics. These metrics define values that typifies web requests. The type of value is different for every metric. This can be a counter, minmax, average or ratio. Every metric can be used later to detect anomalies.

In this chapter, each metric will be document in order to get a better understanding which and how anomalies can be detected.

### Types of metric values

Metrics can have different “types” of values; this is because the value of one metric has to be interpreted on a different way than the values of another metric. The following is a listing of all the possible types of metric values:

* **Counter:** simple counter that is incremented when a certain value occurs. (metric\_day, metric\_time etc.)
* **Ratio:** this is a value that presents how counters of each value are related. (metric\_agent, metric\_location)
* **Minmax:** these values contain the min and max value detected during profiling. (metric\_timespent, metric\_size)
* **Average/deviation:** the deviation is used to determine how far the values are spread from the average. (metric\_timespent, metric\_size)

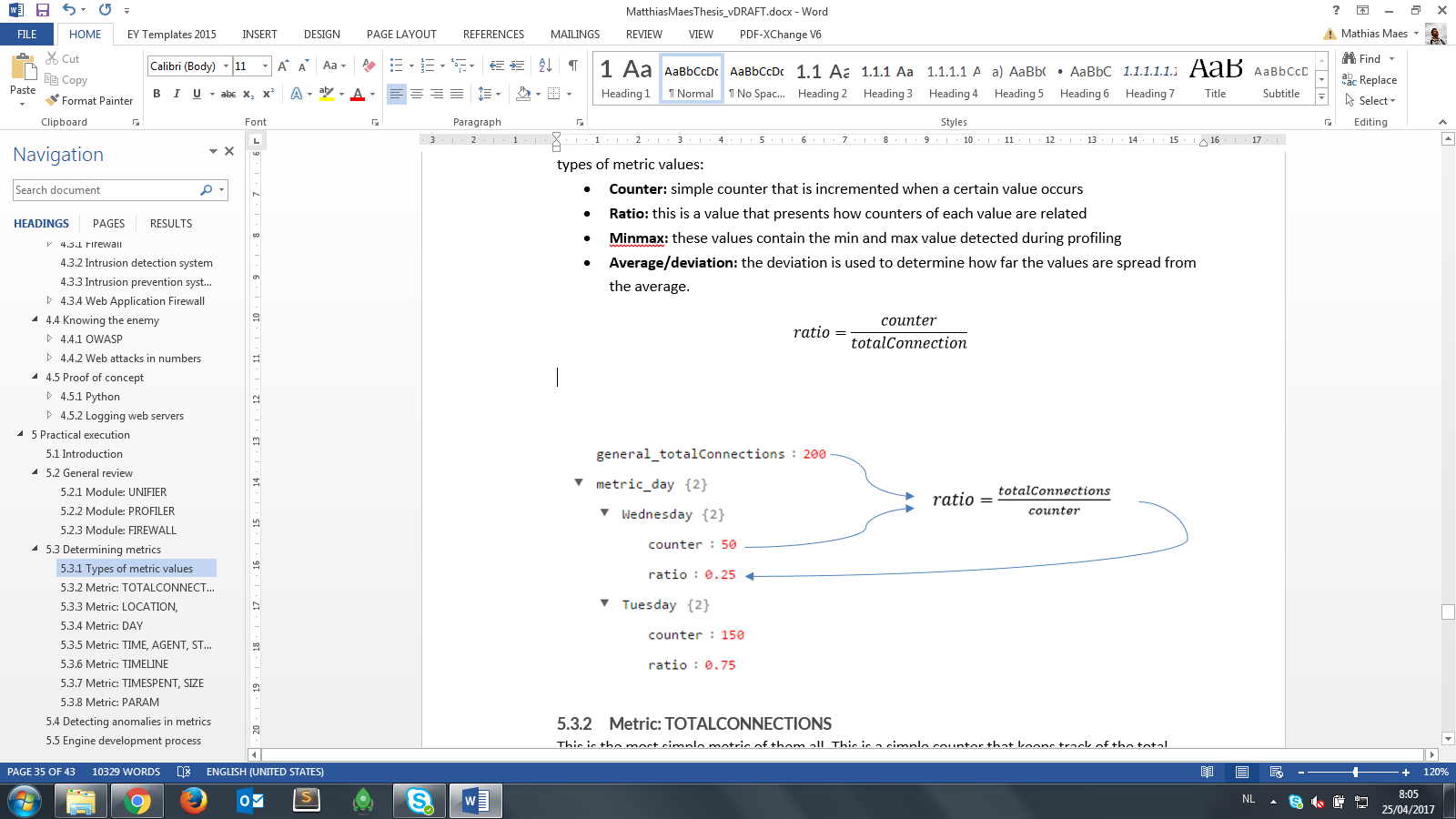
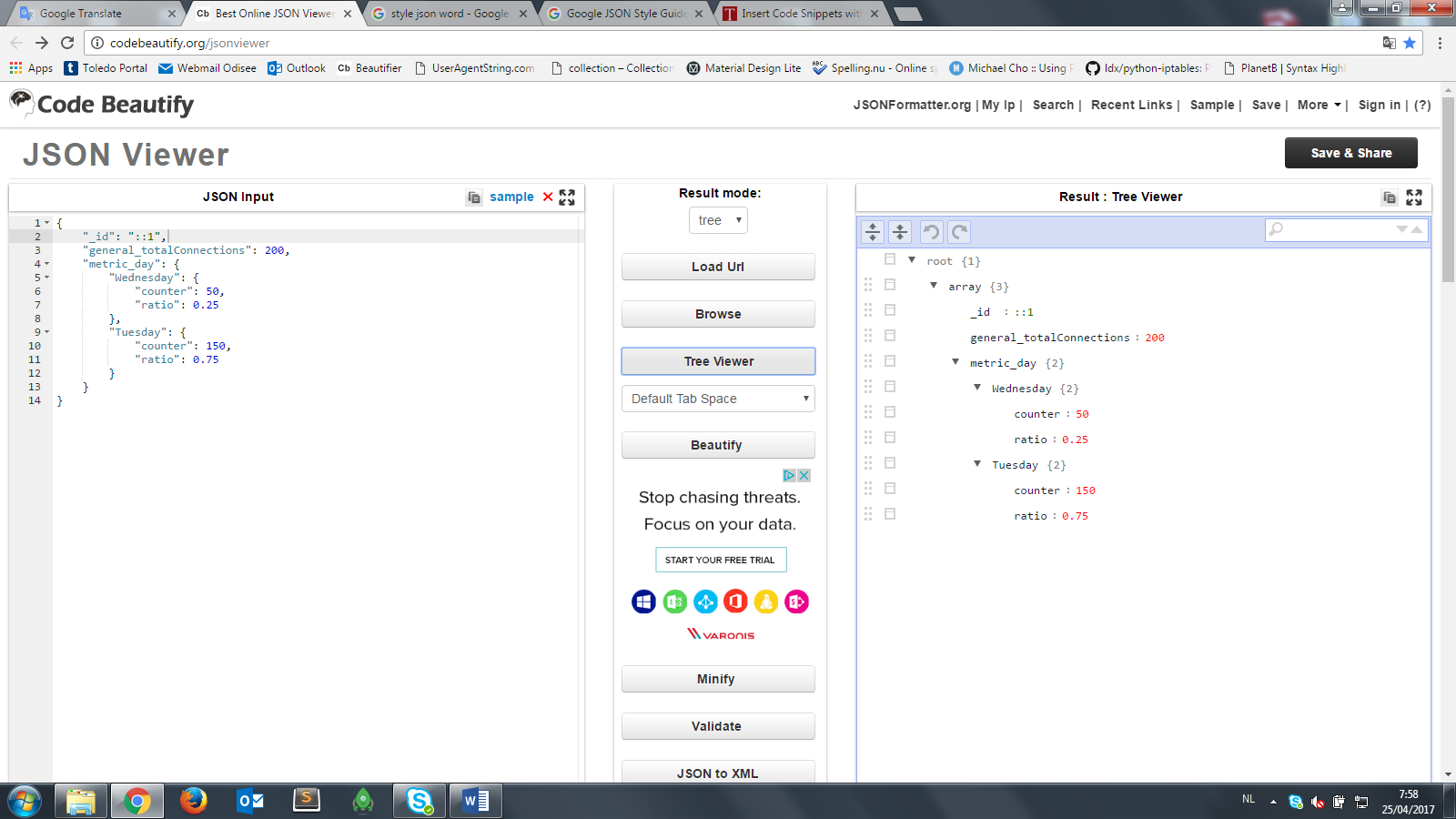
 

Figure 14: illustration ratio calculation

### Metric: TOTALCONNECTIONS

This is the simplest metric of them all. This counter keeps track of the total amount of connections on a URL or by a client IP.

### Metric: LOCATION

This metric uses a python library to determine the location of the client IP. This library uses a static, local bin file as data source. It stores a counter and a ratio. Escpecially the ratio is interesting, if this starts to deviate it can be a sign of DDos/botnet attack.

### Metric: DAY

This metric keeps a counter and ratio for each day off week on which a request was made. This allows the engine to detect an excessive amount of connections on a day to day basis. For example, a web shop typically gets more visits during the weekend. Thousands of connections during the weekend are normal but thousands of connections during a weekday can be suspicious. This metrics allows for making a distinction between a day in the weekend and a day in the week during the anomaly detection.

### Metric: TIME, AGENT, STATUS, METHOD, REQUEST, LOGIN

These metrics are all constructed the same way; every metric has a counter and a ratio.

* Metric\_time: requests at a certain time of the day.
* Metric\_agent: requests using a certain user agent. This metric can be used to detect crawler or scraping attacks.
* Metric\_status: request generating a certain HTTP status code (200, 404…).
* Metric\_method: requests using a certain HTTP method (GET, POST, PUT…).
* Metric\_request: requests that are being made from an URL or by a client IP. This allow profiling of the usual page flow through the web application.
* Metric\_conn: requests using the same base URL (URL without query string).
* Metric\_ext: requests for a certain file type.
* Metric\_login: requests for a certain backend page (normal, user, admin). If the URL contains words linked to backend activity (admin, backend, control panel etc.) it will be categorized as an admin URL. A user that normally connects to non-admin pages and then all of a sudden makes numerous connections to admin related URLs is considered suspicious.

### Metric: TIMELINE

The timeline metric is not directly linked to any anomaly detection but serves as data for the calculation of other metrics. It contains a list with URLs (user profile) or IPs (app profile) linked to a timestamp. This metric serves as data for the “time spent” metric but can be used for any time related metric.

### Metric: TIMESPENT, SIZE

This metric is based upon the metric\_timeline and contains for every URL (user profile) or IP (app profile) the amount of time spent by a user on the URL (app profile) or time spent on pages by a user (user profile). This can also be interpreted as usual time between requests. The time spent is presented in a min max value, an average and a standard deviation (used to test how much the values deviate from the average).

Metric\_size is constructed the same way (counter, ratio, minmax, average, deviation) but used the response size as input data.

### Metric: PARAM

This is the most interesting metric and is the most powerful in detecting anomalies. For this metric, the query sting is analyzed. Every key/value pair is split and for every pair several properties are defined. First of all the value is analyzed for the presence of any special characters, average length and the type (string, integer). For every value a counter is stored to count how many times a certain key/value pair is requested.

This metric can easily detect attempts at SQLi, XSS or other injection attacks because they rely on the use of special characters like ‘ or (.

## Detecting anomalies

After the creation of the profiles based on the baseline period, the firewall is ready to detect anomalies. Each new request is analyzed by the firewall and will contribute to the profile of the current network traffic. This new profile will be compared to the baseline profiles (each time a request was received) to detect anomalies. The method for detecting anomalies depends on the value type of the metric. The following listing describes how every value type will be analyzed:

* Counter/ratio: an anomaly on this value type is detected if the counter or ratio of the current traffic exceeds the counter or ratio of the baseline traffic. This test includes a customizable threshold. This means that the value has the exceed the old value + threshold before triggering any warnings.
* Minmax: an anomaly on this value type is detected if the new value is outside the range defined by the min and max of the value in the baseline profile.
* Average/deviation: an anomaly on this value type is detected if the new value exceeds more than two sigma from the average.

### Severity

Each time an anomaly is detected the incident is reported. Each incidents gets a “severity level”. This level defines how severe a certain incident is. The higher the severity the greater the reputation of the client gets affected. There are three severity levels:

* SEVERITY.LOW
* SEVERITY.HIGH
* SEVERITY.CRITICAL

### Examples

#### Counter/ratio

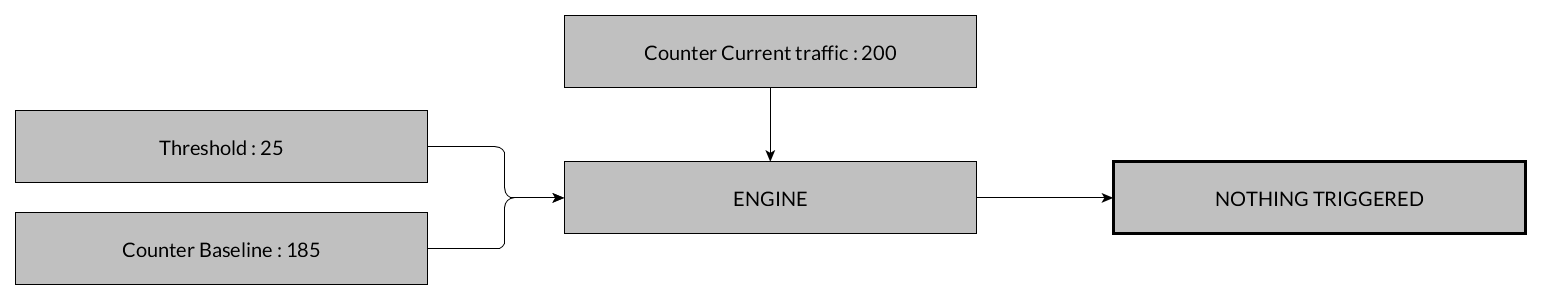


Figure 15: Situation in which the counter/ratio anomaly detection where no incidents are reported

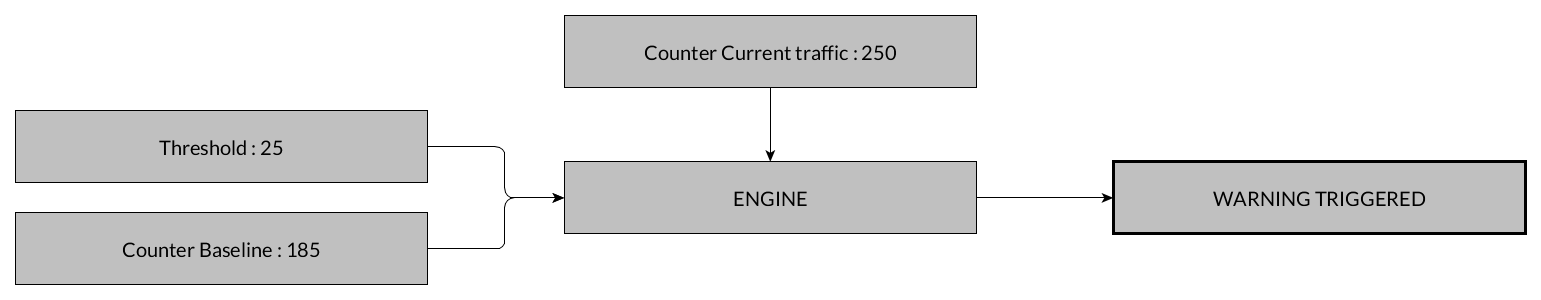


Figure 16: Situation in which the counter/ratio anomaly detection where an incidents is reported

#### Minmax

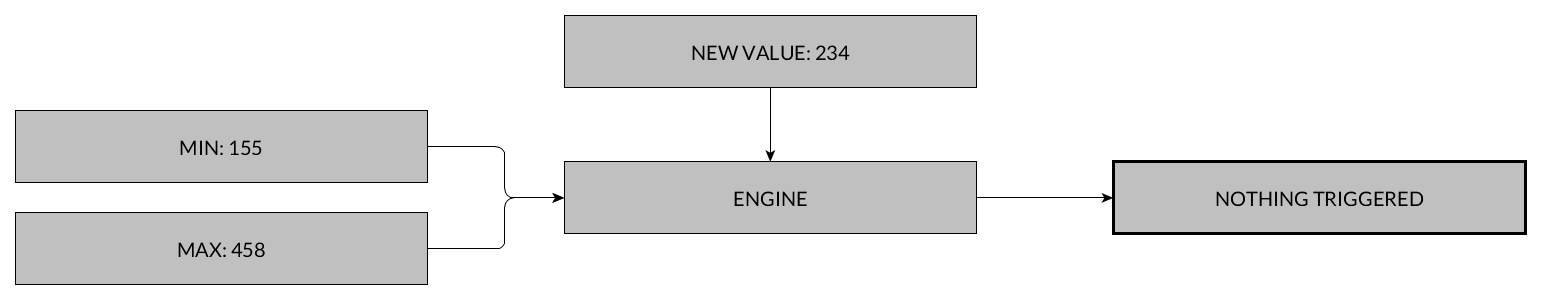


Figure 17: Situation in which the minmax anomaly detection where no incidents are reported

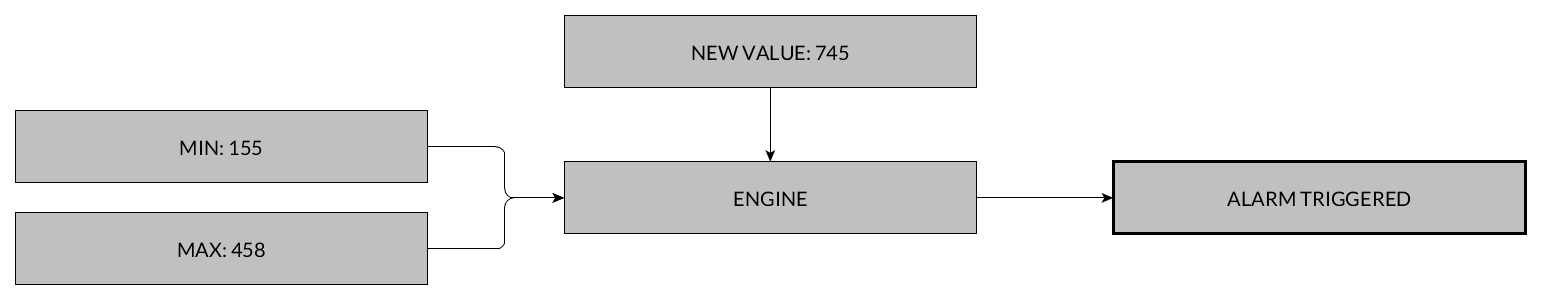


Figure 18: Situation in which the minmax anomaly detection where an incidents is reported

#### Average/deviation

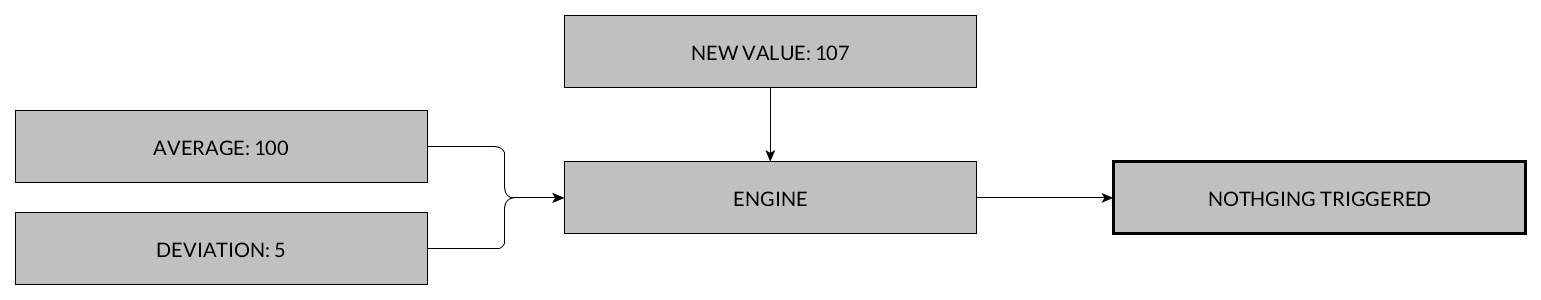


Figure 19: Situation in which the average/deviation anomaly detection where no incidents are reported

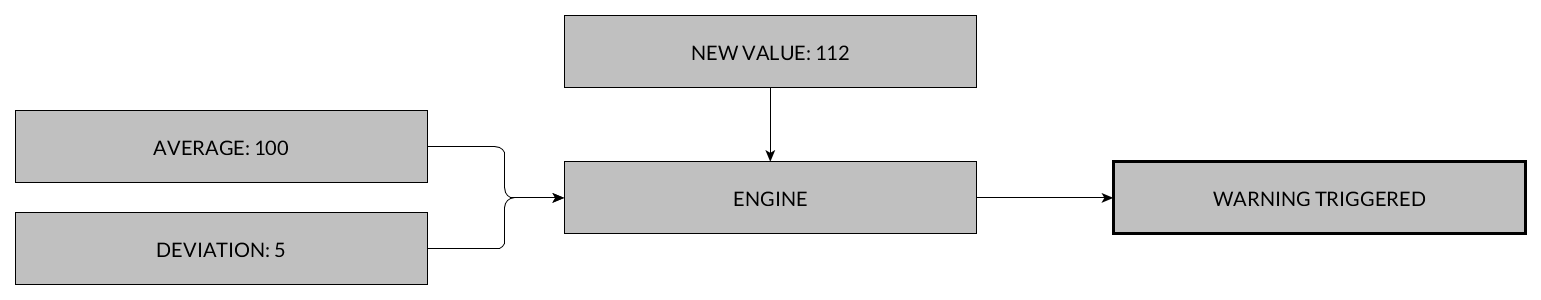


Figure 20: Situation in which the average/deviation anomaly detection where an incidents is reported (Low severity)

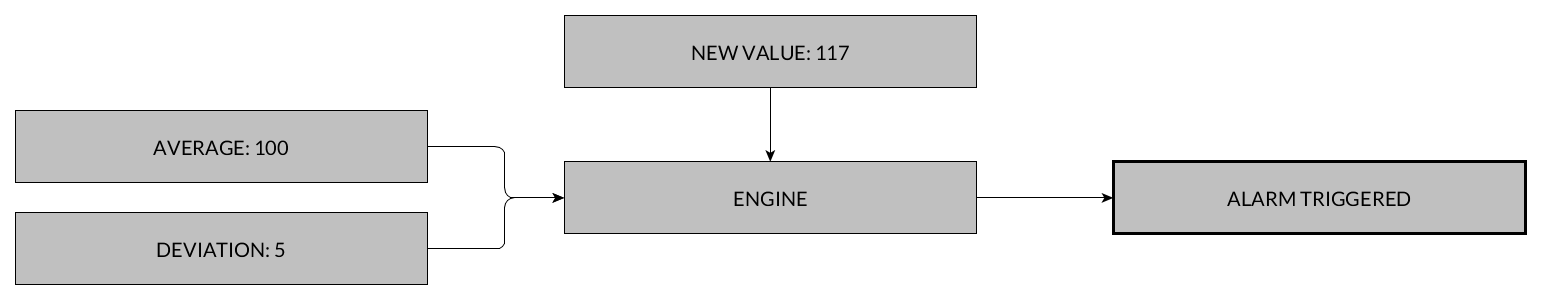


Figure 21: Situation in which the average/deviation anomaly detection where an incidents is reported (High severity)

## Code review

This chapter will cover an in depth code review, it is advised to thoroughly read through the previous chapters to get a better understanding of the project before delving into the code. Only the most relevant and interesting pieces of code will discussed, the countless variable initialization, MongoDB connection will be ignored. The functions of helper.py will be discussed along with the module where those functions have been used.

As mentioned before, the project was programmed using Python (version: 2.7).

In order to run the engine following modules are required: IP2Location, pymongo, progressbar, dnspython, python-iptables.

### Code: UNIFIER

The first step in using the engine is unifying the baseline access.log in order to prepare the data for profiling.

#### Preparing

1. #### Determining amount of lines ####
2. with open(options.log) as f:
3. num\_lines = sum(1 **for** line **in** f)
4. linesToProcess = (num\_lines \* int(options.procentToParse)) / 100
6. #### Determining start and end ####
7. startIndex = int(options.startToParse)
8. endIndex = num\_lines **if** startIndex + linesToProcess > num\_lines **else** startIndex + linesToProcess
9. **print** 'Lines from {} till {} will be processed'.format(startIndex, endIndex)

First, the amount of lines that needs to be processed will be determined. When starting the script the user can define how much of the file needs to be processed in a percent value. At line 4, this value is brought into consideration. The user can also set the start index and/or end index. At line 8 the end index is set, there is an extra test to make sure the end index does not exceeds the total amount of lines. The last line in this block will print a debug message so the user knows which lines will be processed.

#### Assign workload to workers

1. #### Spread workload among workers ####
2. with open(options.log) as fileobject:
3. **for** index, line **in** enumerate(fileobject, startIndex):
5. lines.append(line)
7. #### If lines reach max per thread or eof a worker is started ####
8. **if** index % int(float(options.linesPerThread)) == 0 **or** index >= num\_lines **or** index >= endIndex - 1:
10. #### Wait for a free worker ####
11. **while** str(activeWorkers) == str(options.threads):
12. **pass**
14. #### Start a new worker ####
15. activeWorkers += 1
16. t = threading.Thread(target=formatLine, args=(lines,i,))
17. i += int(options.linesPerThread)
18. threads.append(t)
19. t.start()
21. #### Clear the list after assignment ####
22. lines = list()
24. **if** index == endIndex:
25. **continue**
27. progressBarObj.update(index)

After all the preparation have been made its time to assign a workload to the workers. An iterator over the access.log file adds lines to a list. If this list reaches the maximum amount of lines to be processed at once or the end of the file, the list is ready to be processed by a worker. At line 11 the “while” waits for a worker to free up, once a worker is free the list is passed on (line 16). Once the worker has started, the list is cleared (line 22) and the index is set to process the next set of lines. To prevent any lines after the end index are processed line 24 was added. Each iteration the progress bar gets updated, it passes along the amount of lines that already have been processed.

#### Process access.log entry

1. **def** formatLine(lines, index):
2. """ Format lines into unified object """
3. **for** line **in** lines:
4. **try**:
5. MongoDB.insert\_one(helperObj.processLine(line, index))
6. **except** Exception as e:
7. **print** 'Following error occured: {} on line {}'.format(line, e)
8. **print** e
10. **global** activeWorkers
11. activeWorkers -= 1

The above is the method that will be assigned to a worker. This method iterates over the supplied list and calls the “processLine” method from helper.py at line 5. Once the iteration has finished and all the lines has to be processed, the method finishes by freeing up the worker for a new task (line 11).

The “global” statement at line 10 is necessary because the activeWorkers variable is shared across all treads.

#### Helper.py: processLine

1. **def** processLine(self, inputLine, index):
2. index += 1
3. cleandedLine = filter(None, [x.strip() **for** x **in** inputLine.replace('""','"-"').split('"')])
4. ip = cleandedLine[0].split(' ')[0]
5. fulltime = cleandedLine[0].split(' ')[3].replace('[', '') + ' ' +  cleandedLine[0].split(' ')[4].replace(']', '')
6. method = cleandedLine[1].split(' ')[0]
7. requestUrl = '-' **if** cleandedLine[1] == '-' **else** cleandedLine[1].split(' ')[1]
8. code = cleandedLine[2].split(' ')[0]
9. size = cleandedLine[2].split(' ')[1]
10. url = cleandedLine[3]
11. uagent = cleandedLine[4]
12. **return** FormattedLine(index, ip, fulltime, method, requestUrl, code, size, url, uagent).\_\_dict\_\_

This is the main function of the unifier module and can be found in the helper class. It all starts by replacing empty parameters with a “–“(this would otherwise mess up the split function) and splitting the line on double quotes at line 3. This is done to isolate the first line of the request, referrer and user agent (these are between quotes in the log file). After multiple replacements and splits all, the required values ready and will be returned as a “FormattedLine” object. The worker in the unifier class will insert this object into the MongoDB. If the format deviates from the standard format, only this method has to be altered.

#### Finishing off

1. #### Wait for all threads to finish ####
2. **for** thread **in** threads:
3. thread.join()
5. #### Finish script ####
6. progressBarObj.finish()
7. #### Info to process rest of file ####
8. **if** int(options.startToParse) **is** **not** 0 **or** int(options.procentToParse) **is** **not** 100:
9. **print** 'Next parse should start from index: {}. Use -s {}'.format(endIndex + 1, endIndex + 1)

When all the lines have been processed, the script finishes by joining all threads (making sure all thread are stopped correctly). Once this is done, a small debug message is shown. This message helps setting up the next execution by printing out the start index that should be used upon next execution. This is done so lines will not be excluded by mistake.

### Code: PROFILERS

Some parts of the code in the profiler classes are reused from the unifier class. If lines were already discussed, they will no longer be considered.

#### Assign workload to workers

1. loops = int(math.ceil(float(diffLines) / float(intLinesPerThread)))
2. loops += 2
4. **try**:
5. **for** index **in** xrange(0, loops):
7. #### Hold until worker is free ####
8. **while** str(activeWorkers) == str(options.threads):
9. **pass**
11. #### Start of worker ####
12. activeWorkers += 1
13. t = threading.Thread(target=processLine, args=(startRange, index,))
14. threads.append(t)
15. t.start()
17. #### Set range for next thread ####
18. startRange += intLinesPerThread
20. **except** KeyboardInterrupt:
21. **print** 'Script cancelled by user'
23. **finally**:
24. **for** thread **in** threads:
25. thread.join()
26. progressBarObj.finish()

The workload assignment is done a bit differently than in the unifier class. In the profiler classes, the amount of iterations is determined at line 1. The second line looks a bit odd and random, this line is required to make sure all the necessary lines will be processed (without this lines errors will occur when linesToProcess < linesPerThread). Once the amount of lines has been determined an iterator is started at line 5. This iterator is surrounded by a try/catch block, this is to make sure when the script is cancelled all the threads will exit correctly. When a worker is started it is not with a list as input (like in unifier.py) but with a start and end index. The other lines (waiting for free worker, starting thread) were discussed in the previous chapter.

#### Process FormattedLine

1. **def** processLine(start, index):
2. """ Assign workers with workload """
4. **global** converted
6. **for** inputLine **in** InputMongoDB.find()[start : start + int(options.linesPerThread)]:
8. **if** converted >= diffLines:
9. **print** 'break on: ' + str(converted)
10. **break**
11. **else**:
12. progressBarObj.update(converted)
13. helperObj.processLineCombined(TYPE.APP, SCRIPT.PROFILER, inputLine, options)

The processLine method also shares great similarity with the one in the unifier class. The interesting line of code is the call for the “proccessLineCombined” method from the helper class (line 13). The first two parameters in the method call are what makes it able to reuse that method for the firewall and other profiler. The first parameter “TYPE.APP” specifies that the method is called from the app profiler (and not the user profiler). The second parameter “SCRIPT.PROFILER” specifies that the method is called form the profilers (and not from the firewall). The following is the same line but taken from the user profiler.

1. helperObj.processLineCombined(TYPE.USER, SCRIPT.PROFILER, inputLine, options)

In this case, the TYPE.USER parameter is used to specify the method is called from the user profiler. This is the only line that is different between the two profilers.

#### Helper.py: processLineCombined

This method is the heart of the engine. The app profiler, user profiler and the firewall use it. It is responsible for constructing the profiles. Due to its large size, the method will be split during the review.

1. **if** typeProfile == TYPE.USER:
2. key = inputLine['ip']
3. otherKey = self.getUrlWithoutQuery(inputLine['requestUrl']).replace('.','\_')
4. **elif** typeProfile == TYPE.APP:
5. key = self.getUrlWithoutQuery(inputLine['requestUrl'])
6. otherKey = inputLine['ip'].replace('.','\_')

The method starts of by defining the key and the otherKey. If the method was called form the user profiler the client IP will be used as key, and the otherKey will be the URL. The key and otherKey will be switched if the method was called form the app profiler.

1. **if** self.OutputMongoDB.find({'\_id': key}).count() == 0:
2. **try**:
3. self.OutputMongoDB.update\_one({'\_id': key}, {'$set': {'metric\_param': {}}} , upsert=True)
4. **except** Exception as e:
5. **if** 'E11000' **in** e.message:
6. **pass**
7. **else**:
8. **raise** e

Next, the object will be created in the MongoDB. There is a quite complex if and try structure around the insert (line 3). This is to prevents errors caused by multithreading (see flowchart).

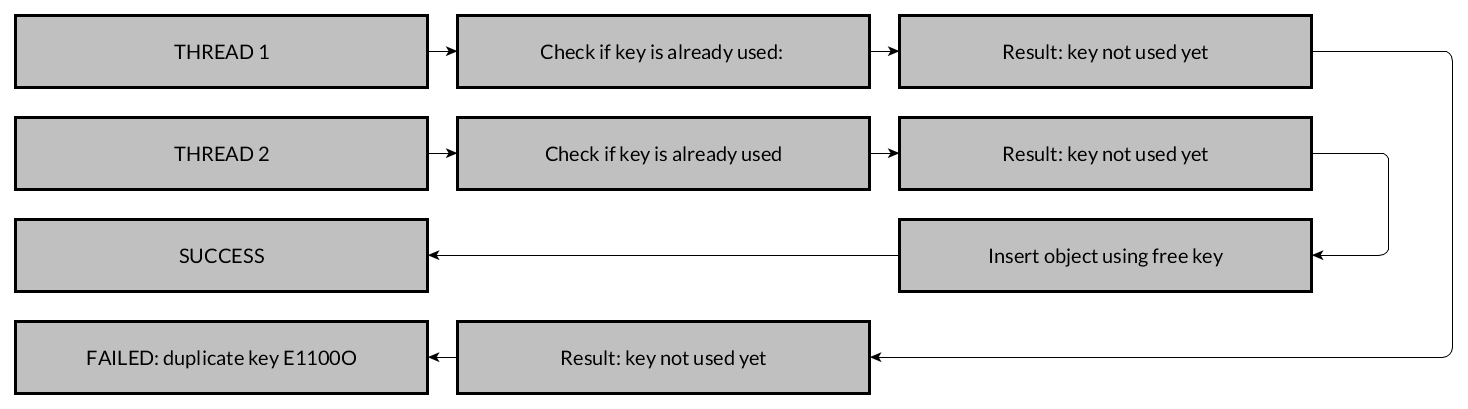


Figure 22: illustration DuplicateKey due to multithreading

The insert is actually an update statement with the upsert parameter set to true. This means that the document will be updated and if it not exists, it will be inserted (not flawless in multithreaded environment).

1. bulk = self.OutputMongoDB.initialize\_unordered\_bulk\_op()
2. bulk.find({'\_id': key}).update\_one({'$inc': {'general\_totalConnections': 1 }})
3. bulk.find({'\_id': key}).update\_one({'$set': {'general\_timeline.' + timestamp.strftime('%d/%b/%Y %H:%M:%S'): otherKey}})
4. **if** typeProfile == TYPE.USER:
5. bulk.find({'\_id': key}).update\_one({'$set': {'general\_location': self.GeoLocate(inputLine['ip'], options.ping) }})
6. **elif** typeProfile == TYPE.APP:
7. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_location.' + self.GeoLocate(inputLine['ip'], options.ping) + '.counter': 1 }})
8. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_day.' + timestamp.strftime("%A") + '.counter': 1 }})
9. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_time.' + timestamp.strftime("%H") + '.counter': 1 }})
10. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_agent.' + inputLine['uagent'].replace('.', '\_') + '.counter': 1 }})
11. bulk.find({'\_id': key}).update\_one({'$set': {'metric\_agent.' + inputLine['uagent'].replace('.', '\_') + '.uagentType': 'Human' **if** self.BotMongoDB.find({'agent': inputLine['uagent']}).count() == 0 **else** 'Bot' }})
12. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_request.' + inputLine['requestUrl'].replace('.', '\_') + '.counter': 1 }})
13. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_ext.' + self.getFileType(inputLine['requestUrl']) +'.counter': 1 }})
14. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_status.' + inputLine['code'] +'.counter': 1 }})
15. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_method.' + inputLine['method'] +'.counter': 1 }})
16. bulk.find({'\_id': key}).update\_one({'$inc': {'metric\_conn.' + otherKey + '.counter': 1 }})

This code fragment is a part of the bulk statement (created at line 1). A bulk statement is a group of MongoDB query that will be executed at once. These statements update the counter for the given metric/value. Whenever a value contains a point it will be replaced with an underscore otherwise MongoDB will interpret is as a child item. Some small helper function are called but they are negligible.

On line 11, an extra check is added to filter the bot user agents from the normal user agents. This is done to filter out request originating from crawlers, search engines…

1. #### Add querystring param ####
2. analysed\_param = list()
3. **if** len(queryString) > 0:
5. **for** param **in** queryString:
6. **if** len(param.split('=')) == 2:
7. pKey = param.split('=')[0]
8. pValue = '-' **if** **not** param.split('=')[1] **else** param.split('=')[1]
10. #### Determine type of param ####
11. **try**:
12. int(pValue)
13. paramType = 'int'
14. **except** ValueError:
15. paramType = 'bool' **if** pValue == 'true' **or** pValue == 'false' **else** 'string'
16. **except** Exception:
17. **print** param
19. #### Detecting special chars in param ####
20. chars = 'special' **if** any(char **in** string.punctuation **for** char **in** pValue) **else** 'normal'
22. #### Add to bulk updates ####
23. bulk.find({'\_id': key}).update\_one({'$set': { 'metric\_param.' + pKey + '.characters': chars}})
24. bulk.find({'\_id': key}).update\_one({'$set': { 'metric\_param.' + pKey + '.length': len(pValue)}})
25. bulk.find({'\_id': key}).update\_one({'$set': { 'metric\_param.' + pKey + '.type': paramType}})
26. bulk.find({'\_id': key}).update\_one({'$inc': { 'metric\_param.' + pKey + '.' + pValue + '.counter': 1}})
27. bulk.find({'\_id': key}).update\_one({'$inc': { 'metric\_param.' + pKey + '.counter': 1}})
29. analysed\_param.append({
30. 'key': pKey,
31. 'characters': chars,
32. 'length': len(pValue),
33. 'type': paramType
34. })

Next is the analysis of the query string. The query string is split to iterate each parameter individually. Inside the iterator each parameter is split into a pKey (= parameter key) and a pValue (=parameter value). First of all the type of the value is determined, this is done by trying some casts and catching “ValueErrors” (casting failed) until a cast succeeds (Integer > Boolean > String). Once the type is determined, the checks for special characters kick off. Python has a built in collection of special chars (string.punctuation). The test for the presence of any special characters is as simple as iterating each character in pValue and testing if that character is part of string.punctuation collection (see line 20). The last metric to be determined is the length of the pValue, tis is done using the builtin function “len()”. Finally, all these results are added to the bulk statement.

At lines 29-34 is an object created, the reason for this object will be discussed later.

1. **try**:
2. bulk.execute()
3. **except** Exception as e:
4. **print** e.details

Finally, the bulk statement is executed.

#### Calculate ratio

1. **def** calculateRatio(self, value, metric):
2. """ Method for calculating the ratio for a given metric """
4. currRecord = self.OutputMongoDB.find\_one({'\_id': value })
5. **for** metricEntry **in** currRecord[metric]:
6. **if** metricEntry **is** **not** '' **or** metricEntry **is** **not** None:
7. self.OutputMongoDB.update({'\_id': value}, {'$set': {metric + '.' + metricEntry + '.ratio': float(currRecord[metric][metricEntry]['counter']) / float(currRecord['general\_totalConnections'])}})

This method is used to calculate the ratio for every metric (except metric\_param). Line 7 is the most crucial one, here the counter value is divided by the total connections and the result is stored in the MongoDB.

1. **def** calculateRatioParam(self, key, queryString):
2. """ Method for calculating the ratio for a given metric """
3. currRecord = self.OutputMongoDB.find\_one({'\_id': key })
5. **if** len(queryString) > 0:
6. **for** param **in** queryString:
7. **if** len(param.split('=')) == 2:
8. pKey = param.split('=')[0]
9. **for** param **in** currRecord['metric\_param'][pKey]:
10. **try**:
11. #### Update ratio on all affected records and metrics (if counter changes on one metric, ratio on all has to be updated) ####
12. self.OutputMongoDB.update({'\_id': key}, {'$set': { 'metric\_param' + '.' + pKey + '.' + param + '.ratio': float(currRecord['metric\_param'][pKey][param]['counter']) / float(currRecord['metric\_param'][pKey]['counter'])}})
13. **except** TypeError:
14. #### Not every metric has a counter/ratio field, this will be catched by the TypeError exception ####
15. **pass**

The ratio calculation for metric\_param is a bit more complicated because there is need for an extra iterator. Line 12 is the one where the actual ratio will be calculated.

1. self.calculateRatio(key, 'metric\_agent')
2. self.calculateRatio(key, 'metric\_time')
3. self.calculateRatio(key, 'metric\_day')
4. self.calculateRatio(key, 'metric\_request')
5. self.calculateRatio(key, 'metric\_status')
6. self.calculateRatio(key, 'metric\_method')
7. self.calculateRatio(key, 'metric\_ext')
8. self.calculateRatio(key, 'metric\_login')
9. self.calculateRatio(key, 'metric\_conn')
10. self.calculateRatioParam(key, queryString)
12. **if** typeProfile == TYPE.APP:
13. self.calculateRatio(key, 'metric\_location')

Those two methods are called when the bulk operation has finished.

#### Calculate min max

1. counter = self.OutputMongoDB.find\_one({ '\_id' : identifier })['metric\_conn'][otherIdentifier]['counter']
3. # MIN #
4. **try**:
5. orgMin =self.OutputMongoDB.find\_one({ '\_id' : identifier })[metric][otherIdentifier]['min']
6. newMin = newVal **if** newVal < orgMin **else** orgMin
8. **except** KeyError:
9. newMin = newVal
10. **finally**:
11. self.OutputMongoDB.update\_one({ '\_id' : identifier }, { '$set' : {metric + '.' + otherIdentifier + '.min': int(newMin)}})
13. # MAX #
14. **try**:
15. orgMax =self.OutputMongoDB.find\_one({ '\_id' : identifier })[metric][otherIdentifier]['max']
16. newMax = newVal **if** newVal > orgMax **else** orgMax
17. **except** KeyError:
18. newMax = newVal
19. **finally**:
20. self.OutputMongoDB.update\_one({ '\_id' : identifier }, { '$set' : {metric + '.' + otherIdentifier + '.max': int(newMax)}})

Another metric value type is min and max. These tests (line 6 and 16) are quite simple (test if new value is smaller/larger than the original min/max)

#### Calculate average and standard deviation

1. # AVERAGE #
2. **try**:
3. orgAvg = self.OutputMongoDB.find\_one({ '\_id' : identifier })[metric][otherIdentifier]['average']
4. newAvg = (orgAvg \* (counter - 1) + newVal) / counter
5. **except** KeyError:
6. newAvg = newVal
7. **except** Exception as e:
8. **print** e
10. **finally**:
11. self.OutputMongoDB.update\_one({ '\_id' : identifier }, { '$set' : {metric + '.' + otherIdentifier + '.average': int(newAvg)}})

14. # DEVIANCE #
15. **try**:
16. orgDeviation = math.pow((self.OutputMongoDB.find\_one({ '\_id' : identifier })[metric][otherIdentifier]['deviation']),2)
17. newDeviation = (((counter-1) \* orgDeviation) + (newVal - newAvg) \* (newVal - orgAvg)) / (counter)
18. **except** KeyError:
19. newDeviation = 0
20. **finally**:
21. self.OutputMongoDB.update\_one({ '\_id' : identifier }, { '$set' : {metric + '.' + otherIdentifier + '.deviation': math.sqrt(int(newDeviation))}})

The two most interesting calculations are the one for the average and the standard deviation. Normally the calculation of an average is very easy but in this case, we have to add values to the average. An average has to be recalculated but the original values are not stored in the db. A formula for adding values to an average is needed:

This function is implemented at line 4.

The same goes for the standard deviation, every iteration a value has to be added without access to the original values. To achieve this the variance formula is reversed. The original formula is as follows:

If the formula is reversed, the following formula is derived. The variance is first multiplied with the amount of elements to obtain the sum of elements (numerator original formula) then the new value is added (NewValue-NewAverage)\*(NewValue-OrgAvg).

After applying this formula (line 17) the only thing left to do is to take the square root of the new variance to obtain the standard deviation. This value is then stored in the MongoDB.

#### Finishing off

The finishing procedure is the same as the one in the unifier class.

### Code: FIREWALL

The firewall is split into three scripts. This is done intentionally to maintain a certain degree of separation and to allow the script to run on different hosts (maybe the client uses another firewall than iptables on the webserver). In order to function correctly the three scripts need to be running at the same time.

* RequestAnalyser.py: the actual firewall and is the one analyzing the new requests and detecting any anomalies.
* ReputationAnalyser: checks the reputation database periodically if any IPs need to be blocked (add rule to IPTables)
* IptablesAnalyser: checks the iptables rules periodically if any of the rules are expired.

#### RequestAnalyser

##### Preparing

1. **if** os.path.exists('C:/wamp64/logs/access.log'):
2. path = 'C:/wamp64/logs/access.log'
3. **elif** os.path.exists('/var/log/nginx/access.log'):
4. path = '/var/log/nginx/access.log'

The script starts by setting the default path for the access.log, this path is different on a windows and a Linux system. This path can be overwritten by the user using the command line options.

1. with open(path) as fileobject:
2. **print** ' - [LOG] [OK] Ready to start processing requests...'
3. fileobject.seek(0,2)
4. **while** True:
5. inputLine = fileobject.readline()

Next the file is opened (line 1), and the file pointer is set to the end of the file (line 3). The script is put into an infinite loop (line 4). The built-in readline function will read a line whenever a new line is added at the end of the access.log (line 5). Once a new line is detected, the processor is started.

1. #### Create line object and insert it in mongodb
2. lineObj = helperObj.processLine(inputLine, index)

5. ## App filtering
6. **print** '\n----- App analysis -----'
7. tmpLastObj = helperObj.processLineCombined(TYPE.APP, SCRIPT.FIREWALL, lineObj, options)
9. **if** ProfileAppMongoDB.find({'\_id': helperObj.getUrlWithoutQuery(lineObj['requestUrl'])}).count() > 0:
10. startAnomalyDetection(lineObj, ProfileAppMongoDB.find\_one({'\_id': helperObj.getUrlWithoutQuery(lineObj['requestUrl'])}), tmpLastObj, TYPE.APP)
11. **else**:
12. **print** 'Not profiled page'

Before starting the anomaly detection, the new line is sent to the “processLine” function. This is the same function as the one used in the unifier. This will return a unified line object (member of the FormattedLine class). The tmpLastObj is used to return certain metric values of that request (ex: day: Monday, timespent: 160…).

1. ## User filtering
2. **print** '\n----- User analysis -----'
3. tmpLastObj = helperObj.processLineCombined(TYPE.USER, SCRIPT.FIREWALL, lineObj, options)
5. **if** ProfileUserMongoDB.find({'\_id': lineObj['ip']}).count() > 0:
6. startAnomalyDetection(lineObj, ProfileUserMongoDB.find\_one({'\_id': lineObj['ip']}), tmpLastObj, TYPE.USER)
7. **else**:
8. **print** 'Not profiled user'

Once the line is unified, the anomaly detection can be started. First the script will add the processedLine to the profile of the current network traffic (line 3). Once this is done the user anomaly detection is started. Before the test is started, a quick test is performed to make sure the app was profiled during the baseline period (line 5)

##### Anomaly detection

1. **def** startAnomalyDetection(packet, profileRecord, tmpLastObj, typeProfile):
2. """ Start anomaly detection process """
4. # Start off by performing the static checks (ip, location...)
5. **if** (anomaly\_StaticChecks(packet)):
6. . . .
7. **else**:
8. FirewallAlarmException('Static list block', 'ip/uagent', 0, SEVERITY.CRITICAL, tmpLastObj['typeProfile'], tmpLastObj['ip'])
9. **def** anomaly\_StaticChecks(packet):
10. """ Check static blocklist with ips and spam user agent list"""
11. **return** IPReputationMongoDB.find\_one({'\_id' : packet['ip']}) == None **and** SpamAgentMongoDB.find\_one({'string' : packet['uagent']}) == None

The first check is the check for static blocks. If this check triggers any alarms the rest of the anomaly checks are skipped. These static checks are just a simple search through a MongoDB of blocked IPs and one with user agents linked to spam. Should the static checks not trigger any alarms the next checks are called.

1. requestRecord = helperObj.OutputMongoDB.find\_one({'\_id': helperObj.getUrlWithoutQuery(packet['requestUrl'])})
3. # Perform most basic checks (Total connections)
4. anomaly\_TotalConnections(profileRecord, requestRecord, tmpLastObj)
5. anomaly\_ParamUnknown(profileRecord, requestRecord, tmpLastObj)
7. # Iterate over all metrics and perform the standard checks (counter, ratio and average)
8. **for** metric **in** ProfileAppMongoDB.find\_one():
9. **if** 'metric' **in** metric **and** 'param' **not** **in** metric **and** 'timespent' **not** **in** metric:
10. anomaly\_GeneralUnknown(metric, profileRecord, requestRecord, tmpLastObj)
11. **if** 'timespent' **in** metric **or** 'size' **in** metric:
12. anomaly\_GeneralDeviation(metric, profileRecord, requestRecord, tmpLastObj)
13. **def** anomaly\_TotalConnections (profileRecord, requestRecord, tmpLastObj):
14. """ Detect to many connections """
15. diff = int(requestRecord['general\_totalConnections']) - int(profileRecord['general\_totalConnections'])
16. **if** threshold\_counter < diff: FirewallAlarmException('Counter exceeded', 'general\_TotalConnections', diff, SEVERITY.LOW, tmpLastObj['typeProfile'], tmpLastObj['ip'])

The first anomaly check is also the simplest. The test “anomaly\_TotalConnections” checks if the total amount of connections is not exceeded. It does this by making the difference between the value in the current profile and the value in the baseline profile (line 3). If this difference is larger than the threshold, a warning is triggered with a low severity (line 4). The “FirewallAlarmException” will be discussed later on. After the total amount of connections have been checked, the check on the query string parameters will kick in.

1. **def** anomaly\_ParamUnknown(profileRecord, requestRecord, tmpLastObj):
2. """ Detect unknowns in parameter metric """

5. **for** analysedParam **in** tmpLastObj['analysed\_param']:
6. **if** analysedParam['key'] **in** profileRecord['metric\_param']:
7. anomaly\_ParamAnomaly(profileRecord, requestRecord, tmpLastObj)
8. anomaly\_ParamAnalyzed(profileRecord, analysedParam)
9. **else**:
10. FirewallAlarmException('Unknown param', 'metric\_param', analysedParam['key'], SEVERITY.HIGH, tmpLastObj['typeProfile'], tmpLastObj['ip'])

The “anomaly\_ParamUnknown” method first checks if the parameters occurs in the profile (line 6), if they do not an alarm is triggered with high severity (line 10). If the parameter does occurs in the profile the next checks are started on the query string.

1. **def** anomaly\_ParamAnomaly (profileRecord, requestRecord, tmpLastObj):
2. """ Detect to many connections on specific querystring parameter """
3. **for** analysedParam **in** tmpLastObj['analysed\_param']:
4. diff = int(requestRecord['metric\_param'][analysedParam['key']]['counter']) - int(profileRecord['metric\_param'][analysedParam['key']]['counter'])
5. **if** threshold\_counter < diff: FirewallAlarmException('Counter exceeded', 'metric\_param', diff, SEVERITY.LOW, tmpLastObj['typeProfile'], tmpLastObj['ip'])

The “anomaly\_ParamAnomaly” method works according the same principle as the check on the total connections. The only difference is that the check in encapsulated by an iterator (because the query string can contain more than one parameter.

1. **def** anomaly\_ParamAnalyzed (profileRecord, analysedParam):
3. # Test for type
4. **if** analysedParam['type'] != profileRecord['metric\_param'][analysedParam['key']]['type']:
5. FirewallAlarmException('Param type mismatch', 'metric\_param', 'Expected: ' + profileRecord['metric\_param'][analysedParam['key']]['type'] + ' - Received: ' + analysedParam['type'] + ' - ON PARAM: ' + analysedParam['key'], SEVERITY.HIGH, tmpLastObj['typeProfile'], tmpLastObj['ip'])
7. # Test for chars
8. **if** analysedParam['characters'] != profileRecord['metric\_param'][analysedParam['key']]['characters']:
9. FirewallAlarmException('Param characters mismatch', 'metric\_param', 'Expected: ' + profileRecord['metric\_param'][analysedParam['key']]['characters'] + ' - Received: ' + analysedParam['characters'] + ' - ON PARAM: ' + analysedParam['key'], SEVERITY.CRITICAL, tmpLastObj['typeProfile'], tmpLastObj['ip'])
11. # Test for length
12. **if** abs(analysedParam['length'] - profileRecord['metric\_param'][analysedParam['key']]['length']) > threshold\_length:
13. FirewallAlarmException('Param length mismatch', 'metric\_param', 'Expected: ' + str(profileRecord['metric\_param'][analysedParam['key']]['length']) + ' - Received: ' + str(analysedParam['length']) + ' - ON PARAM: ' + analysedParam['key'], SEVERITY.LOW, tmpLastObj['typeProfile'], tmpLastObj['ip'])

The last method for analyzing the query string parameters is the most interesting one. This is where the type, character set and length are tested. At line 4 the check for the type is implemented, this compares the type in the current profile with the type in the baseline profile. If the both types do not match, a warning with high severity is triggered. Both the test for the character set (line 8) and the length (line 12) are implemented the same way. This concludes the tests on the query string. Afterwards all the other metrics are tested.

1. # Iterate over all metrics and perform the standard checks
2. **for** metric **in** ProfileAppMongoDB.find\_one():
3. **if** 'metric' **in** metric **and** 'param' **not** **in** metric **and** 'timespent' **not** **in** metric:
4. anomaly\_GeneralUnknown(metric, profileRecord, requestRecord, tmpLastObj)
5. **if** 'timespent' **in** metric **or** 'size' **in** metric:
6. anomaly\_GeneralDeviation(metric, profileRecord, requestRecord, tmpLastObj)
7. **def** anomaly\_GeneralUnknown(metric, profileRecord, requestRecord, tmpLastObj):
8. """ Generic method for detecting unknown anomalies for the given metrics """
9. **if** tmpLastObj[metric] **in** profileRecord[metric]:
10. anomaly\_GeneralCounter(metric, profileRecord, requestRecord, tmpLastObj)
11. anomaly\_GeneralRatio(metric, profileRecord, requestRecord, tmpLastObj)
12. **else**:
13. **if** metric != 'metric\_size' **and** metric != 'metric\_timespent':
14. FirewallAlarmException('Unknown found in', metric, tmpLastObj[metric], SEVERITY.HIGH, tmpLastObj['typeProfile'], tmpLastObj['ip'])
16. # Do not detect unknown on min max metrics (timespent, size etc)
17. anomaly\_GeneralMinMax(metric, profileRecord, requestRecord, tmpLastObj)

The test for unknown in metrics is implemented in roughly the same way as the unknown detection on the query string parameter. The only difference is that this time the checks are not encapsulated in an iterator.

1. **def** anomaly\_GeneralCounter (metric, profileRecord, requestRecord, tmpLastObj):
2. """ Generic method for detecting excessive counter on given metric """
3. diff = int(requestRecord[metric][tmpLastObj[metric]]['counter']) - int(profileRecord[metric][tmpLastObj[metric]]['counter'])
4. **if** threshold\_counter < diff: FirewallAlarmException('Counter exceeded', metric, diff, SEVERITY.LOW, tmpLastObj['typeProfile'], tmpLastObj['ip'])

In addition, the test on the counter shows great similarity with the checks on the parameters and will not be any further discussed for this reason.

1. **def** anomaly\_GeneralRatio(metric, profileRecord, requestRecord, tmpLastObj):
2. """ Generic method for detecting excessive ratio on given metric """
3. diff = float(requestRecord[metric][tmpLastObj[metric]]['ratio']) - float(profileRecord[metric][tmpLastObj[metric]]['ratio'])
4. **if** **not**(-threshold\_ratio <= diff <= threshold\_ratio): FirewallAlarmException('Ratio exceeded', metric, diff, SEVERITY.LOW, tmpLastObj['typeProfile'], tmpLastObj['ip'])

The check on the ratio is something new. This time the difference is made between the ratio of the metric in the current profile and the ratio of the metric in the baseline profile (line 3). At the end of the method is a check that will trigger an alarm when the ratio exceeds the threshold interval (line 4).

1. **def** anomaly\_GeneralMinMax(metric, profileRecord, requestRecord, tmpLastObj):
2. """ Generic method for detecting anomalies in min max from metrics """
4. **try**:
5. # Test on min
6. **if** requestRecord[metric][tmpLastObj['otherkey']]['min'] < profileRecord[metric][tmpLastObj['otherkey']]['min']:
7. FirewallAlarmException('Lower min found', metric, requestRecord[metric][tmpLastObj['otherkey']]['min'], SEVERITY.HIGH, tmpLastObj['typeProfile'], tmpLastObj['otherip'])
9. # Test on max
10. **if** requestRecord[metric][tmpLastObj['otherkey']]['max'] > profileRecord[metric][tmpLastObj['otherkey']]['max']:
11. FirewallAlarmException('Higher max found', metric, requestRecord[metric][tmpLastObj['otherkey']]['max'], SEVERITY.CRITICAL, tmpLastObj['typeProfile'], tmpLastObj['otherip'])
12. **except** KeyError:
13. # Not every metric has a min/max defined
14. **pass**

Next up are the min max checks, they compare the original min/max with the current value. The tests are encapsulated in a try/catch block because not every metric has the min max property. Trying to access the min max property and catching the error requires less computational resources than testing if the min max property exists using an if statement.

1. **def** anomaly\_GeneralDeviation(metric, profileRecord, requestRecord, tmpLastObj):
2. """ Generic method for detecting anomalies in deviation of average """
4. **try**:
5. **try**:
6. newValue = int(tmpLastObj[metric])
7. avg = profileRecord[metric][tmpLastObj['otherkey']]['average']
8. standev = profileRecord[metric][tmpLastObj['otherkey']]['deviation']
9. **except** ValueError:
10. **print** '[DEBUG] VALUE ERROR'
11. **return**
12. **except** TypeError:
13. **print** '[DEBUG] TYPE ERROR'
14. **return**
16. # The further the average deviates the higher the alert becomes
18. **if** newValue **not** **in** xrange(int(avg - standev),  int(avg + standev)):
19. **if** newValue **in** xrange(int(avg - 2 \* standev),  int(avg + 2 \* standev)):
20. FirewallAlarmException('Value deviates between 1 and 2 sigma form average', metric, 'Value (' + str(newValue) + ') within range: ' + str(avg - 2 \* standev) + ' | ' + str(avg + 2 \* standev) , SEVERITY.HIGH, tmpLastObj['typeProfile'], tmpLastObj['ip'])
21. **else**:
22. FirewallAlarmException('Value deviates more than 2 sigma from average', metric, 'Value (' + str(newValue) + ') outside range: ' + str(avg - 2 \* standev) + ' | ' + str(avg + 2 \* standev) , SEVERITY.CRITICAL, tmpLastObj['typeProfile'], tmpLastObj['ip'])
24. **except** KeyError:
25. # Not every metric has a deviation defined
26. **pass**

The final check is the one using the average and the standard deviation. This method will test how far the new value deviates from the average. If the value deviates between one and two sigma (line 18 – 19) an alarm is triggered with a high severity. If the value deviates more than 2 sigma a critical alarm is triggered.

This concludes all the tests for the app anomaly detection, the exact same tests are executed for user anomaly detection and will not further discussed for this reason.

##### FirewallAlarmExecption

1. **class** FirewallAlarmException(Exception):
2. ReputationMongoDB = MongoClient().Firewall.reputation
3. MessageMongoDB = MongoClient().engine\_log.firewall\_messages
5. **def** \_\_init\_\_(self, message, metric, details, severity, typeProfile, ip):
6. self.timestamp = datetime.datetime.now().strftime('[%d/%m/%Y][%H:%M:%S]')
7. self.message = str(message)
8. self.metric = str(metric)
9. self.details = str(details)
10. self.severity = severity
12. **if** typeProfile == TYPE.USER:
13. self.ReputationMongoDB.update\_one({'ip': ip}, {'$inc': {'rep': -1 \* int(severity + 1)}}, upsert=True)
14. self.ReputationMongoDB.update\_one({'ip': ip}, {'$set' : {'registered': False}})
16. self.MessageMongoDB.insert\_one(self.\_\_dict\_\_)
17. **print** self
19. **def** \_\_str\_\_(self):
20. **return** str(self.timestamp) + '[' + str(self.severity) + '] ' + str(self.message) + ' (' + str(self.metric) + ', ' + str(self.details) + ')'

The “FirewallAlarmException” is used by the firewall to report an incident. It not only saves the incident to a MongoDB (line 16), it also updates the reputation in the according MongoDB (line 12 – 14). During debug is also print the incident to the console (line 17).

#### ReputationAnalyser

The reputation analyzer monitors the reputation MongoDB whenever the reputation drops below a certain point the following method will be called to block the IP.

1. **def** blockIpTable(ip):
2. table = iptc.Table(iptc.Table.FILTER)
3. chain = iptc.Chain(table, "INPUT")
4. rule = iptc.Rule()
5. rule.src = ip
6. rule.target = rule.create\_target("DROP")
7. rule.match = rule.create\_match("comment").comment = str(datetime.datetime.strftime(datetime.datetime.now() + datetime.timedelta(minutes=1), "%y-%m-%d %H:%M:%S"))
8. chain.insert\_rule(rule)

An IPtable rule is created using the iptc python module. Line 2 and 3 define the location for the rule. Line 5 defines the IP address that needs to be blocked. Line 6 defines the action, in this case every connections needs to be blocked. Line 7 adds a comment with a timestamp when the rule expires. The last line inserts the rule in the IPtable.

#### IptablesAnalyser

The iptables analyzer monitors the iptable to test if a rule is expired, if a rule is expired it will remove the rule from the iptables.

1. **for** rule **in** chain.rules:
2. index += 1
3. **for** match **in** rule.matches:
4. **if** match.comment != None:
5. datetime\_object = datetime.strptime(match.comment, "%y-%m-%d %H:%M:%S")
7. **if** datetime.now() > datetime\_object:
8. chain.delete\_rule(rule)
9. **print** "Rule successfully deleted"

An iterator is created (line 1) to iterate over all the rules in the INPUT chain. Another iterator is created on each rule to iterate the matches (line 3). The comment is cast to a datetime object (line 5) and if the rule is expired (line 7) it will be removed from the IPtable (line 8).

# Conclusion

## Thesis

### Achievements

## Internship

### Working at EY

### Communication mentors

# References

|  |  |
| --- | --- |
| [1] | EY, "EY," EY, [Online]. Available: http://www.ey.com/be/en/home. [Accessed 07 February 2017]. |
| [2] | Wikipedia, "Ernst & Young," Wikipedia, 07 February 2017. [Online]. Available: https://en.wikipedia.org/wiki/Ernst\_%26\_Young. [Accessed 07 February 2017]. |
| [3] | Wikipedia, "General Data Protection Regulation," Wikipedia, 24 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/General\_Data\_Protection\_Regulation. [Accessed 08 February 2017]. |
| [4] | Wikipedia, "Big Four accounting firms," Wikipedia, 06 February 2017. [Online]. Available: https://en.wikipedia.org/wiki/Big\_Four\_accounting\_firms. [Accessed 07 February 2017]. |
| [5] | Wikipedia, "Threat (computer)," Wikipedia, 06 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/Threat\_(computer). [Accessed 14 February 2017]. |
| [6] | Owasp, "Real World Threat Modeling," 2012. [Online]. Available: https://www.owasp.org/images/a/aa/AppSecEU2012\_PASTA.pdf. [Accessed 14 February 2017]. |
| [7] | S. Simeonova, "Threat Modeling in the Enterprise, Part 2: Understanding the Process," securityintelligence, [Online]. Available: https://securityintelligence.com/threat-modeling-in-the-enterprise-part-2-understanding-the-process/. [Accessed 15 February 2017]. |
| [8] | threatmodeler, "Comparison of Threat Modeling Methodologies," threatmodeler, [Online]. Available: http://threatmodeler.com/comparison-threat-modeling-methodologies/. [Accessed 14 February 2017]. |
| [9] | owasp, "Threat Risk Modeling," owasp, 23 January 2017. [Online]. Available: https://www.owasp.org/index.php/Threat\_Risk\_Modeling. [Accessed 14 February 2017]. |
| [10] | Wikipedia, "Threat model," Wikipedia, 21 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/Threat\_model. [Accessed 15 February 2017]. |
| [11] | isaca, "APPLICATION THREAT MODELING," isaca, November 2013. [Online]. Available: http://www.isaca.org/chapters5/Ireland/Documents/2013%20Presentations/PASTA%20Methodology%20Appendix%20-%20November%202013.pdf. [Accessed 14 February 2017]. |
| [12] | Wikipedia, "STRIDE (security)," Wikipedia, 29 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/STRIDE\_(security). [Accessed 15 February 2017]. |
| [13] | microsoft, "The STRIDE Threat Model," microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. [Accessed 14 February 2017]. |
| [14] | owasp, "Repudiation Attack," owasp, 19 December 2013. [Online]. Available: https://www.owasp.org/index.php/Repudiation\_Attack. [Accessed 14 February 2017]. |
| [15] | dirtycow, "dirtycow," dirtycow, [Online]. Available: https://dirtycow.ninja/. [Accessed 15 February 2017]. |
| [16] | Wikipedia, "Privilege escalation," Wikipedia, 10 November 2016. [Online]. Available: https://en.wikipedia.org/wiki/Privilege\_escalation. [Accessed 15 February 2017]. |
| [17] | DenyAll, "Discover DenyAll IP Reputation Services," DenyAll, [Online]. Available: https://www.denyall.com/products/denyall-ip-reputation-service/. [Accessed 07 February 2017]. |
| [18] | barracuda, "Configuring IP Reputation Filter," barracuda, [Online]. Available: https://campus.barracuda.com/product/webapplicationfirewall/article/WAF/ConfigIPReputationFilter/. [Accessed 06 February 2017]. |
| [19] | citrix, "IP Reputation," citrix, [Online]. Available: https://docs.citrix.com/en-us/netscaler/11/security/reputation/ip-reputation.html. [Accessed 07 February 2017]. |
| [20] | cerberhost, "Reputation based Firewalling," cerberhost, [Online]. Available: http://www.cerberhost.com/avada\_portfolio/ip\_reputation\_based\_firewall/. [Accessed 06 February 2017]. |
| [21] | scmagazine, "Signature-Based or Anomaly-Based Intrusion Detection: The Practice and Pitfalls," scmagazine, [Online]. Available: https://www.scmagazine.com/signature-based-or-anomaly-based-intrusion-detection-the-practice-and-pitfalls/article/548733/. [Accessed 07 February 2017]. |
| [22] | Wikipedia, "User behavior analytics," Wikipedia, 08 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/User\_behavior\_analytics. [Accessed 07 February 2017]. |
| [23] | D. Shinder, "The Pros and Cons of Behavioral Based, Signature Based and Whitelist Based Security," techgenix, 13 November 2008. [Online]. Available: http://techgenix.com/pros-cons-behavioral-signature-whitelist-security/. [Accessed 08 February 2017]. |
| [24] | Wikipedia, "Stateful firewall," Wikipedia, 3 December 2016. [Online]. Available: https://en.wikipedia.org/wiki/Stateful\_firewallStateful firewall. [Accessed 20 February 2017]. |
| [25] | Wikipedia, "Next-Generation Firewall," Wikipedia, 9 February 2017. [Online]. Available: https://en.wikipedia.org/wiki/Next-Generation\_Firewall. [Accessed 20 February 2017]. |
| [26] | Wikipedia, "Intrusion detection system," Wikipedia, 13 February 2017. [Online]. Available: https://en.wikipedia.org/wiki/Intrusion\_detection\_system. [Accessed 20 February 2017]. |
| [27] | paloaltonetworks, "What is an intrusion prevention system?," paloaltonetworks, 2017. [Online]. Available: https://www.paloaltonetworks.com/documentation/glossary/what-is-an-intrusion-prevention-system-ips. [Accessed 20 February 2017]. |
| [28] | J. McMillan, "IDFAQ: What is the Difference Between an IPS and a Web Application Firewall?," sans.org, November 2009. [Online]. Available: https://www.sans.org/security-resources/idfaq/what-is-the-difference-between-an-ips-and-a-web-application-firewall/1/25. [Accessed 20 February 2017]. |
| [29] | AQTRONiX , "AQTRONiX WebKnight - Open Source Web Application Firewall (WAF) for IIS," AQTRONiX , [Online]. Available: https://www.aqtronix.com/?PageID=99. [Accessed 07 Februari 2017]. |
| [30] | Wikipedia, "Application firewall," Wikipedia, 29 January 2017. [Online]. Available: https://en.wikipedia.org/wiki/Application\_firewall. [Accessed 06 February 2017]. |
| [31] | A. Ndegwa, "What is a Web Application Firewall?," maxcdn.com, 31 May 2016. [Online]. Available: https://www.maxcdn.com/one/visual-glossary/web-application-firewall/. [Accessed 20 February 2017]. |
| [32] | B. Causey, "Introduction to Web application firewalls in the enterprise," techtarget, [Online]. Available: http://searchsecurity.techtarget.com/feature/Introduction-to-Web-application-firewalls-in-the-enterprise. [Accessed 06 February 2017]. |
| [33] | OWASP, "Web Application Firewall," OWASP, [Online]. Available: https://www.owasp.org/index.php/Web\_Application\_Firewall. [Accessed 06 February 2017]. |
| [34] | M. Rouse, "Web application firewall (WAF)," techtarget, [Online]. Available: http://searchsecurity.techtarget.com/definition/Web-application-firewall-WAF. [Accessed 06 February 2017]. |
| [35] | M. Rouse, "PCI DSS (Payment Card Industry Data Security Standard)," techtarget, [Online]. Available: http://searchfinancialsecurity.techtarget.com/definition/PCI-DSS-Payment-Card-Industry-Data-Security-Standard. [Accessed 06 February 2017]. |
| [36] | C. Kumar, "5 Open Source Web Application Firewall for Better Security," geekflare, 4 September 2016. [Online]. Available: https://geekflare.com/free-transactional-hosted-marketing-email/. [Accessed 07 Februari 2017]. |
| [37] | Trustwave, "What Can ModSecurity Do?," Trustwave, [Online]. Available: https://www.modsecurity.org/about.html. [Accessed 07 Februari 2017]. |
| [38] | Qualis, "IronBee Reference Manual," Qualis, 04 January 2016. [Online]. Available: https://www.ironbee.com/docs/manual/ironbee-reference-manual.html. [Accessed 07 Februari 2017]. |
| [39] | naxsi, "naxsi," naxsi, 02 August 2016. [Online]. Available: https://github.com/nbs-system/naxsi. [Accessed 07 Februari 2017]. |
| [40] | applicure.com, "Software WAF vs. Appliance WAF," applicure.com, [Online]. Available: http://www.applicure.com/solutions/software-waf-vs-appliance-waf. [Accessed 20 February 2017]. |
| [41] | Barracuda, "Barracuda WAF," Barracuda, [Online]. Available: https://campus.barracuda.com/product/webapplicationfirewall/article/WAF/. [Accessed 21 February 2017]. |
| [42] | M. Fainshtein, "Barracuda Web Application Firewall," Barracuda, [Online]. Available: https://www.barracuda.com/assets/docs/Datasheets/Barracuda\_Web\_Application\_Firewall\_DS\_US.pdf. [Accessed 21 February 2017]. |
| [43] | citrix, "Citrix NetScaler Application Firewall," citrix, [Online]. Available: https://www.citrix.com/content/dam/citrix/en\_us/documents/data-sheet/citrix-netscaler-application-firewall-datasheet.pdf. [Accessed 21 February 2017]. |
| [44] | Imperva, "Imperva SecureSphere Appliances," Imperva, [Online]. Available: https://www.imperva.com/docs/DS\_SecureSphere\_Appliances.pdf. [Accessed 21 February 2017]. |
| [45] | geekflare, "Top 3 Cloud Web Application Firewall to Stop Website Attacks (for Small to Medium Business)," geekflare, [Online]. Available: https://geekflare.com/cloud-waf-to-stop-website-attacks/. [Accessed 21 February 2017]. |
| [46] | OWASP, "Top 10 2013-Top 10," OWASP, 21 August 2015. [Online]. Available: https://www.owasp.org/index.php/Top\_10\_2013-Top\_10. [Accessed 27 Februari 2017]. |
| [47] | Paolo Passeri, "February 2017 Cyber Attacks Statistics," HACKMAGEDDON, 20 March 2017. [Online]. Available: http://www.hackmageddon.com/category/security/cyber-attacks-statistics/. [Accessed 23 March 2017]. |
| [48] | Symantec, "Attackers Target Both Large and Small Businesses," Symantec, [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/infographics/istr-attackers-strike-large-business-en.pdf. [Accessed 23 March 2017]. |
| [49] | calyptix, "Top 5 Cyber Attack Types in 2016 So Far," calyptix, 1 August 2016. [Online]. Available: http://www.calyptix.com/top-threats/top-5-cyber-attack-types-in-2016-so-far/. [Accessed 22 March 2017]. |
| [50] | Wikipedia, "Ruby (programmeertaal)," Wikipedia, 27 Dec 2016. [Online]. Available: https://nl.wikipedia.org/wiki/Ruby\_(programmeertaal). [Accessed 22 March 2017]. |
| [51] | Ruby community, "About ruby," Ruby community, [Online]. Available: https://www.ruby-lang.org/en/about/. [Accessed 22 March 2017]. |
| [52] | Wikipedia, "Python (programmeertaal)," Wikipedia, 2 March 2017. [Online]. Available: https://nl.wikipedia.org/wiki/Python\_(programmeertaal). [Accessed 22 March 2017]. |
| [53] | Apache Software Foundation, "Log Files," Apache Software Foundation, [Online]. Available: https://httpd.apache.org/docs/2.4/logs.html. [Accessed 22 March 20117]. |
| [54] | EY, "EY Logo," [Online]. Available: http://www.ey.com/ecimages/EY.gif. [Accessed 1 March 2017]. |
| [55] | Arthur Young, "Arthur Young Logo," [Online]. Available: https://res.cloudinary.com/crunchbase-production/image/upload/v1457147586/eve4v8gr0liwkkhbvtqw.jpg. [Accessed 1 March 2017]. |
| [56] | Erns & Whinney, "Erns & Whinney Logo," [Online]. Available: http://userscontent2.emaze.com/images/86dbf6c2-1ab1-4300-b458-0e0c31ce5ac3/635348645831478302\_EWlogo.jpg. [Accessed 1 March 2017]. |
| [57] | EY, "EY Corporate Presenation," May 2015. [Online]. Available: http://images.slideplayer.com/24/7228379/slides/slide\_3.jpg. [Accessed 1 March 2017]. |
| [58] | hightechmentor, "Healthcare Security Threats in 2017: Build your Defenses Now!," 2 February 2017. [Online]. Available: https://www.clickssl.net/wp-content/uploads/2015/02/cyber-security-intelligence-threat-management.jpg. [Accessed 27 March 2017]. |
| [59] | OWASP, "OWASP Mobile Security Project," 6 March 2017. [Online]. Available: https://www.owasp.org/index.php/OWASP\_Mobile\_Security\_Project. [Accessed 27 March 2017]. |

1. The primary objective is to give citizens back the control of their personal data. Enterprises that are not complied with these new regulations will risk a fine of 4% of their annual revenue. [↑](#footnote-ref-1)