

IDBData

# Site Security Cyber attacks & Badware

Security & Privacy

Ian Baker



09

## **There are three basic steps to maintaining a clean site:**

1. [Identifying badware behavior on your site](#)
2. [Removing badware behavior from your site](#)
3. [Preventing badware behaviors in the future](#)

### **Identifying badware on your site**

The first step to keeping your website badware-free is to check for any badware or badware behaviors that may already be on your site. Badware is software that fundamentally disregards a user's choice over how his or her computer will be used. Many sites with badware problems are not actually hosting badware themselves, but instead exhibit other "badware behaviors" such as automatic redirects or prominent links to badware on other sites. Often these badware behaviors are the result of hacking attacks or compromised third-party content, such as ads, rather than any deliberate actions by the website's owner. You can learn more about badware and badware behaviors in our [Guidelines](#).

**When looking for badware on your site, especially badware due to hacking attacks, please remember to check the source code of your site as it is currently hosted on your web servers.** Many site owners mistakenly look just at the website files they have on their own computers, and so miss seeing the evidence of attacks to the site itself.

If your site has been flagged with a malware warning by Google, check the [Google Diagnostics](#) page for your site for more information about the problems Google found.

Here are some common types of badware to look for:

### **1. Badware available for download on your site**

Evaluate the software that you are offering for download – including any third-party applications that are bundled with your software – based on StopBadware's [Software Guidelines](#). If the software that you are offering for download violates our guidelines, then it constitutes badware.

If your software is bundled with third-party applications, you may also want to check whether the bundled applications install any dangerous or deceptive code. One method for detecting this is to download the entire software bundle onto a virtual machine and scan it using anti-virus or anti-spyware programs.

### **2. Badware available on sites that you link to**

If your website links to badware, your site's visitors may be in danger, even when the bad software or code exploits are not actually hosted on your site. Your web pages may violate our [Website Guidelines](#) if they automatically redirect to a website that hosts or distributes badware; directly link to executable files that are badware; link to another website that automatically attempts to install badware by exploit onto the

user's computer; or contain substantial links to other website(s) that predominantly host or distribute badware.

Some ways to determine whether the links on your site violate our guidelines would be to check whether any of your links lead to bad software available for download on another site, or whether they lead to an infected page on another site. (We recommend that, when looking for badware, you use a virtual machine to avoid damaging your own computer.) It may also be useful to search through your site's source code and look for links to unknown sites, especially if the links are to executable files. Executable files include files with extensions such as .exe, .bat, .cmd, .scr, and .pif. There are also applications available that will allow you to scan for malicious links within a web page, and you can use these applications to help decide whether to link to that page.

You can also use the StopBadware application alerts and our [Badware Website Clearinghouse](#) as a resource to search for information on the sites and software to which you link or are planning to link.

### **3. Badware distributed through ads running on your site**

Advertising displayed on your site is another potential source of badware, since most ads include direct links to an external web page. Please see [section 1.2](#) above for general information about our guidelines for badware found via links. If you display third-party ads on your website, check that the links do not lead to bad software or to a badware-infected web page. The methods for evaluating the software that is available through ads are similar to those described in [section 1.1](#).

Use internet searches to check out the ad networks you use to learn whether other websites have had badware problems with those ad providers.

### **4. Badware links posted in user-generated areas of your site**

If there are any areas of your site where users can post or upload content, these areas may be a potential source of badware or badware links. Please see sections [1.1](#) and [1.2](#) above for information about badware and badware links.

### **5. Hacking attacks to your site**

Another common source of badware on websites is hacking attacks, which allow third parties to insert code or executables onto poorly secured websites. A common example is the "injection attack," in which a hacker uses a security vulnerability to inject harmful code into one of your web pages. Usually this code will be invisible to visitors to your site, but can trigger a silent badware download in the background of a visitor's computer. You can often detect whether this kind of attack has occurred by looking at the source code of your web pages and determining if contains any code that you did not place there. Be sure to look at the code as it is appearing live on your web server.

## **Two common types of “injection attack” are:**

### **Invisible iframes**

Iframe tags are one of the many kinds of HTML tag codes that can be used as part of the source code that creates a website. An iframe creates a small window on a webpage so that another page can load inside the embedded window. Iframes are not always used for nefarious purposes; one frequent use, for example, is to embed remotely hosted dynamic content such as online maps into web pages. When used by malicious attackers, an iframe can be made so small that it is invisible, and the visitor to the infected page never knows that another page is also loading in the tiny iframe window.

If you see code for an iframe with width="0" and height="0" in the source code of any page on your website, you have found an invisible iframe. Iframes are most commonly inserted at the very top or the very bottom of a web page's source code. A good first place to check for iframes is before the initial tag that starts a web page's standard code, or after the final that ends a page's code.

### **Obfuscated code**

Obfuscated code or scripts are designed to be hidden within the normal code for your site, so they can be hard to detect. The code is written specifically to prevent automated tools from discovering its purpose. The most commonly obfuscated kind of code is javascript, which is used to add functionality to many websites.

Obfuscated code is not necessarily bad; for example, some developers use encoding to make it harder for automated programs to detect email addresses displayed on a site, protecting the addresses from spam harvesters. However, if you write the code on your site and you do not intentionally obfuscate, a block of obfuscated code on your site may indicate an injection attack. The two most common ways code is obfuscated are through encoding and encrypting.

Encoding can sometimes be easy to spot because the encoding uses either “hex” or “unicode/wide” characters. For hex characters, you will see strings of percent signs with two characters after them (e.g. %AA%BB%CC). For unicode characters, you will see strings of “\u” with four characters after (e.g. \u0048\u0069\u0021). These blocks of encoded text can take up several paragraphs.

Encrypted code is harder to find, because there are no set patterns. However, encrypted code will look like a block of unintelligible text. Normal javascript uses a syntax based on actual English words. Encoded or encrypted text appears in a site's source code as completely unintelligible blocks of letters, numbers, and symbols.

While most hacking attacks focus on html code, it is also possible for bad software itself to be uploaded onto a poorly secured site. Bad software can include unknown executables (such as files that end in .exe, .bat, .cmd, .scr, and .pif), javascript files, or even images uploaded to your site without your knowledge. Sometimes attackers

will simply use your website to host badware and link to it from other victim sites. One method for detecting whether you are hosting bad software on your site is to download all of your source code from the live website onto a virtual machine and scan it using anti-virus and anti-spyware programs.

## **Removing badware from your site**

How you should go about removing badware from your site will depend on what kind(s) of badware your site is hosting or linking to. Our general recommendation is to take your website offline while you clean and secure it, to prevent your site's visitors from being unwittingly infected in the interim.

### **1. If your site is hosting bad software for download**

Remove the bad software from your website and don't make it available for download again unless you can be sure that it is no longer badware. You can learn more about what makes a piece of software badware in our [guidelines](#). If you are the creator of the software in question, StopBadware may be able to offer recommendations for bringing your software into compliance with our guidelines.

### **2. If your site links to badware**

Remove all badware links from your website.

### **3. If ads on your site are linking to badware**

Remove all ads that link to badware. If you use an ad network, this may mean removing all the network's ads from your site until you can be sure the network is clean. You may also want to contact your ad provider and let them know that one or more of their ads is causing badware to be linked from your site.

### **4. If badware is linked in user-generated areas of your site**

Remove the badware links from your site. This may involve editing user posts to remove the badware content, or deleting entire user posts.

### **5. If your site has been hacked**

Take the site offline in order to keep from putting your site's visitors and your customers at risk. Then remove all of the offending code and fix all underlying security vulnerabilities before putting your site back online. Finding and removing a specific block of bad code that a hacker has inserted can clean your site for a time, but keeping your site from being infected in the future will require fixing the security vulnerabilities that allowed the hacker to insert the code in the first place. As such, be sure to check for and remove any backdoors left by the attacker. A backdoor will allow an attacker to get back into your site even after you have locked down the site.

Your hosting provider should also be able to help you figure out where the underlying vulnerabilities on your site are, so contacting them should be a top priority if you think your site has been hacked. You can also check your hosting provider's forums to see if any other webmasters using that host have been compromised. Checking user forums for the software used by your site can also help you see if other users have been compromised through flaws in the software, or if there are security updates which your site does not yet have in place.

## **Preventing badware in the future**

### **1. Check software for badware before making it available for download**

See [section 1.1](#) above for general information on badware and our software guidelines.

### **2. Check links for badware before posting them to your site**

See [section 1.2](#) above for general information and our guidelines about badware in links.

### **3. Use only reputable, conscientious ad providers and regularly monitor them to be sure they stay clean**

Make sure your ad network is reputable and actively screens for badware from advertisers. If not, switch and tell them why you switched. Remember that an ad shown on your site, even if provided and hosted by a third party, is still a part of your web page. You should only accept ads from providers that you are confident are diligent about protecting clients from badware. Use internet searches to check out the ad networks you are considering using to learn whether other websites have had badware problems with those ad providers.

### **4. Monitor user-generated areas of your site**

Make sure your terms of use for posting to forums, blogs, and other user-generated areas of your site explicitly forbid posting links to badware. Then actively monitor these areas of your site for suspicious links or executables. You may also choose not to allow users to link directly to any form of executable file or to insert javascript into forum messages or other user-generated content areas. See [section 1.2](#) above for general information and our guidelines about badware in links.

### **5. Close security loopholes to secure your site against hacking**

Some basic steps that can be taken to make your site more secure include the following:

- Use strong passwords.
- Use SSH and SFTP protocols, instead of telnet or FTP. Telnet and FTP are both considered insecure because of their use of plain text protocols. They transmit usernames and passwords in a way that anyone with access to the network can read. SSH and SFTP are based on an encrypted protocol which prevents eavesdropping.
- Scan your site for security vulnerabilities using a vulnerability auditing scanner (both free and commercial versions are available). Use security update management tools to detect missing patches and then apply those patches immediately. The StopBadware online community has created a page of suggestions from our community members [here](#)].
- Keep up to date on news relating to any software you or your host use on your site, and make sure you are always running the most recent versions, including security patches. Subscribe to, and regularly read, any newsletters or alerts offered by your hosting provider and software providers.
- Make sure your hosting provider keeps all software updated, including security patches. If they do not, urge them to do so or switch to a hosting provider that you are confident does its best to keep its clients' websites secure.

When your site is clean, secure, and back online, you may want to notify your site visitors about the badware problem, and the steps you've taken to address it. If a user has become infected with badware after visiting your site, knowing what you've found will help them clean up their own computer. And telling your story can help other webmasters deal with similar situations on their own sites. If you'd like to share your story of how you cleaned your site with other webmasters, or would like to share tips for keeping websites secure, please visit our [online community](#)

This page is an evolving resource. If you have further questions or suggestions for additions, please join our [online community](#) and share your ideas.

## **Improving website security**

### **XSS**

Many web-developers (although not all!) are familiar with Cross Site Scripting attacks, often known as XSS, which are usually the result of not filtering input to applications.

A standard example would be a message-board, or forum, which allowed users to login and post messages which would be displayed literally - so a malicious user could create a message reading: `<script>alert(1);</script>`

This would then result in an alert being displayed by subsequent visitors who viewed the message if they had javascript enabled.

The root cause of these security issues is trust. The implementor of the forum or message board trusted the users input and didn't sanitize it appropriately. The XSS attacks exploit that weakness to steal cookies, etc.

### **SQL Injection**

SQL injection is a problem specific to database-driven websites, especially those written in a hurry without the use of the appropriate language features.

Essentially SQL injections arise because input is passed directly to a database query without being escaped, or processed, correctly.

This is virtually identical to the XSS attack described above, just a different level of attack. (The remote-database rather than the local-client browser.)

Thankfully most programming languages and libraries which have database support make use of "binding" and other mechanisms which should make these issues trivial to prevent. The sad fact is that many programmers don't use them.

### **CSRF**

"Cross Site Request Forging" is the new name for a new problem, or one new to me at least.

The root cause of the CSRF attacks is also trust. The trust of a website by a user of that site. This trust is exhibited by the fact that the user never, or only rarely, logs out.

In a typical scenario the communications between a webserver and client browser looks like this:

- Client makes a request to the server.
- The server responds.
- The client displays/uses the response.

For example to create a new weblog entry the user would browse to this site, click upon the "[add entry](#)" link in the sidebar and submit the resulting form.

Now consider what happens if a malicious *external* site were to copy the "add weblog entry" form and host it upon their site.

They could change the wording, hide the fields, etc. But ultimately there is nothing stopping them from hosting a modified version of the form - and serving it to users.

This is where the cross-site issue comes into play. If that remote site can coerce, persuade, or trick a user logged into this site to submit the form then the request will be processed by this site using their cached credentials.

Because they *trust* this site, and have remained logged in, the submission would be processed and the unsuspecting user would have a new weblog entry posted which they didn't explicitly write, or expect. For example "I like cheese."

Step by step the attack works like this:

- External site copies a form from this one.
- External site persuades/cajoles/tricks a user who remains logged in here to visit their site and submit the form.
  - Perhaps via javascript magic.
- The server here receives the form submission and processes it
  - Because it doesn't realise the form came from a malicious remote source.
  - And because the user was logged in any authentication tests succeed.
- The user now arrives at this site with new content posted in their name.

The solution to this problem is potentially invasive but conceptually simple. Rather than serving forms to clients and then processing them without regard to the submission source the forms each include a "session token".

When a form is submitted this token is examined, and if it matches the token which was sent with the form the processing occurs, if it doesn't an alert is raised.

This utterly prevents the attack because it means that the malicious remote server cannot serve a valid form - it can't predict the secret form session token, so the submission will always fail.

(There are simpler solutions involving the use of the HTTP Referer (sic) header, but these are unreliable as this information might be forged or not present.)

It is worth noting that **many** sites fail to verify the submission of forms in this manner, so they are potentially at risk of malicious (ab)use of their forms.

## Site Changes

With this discussion in mind I will now describe three changes I've made to this site:

## Cookie Safety

Microsoft's Internet Explorer supports an extension to cookies called "[httponly](#)". The intention of this support is that the cookies served by a site will be marked as unavailable for scripting use.

This means that if there were an XSS attack available in this site then users of that browser wouldn't be vulnerable to cookie/session theft.

A bug was filed against Mozilla, but progress appears to have stalled ([#178993](#)).

I'm unaware of any XSS attacks lurking in the current codebase, so this is a small paranoia addition rather than a significant change.

(IE comprises about 9% of visitors to this site, certainly not the majority. However the change involved is sufficiently minimal that it is worth doing just to mitigate any future attacks.)

## Session Safety

On a similar front it is now possible to choose a "secure" login when you visit this site. This actually ties your login session to your IP address.

Again this offers no increased security in the normal course of events, but if there were ever to be a security issue which resulted in a cookie or session theft due to XSS, or similar, then you would be protected if you chose this option.

There is [a brief guide](#) to this facility linked to from the login form.

## Cross-Site Request Forgery Protection

This change is the most significant and is the one which caused me the most pain - as discussed above there is a potential weakness in the handling of many form submissions in online applications.

In the case of this current codebase I now insert a "session token" in each significant form, which will prevent the processing of rogue submissions rather than end-user page views.