Arachni: Web Application Security Scanner

Part 1 of 2: Web Application Security Flaw Discovery and Prevention

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

Prerequisites/dependencies

Ruby 1.9.2 or higher in any *nix environment

This month's issue kicks off a two-part series on web application security flaw discovery and prevention, beginning with Arachni. As this month's topic is another case of mailing lists facilitating great *toolsmith* topics, I'll begin this month by recommending a few you should join if you haven't already. The Web Application Security Consortium mailing list[1] is a must, as are the SecurityFocus lists[2] – I favor their Penetration Testing and Web Application Security lists but they have many others as well. As you can imagine, these two make sense for me given focuses on web application security and penetration testing, and it was via SecurityFocus that I received news of the latest release of Arachni. Arachni is a high-performance, modular, open source web application security scanning framework written in Ruby. It was refreshing to discover a web app scanner I had not yet tested. I spend a lot of time with the likes of Burp, ZAP, and Watobo but strongly advocate expanding the arsenal.

Arachni's developer/creator is Tasos "Zapotek" Laskos, who kindly provided details on this rapidly maturing tool and project.

Via email Tasos indicated that to date Arachni's role has been that of an experiment/learning-exercise hybrid, mainly focused on doing things a little bit differently. He's glad to say that the fundamental project goals have been achieved; Arachni is fast, relatively simple, quite accurate, open source, and quite flexible in the ways which it can be deployed. In addition, as of late, stability and testing have been given top priority in order to ensure that the framework won't exhibit performance degradation as the code-base expands.

With a strong foundation laid and a clear road map, future plans for Arachni include pushing the envelope where version 0.4.2 includes improved distributed, high-performance scan features such as the new distributed crawler[3] (under current development), and a new, cleaner, more stable and attractive web user interface, as well as general code clean-up.

Version 0.5 is where a lot of interesting work will take place as the Arachni team will be attempting to break some new ground with native DOM and JavaScript support, with the intent of allowing a depth/level of analysis beyond what's generally possible today, from either open source or commercial systems. According to Tasos, most, if not all, current scanners rely on external browser engines to perform their duties, bringing with them a few penalties (performance hits, loss of control, limited inspection capabilities, design compromises, etc.), which Arachni will be able to avoid. This kind of functionality, especially from an open and flexible system, will be greatly beneficial to web application testing in general, and not just in a security-related context.

Arachni success stories include incredibly cool features such as WAF Realtime Virtual Patching.[4] At OWASP AppSec DC 2012,[5] Trustwave Spider Lab's Ryan Barnett discussed the concept of dynamic application scanning testing (DAST), exporting data that is then imported into a web application firewall (WAF) for targeted remediation. In addition to stating that the Arachni scanner is an "absolutely awesome web application scanner framework," Ryan describes how to integrate export data from Arachni with ModSecurity, the WAF for which he is OWASP ModSecurity Core Rule Set (CRS) project leader. Take note here as next month in *toolsmith* we're going to discuss ModSecurity for IIS as part two of this series and will follow Ryan's principles for DAST to WAF.

Other Arachni successes include highly-customized scripted audits and easy incorporation into testing platforms (by virtue of its distributed features). Tasos has received a lot of positive feedback and has been pleasantly surprised there has not been one unsatisfied user, even in the Arachni's early, immature phases. Many Arachni users end up doing so out of frustration with the currently available tools and are quite happy with the results after giving Arachni a try, given that Arachni gives users a decent alternative while simplifying web application security assessment tasks.

Arachni benefits from excellent documentation and support via its wiki;[6] be sure to give a good once over before beginning installation and use.

1 http://webappsec.org/lists/.

2 http://www.securityfocus.com/.

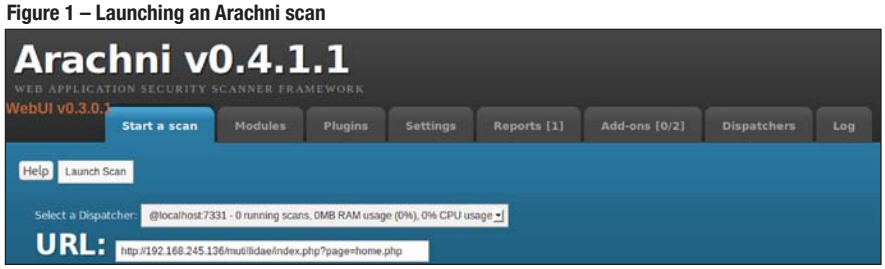
3 http://arachni-scanner.com/blog/preliminary-look-at-the-new-high-performance-distributed-crawler.

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

5 https://www.owasp.org/index.php/OWASP_AppSec_DC_2012/Dynamic_DASTWAF_Integration.

6 http://arachni-scanner.com/wiki.

Figure 1 – Launching an Arachni scan



Installing Arachni

On an Ubuntu 12.10 instance, I first made sure I had all dependencies met via `sudo apt-get install build-essential libxml2-dev libxslt1-dev libcurl4-openssl-dev libsqlite3-dev libyaml-dev zlib1g-dev ruby1.9.1-dev ruby1.9.1`.

For developer's sake, this includes Gem support so thereafter one need only issue `sudo gem install arachni` to install Arachni. However, the preferred method is use of the appropriate system packages from the latest downloads page.[7]

While Arachni features robust CLI[8] use, for presentation's sake we'll describe Arachni use with the Web UI.[9] Start it via `arachni_web_autostart` which will initiate a Dispatcher and the UI server. The last step is to point your browser to http://localhost:4567, accept the default settings, and begin use.

Arachni in use

Of interest, as you begin Arachni use, is the dispatcher which spawns RPC instances and allows you to attach to, pause, resume, and shutdown Arachni instances. This is extremely important for users who wish to configure Arachni instances in a high performance grid[10] (think a web application security scanning cluster with a master and slave configuration). Per the wiki, "this allows scan-time to be severely decreased, by as much as *n* times less under ideal circumstances, where *n* equals the number of running instances."

You can configure Arachni's web UI to run under SSL and provide HTTP Basic authentication if you wish to lock use down. Refer to the wiki entry for the web user interface for more details.

Before beginning a simple scan (one dispatcher), let's quickly review Arachni's modules and plugins. Each has a tab in Arachni's primary UI view. The 45 modules are divided into Audit (22) and Recon (23) options where the audit modules actively test the web application via inputs such as parameters, forms, cookies, and headers; while the recon modules passively test the web application, focusing on server configuration, responses, and specific directories and files. I particularly like the additional SSN and credit card number disclosure modules as they are helpful for OSINT, as well as the Backdoor module, which looks to determine if the web application you're assessing is already owned. Of note, from the Audit options is the Trainer module that probes all inputs of a given page in order to uncover new input vectors and trains Arachni by analyzing the server responses. Arachni modules are all enabled by default. Arachni plugins offer preconfigured auto-logins (great when spidering), proxy settings, and notification options along with some pending plugins supported in the CLI version but not yet ready for the Web UI as of v.0.4.1.1

To start a scan, navigate to the *Start a Scan* tab and confirm that a dispatcher is running. You should see the likes of @localhost:7331 (host and port) along with the number of running scans, as well as RAM and CPU usage. Then paste a URL into the URL form, and select *Launch Scan* as seen in figure 1.

While the scan is running, you can monitor the dispatcher status via the *Dispatchers* tab as seen in figure 2.

From the Dispatchers view you can choose to *Attach* to the running instance (there will be multiples if you've configured a high performance grid) which will give a real-time view to the scan statistics, percentage of completion for the running instance, scanner output, and results for findings discovered as seen in figure 3. Dispatchers provide Instances; Instances perform the scans.

Once the scan is complete, as you might imagine, the completed results report will be available to you in the *Reports* tab. As an example I chose the HTML output, but realize that you can also select JSON, text, YAML, and XML as well as

7 http://arachni-scanner.com/latest.
8 http://arachni-scanner.com/wiki/Command-line-user-interface.
9 http://arachni-scanner.com/wiki/Web-user-interface.
10 http://arachni-scanner.com/wiki/RPC-server - wiki-grid.

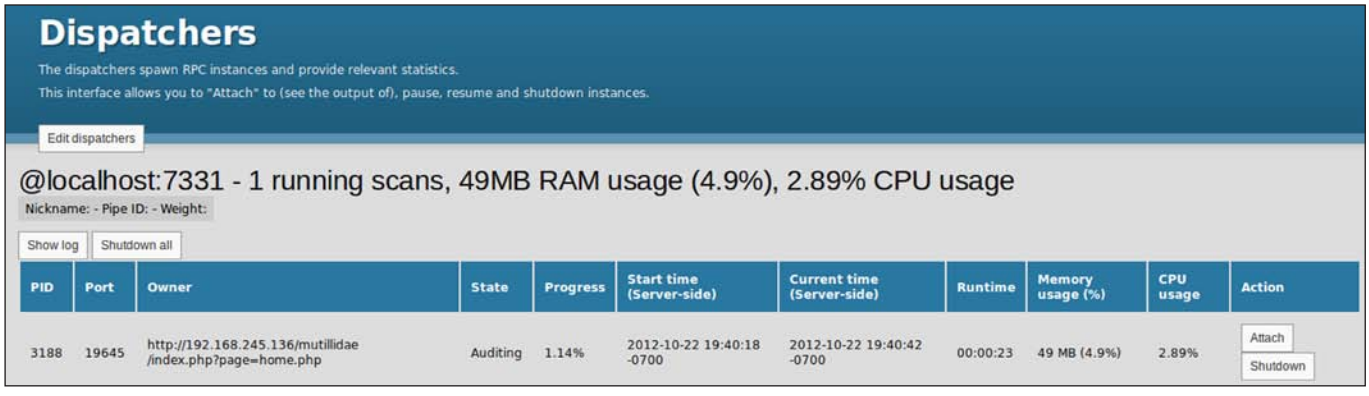
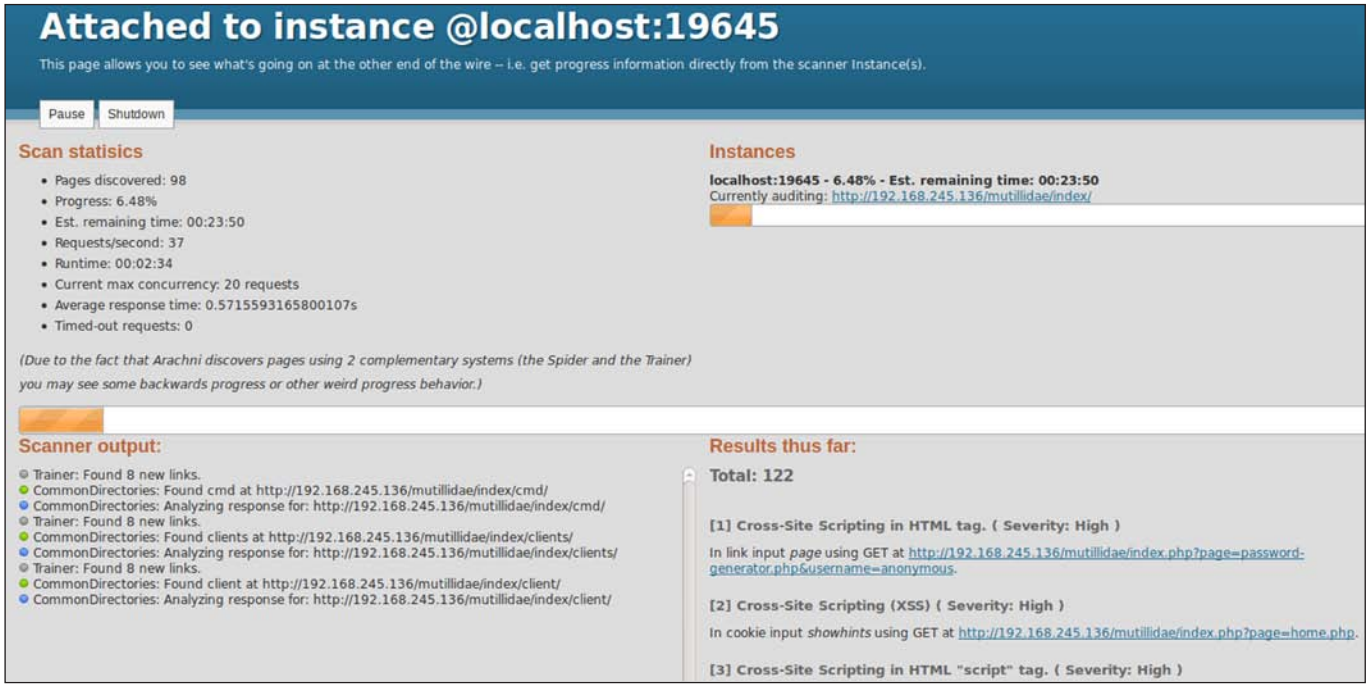


Figure 2 – Arachni Dispatcher status

Figure 3 – Arachni scan status



binary output such as Metareport, Marshal report, and even Arachni Framework Reporting. Figure 4 represents the HTML-based results of a scan against NOWASP Mutillidae.

Even the reports are feature-rich with a *Summary* tab with graphs and issues, remedial guidance, plugin results, along with a sitemap and configuration settings.

The results are accurate too; in my preliminary testing I found very few false positives. When Arachni isn't definitive about results, it even goes so far as to label the result "untrusted (and may in fact be false positives) because at the time they were identified the server was exhibiting some kind of anomalous behavior or there was 3rd-party interference (like

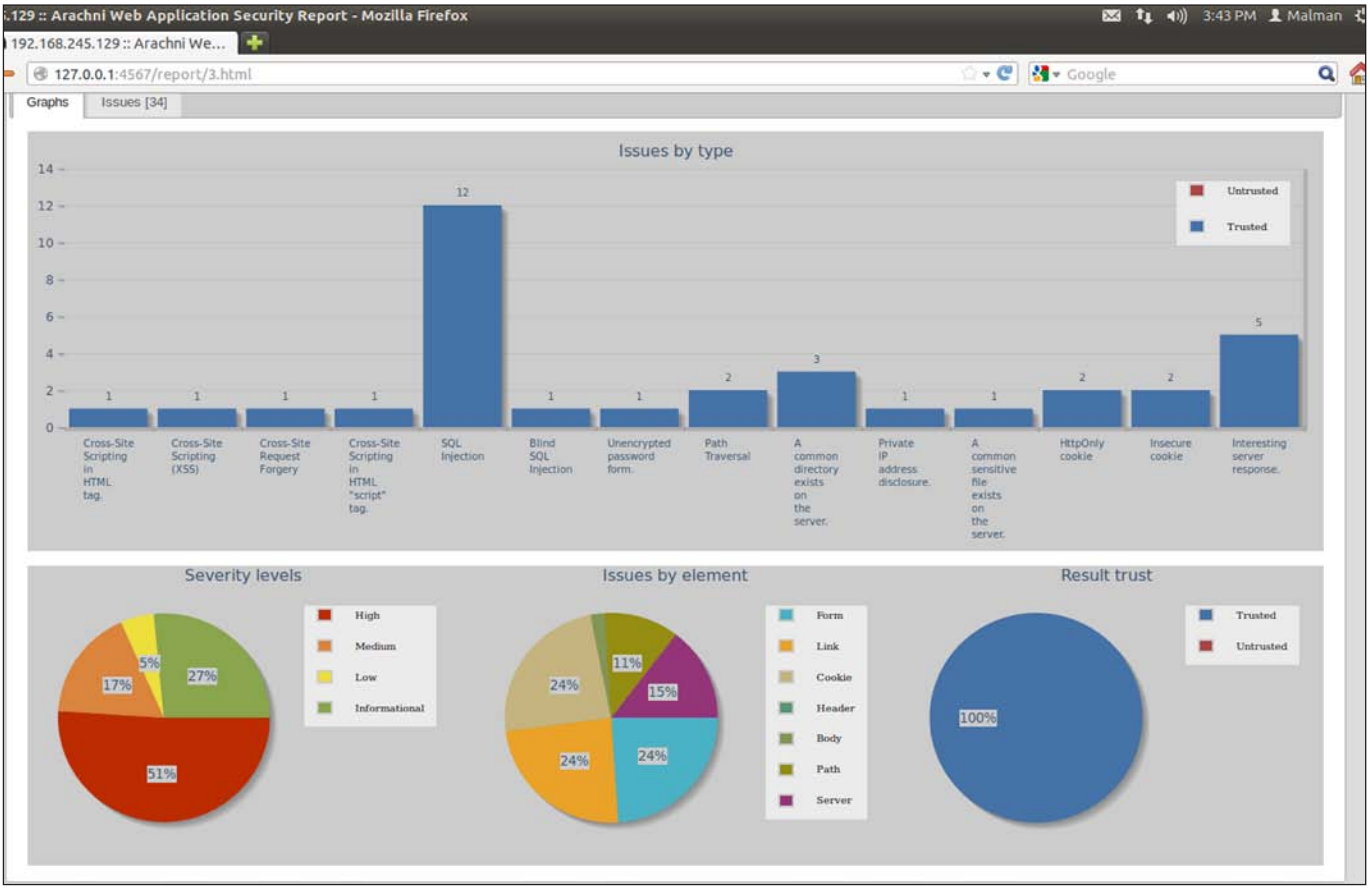


Figure 4 – HTML Arachni results

network latency for example)." Nice, I love truth and transparency in my test results.

I am really excited to see Arachni work at scale. I intend to test it very broadly on large applications using a high performance grid. This is definitely one project I'll keep squarely on my radar screen as it matures through its 0.4.2 and 0.5 releases.

In conclusion

Join us again next month as we resume this discussion when we take Arachni results and leverage them for Realtime Virtual Patching with ModSecurity for IIS. By then I will have tested Arachni's clustering capabilities as well, so we should have some real benefit to look forward to next month. Please feel free to seek support[11] via the support portal, file a bug report via the issue tracker,[12] or to reach out to Tasos via Twitter or email as he looks forward to feedback and feature requests.

11 http://support.arachni-scanner.com/.

12 https://github.com/Arachni/arachni/issues.

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

Cheers…until next month.

Acknowledgements

—Tasos "Zapotek" Laskos, Arachni project lead

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.