



Hacker Repellent:

Techniques for small companies to deter hackers on a shoestring budget

By Amit Klein
Security Group Manager
Sanctum, Inc.
Amit.Klein@SanctumInc.com

Introduction – Minimizing Hacker ROI

Today, hackers scan and probe sites of all sizes to find those most vulnerable to e-shoplifting, data theft or portals to access broader attacks. Large enterprise sites are prime targets of skilled and organized hackers because of the site's high profile, the value of the goods and services accessible, and the potential for media attention. Larger sites typically have well-funded security budgets to protect against hacker attacks, leaving smaller sites, although not targeted by the bigwigs of the hacking community, prey to script kiddies and other low-level hackers. With tight security budgets and fewer resources, small companies must implement *guerrilla* tactics to combat web application attacks.

Small sites, in particular, face the challenge of random probes by hackers who spend just enough time looking at the site to determine if it carries enough value and poor enough security measures to be worth the hackers time to break-in. The hacker's ROI is a sheer question of value-to-effort. Any site broken into within few minutes is worth the effort. A low-value site that puts up a decent fight is not.

The techniques below give smaller companies solutions that can be implemented with relatively little cost and will raise the 'effort to benefit' ratio for the hacker beyond what is worth their time. While not 100% hacker resistant, these measures can make the hacker's work hard enough to cause him/her to give up on the site.

These techniques do not replace the standard security measures even a small site needs to employ – e.g. employing network security measures, especially firewalls, applying security patches and real time fixes, hardening the web server box, doing application audits etc. These suggestions are inexpensive techniques to reduce the threat to web sites which standard network security measures do not address.



Hacking Techniques

Hackers attack web sites in various ways. For a full discussion, refer to http://www.sanctuminc.com/demo/hacking_demo_v1200.html. Two worth noting are:

- **Hidden field manipulation**
Hackers manipulate hidden HTML form fields by setting them to a value different than the original one. They often use a browser for this, as well as low level tools (e.g. wget and telnet). Examples for attacks that can be achieved using hidden field manipulation are: e-Shoplifting (by changing hidden price) and loss of privacy/impersonation (by changing hidden session ID fields).
- **Well known attacks**
Web application servers are often found to be vulnerable to attacks, regardless of the application that is built on top of them. Hackers search security lists and forums (e.g. the BugTraq list, maintained at <http://www.securityfocus.com/>) for these attacks, and carry them out manually (using a browser, or low level tools such as telnet and wget), or automatically, using CGI scanners.

Hacker Repellent Techniques

When talking about smaller companies, the hacker is usually looking for an easy target which can be hacked in several minutes. Not all techniques are applicable to all sites.. Pick the ones that best fit your site and your situation.

- **Forcing SSL access to the site**
Using SSL only, the site will block simple CGI scanners, raw interface mode tools and Internet web worms (the known ones, at least), all of these do not support SSL. Forcing SSL is easily done by configuring the server to work only in SSL (usually on port 443) and blocking traffic to port 80. If an entry page in HTTP is required, it is possible to use a tiny server that will redirect to the HTTPS address.

Downside: Managing the SSL (acquiring certificate), changing absolute links to "https://...", performance impact, and having the main page served in SSL (or having to run another tiny HTTP server).

User Impact: The latter two issues impact user experience.



- **Using the HTTP POST method as extensively as possible.**

Using HTTP POST reduces the ease of raw mode interface tools and browsers. With raw mode interface tools, the attacker is forced to manually craft a POST request, which is more complicated than a simple GET request (due to having to calculate the exact body length for the mandatory Content-Length header). With browsers, in order to manipulate some parameter values, the attacker is required to save the HTML source to disk, modify it, and reload the modified form back into the browser. This is opposed to directly manipulating the parameters of the resulting request (in the URL line) when the request method is GET.

Downside: Converting all (or as many as possible) scripts and HTML pages to work with POST.

User Impact: None.

- **Forcing HTTP authorization on all (or most) of the contents of the site**

This simple method will work well against Internet worms, simple CGI scanners, and will have impact on raw mode interface tools. The server will simply deny any request that is not authenticated. In order to allow anonymous users to work with the site, it may be possible to exclude the entry point page from this scheme, and write the username and password for the rest of the site. As simple as this may sound, it has an enormous effect on simple tools. All the tools that don't support authentication will simply not work on the site. This includes Internet worms and simple CGI scanners. Raw mode interface tools will require the attacker to manually inject the HTTP authentication data with each attack attempt. When using URL interface mode tools, the attacker will have to set the HTTP authentication parameters to the values that should be used with the site.

Downside: Impact on user experience.

User Impact: Anonymous users are forced to enter a username and password upon entry to the site.

- **Customized 404 pages (with HTTP 200 status)**

Since simple CGI scanners scan hundreds of vulnerable scripts, and they consider any request with HTTP status 200 to be successful, it is possible to "drown" any true positive result they may yield, by flooding it with false positive results, that is, by responding to requests for non-existing resources with an HTTP 200 status. The attacker will get a result screen full of successes, and upon checking several of them, will notice these are actually false alarms, and thus he/she will have to abandon the particular tool he/she uses. Configuring the server to respond with a customized page returning HTTP status 200 is pretty easy.

Downside: None.

User Impact: None.



- **Enforcing Referer**

Much like the simple cookie, it is possible to enforce the validity of the Referer field (should contain at least the host name of the current site) in every user controlled script. This again poses a nuisance for raw mode interface tools, as well as using a browser for spoofing POST requests (by editing the HTML on disk and using the browser to resend requests). Browsers do not send a Referer header when the page that originated the request resides on the disk.

Downside: Having to modify all scripts. May have problems supporting Javascript links, since these are sometimes sent (by some browsers) without a Referer header.

User Impact: None.

- **Kick the client out of the application whenever an error occurs**

Regular users do not make many mistakes, at least not in the same "session". If the site has a concept of an application, and if the client is required to carry out several steps in order to obtain a session, then it may be possible to invalidate the session upon detection of an attack. While this should be quite rare for the valid client, it will happen again and again to an attacker, forcing him/her to lose precious time re-entering the application. This should work well with most kinds of tools (except for the various CGI scanners, wherein the scanner does not attack a user script).

Downside: Having to implement this mechanism in all user scripts,

User Impact: Valid users could be kicked out of the application.

- **Secure web programming**

Writing secure code is the only way to truly protect your web application from hacker attacks. The best strategy is to resort to the tried and true security techniques - never trust the client, always check client data (parameters, cookies) and make sure they are valid, legitimate, and match what you expect. In this respect, client side logic usually works contrary to security, by opening a lot of options (which are not a closed set provided by the server), so restricting client side logic to graphic effects is a good idea.

Securing your code during development counters attacks from both occasional hackers, and determined and well equipped attackers. Unlike the previously discussed techniques, which only force hackers to use advanced tools and to spend more time by "hiding" the vulnerability, writing secure code protects your site from attackers regardless of the tools being used or effort exerted, because it rids your site from the vulnerabilities which hackers exploit.



Summary

It is possible for small sites to provide simple and cheap lines of defense that will deter hackers. They serve as a first line of defense, which will keep away random and short attempts by the less trained and less deliberate hackers.

Not all techniques fit all sites. Some techniques are more intrusive, and require modification to the site content, while others require a simple configuration of the server. All the techniques discussed do not provide a substantial defense against a well-trained and/or well equipped hacker. However, very quickly these techniques will provide more value than the effort required to implement them.