The XSS Epidemic: Tools for discovery and remediation

By Russ McRee – ISSA member, Puget Sound (Seattle), WA, USA chapter

Prerequisites

Mozilla Firefox for GreaseMonkey
GreaseMonkey for XSS Assistant
Java for BurpSuite

Similar Projects

Fiddler[1]
TamperIE[2]

Commercial web app tools with free XSS checks

N-Stalker Free Edition[3]
Acunetix WVS 5[4]

Introduction

At the risk of repeating myself (we covered Paros Proxy in December 2006), I am driven to revisit web application security as my fervor over this issue has increased ten-fold since last we visited it. It is my position that the lack of secure coding practices specific to web applications is of epidemic proportions and will likely not improve in the face of trends like Web 2.0 and mashups. Call it what you will, but if you do not sanitize your input, no buzzwords, rebrands, or euphemisms for mangled code will save you from the evildoers. Add to the mix competition in the rich Internet application (RIA) space and the attack vectors multiply exponentially. Adobe Air, the Google Web Toolkit, Microsoft Silverlight, Sun's JavaFX, and open-source offerings like Appcelerator or OpenLaszlo all aid in the creation and deployment of content and applications while integrating HTML, AJAX, and Flash.[5] Adobe has already had their share of XSS issues with authoring software; refer to Thomas Claburn's piece, "Adobe Fixes Flash Vulnerabilities," in *Information Week* for details.[6]

Given how simple it is to code in a fashion that prevents script execution in the context of one's site, I see no possible reason why endless numbers of sites remain vulnerable to cross-site scripting (XSS). The fact that site operators leave open redirects available with the excuse that it's "by design" makes me nuts. Should a simple single tic behind a query string parameter result in an '80040e0c' error, something is surely amiss. Imagine your author ranting like Louis Black about politics, "What's the matter with you people?!" Forgive me in advance; I will be will spending time on my soap box this month.

Conventional rules of engagement apply here: test only what is yours, or that for which you have permission. There is much debate over what is acceptable and what is not with regard to conducting even the simplest of alert tests, so I suggest erring on the side of caution.

We will depend heavily on the NTO Hackme Test Site[7] for our exercises this month. You are welcome to test any or all methods discussed here on this "learning opportunity" but be respectful of the fact that they are trying to maintain the site as vulnerable yet stable. Suffice it to say that pounding their server would be rude.

Given the prevalence of XSS vulnerabilities, which will be our primary focus, we will throw in discussion of redirects for good measure. Redirects are particularly annoying as they so easily benefit those who choose to phish for a living rather than earning an honest wage. They are the lowest hanging fruit with the biggest bang for the buck, thus most often taken advantage of by the nasty and nefarious.

Required reading, should you choose to accept this mission, is *XSS Attacks* from Grossman, Hansen, Petkov, Rager, and Fogie, a Syngress publication.

Manual testing

Nothing beats the straightforward, fundamental approach to manually testing your web applications for vulnerabilities. No tools, no vendors, no problems. There are valuable reference sites for helping you craft manual XSS test strings, the

1 www.fiddler2.com/fiddler2.
2 www.bayden.com/other.
3 www.nstalker.com/products/free/download-free-edition.
4 www.acunetix.com/cross-site-scripting/scanner.htm.
5 Darryl K. Taft, "RIA competition ramps up," *eWeek*, March 3, 2008, p.16.
6 www.informationweek.com/internet/showArticle.jhtml?articleID=205901512.
7 http://hackme.ntobjectives.com.

most useful of which is RSnake's XSS (Cross Site Scripting) Cheat Sheet.[8]

We will start with the open redirect as it is the one vulnerability that causes me the most consternation, for two reasons. First, it is so easily avoided; the "by design" excuse is a lame one. If site operators claim they must absolutely use this method, suggest the use of intermediary pages to advise users of the redirection, or allow redirection only to specifically whitelisted sites. Second, this vulnerability is one so easily exploited to take advantage of the innocent. Consider the following arbitrary URL:

`http://www.issanationalbank.com/partners/page.redir?target=http://www.fdic.gov/`. This is seemingly innocent and intended to take users to a valuable resource. Sadly, the same URL can be manipulated to take users anywhere. So, were I a malicious phisher, I could take the source code from *issanationalbank.com* and create my own fake bank site,. We will call it the *russnationalbank.com*. I can then target email to assumed or known users of the *issanationalbank.com* with the following URL:

`http://www.issanationalbank.com/partners/page.redir?target=http://russnationalbank.com`

To the less wary, the URL looks reasonable, with the right mix of social engineering, fear mongering, and elegant code fakery, innocent users could be easily duped – all because some well intended site designer forgot to lock down `page.redir?target=` to allow only approved partners. The impact of this fundamentally simple attack is profound for both consumers and businesses. Turn them off or tune them up; they will be exploited.

Using a convenient page offered by Duke University, *http://www.phy.duke.edu/~icon/work/clac/examples/inigo.php*, let's investigate some simple test strings. Using this URL we see a simple page that asks you input your name. First hint: input. Before trying any actual script strings, I always throw in the word test to see what parameters are returned in the URL window. Doing so here yields `http://www.phy.duke.edu/~icon/work/clac/examples/inigo.php?myname=test`

Second hint: the variable taking input on this simple little page is myname. Obviously, it accepted test and told me "Hello, test! Welcome to XSS 101." The most fundamental test is the generation of an alert window. You can directly copy and paste a string like `"><SCRIPT>alert('XSS_Alert')</SCRIPT>` into the form field, or you can stick it behind `myname=` like this:

`http://www.phy.duke.edu/~icon/work/clac/examples/inigo.php?myname="><SCRIPT>alert('XSS_Alert')</SCRIPT>`.

If I am going straight to URL manipulation, rather than using the form field, I usually URL encode my string. The URL Encoder[9] is very useful for this effort. `"><SCRIPT>alert('XSS_Alert')</SCRIPT>` encodes like this: `%22%3E%3CSCRIPT%3Ealert%28%27XSS%5FAlert%27%29%3C%2FSCRIPT%3E`.

This can be helpful when rudimentary filtering may be in place on the site, disallowing certain characters. So in this case my test URL would look like this:

`http://www.phy.duke.edu/~icon/work/clac/examples/inigo.php?myname=%22%3E%3CSCRIPT%3Ealert%28%27XSS%5FAlert%27%29%3C%2FSCRIPT%3E`.

Regardless of how you choose to conduct your test, the result from a vulnerable parameter will look like Figure 1.

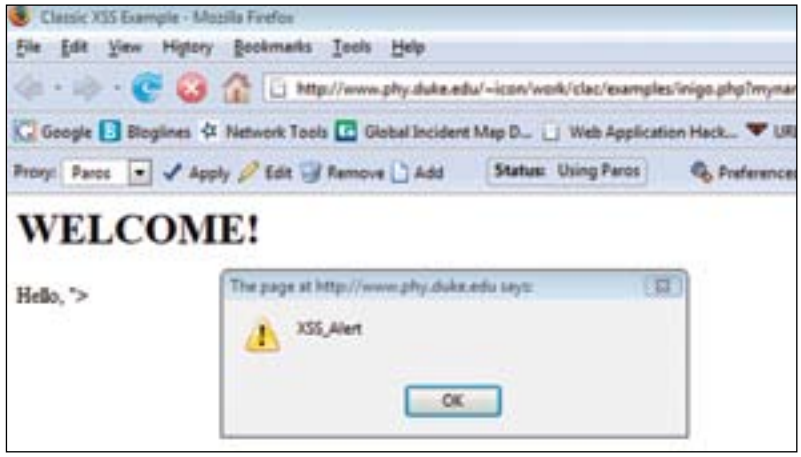

Figure 1 – XSS Alert via manual test

You can render a number of different responses in your browser by varying your test strings.

`"><SCRIPT>alert(document.domain)</SCRIPT>` will return the domain associated with the vulnerable page.

`'"><iframe src=http://xssed.com>` will build an IFRAME displaying the website xssed.com, the fine work of Kevin Fernandez and Dimitris Pagkalos.

`"><INPUT type= "text" name= "txtBox" value= "Please fix me soon!">` is lovely if you need to create an URL to drive your point home to your wayward web developers or management.

Figure 2 – Text box says it all

As you review the sites in your care, you may find it easiest to spider them first (see the Proxy section), then use the strings we have discussed, create your own, or use the cheat sheet. Regardless, spidering will point out all the possible variables

8 http://ha.ckers.org/xss.html.

9 www.webreference.com/cgi-bin/perl/7/encode.pl.

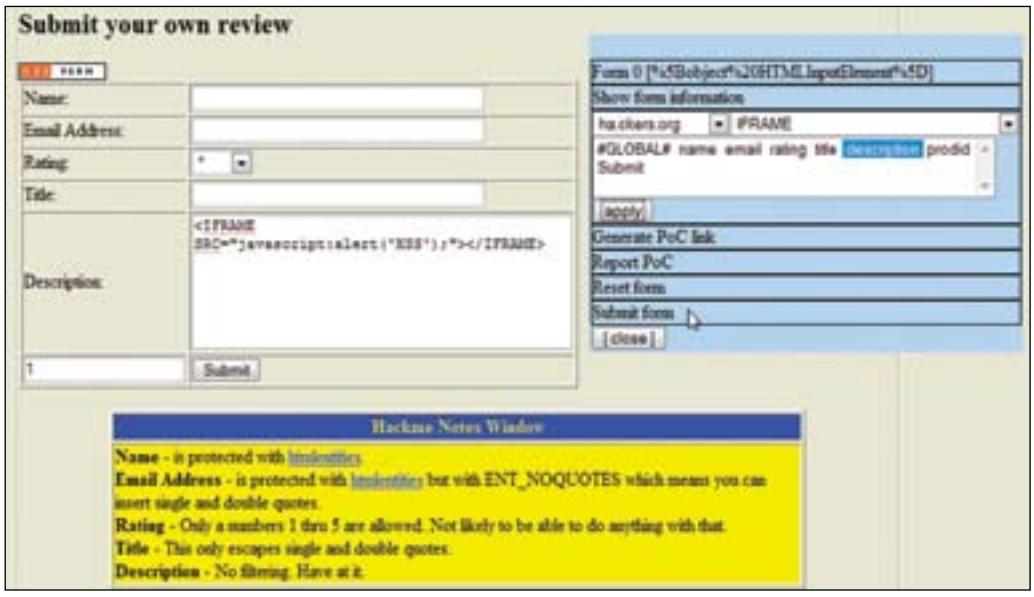

Figure 3 – XSS Assistant

to test. You can also browse directly to pages of concern and put our next feature to immediate use.

GreaseMonkey and XSS Assistant

My current hands-down favorite tool for quick and easy XSS vulnerability testing is XSS Assistant, a script for GreaseMonkey, available at WhiteAcid's Nexus. GreaseMonkey is a Firefox Add-on that can be retrieved by clicking Tools, Add-ons, and then Get Extensions. After installing GreaseMonkey, visit *http://www.whiteacid.org/greasemonkey* and click the Installation link.

Using some example URLs from the NTO Hackme Test Site, let's explore some basic XSS issues using XSS Assistant. In your Firefox browser choose *Tools, GreaseMonkey*, then select *Enabled*. Give it a few seconds, then return to *Tools, GreaseMonkey, User Script Commands*, then *Start XSSing* forms.

In the section referred to as "Episode #02 - Cross Site Scripting (XSS) Part 1 [Intro]," you will find an Attackable Product Review System.[10]

You will notice now as you visit this site, you will see an XSS FORM button by an area of the page with possible input variables. Click the button and a menu will pop up. As the site describes, the easiest variable to monkey with is *Description*, and as you can see in Figure 3, I selected *Description* and chose *IFRAME* under *Select a Vector*. First, click the *Apply* button (you will see the form populate with your chosen string) then *Submit* form.

You will be thanked for your submission then advised to "return to the product details" where you can view your handy work (see Figure 4).

You can find yourself tampering and experimenting for hours and XSS Assistant will offer you endless options.

Proxy Tools

Proxy tools are of great use when analyzing your web applications for security vulnerabilities. I will mention Paros Proxy[11] again, as it is one of favorites, but we will focus on two others well worthy of your consideration.

BurpSuite

BurpSuite is a standalone Java-based platform for assessing web applications that can be leveraged by any browser. BurpSuite excels in handling HTTP requests, authentication, downstream proxies, logging, alerting and extensibility. I consider it a proxy on steroids; if web application assessments were a sport monitored for performance enhancing substances, the BurpSuite would be illegal.

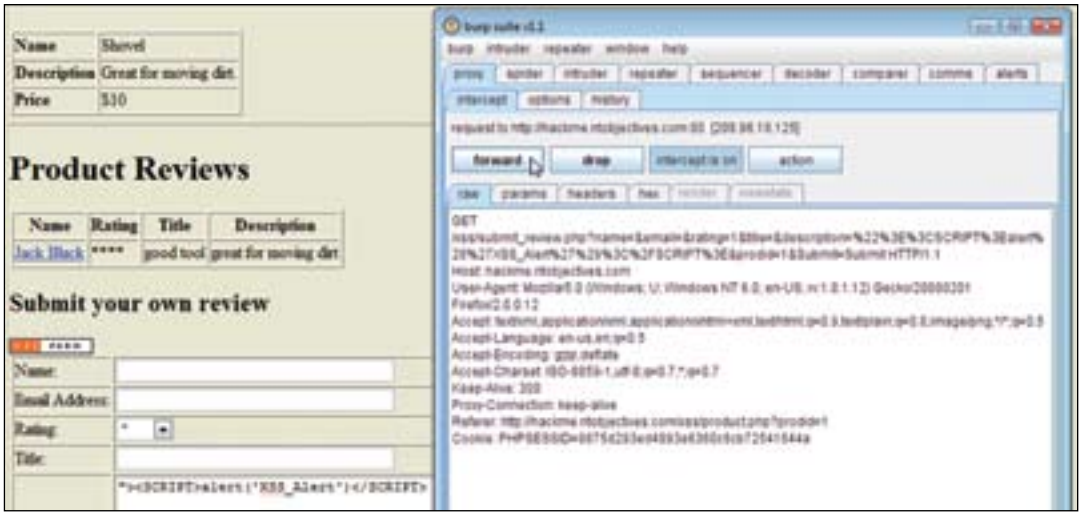

Figure 4 – Results

BurpSuite allows you to combine manual and automated techniques to enumerate, analyze, assess, and exploit web

10 http://hackme.ntobjectives.com/xss/products.php.

11 http://parosproxy.org.

applications. Download BurpSuite, execute `burpsuite.bat` on Windows systems, or run `java -jar -Xmx256m burpsuite_v1.1.jar` on *nix systems. From the BurpSuite site,[12] features include:

- Ability to "passively" spider an application in a non-intrusive manner, with all requests originating from the user's browser
- One-click transfer of interesting requests between tools, e.g., from the Burp Proxy request history, or the Burp Spider results tree
- Detailed analysis and rendering of requests and responses
- Improved analysis of HTTP requests and responses wherever they appear, with browser-quality HTML and media rendering
- Burp Sequencer, a new tool for analysing session token randomness
- Burp Decoder, a new tool for performing manual and intelligent decoding and encoding of application data
- Burp Comparer, a new utility for performing a visual diff of any two data items
- Ability to follow 3xx redirects in Burp Intruder and Repeater attacks
- Improved interception and match-and-replace rules in Burp Proxy
- A "lean mode" for users who prefer less functionality and a smaller resource footprint

Allow me to provide a quick example of BurpSuite in action. We return to the hosed up Product page at the NTO Hackme Test Site. BurpSuite with intercept on will produce every request in raw, params, headers, hex, render, or viewstate tabs. It will ask you as part of its interception whether you'd like to forward or drop the request. As you can imagine, this will allow you to modify and resubmit as you see fit, tweaking your test strings accordingly.

The proxy piece is just the tip of the iceberg when it comes to this suite. Burp Intruder is where the steroids really kick in, providing the ability to enumerate identifiers, harvest useful data, and fuzz for vulnerabilities. "The types of attacks that are appropriate will depend on the application in question, and may include: testing for flaws such as SQL injection, cross-site scripting, buffer overflows and path traversal; brute force attacks against authentication schemes; enumeration; parameter manipulation; trawling for hidden content and functionality; session token sequencing and session hijacking; data mining; concurrency attacks; and application-layer denial-of-service attacks."[13] Lions and tigers and bears…oh my! Add it to your arsenal, if you have not already; you will

Figure 5 – BurpSuite intercept on

quickly learn a great deal about web application behavior and methods by which to manipulate it.

Resources for mitigation and memediation

If I may be so humble as to suggest that, should you find XSS vulnerabilities in your random travels about the WWW, you report them to the site owners and offer them resources to help them improve their online posture. You would be amazed at how responsive most businesses and developers are when they realize that you come in peace and do not mean them harm. Excellent resources include:

— OWASP PHP AntiXSS Library Project
— www.owasp.org/index.php/Category:OWASP_PHP_AntiXSS_Library_Project
— Microsoft Anti-Cross Site Scripting Library V1.5
— http://msdn2.microsoft.com/en-us/library/aa973813.aspx#a nticross_sitescripting_topic4_four
— CGISecurity.com
— www.cgisecurity.com/articles/xss-faq.shtml
— </xssed> xss attacks information
— http://xssed.com/xssinfo

Benefits and drawbacks

Improving web application security for the businesses you work for or the customers you serve is essential to their reputations and fiscal well being (think T.J. Maxx). Protecting these assets brings obvious benefit.

The drawbacks might include push back from developers or management members who "do not get it." Further, you can get yourself in trouble testing applications without permission or those that do not belong to you.

12 http://portswigger.net/suite.

13 http://portswigger.net/intruder/help.html#what.

In conclusion

There will be constant, evolving development in the online space, complete with merging technologies and applications. Where that development fails to follow secure coding practices, opportunities will abound for those you'd rather keep out of your customer data, critical internal assets, or user sessions.

Ensuring that you or someone on your staff is tasked with monitoring and securing your web applications is an essential part of protecting brand, consumer, and reputation. May these tools and resources lead to your continued, successful defense.

Cheers…until next month.

Acknowledgments

Kevin Fernandez and Dimitris Pagkalos for their dedication in fighting this epidemic.

Jeremiah Grossman and all the contributors at Planet-Websecurity.org for their relentless pursuit of a more secure Intarweb.

Dafydd Stuttard (Portswigger) and WhiteAcid, for obvious reasons.

Resources

— http://hackme.ntobjectives.com
— www.microsoft.com/technet/community/columns/secmvp/sv0505.mspx
— www.whiteacid.org
— www.mightyseek.com/web-hacking-toolkit
— http://msdn2.microsoft.com/en-us/library/aa973813.aspx#anticross_sitescripting_topic4_four
— www.owasp.org/index.php/Category:OWASP_PHP_AntiXSS_Library_Project
— http://xssed.com
— http://portswigger.net
— http://planet-websecurity.org

About the Author

Russ McRee, GCIH, GCFA, CISSP, is a security analyst working in the Seattle area. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ@holisticinfosec.org.