Protecting your site and your clients

WEB SECURITY

Learn more about protecting your website with NoScript, ModSecurity, and Site Security Policy.

BY KURT SEIFRIED

Like many security issues, the World Wide Web presents two very different sets of problems with some very different solutions. On the one side, most of us use a web browser on a regular basis and want to prevent our web clients from running an attacker's code, letting them take over our machine. On the other side are web servers, which you don't want to see compromised, under constant attack (XSS, SQL injection, etc.). So what's the answer? Well, there is no single answer. You need to take steps to protect both the clients and the servers because no matter how security conscious you are, you will interact with servers or clients that are less secure.

JavaScript and NoScript

For the Firefox web browser, out of 196 security advisories, 62 listed disabling JavaScript as a workaround. Additionally, the JavaScript-based vulnerabilities tend to be the ones that allow for arbitrary code execution, so any preemptive security measure that deals with them will have a significant effect.

Securing web clients against attacks is relatively simple; however, some websites might not work properly. Disabling JavaScript entirely is one option, but many sites now rely on JavaScript to present content, forms, and so on.

A more fine-grained approach is available with the NoScript plugin for Firefox [1]. The default is to block JavaScript execution, and then you can choose to allow JavaScript to run temporarily or permanently. Or, you can permanently mark a site as untrusted to prevent any JavaScript from ever being executed from that site (Figure 1).

The major downside to this plugin is that you need to pay attention to the information bar that pops up at the bottom of the screen when JavaScript is blocked (Figure 2) and decide whether or not to allow it. If you don't, you will find yourself, as I have, staring at a website wondering why it is mostly blank, or why an online form isn't working properly.

Additionally, NoScript has some basic cross-site scripting protection – URLs with characters such as " > " in them will generate a warning and give the user a chance to block their loading.

Securing the Server

As a server admin, you can't force clients to be secure, but you can protect your own server and web-based applications from attack. Protecting your server can also prevent broken clients or users who have visited hostile sites from taking actions that might harm their accounts or the data hosted on your site – for example, from a cross-site scripting attack that interacts with the user's account to change the password on their account at your site.

Apache ModSecurity

Like many security projects, ModSecurity started out as an open source project, licensed under the GPL v2 and aimed at adding a layer of security to the Apache web server [2]. The project appears to have been commercialized successfully; however, like many open source security applications, free versions are still available.

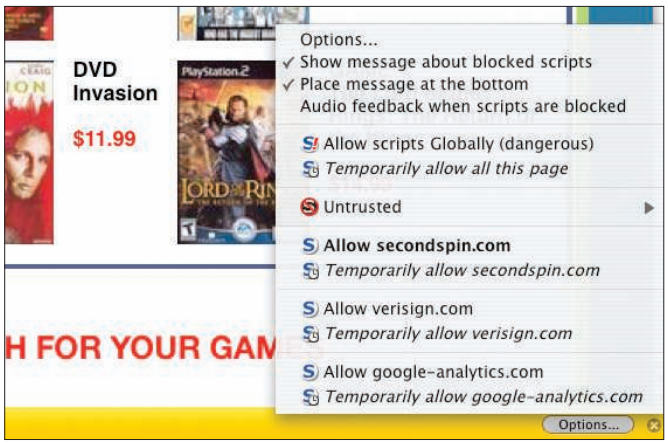

Figure 1: NoScript blocks JavaScript by default, and you can choose from a list of options.

Figure 2: An information bar at the bottom of the screen says that JavaScript was blocked.

The main benefit of ModSecurity is that you can use it to provide security for any application running on your system. On the downside, you must be able to insert a custom module into Apache – meaning you need to have control over the server – and have enough CPU power to handle the additional processing required by this module, which can be significant. The ModSecurity module allows requests to the server to be examined at various stages in the process: when the request header is first processed, when the request body is processed, when the response headers are created, when the response body is processed, and at the logging phase.

Another advantage of ModSecurity is that it supports Perl-Compatible Regular Expressions (PCRE), and the rules it supports can also trigger a variety of actions, including to allow or block a request, to drop the connection by sending a FIN packet, or to execute an external program. For example, this allows site admins to filter out characters such as " < " and " > " from requests – a likely indicator of a cross-site scripting attack – or to look for personal information such as 12-digit credit card number strings within outgoing requests (e.g., triggered by an SQL injection attack) and block that data from being served to the attacker. See the "Example Rule" box.

As you can imagine, all this power and flexibility comes at the cost of complexity; however, this is alleviated because a powerful default rule set has been made available for free (licensed under the GPL v2), which can be used as a starting point for most sites.

Example Rule

A simplistic example to detect and block any 12-digit number in outgoing web pages:

```
SecRule RESPONSE_BODY ➥
"[0-9]{12}" \ ➥
"phase:4,t:none,ctl:auditLogPa
rts➥
=+E,deny,log,auditlog,➥
status:500,msg:➥
'a 12 digit number'➥
,id:'1',tag:➥
'LEAKAGE/ERRORS',severity:'1'"
```

Site Security Policy

Site Security Policy is an interesting approach that is still in the formative stages [3]. The idea is that a web server hosts a file that specifies how a client should interact with the server, thus preventing unsafe interactions such as cross-site scripting (XSS) attacks or cross-site request forgery attacks. On the client side, there is either built-in support for this standard, or a plugin – available for Firefox – that allows the client to download and parse the policy file before interacting with the web server.

One interesting side effect to this approach is the possibility of having web proxies such as Squid support the standard, in effect protecting all the web clients behind them from potentially unsafe actions at sites that choose to support the Site Security Policy standard.

Conclusion

Web security has no simple solution: No matter how hard we try, the bad guys will either run hostile web servers or compromise other web servers. On the client side, things are basically a disaster. If you are running Linux, however, chances are quite low that you will be targeted, and chances are good that you keep your software up to date because almost all distributions update automatically by default, thus putting you ahead of the game!

By plugging the holes as they are identified and by applying additional security measures – such as NoScript and ModSecurity – you can improve the chances of "healthy" servers and clients staying that way.

Ultimately, this reduces the time and energy you have to spend on repetitive cleanup, which is something everybody wants, anyway. ■

INFO

[1] NoScript plugin for Firefox: *http://noscript.net/*

[2] ModSecurity for Apache: *http://www.modsecurity.org/*

[3] Site Security Policy: *http://people.mozilla.com/~bsterne/site-security-policy/*

THE AUTHOR

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He is married and has four cats but no fish (because the cats are more hungry than afraid of water). He often wonders how it is that technology works on a large scale but often fails on a small scale.