ModSecurity for IIS:

Part 2 of 2 - Web Application Security Flaw Discovery and Prevention

By Russ McRee – ISSA Senior Member, Puget Sound (Seattle), USA Chapter

Prerequisites/dependencies

Windows OS with IIS (Win2k8 used for this article)

SQL Server Express 2004 SP4 and Management Studio Express for vulnerable web app

.NET Framework 4.0 for ModSecurity IIS

December's issue continues where we left of in November with Part 2 in our series on web application security flaw discovery and prevention. In November we discussed Arachni, the high-performance, modular, open source web application security scanning framework. This month we'll follow the logical work flow from Arachni's distributed, high-performance scan results to how to use the findings as part of mitigation practices. One of Arachni's related features is WAF Realtime Virtual Patching.[1]

Trustwave Spider Lab's Ryan Barnett has discussed the concept of dynamic application scanning testing (DAST) data that can be imported into a web application firewall (WAF) for targeted remediation. This discussion included integrating export data from Arachni into ModSecurity,2 the cross-platform, open source WAF for which he is the OWASP ModSecurity Core Rule Set (CRS) project leader. I reached out to Ryan for his feedback with particular attention to ModSecurity for IIS, Microsoft's web server.

He indicated that WAF technology has gained traction as a critical component of protecting live web applications for a number of key reasons, including:

1. Gaining insight into HTTP transactional data that is not provided by default web server logging
2. Utilizing virtual patching to quickly remediate identified vulnerabilities
3. Addressing PCI DSS Requirement 6.6

The ModSecurity project is just now a decade old (first released in November 2002), has matured significantly over the years, and is the most widely deployed WAF in existence, protecting millions of websites. "Until recently, ModSecurity was only available as an Apache web server module. That changed, however, this past summer when Trustwave collaborated with the Microsoft Security Response Center (MSRC) to bring the ModSecurity WAF to both the Internet Information Services (IIS) and nginx web server platforms. With support for these platforms, ModSecurity now runs on approximately 85% of internet web servers," Ryan explained.

Among the features that make ModSecurity so popular, there are a few key capabilities that make it extremely useful:

- It has an extensive audit engine which allows the user to capture the full inbound and outbound HTTP data. This is not only useful when reviewing attack data but is also extremely valuable for web server administrators who need to trouble-shoot errors.
- It includes a powerful, event-driven rules language which allows the user to create very specific and accurate filters to detect web-based attacks and vulnerabilities.
- It includes an advanced Lua API which provides the user with a full-blown scripting language to define complex logic for attack and vulnerability mitigation.
- It also includes the capability to manipulate live transactional data. This can be used for a variety of security purposes including setting hacker traps, implementing anti-CSRF tokens, or cryptographic HASH tokens to prevent data manipulation.

In short, Ryan states that ModSecurity is extremely powerful and provides a very flexible web application defensive framework that allows organizations to protect their web applications and quickly respond to new threats.

I also sought details from Greg Wroblewski, Microsoft's lead developer for ModSecurity IIS:

> "As ModSecurity was originally developed as an Apache web server module, it was technically challenging to bring together two very different architectures. The team managed to accomplish that by creating a thin layer abstracting ModSecurity for Apache from the actual server API. During the development process it turned out that the new layer is flexible enough to create another ModSecurity port for the nginx web server. In the end, the security community received a new cross-platform firewall, available for the three most widely used web servers."

1 http://blog.spiderlabs.com/2012/06/dynamic-dastwaf-integration-realtime-virtual-patching.html.

2 http://www.modsecurity.org.

The current ModSecurity development process (still open, recently migrated to GitHub3) preserves compatibility of features between three ported versions. For the IIS version, only features that rely on specific web server behavior show functional differences from the Apache version, while the nginx version currently lacks some of the core features (like response scanning and content injection) due to limited extensibility of the server. Most ModSecurity configuration files can be used without any modifications between Apache and IIS servers. The upcoming release of the RTM version for IIS will include a sample of ModSecurity OWASP Core Rule Set4 in the installer.

Installing ModSecurity for IIS

In order to test the full functionality of ModSecurity for IIS, I needed to create an intentionally vulnerable web application and did so following guidelines provided by Metasploit Unleashed.5 The author wrote these guidelines for Windows XP SP2; I chose Windows Server 2008 just to be contrarian. I first established a Win2k8 virtual machine, enabled the IIS role, downloaded and installed SQL Server 2005 Express SP4,6 .NET Framework 4.0, as well as SQL Server 2005 Management Studio Express,7 then downloaded the ModSecurity IIS 2.7.1 installer.8 We'll configure ModSecurity IIS after building our vulnerable application.

When configuring SQL Server 2005 Express, ensure you enable SQL Server Authentication, and set the password to something you'll use in the connection string established in Web.config. I used `p@ssw0rd1` to meet required complexity. J Note: It's "easier" to build a vulnerable application using SQL Server 2005 Express rather than 2008 or later; for time's sake and reduced troubleshooting just work with 2005. We're in test mode here, not production. That said, remember, you're building this application to be vulnerable by design. Conduct this activity only in a virtual environment and do not expose it to the Internet. Follow the Metasploit guidelines carefully but remember to establish a proper connection string in the Web.config (line 4) and build it from this sample9 I'm hosting for you rather than the one included with the guidelines. As an example, I needed to establish my actual server name rather than localhost. I defined my database name as crapapp instead of WebApp per the guidelines, and used p@ssw0rd1 instead of password1 as described:

```
<add name="test" connectionString="server=WIN2K8-
VM\SQLEXPRESS;database=crapapp;uid=sa;password=p@
ssw0rd1;" providerName="System.Data.SqlClient"/>
```

I also utilized configurations recommended for the pending ModSecurity IIS install so go with my version.

3 https://github.com/SpiderLabs/ModSecurity/.

4 https://github.com/SpiderLabs/owasp-modsecurity-crs.

5 http://www.offensive-security.com/metasploit-unleashed/Creating_A_Vulnerable_Webapp.

6 http://www.microsoft.com/en-us/download/details.aspx?id=184.

7 http://www.microsoft.com/downloadS/details.aspx?familyid=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=en%7CSQL.

8 https://github.com/SpiderLabs/ModSecurity/downloads.

9 http://holisticinfosec.org/toolsmith/files/modsecurityiis/Web.config.

Once you're finished with your vulnerable application build you should browse to http://localhost and first pass credentials that you know will fail to ensure database connectivity. Then test one of the credential pairs established in the users table, admin/s3cr3t as an example. If all has gone according to plan you should be treated to a successful login message as seen in figure 1.

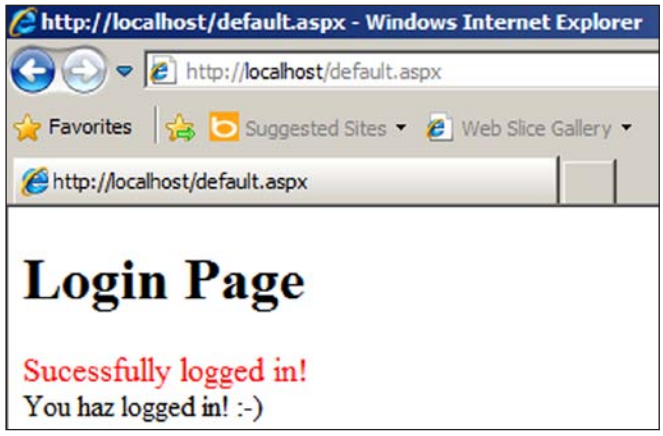

Figure 1 – A successful login to CrapApp

ModSecurity IIS installation details are available via TechNet,10 but I'll walk you through a bit of it to help overcome some of the tuning issues I ran into. Make sure you have the full version of .NET 4.0 installed and patch it in full before you execute the ModSecurity IIS installer you downloaded earlier.

Download the ModSecurity OWASP Core Rule Set (CRS), and as a starting point copy the files from the `base_rules` to the `crs` directory you create in `C:\inetpub\wwwroot`. Also put the test.conf file11 I'm also hosting for you in `C:\inetpub\wwwroot`. This will call the just-mentioned ModSecurity OWASP Core Rule Set (CRS) that Ryan maintains and also allow you to drop any custom rules you may wish to create right in `test.conf`.

There are a few elements to be comfortable with here. Watch the Windows Application logs via Event Viewer to both debug any errors you receive as well as ModSecurity alerts once properly configured. I'm hopeful that the debugging time I spent will help save you a few hours, but watch those logs regardless. Also make regular use of the Internet Information Services (IIS) Manger to refresh the `DefaultAppPool` under `Application Pools` as well as restart the IIS instance after you make config changes. Finally, this experimental installation intended to help get you started is running in active mode versus passive. It will both detect and block what the CRS notes as malicious. As such, you'll want to initially comment out all the HTTP Policy rules in order to play with the CrapApp we built above. To do so, open `modsecurity_crs_30_http_policy.conf` in the `crs` directory and comment out all lines that start with SecRule, or simply pull

10 http://blogs.technet.com/b/srd/archive/2012/07/26/announcing-the-availability-of-modsecurity-extension-for-iis.aspx.

11 http://holisticinfosec.org/toolsmith/files/modsecurityiis/test.conf.

the whole policy file from the `crs` directory. Again, we're in experiment mode here. Don't deploy ModSecurity in production with the `SecDefaultAction` directive set to "block" without a great deal of testing in passive mode first or you'll likely blackhole known good traffic.

Using ModSecurity and virtual patching to protect applications

Now that we're fully configured, I'll show you the results of three basic detections, then close with a bit of virtual patching for your automated web application protection pleasure. Figure 2 is a mashup of a login in attempt via our CrapApp with a path traversal attack and the resulting detection and block as noted in the Windows Application log.

Similarly, a simple SQL injection such as `'1=1--` against the same form field results in the following Application log entry snippet:

```
[msg "SQL Injection Attack: Common Injection
Testing Detected"] [data "Matched Data: '
found within ARGS:txtLogin: '1=1--"] [severity
"CRITICAL"] [ver "OWASP_CRS/2.2.6"] [maturity "9"]
[accuracy "8"] [tag "OWASP_CRS/WEB_ATTACK/SQL_
INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_
TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag
"PCI/6.5.2"]
```

Note the various tags including a match to the appropriate OWASP Top 10 entry as a well as the relevant section of the PCI DSS.

Ditto if we pop in a script tag via the `txtLogin` parameter:

```
[data "Matched Data: <script> found within
ARGS:txtLogin: \x22><script>alert(document.
cookie)</script>"] [ver "OWASP_CRS/2.2.6"]
[maturity "8"] [accuracy "8"] [tag "OWASP_CRS/WEB_
ATTACK/XSS"] [tag "WASCTC/WASC-8"] [tag "WASCTC/
WASC-22"] [tag "OWASP_TOP_10/A2"] [tag "OWASP_
AppSensor/IE1"] [tag "PCI/6.5.1"]
```

Finally, we're ready to connect our Arachni activities in Part 1 of this campaign to our efforts with ModSecurity IIS. There are a couple of ways to look at virtual patching as amply described by Ryan. His latest focus has been more on dynamic application scanning testing as actually triggered via ModSecurity. There is now Lua scripting that integrates ModSecurity and Arachni over RPC where a specific signature hit from ModSecurity will contact the Arachni service and kick off a targeted scan. At last check this code was still experimental and likely to be challenging with the IIS version of ModSecurity. That said we can direct our focus in the opposite direction to utilize Ryan's automated virtual patching script, `arachni2modsec.pl`, where we gather Arachi scan results and automatically convert the XML export into rules for ModSecurity. These custom rules will then protect the vulnerabilities discovered by Arachni while you haggle with the developers over how long it's going to take them to actually fix the code.

To test this functionality I scanned the CrapApp from Arachni instance on the Ubuntu VM I built for last month's article. I also set the `SecDefaultAction` directive set to "pass" in my `test.conf` file to ensure the scanner is not blocked while it discovers vulnerabilities. Currently the `arachni2modsec.pl` script writes rules specifically for SQL Injection, Cross-site Scripting, Remote File Inclusion, Local File Inclusion, and HTTP Response Splitting. The process is simple; assuming the results file is `results.xml`, `arachni2modsec.pl –f results.xml will create modsecurity_crs_48_virtual_patches.conf`. On my ModSecurity IIS VM I'd then copy `modsecurity_crs_48_virtual_patches.conf` into the `C:\inetpub\wwwroot\crs` directory and refresh the `DefaultAppPool`. Figure 3 gives you an idea of the resulting rule.

Note how the rule closely resembles the alert spawned when I passed the simple SQL injection attack to CrapApp earlier in the article. Great stuff, right?

In Conclusion

What a great way to wrap up 2012 with the conclusion of this two-part series on Web Application Security Flaw Discovery

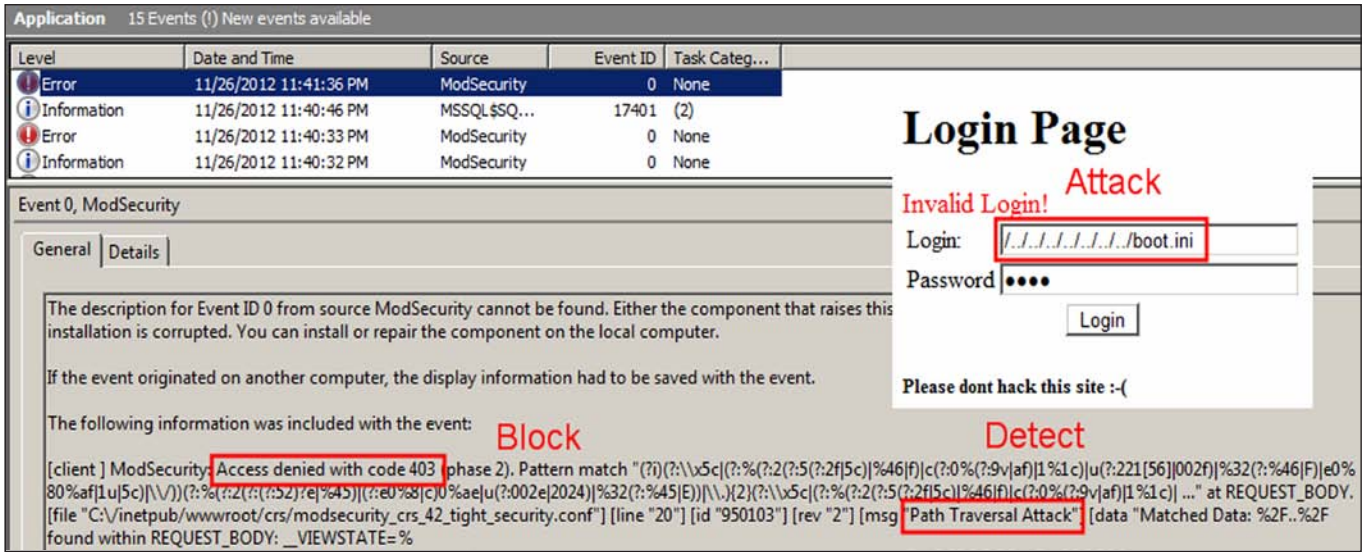

Figure 2 – Path traversal attack against CrapApp denied

Figure 3 – arachni2modsec script creates rule for ModSecurity IIS

```
#
# Arachni Virtual Patch Details:
# ID: 11
# Type: SQL Injection
# Vulnerable URL: 192.168.245.129
# Vulnerable Parameter: cat
#
SecRule REQUEST_FILENAME "192.168.245.129" "chain,phase:2,t:none,block,msg:'Virtual Patch for SQL
 Injection',id:'11',tag:'WEB_ATTACK/SQL_INJECTION',tag:'WASCTC/WASC-19',tag:'OWASP_TOP_10/A1',tag
:'OWASP_AppSensor/CIE1',tag:'PCI/6.5.2',logdata:'%{matched_var_name}',severity:'2'"
        SecRule &TX:'/SQL_INJECTION.*ARGS:txtLogin/' "@gt 0" "setvar:'tx.msg=%{rule.msg}',setvar:
tx.sql_injection_score=+%{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%{tx.critical_anoma
ly_score}"
```

and Prevention. I'm thrilled with the performance of ModSecurity for IIS and really applaud Ryan and Greg for their efforts. There are a number of instances where I intend to utilize the ModSecurity port for IIS and will share feedback as I gather data. Please let me know how it's working for you as well should you choose to experiment and/or deploy.

Good luck and Merry Christmas.

Stay tuned to vote for the 2012 Toolsmith Tool of the year starting December 15.

Acknowledgements

—Ryan Barnett, Trustwave Spider Labs, Security Researcher Lead

—Greg Wroblewski, Microsoft, Senior Security Developer

Ping me via email if you have questions (russ at holisticinfosec dot org).

Cheers…until next month.

About the Author

Russ McRee manages the Security Analytics team (security incident management, penetration testing, monitoring) for Microsoft's Online Services Security & Compliance organization. In addition to toolsmith, he's written for numerous other publications, speaks regularly at events such as DEFCON, Black Hat, and RSA, and is a SANS Internet Storm Center handler. As an advocate for a holistic approach to the practice of information assurance Russ maintains holisticinfosec.org. He serves in the Washington State Guard as the Cybersecurity Advisor to the Washington Military Department. Reach him at russ at holisticinfosec.dot org or @holisticinfosec.