JavaScript

Internet Security. If ever there was an oxymoron, that is it.

The Internet was not designed to be insecure. The actual issue is that the internet was initially designed to be a simple communications tool between a few colleges. This tool was part of President Eisenhower's Advanced Research Projects Association, or ARPA for short (A Technical History of the ARPANET). There is no thought of security because nobody had ever done such a thing before.

We are now in a time where web browsers are powerful enough that many applications are written to run in a browser instead of an operating system. This power makes it a deadly vector for evildoers.

There are two sides to internet security. The programmer's side, and the user's side. I am addressing this to the programmer.

As a web master, you must keep two vital goals in mind. You must get your employer's/client's message to the target audience by whatever trickery you can. You must get your site listed highly on search engines (no cheating allowed). You must also think of you audience's safety. Having your site hijacked to sell Viagra or Canadian pharmacy products will get you blacklisted on filtering software. It is very difficult to get a site un-blacklisted.

The best thing you can do for your employer/client is learn how to write secure code. There are many facets to this subject. Some languages are more secure than others. Nonetheless, no matter what language you choose there are techniques that will improve the security of the site you are working on. Mastering these techniques will improve your reputation and therefore your employability.

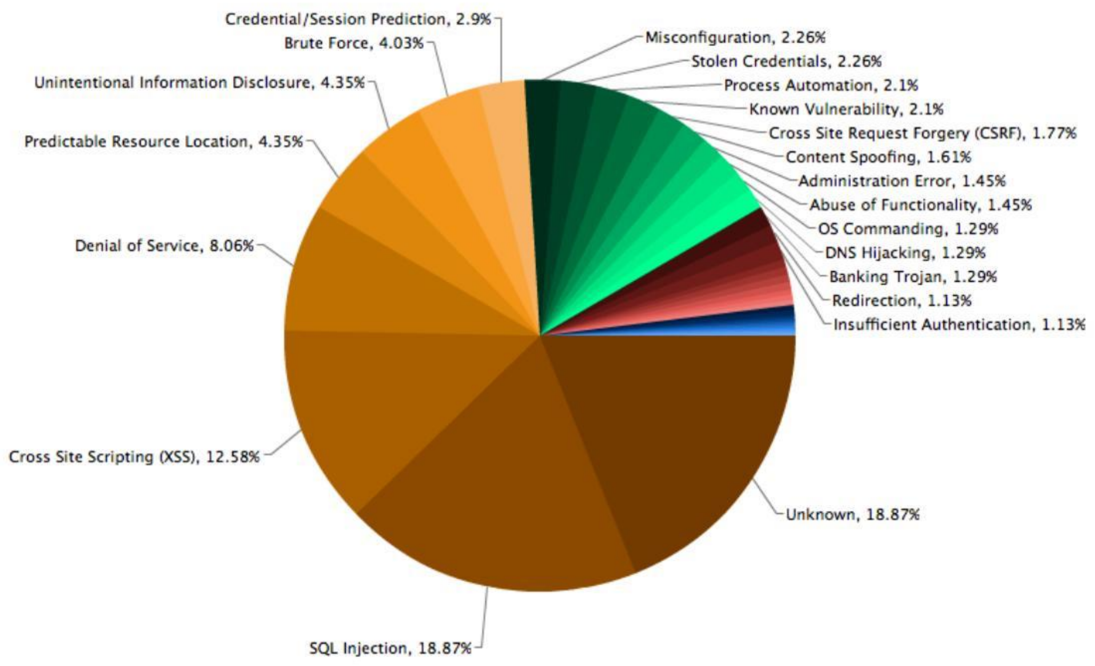
For your own sanity, get this into your head: **you cannot stop all attacks!** No matter how hard you try. The bad guys are smart. The bad guys are patient. The bad guys have lots of time on their hands (for some, it is a full-time job). The bad guys learn from one another. The bad guys are well motivated (money is a very good motivator). And they are legion

Some of these evildoers are called ‘script kiddies’. They download programs and scripts, tweak them, and let them loose. Many of these are funded by persons from countries where the laws are lax or non-existent. They are paid a small fee for each username/password pair, credit card number, or other account information. These handlers are getting smarter about identity theft. Instead of draining a few accounts dry, they are sucking small amounts at a time. Most people don’t watch their accounts very closely, and even if they do notice something amiss, it is very easy to get a small charge removed or reversed. Individuals are unlikely to report to the Attorney General. If the bad guys can get $14.00 from 50,000 accounts, they have a cool $700000.00. Not much, but okay for a day’s work. Like I said, money is a very good motivator.

Having gotten that out of the way, we can return to our program already in progress.

Javascript is one of the most popular web programming languages, and one of the least secure. Javascript was invented to solve a problem: web pages were totally static. HTML as originally designed allowed for headers, colors, links, and not much else (Tim Berners-Lee, 1993). Javascript allowed certain browsers (do you remember Netscape?) to bring web pages to life. It gave us many new capabilities. These capabilities unintendedly brought new risks.

The biggest problem currently is called Cross-Site Scripting. This is not actually a Javascript problem, but Javascript is one of the most commonly used languages[1] to implement attacks.



There are many examples on the web that explain how to do it better than I could. Some are better than others. I like QuackWare's YouTube videos.

To hugely oversimplify, a Cross-Site Script attack (also known as XSS or CSS) occurs when a crafty villain crafts a crafty script that can be inserted into a poorly crafted website. There are many ways for this to happen. Because the 'engine' that displays text is linked to the script interpreter (I have no idea why). So a response to a form can actually have Javascript script entered. When this entry is accepted, the script will be executed. The same is true of a site that asks for customer comments. If a script is entered as title or content, the next person who logs in and views the comments will reap the "benefits" of this script. This script may simply display a snide remark about the

[1] PHP is gaining in popularity quickly

site's security, or it might redirect the customer to a different site, or it might steal their cookies.

What do I care about someone getting their hand into the cookie jar? Consider the number of people who click the 'remember me' box on a web page. This puts a cookie on their computer that has their username and password. Granted, they are encrypted (at least, they should be), but the bad guys are not necessarily interested in actually decoding usernames and passwords. Why go to the trouble when all I need to do is put the cookie on my computer and visit the website? The cookie is read, the password checks, and I am in. On the other hand, the bad guys have software that can decrypt the username and password. Since between 30% and 80%[2] of internet users use the same password for Facebook as they do for their home banking, having that one cookie can be a gold mine.

A Cross-Side Script works like this. Since an iframe gets its content from another page, suppose I have code which does something like this which is displayed in an iframe:

```
<head>
function showstuff()
  {
    document.forms[0].Text1.value = TheText;
    document.write(TheText);
  }
</script>
</head>

<body>
    justa testie
    <p>&nbsp;</p>
    <p>&nbsp;</p>
    <p>&nbsp;</p>
    <form id="stupid"method="post">
        <input name="Text1" style="width: 1006px; height: 179px" type="text">
        <br><br>
        <input name="submit" type="button" onclick="showstuff()" value="click
me!">&nbsp;
    </form>
```

[2] Depending on which survey you disbelieve least.

```
</body>

</html>
```

Yes, this is a dumb example. Work with me here. If I were to type the following:

```
<script>if (top!=self) {self.location="http://www.youtube.com";}</script>
```

As soon as I punch the “click me!” button, I will find myself looking at youtube.com in the frame.

Let us consider what might happen if I had a page with a comments section. Were I to put some code like the above into the subject or comments box, the next lucky person to read the comments will be greeted with youtube instead of the scintillating commentary expected. (so will everyImagine if I instead used an address like http://givemeyourpersonalinformatin.xxx.

The best defense against this sort of thing is called input sanitizing. You as a girl/boy genius must never trust the browser to take care of your customers. Your web page code must check any incoming data before you allow it to be displayed. If, for example, your code takes the input into a Javascript or PHP or Perl (or vb, or…) string it can be examined prior to accepting it for display. PHP5 includes some handy functions:

- FILTER_SANITIZE_EMAIL
- FILTER_SANITIZE_ENCODED
- FILTER_SANITIZE_MAGIC_QUOTES
- FILTER_SANITIZE_NUMBER_FLOAT
- FILTER_SANITIZE_NUMBER_INT
- FILTER_SANITIZE_SPECIAL_CHARS
- FILTER_SANITIZE_STRING
- FILTER_SANITIZE_STRIPPED
- FILTER_SANITIZE_URL
- FILTER_UNSAFE_RAW

Javascript functions can be written (or downloaded) to perform similar functions, but it is not recommended. It is difficult and will make your pages load slowly.

Here are some general steps you as a careful and conscientious webmaster can take:

- Learn Javascript's weaknesses and how to work around them.
- Avoid doing sanitizing on the client machine. Send text to the host and check it out there. If you do not allow an evil script to be displayed, it will not be executed.
- Scan user inputs for *allowed* characters. The bad guys are constantly coming up with new combinations of characters to do evil. Testing for disallowed input can be spoofed if the hacker is motivated.
- Watch out for unusual characters like "<" and ">".
- Beware of the "%" character. As in "%3C" and "%3E". Also the Unicode version, "\u003C" and "\u003E".
- Use a javascript obfuscator. The Evil Ones will scan your code before deciding on which hack to use. If they can't read your code, they will look elsewhere. As I said before, you can't stop them if they have decided you have something they want.
- Who do they think they are fooling with URLs like file:// or javascript:// or vbscript://. Reject them out of hand.
- Use *signed scripts*. This technology was originally developed by Mozilla.org. it involves wrapping your .js files in a jar file, and obtaining a security certificate from a trusted authority like thawte.com or verisign.com. http://www.mozilla.org/projects/security/components/terms.html#sig has detailed information.

- Use *put* instead of *get*. Get sends all the information as part of the url[3]. Not only does that make your data visible to anyone listening, it also will be kept in the browser's history and cache.
- Be careful with JSON. Always require authorization
- Suggest (politely) that your customers use the most current version of their browser.
- Beware of 'new and improved' web products. Microsoft Silverlight is pretty cool, but it requires an activex download onto the client site.
- Use HTML5. There are many improvements, and most current browsers support almost all of its features.
- Learn to write secure code.
- Subscribe to a security newsletter (see websites in appendix A). Learn proper syntax, best practices, and much more at www.w3c.org, the (semi) official World Wide Web standards organization. Submit your code to their compliance checker, and earn the right to their mark on your web site. The checker will alert you to subtle errors that your can probably get away wit today, but maybe not tomorrow. Like having a *form* with no *action* specified.
- If your site is involved in ecommerce It will probably be worth it to have your site checked by a company such as Cybertrust. If you check a site like Best Buy you will find a seal indicating the site has been checked by Cybertrust, McAfee, Norton, or a similar service. Your site will be scanned regularly for vulnerabilities and ensure it has not been hacked.

[3] You already knew that.

❖

There is much more, but there is not room here. See the website list in appendix a for sources of further information.

The second best thing you can do for your employer/client and their clients is to scan your site regularly. Daily is no too frequent. Just as it is possible for a poorly secured personal computer can be hacked, so can a poorly secured web server. It is almost a non-issue once the server is penetrated to hack your site. Your pages can be modified in any way desired or replaced altogether.

It only takes a little bit of time to compare file sizes and modified times. Being the brilliant web app author, you can probably toss off a script to do this automatically. If you haven't changed your site, it should not have changed.

PLEASE BE AWARE THAT MISUSE OF WEBSITE SCANNING TOOLS CAN BE CONSIDERED AS HACKING. HACKING IS A FEDERAL OFFENSE WITH SEVERE PENALTIES. NEVER USE A WEBSITE SCANNING TOOL FOR ANY WEBSITE THAT YOU DO NO OWN WITHOUT ***WRITTEN*** PERMISSION FROM THE OWNER. DON'T BELIEVE FOR A MINUTE THAT YOU WON'T GET COUGHT.

There are many programs available to scan your site for busted links and poorly secured code.

I use Xenu (http://home.snafu.de/tilman/xenulink.html) for broken link studies.. There are some others out there, but most of them either are trial versions or hopelessly out of date, or both. When you get out into the actual 'real' world, you will probably have a budget and some leisure time to choose a commercial product. But for now, Xenu will give you a taste.

Xenu is amazingly well designed and hugely easy to use. You, as a fledgling web security guru, start it, give it a url to scan, and it scans it There a few options available, but they can be easily ignored. When it is finished, it will ask you if you want a report (duh!). It will ask for FTP information. If you simply click ok without giving it the information it craves, it will claim an error. Wah. The report will be downloaded and displayed in your browser[4].

Webcruiser (http://sec4app.com/) is a vulnerability scanner. It is a bit complex. I recommend studying the help to learn how to use it.

Learn how to write secure code.

[4] Am I the only person who sees the irony here?

APPENDIX A – IMPORTANT WEB SITES

http://www.mcafee.com/us/mcafee-labs.aspx

http://www.techrepublic.com/blog/security/join-this-upcoming-webcast-on-malicious-javascript-threats/4824 (This is a webcast on Javascript security).