Web Development – one day we'll get it right

Web App Offense

A few tools and tricks can find and correct web app vulnerabilities. *By Kurt Seifried*

So, I just checked – from January 1, 2010, to October 9, 2011, 8,917 Common Vulnerabilities and Exposures (CVEs) were issued. Of these, 873 were related to cross-site scripting (XSS), which is just a hair under 10 percent. Of the remaining web-related vulnerabilities, cross-site request forgery (CSRF) [1], file source disclosure (whereby PHP contents are shown as text rather than a rendered web page), and so on were well represented. But over time, things are getting better, right? Not really; out of 51,353 total CVEs 6,580 are related to XSS, and if you plot the bugs over time, a generally upward-trending curve appears.

So, how does one go about fixing this situation? The good news is that some excellent resources are freely available. The bad news is that people don't seem to be using them.

Design and Architecture

The best place to start with security is at the beginning. The general design and architecture of web applications used to be quite similar to other applications, but this has changed. Now a web application commonly gets data not only from the user but from other services and sites, not all of which can be trusted. Even if you don't have the time to build all the security into the first release, at least put in stub classes and code that can later be filled out. For example, authentication code that initially uses just a shared secret but implements the idea of separate users will make it easier later to add the real authentication system. Unfortunately, most people are dealing with code that has already been written, and it's time to start fixing it and retrofitting security to it. For a good overview on all this, I suggest the "Guide to Secure Web Services" [2], which is published by NIST.

Adding Security

Even with good design, you might choose to retrofit security onto an existing system. For one thing, new techniques and technologies are coming out all the time, and things that were once expensive might now be cheap.

A great example of this is two-factor authentication using a hardware token. Although they haven't gotten cheaper, software tokens that can run on a smartphone are widely available now (an RSA token will still cost you roughly US$ 40-50). Smartphones also have become quite common; about 20% of all phones are smartphones, and I suspect among technology users, the percentage is higher. Even if you don't have a smartphone, you can still do two-factor authentication using SMS (which has sort of gotten cheaper, sigh) by sending a one-time code to the phone when the user is logging in. The real beauty of two-factor systems is that you no longer have to worry as much about your password being exposed to an attacker. My advice here is to watch what companies like Google are offering to users and consider imitating them.

Input and Output Validation

I/O validation is by far the biggest problem in most web applications and leads to problems like XSS, CSRF, and SQL injection. To further

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

complicate matters, things like JSON have become the de facto data transport method, mostly because XML is so over-engineered and parsing can consume a lot of resources.

Because JSON can contain pretty much arbitrary data, it is extremely important to ensure the text strings or arrays you think you are receiving from the remote end are, in fact, what they claim to be. A perfect example of such misplaced trust is the way Ruby handles the `X-Forwarded-For` header. This header contains a list of the system(s) that have forwarded the web request on behalf of a client, and it should only contain one or more IP addresses separated by commas. (At the time of this writing, the bug wasn't fixed yet.)

Unfortunately, Ruby fails to verify that this string of text is in fact a list of IP addresses, allowing pretty much any content (a string of text containing JavaScript, for example) to be included. So, if you ever report back to the user what their IP address is, unless you specifically check to make sure it is an IP address, your Ruby application has a XSS vulnerability. The good news is that you can't easily set the `X-Forwarded-For` on the client, so an attack would probably need an additional vulnerability (most likely in the web client) to actually pull it off.

OWASP

And this is where the Open Web Application Security Project (OWASP) [3] comes in. By the time this article comes out in print, the OWASP Foundation should be celebrating its 10th birthday. The thing that always amazes me is the sheer breadth and depth of their work. With 123 projects covering everything from web protocol fuzzing to educational tools such as WebGoat (a vulnerable web server that you protect using modsecurity [4], learning how to block some very sophisticated attacks in the process), OWASP definitely has something for everyone.

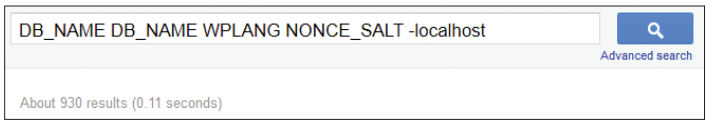

Figure 1: Google search results for wp-config.php files.

OWASP also publishes a large number of books [5], many of which are available for free (the rest are pretty reasonably priced). My favorite is *OWASP Code Review*, which goes over language-specific features and problems to look for when doing a source code audit. It also covers some of the trade-offs that can be made (e.g., if you can't sanitize incoming data, you can often encode it so it is less dangerous). OWASP also holds an annual conference with slides and videos posted online [6].

Web Security Tools and Documentation

A number of good and great security tools are available that you can use to scan web applications and servers for problems. One of easiest tools to use is Nikto [7], which includes several thousand checks and will find a lot of things without a lot of time spent configuring it. It also includes references to documentation explaining why a particular issue could be a security risk, which is extremely valuable if you need help convincing someone to fix it.

You can also write custom checks and modify existing ones easily, and the documentation is great. For more in-depth checks, a number of proxy products can manipulate requests. This makes interactive testing much easier, because you can browse through the site and then start poking at specific components (like shopping carts and login forms). If you combine Nikto with the OWASP testing guide, you're almost guaranteed to find security issues.

Thinking Outside the Box

Two more tools let me poke websites and applications. The first one I like to use is a LinkChecker [8]. Back when the web was a huge pile of static HTML pages, keeping links up to date was a full-time job. Now, with dynamic websites, at least in theory, you shouldn't have any dead links, right? Dead links are a great indication for out-of-date, misconfigured, or just poorly written software. Combined with attack proxy software, this can be a deadly combination.

The other side of link checking is that it makes for a great load test. For example, an untuned WordPress site will typically fall over after a few dozen requests in rapid succession. Another thing to look for while checking links is page load time. If page load time varies a lot, that's a good indication you have software that could be abused.

My second favorite tool is Google (Figure 1), although other search engines will also work. Search terms like *inurl:wp-config.php* and *DB_NAME* will, sadly, locate working instances of `wp-config.php`. Even better, or worse, toss a *-localhost* in there to find remotely accessible database servers. ■■■

INFO

[1] "Attack of the CSRF" by Kurt Seifried, *Linux Magazine*, February 2009, pg. 66, *http://www.linuxpromagazine.com/Issues/2009/99/Security-Lessons/%28kategorie%29/0*

[2] Guide to Secure Web Services, NIST Special Publication 800-95, *http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf*

[3] OWASP: *https://www.owasp.org/*

[4] "Web Security" by Kurt Seifried, *Linux Magazine*, September 2008, pg. 45, *http://www.linuxpromagazine.com/Issues/2008/94/Security-Lessons/%28kategorie%29/0*

[5] OWASP Store: *http://www.lulu.com/spotlight/owasp*

[6] AppSecUSA 2011: *http://www.appsecusa.org/schedule.html*

[7] Nikto2: *http://cirt.net/nikto2*

[8] LinkChecker: *http://linkchecker.sourceforge.net/*