2012

Browsers Anti-XSS methods in ASP (classic) have been defeated!

The intention of this paper is to prove the client-side XSS protection methods must have rules for different web application languages, otherwise they will be bypassed. This research is based on ASP classic web applications, but it can be performed in other web application languages as well.

Soroush Dalili - @irsdl
www.SecProject.com
irsdl at yahoo dot com

June 2012

Browsers Anti-XSS methods in ASP (classic) have been defeated!

This time, I want to start with the summary section first to break the rules!

Summary

The intention of this paper is to prove the client-side XSS protection methods must have rules for different web application languages, otherwise they will be bypassed. This research is based on ASP classic web applications, but it can be performed in other web application languages as well.

Introduction

I researched different methods of sending inputs to an ASP (classic) page. I found out that almost all of the browsers' Anti-XSS protection methods are not aware of different features of ASP that accept the inputs; therefore, all of them can be bypassed.

Note: NoScript has already added all of these rules to its application and it is more secure than the others currently (thanks to Giorgio Maone for patching the application as quickly as possible). IE9 has better sense about ASP than Google Chrome, but it does not still have all the rules.

Description

In order to make you more interested, I will start with two examples:

Example 1: Do you think Anti-XSS methods should detect this easy XSS attack?

http://www.sdl.me/xssdemo/getxss.asp?**input1**=<script/&**&input1**=FOOBAR&**input1**=>alert('@IRSDL');</script>

Please try it in IE8/9/10 and Google Chrome to see the result.

Example 2: What about this?

http://www.sdl.me/xssdemo/getxss.asp?**input1**=<script/&**in%u2119ut1**=>al%u0117rt('@IRSDL')</script/

Example 3: Or, sometimes, the bypass can be complicated! This is how I solved my XSS1 and XSS2 questions with a single solution in SecProject.com Challenge Series 1:

http://sdl.me/challenge1/xss1/JsChallenge1.asp?**I%%NPUT2**=Somet%%hing&**iN%%PUT2**=')1&**inP%%UT2%00%00**=1};lt=1;1&**In%u2119ut2**=1%26<1&**input2**=0<ale%%rt(/AWESOME_IRSDL/&**in%u2119U%%T2%00%00%0%%0%00%0%%0**=1);1&**in%u2119uT%%2%00**=1;i%%f(0&**in%u2119ut2%%**=1){{1&**I%%n%%PuT2%00%00%00**=1/%%*%%/&**iN%%p%%Ut2**=1/%%/

And

http://sdl.me/challenge1/xss2/JsChallenge2.asp?**I%%NPUT1**=Somet%%hing&**iN%%PUT1**=')1&**inP%%UT1%00%00**=1};lt=1;1&**In%u2119ut1**=1%26<1&**input1**=0<ale%%rt(/AWESOME_IRSDL/&**in%u2119U%%T1%00%00%0%%0%00%0%%0**=1);1&**in%u2119uT%%1%00**=1;i%%f(0&**in%u2119ut1%%**=1){{1&**I%%n%%PuT1%00%00%00**=1/%%*%%/&**iN%%p%%Ut1**=1/%%/

As you see, I am only using 1 input parameter to bypass everything! (Note: this special page in xss1 converts “<” and “>” to “<” and “>” which was used to bypass NoScript as well – it is not a NoScript bug)

Why can you bypass XSS protections? I will tell you now.

Interesting ASP Input Features

1- HTTP Parameter Pollution (HPP): ASP is one of the web application languages which can receive several inputs with one single name. Although this feature was/is used legitimately in some of the web applications, it can be useful for attackers to bypass some restrictions as well [1].

2- Certain UTF-8 characters will be transformed to their ASCII equivalents [2], [3]. It can be used in both of parameter names and their values. Therefore, “inPut1=<scriPt/>” is equal to “%u0131n%u2119ut1=%u3008scr%u0131%u2119t>”

3- Parameter names in ASP are not case sensitive. Therefore, “input1” is equal to “InPuT1”.

4- Anything after the Null character will be ignored in parameter names and their values. Therefore, “input1=test” is equal to “input1%00Something=test%00Anything”

5- Percentage characters (“%”) will be ignored when there is no Hex value after them in parameter names and their values. Therefore, “input1=test” is equal to “%input1%=t%%est%”

6- When a parameter name after the ampersand character (“&”) is not followed by an equal sign (“=”), ASP does not count it as a separate input. As a result, in “?&input1=test” the parameter name is “&input1”; or, in “?&input1&input1=test” the parameter name is “&input1&input1”.

Bypassing browsers Anti-XSS protections

Now we know many different interesting features of ASP. We can mix these features together to bypass the browsers protections which do not understand these rules. Please see the above examples again to identify the feature types which have been used.

Note 1: URL Encoding can be used in ASP to obfuscate the attack.

Note 2: Many UTF-8 vectors such as “%u1111” will be translated to “?” in ASP which can be used in JavaScript.

Note 3: Normally, a UTF-8 encoded string should have a lowercase “u”. Therefore, “%u0041” (which is “A”) is not equal to “%U0041” (which is “U0041”). However, sometimes server configurations can make these equal!

Note 4: If you have more than 1 input (multi-injection), reordering the input parameters may bypass the protections (input disorder method [4]).

Finally

Please let me know via twitter or email if you know or have found any other interesting features.

This research was based on ASP classic language. However, other languages such as PHP can be studied in the same way; for example, PHP ignores spaces before the parameter names and anything

after the “[]” or a null character (“%00”) in the parameter names, or in PHP, space, dot, and a lone square-bracket characters (“ .[”) in parameter names will be converted to an underscore character (“_”).

References

[1] HTTP Parameter Pollution, URL:
https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

[2] NoScript New Bypass Method by Unicode in ASP, URL:
http://soroush.secproject.com/blog/2010/08/noscript-new-bypass-method-by-unicode-in-asp/

[3] Lost in Translation (ASP’s HomoXSSuality), URL: http://hackademix.net/2010/08/17/lost-in-translation-asps-homoxssuality/

[4] SecProject Web AppSec Challenge Series 1 Results, URL:
http://soroush.secproject.com/blog/2012/06/challenge-series-1-result-and-conclusion/