

Re: Tricky Request Filtering Issue

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.aspnet/2008-09/msg00073.html>

- *From:* "Joseph Geretz" <jgeretz@xxxxxxxxxx>
 - *Date:* Wed, 3 Sep 2008 01:09:47 -0400
-

Some additional diagnostic data:

The following method is functionally the same as the one I presented in my last post. What I've done though is break down a couple of 'compound' statements into their components. (Again, I've numbered these for clarity.)

```
private string zGetRequestBody(HttpContext httpCtx)
{
1. Stream S = httpCtx.Request.InputStream;
2. int Len = (int)S.Length;
3. byte[] buffer = new byte[Len];
4. S.Read(buffer, 0, Len);
5. return Encoding.UTF8.GetString(buffer, 0, Len);
}
```

What happens now is that statement number 1 is now firing the filter's Read method (twice), even though this statement is simply the assignment of an object reference to a variable! This really seems to be at the heart of my difficulty – the first reference to the Request.InputStream object after the filter has been wired up results in a double firing of the filter's Read method. This is bizarre! Any idea what might account for this?

Thanks for your help!

– Joseph Geretz –

"Joseph Geretz" <jgeretz@xxxxxxxxxx> wrote in message
<news:OVf917XDJHA.484@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>

Hi Bruce,

I'm not 100% sure myself what the problem is. I'm seeing a number of anomalies. Perhaps you can help me 'peel this onion' to get to the bottom of the problem.

In addition to the Request Filter, my application implements an HttpModule. The module's purpose in life is basically to wire up the Request and Response filters at the start of each transaction. The module

Re: Tricky Request Filtering Issue

will also log the request and do certain other application wide processing. Here's the code in the HttpModule which attempts to get the body of the Request:

(I've numbered these lines just for the clarity of the discussion.)

```
private string zGetRequestBody(HttpContext httpCtx)
{
1. byte[] buffer = new byte[httpCtx.Request.InputStream.Length];
2. httpCtx.Request.InputStream.Read(buffer, 0,
(int)httpCtx.Request.InputStream.Length);
3. return Encoding.UTF8.GetString(buffer, 0, buffer.Length);
}
```

The first anomaly I am noticing is that line 1 triggers the *Read* method of my Request filter (huh?):

```
public override int Read(byte[] buffer, int offset, int count)
{
int ContentLength;
ContentLength = m_sink.Read(buffer, offset, count);
return ContentLength;
}
```

(Actually, I observe this method fire twice, for this line of code.)

Now why in the world would this be happening? I've set a breakpoint in the Length method, yet this breakpoint is never hit!!

```
public override long Length
{
get { return m_sink.Length; }
}
```

So this is my first question on the way to figuring this out: Why does `httpCtx.Request.InputStream.Length` trigger the *Read* method of my filter? Shouldn't it be triggering the Length property?? At this point, I've completely removed the entire issue of encryption/decryption from the exercise. The code I've got in place now works without encryption/decryption. but I simply don't understand the mechanics and when I add encryption/decryption which dynamically changes the length, things go from bad to worse. I suspect that the matter will ultimately resolve itself as you suggest, however the first thing we need to do is figure out why the interrogation of the Length property fires off the filter's Read method.

Thanks for your help with this!

– Joseph Geretz –

"bruce barker" <nospam@xxxxxxxxxxx> wrote in message

Re: Tricky Request Filtering Issue

news:Ok1qnjXDJHA.5196@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

i'm not sure what your problem is. the filter is a stream, you read the stream in, and write it out modified anyway your want. nothing says you have to write the same number of bytes you read.

— bruce (sqlwork.com)

Joseph Geretz wrote:

I'm using the Request Filter documentation which can be found here:

<http://msdn.microsoft.com/en-us/library/system.web.httprequest.filter.aspx>

In this example, two filters are installed, one filter uppercases every alphabetic character and then the second filter replaces every 'E' with "#". OK, very nice.

What's trivial about these examples is that the returned string is exactly the same length as the string which is actually read from the Request stream. I have a slightly more complicated situation where I'm using a Filter to decrypt the data which is pulled from the Request stream. (I'm implementing cryptography in the Filter so that application developers don't need to worry about dealing with this issue; application developers simply retrieve Request data and the filter ensures that it is always delivered in PlainText, regardless of whether it was POSTed in cipher or plain text.)

So how do I deal with this issue, where the Request stream contains let's say 2000 bytes, yet the decrypted plaintext is now only 1000 bytes? If I return a count of 1000, then the Read event fires again, seemingly because it appears as though there remains 1000 bytes to remove from the stream. If I return the original count, then the buffer returned is much too large, larger than the actual data which causes

Re: Tricky Request Filtering Issue

problems when attempting to decrypt the ciphertext. If you've ever developed such a scenario, I appreciate any advice which you can provide.

Thanks!

– Joseph Geretz –