UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

TESIS DE MAESTRÍA INFORMÁTICA

Enhancing web application attack detection using machine learning

Author:
Rodrigo MARTÍNEZ

Supervisors:
Dr. Ing. Gustavo BETARTE
Dr. Ing. Álvaro PARDO

A thesis submitted in fulfillment of the requirements for the degree of Magister en Informática

in the

PEDECIBA Informática
Instituto de Computación

November 26, 2019

Declaration of Authorship

I, Rodrigo MARTÍNEZ, declare that this thesis titled, "Enhancing web application attack detection using machine learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

Abstract

Facultad de Ingeniería
Instituto de Computación

Magister en Informática

Enhancing web application attack detection using machine learning

by Rodrigo MARTÍNEZ

Despite all effort of the security community, for example initiatives as the OWASP Top 10, it is a known fact that web applications are permanently being exposed to attacks that exploit their vulnerabilities. Some web applications vulnerabilities can only be discovered as a result of a process of trial and error performed by an attacker. The identification and determination of a user's behavior using attack detection techniques become crucial, these techniques assist in aspects such as preventing attackers to identify/verify successfully the existence of vulnerabilities in applications and to minimize the number of false positives (non-malicious activity identified as such). A technological alternative for performing real-time attack analysis is the use of a Web Application Firewall (WAF), systems that intercepts and inspects all traffic between the web server and its clients, searching for attacks in the communication's content. Most WAF works by using a set of statics rules defined to identify attacks.

In this thesis, we analyze the use of machine learning techniques to enhance web applications attack detection in MODSECURITY, an open source WAF that has became a `de facto` standard implementation.

We first propose a characterization of the problem by defining different scenarios depending on whether we have application' specific or generic data, as well as, valid and/or attack traffic available for training. We also analyze existing dataset to use in this context and we have created our own dataset by capturing real traffic to a real life application.

We finally present two supervised machine learning solutions. The first is a classic discrimination approach between two classes (valid traffic and attacks) [18]. The second is a one-class classification solution for a more realistic scenario when only valid data is available. In the one-class classification approach it is assumed that one of the classes can be properly modeled using data from the training set (in our case the valid traffic) while the other class (in our problem attacks) can not be modeled by total or partial lack of training samples [33]. We present results using both approaches and compare them with MODSECURITY configured with the OWASP Core Rule Set out of the box, which is the most widely deployed set of rules.

UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

Resumen

Facultad de Ingeniería
Instituto de Computación

Magister en Informática

Enhancing web application attack detection using machine learning

por Rodrigo MARTÍNEZ

A pesar de todos los esfuerzos de la comunidad de seguridad, por ejemplo, iniciativas como el OWASP Top 10, es un hecho conocido que las aplicaciones web están permanentemente expuestas a ataques que explotan sus vulnerabilidades. Algunas de estas vulnerabilidades solo se pueden descubrir como resultado de un proceso de ensayo y error realizado por un atacante. La identificación y determinación del comportamiento de un usuario utilizando técnicas de detección de ataques se vuelven cruciales. Estas técnicas ayudan en aspectos tales como evitar que los atacantes identifiquen/verifiquen con éxito la existencia de vulnerabilidades en las aplicaciones, así como minimizar el número de falsos positivos (actividad no maliciosa identificada como tal). Una alternativa tecnológica para realizar análisis de ataques en tiempo real es el uso de un firewall de aplicaciones web (WAF por su siglas en inglés). Estos sistemas interceptan e inspeccionan el tráfico entre el servidor web y sus clientes, buscando ataques en el contenido de la comunicación. La mayoría de los WAF funcionan mediante el uso de un conjunto de reglas estáticas definidas para identificar ataques.

En esta tesis, analizamos el uso de técnicas de aprendizaje automático para mejorar la detección de ataques de aplicaciones web en MODSECURITY, un WAF de código abierto que se ha convertido en un estándar *de facto*.

Primero, proponemos una caracterización del problema definiendo diferentes escenarios dependiendo de si contamos para el entrenamiento con datos específicos o genéricos de la aplicación, así como también tráfico válido o ataques. También analizamos los conjuntos de datos públicos existentes para usar en este contexto y hemos creado nuestro propio conjunto de datos capturando tráfico real sobre una aplicación en producción.

Finalmente, presentamos dos soluciones de aprendizaje automático supervisado. La primera es un enfoque clásico de discriminación entre dos clases (tráfico válido y ataques) [18]. La segunda es una solución de clasificación de una clase (one-class) para un escenario más factible cuando solo hay datos válidos disponibles. En la clasificación de una clase se asume que una de las clases puede ser correctamente modelada a partir de datos del conjunto de entrenamiento (en nuestro caso el tráfico válido) mientras que la otra clase (en nuestro problema los ataques) no puede ser modelada por falta total o parcial de muestras de entrenamiento [33]. Presentamos los resultados utilizando ambos enfoques y los comparamos con MODSECURITY configurado con el OWASP Core Rule Set por defecto, que es el conjunto de reglas más ampliamente utilizado.

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my master degree.

I would like to first say a very big thank you to my supervisors Gustavo Betarte and Alvaro Pardo for all the support, encouragement and knowledge they gave me during this endless thesis. They dedicated long hours to discuss and work together to enrich this work.

Many thanks also to Tilsor and my coworkers for the time, support and inspiration they gave me. Much of the motivation for this work comes from the tasks performed there.

I gratefully acknowledge the funding received towards my thesis from the Information and Communication Technologies for Vertical center[1] (ICT4V). I am also grateful to the funding received through ANII as this work was done in the context of the project FMV 1 2017 136337.

A very special thank you to my family for always believing in me and encouraging me to follow my dreams.

Finally a heartfelt thank you to Gabriela for all the patience and for being there with words of support and pushing me to go on. And to Nahuel for waiting there and making it possible for me to complete what I started.

[1]http://ict4v.org

Contents

Declaration of Authorship **iii**

Abstract **v**

Resumen **vii**

Acknowledgements **ix**

1 Introduction **1**
1.1 Problem statement 2
1.2 Contributions and Publications 2
1.3 Outline 3

2 Background **5**
2.1 Web applications and the HTTP protocol 5
2.1.1 HTTP protocol 6
2.2 Web application vulnerabilities 7
2.3 Protecting web application 7
2.4 Machine learning techniques 8
2.4.1 Transformation techniques 9
Bag of words 10
Term Frequency times Inverse Document Frequency 10
Information gain 11
2.4.2 Classification approaches 11
Supervised vs unsupervised 11
Multi-class vs one-class 12

3 Related work **15**
3.1 Application layer protection 15
3.1.1 SQL communications analysis 16
3.1.2 HTTP communications analysis 16
3.2 Machine learning techniques 16
3.2.1 Analyzing the whole HTTP request 16
3.2.2 Fine granularity in the analysis 17
3.2.3 Token abstraction 17
3.2.4 Type of traffic used during training 17

4 Adaptive application security **19**
4.1 Learning scenarios 19
4.2 Datasets 20
4.2.1 PKDD2007 21
4.2.2 CSIC2010 21
4.2.3 DRUPAL 21
4.3 The learning architecture 22

4.3.1 Parser . . . 23
Information decoding . . . 23
Headers filtering . . . 23
Special characters preservation . . . 24
Parsing result . . . 24
4.3.2 Tokenizer . . . 25
Classic information retrieval approach . . . 25
Expert assisted approach . . . 25
4.3.3 Classification . . . 26
Two-class . . . 26
One-class . . . 27
4.4 Limitations . . . 28
4.4.1 The parser and encoding mechanisms . . . 28
4.4.2 HTTP headers and cookies attacks . . . 28
4.4.3 Request body content type . . . 29

5 Implementation **31**
5.1 The WEKA tool . . . 31
5.2 Dataset loader and parser . . . 31
5.2.1 PreprocessCommon subsystem . . . 32
5.2.2 Loader subsystems . . . 32
PKDD2007 loader . . . 33
CSIC2010 loader . . . 33
DRUPAL loader . . . 34
5.2.3 Transformer subsystem . . . 34
5.2.4 Generator subsystem . . . 35
5.3 Tokenizer . . . 35
5.3.1 Classic information retrieval tokenizer . . . 36
5.3.2 Expert assisted tokenizer . . . 37
5.4 Two-class classifier . . . 37
5.5 One-class classifier . . . 38
5.5.1 Mahalanobis distance . . . 39
5.5.2 One-class evaluator . . . 40
5.6 MODSECURITY requests generator . . . 41
5.6.1 MODSECURITY evaluator . . . 42
5.6.2 Request generator . . . 42

6 Experimental results **43**
6.1 Datasets . . . 43
6.2 Baseline . . . 44
6.3 Scenario 1 . . . 45
6.4 Scenario 2 . . . 47
6.5 Scenarios 4 and 5 . . . 48
6.6 Discussion . . . 51

7 Conclusion and further work **55**
7.1 Further work . . . 55
7.2 Conclusions . . . 56

Bibliography **59**

List of Figures

2.1 Statistical pattern recognition architecture [30] 9

4.1 Learning architecture 22

5.1 Dataset Loader and Parser - Component Diagram 32
5.2 PreprocessCommon subsystem - Class Diagram 33

6.1 *sc*4: ROC curve generated by moving the threshold of the one-class approach using DRUPAL and DRUPAL validation datasets 48
6.2 *sc*5: ROC curve generated by moving the threshold of the one-class approach for DRUPAL dataset 49
6.3 *sc*4 and *sc*5: ROC curve generated by moving the threshold of the one-class approach for PKDD2007 dataset 50
6.4 *sc*4 and *sc*5: ROC curve generated by moving the threshold of the one-class approach for CSIC2010 dataset 51

List of Tables

4.1 Learning scenarios 20
4.2 Headers filtered during the parsing process 24
4.3 Selected features by the security specialist 26

6.1 Requests per dataset and the distribution for training and testing . . . 43
6.2 Summarized results of the ModSecurity baseline 44
6.3 Summarized results of both tokenization approaches on *sc*1 45
6.4 Summarized results of generalization of the classic information retrieval approach on *sc*1 46
6.5 Summarized results for *sc*2 47
6.6 Summarized results for the PKDD2007 dataset 51
6.7 Summarized results for the CSIC2010 dataset 52
6.8 Summarized results for the DRUPAL dataset 53

Listings

2.1 HTTP Request 6
4.1 Parsed HTTP Request 24
5.1 Generator: command line options 35
5.2 WEKA command to execute the classic information retrieval tokenizer 36
5.3 Feature selection using WEKA 36
5.4 Two-class classification using WEKA 37
5.5 One-class classification algorithm: pseudocode 38
5.6 Mahalanobis distance initialization: pseudocode 39
5.7 One-class evaluator: input format example 40
5.8 One-class evaluator: output format example 41

For my couple and son Gabriela and Nahuel

Chapter 1

Introduction

Out of the box software often does not easily adapt to the requirements posed by the users, what makes it necessary the development of custom solutions that provide support for specific business processes. The potential of web applications in this regard has allowed its use to spread significantly, the possibility of remote management and transparency from the technological platform has strongly encouraged the development and use of them. It is a known fact though that web applications are permanently being exposed to attacks that exploit their vulnerabilities.

Initiatives like the OWASP Top 10 [42] have greatly contributed to rise awareness concerning the security of web applications but have not prevented the ever increasing amount of web application (successful) attacks. For specific web applications it is the case that software vulnerabilities can only be discovered as a result of a process of trial and error performed by an attacker. In this context, attack detection techniques become necessary. These techniques involve procedures that help distinguishing between the behavior of a valid system user and a malicious agent. The identification and determination of a user's behavior should consider whether each detected event is simply suspicious or actually it is an event that is part of an attack. Attack detection techniques become crucial when determining appropriate thresholds for response actions. These types of techniques assist in aspects such as preventing attackers to identify/verify successfully the existence of vulnerabilities in applications and to minimize the number of false positives (non-malicious activity identified as such).

A technological alternative for performing attack detection is the use of a Web Application Firewall (WAF). A WAF is a piece of software that intercepts and inspects all the traffic between the web server and its clients, searching for attacks inside the HTTP packet contents (a description of the protocol is presented in Section 2.1.1). Once recognized, the suspicious packets may be processed in a different, secure way, for instance being logged, suppressed or derived to a honeypot application.

An implementation of an open source WAF that has become a *de facto* standard is MODSECURITY [58], which allows the analysis of the users requests and the application responses by enabling real-time web application monitoring, logging and access control. The actions MODSECURITY undertakes are driven by rules that specify, by means of regular expressions, the contents of the HTTP packets to be spotted.

Different vendors provides rule sets for MODSECURITY (see Section 2.3), the most widely deploy rule set is known as the OWASP Core Rule Set (OWASP CRS) [43], for handling the most usual vulnerabilities included in [44]. However, an approach only based on rules also has some drawbacks: rules are static and rigid by nature, so the OWASP CRS usually produces a rather high rate of false positives, which in some cases may be close to 40% [22]. Rule tuning is a time consuming and error prone task that has to be manually performed for each specific web application. In traditional networks firewalls and IDS, the approach based on rules has been successfully complemented with other machine learning-based tools, anomaly detection and other statistical approaches which provide higher levels of flexibility and adaptability. Those approaches take advantage of sample data, from which the normal behavior of the web application can be learned, in order to spot suspicious situations which fall out of this nominal use (anomalies), and which could correspond to on-going attacks.

1.1 Problem statement

The main objective of this thesis is to improve web application attack detection by using machine learning techniques. More specifically, this dissertation addresses two main problems, summarized as follows:

- In order to protect web applications by using machine learning techniques, data is needed to train the models. In classic supervised classification techniques this data has to be labeled in order to be used for training. Different approaches could be used depending on the data at hand. We will propose different scenarios to classify the approaches with the pros a cons to its use in practical cases.
- As already mentioned, MODSECURITY combined with the OWASP CRS requires rule tuning tasks before being able to protect web applications. We will propose machine learning models to complement the rule based approach in order to:
  - improve the OWASP CRS attack detection and
  - diminish the false positive generated by the OWASP CRS to reduce the initial time consuming tuning task.

1.2 Contributions and Publications

There are almost no solutions based on machine learning techniques to protect web applications using WAF. Our contributions addressed this issue by validating the use of classic machine learning techniques for this specific application. We also validated, in the professional community[1], the need to apply these type of techniques to improve the difficulties they are having with the use of rules.

We propose a characterization of the problem by defining different scenarios depending if we have valid and/or attack data available for training. These scenarios allow to have different methods to deploy the models, varying from the more complicated one where data with valid and attack requests of the application to be protected is needed to generic methods where only one of attack or valid data is required.

We then put forward the implementation of two approaches: first a two-class approach for the scenario when valid and attack data is available; and second a one-class solution when only valid data is at hand. The experimental results of both approaches are presented and discussed showing that it is feasible to apply these techniques in this context.

[1]After having private communications with one of the project leaders of the OWASP CRS, we were invited to present our results at the OWASP CRS Summit held in London during the 2018 OWASP AppSec Europe Conference

All these contributions were published in the following articles:

- Betarte, G., Giménez, E., Martínez, R., & Pardo, Á. (2018). Machine learning-assisted virtual patching of web applications. arXiv preprint arXiv:1803.05529.
- Martinez, R. (2018, October). Enhancing web application attack detection using machine learning. In 2018 8th Latin-American Symposium on Dependable Computing, Student forum.
- Betarte, G., Pardo, Á., & Martínez, R. (2018, December). Web Application Attacks Detection Using Machine Learning Techniques. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 1065-1072). IEEE.
- Betarte, G., Gimenez, E., Martinez, R., & Pardo, A. (2018, December). Improving Web Application Firewalls through Anomaly Detection. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 779-784). IEEE.

As we have already mentioned, the profesional community identifies the need for this type of research, so we were invited to present our work in the first OWASP CRS Summit held in London during the 2018 OWASP AppSec Europe Conference [46]. At that conference, we presented the results of this thesis and our proposal of alternative mechanisms to integrate MODSECURITY and the OWASP CRS .

1.3 Outline

The structure of the rest of the thesis is as follows: Chapter 2 provides a primer on web application, web application security and protection mechanisms. It also discusses the learning techniques we have applied. Related work is discussed in Chapter 3. In Chapter 4 we present the analysis and design of the learning framework we have used to carry out the experiments. Chapter 5 presents the implementation of the framework. The experimental results are described and discussed in Chapter 6. Further work and conclusions are presented in Chapter 7.

Chapter 2

Background

This section provides a concise primer on web application, including the HTTP protocol, security issues and protecting mechanisms and briefly discusses the machine learning techniques that have been used in this work.

2.1 Web applications and the HTTP protocol

A web application is a client-server software where the application itself is hosted in a web server and the client is, usually, a web browser. Web applications are build to provide functionalities to the client. This means that a key characteristic of web applications is the interaction with the client. Typically, it is expected that web application receive input from the client, processed by the server, and then returned the results back to the client. A feature that every web application must comply with is that the client-server communication is over the HTTP protocol.

Web applications are designed using a n-tier software architecture. The most classic architecture is the 3-tiered application with: i) a presentation tier responsible of the interaction with the user, which usually runs in the web browser; ii) a business tier that implements the business logic and runs in the web server and iii) a storage tier that is usually a database. As web application have evolved and they brought complex functions to the clients, architectures had to evolved too, usually splitting the business or storage tier in different tiers or components with more specific functionalities. For example, some data could be stored in a SQL database, but documents are stored in a file system or a NoSQL database and the users are persisted in a directory service using the Lightweight Directory Access Protocol (LDAP).

Other important change in the architecture is due to the need of interconnect different system with each other, not only within the organization itself, but also with third party providers or partners. Most of these interconnections promote the use of technologies such as *SOAP* web services [15] or *REST API* [53]. The web services run as web applications, where the client is other business tier and they exchange information using XML content over the HTTP protocol. On the other hand, REST API behaves like web services, but the main difference is that the information is exchange using JSON [9]. This interconnections added new impacts from a security perspective, since organizations not only have to take care of the potential attacks to their own infrastructure, but it could also may have a security breach throughout a problem of one of this third party interconnected organizations.

As most of these tiers execute on different platforms, idifferent programming languages are used for each tier. For example, the presentation tier could use HTML and *Javascript* whereas the storage tier could use SQL. Typically, the business logic is implemented using programming language such as PHP or Java which executes in the web server, and dynamically generates code for each of the other tiers. For instance, if a client ask the server to return his profile, the application logic will generate an SQL query to get the profile info, then with the results the application logic will generate a response coded in HTML and *Javascript* which will be executed by the web browser.

2.1.1 HTTP protocol

As we mentioned before, client-server communications in web applications are performed using the Hypertext Transfer Protocol (HTTP). HTTP is a request-response message protocol, the widely use version is 1.1 [25]. The new version HTTP/2 from 2015 focus on improving the use of network resources. HTTP is a semi-structured text protocol that function as requests-response protocol. Each communication is started by the client by sending an HTTP request. The request is processed by the server which returns a HTTP response to the client.

```
1  POST /user/login?destination=search%2Fnode%2Flimite%20de%20cursadas
       HTTP/1.1
2  Host: www.fing.edu.uy
3  Connection: keep-alive
4  Content-Length: 200
5  Cache-Control: max-age=0
6  Origin: https://www.fing.edu.uy
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
       (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36
9  Content-Type: application/x-www-form-urlencoded
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
       webp,*/*;q=0.8
11 Referer: https://www.fing.edu.uy/user/login?destination=search%2
       Fnode%2Flimite+de+cursadas
12 Accept-Encoding: gzip, deflate, br
13 Accept-Language: es-ES,es;q=0.8,en;q=0.6
14 Cookie: SESS77d4c056e4744b899299483351de0e63=2
       pp35g1jp42mjm9216g04m1bm2; _pk_ref.1.af1c=%5B%22%22%2C%22%22%2
       C1489595176%2C%22https%3A%2F%2Fwww.google.com.uy%2F%22%5D;
       has_js=1; _pk_id.1.af1c=9bd9ba450973189b
       .1487191441.5.1489595186.1489595176.; _pk_ses.1.af1c=*
15
16 name=jose.perez&pass=**********&form_build_id=form-
       Kh0_uekNLVtroZXTLGt-71ZmRx3TJEwP7jz3K2DWQI0&form_id=user_login&
       securelogin_original_baseurl=https%3A%2F%2Fwww.fing.edu.uy&op=
       Iniciar+sesi%C3%B3n
```

LISTING 2.1: HTTP Request

The Listing 2.1 presents an HTTP Request. Line 1 is called the request line and it is composed of three fields: the requests method, the request URI and the HTTP version. The request method corresponds to the action that the server has to execute (in this example the POST represents the push of information). The URI identifies the resource being executed. Finally the field HTTP version indicates the version of the protocol used by the client.

Lines 2 to 14 are the HTTP Headers, each header line starts with the header *name*, followed by a colon (:) and finally the value. Each header may have more than one value, each value is separated by a comma or semicolon. They are mostly used to exchange information between the client (usually the browser) and the server. In Line 14 there is a special header that is called the *Cookie*. The HTTP is a stateless protocol and cookies was an addition to the protocol introduced in the RFC 6265 [4] in order to exchange a token between the client and the server to implement the session. The empty line after the headers (line 15) represents the end of the headers section.

Finally, line 16 contains the request body. This is an optional part of the requests and its formats depends on the HTTP method and the *Content-Type* header. The *Content-Type* indicates the nature of the date included in the body, by specifying the type and subtype of data. Most common content types used in the HTTP requests by the browsers are: *application/x-www-form-urlencoded*, *multipart/form-data*, *text/xml* and *text/json*.

2.2 Web application vulnerabilities

There are many types of vulnerabilities, for example Common Weakness Enumeration (CWE) [12] is a list of software weaknesses developed by MITRE with the support of the community. Its main objective is to provide a common language to describe software security weakness. In this work we are going to focus on those vulnerabilities related to input validation. When the software does not properly validate inputs, an attacker is able to send crafted payloads that are unexpected for the application that may affect the application control flow or data flow. The two most important vulnerabilities related to improper input validation are Injection and Cross Site Scripting (XSS).

The most critical web application security risk defined by the OWASP Top 10 2017 [44] is *Injection*. Injection occurs when data sent by the user to the application is used, without proper validation, to construct an instruction that an interpreter executes in the backend producing unexpected (by the designer of the application) results. There are different types of injections depending on the interpreter being exploited: SQL, LDAP, XPath, NoSQL queries, OS commands, XML parsers, SMTP headers, among others. A SQL Injection takes place when an attacker sends SQL code as input to the application and it results in a query that returns, in unauthorized manner, sensitive data.

In the case of a XSS, the application receives malicious code (usually JavaScript or HTML) from an attacker and sends it to other users (victims) without proper validations. The attacker's input executes scripts in a victim browser to, for example, redirect users to malicious web sites or hijack user sessions.

Injection and XSS have always been include in the OWASP Top 10 since the first publication in 2007. Nevertheless applications still suffer from these type of vulnerabilities. Thus, input validation is critical for software security. One such validation consist of verifying and filtering all data that flow to the system before they are effectively used. These procedures usually are not introduced at the design stage of applications leaving then unattended vulnerabilities that might be critical, specially to web applications. When data is processed without proper validation an attacker could lead the system to unpredictable states and exploit this for his own benefit. This data may also produce unexpected results that could be analyzed by the attacker to infer further information to proceed with his activity.

2.3 Protecting web application

The principle of *in-depth security* means that security mechanisms are layered around the system being protected so if an attack bypass one mechanism there exist other mechanisms to provide the needed security. When concerned with the security of a web application, the outer layer to be protected is the organization's network perimeter. Typically, network firewalls are deployed to protect in/out traffic. As to the network itself, it is common practice to use an Intrusion Prevention System (IPS), for instance, to analyze network traffic in order to detect potential threats. Additionally, it has become a good security practice to deploy a Web Application Firewall (WAF) to analyze the request/response flow through the communication channel to identify attacks that exploit vulnerabilities proper of web applications. In the general case, both traditional firewalls and IPS inspect traffic at the network layer. In contrast to traditional network firewalls, WAF are designed to perform packet inspection at the application layer of the OSI Model [65] to prevent web application attacks. Secure communications using SSL and TLS are done by the WAF so our work focus on the HTTP protocol analysis.

A WAF often supports different security model configurations: normally it makes it possible to enforce both positive and negative security models. A positive security model only allows to pass known good traffic, all other traffic is blocked. A negative one allows to pass all traffic except what is known to be malicious. ModSecurity is a *de facto* standard WAF in the open community, it is open source, flexible and extensible. Large organizations like Verizon [59] are currently using this technology to protect large amount of applications. It can be used to control, monitor and log web traffic from and to web applications and has two working modes: detection and prevention. In the first mode, logs are generated for every detected potential attack and it is used to monitor specific rules. Normally this mode is used, when adding new rules and monitoring for false positives. The second mode is when the WAF is really useful: by correctly configuring rules and directives it is capable of blocking potential malicious web traffic to and from applications. The core of MODSECURITY implements a flexible rule engine [55]: variables, operations and actions can be defined using the MODSECURITY Rule Language. These rules can be applied in every application transaction. The rule engine could be configure to load a set of rules that it will apply in the defined order for every application transaction. Each rule in the rule set could inspect for a specific problem on the transaction and also specified a specific action in case that it match. Different vendors provides their own rule set or the MODSECURITY administrator could implement its own.

To detect and prevent the exploitation of well-known and common vulnerabilites the Open Web Application Security Project [42] (OWASP) has defined a generic rule set that is known as the OWASP Core Rule Set [43] (OWASP CRS). The OWASP CRS is widely deployed in large organizations as Akamai, Azure, CloudFare, Fastly and Verizon. The goal of the OWASP CRS is to provide a set of generic attack detection rules that when fed to MODSECURITY provide a base level of protection for any web application. The OWASP CRS implements a negative model, where the rules are designed to detect known attacks patterns.

The last Gartner's Magic Quadrant for Web Application Firewalls [49], reviews and ranks several proprietary WAF, among others Akamai, F5 and Imperva. In that report it is remarked the rare and still unproven use of machine learning techniques to leverage the detection capabilities of those technologies.

2.4 Machine learning techniques

We have experimented with different machine learning techniques in this work, but all of them could be classified as what is known as statistical pattern recognition system. The objective of a statistical pattern recognition system is to classify data

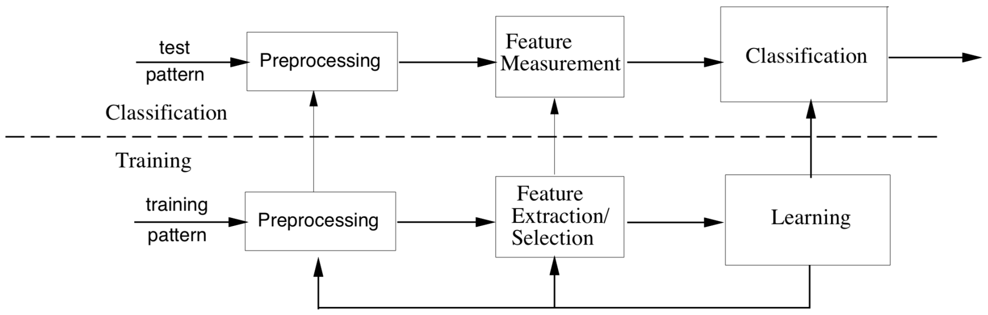

FIGURE 2.1: Statistical pattern recognition architecture [30]

automatically. To achieve this objective, instances of the observed data are used to train a model which later (in the classification mode) will classify new instances. Depending on the data used during training and if the data is labeled or not, the system can be categorized in different ways. In section 2.4.2 we present some of these categorization approaches.

In [30], Jain et all presented a review on statistical pattern recognition and define a general architecture of a pattern recognition system. As shown in Figure 2.1, the system is operated in one of two modes: training mode (where training instances are used to adjusts the models parameters) and a classification mode (where new instances are classified by the system).

The preprocessing module is the first module of the pipeline and is responsible for cleaning the raw data to remove noise. Then the feature extraction/selection module transformed the resulting data into a vector where each position correspond to the value of an attribute or feature that is extracted from the input. This module is also responsible of doing normalization of the data. Finally the learning module train a classification model using the extracted features. The training process consists in adjusting the parameters of the model in a way that better partition the feature space.

After training the learning model, the system can be used to classify new instances using the classification mode. Once again, the first module of the classification mode is the same used in the training mode. The second module is responsible of transforming the new instance into a vector of numbers. In the case where normalization was done during the training mode, this module has to apply the same normalization methods and parameters. Finally, the classifier will classify the instance using the parameters adjusted in the training mode.

In what follows we describe some transformation techniques and classification approaches that were used in our work.

2.4.1 Transformation techniques

As stated before, one of the major steps in a pattern recognition system is the feature extraction/selection to transform the raw data into a set of numbers. There exists different techniques, in our work we had applied classic text classification techniques. In what follows, we will introduce the transformation techniques used during this work.

Bag of words

A classic feature extraction technique used in information retrieval system is the *Bag of Words* model. In this model, a text document is transformed into a vector of numbers, where each position of the vector corresponds to a word. The value of the position corresponds to the times the corresponding word appears in the text.

One important concept of this model is how to split the text documents into tokens that represents the words of the model. Most of the *bag of words* algorithms use a *tokenizer* to implement the way to split the text. Two classic implementations are the ngram and word tokenizer. The first one splits the text into tokens of a fixed length *ngrams*. For example, ngrams of size 1 represents each letter of the text. On the other side, the word tokenizer uses a set of characters known as the *delimiters* and spit the text each time one of this characters appears. For example to split an English document into its words, usually the space and punctuation signs are used as the delimiter characters.

The set of words (a.k.a. features or attributes) used in the *bag of words* model is called the vocabulary and it can be fixed o learned during the training phase. In a fixed vocabulary approach, usually an expert uses its knowledge to define the set of features that better characterize the problem. Using this set of features, the *bag of words* model captures from the input text the values of the features defined by the expert. The second approach corresponds to the case where the algorithm defines the vocabulary by it self. In this case, it uses the training set to learn the features to be used. This usually results into too many words, generating sparse vectors. In order to better process this vector, after the feature extraction, usually a feature selection algorithm is used to reduce the size of these sparse vectors.

Term Frequency times Inverse Document Frequency

Information retrieval systems are usually used to classify documents depending on its topic [36]. In order to do so, is necessary to find the special words that better characterize the topic. However, until we have made the classification, it is not possible to find these words.

It is common to imagine that the most common words in the document set are representative. This is usually not true, as normally the most common words such as "the" or "and" does not carry any significance themselves. In fact, indicators of the topics are usually relatively rare words. For example in order to classify documents about soccer, words like "goalkeeper", "offside" or "penalty" are good indicators, but extremely rare words as "notwithstanding" or "albeit" do not tell anything useful. The challenge is to discriminate between rare words that carry any significance from those that does not.

One approach for this challenge is the use of the *Term Frequency time Inverse Document Frequency* (TF-IDF) measure. If we have a collection of N documents, lets defined f_{ij} to be the *frequency* of the word i in the document j. Then the term frequency (TF_{ij}) is defined in Eq 2.1.

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}} \tag{2.1}$$

In other words, TF_{ij} is 1 for the most common word in document i, and is a fraction of 1 for the rest of the words, depending of their frequency in the document.

If the word i appears in n_i of the N documents of the collection, the IDF_i is defined in Eq 2.2.

$$IDF_i = \log_2 \frac{N}{n_i} \tag{2.2}$$

Then the TF-IDF score of the word i in the document j is calculated as: $TF_{ij} * IDF_i$. The words with highest TF-IDF score are usually the words that better characterize the documents. We shall came back on how to combine TF-IDF and the *Bag of Words* model in section 4.3.2.

Information gain

Usually in pattern recognition systems, a set of features is generated as a result of applying the feature extraction techniques. But not all features are relevant to making predictions, in fact, some of them could add noise to the classification algorithms. The process for selecting the features that better characterize the documents is the feature selection phase.

One common approach for feature selection is the information gain algorithm [41]. The objective is to evaluate how the entropy of the class changes if the feature is selected. Lets say that the class A has an entropy $H(A)$ and the feature selected is f, the information gain score is defined as Eq 2.3.

$$InfoGain(A, f) = H(A) - H(A|f) \tag{2.3}$$

$H(A|f)$ is the conditional entropy of class A given the feature f. The information gain score varies from 0 (no information) to 1 (maximum information). The features with major score will be kept and the ones with lower score could be removed.

2.4.2 Classification approaches

There are many different ways to categorize machine learning classification approaches depending on the objective. This section introduce two categorization mechanisms: one based on the type of labeled data needed during training (supervised vs unsupervised) and the other one depending on the number of classes used during training (multi-class vs one-class).

In this work we experiment with two supervised approaches: multi-class approach and one-class classification.

Supervised vs unsupervised

During training, classification algorithm use data to generate the model. The learning method depends on the data that we have at hand and how it is labeled.

In supervised learning algorithms [34] the training data has to be encoded as pairs of instance-output, where the instance is an example of data and the output corresponds to the label of the class that the instance belongs to. The objective of this learning method is to estimate the parameters of a function, which objective is to classify an instance into its corresponding class.

In unsupervised learning algorithms, training data is not associated to an output. The objective of these algorithms is to discover the structure, relationships and groups between the training instances in order to generate classes. Finally during classification, the algorithm classify new instances depending on the closest class that were defined during training.

In this work we used the supervised learning method, where all training instances were labeled with its corresponding class.

Multi-class vs one-class

Another way to categorize classification algorithms depends on the number of classes that we have during the training phase. In particular, we will focus on two classic classification approaches that were used during this work: multi-class and one-class classification.

Multi-class classification [34] uses a dataset that contains instances of all possible classes that the algorithm could predict. For example, in this work will be used a dataset where one-class defines the normal traffic and one or more classes define what is understood as attacks. Several algorithms implement this approach, in what follows we will present three algorithms: Support Vector Machine (SVM), RANDOM FOREST and K-near neighbor (KNN).

On the other hand, one-class classification also known as Anomaly detection method [33], has been used to address the problem of having instances of only one of the classes. In this approach, instead of modeling all classes, the algorithm focus on modeling one of the classes (the one that the instances belongs to) and labeling as anomalies those instances that move away from the model. The main challenge with this approach is the definition of the threshold from which an instance is considered anomalous. The definition of this threshold is critical in order to have good performance indicators, in particular to prevent the generation of false positives.

Support Vector Machine The objective of the SVM algorithm [11] is to find the best hyperplane that separates two data classes. For example, in the linear case where we have two features, the objective of SVM is to find the straight line in the plane that better separates the instances of each class. If we increase the number of features, we move from the plane into the hyperspace, and in this general case, SVM tries to find the best hyperplane that separate both classes. In general, the training examples that are closest to the hyperplane are called support vectors. Support vectors are the training instances that lay on the margin on both sides of the hyperplane. The notion behind SVM is to maximize the margin between the instances of each class to the hyperplane. The margin is calculated as the minimum distance to the hyperplane for the positive instances plus the minimum distance to the hyperplane for the negative instances. In the case where the instances could not be completely separable, the algorithm generalizes, by allowing to have a number (define by the specialist) of false positives. In the case, where we want to use SVM to classify with more than two classes, we have to complement the algorithm with a combination strategy that could be: one vs all and one vs one. In the one vs all strategy, we generate one model for each class where we train this model using the instances of the class and the rest of the instances as the second class. The one vs one strategy generates models for each pair of classes, in the case of n classes, this models generates $n * (n-1)/2$ models.

RANDOM FOREST A RANDOM FOREST is a meta classifier that fits a number of decision tree classifiers on various sub-samples of the training dataset and uses averaging to improve the predictive accuracy and control over-fitting. A decision tree is a tree that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. The leaves of the tree represents the output class. One interesting characteristic of decision trees is that can be easily translated into rules, as they combine the values of the features to calculate the corresponding classification class.

K-nearest neighbor The KNN algorithm is an instance-based learning algorithm. These algorithms are lazy learning meaning that they delay the model construction until classification is performed. The KNN algorithm is based on the principle that instances of the same class will be closed to each other. So classification is perform by measuring the distance of the new instance to the k nearest instances from the training dataset and calculate the output class by evaluating the class of these instances.

Chapter 3

Related work

There exist in the literature several works related to the application of machine learning techniques to detect information systems attacks. As we have already mentioned, our objective is to improve the capabilities to detect web application attacks on web applications. In section 2.3, we presented several ways to protect applications focus on different layers based on the *defense in depth* principle, we are going to focus on the application layer protection mechanism.

In what follows, we present different approaches to web application protection at the application layer as well as the machine learning techniques used.

3.1 Application layer protection

In this context, we refer to application layer protection to techniques which analyze the communications at layer 7 (from the OSI Model) in order to protect a web application. In general, web applications are designed using three major tiers: a thin client (web browser), the business logic and a datastore. Application layer protection may be applied to the communications between any of those tiers. The type of application layer protection mechanisms can be classified depending on the communication being analyzed. In the case that we analyze the communications between the business logic and the datastore, we usually analyze SQL sentences, in the other hand, if we analyze the communication between the client and the business logic, the protocol to be analyze is the HTTP.

These application layer mechanisms are also known as *Virtual Patching*. Virtual Patching [**barnett2009waf**] is not a specific web application term and it may be applied to any application layer protocol however currently it is associated to WAF. It is defined as: *"A policy for an intermediary device that is able to identify and block attempts to exploit a specific application vulnerability."* Virtual Patching implementation could be done using a positive or negative security model. Usually positive security models are implemented using rules, but, as we have already mentioned in section 2.3, implement negative models using rules may generate high amount of false positives. There exists several proprietary implementations of WAF technology. They have been reviewed and ranked in a recent report of the Garner Magic Quadrant [49], where, in particular, one can find the following statement concerning the reviewed technology: *"Use of machine learning is rare and often still unproven"*. The lack of application of these techniques in WAF were a strong motivation for this work.

In what follows we will present application layer protection mechanisms using machine learning techniques organized depending on the protocol being analyzed.

3.1.1 SQL communications analysis

In recent years work has been reported [8, 31, 32] concerning the use of database firewalls (DBF) as remediation tools. In all of these works anomaly detection techniques have been applied to analyze the flow of SQL sentences from the business logic tier of the application to the database with the objective of detecting (only) SQL injection attacks. As a DBF can only inspect the traffic between the business logic and the database server it is not capable of detecting attacks that are not directed to the data layer of the application. In our work, we focus on detect attacks generated by malicious input to the web application. For this reason we are going to work analyzing the communications between the client and the business logic.

3.1.2 HTTP communications analysis

One of the seminal works in anomaly detection techniques was developed by Wang and Stolfo [61], where they introduced PAYL, a payload-based anomaly detector for intrusion detection. Their approach consists in comparing the byte frequency distribution in network packets using (a simplified) Mahalanobis distance. Accordingly with the current attack vectors of that time, their objective was focused on protecting an internal network from packets carrying worms or other forms of malware. Consistently, their model works on TCP packets, that is, on the network OSI layer. As the byte distribution heavily depends on the application protocol, they divide the stream into smaller groups of packets, grouping them by destination port and by payload length. They applied PAYL to the traffic to port 80 (HTTP) of the 1991 DARPA IDS dataset, obtaining excellent detection results. However, their results relate to the attack vectors of 1999. Indeed, Ingham and Inoue report in [29] that the DARPA IDS 99 dataset only contain four web attacks. Moreover, it does not contain any of the OWASP TOP 10 attacks [40][1]. As a consequence, while [61] reports on a 100% detection rate for traffic on port 80 of the DARPA IDS dataset based on byte frequency distribution of the unparsed TCP payloads, performing the same analysis on the CISC2010 dataset yielded a detection rate of only 40% of the true anomalous cases. This low performance has been also independently reported by Ingham and Inoue in [29]. The reason is that token distribution varies from one HTTP field or web application parameter to another, which are spread along several TCP packets. Protecting web applications requires shifting the analysis to the application OSI layer, focusing on HTTP requests.

3.2 Machine learning techniques

Several design decisions have to be taken when implementing a machine learning system. In what follows we present previous works that discuss the main decisions related to our problem.

3.2.1 Analyzing the whole HTTP request

In [35], Kruegel and Vigna propose an anomaly detection approach where the attributes of the model capture information about the parameters of the URI. They focus on characteristics like the parameter length, the order in which they appear

[1]This is not surprising, as web applications arose during the 2000, and SQL injection was first reported in December 1998 [50]

and even generate a probabilistic grammar for each parameter. In both of our approaches we analyze the hole request information and not only the URI parameters. We think this provides further guarantees because all fields are subject to malformed input attacks, as illustrated in the PKDD2007 dataset [47].

3.2.2 Fine granularity in the analysis

Among the works focusing on the application layer, a few of them consider each HTTP request as a single unstructured document to be analyzed and classified [2, 24]. This approach is prevalent when focusing on the current state of the art in attacks. Other works perform some partial parsing of the HTTP structure, for example in [19, 35]. However, in this latter works, the parsing is reduced to the structure of the URI, splitting the URI into the URL and the web application parameters. In our work, we use the whole HTTP as a text document, after analyzing its structure. We parse the HTTP message analyzing the symbols that are part of the protocol in order to separate these symbols from attacks symbols.

3.2.3 Token abstraction

The contents of each request field is a string, so most of the anomaly detection approaches make use of document classification, NLP and information retrieval techniques. As content languages may significantly vary from one application to another (and even from one HTTP field to another), in most of the previous work tokens are just n-grams [3, 13, 35, 57, 60, 61, 63]. An n-gram is usually a sequence of consecutive characters, even though [3] shows that using n characters which are m positions away one from the other may also be an alternative. Several authors do not directly work on the tokens themselves, but rather on some simplification of them. In [13], Corona et al. abstract away numbers and alphanumeric sequences, representing each category with a single symbol. Torrano, Perez and Marañón [57] present an anomaly detection technique where instead of using the tokens themselves, they use a simplification that only considers the frequencies of three sets of symbols: characters, numbers and special symbol, as well as the list of special symbols used in each field. In general, what is counted is the number of symbols after applying some forget function on the set of tokens, which abstracts away irrelevant differences. Removing information simplifies and speeds up the analysis by reducing the combinatory of possible tokens. It also reduces the risk of overfitting. However, if too much information is removed from the tokens some attacks may become indistinguishable from correct behavior, so an adequate compromise shall be found.

In our experiments, we work in two approaches: a classic information retrieval approach and an expert assisted. The first approach uses the bag of words model to split the information contained in the HTTP requests. This process is executed after the parsing which differentiates symbols from the HTTP protocol with attacks payloads. In the expert assisted approach we use the knowledge of a security expert in order to identify tokens that are usually part of attacks payloads. In this case, the bag of words model uses these tokens as the dictionary.

3.2.4 Type of traffic used during training

The pattern recognition systems could be classified depending on the type of traffic that it is required for training the model. Two major approaches in supervised systems are: multi-class or one-class (also known as anomaly detection). The first approach requires traffic labeled with all the classification classes during training. On the other hand, the one-class approach models the normal behavior of the application by learning patterns from positive instances. When a new instance arrives, this instance is classified by the model resulting in a score. This score, that is not necessary a probabilistic one, is compared to a threshold to determine whether this new request belongs to the model. This approach, which could also be called *Anomaly*, *Outlier* and *Normality* detection, as far as we know has never been applied to web application protection before.

In the PKDD2007 challenge [2], the objective was to classify web application requests using a multi-class approach, in which the training dataset contains labeled HTTP requests which were classified into normal ones and attacks. Several solutions were presented [20, 24, 47]. In particular, Gallagher et al [24] presented a solution that used classical techniques of information retrieval. Even if their solution performs quite well it is restricted to detect only known attacks and relies on a labeled dataset which contains dozens of thousands of requests. In our case, we present two solutions: the two-class approach and a one-class approach.

The two-class approach is an extension on the multi-class approach presented by Gallagher et al, where we include a pre-processing phase where we used the knowledge of the HTTP protocol in order to parse the requests before applying the bag of words model.

The one-class approach only requires a training dataset that contains normal traffic. In addition to this, we have shown that the one-class approach works quite well even in the absence of a dataset proper of the application being protected. The scenario in which the WAF is trained with a dataset containing requests from applications other than the one being protected our results outperform MODSECURITY with the OWASP CRS out of the box. In [51], Raïssi et al conclude, based on the results of the PKDD2007 challenge and a feedback survey written by the challengers, that using machine learning techniques to detect web application attacks requires to involve the security experts earlier in the knowledge discovery process. In the one-class approach we have pursued, the feature selection phase uses the knowledge of the security expert to identify the tokens to be considered in the model construction.

Chapter 4

Adaptive application security

MODSECURITY is usually configured to work using a negative model because defining the rules that describe normal behavior of the application in real life is an almost impossible task. The problem with this operational mode of the WAF is the high number of false positives that are usually generated and therefore the amount of tuning needed during the learning phase. One of the main objectives of the research reported here is to make MODSECURITY behavior more flexible when used in a negative model by using machine learning techniques to adapt its (defensive) behavior to that of the application that is protecting. Additionally, depending on the operational context of the application to protect one may consider alternative learning scenarios. The ideal situation of having available a labeled dataset of application requests that represent valid and attack behavior of a specific application is not always possible so we have investigated different scenarios that now we proceed to discuss.

4.1 Learning scenarios

Table 4.1 presents the different scenarios that have been considered. The scenario *sc*1 corresponds to the idealized situation where real application traffic is available which has been tagged discriminating normal (valid) requests from attacks. In what follows we are going to present two different approaches of supervised two-class classification techniques and the results obtained on this scenario.

The scenario *sc*2 represents the case where real valid traffic (obtained from valid requests to the application) is available and the requests labeled as attacks have been collected from generic attacks. In this context, a dataset composed of generic attacks refers to attack requests to other applications and not to the application being protected. In other words, we can construct a generic attack dataset by capturing attacks to different applications and it could be improved when new attacks are identified. The objective of this scenario is to be able to protect an application having, on the one hand, valid traffic of the application (which could be generated as part of an acceptance test), and on the other hand a generic attack dataset. This scenario allows to apply the two-class approach requiring only valid data of the protected application. In Section 6.4 we present the results obtained on this scenario.

In the next scenario, *sc*3, both valid and attack traffic is collected from different sources of examples of valid and attack requests. Since tagged valid and attack requests are available in the three cases it is possible to apply classic supervised two-class techniques to classify the requests. Due to the particularity of the datasets we had available we were not able to carry out experiments on a *sc*3-like scenario.

In the scenario *sc*4 only valid requests are known, no requests tagged as attacks are available. We believe that this is a quite realistic approach, where valid traffic could be collected, for instance, from the result of performing functional testing of

Scenario	Valid Requests	Attack Requests	Scenario Type
sc1	Real Traffic	Real Attacks	Two-class supervised
sc2	Real Traffic	Generic Attacks	Two-class supervised
sc3	Generic Traffic	Generic Attacks	Two-class supervised
sc4	Real Traffic	No Attacks	OneClassClassification
sc5	Generic Traffic	No Attacks	OneClassClassification
sc6	No Traffic	Artificial Attacks	OneClassClassification

TABLE 4.1: Learning scenarios

the application. This scenario is similar to *sc*2, but it does not need to construct a generic dataset of attacks.

Scenario *sc*5 generalizes scenario *sc*4 where only valid data is at hand, but in this case, the valid data corresponds to requests of other application, not the application being protected. This scenario allows to have a trained model for fast deployment without having to even do a functional testing on the application being protected. We have pursued a supervised one-class classification approach to handle both scenarios, the results obtained are described in Section 6.5.

The last scenario, *sc*6, the only information at hand are examples of attack requests. This case can be grasped as conceptually equivalent to working with the OWASP CRS, where the rules capture the behavior of different types of attacks. We did not elaborate on this scenario, but we plan to study this approach.

4.2 Datasets

One of the major challenges when working with machine learning techniques is to have the right dataset when performing the experiments. Since our main objective is to improve MODSECURITY, the first requirement to the dataset is that each instance corresponds to a complete HTTP request including both its header and its body. In addition, as we are working with supervised classification approaches, all of the instances must be tagged or labeled at least in these two classes: Valid and Attack.

As far as we know, there exists only three public datasets that comply to these requirements: 1998 DARPA Intrusion Detection Evaluation Data Set [14], the 2007 ECML and PKDD Challenge (PKDD2007) [2] and the 2010 CSIC dataset (CSIC2010) [28].

The 1998 DARPA Intrusion Detection Evaluation Data Set was discarded for the following reasons: first of all it was generated recording network traffic (not only web application traffic); it was recorded more than two decades ago and the HTTP traffic changed significantly and there where very few web application attacks (Indeed, Ingham and Inoue report in [29] that only contain four attacks).

Even thought the other two datasets meet both requirements and the data is exclusively about HTTP requests, they have two disadvantages: they are quite old (given the evolution of web applications in the last decade) and have been artificially crafted. In order to validate our approach with recent and real life requests we have generated a dataset, called here DRUPAL, based on the real traffic to the public Website of Facultad de Ingeniería - Universidad de la República del Uruguay.

So, classification experiments have been performed on three different datasets that are briefly described in what follows.

4.2.1 PKDD2007

In 2007 the 18th European Conference on Machine Learning (ECML) and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), put forward a challenge on Analyzing Web Traffic. The challenge goal was to construct an algorithm based on machine learning techniques to perform multi-class and contextual classification and isolation of the attack patterns. In our case, as we are working on a two-class classification problem, we are going to merge all instances classified as different attacks to one class called *Attack*. As part of the challenge it was provided a dataset which contained full requests that where labeled in valid traffic and seven different types of attacks. The dataset contains 35.006 requests classified as valid and 15.110 requests classified as attacks.

The PKDD2007 dataset was generated by recording real traffic which was then processed to sanitize the information. This masking process consisted in renaming every URL, parameter names and values with randomly generated strings. During this process no consideration were taken when masking the same information. In other words, two requests with the same URL and parameter names where transform in two different requests. This masking process had a negative impact on the results that will be explain in Section 6.5.

4.2.2 CSIC2010

This dataset was developed at the "Information Security Institute" of the Spanish Research National Council (CSIC) in 2010 to test web application attack protection systems. This dataset also contains full requests and the instances are labeled Normal Traffic and Anomalous Traffic. It was automatically developed based on real request to an e-commerce application. The dataset contains 36000 valid requests for training, other valid 36000 request for testing and 25000 requests of anomalous traffic.

In order to generate the anomalous traffic there were used tools such as Paros (which later became OWASP Zed Attack Proxy (ZAP) [45]) and w3af [52]. In addition to that, some valid requests were modified with typos errors in parameter names. Using this tagging mechanism, the anomalous traffic does not necessary corresponds to web application attacks, but it could also be requests generated by the crawler used by the different tools or typos errors in the parameter names. Unfortunately, we do not know how real attacks and anomalous traffic distribute in this dataset. However, this dataset seems to be a reference in the ModSecurity community, as may be understood from a note written by Christian Folini[1] who is the project leader of the OWASP CRS project.

4.2.3 DRUPAL

In order to experiment with a real life application, based on real requests and real attacks, we crafted a dataset by capturing incoming traffic to the public Website of Facultad de Ingeniería - Universidad de la República del Uruguay [2] which is based on the Drupal portal [17].

This dataset was developed using an instance of ModSecurity which recorded the incoming traffic for a period of six days to the website. We specially marked the

[1] https://github.com/SpiderLabs/owasp-modsecurity-crs/issues/1016#issuecomment-366602493

[2] This dataset is not public, but it is available on demand to other researchers by writing to the authors

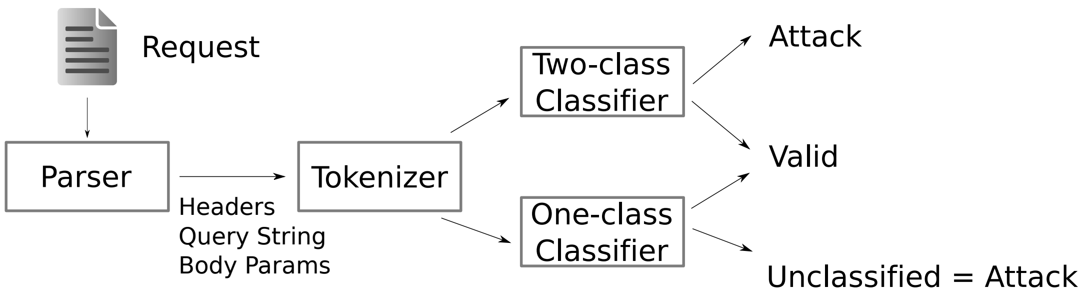

FIGURE 4.1: Learning architecture

requests that were generated to the Drupal portal, discarding all other requests , e.g. access to static resources as images, PDF among others.

The instances of this dataset where labeled as Valid and Attacks. As the web site is protected by an instance of MODSECURITY featuring the OWASP CRS, which has been tuned for several years by a team of security and infrastructure experts. We therefore used this instance of MODSECURITY as the labeling tool: those requests that were accepted by MODSECURITY were considered as valid traffic, while those requests that MODSECURITY rejected were tagged as attacks.

After the tagging process, the only post-processing of this dataset consisted in blurring password values in the request.

Since the requests are from real life traffic, this dataset is less balanced, we divided the incoming request in two set, the first three days of traffic that were used to construct the training and testing datasets and the last three days of traffic that were used to create a validation dataset. The train and testing datasets contains: 65582 valid request and 1287 real attacks. The validation dataset contains: 41518 valid request and 2226 real attacks. One observation is that the validation dataset contains almost double of attacks and less valid request. We presume that this attack behavior change corresponds to the days of week, as the first dataset was created using the traffic from Wednesday to Friday and the validation dataset using the traffic from Saturday to Monday.

4.3 The learning architecture

In this section we review the main components of the learning architecture we have conceived to perform the reported experiments. They are depicted in Figure 4.1.

The input to our system are the raw HTTP requests as they are received by any web application server. The parser is the responsible of normalize and parse this request (based on the HTTP protocol structure) before sending to the tokenizer. After pre-procesing, the tokenizer transform the text into a vector of numbers. During the experiments we worked on two different approaches for this component, one based on the *Bag of Words* model and generating features base on the words frequency (using the TF-IDF measure). The other one is based on the knowledge of a security expert to identify the tokens used as the vocabulary to build the *Bag of Words* model. Finally, depending on the classification approach, we experiment with muti-class and one-class classifiers. In what follow we describe in most detail each of these components.

4.3.1 Parser

In this specific problem the samples are instances of the semi-structured text protocol HTTP *Request*. This protocol is used to exchange information between the client and the server. To take advantage of the protocol structure, the parser has three main objectives: information decoding, headers filtering and special characters preservation.

Information decoding

The HTTP protocol is text-based so when non-ASCII information has to be exchanged some encoding has to be employed [56]. Many different encoding mechanisms could be used such as URL encoding [27], Unicode Encoding, Hex Encoding and Base64 Encoding. The main encoding mechanism used in the Web is defined in the RFC 3986 [5], where it defines a method of encoding for non-ASCII characters transformation in the URL known as the URL encoding format. Encoding has been used by many different web application attacks exploits to bypass or evade security controls by mixing different types of encoding and even use double encoding. For example the following classic XSS injection payload injected in the parameter named *param*:

$$param = javascript : alert(document.cookie)$$

could be encoded using base64 as follows:

$$\begin{aligned} param =&data : text/html; base64, \\ &PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4 = \end{aligned}$$

MODSECURITY includes several functions to decode the information as a configurable pipeline; where the administrator defines which functions to use and in what order. In this work we focus on URL encoding format as is the major encoding format used in the Web, extensions on this aspect can be implemented re-using MODSECURITY features. The first action of the parser is to decode all the URL encoded information so the learning algorithm can work with the real information. The URL encoding format could be used not only in the URI, but also for the query string and the parameters sent in the body. This encoding mechanism substitutes specials character by three characters: the special character % and two hexadecimal numbers corresponding to the US-ASCII value of the character. For example, the following encoded URI:

$$/user/login?destination = search\%2Fnode\%2Flimite\%20de\%20cursadas$$

will be decoded as:

$$/user/login?destination = search/node/limitedecursadas$$

Headers filtering

Additionally, information contained in headers that are specific to the training request, and therefore should not be considered to infer application behavior, is filtered. The objective of this task is to eliminate information in the requests that are specific to the training environment that could lead to the machine learning algorithm to learn this particularities in order to classify new requests. Examples of

host	x-forwarded-for	via	client-ip	referer
if-none-match	set-cookie	last-modified	if-modified-since	—-
proxy-authorization	if-range	if-match	from	upgrade
authorization	max-forwards	ua-cpu	ua-disp	ua-os
cookie	etag	accept-charset	accept-encoding	if-unmodified-since
ua-color	ua-pixels	x-serial-number	pragma	range
cache-control	accept-language	trailer	expect	

TABLE 4.2: Headers filtered during the parsing process

these cases are the value of a cookie or the timestamp of the last time the web page was modified. The whole headers filtered during the pre-process are presented in Table 4.2.

Special characters preservation

Several attacks make use of specially crafted input to make the server to execute a not expected functionality. Most of these inputs use as part of the payload special characters, for instance . , ; $<$ $>$ $=$ /, which are normally used in information retrieval to split the documents. In order to preserve those characters, in the tokenization phase, the parser uses the knowledge of the HTTP protocol in order to preprocess the request. During this phase, the parser analyzes the parameters from the query string, the body and headers and split them in name and parameter value as a different text string using the space character as separator. For example, the *destination* parameter that we saw earlier would be transform as:

$$destinationsearch/node/limitedecursadas$$

In this way, the / character in the parameter value would be kept in the text during the tokenization process. Another example could be the XSS payload on the *param* parameter, in this case, after parsing it would be transformed as:

$$paramjavascript : alert(document.cookie)$$

in this case, we change the $=$ sign (that is part of the protocol) in order to keep both the name of the parameter and the value as two different text strings.

Parsing result

In Section 2.1.1 listing 2.1 we present one of the request of the DRUPAL dataset. This requests is compose of a *Request Line* that contains the URI and the query string, the requests headers and a requests body on line 16.

After processing this request with our parser, the result of the transformation is presented in Listing 4.1. As shown, we could see that the original 16 lines of the request gets transformed in one line of words separated by the space character.

```
POST HTTP/1.1  destination search/node/limite de cursadas connection
    : keep-alive content-length: 200 origin: https://www.fing.edu.uy
     upgrade-insecure-requests: 1 user-agent: mozilla/5.0 (windows
    nt 10.0; wow64) applewebkit/537.36 (khtml, like gecko) chrome
    /56.0.2924.87 safari/537.36 content-type: application/x-www-form
    -urlencoded accept: text/html,application/xhtml+xml,application/
    xml;q=0.9,image/webp,*/*;q=0.8  name jose.perez pass **********
    form_build_id form-Kh0_uekNLVtroZXTLGt-71ZmRx3TJEwP7jz3K2DWQI0
    form_id user_login securelogin_original_baseurl https://www.fing
    .edu.uy op Iniciar sesi\'on
```

LISTING 4.1: Parsed HTTP Request

4.3.2 Tokenizer

The tokenization process adopted was highly dependent of the learning scenario. We use a classic information retrieval approach by applying a *bag of words* model with the TF-IDF scheme and an expert assisted approach in which an expert selected the features to use. Both models were applied after the parsing process.

In *sc*1 we apply both tokenization methods, since we are using real data to the application both models could be applied.

In the scenarios *sc*2, *sc*4 and *sc*5, the use of features blindly generated from the *bag of words* model, did not work. The main problem was the large amount and sparsity of the extracted features, because few of them were active in each instance. In this feature space it was difficult to distinguish attacks from valid requests. To address this difficulty we incorporated to the analysis the experience of a security expert to identify the features manually.

Classic information retrieval approach

In order to apply the *bag of words* model, first of all the vocabulary to be used has to be defined. In this case, the vocabulary is defined by tokens obtained from the requests in the training datasets. In this process, the tokenizer performs the splitting using only spaces preserving special characters that may be included in parameters values or headers.

After tokenization, using the *bag of words* tokenizer, each request is transformed into a vector applying the classic Term Frequency Inverse Document Frequency (TF-IDF) [54] scheme in order to calculate the corresponding weights of each of the token in the request.

The tokenization process generates too many features for the machine learning algorithm. Features corresponding to terms in our vocabulary, which are generated as a result of the tokenizer. To reduce the set of features, feature selection is performed using the information gain algorithm [64], keeping the features that best help to distinguish between the classes.

Expert assisted approach

In what is cal *expert-assisted* approach, a set of features that better characterize different web application attacks are defined by the expert. Validation of the proposed features has been performed applying an information gain [62] algorithm on the three datasets. The results showed that all features have a positive information gain in at least one of the datasets. The CSIC2010, PKDD2007 and DRUPAL dataset have

<	../	alert	exec	password
<>	’	alter	from	path/child
<!–	“	and	href	script
=	(	bash_history	#include	select
>	)	between	insert	shell
|	$	/c	into	table
||	*	cmd	javascript:	union
-	*/	cn=	mail=	upper
–>	&	commit	objectclass	url=
;	+	count	onmouseover	User-Agent:
:	%00	-craw	or	where
/	%0a	document.cookie	order	winnt
/*	Accept:	etc/passwd	passwd	

TABLE 4.3: Selected features by the security specialist

43, 38 and 59 (respectively) features out of 64 with a positive information gain. Table 4.3 lists the defined features.

In this approach, instead of splitting the string, its simply counts for each request the number of appearances of the substring defined by the feature in the request. It is not necessary that the feature appears as a word after tokenization, but as a substring inside the request. For example, the parameter value $javascript : alert(document.cookie)$ will add one for each of the following features: $javascript$:, :, $alert$, (, $document.cookie$ and). Is important to notice that the character : is itself a feature and also occurs in the feature $javascript$:.

4.3.3 Classification

The last component of the learning architecture is the classifier its self. We have worked with two different approaches: a supervised two-class and a supervised one-class classification.

The supervised two-class approach requires working with a labeled set with two classes of requests, in our case: valid and attack. This approach was applied to $sc1$ and $sc2$.

The one-class classification approach (as the name says) uses one-class for modeling the behavior during training phase. During classification this approach indicates if the new instance behaves as the modeled class or if it differs. This approach was applied to $sc4$ and $sc5$ where we have valid request during training (so we model the normal behavior of the application) and during classification, the one-class indicates how close or not a new request is to this normal behavior.

Two-class

We have carried out two variants of supervised two-class classification: the first one follows a classical information retrieval approach and has been applied to $sc1$. The second variant uses the *expert assistance* tokenization process and was applied to both scenario $sc1$ and $sc2$.

In this approach we have trained different classifiers using WEKA: Support Vector Machine [48](SVM), K-nearest neighbours [1] (KNN) and RANDOM FOREST [10]. A brief description of this algorithms has been presented in Section 2.4.2. The decision of using each of this algorithms come for the following reasons:

- the SVM algorithm was selected as it was used in a previous work by Gallagher et all [24] in order to reproduce and compare the results
- the KNN algorithm was selected to measure the complexity of this particular problem to machine learning techniques, as this is one of the simplest algorithm to apply
- finally the RANDOM FOREST was selected in order to validate the existing approach taken by the MODSECURITY and the OWASP CRS of using rules to detect attacks. It is also an algorithm used in the literature to detect fraud, which is a similar scenario

One-class

Scenarios $sc4$, $sc5$ and $sc6$ have requests that belong to one of the classes, valid or attack, which we called the *target class*. For this reason we have investigated a one-class classification approach [33] where there are available instances of one-class and none or very few samples of the other one. In this work we have addressed scenarios $sc4$ and $sc5$. Scenario $sc6$ requires dataset containing large amounts of attacks, so we will studied in a future work.

The proposed anomaly detection classifier organizes samples of the target class into clusters and then uses the distance to these clusters as a measure of anomaly; samples away from the clusters are classified as anomalies.

Using the Expectation Maximization (EM) algorithm [16] we cluster the training set containing only target samples. The EM algorithm was used to estimate the parameters of a Gaussian Mixture Model (GMM) [18] and the number of clusters (components in the GMM). In our case, each component of the GMM constitutes a cluster that captures the distribution of the target class. To capture the intra-cluster variability we use the distance between samples in the cluster and its centroid. Each sample is represented using the vector $\vec{x}$, which is generated as a result of applying the expert assisted tokenizer. The distance between a given $\vec{x}$ and the cluster C is computed using the Mahalanobis distance [37]:

$$dist(\vec{x}, C) = \sqrt{(\vec{x} - \vec{\mu})\mathbf{\Sigma^{-1}}(\vec{x} - \vec{\mu})} \tag{4.1}$$

where $\mathbf{\Sigma}$ represents the full covariance matrix and $\vec{\mu}$ the centroid of the cluster calculated during the training phase. If one of the features is not seen during the training phase, the corresponding dimension will have a standard deviation of 0, and the Mahalanobis distance cannot be calculated. For this reason, we adjust the covariance matrix by adding a regularization term to it $\epsilon * \mathbf{Id}$, where ϵ is the smallest standard deviation in diag($\mathbf{\Sigma}$) different from 0 and $\mathbf{Id}$ is the identity matrix.

If the distance of a sample to a given component is within the observed intra-cluster variability during training, the sample will be labeled as valid. To apply this idea we need a distance threshold for each cluster. Therefore, for each cluster, we obtain the corresponding mean distance (μ_d) and the standard deviation of the distances (σ_d) of the samples assigned to it. On this basis, the threshold is defined as shown in Eq 4.2.

$$t = \lambda[\mu_d + 10 * \sigma_d], \lambda \in (0, 1] \tag{4.2}$$

The parameter λ allows us to change the size of the cluster from 0 (where only instances that correspond to the centroid of the cluster are classified) to 1 where almost all requests are classified as the target class (as the threshold for the cluster is 10 times the σ_d). The constant 10 was calculated during the experiments by increasing the size of the cluster to include 99% of the instances.

During classification we calculate the Mahalanobis distance of the requests to the clusters: if the distance is equal or less than the threshold the request is assigned to the cluster. Requests that are not assigned to any cluster are classified as attacks. The threshold of each cluster then might be defined by setting the number of false positives that we are willing to accept. This can be done observing the distribution of intra-cluster distances. All samples with distances above the threshold will constitute false positives. In future work we plan to model this distribution in order to obtain an estimation of the false positive rate given the selected threshold.

4.4 Limitations

In this subsection we present design decisions that were taken during this work in order to restrict the scope.

4.4.1 The parser and encoding mechanisms

As described in Section 4.3.1, when inspecting HTTP traffic many encoding methods could be used in order to transmit the information. Depending on the web server, it could be also possible to use multiple encoding methods one after other (a.k.a: *double encoding*). This characteristic is often used by attackers in order to bypass control mechanisms (e.g. WAF) that process the user input.

Different approaches to protect against this type of attacks could be implemented, but they are highly dependent on the protected application. For example, the OWASP CRS rules use a specific rule to detect double encoding. If the request uses more than one encoding then it is rejected. These rules usually work fine as most of the protected application do not use multiple encoding mechanism. In the case where the protected application uses many encoding for some specific requests, the parser has to be tuned to decode all encoding used.

In our work, we only decode the URL encoding mechanism as is the classic mechanism used in the major of the web application. Other types of encoding are not supported in this version of the parser.

4.4.2 HTTP headers and cookies attacks

There exists many attacks that aim at the HTTP protocol (such as HTTP splitting, HTTP header injection) or at the session management (such as Session Fixation, Session Hijacking). Generally this type of attacks abuses of user information provided in some HTTP headers or the way that is processed by the web server.

As we mentioned in Section 4.3.1 some HTTP headers filtering is done during parsing. The objective of this filtering is to eliminate from the request information that could be specific of the training dataset so the machine learning algorithm do not use this information to learn user behavior.

One example of this filtering is the cookie session. The reason is that if in the training dataset have few session of valid users, the machine learning algorithm could use the value of the application cookie (the classic approach to keep session in the HTTP) in order to classify valid versus attack requests. Even this could be a good feature in the training set to discriminate attacks from valid requests, it will not work when new valid users sessions are presented.

By applying this filtering, some types of attacks that are dependent on user inputs could not be detected by our approach.

4.4.3 Request body content type

HTTP defines several content types to be send by the client to the server. Depending on the content type that the client wants to send defines how the web server will process the request body. Some of the classic content types used by web applications are: *application/x-www-form-urlencoded*, *multipart/form-data*, *text/xml* and *text/json*. Each content type defines the way that body has to be processed.

In our work, we focus on the *application/x-www-form-urlencoded*, since most of the requests in the datasets that we have used only contains requests with this particular content type. Specifically the PKDD2007 and CSIC2010 datasets only contain requests with this content type. In the DRUPAL dataset there were 2012 requests with the *text/xml* (but all with almost the same value) and the rest with the *application/x-www-form-urlencoded* content type.

In order to be able to process requests with other content types, the parser has to be extended in order to apply different parsing mechanisms depending on the request body content type. MODSECURITY has implemented support for the four classic content types. As our objective is to improve MODSECURITY with machine learning approaches, further work will be focused on this integration.

Chapter 5

Implementation

Most of the work presented in this document is done applying classic machine learning techniques to a new application area (*Web Application Attacks*). To apply these techniques we worked using WEKA [26], a classic Data Mining tool. We have also implemented a specific algorithm in order to tackle the one-class approach.

In what follows we present the integration with the WEKA tool and our specific implementation

5.1 The WEKA tool

WEKA is a *workbench* where several standard machine learning techniques used for the different phases of the classic statistical pattern recognition system are included. For example, there are many algorithms for: pre-processing the data (feature extraction and feature selection), clustering and classification.

WEKA provides different ways to use it, as a standalone tool, where the expert could use the *Explorer* or *KnowledgeFlow* or as a system library to be used from your own source code. During this work, we used both of these approaches.

For the classic information retrieval two-class approach, we used the *Explorer* and the *KnowledgeFlow* in order to implement, process and test the algorithm used.

5.2 Dataset loader and parser

In this section we present the implementation related to the loading and parsing of the dataset. Our first task was to load the datasets in different formats. We started by implementing a loader which is responsible of understanding the different dataset formats and translated them into a single format. After the information was loaded, the next step is to apply the parsing decisions presented in Section 4.3.1.

Figure 5.1, presents an UML component diagram which describes the involved sub-systems and their relationship. The main subsystem is the *Generator*, which acts as a coordinator between all subsystem and it is responsible of the dataset generation. The *Generator* uses the interfaces presented by the *Loader* and *Transformer*. The *Loader* subsystem and its extensions parse the datasets from their original format and standardize them into the format needed by the *Generator*. The *Transformer* subsystem, transforms a sample from our format to the WEKA expected format. Finally the *PreproccessCommon* defines the main interfaces shared between all subsystems. What follows describes each sub-system in detail.

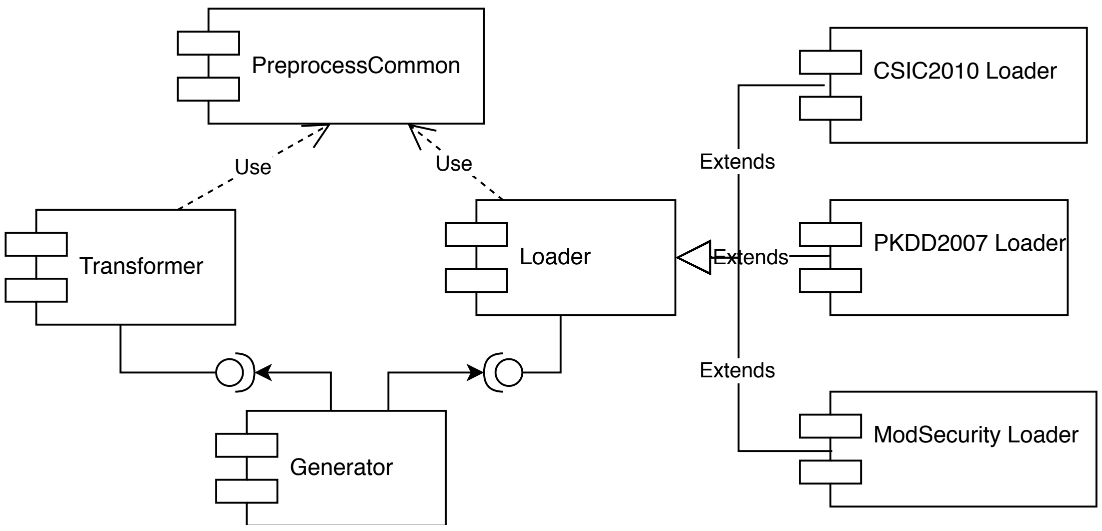

FIGURE 5.1: Dataset Loader and Parser - Component Diagram

5.2.1 PreprocessCommon subsystem

This is a helper subsystem which defines the standardized structure of the information to be retrieve by the different Loaders. It is also responsible of defining the interfaces to be used by the communications between the different subsystems.

Figure 5.2, presents the main interfaces defined in the PreprocesssCommon subsystem as a class diagram.

The main interface defined is the *Sample* interface, that defines the behavior of each sample. In our case, a sample is compose of a:

- Request, which represents an HTTP Request
- Class, that defines the class of the sample, in general, the values are *Valid* or *Attack*. In the case of the PKDD2007 dataset, it could also have the attack class divided into the seven attack classes that are define in the dataset
- Context, which is a mapping of the form attribute and value, that depends on contextual information about the requests
- Id, is a String that uniquely identifies that instance

The *Request* interface represent the main information of the HTTP request. The *WekaTransformer* and the *DataSetReader* interfaces correspond to the interfaces used by the *Generator* to communicate with the Transformer and Loader subsystem respectively.

Finally, the *DataSetFileReader* is a class that implements the *DataSetReader* to centralize de implementation of accessing a file as all readers in a way or another access information from disk.

5.2.2 Loader subsystems

Each dataset has its own format, so the objective of this subsystem is to standardize an unified the dataset in order to work as an input to the algorithms. All of the subsystems that conforms the *Loader*, uses the interfaces defined in the Preprocess-Common and implements what is necessary to transform from the specific input to this format.

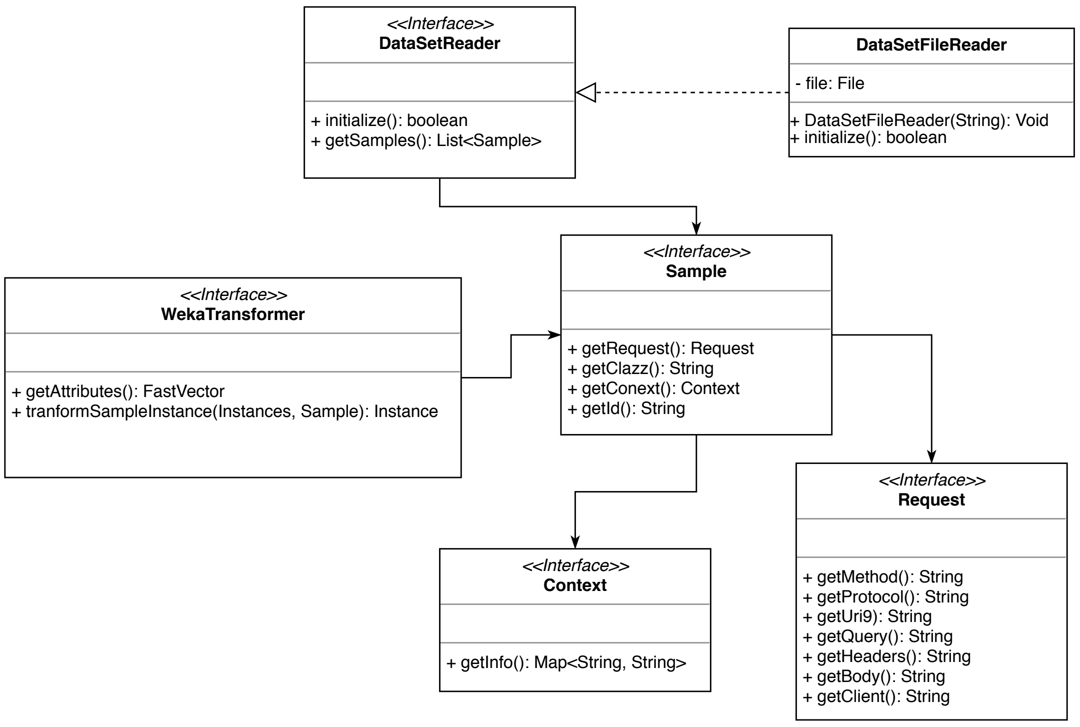

FIGURE 5.2: PreprocessCommon subsystem - Class Diagram

PKDD2007 loader

The PKDD2007 dataset has a proprietary XML format that is described in [2]. One XML file contains the whole dataset. Each entry of the XML corresponds to an instance of the dataset and includes the following information:

- id, an incremental integer
- reqContext, where there is additional contextual information such as: Operative System and web server
- the class of the requests, that could be valid, XSS, SQLInjection, LdapInjection, XPathInjection, PathTrasversal, OsCommanding and SSI
- and the request itself

The Java^{TM} Architecture for XML Binding (JAXB) [21] was used in order to map the XML documents into Java objects. This framework provides a set of tools that were used to create java objects from an XSD and an API to manipulated them.

CSIC2010 loader

The CSIC2010 dataset is presented as a plain text, where all the requests are layout one after another with two empty lines separating them. There exists three plain text:

- normalTrafficTraining, which contains all the requests that corresponds to the Valid class and are used for training
- normalTrafficTest, similar to the previous but for testing

- anomalousTrafficTest, all the requests in this file we call them attacks. It is important to be notice that not only attacks, but also simple anomalous traffic for this application are in this file, but we do not have a way to split them

In this case, the parsing was implemented by reading line by line of the input file and parsing manually each of the line in order to obtain the information needed by the system. Depending on the file being read, is the class of the sample. Finally, no contextual information is available, so this information is not loaded and the id corresponds to the position of the requests in the file.

DRUPAL loader

As we mentioned before, the DRUPAL dataset was generated by using an instance of MODSECURITY recording traffic to a website. For this reason, even though this subsystem is call DRUPAL *Loader*, it actually loads requests saved using the MODSECURITY logging format [23].

MODSECURITY has several formats to log requests, in this work we work with the *Concurrent* format. In this format one plain text file is written for each request to be logged. This plain text files are identified by the time of the request concatenated with a unique token generated by the Apache's mod_unique module. Inside the file, we find the request, its body and the response of the server.

We have evaluated several existing libraries to parse MODSECURITY logs. All of them require the main indexer file generated by MODSECURITY's audit engine. Since we did not have this file we could not use these libraries and therefore we implemented our own parser, called DRUPAL Loader.

The DRUPAL Loader receives as input the folder where each of the individual MODSECURITY logs are in. It reads each folder recursively and generates a sample for each file found. Then each sample is responsible for parsing its file to obtain the information needed. The plain file is divided into sections. In [23] Folini et al define the log format and the information contained in each section, the DRUPAL Loader uses sections A, B, C and H.

From section A it gets the information about the client and the unique id of the requests (that is used as the sample id). The section B contains the request header (requests line and the headers). In section C, if exists corresponds to the request body.

Finally the class of the requests is calculated depending on the response of this high tuned MODSECURITY. If it responses with an *"Access denied with code 403"* then the requests is mark as an attack, otherwise is a valid requests. There are two exceptions to this behavior and corresponds to requests that after the dataset being manually analyzed where identified as false positives of MODSECURITY. The first case corresponds to a false positive related to the rule with id 981260 of the OWASP CRS version 2 referring to *"SQL Hex Encoding Identified"* and the second case is a false positive generated by the rule with id 981320 related to the text *"msdb"* found in a cookie.

5.2.3 Transformer subsystem

The transformer subsystem has two major objectives: the first objective is applying the parsing characteristics described in Section 4.3.1 and also is the responsible of transforming an instance in the *Sample* format defined above into an instance in WEKA format.

In order to fulfilled with the first objective, during transformation the following actions takes place:

- all information contained in the request is URL decoded
- the query string of the request is parsed, changing the special characters that defined by the protocol (such as: "&" and "=") with the space character
- the requests headers are filtered (using the criteria presented in Section 4.3.1) and concatenated in a single line separated by the space character
- if the body of the request exists, it is parsed in the same manner as the query string

We have implemented two variants of the process that transform the sample format into the WEKA format: the multiple attributes and the single one. The first version, generates a WEKA attribute for each part of the requests. This allows the expert to apply different types of techniques to each part of the request. This multiple attribute version was implemented, but it was not used during in the results presented in this work. The single attribute version, generates an instance with two attributes: the text (that corresponds to the whole requests after parsing) and the class attribute. This version was used to generate the results presented in this work.

5.2.4 Generator subsystem

This subsystem is the coordinator that uses the previous subsystem in order to generate the dataset to be used. Listing 5.1, presents the options that receives the Generator in order to execute.

```
Usage: java Generator <input type> <path to input file> <output type
    > <path to output file> <transformer> [-toMallet <
    path_mallet_file>]
input type: xml, modSecurity and csic
path to input file/directory: path to xml to load or to directory
    with ModSecurity logs
output type: csv, arff
path to output directory
transformer indicates the class used to build the WEKA instances
toMallet indicates to save to mallet format
path mallet file: path to file to save in mallet format
```

LISTING 5.1: Generator: command line options

The last two options correspond to transform the output generated for WEKA to other tool called Mallet. This options are optional and not used during this work.

5.3 Tokenizer

As we mentioned earlier, the result after executing the *Generator* is a training and testing dataset with two attributes: text (corresponding to the requests preprocessed) and the corresponding class. In order to use this dataset in machine learning algorithms is necessary to transform the text attribute into a vector of numbers. To implement this transformation, we use the following approach (as presented in Section 4.3.2): classic information retrieval and expert assisted.

5.3.1 Classic information retrieval tokenizer

To implement this approach, we used the *StringToWordVector* function of WEKA. The *StringToWordVector* algorithm uses a tokenizer in order to split the documents (in our case correspond to the *text* attribute of the dataset) and creates the features based on the tokens generated. In Listing 5.2 we present the WEKA command and the arguments used.

```
java weka.filters.unsupervised.attribute.StringToWordVector
-R first
-W 4000
-prune-rate -1.0
-C
-T
-I
-N 0
-L
-stemmer weka.core.stemmers.NullStemmer
-M 1
-tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r \\
    t\""
-b
-i input.arff
-o output.arff
```

LISTING 5.2: WEKA command to execute the classic information retrieval tokenizer

The $-T$ and $-I$ indicates to use the TF-IDF measure. $-C$ outputs word counts in order to be used by the TF-IDF measure and the $-L$ specify to be case insensitive. To preserve the special characters used in attacks, the *WordTokenizer* uses the following character to split the request: `␣\r \t`. Another problem of this approach is the amount of attributes generated, for example the DRUPAL dataset generates over one million attributes.

To reduce the amount of attributes to work with we have limited the token generation by using the parameter $-W$ to keep at most 4000 tokens of each class. This means that as a result of the tokenization process the attributes generated could range from 4000 to 8000 as it could exists repeated tokens in documents of both classes.

This amount of attributes is still too large for the machine learning algorithm. To reduce the amount of features, feature selection is performed using a combination of two functions of WEKA: the information gain algorithm and the ranker. The fist one calculates how much information could be gain by an attribute to respect of a class, this generates a measure of how worth is this attribute. The ranker is responsible of ranking the attributes based the results of the information gain result and keep the first 1000 with a positive information gain. Listing 5.3 present the commands corresponding to this feature selection process.

```
Evaluator:    weka.attributeSelection.InfoGainAttributeEval
Search:       weka.attributeSelection.Ranker -T 0.0 -N 1000
```

LISTING 5.3: Feature selection using WEKA

5.3.2 Expert assisted tokenizer

As described in Section 4.3.2, the features of this approach are defined by a security expert instead of being blindly generated as in the classic information retrieval approach.

WEKA provides a *FixedDictionaryStringToWordVector*, but it works similar to the *StringToWordVector*, as it first split the text documents using a tokenizer and then count the generated tokens and keeps only the ones defined in the given dictionary. In our approach, the requirement is to find the number of appearances of a specific sequence of characters in the document. So we implement a different way of counting the tokens, where we use the whole input document as a text string, and we count, for each token of the giving dictionary, the number of appearances of the feature as substring of the document. The reason of implementing this approach is that our dictionary is compose of tokens that usually appears as part of the payload of an attack. The important concept here is that are part of the payload, not the whole payload (as it can change in every attack) so is not feasible to implement a tokenzier that could separate the tokens in a way that could be useful for the WEKA algorithm.

Our implementation of the tokenizer, called *FixedDicNoTokenizeStringToWordVector*, was done as an extension of the WEKA *SimpleStreamFilter* so it could be integrated into WEKA's set of algorithms.

5.4 Two-class classifier

For the two-class approach we used the WEKA's implementation of the algorithms. We use the *Explorer* interface of WEKA to load the different datasets and classifying them using the algorithms presented in Section 4.3.3.

After executing the algorithm proving different parameters and their values, despite of the fact that for some specific dataset some parameters work better than others, in general, we have experimentally proved that the default parameters perform well in all datasets. Listing 5.4 present the WEKA command corresponding to the different algorithm used for multi-class classification.

```
weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 1 -V -1
     -W 1 -K "weka.classifiers.functions.supportVector.RBFKernel -G
    0.01 -C 250007" -calibrator "weka.classifiers.functions.Logistic
     -R 1.0E-8 -M -1 -num-decimal-places 4"

weka.classifiers.lazy.IBk -K 3 -W 0 -X -A "weka.core.neighboursearch
    .LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"
    "

weka.classifiers.trees.RandomForest -P 100 -I 10 -num-slots 1 -K 0 -
    M 1.0 -V 0.001 -S 1
```

LISTING 5.4: Two-class classification using WEKA

The *SMO* algorithm correspond to WEKA's implementation of Platt [48] sequential minimal optimization algorithm for training a support vector classifier. We used most of the standard parameters for this experiments (we did not do an exhaustive search), the following parameters were changed:

- $-N1$, this parameter indicates weather to normalize or to standardized the training data before executing the algorithm, in our case we standardized the values. This means that before training, the values are re-scaling so that they have a mean value of 0 and a standard deviation of 1

- $-K$”, this parameter defines the kernel to be used, in our case the best results correspond to the *RBFKernel*

Line 3, shows the execution line corresponding to the KNN algorithm. For this algorithm the only parameter that we changed is the number of neighbors to use, in our case $-K3$. Line 5 is the corresponding to the RANDOM FOREST algorithm. In this case, we only changed the max number of iterations to be performed by the algorithm: $-I10$.

5.5 One-class classifier

The one-class classification approach (presented in [6] and [7]) as described in Section 4.3.3, is based in a classical approach of clustering with some specific modification introduced during this work. In Listing 5.5 we present a pseudocode of the algorithm implemented which calculates the one-class classification approach for 500 different lambda values. This algorithm uses WEKA as a library.

```
oneClassClassifier ( trainPath , testPath : filePath) : double[][]
    Instances train = getDataset(trainPath)
    Instances test = getDataset(testPath)

    EM em = new EM(maxIter:100, minStdDev:0.000001,
        MinLogLikelihoodImprovementIterating: 1e-2)
    em.buildClusterer(train)

    centroids = getCentroids(em)

    train = addClusterAssigned(train, em)

    Instances distance = calculateMahalanobisDistance(train,
        centroids)

    for each cluster c
        Instances clusterInstances = getInstanceInCluster(c,
            distance)
        mean[c] = clusterInstances.getMean(c)
        devStd[c] = clusterInstances.getDevStd(c)
        maxRadius[c] = 10 * devStd[c] + mean[c]

    for each lambda l in 1..500
        threshold = maxRadius * l/500
        for each instance inst in test
            isClassifiedAsValid = false
            for each cluster c
                if (!isClassifiedAsValid && (distance(inst, c) <
                    threshold[c] ))
                    isClassifiedAsValid = true;

        result[l][inst] = isClassifiedAsValid

    return result
```

LISTING 5.5: One-class classification algorithm: pseudocode

The algorithm receives two datasets: a training and a testing dataset. In case that only one is provided, it is splitted in a 70/30 proportion for training and testing respectively.

The first step is to calculate the Expectation Maximization (EM) using as an input the training dataset. This is done using the implementation of the EM algorithm provided by Weka. The EM algorithm calculates the number of clusters, calculates the centroid of each cluster (as shown in line 6) and assigns each instance to a cluster (line 10).

After having the centroids of each cluster, we calculate the distance of each instance of the training dataset to each centroid. This could be done specifically for the corresponding centroid (depending on the cluster assign), but was done in this way since all distances were needed during different phases of the work.

The next step is to calculate the maximum threshold for each cluster. In this task is important to notice that we do not used the mean and standard deviation calculated by EM. We use the mean and standard deviation of the mahalanobis distances. This distances are calculated for each instance assign to the cluster to the centroid of the cluster (this is shown in lines 14-18).

Finally we have to estimate the lambda. In order to do that, we set the threshold in line 21 as maxRadius times a lambda value that range between 0 and 1. With the threshold defined, we calculate the mahalanobis distance for each instance to each cluster. If any of this distances is less than the corresponding threshold, then the instance is classified as valid, otherwise is classified as an attack.

5.5.1 Mahalanobis distance

The implementation of the one-class classifier requires to calculate the mahalanobis distance. As we already presented in Section 4.3.3, the mahalanobis distance used in this algorithm uses the full covariance matrix. We have implemented the *MahalanobisDistance* as an extension of Weka's *NormalizableDistance*, so it could be included in Weka.

We include *The Apache Commons Mathematics Library* [39] to delegate all the operations on matrix and vector composed of real numbers.

Listing 5.6 presents a pseudocode of the *MahalanobisDistance* initialization process.

```
initialize(train : Instances, divideEpsilon : double) : RealMatrix
    Covariance cov = new CovarianceMatrix(train)

    epsilon = MAX_VALUE

    for each d in diag(cov)
        if d <> 0 and d < epsilon
            epsilon = d

    epsilon = epsilon / divideEpsilon

    DiagonalMatrix diag = new DiagonalMatrix(epsilon)

    cov = cov.addMatrix(diag)

    return LUDecomposition(cov).getInverse();
```

LISTING 5.6: Mahalanobis distance initialization: pseudocode

The initialization process receives as an input the training instances (assign to the cluster) and a double value used to parametrically change the value of epsilon. The first step is to create the covariance matrix from the input training set. Then epsilon is calculated as the minor positive value in the diagonal of the covariance matrix, namely, the smaller positive variance. After finding the smaller variance, this is divided by the input *divideEpsilon*. When larger is dividedEpsilon, smaller is the epsilon to be added to the covariance matrix. We empirically found that a good value for *divideEpsilon* is 100. Finally we use the lower–upper (LU) decomposition in order to invert the covariance matrix.

After the initialization process, we need to calculate the distance of an instance to the centroid of the cluster. To do this we just apply the equation 4.1. Where Σ is the resulting matrix of the initialization process, x is the instance and μ is the centroid calculated with the EM algorithm for the corresponding cluster.

5.5.2 One-class evaluator

In a first attempt to integrate the one-class classifier with ModSecurity, we have implemented an evaluator. The evaluator is only responsible of evaluate new request, using a model previously calculated with the *One-class classifier*.

The one-class evaluator can be executed as a Java client application (executable jar) [1]. During execution the Java client application requires to be set the *app.home* variable in the environment. This variable contains the path to the folder where the model information and the dictionary with the tokens defined by the expert are specified.

The model is composed of one plain text file per cluster, where each file contains the following information:

- threshold, a double value that corresponds to the threshold defined for this cluster. It is important to notice that in this point it is assumed that λ is already defined and the threshold is a specific value
- centroid, an array of double with the value of the centroid of the cluster
- inv-cov, the matrix with the inverse of Σ. The evaluator receives the inverse already calculated for each cluster

The dictionary is also a plain text with one feature per line.

For the classification of a new request the evaluator must be provided as input a JSON file and it will produce a JSON with the corresponding output. Listing 5.7 presents an example of the input file format.

```
{
        "uniqueId": "152597966420.886223",
        "method": "POST",
        "protocol": "HTTP\/1.1",
        "uri": "\/index.html?arg1=1&argn=n",
        "body": "argPost1=post 1&argPostn=post n",
        "argsGet": {
                "arg1": "1",
                "argn": "n"
        },
        "headers": {
                "User-Agent": "Mozilla\/5.0 (X11; U; Linux x86_64; en
                    -US; dfdfd) Gecko Firefox",
                "Accept": "*\/*",
                "Accept-Encoding": "gzip",
                "Host": "localhost",
                "Connection": "Keep-Alive",
                "Content-Type": "application\/x-www-form-urlencoded",
                "Content-Length": "31"
        },
        "argsPost": {
```

[1]This project is publicly available at https://gitlab.fing.edu.uy/gsi/waf-ml-oneclass

```
                "argPost1": "post 1"
                "argPostn": "post n"
        }
}
```

LISTING 5.7: One-class evaluator: input format example

As it can be noticed, this JSON format include all the information of the request parsed in different fields. Is also important to notice that in some cases the information is repeated, for example, the body of the requests could be found in raw format in the *body* tag and it could also be found with the arguments parsed in the *argPost* collection. This corresponds to the fact that this information is already parsed by MODSECURITY, and we want to take advantage of this parsing implementation in the machine learning algorithms.

```
{
  "results": [
    {
      "clazz": "Valid",
      "clazzDescription": "Class that represent valid request",
      "probability": 1.0,
      "crsClazz": "N/A"
    },
    {
      "clazz": "Attack",
      "clazzDescription": "Class representing an attack request",
      "probability": 0.0,
      "crsClazz": "Inbound Score"
    }
  ]
}
```

LISTING 5.8: One-class evaluator: output format example

Listing 5.8 is an example of the output file generated by the One-class evaluator. The output includes the information for each class: name, description, probability and map to the OWASP CRS.

5.6 MODSECURITY requests generator

To compare our results with the baseline, we have to evaluate each dataset against an installation of MODSECURITY configured with the OWASP CRS. As we have mentioned earlier, each dataset has its own format. To generate our baseline, we implemented a component that we call the MODSECURITY *request generator*. This generator is capable of:

- handling the different formats for each dataset
- inferring the evaluation result
- obtaining for each request the information of the MODSECURITY evaluation

The generator embodies two major sub-components: the MODSECURITY evaluator and the request generator.

5.6.1 ModSecurity evaluator

To make more flexible the installation of ModSecurity, this sub-component runs in a separate context (such as a virtual machine or a Docker instance) where it has to be installed a web server with the ModSecurity module and the rules to be evaluated (in our case the OWASP CRS version 2 and 3).

There are two specific configuration for the web server installation: the first one correspond to the ability of the web server of receive any request independently of it *Host* header. The second configuration is that if ModSecurity detects the request as an attack, the web server has to return a *403 Forbidden* error, otherwise it has to respond with a *200 Ok*.

5.6.2 Request generator

The request generator sub-component is responsible of loading a dataset and then sending the requests to an instance of ModSecurity. All the logic needed to load the dataset is delegated to the *Loader* subsystem presented in Section 5.2.2.

After the requests are loaded, the generator opens a TCP/IP connection to ModSecurity and sends the request information as it is defined in the dataset. The only processing performed during this task is the addition of a custom header (called *Instance*) with a number, which indicates the requests number executed for this dataset. This information could be obtained from the ModSecurity logs in order to understand how ModSecurity evaluated the requests.

Finally, the classification depends on the web server response, if it is a *403 Forbidden* the requests is classified by ModSecurity as an attack, otherwise is a valid request.

Chapter 6

Experimental results

This section is devoted to present and discuss the outcomes of the experiments that have been carried out. The results will be presented in terms of Precision, Recall, True Positive Rate (TPR) and False Positive Rate (FPR). In our case, the positives cases are the attacks, so the TPR and FPR indicate the ratio of requests correctly and incorrectly classified as attacks, respectively.

6.1 Datasets

In supervised machine learning approaches one of the major challenges is to have enough data labeled in a way to be useful for training and testing the algorithm. As we presented in Section 4.2, as far as we know, three dataset are publicly available and only two of them meet our requirements (PKDD2007 and CSIC2010). In order to complement these results we created our own dataset (DRUPAL) to compare our results against a currently running web application.

In order to compare the results in the different scenarios defined and to take care of using part of the dataset for training and the other part for testing, we applied different splitting criteria depending on the dataset. Table 6.1 presents the total requests labeled as Valid and Attack per dataset for training, testing and total.

In the case of the DRUPAL and PKDD2007 we use the classical criteria of splitting 70% for training and 30% for testing, after applying a shuffle function to the requests. In the case of CSIC2010, the dataset already came divided into training and testing, but since this dataset was created for anomaly detection approach, it only includes valid requests for training. So this division did not work for us, for this reason we mix all normal traffic (training and testing) and apply the same criteria as the other two.

Additionally for the case of the DRUPAL dataset, we have a second dataset (DRUPAL Validation). DRUPAL Validation was generated using the next three days period of the captured requests (see 4.2.3). This dataset was kept separately in order to use it strictly as a validation dataset.

	Train		Test		Total	
Dataset	Valid	Attack	Valid	Attack	Valid	Attack
DRUPAL	45903	905	19679	382	65582	1287
DRUPAL Validation	N/A	N/A	41518	2226	41518	2226
PKDD2007	24482	10599	10524	4511	35006	15110
CSIC2010	50346	17599	21654	7466	72000	25065

TABLE 6.1: Requests per dataset and the distribution for training and testing

Dataset	MODSECURITY with OWASP CRS	Precision	Recall	TPR	FPR
DRUPAL	OWASP CRS v2	0.03	0.75	75.00%	39.68%
	OWASP CRS v3 – PL 1	0.04	0.30	29.55%	15.57%
	OWASP CRS v3 – PL 2	0.03	0.78	77.89%	49.93%
	OWASP CRS v3 – PL 3	0.03	0.78	78.42%	56.82%
	OWASP CRS v3 – PL 4	0.02	0.80	80.00%	61.92%
DRUPAL Validation	OWASP CRS v2	0.05	0.28	27.77%	30.30%
	OWASP CRS v3 – PL 1	0.10	0.12	11.50%	5.49%
	OWASP CRS v3 – PL 2	0.04	0.28	28.07%	35.38%
	OWASP CRS v3 – PL 3	0.04	0.31	31.11%	42.48%
	OWASP CRS v3 – PL 4	0.05	0.46	46.39%	50.28%
PKDD2007	OWASP CRS v2	0.56	0.86	86.17%	28.76%
	OWASP CRS v3 – PL 1	0.64	0.79	79.43%	27.53%
	OWASP CRS v3 – PL 2	0.57	0.89	89.09%	41.25%
	OWASP CRS v3 – PL 3	0.53	0.95	95.04%	51.49%
	OWASP CRS v3 – PL 4	0.29	1.00	99.88%	99.99%
CSIC2010	OWASP CRS v2	0.50	0.34	34.32%	23.93%
	OWASP CRS v3 – PL 1	1.00	0.27	26.62%	0.00%
	OWASP CRS v3 – PL 2	1.00	0.29	29.48%	0.00%
	OWASP CRS v3 – PL 3	0.56	0.53	52.61%	13.95%
	OWASP CRS v3 – PL 4	0.45	0.76	75.86%	31.44%

TABLE 6.2: Summarized results of the MODSECURITY baseline

6.2 Baseline

As we mentioned before, one of the main objectives of this work is to decrease the number of false positives of MODSECURITY. In Table 6.2, we present the results of MODSECURITY configured with the OWASP CRS out of the box for each dataset. This baseline was generated using the MODSECURITY *requests generator*. The MODSECURITY *evaluator* was configured using a virtual machine with CentOS 7, an Apache HTTP Server 2.4, configured with MODSECURITY 2.7. We use two different version of the OWASP CRS.

The first results were obtained using the OWASP CRS version 2, more specifically version 2.2.9 configured in collaborative detection mode with the standard configuration. The collaborative detection mode (a.k.a. anomaly scoring mode) is a special mode of detection defined by the OWASP CRS with the objective of decrease the false positives generation by combining the results of the execution of all the rules and blocking a request only if a threshold is exceeded.

At the same time as we develop our experiments a new version of the OWASP CRS was developed (OWASP CRS version 3) with focus on diminishing the amount of false positive generated. In order to do that, they defined the notion of *Paranoia levels* (PL). The PL is a configuration value that indicates how paranoid the WAF should be from a security perspective. This value ranges from 1 to 4 and it is defined as:

- PL 1: is the default paranoia level and is advised for beginners as it should face rarely FP. This setup is for applications with standard security requirements
- PL 2: is advised for moderated to experienced administrators and installations of elevated security requirements. This level comes with FP to be handle by the administrator
- PL 3: is aimed at administrators with experience at the handling of FP and at installations with a high security requirement

Tokenization	Dataset	Algorithm	Precision	Recall	TPR	FPR
Classic Information retrieval	DRUPAL	KNN-3	0.97	0.91	91.36%	0.05%
		RANDOM FOREST	**0.97**	**0.94**	**93.98%**	**0.05%**
		SVM	0.98	0.92	91.88%	0.04%
	PKDD2007	KNN-3	0.97	0.72	71.78%	0.98%
		RANDOM FOREST	**0.96**	**0.89**	**88.65%**	**1.76%**
		SVM	0.96	0.87	86.79%	1.44%
	CSIC2010	KNN-3	0.99	0.57	57.30%	0.17%
		RANDOM FOREST	**0.98**	**0.68**	**67.76%**	**0.36%**
		SVM	0.99	0.68	67.87%	0.34%
Expert assisted	DRUPAL	KNN-3	0.95	0.88	88.48%	0.08%
		RANDOM FOREST	**0.96**	**0.92**	**91.88%**	**0.07%**
		SVM	0.91	0.66	66.49%	0.13%
	PKDD2007	KNN-3	0.95	0.73	72.73%	1.54%
		RANDOM FOREST	**0.96**	**0.82**	**82.49%**	**1.59%**
		SVM	0.94	0.72	71.58%	1.84%
	CSIC2010	KNN-3	0.94	0.39	38.67%	0.80%
		RANDOM FOREST	**0.94**	**0.39**	**39.45%**	**0.83%**
		SVM	0.84	0.35	35.17%	2.24%

TABLE 6.3: Summarized results of both tokenization approaches on $sc1$

- PL 4: is the highest level and is advised for experience administrators protecting installations with very high security requirements

The PL allows to change the behavior of MODSECURITY depending on the main objective, have less FP or detect most attacks. By analyzing the TPR and FPR values we observe that in general PL 3 and PL 4 have a minor increment in TPR by producing high amount of FP. For these reason and the expertise required for tuning a WAF with high PL, in what follows we are going to compare our results with PL 1 and PL 2.

6.3 Scenario 1

We recall that in this scenario we have trained different two-class classifiers after tokenization of the requests using both the classical information retrieval and the expert assisted approach. To evaluate the algorithms, we use the training and testing datasets described in Section 6.1. We applied the KNN (with K=3), RANDOM FOREST and SVM algorithms using WEKA as described in Section 5.4. The results are summarized in Table 6.3 where the tokenization column indicates the approach applied.

In the classic information retrieval approach it can be noticed that in all datasets the results show good performance in terms of precision, recall, TPR and FPR for all the evaluated classifiers. In particular, we can say that in an overall analysis, the RANDOM FOREST classifier has better performance. This behavior is also observed in literature about fraud detection. For the DRUPAL dataset, we observe that the results of the RANDOM FOREST improve MODSECURITY with both version of OWASP CRS. In the case of the PKDD2007, RANDOM FOREST has better or equal results in terms of TPR and the FPR is less than 2%. Finally the CSIC2010 dataset, is a particular case, where both MODSECURITY and the Classic Information Retrieval approach obtain good results in terms of FPR, very close to 0%. In terms of TPR, RANDOM FOREST improves the OWASP CRS in both versions and PL.

Trainng Dataset	Testing Dataset	Algorithm	Precision	Recall	TPR	FPR
CSIC2010	PKDD2007	RANDOM FOREST	0.49	0.50	50.25%	22.26%
	DRUPAL	RANDOM FOREST	0.12	0.42	42.15%	5.91%
DRUPAL	CSIC2010	RANDOM FOREST	Error	Error	Error	Error
	PKDD2007	RANDOM FOREST	Error	Error	Error	Error
PKDD2007	CSIC2010	RANDOM FOREST	Error	1.00	100.00%	100.00%
	DRUPAL	RANDOM FOREST	Error	1.00	100.00%	100.00%

TABLE 6.4: Summarized results of generalization of the classic information retrieval approach on *sc*1

One major limitation on the classic information retrieval approach corresponds to the fact that labeled data from both classes is needed. Based on the good results obtained and with the objective of minimize this limitation, we studied on the generalization capacity of this approach. In Table 6.4 we present the results of building the model with the training data from one of the dataset and testing the results with data from the other two. In this Table, we only present the results from the RANDOM FOREST algorithm as it has the best results from the three tested. The only model that has generalized from the three tested is the CSIC2010 model. In this model, we could observed that the results are worse than the ones from the classic information retrieval approach. We also observed that for the other two datasets, the generated model could not be used to classify instances (the case of the DRUPAL dataset) or the resulting model classifies all instances as attacks (the case of the PKDD2007 dataset). We recall that in this scenario we use a classic information retrieval tokenizer where the dictionary is created based on the generated tokens during training. We believe that this behavior occurs because tokens generated during training (using one of the dataset) are not present in new instances to be evaluated from other datasets. In other words, tokens generated during training are strongly related to the dataset.

The main conclusion from these results is that the application of the two-class approach in this case is feasible and in most of the cases has good performance scores. However, this approach has two important limitations. On the one hand, labeled data from both classes is needed in order to train the classifiers (see discussion in Section 4.1). On the other hand, the classifier designed for one dataset cannot be used in the other ones. Even using the same features we could not obtain good performance scores when applying, for instance, the model trained for PKDD2007 and then tested using DRUPAL. In other words, the models do not seem to generalize from one web application to another one.

Analyzing the results, we found that the features that were blindly generated using the classic *Bag of words* model captured specific information about the dataset and for this reason the generated models did not generalize. In order to have more generic models, we changed the tokenization method to the *Expert assisted* approach (described in Section 4.3.2). In Table 6.3, in the section *Expert assisted* are presented the results of this approach. As in the case of the classic information retrieval, in an overall analysis we can say that RANDOM FOREST performs better. If we compare the results of these two approaches, we can observe that the results of the *Expert assisted* approach has a minor decrement in terms of TPR, but maintains the value of the FPR. We also notice that in general we have better results that the baseline (see Table 6.2). Another appreciation of these results is the case of the CSIC2010 dataset, that had high reduction of the TPR. This can be explained like in the expert assisted approach: we measure the occurrence of tokens that are normally found in attacks payloads, but in this dataset the attacks are not only real attacks but also anomaly traffic (e.g. requests with typo errors). This behavior could also be directly compared

Dataset	Algorithm	Precision	Recall	TPR	FPR
DRUPAL	**RANDOM FOREST**	**0.95**	**0.48**	**47.57%**	**0.05%**
	KNN-3	0.90	0.07	6.99%	0.01%
PKDD2007	**RANDOM FOREST**	**0.96**	**0.23**	**22.56%**	**0.45%**
	KNN-3	1.00	0.12	11.70%	0.02%
CSIC2010	**RANDOM FOREST**	**0.91**	**0.32**	**32.06%**	**2.27%**
	KNN-3	0.66	0.50	50.43%	17.85%

TABLE 6.5: Summarized results for *sc*2

with MODSECURITY and the OWASP CRS results as they also spot attacks payloads. The TPR for OWASP CRS version 2 is 34.32% and for version 3 is 26.62% and 29.48% for PL 1 and PL 2 respectively. The three cases are below the 39.45% obtained with the RANDOM FOREST algorithm. In conclusion, we believe that even if the *Expert assisted* approach is less performant than the *Classic information retrieval* approach, it learns to distinguish valid requests from attacks, using tokens that normally occur in classic attacks payloads.

6.4 Scenario 2

In *sc*2 we assume to have real valid requests and generic attacks. As we have already mentioned in section 4.1, the generic attacks dataset is composed of attack requests that are not targeted to the application, instead they are targeted to other applications. In other words, to generate the training datasets, we use the valid requests from the application we are going to test and the attacks requests from the other two applications (taken as generic attacks). As we previously mentioned, attacks contained in the CSIC2010 dataset not only include attacks but also valid requests with typos. For this reason, we did not use attack requests from the CSIC2010 dataset in the experiments of this scenario. In other words, in the case of the DRUPAL and PKDD2007 dataset we only used each other attacks to simulate the generic attacks.

After generating the datasets, we have applied the expert-assisted two-class approach. We proceeded to experiment with the KNN (with K = 3) and RANDOM FOREST classifiers using the training and testing datasets generated in the previous step.

In Table 6.5, we present the results of applying this technique on the three datasets. As can be noticed, the performance in terms of TPR decreased comparing to the results of *sc*1. In the case we compare the results with our baseline, we can notice in general a major decrements in the number of attacks detected (decrements of the TPR), with a great improvement in terms of the FPR. We also validated that in the case that more generic attacks are added to the training datasets the attack detection increased without increasing the false positives. As already mentioned in Section 4.1 in order to improve the results is required that the set of generic requests (in this scenario the attack requests) characterize as much as possible the universe they represent. Once again, we can say that the RANDOM FOREST is the classifier with better performance analyzing the overall results.

This approach has a major advantage as only needs valid requests of the protected application and the attacks examples can be obtained from different applications or be artificially crafted. The results are promising, but a more complete dataset (from an attack point of view) is needed to improve the attack detection capabilities.

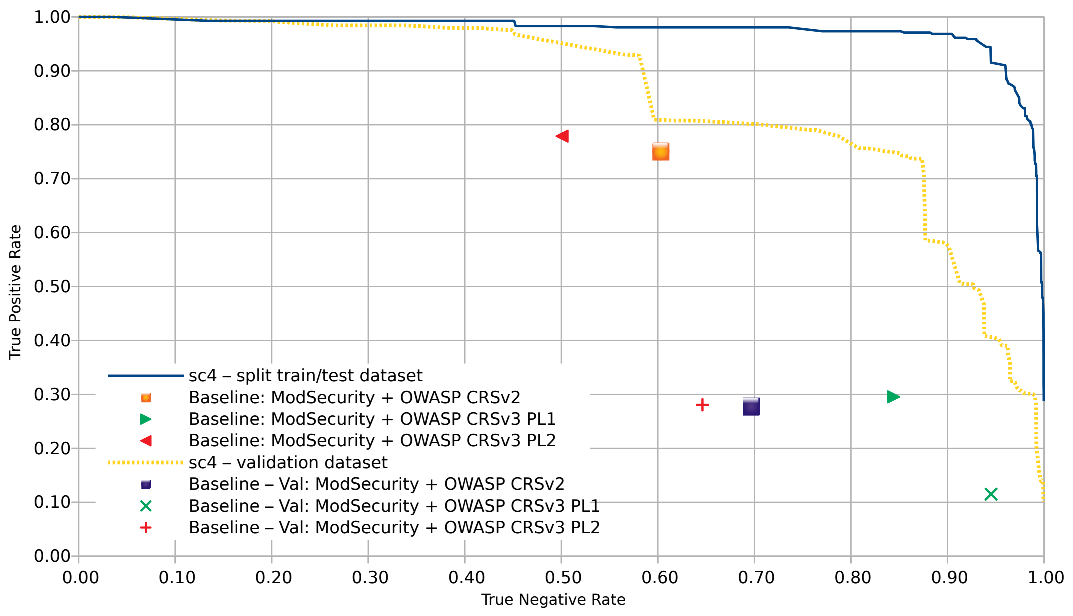

FIGURE 6.1: *sc*4: ROC curve generated by moving the threshold of the one-class approach using DRUPAL and DRUPAL validation datasets

6.5 Scenarios 4 and 5

In this section we present the evaluation of the one-class approach (described in Section 5.5) on both *sc*4 and *sc*5. We recall that *sc*4 corresponds to the case where we have a training dataset containing valid requests of the protected application and *sc*5 is the scenario where we only have valid requests, but they do not correspond to the application being protected (generic valid traffic).

The experiments have been carried out by varying the threshold to obtain 500 different operation points. In the one-class approach the threshold is governed by λ (the only free variable in Equation 4.2). During these experiments we varied λ from 0 to 1 (increasing by $1/500$) and calculated for each λ the different performance metrics (Precision, Recall, TPR and FPR). As we have 500 different operational points, we present for each dataset a figure with the results in terms of a ROC curve where the x-axis corresponds to the True Negative Rate (calculated as $1 - FPR$) and in the y-axis the TPR. In order to apply these results to protect a real application, it is required to define the value of λ. One approach to define this parameter was presented in Section 4.3.3. In each figure, the blue straight line corresponds to the results of the *sc*4, the dashed orange line corresponds to the *sc*5 and the MODSECURITY with the OWASP CRS version 2 is represented by the orange square. Finally, the different PL of the OWASP CRS version 3 are represented by triangles facing right for PL1 and left for PL2.

In Figure 6.1 we present the results for scenario *sc*4 using DRUPAL and DRUPAL Validation datasets. The blue straight line corresponds to the case of building the model and testing using the DRUPAL dataset. We can identify several operational points where the one-class approach improves MODSECURITY baseline in both TPR and FPR. For instance, if we select the operational point with the same number of TPR than MODSECURITY with the OWASP CRS version 2, we have less than 2% of FPR. For this dataset, as we did in earlier explanations, we present the results using the validation dataset as a yellow dotted line. In this case, we used the same model

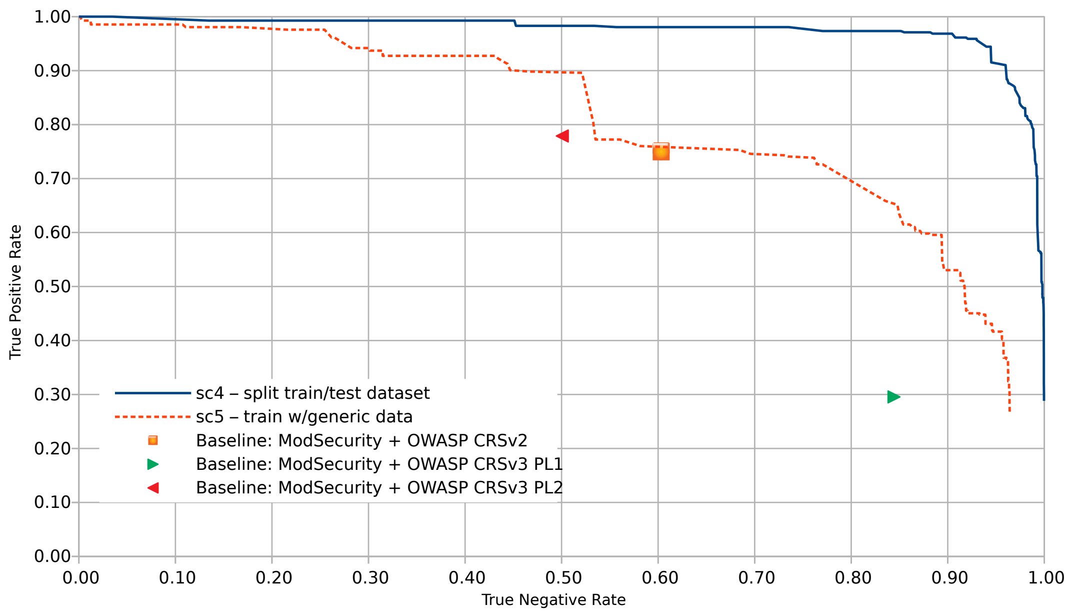

FIGURE 6.2: *sc*5: ROC curve generated by moving the threshold of the one-class approach for DRUPAL dataset

generated using the training set from the DRUPAL dataset, but testing was performed using the DRUPAL validation dataset, which corresponds to traffic logged in a subsequent interval of time. We also include the MODSECURITY baseline for validation dataset. As in the train/test case, we could identify several operational points where the one-class approach improves MODSECURITY baseline. If we select an operational point with the same TPR than MODSECURITY with the OWASP CRS version 2 (blue square), once again, the FPR is less than 2%. Is important to notice that even if the validation curve is least performant compared with the train/test case, we can also notice that the MODSECURITY results also decrease. This could be an effect of the traffic proportions of both datasets, as they come from different periods of time. In particular, the validation dataset corresponds to a weekend: we have 2/3 of the normal traffic and almost twice of attacks.

Figure 6.2 shows the comparison for the scenarios *sc*4 and *sc*5 with the DRUPAL dataset. The blue line corresponds to the scenario *sc*4 with the DRUPAL dataset. The dashed orange line corresponds to the results for *sc*5 using the same dataset. In this scenario we used valid data from the PKDD2007 and CSIC2010 dataset to train the model to be tested using the DRUPAL test set. In this case we observe a reduction compared with *sc*4, but we could still spot several operational points that improves MODSECURITY baseline. This is a promising approach in order to have a trained model that could be used to protect applications out of the box without configuration needed. These results were obtained with datasets of applications that are not related. We believe that if we use data from similar applications (for example other portals based on DRUPAL) these results can be significantly improved.

The results for the PKDD2007 dataset are presented in Figure 6.3. For the scenario *sc*4, we observe that there exist several points where the one-class behaves better than MODSECURITY with OWASP CRS 3 in PL 1. On the other hand the one-class approach does not improve any of the other baseline cases. For the *sc*5, the one-class could not capture the normal behavior of the PKDD2007 dataset, when we train using DRUPAL and CSIC2010 datasets.

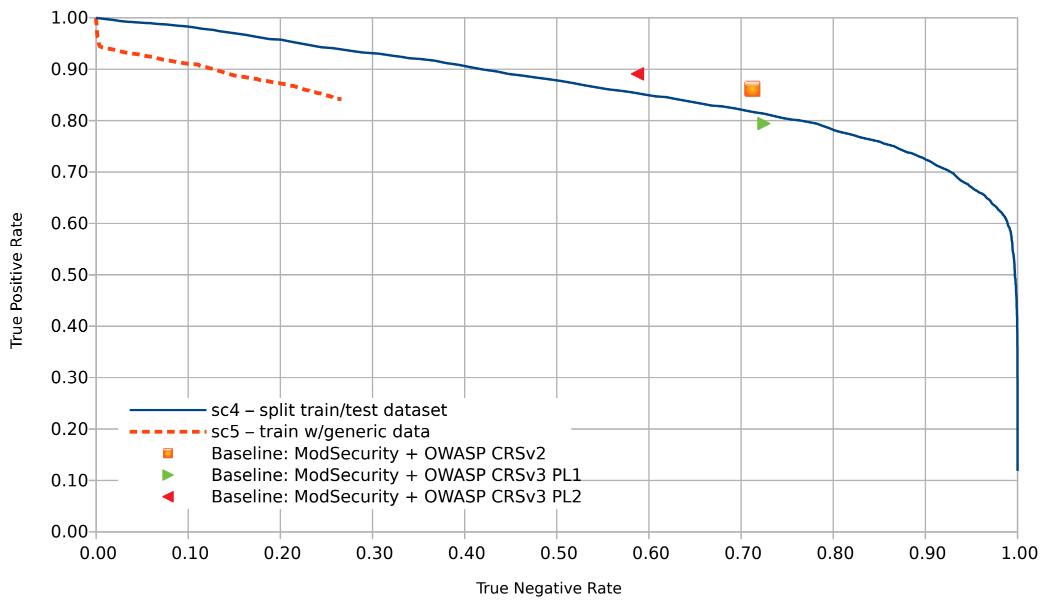

FIGURE 6.3: *sc*4 and *sc*5: ROC curve generated by moving the threshold of the one-class approach for PKDD2007 dataset

It is important to notice that the PKDD2007 dataset was subject to an anonymization process where all valid requests had the URL, parameters and values replaced with random data without any relationship. We believe that the anonymised dataset losses important characteristics that represent valid traffic of the application. We face the same problem when investigating an alternative approach [6], where we compared the one-class approach with an anomaly detection approach using n-grams. As this n-gram approach generates one model for each parameter of the application, it turned out useless when applied to the anonymized dataset. As our approach try to learn from valid data, this process generates noise which makes it harder for our algorithm to learn. On the other hand, MODSECURITY uses its rules to spot attacks. As the attack requests were not modified, we can observe high number of TPR in all of the MODSECURITY baseline metrics.

Finally Figure 6.4 presents the results for *sc*4 and *sc*5 using the CSIC2010 dataset. In both scenarios the one-class approach improves MODSECURITY with OWASP CRS version 2. Comparing the results with the OWASP CRS version 3 we could find operational points that improve the TPR or the FPR, but not both metrics simultaneously. This dataset has a special characteristic: its attacks requests are not only attacks but also anomalous traffic. Notice that MODSECURITY with the OWASP CRS, which is prepared to detect attacks, has also a low TPR. The low TPR observed in the case of the one-class is because the features selected by the expert also try to identify classic attacks payloads. In other words, the selected features did not capture this dataset anomalous traffic essence. A clear indicator of this could be observed in Table 6.3 where we have a high decrement in the TPR for the RANDOM FOREST algorithm when using the *Classic Information retrieval* compared to the *Expert assisted* approach. This shows that the blindly selected features from the dataset could easily separate valid and attack requests, but when we use features that try to identify attacks payloads the requests get mixed.

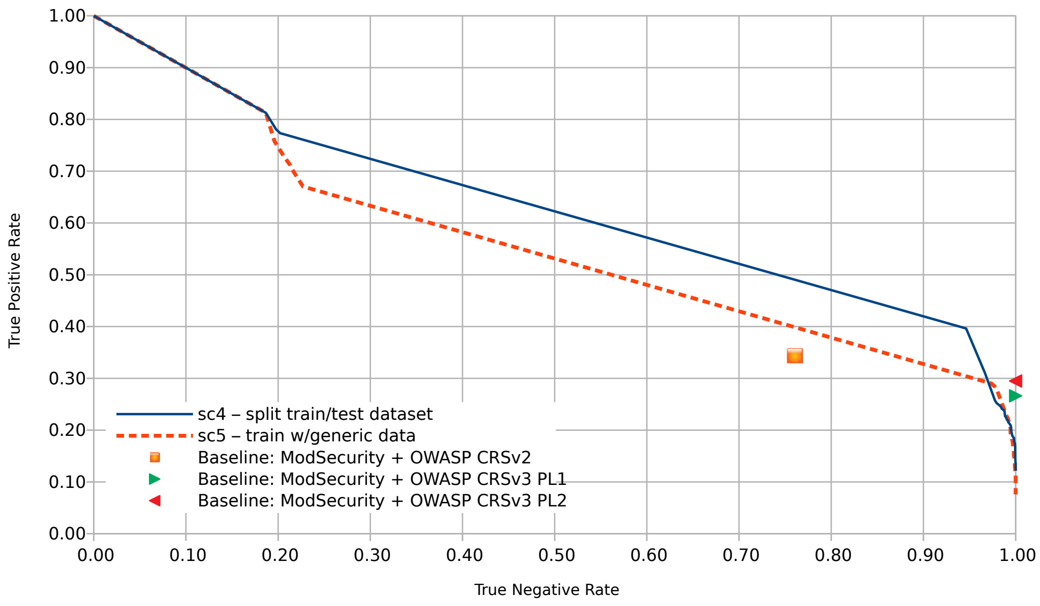

FIGURE 6.4: *sc*4 and *sc*5: ROC curve generated by moving the threshold of the one-class approach for CSIC2010 dataset

Scenario	Algorithm	Precision	Recall	TPR	FPR
Baseline MODSECURITY	OWASP CRS version 2	0.56	0.86	86.17%	28.76%
	OWASP CRS version 3 PL1	0.64	0.79	79.43%	27.53%
	OWASP CRS version 3 PL2	0.57	0.89	89.09%	41.25%
*sc*1: Classic information retrieval	KNN-3	0.97	0.72	71.78%	0.98%
	RANDOM FOREST	0.96	0.89	**88.65%**	**1.76%**
	SVM	0.96	0.87	86.79%	1.44%
*sc*1: Expert assisted	KNN-3	0.95	0.73	72.73%	1.54%
	RANDOM FOREST	0.96	0.82	**82.49%**	**1.59%**
	SVM	0.94	0.72	71.58%	1.84%
*sc*2: Expert assisted	RANDOM FOREST	0.96	0.23	22.56%	0.45%
	KNN-3	1.00	0.12	11.70%	0.02%
*sc*4	One-class w/λ=0.246	0.57	0.81	**80.96%**	**26.58%**
*sc*5	One-class	N/A	N/A	N/A	N/A

TABLE 6.6: Summarized results for the PKDD2007 dataset

6.6 Discussion

We recall once more an important premise of our work: false positives of MODSECURITY configured with OWASP CRS out of the box often lead to a denial of service to valid users. We have presented four approaches for addressing the problem of attack detection in web applications. We validated our approaches using two public datasets and one built during this work. In what follows we will discuss the results for each dataset.

As we have already mentioned, the PKDD2007 dataset was built to be used in a machine learning challenge in 2007. Given the way this dataset was created two particularities that could affect the results arise: the anonymization process of valid traffic and how attacks were generated. Valid traffic was logged from an application and it was artificially sanitized throughout a process of random transformation of the data. On the other hand, the attacks contained in this dataset were artificially created. Table 6.6 presents the summarized results for the baseline and our

Scenario	Algorithm	Precision	Recall	TPR	FPR
Baseline MODSECURITY	OWASP CRS version 2	0.50	0.34	34.32%	23.93%
	OWASP CRS version 3 PL1	1.00	0.27	26.62%	0.00%
	OWASP CRS version 3 PL2	1.00	0.29	29.48%	0.00%
*sc*1: Classic information retrieval	KNN-3	0.99	0.57	57.30%	0.17%
	RANDOM FOREST	0.98	0.68	**67.76%**	**0.36%**
	SVM	0.99	0.68	67.87%	0.34%
*sc*1: Expert assisted	KNN-3	0.94	0.39	38.67%	0.80%
	RANDOM FOREST	0.94	0.39	**39.45%**	**0.83%**
	SVM	0.84	0.35	35.17%	2.24%
*sc*2: Expert assisted	RANDOM FOREST	0.91	0.32	**32.06%**	**2.27%**
	KNN-3	0.66	0.50	50.43%	17.85%
*sc*4	One-class w/λ=0.08	0.72	0.40	**39.63%**	**5.37%**
*sc*5	One-class w/λ=0.128	0.79	0.29	**29.02%**	**2.64%**

TABLE 6.7: Summarized results for the CSIC2010 dataset

four approaches. For the scenario *sc*1, in both cases (*Classic information retrieval* and the *Expert assisted*), the three algorithm used show good performance scores. In all cases, we could observe values of TPR similar to the baseline, at the same time, the FPR has major improvement decreasing from 28%-41% to 1%-2% comparing the baseline and our results respectively. The results corresponding to the scenario *sc*2 show good performance scores in terms of FPR, but the TPR (detection of attacks) scores are bad. In this scenario we trained the model using a generic attack dataset. The generic attack dataset used was composed only by the DRUPAL attack training instances (only 905 instances). We believe that the few instances and attack type examples contained in this generic attack dataset do not allow algorithms to find the optimal limit to distinguish valid requests from attacks. Finally Figure 6.3 shows all operational points for the one-class approach (*sc*4 and *sc*5) using this dataset, in Table 6.6 we present for *sc*4 the operational point with $\lambda = 0.246$ which corresponds to the case where the FPR is similar to the baseline. We observed that the OWASP CRS approach has a high number of TPR (over 80%). For this particular dataset, even if the one-class approach produced good performance scores, they are lower than MODSECURITY with the OWASP CRS. This result is due to the fact that the attacks contained in this dataset are easy to detect by MODSECURITY and the sanitized valid traffic prevented the one-class approach to model normal behavior. The results for *sc*5 are not good, since the model generated using traffic of other applications does not allow the one-class approach to characterize the behavior of randomly generated data. In sum, in most of the scenarios proposed showed good performance result on the PKDD2007 dataset. The poor results correspond to the scenario where no comprehensive data for training (*sc*2) were available or due to the process of random anonymization of the dataset (*sc*5).

The second dataset used during our experiments is the CSIC2010 dataset. This dataset was also artificially created as a benchmark to test web application protection mechanisms. In Table 6.7 we present the results corresponding to the baseline and our approaches on this dataset. A relevant property of this dataset is that the labels used are anomalous and normal traffic. Anomalous traffic is compose not only of attacks, but also unintentional illegal requests that do not have malicious intention (for example valid requests with typos). Once again the *Classic information retrieval* approach in *sc*1 showed good performance results where almost no FP where found and the attack detection double the baseline (from 60% to 30%). Both *Expert assisted* approaches (*sc*1 and *sc*2) showed a significant reduction of the TPR in almost a half, from 68% to 39%. This last value is very close to the baseline (OWASP CRS version

Scenario	Algorithm	Precision	Recall	TPR	FPR
Baseline ModSecurity	OWASP CRS version 2	0.03	0.75	75.00%	39.68%
	OWASP CRS version 3 PL1	0.04	0.30	29.55%	15.57%
	OWASP CRS version 3 PL2	0.03	0.78	77.89%	49.93%
*sc*1: Classic information retrieval	KNN-3	0.97	0.91	91.36%	0.05%
	Random Forest	0.97	0.94	**93.98%**	**0.05%**
	SVM	0.98	0.92	91.88%	0.04%
*sc*1: Expert assisted	KNN-3	0.95	0.88	88.48%	0.08%
	Random Forest	0.96	0.92	**91.88%**	**0.07%**
	SVM	0.91	0.66	66.49%	0.13%
*sc*2: Expert assisted	Random Forest	0.95	0.48	**47.57%**	**0.05%**
	KNN-3	0.90	0.07	6.99%	0.01%
*sc*4	One-class w/λ=0.176	0.25	0.94	**94.43%**	**6.00%**
*sc*5	One-class w/λ=0.38	0.06	0.74	**73.85%**	**23.85%**

TABLE 6.8: Summarized results for the DRUPAL dataset

2 and OWASP CRS version 3 with PL 1 and PL 2). We believe that this behavior is due to the fact that we used the anomalous label as attack traffic. Is important to consider that both approaches, the *Expert assisted* and the OWASP CRS, focus on attacks payload detection. For this reason, when we compare the features of a valid request with an anomalous request (created by introducing a typo error into a valid request) we can not differentiate among them. In Figure 6.4 we present the results for scenarios *sc*4 and *sc*5, where there are several operational points with good performance scores. Once again the definition of anomalous traffic leads to a strange behavior in both scenarios. We are convinced that makes the one-class approach to confuse valid requests with anomalous ones. It can be noticed that a small change in the threshold produces a great fall in the attack detection and a at the same time a great decrement of the FPR.

Both public datasets used during these experiments share two disadvantages: they are created artificially and are quite old. We have already mentioned different difficulties generated because of the artificial nature of theses datasets. But it is also important to note that web applications and attacks on them have evolved a lot in the last 10 years. For these reasons we have decided to create our own dataset, where we recorded traffic to a production site. Both valid and attack requests are real attacks to this production web application. Table 6.8 presents the results for this dataset. The *Classic information retrieval* and *Expert assisted* approaches in scenario *sc*1 showed great results with high TPR and low FPR compared to the baseline. Scenario *sc*2 shows good results in terms of FPR, but a reduction on the attacks detection (reduction of the TPR). Once again, we are convinced that this behavior relates to the way we construct the generic attack dataset by using only attacks samples from the datasets that we had at hand. In this particular case, the attack dataset it is only compose of instances from the training set of the PKDD2007 dataset. Finally in Figure 6.2 we present the results for scenarios *sc*4 and *sc*5. For this dataset, we found several operational points that outperform the ModSecurity baseline. In Table 6.8 we present the operational point that maximizes the TPR and at the same time minimizes the FPR. In *sc*4 with $\lambda = 0.176$ the TPR improves in almost 20% the baseline and the FPR decrements from around 40% to 6%. Finally, for *sc*5 we select $\lambda = 0.38$, where the TPR is similar to the baseline, but we observed a reduction of the FPR to almost half from 40% to 24%. In sum, this dataset created with real traffic to a modern web application presents good performance results for the one-class approach, not only using data from the application, but also training with traffic from others applications. We believe that these results can be drastically improved if the data used for training, even if it is from other applications, it is from applications with similar characteristics (e.g. in this case using data from other Drupal portals).

Chapter 7

Conclusion and further work

7.1 Further work

The major objective of this work was to validate the use of machine learning techniques to protect web applications. During the work some design decisions were taken in order to narrow the scope. In Section 4.4 we presented some limitations. In what follows we will outline how these limitations can be addressed.

Many web application attacks bypass controls by using different encoding mechanisms simultaneously (as presented in Section 4.4). In our parser we only decode URL encoding strings, other encoding mechanisms are not supported. This behavior may allow an attacker to bypass our machine learning model. Special care has to be taken during the parsing phase. One solution could be simply to reject requests that have multiple encoding. It is also possible to use a series of transformations in order to decode different formats, before parsing the requests.

Some other attacks aim to the HTTP protocol by manipulating the header. In our work, some headers are filtered before doing training/classification. This decision was taken so data in these headers will not generate an overfitting of the algorithms to the training dataset. Further analysis on how this information could be included in the models is required.

The datasets we have at hand when developing our work have classic web application requests, where the bodies only include *application/x-www-form-urlencoded* content types. Other type of body content types where not tested during this work. ModSecurity implements different body parsers depending on the content-type. We plan to work on an integration method, where we could use the body parsers included in ModSecurity before invoking our models. In this way, we would use the ModSecurity parsers enhancing performance since the body is parsed only once.

One of the biggest challenges we had to face when conducting this work is the lack of publicly available datasets with complete HTTP requests and classified at least as valid or attacks. We were able to find only three datasets, two of them were used to experiment (see Section 4.2) and the third one (1998 DARPA Intrusion Detection Evaluation Data Set) was discarded since it was constructed based on network traffic, not only web application traffic. Additionally, these datasets are at least 10 years old. Given the advance in the last years of the web application technologies the traffic has dramatically evolved, both in number and kind. We think those datasets no longer represent the current state of the technologies. We plan to continue working on the construction of new datasets in order to produce new examples of attacks and valid application traffic that represents nowadays applications. Along with these new datasets, we plan to experiment with scenario *sc*3 as well as repeat the results of *sc*2 and *sc*5 that we believe could be improved by using more complete and newer requests, as discussed in Section 6.6.

From the obtained results we identify that a greater understanding of the masking mechanism applied to the PKDD2007 dataset is required since it can have an impact on the applied techniques. This confirms the need to have new datasets.

It has been shown that in order to apply the mechanisms implemented in scenarios *sc*4 and *sc*5 to protect an application it is necessary to define the threshold. The threshold is governed by the λ variable (which is the only free variable in Equation 4.2). In Section 4.3.3 we proposed to define λ by fixing the number of False Positives that we are willing to accept. We plan to continue studying the distributions of the intra-cluster distances to estimate the threshold by understanding its distribution. As we are grouping valid requests with several instances of Gaussian Mixture Model and the distances are measured using the Mahalanobis distance, we believe that the distribution of the distances should approximate into a chi-square distribution. We are convinced that a deeper understanding of that distribution would allow us to find a method to define λ in a theoretically grounded way.

A first proof-of-concept (PoC) was developed that integrates the one-class approach with ModSecurity (see Section 5.6.1), but this integration could be improved by taking into account performance and interoperability requirements. This PoC shows that the integration is feasible, but several performance issues were found. We also identifyed some interoperability requirements, for example, have a plug-able method where several machine learning models could be use simultaneously in one ModSecurity instance. Some combination method is needed in order to combine the results of the rules with the different machine learning modes. In [38] we propose an integration architecture which takes into account the requirements presented.

Another line of work is to study the special case of scenario *sc*6, where only attacks requests are available. This scenario has the special characteristic that is like the OWASP CRS approach, where the rules try to identify attack patterns. In order to work in this scenario datasets with more attack requests are needed.

7.2 Conclusions

We are interested in applying machine learning and pattern recognition techniques to improve the detection capabilities of the WAF ModSecurity giving particular importance to the task of diminishing the false positives generated by this tool when is set out to protect a web application. We provide a characterization of the problem by identifying different scenarios depending on the availability of data to train the learning models. The scenarios vary from the rare, but best case, where we have a dataset with real application traffic to more practical scenarios where we have only valid request to an application that could be collected, for instance, during the functional testing phase. We also studied the more flexible scenario where a model could be generated using generic valid requests, allowing to protect applications with a generic out of the box model.

We have presented four different approaches for addressing the problem of attack detection in web applications, depending on the scenario analyzed.

The first approach, in which the two-class paradigm is used, resulted in very good performance scores (see Section 6.3). The fact that KNN provided good results, very close to the ones of Random Forest and SVM, indicates that if real samples of valid requests and attacks are available, the classification problem is not very complex. That is, the Bayes error (the theoretical lowest error attainable knowing the class distributions) is not large. However, this approach has two limitations: labeled data from both classes are needed in order to train the classifiers and a classifier designed for one dataset can not be used with another one.

The second approach is also based on a two-class paradigm, but in this scenario the attack traffic came from a generic dataset constructed from attacks to other applications. This approach has the advantage that only real valid traffic from the web application is needed for training. In addition, a low rate of false positive was obtained, but the performance on attack detection decreased (see Section 6.4). This indicates that the generic attack dataset that we have built does not include enough attacks examples and therefore the classification boundaries do not capture the optimal solution as can be done in $sc1$. We are convinced that this behavior relates to the way we construct the generic attack dataset by using only attacks samples from the datasets that we had at hand. In other words, for this approach to work we need to build a generic attack training dataset by adding more attacks examples. It is not possible to know in advance how many attacks samples have to be added or which type of attacks are needed in order to improve this approach. However, given the good TPR rates obtained in $sc1$ we believe that if we manage to construct appropriate datasets good results might be achieved.

In both approaches ($sc1$ and $sc2$), we identify that the RANDOM FOREST classifier has better performance. For this reason, we plan to continue working in an integration approach where the RANDOM FOREST classifier is used to generate specific rules for the trained application.

The third, more realistic approach, uses only valid requests to construct a detection model. In this case, the outcome is quite promising, the results outperform the ones obtained with the classic rule based MODSECURITY solution with OWASP CRS version 2 and in some of the cases it also improves OWASP CRS version 3. As we have seen in Section 6.5 this one-class approach reduced the number of false positives while not greatly increasing the false negatives. Furthermore, this approach has a threshold that can be tuned depending on the number of false positives that we are willing to accept (see Figures 6.1, 6.3 and 6.4). We plan to experiment with one-class algorithms like SVM, instead of using classic distances.

Finally, we extend the one-class approach by using generic data to train the classifier. The preliminary results are promising as we improve MODSECURITY with OWASP CRS version 2 in two of the three datasets. This approach has a major advantage because we can use models already trained in order to protect new applications out of the box. We believe that these preliminary results can be drastically improved if the data used for training, even if it is from other applications, it is from applications with similar characteristics. For example, training the model using traffic from different web portals based on *DRUPAL* technology could be used to protect a new instance of this technology. It is important to note that in these experiments the datasets used correspond to applications of very different nature, both in their functionalities and in their age. We plan to improve this scenario by training generic models for different types of applications. For example, create a generic model for Drupal-like sites.

Bibliography

[1] D. Aha and D. Kibler. "Instance-based learning algorithms". In: *Machine Learning* 6 (1991), pp. 37–66.

[2] *Analyzing Web Traffic: ECML/PKDD 2007 Discovery Challenge*. http://www.lirmm.fr/pkdd2007-challenge/.

[3] Davide Ariu et al. "McPAD and HMMWeb: two different approaches for the detection of attacks against Web applications". In: *Italian Workshop on Privacy and Security*. Rome, 2008.

[4] A Barth and UC Berkley. "HTTP State Management Mechanism (RFC 6265)". In: *Internet Engineering Task Force (IETF)* (2011).

[5] Tim Berners-Lee, Roy Fielding, and Larry Masinter. "Uniform resource identifier (uri): Generic syntax (RFC 3986)". In: *Network Working Group* (2005).

[6] Gustavo Betarte et al. "Improving Web Application Firewalls through Anomaly Detection". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 779–784.

[7] Gustavo Betarte et al. "Machine learning-assisted virtual patching of web applications". In: *arXiv preprint arXiv:1803.05529* (2018).

[8] Christian Bockermann, Martin Apel, and Michael Meier. "Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling (Extended Abstract)." In: *DIMVA*. Ed. by Ulrich Flegel and Danilo Bruschi. Vol. 5587. Lecture Notes in Computer Science. Springer, 2009, pp. 196–205.

[9] T.. Bray. "The JavaScript Object Notation (JSON) Data Interchange Format". In: (2017).

[10] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[11] Christopher JC Burges. "A tutorial on support vector machines for pattern recognition". In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[12] *Common Weakness Enumeration, a community-developed dictionary of software weakness types*. 2019. URL: https://cwe.mitre.org/index.html.

[13] Igino Corona, Davide Ariu, and Giorgio Giacinto. "HMM-Web: A Framework for the Detection of Attacks Against Web Applications". In: *Proceedings of ICC 2009*. 2009, pp. 1–6.

[14] Robert K Cunningham et al. *Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation*. Tech. rep. MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 1999.

[15] Francisco Curbera et al. "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI". In: *IEEE Internet computing* 6.2 (2002), pp. 86–93.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875.

[17] *Drupal*. 2018. URL: https://www.drupal.org.

[18] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[19] Juan M. Estévez-Tapiador, Pedro García-Teodoro, and Jesús E. Díaz-Verdejo. "Measuring normality in HTTP traffic for anomaly-based intrusion detection". In: *Computer Networks* 45.2 (2004), pp. 175 –193. ISSN: 1389-1286.

[20] Matthieu Exbrayat. "ECML/PKDD challenge: analyzing web traffic a boundaries signature approach". In: 2007, p. 53.

[21] Joseph Fialli and Sekhar Vajjhala. "The Java architecture for XML binding (JAXB)". In: *JSR Specification, January* (2003).

[22] Christian Folini. *Handling False Positives with the OWASP ModSecurity Core Rule Set*. 2016. URL: https://www.netnea.com/cms/apache-tutorial-8_handling-false-positives-modsecurity-core-rule-set/.

[23] Christian Folini and Ivan Ristic. *Modsecurity handbook*. Feisty Duck, 2017.

[24] Brian Gallagher and Tina Eliassi-Rad. *Classification of http attacks: a study on the ECML/PKDD 2007 discovery challenge*. Tech. rep. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2009.

[25] Network Working Group et al. "RFC 2616 Hypertext Transfer Protocol–HTTP/1.1". In: *R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee* (1999).

[26] Mark Hall et al. "The WEKA data mining software: an update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[27] *HTML URL Encoding Reference*. URL: https://www.w3schools.com/tags/ref_urlencode.asp.

[28] *HTTP DATASET CSIC 2010*. URL: http://www.isi.csic.es/dataset/.

[29] Kenneth L Ingham and Hajime Inoue. "Comparing anomaly detection techniques for http". In: *RAID*. Vol. 4637. Springer. 2007, pp. 42–62.

[30] A. K. Jain, R. P. W. Duin, and Jianchang Mao. "Statistical pattern recognition: a review". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.1 (2000), pp. 4–37. ISSN: 0162-8828. DOI: 10.1109/34.824819.

[31] Debabrata Kar, Suvasini Panigrahi, and Srikanth Sundararajan. "SQLiDDS: SQL injection detection using document similarity measure". In: *Journal of Computer Security* 24.4 (2016), pp. 507–539.

[32] Debabrata Kar, Suvasini Panigrahi, and Srikanth Sundararajan. "SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM". In: *Computers & Security* 60 (2016), pp. 206–225.

[33] Shehroz S Khan and Michael G Madden. "A survey of recent trends in one class classification". In: *Irish conference on Artificial Intelligence and Cognitive Science*. Springer. 2009, pp. 188–197.

[34] Sotiris B Kotsiantis. "Supervised machine learning: A review of classification techniques". In: (2007).

[35] Christopher Kruegel and Giovanni Vigna. "Anomaly detection of web-based attacks". In: *Proceedings of CCS 2003*. ACM. 2003, pp. 251–261. ISBN: 1-58113-738-9.

[36] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[37] Prasanta Chandra Mahalanobis. “On the generalized distance in statistics”. In: National Institute of Science of India. 1936.

[38] Rodrigo Martinez. “Enhancing web application attack detection using machine learning”. In: *8th Latin-American Symposium on Dependable Computing* (2018).

[39] Commons Math. “The apache commons mathematics library”. In: *Np, nd Web* 9 (2016).

[40] Lincoln Laboratory MIT. *DARPA Intrusion Detection Evaluation - Intrusion Detection Attacks Database*. https://www.ll.mit.edu/ideval/docs/attackDB.html. 2012.

[41] Tom M Mitchell et al. “Machine learning. 1997”. In: *Burr Ridge, IL: McGraw Hill* 45.37 (1997), pp. 870–877.

[42] OWASP. *Open Web Application Security Project*. URL: https://www.owasp.org.

[43] OWASP. *OWASP ModSecurity Core Rule Set Project*. URL: https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project.

[44] OWASP. *OWASP Top Ten Project*. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[45] OWASP. *OWASP Zed Attack Proxy (ZAP)*. URL: https://www.owasp.org.

[46] *OWASP AppSec Europe 2018*. URL: https://2018.appsec.eu.

[47] Konstantinos Pachopoulos et al. “Feature extraction from web traffic data for the application of data mining algorithms in attack identification”. In: Citeseer. 2007.

[48] J. Platt. “Fast Training of Support Vector Machines using Sequential Minimal Optimization”. In: *Advances in Kernel Methods - Support Vector Learning*. Ed. by B. Schoelkopf, C. Burges, and A. Smola. MIT Press, 1998. URL: http://research.microsoft.com/\~jplatt/smo.html.

[49] Magic Quadrant. “Magic Quadrant for Web Application Firewalls”. In: *Analyst (s)* (2017), G00314552.

[50] Jeff Forristal (signing as rain.forest.puppy). “NT Web Technology Vulnerabilities”. In: *Phrack Magazine* 8.54 (1998).

[51] Chedy Raıssi et al. “Web analyzing traffic challenge: description and results”. In: *Proceedings of the ECML/PKDD*. 2007, pp. 47–52.

[52] Andrs Riancho. “w3af-web application attack and audit framework”. In: *World Wide Web electronic publication* (2011), p. 21.

[53] Leonard Richardson and Sam Ruby. *RESTful web services*. " O’Reilly Media, Inc.", 2008.

[54] Gerard Salton and Michael J McGill. “Introduction to modern information retrieval”. In: (1986).

[55] SpiderLabs/ModSecurity. *Reference Manual*. URL: https://github.com/spiderlabs/modsecurity/wiki/reference-manual.

[56] Dafydd Stuttard and Marcus Pinto. *The web application hacker’s handbook: Finding and exploiting security flaws*. John Wiley & Sons, 2011.

[57] Carmen Torrano-Gimenez, Alejandro Perez-Villegas, G Álvarez Marañón, et al. "An anomaly-based approach for intrusion detection in web traffic". In: *Journal of Information Assurance and Security* 5.4 (2010), pp. 446–454.

[58] Inc. Trustwave Holdings. *ModSecurity: Open Source Web Application Firewall*. URL: http://www.modsecurity.org/.

[59] Verizon. *Podcast: Advantages of Web Application Firewalls and open-source WAF*. URL: https://www.verizondigitalmedia.com/blog/2017/11/advantages-of-web-application-firewalls-and-open-source-waf/.

[60] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. "Anagram: A content anomaly detector resistant to mimicry attack". In: *RAID 2006*. Vol. 4219. Springer. 2006, pp. 226–248.

[61] Ke Wang and Salvatore J Stolfo. "Anomalous payload-based network intrusion detection". In: *RAID 2004*. Vol. 3224. Springer. 2004, pp. 203–222.

[62] Ian H Witten et al. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[63] Christian Wressnegger et al. "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification". In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. AISec '13. New York, NY, USA: ACM, 2013, pp. 67–76.

[64] Yiming Yang and Jan O Pedersen. "A comparative study on feature selection in text categorization". In: *Icml*. Vol. 97. 1997, pp. 412–420.

[65] Hubert Zimmermann. "OSI reference model-the ISO model of architecture for open systems interconnection". In: *IEEE Transactions on communications* 28.4 (1980), pp. 425–432.