double escape sequence inside a url : The request filtering module is configured to deny a request that contains a double escape sequence

Asked 13 years, 2 months ago Modified 1 year, 3 months ago Viewed 113k times

112

On my ASP.NET MVC application, I am trying to implement a URL like below :

> /product/tags/for+families

When I try to run my application with default configurations, I am getting this message with 404.11 Response Code :

> **HTTP Error 404.11 - Not Found**
>
> The request filtering module is configured to deny a request that contains a double escape sequence.

I can get around with this error by implementing the below code inside my web.config :

```
<system.webServer>
  <security>
    <requestFiltering allowDoubleEscaping="true" />
  </security>
</system.webServer>
```

So, now I am not getting any `404.11`.

What I am wondering is that what kind of security holes I am opening with this implementation.

BTW, my application is under `.Net Framework 4.0` and running under `IIS 7.5`.

asp.net asp.net-mvc asp.net-mvc-3 iis iis-7

Share Improve this question Follow

asked Oct 12, 2011 at 11:18
tugberk
58.4k ● 69 ● 250 ● 342

Is it possible to reach the desired resource using `/product/tags/for%20families` instead? Then you have a workaround for ids containing spaces. Or am I completely off here? – Anders Marzi Tornblad Oct 12, 2011 at 12:25

1 @atornblad a little bit off I guess. My question : **What I am wondering is that what kind of security holes I am opening with this implementation.** – tugberk Oct 12, 2011 at 12:28

6 IIS is throwing on the "+" character, which is Microsoft's default behavior. – Todd Shelton Apr 8, 2016 at 21:48

7 Answers

Sorted by: Highest score (default)

63

The security holes that you might open up have to do with code injection - HTML injection, JavaScript injection or SQL injection.

The default settings protect you from attacks semi-efficiently by not allowing common injection strategies to work. The more default security you remove, the more you have to think about what you do with the input provided through URLs, GET request querystrings, POST request data, HTTP headers and so on...

For instance, if you are building dynamic SQL queries based on the `id` parameter of your action method, like this:

```
public ActionResult Tags(string id)
{
    var sql = "SELECT * FROM Tags Where tagName = '" + id + "'";
    // DO STUFF...
}
```

(...which is **NOT** a good idea), the default protection, put in place by the .NET framework, might stop some of the more dangerous scenarios, like the user requesting this URL:

```
/product/tags/1%27;drop%20table%20Tags;%20--
```

The whole idea is to treat every part of urls and other inputs to action methods as possible threats. The default security setting does provide some of that protection for you. Each default security setting you change opens up for a little more potential badness that you need to handle manually.

I assume that you are not building SQL queries this way. But the more sneaky stuff comes when you store user input in your database, then later displaying them. The malevolent user could store JavaScript or HTML in your database that go out unencoded, which would in turn threaten other users of your system.

Share

Improve this answer

Follow

edited Sep 11, 2018 at 14:59

answered Oct 12, 2011 at 15:04

Anders Marzi Tornblad

19.3k ● 9 ● 52 ● 66

16

Security Risk

The setting `allowDoubleEscaping` only applies to the `path` (cs-uri-stem) and is best explained by [OWASP Double Encoding](). The technique is used to get around security controls by URL Encoding the request twice. Using your URL as an example:

```
/product/tags/for+families --> /product/tags/for%2Bfamilies -->
/product/tags/for%252Bfamilies
```

Suppose there are security controls specifically for `/product/tags/for+families`. A request arrives for `/product/tags/for%252Bfamilies` which is the same resource though is unchecked by the aforementioned security controls. I used the generalized term of security controls because they could be anything such as requiring an authenticated user, checking for SQLi, etc.

Why Does IIS Block?

The plus sign (+) is a reserved character per [RFC2396]():

> Many URI include components consisting of or delimited by, certain special characters. These characters are called "reserved", since their usage within the URI component is limited to their reserved purpose. If the data for a URI component would conflict with the reserved purpose, then the conflicting data must be escaped before forming the URI.
>
> ```
>  reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
> ```

```
                 "$" | ","
```

Wade Hilmo has an excellent post titled [How IIS blocks characters in URLs](). There's lots of information and background provided. The part specifically for the plus sign is as follows:

> So allowDoubleEscaping/VerifyNormalization seems pretty straightforward. Why did I say that it causes confusion? The issue is when a '+' character

> appears in a URL. The ‘+’ character doesn’t appear to be escaped, since it does not involve a ‘%’. Also, RFC 2396 notes it as a reserved character that can be included in a URL when it’s in escaped form (%2b). But with allowDoubleEscaping set to its default value of false, we will block it even in escaped form. The reason for this is historical: Way back in the early days of HTTP, a ‘+’ character was considered shorthand for a space character. Some canonicalizers, when given a URL that contains a ‘+’ will convert it to a space. For this reason, we consider a ‘+’ to be non-canonical in a URL. I was not able to find any reference to a RFC that calls out this ‘+’ treatment, but there are many references on the web that talk about it as a historical behavior.

From my own experience I know that when IIS logs a request spaces are substituted with a plus sign. Having a plus sign in the name may cause confusion when parsing logs.

Solution

There are three ways to fix this and two ways to still use the plus sign.

1. `allowDoubleEscaping=true` - This will allow double escaping for your entire website/application. Depending on the content, this could be undesirable to say the least. The following command will set `allowDoubleEscaping=true`.

   ```
   appcmd.exe set config "Default Web Site" -
   section:system.webServer/security/requestFiltering /allowDoubleEscaping:True
   ```

2. `alwaysAllowedUrls` - Request Filtering offers a whitelist approach. By adding that URL path to alwaysAllowedUrls, the request will not be checked by any other Request Filtering settings and continue on in the IIS request pipeline. The concern here is that Request Filtering will not check the request for:

- Request Limits: maxContentLength, maxUrl, maxQueryString
- Verbs
- Query - query string parameters will not be checked
- Double Escaping
- High Bit Characters
- Request Filtering Rules
- Request Header Limits

The following command will add `/product/tags/for+families` to `alwaysAllowedUrls` on the Default Web Site.

```
    appcmd.exe set config "Default Web Site" -
section:system.webServer/security/requestFiltering /+"alwaysAllowedUrls.
[url='/product/tags/for+families']"
```

3. **Rename** - yes, just rename the file/folder/controller/etc. if possible. This is the easiest solution.

Share

Improve this answer

Follow

edited Oct 7, 2021 at 8:14

Community Bot
1 ●1

answered Dec 4, 2018 at 20:42

user2320464
389 ●1 ●6 ●12

1 To allowDoubleEscaping during development with IIS Express here's what I did: stackoverflow.com/q/56463044/381082 – DeveloperDan Feb 19, 2020 at 18:40

@user2320464 - excellent summary. CAVEAT TO READERS: Do NOT set "allowDoubleEscaping=true": that's just a bad idea. Alternatives include using Uri.EscapeUriString() or Uri.EscapeDataString() – paulsm4 Mar 20, 2021 at 5:39

4 @paulsm4, as the post states, `with allowDoubleEscaping set to its default value of false, we will block [+] even in escaped form.` I tested with `EscapeUriString` and `EscapeDataString` and an escaped `+` still does not go through. – Lukas Apr 19, 2021 at 15:18

6

I have made a work arround for this . so when you want to put the encripted string inside the url for(IIS) you have to clean it from dirty :{ ";", "/", "?", ":", "@", "&", "=", "+", "$", "," }; and when you want to decript it and use it again , you have to make it dirty again before decript it (to get the wanted result) .

Here is my code , i hope it helps somebody :

```
 public static string cleanUpEncription(string encriptedstring)
        {
            string[] dirtyCharacters = { ";", "/", "?", ":", "@", "&", "=",
"+", "$", "," };
            string[] cleanCharacters = {
"p2n3t4G5l6m","s1l2a3s4h","q1e2st3i4o5n" ,"T22p14nt2s", "a9t" ,
"a2n3nd","e1q2ua88l","p22l33u1ws","d0l1ar5","c0m8a1a"};

            foreach (string dirtyCharacter in dirtyCharacters)
            {
                encriptedstring=encriptedstring.Replace(dirtyCharacter,
cleanCharacters[Array.IndexOf(dirtyCharacters, dirtyCharacter)]);
            }
            return encriptedstring;
        }

        public static string MakeItDirtyAgain(string encriptedString)
        {
            string[] dirtyCharacters = { ";", "/", "?", ":", "@", "&", "=",
"+", "$", "," };
```

```
            string[] cleanCharacters = { "p2n3t4G5l6m", "s1l2a3s4h",
"q1e2st3i4o5n", "T22p14nt2s", "a9t", "a2n3nd", "e1q2ua88l", "p22l33u1ws",
"d0l1ar5", "c0m8a1a" };
            foreach (string symbol in cleanCharacters)
            {
                encriptedString = encriptedString.Replace(symbol,
dirtyCharacters[Array.IndexOf(cleanCharacters,symbol)]);
            }
            return encriptedString;
        }
```

Share Improve this answer Follow

answered Apr 11, 2019 at 16:28
Mark Dibeh
459 ●1 ●7 ●21

What is pntGlm? – StuperUser May 17, 2019 at 16:29

@StuperUser just random characters – Mark Dibeh May 21, 2019 at 5:41

1 The string should be `base64` encoded instead of using this substitution technique. For example, Basic Authentication uses `base64` to encode the submitted credentials as it may contain special/reserved characters. – user2320464 Jun 26, 2020 at 13:09

problem with base64 is / is a base64 character and can break the routes – Geoduck Oct 24, 2020 at 18:55

1 This works 97% of the time. If your initial string doesn't have "@" but has at9, the Make dirty will replace with @ and your decrypt will bomb. This goes for any of the other replacement strings. – user789221 Jul 14, 2022 at 13:37

3

Encode the encrypted string separated:

```
return HttpServerUtility.UrlTokenEncode(Encoding.UTF8.GetBytes("string"));
```

Decode the encrypted string separated:

```
string x = Encoding.UTF8.GetString(HttpServerUtility.UrlTokenDecode(id));
```

Share
Improve this answer
Follow

edited Nov 2, 2020 at 19:29
mustaccio
18.9k ●16 ●50 ●58

answered Nov 2, 2020 at 18:51
u
unvascorriendo
57 ●2

3

So I ran into this when I was calling an API from an MVC app. Instead of opening the security hole, I modified my path.

First off, I recommend NOT disabling this setting. It is more appropriate to modify the design of the application/resource (e.g. encode the path, pass the data in a header or in the body).

Although this is an older post, I thought I would share how you could resolve this error if you are receiving this from a call to an API by using HttpUtility.UrlPathEncode method in System.Web.

I use RestSharp for making calls out, so my example is using the RestRequest:

```
var tags = new[] { "for", "family" };
var apiRequest = new
RestRequest($"product/tags/{HttpUtility.UrlPathEncode(string.Join("+",
tags))}");
```

This produces a path equal to:

> /product/tags/for%2Bfamilies

On another note, do NOT build a dynamic query based on a user's inputs. You SHOULD always use a SqlParameter. Also, it is extremely important from a security perspective to return the values with the appropriate encoding to prevent injection attacks.

Share

Improve this answer

Follow

edited Sep 17, 2021 at 6:59
Nimantha
6,438 ●6 ●30 ●75

answered Aug 9, 2018 at 18:36
Rogala
2,773 ●27 ●27

+1 for recommending *NOT* disabling "allowDoubleEscaping", and for recommending *ALWAYS* using a SqlParameter when needed. However, I would recommend using Uri.EscapeUriString vs. a 3rd party package list RestSharp. – paulsm4 Mar 17, 2021 at 23:28

It does not work because IIS returns error 404.11 if there is %2B in the url. – palota Aug 22 at 10:44

1

I've just came across this issue with a Delete request. I didn't expect the below to work because it wasn't mentioned above, but I changed the request to instead pass the value as a querystring and it works fine, as long as you also encode the value (otherwise it treats the + as a space).

So from:

```
[HttpDelete("Example/{id}")]
public async Task<ActionResult<bool>> ExampleAsync(string id)
```

To:

```
[HttpDelete("Example")]
public async Task<ActionResult<bool>> ExampleAsync([BindRequired] string id)
```

If there is a reason you shouldn't do it this way, please let me know

Share Improve this answer Follow

answered May 17, 2023 at 15:26
Jon
69 ● 6

Sadly this doesn't work for me with a `HttpGet` (.NET6) – ArieKanarie Oct 19, 2023 at 10:03

It also works with `HttpGet` (.NET 8) – palota Aug 22 at 11:20

0

This is just a joke from .net framework: by trying to url encode inside client using:

```
Uri uri = CreateUri(REPO_PREFIX +
$"Name/{Id}/{WebUtility.UrlEncode(name)}/{WebUtility.UrlEncode(data)}");
client.PutAsync(uri, null);
```

cause indeed this issue with url including a '+'

however removing explicit url encode:

```
 Uri uri = CreateUri(REPO_PREFIX + $"Name/{Id}/{name}/{data}");
```

asp net core server receive query url encoded:

```
PUT localhost:10000/api/v1/xx/Name/8331/Backxxzzzzz%20afternoon/x HTTP/1.1
```

that i dump in logs via:

```
        request.AppendLine(Request.Method + " " + Request.Host + Request.Path +
" " + Request.Protocol);
```

Share Improve this answer Follow

answered Aug 31, 2023 at 13:50
Thierry Brémard
901 ● 1 ● 7 ● 18