

Evaluating the safety of your Web presence

Web Application Testing

By Brad C. Johnson

There is inherent risk on the Internet – the best way to protect a Web site from these risks is to identify problems proactively and develop strategies to mitigate the vulnerabilities found.

Anybody who has a Web presence understands that there is inherent risk on the Internet. The more fully featured your Web site, the more likely it is that it will come under assault. Exploiting Web applications is, not surprisingly, one of the fastest growing security exploit areas and also one of the most likely ways for outsiders to get access to sensitive data and systems.

The best way to protect your website from these risks is to identify problems proactively and develop strategies to mitigate the vulnerabilities you find. Nobody writes perfectly safe Web applications and therefore it is important to understand the types of risks that are latent in your Web applications and implement policies and procedures to effectively deal with them.

Identifying risks: different viewpoints

There are several different topics one should consider to reduce the risks of your Web presence: security requirements, coding practices, resource hardening, and remote vulnerability assessments. The way that you look for risks in each of these areas is completely different, and each one answers a different business question. This article primarily focuses on remote vulnerability assessments, but let's take a quick look at how we would review the other areas for risks and what basic question we would be trying to answer.

Have we designed a reasonably safe Web environment?

The best way to answer this question is to review your security requirements with a top-to-bottom analysis, beginning with the purpose of your Web environment, understanding the characteristics (e.g., sensitivity of data that carries regulatory requirements such as electronically protected health information – HIPAA) of the data being processed and determine if it is properly handled. This kind of analysis is normally a combination of specification reviews and interactive

discussions with the people who actually designed and built all of the parts of the Web environment. It requires direct access to proprietary documents and key staff members.

Have we built a Web application using sound coding practices?

This question is obviously answered by performing a code review. A code review is normally performed in two ways. First, you will run static code analysis tools depending on the type of source code involved. These tools (such as Jlint, Find-Bugs, RATS, BOON, JsLint, Perl::Critic, and Pixy) analyze your static code (i.e., not running) for all sorts of issues related to command injections, buffer overflows, common poor coding practices, and well-known bad security practices. The ones listed here are free or public domain, and there are other commercial tools as well. Second, you will need to perform a hands-on review of your code (or selected parts), looking for more subtle or design-oriented flaws. Tools cannot resolve differences between actual code and design requirements. You may have taken the time to define security requirements for your Web application, but that does not mean your code was written to match those requirements. A common security requirement is that data should be scrubbed (sanitized) before it is stored to prevent injection of malicious data and when it is fetched in case it was maliciously manipulated in storage. No tool is going to tell you that your specified data scrubbing was not actually coded.

Have we deployed our Web facilities on systems that are hard to break into?

At some point, your organization decided what resources were going to be used to build the Web environment. That includes operating systems, Web servers, databases, development languages, and all sorts of other tools or programs that will be used to run and administer your environment. Each of those resources has to be installed and configured to both support the functions you need and to be safe from malicious

users. Performing this kind of resource-hardening analysis requires direct access to all of the systems and programs that will be in the production environment and ensuring that they are setup to remove unnecessary features, protect sensitive information, and not allow unauthorized access. There are some operating system specific tools that one can run, but like the code review, determining if your resources are effectively hardened to minimize security risks is going to require manual inspection to look for subtle issues that the tools cannot discover.

Each of the above areas is important in its own right and you can see, even from this high level, that the methods in identifying risk in each area are inherently different; and each requires a separate, focused effort. The rest of this article focuses on the fourth important question.

Have we tried to break into our own Web application like a hacker would?

Identifying risks in this area, unfortunately, is not easy. There are no tools that one can run that will tell you what all your important problems are. From a tool perspective, this area is the least mature of the four areas we have called out for reducing Web-environment risk. There are starting-place tools you can run – *and you should run them* because a determined intruder would do the same thing. Let's take a quick look at just a few.¹

- **Nessus:** a public domain tool with over 20,000 plugins that looks for a number of host and Web server flaws
- **Retina:** a commercial tool that looks for host-oriented vulnerabilities
- **Metasploit:** a public domain “framework” tool to build and test coding exploits
- **Nikto:** a Web-scanning tool with over 3,200 test items
- **ProxyStrike:** Web-application proxy to attempt injections and perform parameter tests

The problem is that these tools are not going to find subtle issues and are certainly not going to find design flaws. For example, you may have a security requirement that no authentication information is passed in the clear or without obfuscation between the client and the server. The only way to verify that you are meeting this requirement is to manually inspect the traffic as it traverses over the Internet. It will be many years before such tools can successfully take the place of the skills of an expert tester.

Performing Web-application vulnerability testing requires a combination of a methodical and consistent way of reviewing your environment and raw expertise that requires understanding a wide variety of technologies and principles that come into play such as application design, coding practices, security practices, Web architecture, as well as network design, network technologies, and protocols.

1 See <http://sectools.org/> for a list of the top 100 Network Security Tools posted by Insecure.org.

There are several well-known security standards you can use as a framework to perform a review (audit) of your Web environment. The main ones being used today are following:

- **OWASP, Open Web Application Security Project: The “Top Ten” Project** identifies the areas where most Web application security problems are and offers guidance on how to test for them.²
- **ISO, International Standards Organization, 17799/27002:** The 17799 security standard (which was recently renamed to 27002) identifies 10 security practices areas and over 130 specific controls that should be in place.³
- **PCI DSS, Payment Card Industry Data Security Standard:** The major credit card companies including American Express, Discover Financial Services, JCB International, MasterCard and Visa have defined a security standard to ensure that organizations properly protect account data.⁴

Business focus

So far, this article has centered on technology and standards, but the fact is the most important topics in application vulnerability testing revolve around business issues. Technology for the sake of technology is unimportant. Technology exists for the stability and growth of the business and to provide functionality that supports long-term needs of the business. The purpose of Web-application testing is to answer a few important business questions. Let's take a look at the key questions and briefly describe how one might answer each question.

- **Can an unauthenticated user access data reserved for authenticated users?** Log into the Web application, map out the various places you can go and the resources you have access to and then try to access those same places and resources as an unauthenticated user.
- **Can an authenticated user see other users' data?** Log into the Web application as *User A* and do the same as mentioned above. Log into the Web application as *User B* and see if one user can see information reserved for the other: this often requires making manual changes to the requests to the server to bypass or overlay authentication information.
- **Can a user escalate his privileges?** Log into the Web application as a special user (e.g., administrator, master account) and map out how special calls are performed (e.g., delete or change accounts, change server, or application configuration settings). Log into the Web application as a “normal” user and modify the requests to the server to include the special call pa-

2 http://www.owasp.org/index.php/OWASP_Top_Ten_Project.

3 <http://www.17799.com/index.php>.

4 <https://www.pcisecuritystandards.org>.

parameters that are supposed to be reserved for special accounts.

Protecting sensitive information

It is easy to see that all of these questions are related in some way to identity. Do we know who somebody is? Do we have limits on what and when somebody can do something? Do we know what actions he has performed? Are we sure that we can control who has access to our data and under what conditions?

In security, and related to all of these questions, it is important that we have mechanisms in place to provide protection over our resources, but it is just as important to audit what actions have actually been executed. Often times, after the fact, we find that actions have been performed at times when they should not have been and by people who should not have had authority to perform them.

Every Web application should be designed to create log entries or audit events whenever suspicious requests have been made, even if the malicious request fails at getting access to important resources. Here is a small list of such triggers that should generate audit events:

- Any malformed request to the server
- Requests to access a resource without appropriate permissions
- Any input field that contains cross-site scripting (XSS), command injections, or code-like syntax
- Any request where security credentials or cookies have unexpectedly changed since logging in
- Calls to functions with the wrong number of parameters
- Calls to functions with pre-populated parameter field values that are have changed

We all know that identity theft is one of the key concerns for every business on the Internet. In many cases just knowing your identity, and what you own or do, or what you have access to, is more important than actually performing actions on someone's behalf (e.g., making an illegal credit card transaction). In any event, what is important is that you understand what identity risks exist in your Web environment and proactively deal with them. Auditing failed Web application requests is one good way to understand how people are trying to subvert your application's security measures.

Tools to break into your Web application

At a high level, Web environments are straight-forward. A Web server holds the information and applications, the browser allows you to see the content and functions the sever makes available, and the network (the Internet) carries the information back and forth between these two entities. What makes the Web environment so fundamentally different than most business applications is that everyone can "see" the code for the client-side by simply using the *View Source* function

of your browser or by looking at the network traffic. You do not have to be a network specialist to understand a lot of what you see. Okay, what will we see if you do this?

There are two main things we learn by using the browser's *View Source* function and looking at the HTML code that was sent to your system by the Web server. We can see exactly how the browser-client passes input data back and forth to the server and we start to learn what kind of coding conventions they use for parsing and displaying information and, most importantly, how they invoke functions on the server. Let's delve a little more into the latter issue.

Some applications are built to have a common program that is called (e.g., *Main.php*) with additional information passed in to identify a specific function to perform (e.g., update contact information, change a password, make a transaction). Other applications have separate programs to call for each function (e.g., *ChangePassword.jsp*, *DebitAccount.jsp*, *CreditAccount.jsp*). The first step in finding ways to bypass or manipulate important authentication or authorization calls is to understand how these calls to the server are constructed and how data is passed into those functions. A lot of this can be discerned by carefully looking at the source code that sits on your system in the browser.

What you need to do this type of Web application analysis, or essentially to try and break into your own Web application, is built into every single computer: that is, it's the browser itself and other browser supporting programs and plugins. Therefore, the hard part is not getting access to tools (like it can be for other types of Internet or application analysis), but rather taking the time to understand what is going on and figuring out how to exploit it.

One specialized tool that is helpful for performing this kind of analysis is called a Web-application proxy server. This is normally not installed by default on your computer. It is a piece of software that you run on your computer that "sits" in between the Web server and the browser. It allows you to look and change data before it gets to your browser and then the opposite, to look and change data before it goes to the server. While it takes some degree of technical knowledge to understand and expertly use this kind of tool, it is also not rocket science and these tools are free.

There are actually a variety of proxy servers available including a caching server, a reverse proxy server, a content-filtering proxy, and an anonymizing proxy to name a few. Wikipedia has a good section for understanding the purpose and value of these different types of proxy servers.⁵

Two of the more commonly used Web application proxies are Paros⁶ (Figure 1) and Burp.⁷ Both are used for the same thing: to intercept all HTTPs data between the server and the client, including cookies and form data which can be viewed or modified.

5 http://en.wikipedia.org/wiki/Proxy_server#Web_proxy.

6 <http://www.parosproxy.org/index.shtml>.

7 <http://portswigger.net/proxy>.

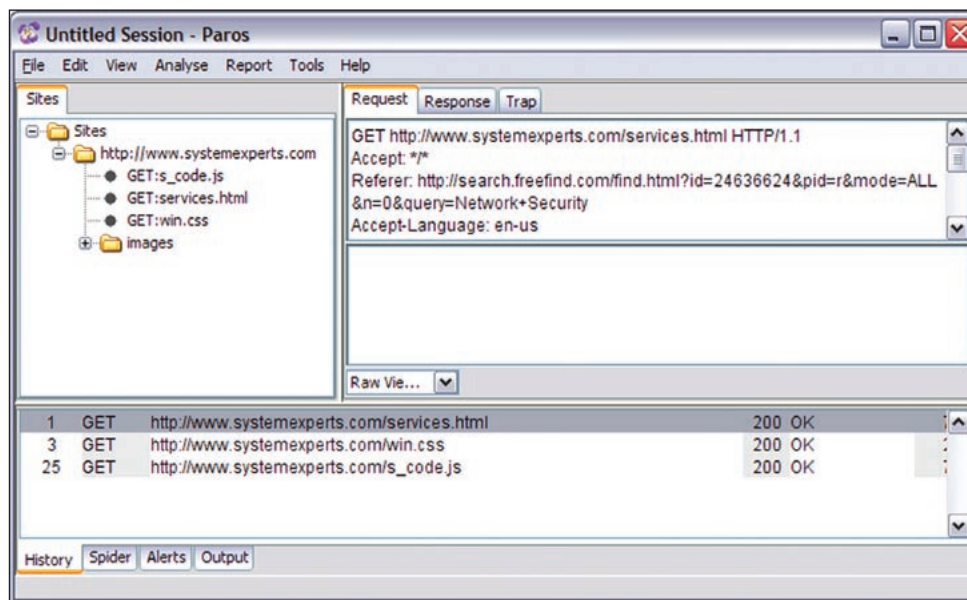


Figure 1 – Screen shot of the Paros Web application proxy viewing HTTP traffic

Using a Web-application proxy we can make both coarse- and fine-grained changes to the cookies, parameters, and function calls themselves in an attempt to subvert or bypass authentication or authorization information to get access to sensitive information or functions.

Common Web application design problems

Despite the fact that every Web site is different, there are a number of problems that many sites share that make them vulnerable to attack. First, the people who maintain and deploy your Web server are separate from the people who write the Web applications who themselves are different than the people who maintain and configure the supporting hosts and most likely, none of these people are from the business side of the organization that defined the need for the Web environment in the first place. This compartmentalization of responsibilities often leads to mismatched expectations, false assumptions, and overlooked requirements that accidentally create security problems.

Early on we described four topics (security requirements, coding practices, resource hardening, and remote vulnerability assessments) that need to be analyzed in order to reduce the risk of our Web environment. If you take the time to assess all of these areas, you will go a long way in getting over some of these compartmentalization problems. If you include staff members from each of these areas in all four of the assessments, things will be even better still because you will reveal a lot of cross-organizational knowledge and expertise that would not otherwise come out.

Second, unless you have a static, brochure-type Web site, you are going to need to keep track of “state” information: who is logged in, what privileges does he have, what actions has he taken, and what transaction information does his actions require? State information can be stored on the server, in back-end application data stores, or on the client. Most Web

applications use a combination of these storage mechanisms, but most do not use them wisely, protect them adequately, or ensure that the state information has not changed unexpectedly. One of the most common techniques for a determined intruder is to change the state information to masquerade as another user or attempt to perform actions that were reserved for another user. This is exactly the kind of action you would do with the Web-application proxy that was mentioned earlier.

Third, and finally, most applications are built and tested with a primary focus on functionality, not on security. The

result of this typical development approach is that the Web application may perform well as long as all input and output data is normal, but they are not robust or prepared to handle unexpected data or situations. Here are some examples of unexpected data:

- Cross-site scripting code that has been inserted into a form field or a search field
- Alphabetic input fields that have been changed into numeric data and vice versa
- Parameters that are missing assigned values
- Parameter values that contain assigned numbers (e.g., to reference a file) that have been incremented or decremented
- HREF HTML tags that have been modified to use either local file names or other resources, e.g., change `href="docs/file.html"` to `href="c:windows/control.ini"`
- Using a Web-application proxy to remove all JavaScript functions that screen field values and then insert invalid data and submit it to the server

Determined intruders purposely create these odd error conditions and attempt to learn how the application handles different situations to find instances where the application will perform actions it should not. Sometimes, for example, the Web server will reveal internal error codes, explicit local file names, or give internal Web server information. In the case of sequentially numbered parameters, sometimes the Web application will not perform appropriate authorization checks and just assume that the client would not be asking for the object (e.g., a file) unless it had permission to do so and return a file that was reserved for another user.

Final word

Today, almost every organization provides some information or service over the Internet. The Internet is an inherently hostile environment as you have no control over who is using it or what their intentions are. The prudent thing to do is to understand what risks your Web presence introduces and ensure that your web servers are properly hardened, your web applications are properly designed and implemented, and that you have compensating controls in place to address any residual risks.

In this article we mentioned four different areas that should be assessed to reduce risk and we spent most of our time talking about one area in particular: Have we tried to break into our own Web application like a hacker would?

Web application testing should always be performed before any new application is released or if it has undergone major changes and this testing should also be done periodically to ensure that new risks have not crept into the production environment.

Unfortunately, there are no tools available today, and will not be for quite some time, that can analyze your Web application and identify most of the important risks. There are some tools available like Nessus, Nikto, and MetaSploit that can help us identify some Web server or host specific problems, but performing the kind of analysis that will yield more subtle problems and focus on authentication and authorization issues must be done by hand and requires a unique set of training and expertise.

About the Author

Brad Johnson is Vice President of System-Experts Corporation. He is a well-known authority in the field of distributed systems and speaks on the subjects of security standards, penetration testing, middleware, and practical intrusion detection. He has participated in seminal industry initiatives including the Open Software Foundation (OSF), X/Open, and the IETF, and has published extensively about open systems. Brad holds a BA in Computer Science from Rutgers University and a MS in Applied Management from Lesley University. He may be reached at brad.johnson@systemexperts.com.

