



SANS IT Audit
with David Hoelzer

IT Audit: Security Beyond the Checklist

This paper is from the SANS IT Audit site. Reposting is not permitted without express written permission.

Copyright SANS Institute
Author Retains Full Rights

Interested in learning more?

Check out the list of upcoming events offering
"IT Security Audit and Control Essentials (Audit 410)"
at <http://it-audit.sans.org><http://it-audit.sans.org/events/>

Audit of a Small LAMP (Linux, Apache, MySQL, and PHP)
Web Application

SANS GIAC Systems and Network Auditor (GSNA) practical
Version 3.1 — Option One

Herschel Gelman

May 2, 2004

© SANS Institute 2004, Author retains full rights.

ABSTRACT

This paper contains an audit of a web application available on the Internet that is run on PHP, MySQL, Apache, and Linux—a combination commonly known as a LAMP system. As a web hosting company hosts the application, the scope of the audit encompasses only those components available to the developer: the PHP source code and any site configuration options available to the developer. In the first section, the paper will cover initial research into the system, risks to the system, and current practices regarding web application security. Part two contains the audit checklist, with testing procedures and compliance criteria. Part three gives the results of the audit. Part four contains the audit report, listing the findings and recommendations.

© SANS Institute 2004, Author retains full rights.

TABLE OF CONTENTS

ABSTRACT.....	2
TABLE OF CONTENTS.....	3
1 Research in Audit, Measurement Practice, and Control	6
1.1 SYSTEM IDENTIFICATION.....	6
1.2 MOST SIGNIFICANT RISKS TO THE SYSTEM	7
1.2.1 Threats to the System	7
1.2.2 Information Assets Affected by Audited Device.....	8
1.2.3 Major Vulnerabilities of the Web Application	9
1.3 CURRENT STATE OF PRACTICE.....	11
1.3.1 Articles, Papers, and Mailing Lists	11
1.3.2 Tools.....	13
2 Audit Checklist.....	16
2.1 CHECK FOR HIDDEN COMMENTS IN HTML	16
2.2 SESSION HIJACKING VIA COOKIE MANIPULATION.....	17
2.3 SQL INJECTION.....	18
2.4 TEST FOR ADEQUATE SAFEGUARDS AGAINST BANDWIDTH THEFT	19
2.5 SCAN FOR SAMPLE FILES OR SCRIPTS.....	21
2.6 TEST BACKUP PROCEDURES.....	23
2.7 UNSAFE HIDDEN FORM ELEMENTS	24
2.8 ENSURE DIRECTORY BROWSING SETTINGS ARE CORRECT.....	26
2.9 ATTEMPT TO BRUTE FORCE ADMINISTRATIVE ACCOUNT.....	27
2.10 VERIFY SECURITY OF ANY CLIENT-SIDE JAVASCRIPT	29
3 Audit Testing, Evidence, and Findings	32
3.1 CHECK FOR HIDDEN COMMENTS IN HTML	32
3.1.1 Evidence.....	32

3.1.2 Findings.....	33
3.2 SESSION HIJACKING VIA COOKIE MANIPULATION.....	33
3.2.1 Evidence.....	33
3.2.2 Findings.....	34
3.3 SQL INJECTION.....	34
3.3.1 Evidence.....	34
3.3.2 Findings.....	36
3.4 TEST FOR ADEQUATE SAFEGUARDS AGAINST BANDWIDTH THEFT	37
3.4.1 Evidence.....	37
3.4.2 Findings.....	38
3.5 SCAN FOR SAMPLE FILES OR SCRIPTS.....	38
3.5.1 Evidence.....	38
3.5.2 Findings.....	39
3.6 TEST BACKUP PROCEDURES	39
3.6.1 Evidence.....	39
3.6.2 Findings.....	40
3.7 UNSAFE HIDDEN FORM ELEMENTS.....	41
3.7.1 Evidence.....	41
3.7.2 Findings.....	42
3.8 ENSURE DIRECTORY BROWSING SETTINGS ARE CORRECT.....	42
3.8.1 Evidence.....	42
3.8.2 Findings.....	43
3.9 ATTEMPT TO BRUTE FORCE ADMINISTRATIVE ACCOUNT.....	43
3.9.1 Evidence.....	44
3.9.2 Findings.....	44
3.10 VERIFY SECURITY OF ANY CLIENT-SIDE JAVASCRIPT	45
3.10.1 Evidence.....	45
3.10.2 Findings.....	46
4 Audit Report.....	47
4.1 EXECUTIVE SUMMARY	47
4.2 AUDIT FINDINGS	47
4.2.1 Check For Hidden Comments in HTML	47
4.2.2 Session Hijacking Via Cookie Manipulation.....	48
4.2.3 SQL Injection	48
4.2.4 Test for Adequate Safeguards Against Bandwidth Theft.....	48
4.2.5 Scan for Sample Files or Scripts	48

4.2.6	Test Backup Procedures	49
4.2.7	Unsafe Hidden Form Elements	49
4.2.8	Ensure Directory Browsing Settings Are Correct.....	49
4.2.9	Attempt to Brute Force Administrative Account.....	49
4.2.10	Verify Security of any Client-Side Javascript.....	50
4.3	AUDIT RECOMMENDATIONS	51
4.3.1	Highly Recommended Actions	51
4.3.1.1	Protect Against Bandwidth Theft.....	51
4.3.1.1.1	Description.....	51
4.3.1.1.2	Costs	51
4.3.1.1.3	Compensating Controls.....	52
4.3.2	Lower Priority Recommendations	52
4.3.2.1	SQL Injection	52
4.3.2.2	Hidden form elements.....	52
4.3.2.3	Future password safety	52
	References.....	54

© SANS Institute 2004, Author retains full rights.

1 Research in Audit, Measurement Practice, and Control

1.1 System Identification

The system to be audited is a database-driven web application that is available on the Internet. It allows the public to create free accounts, search the review database, submit new votes and reviews on items in the database, and add new items to the database. It also has administrative functionality, so those users who are granted the appropriate rights can perform administrative tasks through the same web interface.

For the purposes of this document, we will refer to the application as AuditApp.

The application design and development was a one-man effort, and therefore only this single developer has reviewed the system. This is also this developer's first time using PHP and SQL, which increases the likelihood that potential security holes have made their way into the code. The web application was made available to the public via the Internet with no comprehensive security review. The goal of this audit is to provide an independent security evaluation of the web application.

The application is powered by what is known as a "LAMP" system. This acronym refers to the open source combination of Linux as the operating system, Apache as the web server, MySQL as the backend database server, and PHP, Perl, or Python as the scripting language. In this specific case, the system is running Debian Linux 3.0r2, Apache 1.3.29, MySQL 4.0.17, and PHP 4.2.3. A large web hosting company runs the web and database servers.

The scope of this audit encompasses the web application level of this system: the PHP code itself. It also covers the customer's workflow and interactions with the web hosting server, as potential vulnerabilities could be introduced in that way as well. In addition, it covers any configuration options for the web site that are available to the developer, but not options that are set by the web hosting company that the customer has no control over. The MySQL database, Apache server, and the operating system itself are outside the scope of this audit. The customer has no control over any of these components, as the web hosting company manages these portions of the system. Ideally, these aspects should also be examined in a separate audit.

However, any obvious security issues with the web host's configurations that are discovered during the course of the audit will be reported, as the choice of a web hosting company is still within the developer's control. If it turns out that this web

host uses poor security practices, the developer has the option of switching to a company with tighter security.

1.2 Most Significant Risks to the System

1.2.1 Threats to the System

Because of the single owner and developer of this web site, intentional internal threats are not an issue; there are no disgruntled employees that may be attempting to damage the site. The system administrators at the web hosting company are considered external in this case, because our audit scope is focusing on the web application code itself.

The data stored on the web server is all drawn from a combination of publicly available sources and input from visitors to the web site. The only data stored on the site that may be of possible interest to an outside party would be the collection of e-mail address in the database, as all users who sign up on the site are required to include a valid e-mail address. Those addresses could be sold to spammers, and therefore might have some small value to an intruder.

The following table details some of the possible threats to this system:

Threat	Effect
Accidental programming error by application developer	Web site visitors receive error messages or see improper site operation. Could divulge sensitive information (database table names, directory paths, usernames). Loss of confidence in site by the public, leading to possible loss in revenue. Could also give administrative access to the web application to all visitors.
Exploit against programming error in web application code	Attacker could gain access to user-level account on web host, full access to customer's database on database server. That gives full access to all e-mail addresses stored in the database, plus access to modify or delete any information in the database. The net effect is a loss of privacy for users of the site, and possible loss in revenue due to public's loss in confidence of the site, and/or due to loss of data.
Loss of data by web hosting company—this could be due to an attack on their systems, environmental threats, etc.	If the web hosting company lost all the data for the web site—application code and database contents—then the customer would need to fall back to his own backups. If there were no backups, or if the backups were not functional, the developer would need to build the web site from scratch, including all code and data, and all users would need to register again. This would be a cata

Threat	Effect
	<p>strophic loss, and it is possible that the site simply would not recover from this, due to the large amount of data that would need to be re-entered. It would also be impossible to retrieve all user-submitted reviews in this case.</p>

1.2.2 Information Assets Affected by Audited Device

As this one-person “organization” exists entirely to create and support this web site, the audited web application directly affects practically every aspect of the organization. This includes all data owned by the organization, and all services provided by the organization. E-mail is the only function which is used by the site owner which is unaffected by changes to this application.

Information Asset	Description
<p>Source code to the web application</p>	<p>The web application represents a significant amount of development work, and may be used as a basis for future commercial work by the developer.</p> <p>Disclosure of the code to the public could also compromise the security of the site, as any security holes in the code would become public knowledge. If the code is well-written, though, this would not be a concern.</p>
<p>Public data stored in the database</p>	<p>Most of the information stored in the database is publicly available through the web application, and therefore confidentiality is not a requirement.</p> <p>However, AuditApp would be useless without this data, and therefore its availability is critical to the successful functioning of the web site.</p> <p>The data integrity is also important, arguably as important as availability. The reason that the public visits the web site is to access this data; if the data they were viewing on the web site was inaccurate, and they realized this, they would be less likely to return in the future.</p>
<p>Private data stored in the database</p>	<p>While most of the database stores publicly available information, some data in it is not accessible to the public. This data includes:</p> <ul style="list-style-type: none"> • Real names of users who have accounts on the site (only usernames are visible to other users, which may or may not have anything to do with the users’ real names.) • MD5 hashes of all users’ passwords • All users’ e-mail addresses

Information Asset	Description
	<ul style="list-style-type: none"> • Date that each user registered for their account • Date and time of each users' last login to the web application • Access levels of each user on the system: most users have the most basic access levels, but some have additional rights to perform administrative tasks on the site.
Bandwidth	The web host limits the bandwidth available to the site. Usage of bandwidth over that limit—whether through legitimate web traffic or “theft” of bandwidth by another site—would result in additional costs or the temporary shutdown of the web site.
Service provided by web site	The web site provides a vast amount of data to the public. This information consists of information available at other web sites in other formats, as well as large amounts of unique content contributed by visitors to the site as well as the owner of the site. The value of the web site mostly relies on this data plus the code that powers the application.

1.2.3 Major Vulnerabilities of the Web Application

The major potential vulnerabilities in this web application are listed below. For each vulnerability we will list the likelihood of it being exploited on a scale of one through five, with one being low and five being high. In addition, the impact of a successful exploitation of the vulnerability on the web site is also listed using the same rating scale.

Vulnerability	Likelihood (exposure)	Impact
1. Programming error in application leaves site vulnerable for an attacker to get administrative access to the web site, through the application's own web interface to the public.	3	4
2. Malicious attacker exploits programming error in application to get full access to the site's database.	3	4
3. Catastrophic loss of data at web hosting company. This could be due to environmental causes, malicious attackers, hardware failures, etc. As the customer has no control over any of these, they are all treated together from this audit's point of view.	1	5
4. Cross-site scripting attack. This could yield ad	3	4

Vulnerability	Likelihood (exposure)	Impact
ministrative rights to the web application		
5. SQL injection attack. This vulnerability could also yield administrative rights to the application	3	4
6. Non-critical programming error in the PHP or SQL code that leads to loss of some or all functionality of the site (e.g., error in PHP cause some pages to give errors to the user)	3	2
7. Session hijacking by spoofing valid session identifier. This could give administrative access to the application if an administrative account's session is hijacked	2	4
8. Denial of service attack	3	3
9. Leak of hidden information through HTML comments	2	3
10. Session hijacking by modifying client's stored cookie. This could give administrative access to the application if an administrative account's session is hijacked	3	4
11. Password to an administrative account is guessed or brute-forced	2	4
12. Sample files, scripts, or applications from the web server or other installed software are left active on the web site	3	4
13. Modification of hidden form fields allows unexpected behavior	4	3
14. Leak of hidden information through client-side Javascript	2	3
15. Weak protection employed through client-side Javascript	3	4
16. Usernames could be collected through login error message that reveal that a valid username was attempted (i.e., if the error message the web site shows on a failed login attempt changes depending on whether the username was valid or not)	4	1
17. Theft of bandwidth caused by another web site linking directly to images or other media on this web site. This could use up all of the bandwidth allocated to the web site by the web host, causing the site to disabled. In effect, this would create a denial of	3	2

Vulnerability	Likelihood (exposure)	Impact
service against the web site		
18. Web server automatically indexes site's directories, and makes them available if users manipulate the URL in their browser. For example, if http://www.site.com/includes/main.css is referenced in the HTML code for the site, a user who points their browser to http://www.site.com/includes would see the full contents of that directory. This can expose files that the user should be able to view.	2	3

1.3 Current State of Practice

1.3.1 Articles, Papers, and Mailing Lists

Given the sheer size of the World Wide Web today, and the large number of companies that rely on their public web site as their primary source of revenue, it's somewhat surprising that so little attention has been focused on web application security in the past. In recent years, however, this situation has been improving. Today there are many articles, papers, and discussions available on the Internet devoted to the specific issues of web application security.

SecurityFocus¹ hosts one such discussion: a mailing list devoted to web application security named, appropriately enough, the Web Application Security Mailing List². The list archives are available online³ and extend from January 2001 to the present. In these archives are hundreds of discussions of web application security issues. Some of these are specific to a certain web scripting language or operating system, but many are general enough to be applicable to all web applications.

Another collection of information is available at The Open Web Application Security Project (OWASP)⁴. This site has news and columns on web application security, as well as auditing tools. From my assessment of the site, the most useful item was the OWASP Guide to Building Secure Web Applications and Web

¹ <http://www.securityfocus.com>

² http://www.securityfocus.com/popups/forums/web_application_security/intro.shtml

³ <http://www.securityfocus.com/archive/107>

⁴ <http://www.owasp.org>

Services⁵, which is a document that attempts to cover every aspect of web application security. This site also promises a future resource called the OWASP Testing Guide⁶, the goal of which is to “[document] strategies and techniques to test web applications for security vulnerabilities.” As of April 2004 the testing guide is not yet available.

The SANS Reading Room also has several papers which are of use to someone auditing the security of a web application:

- “Securing e-Commerce Web Sites” by Ariel Pisetsky⁷. This paper focuses more on the server end than the web application end, but it does give a summary of some of the types of attacks a web site may be expected to cope with. It also discusses the pros and cons of several network configurations.
- “Web Application Security — Layers of Protection” by William Fredholm⁸. This paper provides a good overview of web application security. It also refers to OWASP as a valuable resource, and then covers the security that different layers of the web application can provide. It also goes into the process of testing the security.
- “Cross-Sight [sic] Scripting Vulnerabilities” by Mark Shiarla⁹. This provides a fairly complete discussion of cross-site scripting, including examples of attacks and methods of protecting a web site or web application from these attacks.

While on the subject of cross-site scripting, CERT has a very good document¹⁰ discussing the issue. Their focus is more at the end user than the auditor or web developer, but it is still a very good description of the issue.

Another common issue in web applications is SQL injection attacks. SitePoint¹¹ has an excellent article from 2002 in their archives called “SQL Injection Attacks

⁵ <http://www.owasp.org/documentation/guide>

⁶ <http://www.owasp.org/documentation/testing>

⁷ <http://www.sans.org/rr/papers/index.php?id=303>

⁸ <http://www.sans.org/rr/papers/index.php?id=965>

⁹ <http://www.sans.org/rr/papers/index.php?id=478>

¹⁰ http://www.cert.org/archive/pdf/cross_site_scripting.pdf

¹¹ <http://www.sitepoint.com>

— Are You Safe?”¹² by Mitchell Harper. This article is aimed at the web application developer, and gives thorough examples of what SQL injection is as well as how to protect against it.

A second good discussion of SQL injection attacks is “SQL Injection Walk-through”¹³ at SecuriTeam.com. This page focuses on exploiting a web site using SQL injection, which makes it a good resource for auditors and penetration testers. It also includes a number of links to other Web resources on this subject. It provides very little coverage of preventing vulnerabilities in web applications, unless the exploitation examples can assist a web developer in understanding how to properly secure his/her application.

Gunter Ollmann published a paper entitled “Application Assessment Questioning: What should a consultant be looking for when conducting an application assessment?”¹⁴. This paper gives a very comprehensive checklist of questions that can be used by an auditor conducting an assessment of any application, web-based or not.

1.3.2 Tools

There are many free tools available on the Internet that can assist in the process of auditing a web application:

Nessus¹⁵, the popular open source security scanner, has a large number of plug-ins that scan for web application vulnerabilities. Most of these are targeted at specific vulnerabilities in specific web applications, but there are some general tests that can be of value to someone auditing a custom web application. In addition, since the program and all the scanning plug-ins are open source, we can view the source code that is doing these tests to look for similar potential problems in our audits. For example, if one had been unable to find a decent online resource on cross-site scripting, I counted 64 plug-ins that test for cross-site scripting issues in various web applications. Looking at the source code to those would give invaluable insight into the process of actually exploiting some of those vulnerabilities, which we could then attempt to adapt to the web application we are auditing. In addition, if the scope of an audit encompassed more than this one does—for example, if the auditor was assessing the security of the operating

¹² <http://www.sitepoint.com/article/794>

¹³ <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

¹⁴ <http://www.technicalinfo.net/papers/AssessmentQuestions.html>

¹⁵ <http://nessus.org>

system being used as well—Nessus has a large number of tests that would assist in that task.

Another tool that I find extremely useful is a bookmarklet that will show all hidden form elements on a web page¹⁶, and allow you to modify any of them directly within that page. If you drag that bookmarklet to your personal toolbar in Mozilla—or other similar location in other browsers—you can then click on it while viewing any page, and instantly have editable access to all hidden form elements. As insecure usage of hidden form elements is very common on web sites, there are many instances when this tool will help with auditing efforts.

An additional useful tool that OWASP recently made available is WebScarab¹⁷. This tool combines a number of different functions that are useful to a web application auditor:

- A local interception proxy server that allows one to capture all requests sent through it, and modify them before passing them along. These modifications are scriptable.
- A spider function to traverse all links on the site.
- A visual graph of session IDs, to determine if the session IDs sent by the application are sufficiently unique and random.
- A quick display of which pages on the site contain Javascript, which contain HTML comments, and which pages set cookies.

Brutus¹⁸ is another valuable tool for web application testing. As the name implies, it performs brute-force username/password guessing against web sites.

To use Brutus to maximum effect, one needs sufficiently large word lists. “Kevin’s Word List Page”¹⁹ is an excellent collection of dictionary files plus links to other word lists.

Lilith²⁰ is a Perl script that will attempt to automatically spider a site and check for insecurities in form elements, by passing special characters to the application.

¹⁶ http://www.squarefree.com/bookmarklets/forms.html#show_hidden

¹⁷ <http://www.owasp.org/development/webscarab>

¹⁸ <http://www.hoobie.net/brutus/>

¹⁹ <http://wordlist.sourceforge.net/>

However, in testing it for potential use in this audit, I found that the current version gave far too many false positives for it to be helpful. In a test run, it essentially listed every form element in the test page as vulnerable to SQL injection, when a code inspection showed that this was not the case.

© SANS Institute 2004, Author retains full rights.

²⁰ <http://users.pandora.be/0xfffffice/scanit/tools/lilith/>

2 Audit Checklist

The following is the checklist for this audit:

2.1 Check For Hidden Comments in HTML

Checklist Item Number: 001

Checklist Item Name: Check For Hidden Comments in HTML

Reference: OWASP Guide to Building Secure Web Applications and Web Services, pages 50-51.

Risk: This test addresses vulnerability number nine in section 1.2.3 of this document. HTML comments are not shown to a visitor to the web site, but are available in the actual HTML code. Any visitor can therefore view HTML comments simply by using the “view source” function in their web browser, which means that sensitive information should never be placed in HTML comments. The degree of exposure is considered to be fairly low, since with a single developer, it is easier to keep track of what comments are placed in the HTML. The severity of loss is medium: the comments could contain anything. In the worst case, they could mention incomplete files on the system which do not properly restrict access, or files that contain back doors to the application. This has the potential to give an outside user full access to the site’s database and all of their content.

The net risk is therefore **medium-low** (2.5).

Testing Procedure/Compliance Criteria: Use the WebScarab tool (see section 1.3.2). Click on the “Spider” tab, and select “Fetch Recursively.” Next, click on the “Manual Request” tab, and enter application’s URL into the URL field at the top of the “Request” section. Click on “Fetch Response” at the bottom of that screen. Then, go back to the “Spider” tab, click on the top level of the web site, and click the “Fetch Tree” button. One can then go to the “Summary” tab and view the details for every file on the web server. If there is a check mark in the “comments” column next to a file, then there are HTML comments in that file. Right-click on each file with comments and select “show comments.”

If the comments that WebScarab shows are innocuous, such as comments separating sections of the web page in the code (e.g., “Ad banner code:” or “Menu bar”), or other comments that do not leak any potentially sensitive information,

then the site passes this test. Items which would cause a failure on this test include the following: filenames on the web site which are not normally accessible (i.e., do not show up in the WebScarab spider results); usernames or passwords; comments that reveal holes in the site's code (e.g., "Fix this: it allows anyone to delete the site by clicking on this link"). Any other comments which seem to divulge too much information would also cause the site to fail this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.2 Session Hijacking Via Cookie Manipulation

Checklist Item Number: 002

Checklist Item Name: Session Hijacking Via Cookie Manipulation

Reference: OWASP Guide to Building Secure Web Applications and Web Services, chapter 7.

Risk: This test is against vulnerability number ten in section 1.2.3 of this document. The exposure is considered to be medium; since this application was developed by a novice web developer, some basic mistakes such as weak session management code may have been made. The severity of impact is fairly high, as session hijacking could allow an attacker to impersonate a legitimate administrator of the site, giving them full access to read, change, or delete all content in the database.

The net risk is therefore **medium-high** (3.5).

Testing Procedure/Compliance Criteria: Log into the web site. After logging in, view the stored cookies in your web browser for the site. In Mozilla, this can be done by going to Edit > Preferences > Privacy & Security > Cookies, and then clicking on "Manage Stored Cookies." Alternatively, an intercepting proxy tool such as Achilles or WebScarab will also show cookies that have passed through it.

If the cookie name is "PHPSESSID" and the cookie value is a 32-byte hex string, then the site is using PHP's build-in session management functions, which use MD5 hashes and are not vulnerable to simple client-side manipulation. The site would therefore pass this test. If the site is not using PHP's session functions, check for any fields in the cookie which appear to be changeable. These vulnerabilities include setting the user's ID number in the cookie, as that could be

changed to be any user ID. Another possibility is storing the user's access level in the cookie. Any weak use of cookies for session management such as these examples are a strong indication that session hijacking is a possibility, and the site would fail this test. If there are no such fields, the site passes this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.3 SQL Injection

Checklist Item Number: 003

Checklist Item Name: SQL Injection

Reference: OWASP Guide to Building Secure Web Applications and Web Services, pages 36-39.

"SQL Injection Attacks — Are You Safe?" by Mitchell Harper.

Risk: This test addresses vulnerability number five in section 1.2.3 of this document. The degree of exposure is considered to be medium, since there are a large number of SQL queries in the application, many of which use data provided by the user, and the application developer was a novice at web application development. The severity of loss is fairly high, as a successful SQL injection attack would allow the attacker to run any SQL queries he/she wished against the site's database. This would let them read, change, or delete any or all of the site's data, including user e-mail addresses.

The net risk is therefore **medium-high** (3.5).

Testing Procedure/Compliance Criteria: To fully test this item, we will need to examine the source code to the site. First, search the code for all database queries. In a PHP and MySQL application, these usually performed using the `mysql_query` function, and so the auditor should search for that. However, the site may have written its own query function that performs other work before calling the built-in query function. Because of this, the code must be examined first in order to find out what function or functions are used for database requests. Once this has been determined, the auditor can then search the code for all instances of these requests.

Next, for each SQL query found, look at all the variables that are used within the query. For each of those variables, check back through the code to see where

they are assigned. If any of them are taken from any item sent by the browser—appended to the URL in a GET request, in a POST request, or from a cookie—then there should be code in the application to sanitize the data. The PHP “stripslashes” function is one possible way to sanitize the data and make it safe to use in a SQL query. If all user data is sanitized, then the application passes this test.

If not all user input is being validated and/or sanitized, then one needs to check the setting of “magic_quotes_gpc” on the server. This setting forces PHP to automatically escape quotes and null characters in all user input, leaving it safe to use in a SQL query. The “gpc” section of the function name refers to the three types of browser-supplied input that are sanitized: GET requests, POST requests, and Cookies. To check if this is enabled, create a new file on the web site called phpinfo.php. The contents of the file should be the following:

```
<?php phpinfo(); ?>
```

View this file in a web browser by going to <http://web.site.address/phpinfo.php>. In the “PHP Core” subsection of the “Configuration” section of the page, there will be a line labeled “magic_quotes_gpc.” Ensure that it is listed as “on”. If magic_quotes_gpc is enabled, then the application passes this test.

If magic_quotes_gpc is not enabled, and not all user input is sanitized by the application code, then this test is failed.

Additional testing can be performed to confirm that unsafe variables are exploitable, or that the code successfully sanitizes the user’s input. It would be impossible to detail how to exploit potential SQL injection vulnerabilities in this document, as the exact method needs to be based on the specific details of the page in question. For full details on exploiting a potential SQL injection vulnerability, refer to one of the detailed guides to it listed in the references section (section 1.3.1 of this document).

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.4 Test for Adequate Safeguards Against Bandwidth Theft

Checklist Item Number: 004

Checklist Item Name: Test for Adequate Safeguards Against Bandwidth Theft

Reference: <http://wordworx.com> and
<http://www.thesitewizard.com/archive/bandwidththeft.shtml>

Risk: This test addresses vulnerability number 17 in section 1.2.3 of this document. The degree of exposure is considered to be medium, since on the one hand this web site is very small at the moment, and therefore the likelihood of anyone using images hosted on it is low. In addition, the vast majority of the images hosted on the site are available on many other web sites. On the other hand, some images stored on the site are unique to the site. In addition, bandwidth theft is trivial to perform, and is often done accidentally by someone who did not fully understand the consequences. As a result, the likelihood is rated as medium. The severity of this vulnerability is fairly low. The worst case scenario is that sufficient bandwidth is consumed through linking directly to the site's images that the web hosting company disables the site until they are paid for the bandwidth used. Given the size of images on the site, which average 5 kilobytes each, it is unlikely for this to be prohibitively expensive. Therefore, the main consequence is a denial of service against the web site, as the web hosting company's current procedures would take the site offline until they have received payment for the additional bandwidth.

The net risk is therefore **medium-low** (2.5).

Testing Procedure/Compliance Criteria: The easiest way to test this element is to create a web page which links to an image hosted on the web site that is being tested. However, this file must be hosted on a web server for the test to be accurate. My initial testing plan was to create an HTML file on the auditor's computer that referenced an image on the web site. However, preliminary testing showed that the web browser did not pass a referrer field to the web site, meaning that safeguards against bandwidth theft would not be effective.

Therefore, as an alternative to creating this test file, I recommend performing the request manually. To do so, telnet to the web site on port 80. Send the following lines:

```
GET /image.gif HTTP/1.1
Host: web.site.name
Referer: www.someothersite.com
```

"image.gif" should be replaced with the location of an actual image on AuditApp. Also, note that the HTTP specification calls for "referrer" to be misspelled, as shown in the example. If there are images stored in more than one directory, this test should be performed several times, testing at least one image in each directory. The reason for this is that some of the methods used to protect against direct image linking can be applied on a per-directory basis. As a result, some directories on the site could be protected while others are not.

If the web server returns an image from any tested directory—it will appear as lots of garbage in the telnet window, being a binary file—the site fails this test. If an error page or no response is received, the site passes.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.5 Scan for Sample Files or Scripts

Checklist Item Number: 005

Checklist Item Name: Scan for Sample Files or Scripts

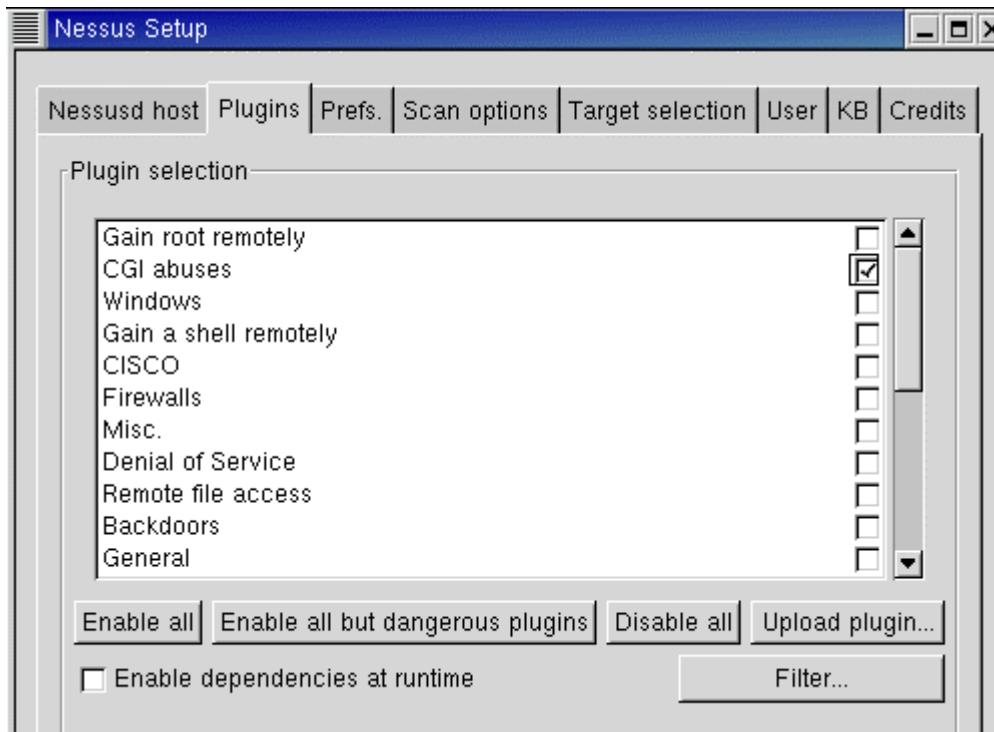
Reference: OWASP Guide to Building Secure Web Applications and Web Services, page 50, “System Configuration” section.

http://www.uniras.gov.uk/l1/l2/l3/tech_reports/niscctechnicalnote0603.htm — National Infrastructure Security Co-ordination Centre (NISCC) Technical Note 06/03: Guidance on Securing Web Sites

Risk: This test addresses vulnerability number 12 in section 1.2.3 of this document. The exposure for this vulnerability is rated as medium. While most administrators and webmasters these days know that leaving samples files and scripts on the server is a security risk, there are still some who do. In addition, there are a large number of automated tools that are constantly scanning for these scripts in an attempt to exploit them. Because of this, there is a high risk of compromise if any of these files remain installed on a web server. The severity of this vulnerability is rated as medium-high. A successful exploit against one of these scripts has the potential to give an attacker full access to the web site.

The net risk is therefore **medium-high** (3.5).

Testing Procedure/Compliance Criteria: Use Nessus to scan the site. First, from a command prompt, update the collection of installed Nessus plug-ins to the most recent set by typing `nessus-update-plugins`. Next, launch Nessus and log in. In the “Plugins” tab, select only the “CGI abuses” category of plug-ins. While the other categories contain very useful scans, they are outside of the scope of this audit. For an audit with a wider scope, one would most likely want to enable almost all of these; the “dangerous plug-ins” option should obviously be used with extreme caution if production servers are being scanned, as these tests have the capacity to crash a server if they are successful.



Next, go to the “Target Selection” tab, and enter the IP address of the web site. Then, click the “Start the Scan” button to begin testing.

One should always save a copy of the scan results when the scan is completed, for inclusion in the audit report. Analyze these results, keeping an eye out for items that Nessus categorized as a security hole, with a severity of high, serious, or medium severity. Based on my personal experience, Nessus—as with many vulnerability scanners—often gives false positives. If possible, verify any findings that Nessus gives. For example, if Nessus reports a security vulnerability because the site is running a version of the Foobar application older than version 1.5, query that application directly to determine what version it is. If there is any doubt or confusion as to whether a finding is a false positive or not, one should attempt to find further information on exploiting the vulnerability. In most cases, web server vulnerabilities such as these can be triggered simply by entering a specially crafted URL into any web browser. Check online for further details for the specific vulnerability in question.

If Nessus finds any high, serious, or medium severity issues that are not false positives, the site fails this test. Low severity issues should be brought to the attention of the site owner, but do not cause the site to fail this test on their own. If Nessus gives only warnings, no findings at all, or if all the higher-severity findings were determined to be false positive, the site passes this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.6 Test Backup Procedures

Checklist Item Number: 006

Checklist Item Name: Test Backup Procedures

Reference: After searching web pages and archived Usenet posts, I was unable to find any worthwhile references on backup strategies and recommendations specifically for web sites. These will have to be based on my personal experiences.

For backups of the MySQL database, there is far more information available. The most direct reference is in the MySQL manual itself: <http://dev.mysql.com/doc/mysql/en/mysqldump.html> is the documentation for the mysqldump program, which is designed for automated extraction of all data from a MySQL database.

Risk: This test addresses many of the vulnerabilities listed in section 1.2.3 of this document. For example, any vulnerability in which an attacker could compromise the data stored in the database or modify the pages on the site would cause the site owner to respond by attempting to restore from backups. In those cases, having a good backup would allow easy recovery. This test is most important for vulnerability number three, however, in which the web hosting company suffers a catastrophic loss. In most cases the web hosting company would have their own backups, which could be used in the case of a web page defacement, for example. The site owner's backups would simply be another option. However, the site owner should have at least one local backup of all his PHP code, HTML files, and database contents, in the event that the web hosting company loses all backups, for whatever reason.

The likelihood of exposure to this vulnerability is considered low. As discussed above, these are secondary backups, which supplement the backups being performed by the hosting company. However, the severity of impact is high. If the web host does lose all backups and current data, the web site would be severely crippled. All code would need to be re-written, all users would need to re-register, and all data in the database would need to be re-entered. Because so much of the content is supplied by visitors to the site, it would be impossible to exactly recreate it all.

The net risk is therefore **medium** (3).

Testing Procedure/Compliance Criteria: There are two separate components of this test. First, interview the site owner, and find out what the backup proce

ture is, if any. Ensure that there are frequent backups performed. The frequency of backups is a subjective area, and depends on the needs of the site and the site owner. Since the web host already performs their own regular backups, the site owner's backups are an extra level of protection. Considering the size of this site, weekly backups may be sufficient. Anything less often would be difficult to justify, and more often—for example, daily or every other day—would be preferred. The backups must consist of all HTML, PHP, and CSS files used in the web site, as well as the database contents. If the backup is incomplete in any way, the site fails this test. If the auditor's assessment is that the backup is complete and performed regularly enough for the needs of the site, it passes this initial portion of the test.

Following this initial assessment, the auditor should actually test the backup, with the site owner's cooperation. Have the owner remove all files from the development/staging server, and delete the database (for example, `rm -rf /var/www/site` to delete the files, and `DROP DATABASE database-name` from within the MySQL console to delete the database.) Verify that the site is no longer available by attempting to view the site hosted on this development server via a web browser. Then have the owner restore both the files and the database contents. Verify that the site has been restored by again attempting to view the site via a web browser. Verify that the content in the database and the version of the code are as recent as the backup date indicates they should be. If the site was successfully restored, it passes this portion of the test. If the data is out of date, or cannot be fully restored from backups, the site fails this portion of the test.

If both of these components—the subjective assessment of the backup strategy and the test of the backups—are passed, the site passes this test. If either or both fail, the site fails this test.

In most audits, documented backup and recovery procedures would be examined. However, as this site is a one-person operation, it is a special case. I do not believe that documented backup and recovery procedures are a requirement in this situation.

Test nature: Both objective and subjective. The first phase of the test is partially subjective, as the auditor has to determine what a reasonable backup strategy is for this application. The second phase of the test is objective.

Evidence: To be determined

Findings: To be determined

2.7 Unsafe Hidden Form Elements

Checklist Item Number: 007

Checklist Item Name: Unsafe Hidden Form Elements

Reference: OWASP Guide to Building Secure Web Applications and Web Services, pages 46-47.

Risk: This test addresses vulnerability number 13 in section 1.2.3 of this document. The exposure of this vulnerability is rated as medium-high. This is a common mistake in programming web applications, and even though it is well-known, many sites are still making this mistake today. And because it is a well-known issue, it is a vulnerability that many people can easily exploit. The severity of impact is also rated as medium-high. In the worst case scenario, the application may allow an attacker to give themselves administrative rights to the web application, allowing them to read, change, or delete any or all items in the database.

The net risk of this vulnerability is therefore **serious** (4).

Testing Procedure/Compliance Criteria: In this test, we will use `wget`²¹ to copy all files from the web site so we can search them. To mirror the site with `wget`, use the following command:

```
wget -m http://web.site.address
```

This will copy all pages from the site to the current directory, preserving as many of the original attributes as possible. Next, we need to search the resulting pages for any hidden form fields. These fields will all say `type="hidden"` in the HTML form input options. The quotation marks are technically required, but browsers will accept the option without the quotation marks, so we are not guaranteed that they will be there. Therefore, the auditor needs to search for both `type=hidden` and `type="hidden"` in all pages retrieved by `wget`.

The method used for searching all files depends on the operating system in use. On Unix-like operating systems (e.g., Linux, FreeBSD), use the `grep` tool. This tool is also available for Windows for those auditors who wish to use it. Alternatively, the built-in "find file" functionality in Windows will allow you to search the contents of all files in that directory for the string. The exact method depends on the version of Windows in use, but is generally along the lines of Start > Find > Files and Folders to bring up the find file dialog box. The option to search in files may be on that screen, or may be in other options; again, it depends on the version of Windows in use.

²¹ <http://www.gnu.org/software/wget/wget.html>

If any web pages are found containing hidden form elements, additional assessment is required. If the hidden form elements are simply items that were entered by the user in another form, or otherwise supplied by the client, then changing them does not give the attacker any advantage. However, if the hidden elements contain any field that the user should not be able to change, the site may be vulnerable. Whether it is or not depends on if the application does further validation of these inputs or not, and if the hidden elements control any valuable variables (for example, user ID, or access level.)

To test possibly vulnerable forms further, the “Show Hiddens” bookmarklet²² is recommended. By making this bookmarklet available in your web browser, one click on it exposes all hidden fields to you on the web page as user-editable fields. The auditor should attempt to change the values in these elements and discover if any changes cause insecure behavior. If so, the site fails this test.

If there are hidden elements that cannot be determined to be vulnerable, but which should not be user-editable, the site may pass this test, but the site owner should receive a warning about using such hidden elements.

If there are hidden elements, but they only contain data that was supplied by the user or which the user should be able to edit, the site passes this test. The web developer may wish to consider storing such data in session variables rather than hidden form elements, however.

If there are no hidden elements found in form fields, the web site passes this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.8 Ensure Directory Browsing Settings Are Correct

Checklist Item Number: 008

Checklist Item Name: Ensure Directory Browsing Settings Are Correct

Reference: I could not find any direct references to the security implications of this configuration setting, so this is based on personal experience. However, one useful reference on the actual configuration for this item in Apache is the documentation for the autoindex module in Apache:

²² http://www.squarefree.com/bookmarklets/forms.html#show_hiddens

http://httpd.apache.org/docs/mod/mod_autoindex.html

Risk: This test addresses vulnerability number 18 in section 1.2.3 of this document. This vulnerability can allow a site visitor to explore directory contents in cases where the default index page (e.g., index.html or index.php) is not available. If auto-indexing is enabled when a user requests a directory with no default index page, Apache will automatically generate a listing of all files in that directory. This can expose unwanted files and information to the user. The degree of exposure is medium-low; most web hosting companies disable automatic indexing by default because of the security issues. The severity is medium.

The risk is therefore **medium-low** (2.5).

Testing Procedure/Compliance Criteria: Find all directory paths used on the site. This could be based on the directories that wget created during its mirror of the site in checklist item 007, but if possible it should be based on actual directory listings of the web site provided by the site owner. It is likely that some paths used internally in the PHP code may be accessible from a web browser even if they are never mentioned within the HTML pages.

To test each directory, simply use a web browser to go to that directory on the site. For example, if the site directory listing shows a 'testing' subdirectory, point the web browser to <http://www.site.name/testing/> and see what is returned. If all directories give either a legitimate web page or an error page, then the site passes this test.

If directory browsing is enabled for some directories, the auditor must then interview the site owner. It is possible that this was intentional in some instances, and the owner fully understands the risks and knows what files are revealed this way. While not a best practice from a web design point of view, this would not be a security issue if done intentionally; the security vulnerability here is due to *accidental* exposure of files. If directory browsing is enabled and the site owner has done this on purpose, the site can pass this test. However, if automatic indexing should not have been enabled, and the site owner was not aware of the files being disclosed due to this setting, the site would fail this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.9 Attempt to Brute Force Administrative Account

Checklist Item Number: 009

Checklist Item Name: Attempt to Brute Force Administrative Account

Reference: OWASP Guide to Building Secure Web Applications and Web Services, pages 19-20.

Brutus: <http://www.hoobie.net/brutus/>

“Strong Passwords a Must For Web Apps”, ZDNet UK:
<http://insight.zdnet.co.uk/hardware/servers/0,39020445,2132449,00.htm>

Risk: This test is against vulnerability number 11 in section 1.2.3 of this document. If the username and password to an administrative account can be brute forced, then the attacker would have full access to read, change, or delete any or all entries in the database. The exposure of this vulnerability is medium. Usernames of some existing accounts can be seen by the public on the web site, as they are listed with the user’s reviews on the site. This gives an attacker a starting point of usernames to attempt. In addition, there are many web site password-guessing applications available. The severity of impact of this attack is medium-high. In the worst case scenario, if an attacker does retrieve a valid username and password for an account with administrator-level access, they could change or delete all entries in the database.

The risk of therefore **medium-high** (3.5).

Testing Procedure/Compliance Criteria: Launch Brutus (see reference section for URL). Enter the URL of the login page in the “target” field. This web application uses a form on the page login.php for logins, so set the “type” option to “HTTP (Form)”. Under “HTTP Form options”, set the method to POST. Click on “modify sequence.” On this screen, Brutus will analyze the login form to find the correct variable names for username and password. Enter the login URL into the “Target form” box, and click “Learn form settings.” A new dialog box will appear. If more than one form is available on the page, the auditor will need to select the correct one from the “form name” pull-down list. In the field list, find the form field for the username, click on it, and then click on “Username” below. Do the same for the password field and button. This instructs Brutus to use those two fields. Click on the “Accept” button on that dialog box to return to Brutus’ form definition screen.

Next the auditor needs to put in text that will appear on a successful login screen, which will need to be obtained from the site owner. This is required so that Brutus will know when it has seen a successful login. If possible, the auditor can enter text which will only appear after a successful administrator-level login. Enter this text in the “Primary response” field, and select the “this response is positive” option. Click “OK” to return to the main Brutus screen.

At this point Brutus is ready to scan the site. Click on “Start” at the top of the window. Progress will be shown at the bottom of the screen. Once Brutus has finished its scan, the “Positive Authentication Results” section will show all usernames and passwords that gave a successful login. These should be verified outside of Brutus, by attempting to actually log into the web application using them.

If any usernames and passwords are found by Brutus, and they are verified to work on the web application, the site fails this test. If none are found, or if they are found to be false positives, the site passes this test.

In addition, if there does not appear to be any account lockout procedures after a number of incorrect login attempts, the site owner should receive a warning.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

2.10 Verify Security of any Client-Side Javascript

Checklist Item Number: 010

Checklist Item Name: Verify Security of any Client-Side Javascript

Reference: OWASP Guide to Building Secure Web Applications and Web Services, pages 32-33.

Risk: This test addresses vulnerability number 15 in section 1.2.3 of this document. Many web applications use Javascript that executes in the client’s browser as a means of providing security. For example, the Javascript could validate input to ensure that invalid responses are never passed to the web server, or, in the worst case, could actually check passwords. Since it is trivial to view, change, or remove these checks, they offer no real security. For example, code to check password in Javascript would require the correct password to appear in the code sent to the browser. The degree of exposure is considered to be medium. Many web sites have made this mistake, and it is extremely easy to exploit. The severity of a successful exploit is medium-high. In the worst case scenario, poorly written Javascript controls could give an attacker administrative access to the web site.

The net risk is therefore **medium-high** (3.5).

Testing Procedure/Compliance Criteria: For this test we will once again use the mirror of the web site that we obtained with wget in test 007. In this test, the

auditor will use the same methods to search the files that was described in that testing procedure, but in this case the search is for the string `<script>`. This will allow us to find all client-side scripting in the application. If desired, WebScarab can be used to show all files with scripting; the “scripts” column on the page summary screen will have a check box if that file uses scripting.

If any client-side scripting is in use, the auditor will need to review the code in search of insecure practices. These include testing for correct passwords in Javascript, or validating user input in Javascript instead of on the server. As an example, here is a sample Javascript function that ensures that the data entered in a specific year field is greater than 1980. It looks in the form named “form”, in a field named “year”, and checks the value in it. If it is less than 1980, it displays an alert box for the user.

```
<script language="JavaScript" type="text/javascript">

function validateInput () {

    if (document.form.year.value < 1980) {

        document.form.year.focus();
        alert("The year entered must be 1980 or later");
        return false;

    }

    return true;

}

</script>
```

If any such client-side scripting is found, the auditor should then see what happens if it is bypassed. The easiest way to test this is to simply turn off Javascript in the web browser so that it cannot run the validation functions. If the application is poorly written and relies on having client-side scripting enabled, then the page will need to be edited, either by editing the mirrored copies on the auditor’s computer, or through a intercepting proxy such as WebScarab or Achilles. The auditor can then assess what occurs when the form is used with invalid inputs, and the protective Javascript code is no longer available. It may be that server-side processing checks all inputs again, or it may be that since the server-side code never expected to see these invalid inputs, the application break in some way that may be leveraged to gain additional access to the site.

If client-side scripting is used for security, and the subsequent testing shows that the server-side scripting is not double-checking the input, the site fails this test. If client-side scripting is used, but the server appears to be double-checking the

input as an extra level of protection, the site passes, but the site owner should be warned. If no client-side scripting is being used for protection, the site passes this test.

Test nature: Objective

Evidence: To be determined

Findings: To be determined

© SANS Institute 2004, Author retains full rights.

3 Audit Testing, Evidence, and Findings

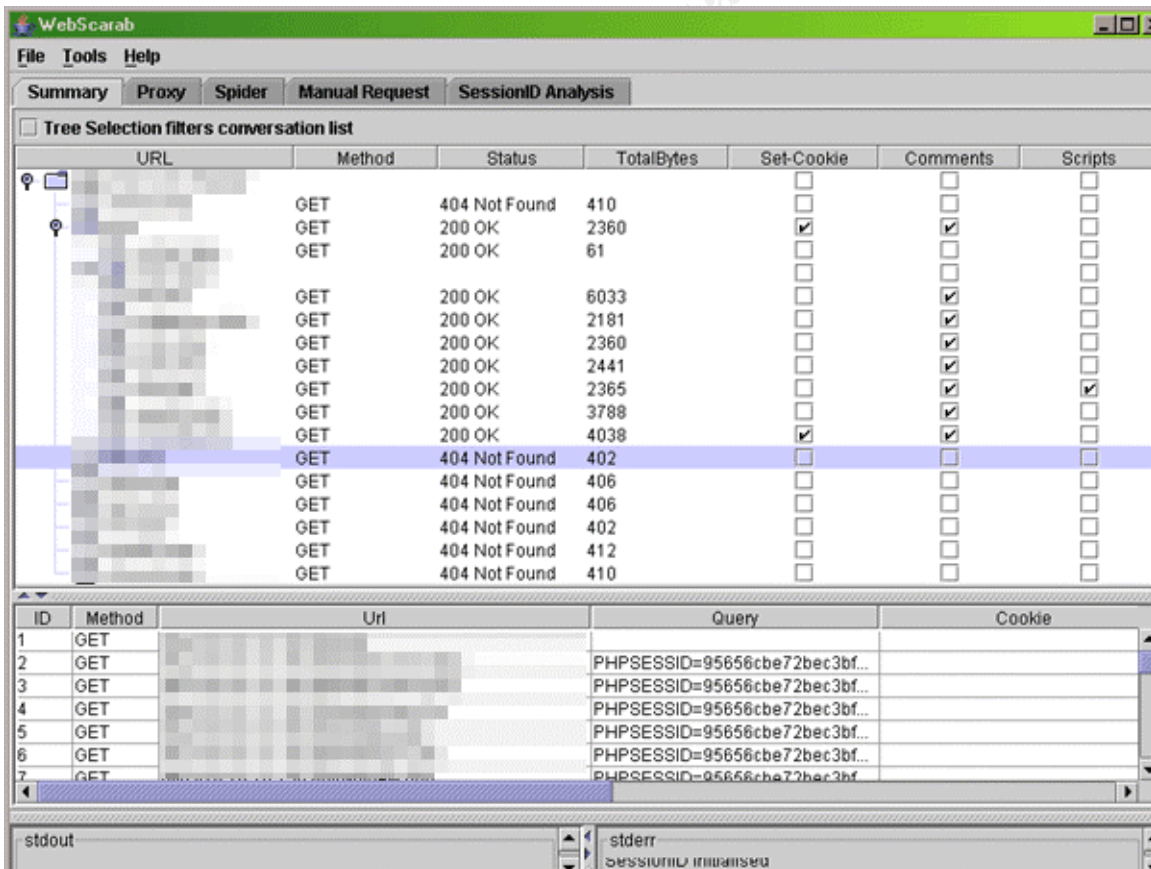
The following sections describe the testing of the checklist items listed in section 2, as well as all findings from these tests.

3.1 Check For Hidden Comments in HTML

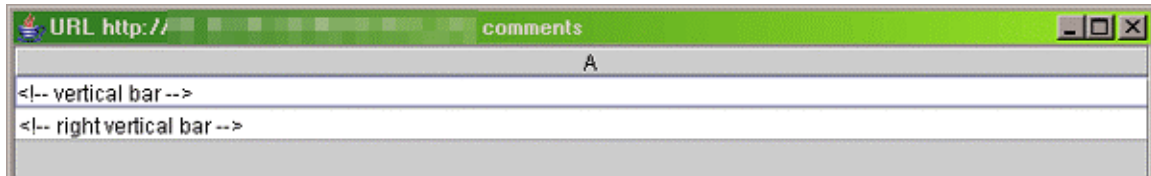
As described in section 2.1, WebScarab was used to spider the site and find all files that have HTML comments within them.

3.1.1 Evidence

Below is a screenshot showing the results of the WebScarab spider of the site:



As shown in the results, eight files in the application contained HTML comments. For each of these, right-clicking on the filename and selecting “View comments” displayed a window containing all comments within that file. All of these files had identical comments; below is a screenshot of one of these files:



3.1.2 Findings

As all comments were harmless, merely separating various sections of the page—vertical menu bars from main content—there was no further testing required.

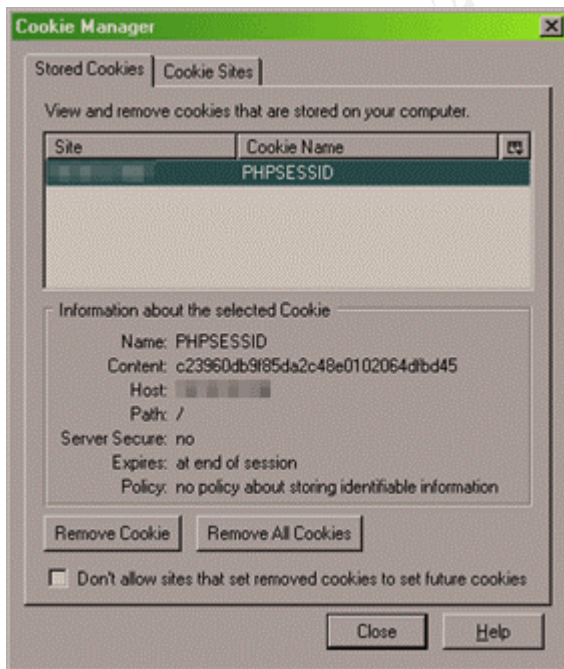
RESULT: PASS

3.2 Session Hijacking Via Cookie Manipulation

In this case, before beginning the test procedure given in section 2.2, I first created a new user profile in Mozilla, the web browser used for this test. This allowed me to start with a clean slate, so to speak; the browser had all default settings and not a single cookie had been set. Following this, I used the procedure described in section 2.2.

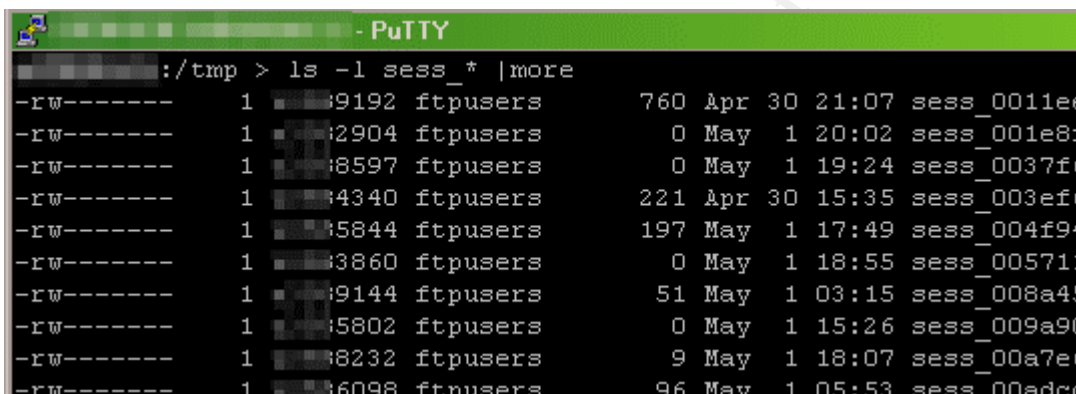
3.2.1 Evidence

I logged into an existing account on the site that had no special privileges. After logging in, the following cookie was set:



3.2.2 Findings

As described in the compliance criteria section in the audit checklist, this cookie shows that the site is using PHP's built-in session management functions. PHP's session management technique is well known, and is not vulnerable to hijacking by manipulating the cookie data. There are still session management issues to be aware of, though. For example, by default PHP stores the session data in the shared temporary directory (/tmp) on the web server. In a shared hosting environment, the web hosting company needs to ensure that other web sites on the same server do not have read access to other sites' user session data. If these precautions are not taken, anyone webmaster using that same server could view the cookie contents of any session on any web site. As a supplemental test, I checked the security settings of these files:



```
- PuTTY
: /tmp > ls -l sess_* | more
-rw-rw-r-- 1 ftpusers 760 Apr 30 21:07 sess_0011ee
-rw-rw-r-- 1 ftpusers 0 May 1 20:02 sess_001e8f
-rw-rw-r-- 1 ftpusers 0 May 1 19:24 sess_0037f6
-rw-rw-r-- 1 ftpusers 221 Apr 30 15:35 sess_003ef6
-rw-rw-r-- 1 ftpusers 197 May 1 17:49 sess_004f94
-rw-rw-r-- 1 ftpusers 0 May 1 18:55 sess_005711
-rw-rw-r-- 1 ftpusers 51 May 1 03:15 sess_008a49
-rw-rw-r-- 1 ftpusers 0 May 1 15:26 sess_009a90
-rw-rw-r-- 1 ftpusers 9 May 1 18:07 sess_00a7e6
-rw-rw-r-- 1 ftpusers 96 May 1 05:53 sess_00adca
```

This screenshot shows that each session file is owned by the user associated with that web site. The permissions column shows that only the file owner is allowed to read or write the file. Therefore, other users are not able to view the contents of AuditApp's session files. There is also no way to correlate the username shown with a web site in order to set the session cookie to the value shown in the filename. Therefore, the web host has sufficiently protected the session files.

RESULT: PASS

3.3 SQL Injection

The site owner gave us read access to the PHP source files that make up the site, and I “grepped” them for calls to the `mysql_query` function. After finding all of the SQL queries, I then analyzed their safety.

3.3.1 Evidence

The very first query I saw in the code was an excellent example of unsafe coding practices, in a file called `authenticate.php`. It is expected to be called using a

URL such as `http://web.site.name/authenticate.php?auth=authstring`. The code in question is below:

```
$auth = $_GET['auth'];  
  
$result = @mysql_query("SELECT * FROM user_auth WHERE  
auth='$auth'");
```

When the user visits the example URL given earlier (`authenticate.php?auth=authstring`), the `$auth` variable in this code would be set to `'authstring'`. This variable supplied by the user is then inserted directly into a SQL query with no sanitization. As the function of this script is to ensure that the user knows the correct authentication string stored in the database, it is definitely vulnerable to attack. If the user could set `auth` to be `' OR 1=1`, the SQL query would become

```
SELECT * FROM user_auth WHERE auth='' OR 1=1
```

Instead of the desired result where the SQL results would be the row containing the authentication string passed from the browser, the result set would be every row in the `user_auth` table, since 1 is always equal to 1. This is because the user was allowed to send that single quote, closing the string that the site was trying to test, and add a new clause to the query. The code then checks to see if any results were returned from this query, and if a result was returned, the code believes that the user supplied a valid authentication string. In the modified version of the query, there will always be rows returned from the query, because of the `1=1` clause. This is a classic SQL injection security hole.

Because of the existence of this hole, I then created the `phpinfo.php` file discussed in the testing procedures, and viewed the setting for `magic_quotes_gpc`:



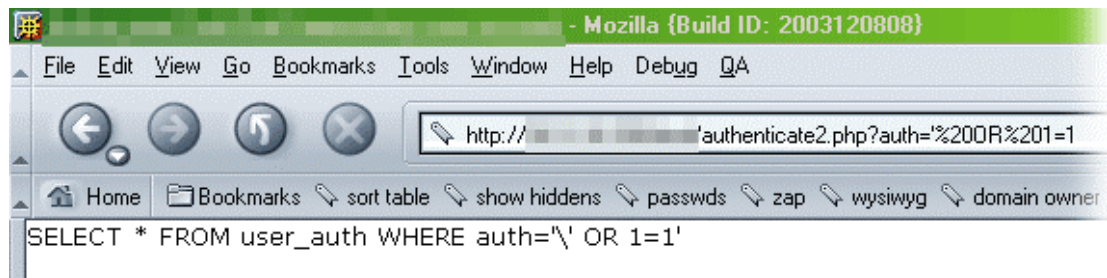
Because it is set to on, PHP should be automatically sanitizing all GET, POST, and cookie data sent from the browser.

To verify this, I then attempted to exploit the hole found above. I entered the following URL in the browser:

```
http://web.site.name/authenticate.php?auth='%20OR%20=1
```

As spaces are not legal characters in an HTTP request, they must be replaced by the `'%20'` string. The resulting page from the site was an error page indicating that I could not be authenticated. To test this further, the site owner and I created an alternate version of the authentication file, named `authenticate2.php`.

This one displayed the SQL query string on the page rather passing it to the MySQL server (essentially, replacing the `mysql_query` function with a print-to-page function). For the purposes of this report, we also removed all site headers and graphics from the document, leaving just the SQL query on the page. After trying the URL request again against this new version of the script, the server gave the following page:



As the displayed string shows, the `magic_quotes_gpc` option escaped the single quote at the beginning of our query. Therefore, instead of ending that string and allowing us to add an additional clause to the SQL query, MySQL would actually be testing for the existence of the string `\' OR 1=1`. This therefore makes it impossible to inject unexpected SQL queries.

Some MySQL users feel that they are already protected against SQL injection attacks because MySQL does not allow for multiple commands in a single function call. For example, we would not have been able to change the function to `SELECT * FROM user_auth WHERE auth=''; DROP TABLE users;` for example. That injection attempt uses a semi-colon after closing the “auth” string to add additional SQL commands—in this case, erasing all user information. A more likely malicious attack might be to create a new account with full permissions to the site. However, as the example found in this audit shows, even though MySQL is automatically immune to this type of attack, there are certainly other instances where SQL injection attacks can be used. Using MySQL as the database server does not automatically protect the site from all SQL injection attacks.

3.3.2 Findings

Because the `magic_quotes_gpc` option was enabled, AuditApp is not currently vulnerable to SQL injection attacks. However, the code itself is not very safe; I found many more examples of insecure SQL queries such as the one above. Therefore, if the site owner ever switches to a different web host, he must be extremely careful that they turn `magic_quotes_gpc` on by default, or allow him to enable it for his site. In addition, he is also placing the security of his site in his current web host’s configuration management process. If they rebuild his server and do not set this option, his site would instantly be vulnerable to a number of SQL injection attacks.

It is recommended that the code be modified to be safe regardless of this site setting. It is possible to write a short function that every PHP page calls before processing any user data that checks the site's `magic_quotes_gpc` setting. If it is enabled, the function does nothing. If it is disabled, then the function escapes all dangerous characters.

RESULT: PASS

3.4 Test for Adequate Safeguards Against Bandwidth Theft

As described in the testing procedure in section 2.4, I sent the following requests directly to port 80 on the web server being tested:

```
GET /go.gif HTTP/1.1
Host: website.name.com
Referer: www.someothersite.com
```

and

```
GET images/1.jpg HTTP/1.1
Host: website.name.com
Referer: www.someothersite.com
```

3.4.1 Evidence

For both of those requests, the web server returned the image requested. To ensure that the site was seeing the correct request with the correct referrer string, I viewed the access log of the server, and found the request. The sanitized version is shown below:

```
my.ip.address.here - - [01/Apr/2004:23:42:15 -0400] "GET
/go.gif HTTP/1.1" 200 1124 website.name.com
"www.someothersite.com" "-" "-"
```

This shows that the server correctly received the referrer, claiming to be a different web site, and that the result code was 200, indicating a successful request for the file.

As the testing was performed through a telnet window, there was no way to view the image file that the server actually sent. There remained the option that the site was configured to send a different image in this case. Some web hosts that offer free web hosting do this automatically; any requests for images that do not come from pages hosted on their site are sent a tiny graphic that explains that this form of linking is not allowed.

Therefore, I then set up a test web page on another web server, so that I could verify the contents of the actual image. I created a web page that had those two

test graphics above, linked directly to the images on AuditApp. Viewing that page in a web browser verified that the web server was sending the actual images requested.

3.4.2 Findings

Because all images were successfully retrieved even when the server was told—through the referrer string—they were linked to from a different web site, there are shown to be no safeguards against bandwidth theft on AuditApp.

RESULT: FAIL

3.5 Scan for Sample Files or Scripts

As described in the testing procedures from section 2.5, I performed a Nessus scan of AuditApp with only the “CGI Abuses” option selected. The scan completed successfully with no error messages.

3.5.1 Evidence

The Nessus scan results are included here:

```
Nessus Scan Report  
-----
```

SUMMARY

- Number of hosts which were alive during the test : 1
- Number of security holes found : 0
- Number of security warnings found : 0
- Number of security notes found : 1

TESTED HOSTS

```
web.site.name.com
```

DETAILS

```
+ web.site.name.com :  
  . List of open ports :  
    o ftp (21/tcp)  
    o ssh (22/tcp)
```

- o http (80/tcp)
- o general/tcp (Security notes found)

. Information found on port general/tcp

Nmap found that this host is running Linux Kernel 2.4.0
- 2.4.17 (X86)

This file was generated by the Nessus Security Scanner

3.5.2 Findings

This scan shows no holes of any kind. The only result is a note to indicate Nessus' guess at what operating system the web site is running. No additional testing was necessary.

RESULT: PASS

3.6 Test Backup Procedures

First I interviewed the site owner to determine the current backup procedures. Following this, we both performed a test of the backup.

3.6.1 Evidence

The site owner does all development on a staging Linux server at his home. All code development is done there, and tested on a local copy of the database, before he deploys the updated code to the production web site. Nightly backups of the local server's files and MySQL database are done automatically, using tar to backup the files and mysqldump to backup the database. Old backups are kept indefinitely.

Since the most important copy of the database is the one on the live web site, a scheduled job on the live web server also performs a mysqldump command nightly to create backups of the database's data. The directory on the web server containing these nightly backups is manually mirrored to the local staging server periodically. This is done with the same script that copies the access logs to the owner's local machine for analysis. He estimates that this is done twice a day, on average. The database backups from the live server are occasionally

used to replace all content in the staging server's database, for two reasons. First, this allows the owner to test the backups; if there are any problems with these backups from the live server, he will discover this when he attempts to import them into his database server. Second, it allows him to do site testing with database contents that very closely resemble the contents of the live site, minus any changes that occurred more recently than the last import.

We then proceeded to test the backups. First, we deleted the database on the development server (using a `DROP DATABASE database-name` command in the MySQL console interface), and attempted to view the AuditApp home page. As expected, we received a MySQL error, as it was unable to find the database being requested in the code. Next, we deleted the root directory of the web server, and again attempted to view the site's home page. This time, we simply received Apache's default "page not found" page. We were therefore able to verify that both the site's HTML and PHP code, along with the database contents, had been deleted.

The site owner then extracted the contents of the previous night's tar file backup to a newly created root directory for the web site. Following this, he uncompressed the previous night's mysqldump backup file, manually created a new empty database in MySQL, and then imported the SQL commands in the backup file into the new database. I then viewed the AuditApp home page in my web browser, and confirmed that it was fully functional.

We then took a closer look at the data stored in the database, and confirmed that it did come from the most recent database backup.

3.6.2 Findings

Nightly backups are sufficiently frequent for this site. The restore procedure was able to successfully restore the site from the most recent backups available with no errors.

The only concern is that the site owner is manually copying the backups of the live site to the local server. This raises the possibility of these backups occurring infrequently if the site owner forgets or is unable to copy the backups at some point. However, given that this copying procedure is a necessary step for the owner to analyze the site's access statistics, it is likely to occur fairly often. In addition, the only way to automate this would be to include a copy of the SSH password in the script, which would bring up other security issues. Therefore, while this is a concern, it is not a major one.

RESULT: PASS

3.7 Unsafe Hidden Form Elements

As described in the testing procedure, I used wget to make a local copy of all files on the server, and searched them for hidden form elements. In addition, I obtained the original PHP files and searched those as well.

3.7.1 Evidence

I found eleven different hidden form elements in use on this site. Four of these are only available through the administrative interface, and therefore were not analyzed further. Those administrative pages all check the user's access level before displaying the page, so the user must be logged into an account with administrative privileges to receive the hidden elements on the page. As a result, there is no further privilege escalation for them to attempt; these users already have full access to the database.

The seven remaining hidden elements available to the public are described below:

- One is on the change password page. It takes an authentication string the user passed in the URL of the page, and includes it in a form submission to a second page. The reason for this design is so the user account details can be accessed using that string from the second page. As this is simply providing user input to a second form, this is safe.
- Two hidden elements are on the page that displays the full details and all content for an item in the database. These two elements on this page are both identical, simply used in different branches of the PHP code, and therefore will be treated as a single element. They are located in a form that allows the user to submit information about the item, and includes the item number used in the database as a hidden field. This is not a value previously provided by the user, as they have no reason to know the internal database ID for that item. However, it does not provide any security risks. The users are permitted to provide information about any item in the database, and therefore changing this internal identification number would have the same effect as if they had gone to a different item's page before submitting their information. Therefore, these two hidden elements are safe.
- A fourth hidden element is on a different page that allows the user to enter a different set of information about an item in the database. This has the exact same function as the previous two hidden elements. The only effect changing this hidden data would have is to add the user input to a different item, which they could have done through legitimate means via the web site's interface. Therefore, this is safe.

- The remaining three hidden elements are all on the same page, which allows users to enter new items into the database. The process of entering new items is a two-step process; the application first asks for three pieces of basic information, and then creates a second page for the user to continue entering information. In order to provide the data entered on this first screen to the final page that actually adds the information to the database, these three items are included on the second page as hidden form elements. This is safe, as the hidden elements are items that the user provided previously. Changing the values hidden in the form would have the same effect as entering different information on the first page.

3.7.2 Findings

All hidden elements in forms on the site have been shown to be safe. However, in every one of the cases examined above, a better solution would be to use sessions. The web site is already using session management to hold user data such as username and access level. Expanding the session variables to include the data that would otherwise be stored in a hidden form element would be a preferable solution to using these hidden elements. As all of these uses are safe, though; this suggestion merely is a preferred implementation. Therefore, the site passes this test.

RESULT: PASS

3.8 Ensure Directory Browsing Settings Are Correct

From earlier tests, I already had a full list of all directories on the web server. This includes two directories that should never be visible to the public. The testing procedures described in the audit checklist were performed on all of these directories. This allowed me to test both the directory browsing settings as well as any other protection mechanisms in place to prevent the public from accessing protected portions of the site.

For example, phpMyAdmin is installed on the live web site. This is a web-based interface to MySQL, giving the site owner a front-end that allows him to view, modify, or delete any item in any table in the database. It also allows any query to be pasted in a plain text query window. As a result, if an unauthorized user was able to access phpMyAdmin, he/she would have full access to the site's database. Obviously, this directory should be well-protected from unauthorized users.

3.8.1 Evidence

The following directories were checked:

Directory	Result
/images	The web server returned an HTTP 403 error page, which means the request was forbidden.
/css	The web server returned an HTTP 403 error page, which means the request was forbidden.
/backup	The web server asked for a username and password using basic authentication. When none was given, the web server returned an HTTP 401 error page, which means authorization is required to view the page.
/logs	The web server asked for a username and password using basic authentication. When none was given, the web server returned an HTTP 401 error page, which means authorization is required to view the page.
/phpmyadmin	The web server asked for a username and password using basic authentication. When none was given, the web server returned an HTTP 401 error page, which means authorization is required to view the page.

Screenshots of the pages returned are not included here, as they were the default pages created by the web hosting company, and therefore would divulge information about the web host in use for this site.

3.8.2 Findings

Browsing the first two directories—/css and /images—was denied because the web server is configured to not automatically create index pages, even though the user is allowed to view files contained in those directories. This is the main configuration item I was testing here, and the site passed.

Accessing the other three directories requires additional authentication. This is correct, as these directories should not be accessible to the public.

The site passes the test. As a side note, it is recommended that the site owner replace the stock error pages supplied by the web host with custom pages that are more useful to the user. While it is unrelated to the security of the site, it would give the site a more professional feel.

RESULT: PASS

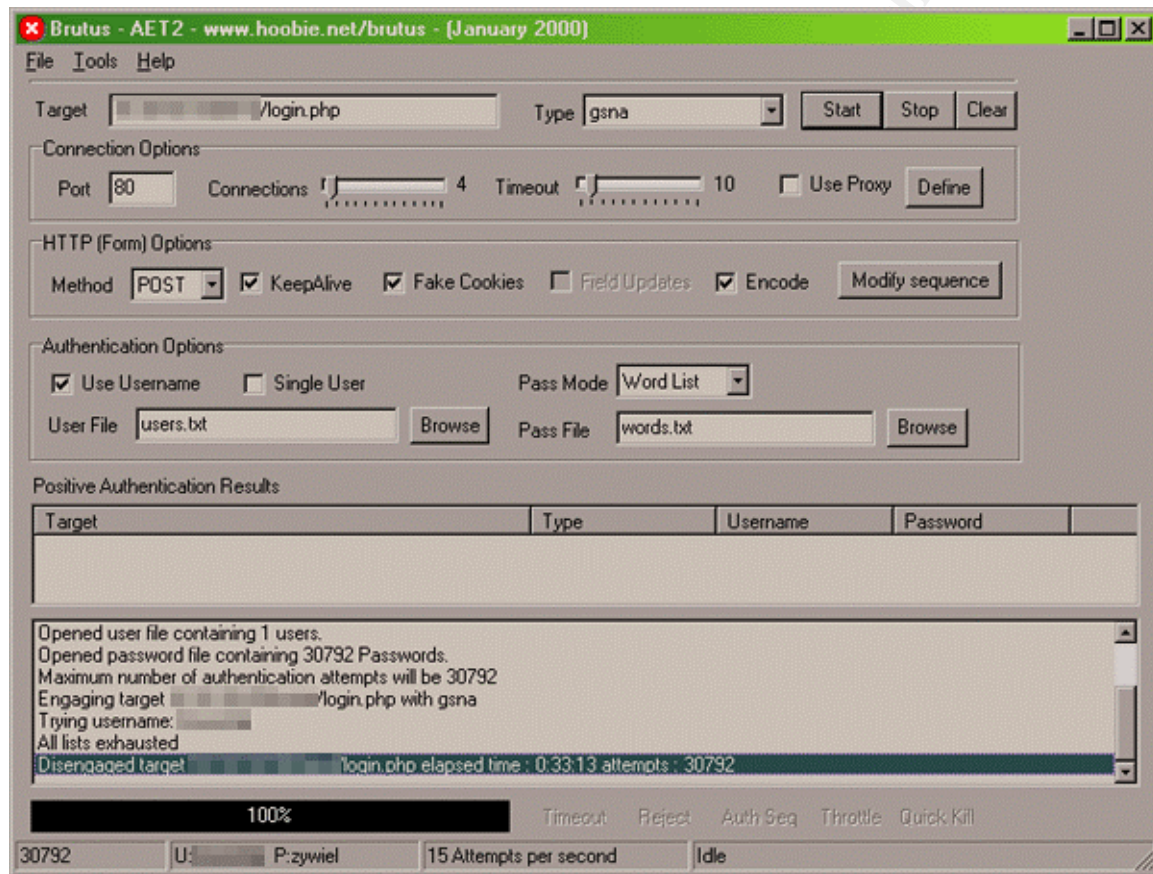
3.9 Attempt to Brute Force Administrative Account

Brutus was launched and configured for the login page for this site. I also created a custom user list that contained only the one administrative account that is currently in use on the web site. The word list was created from a list of common

passwords, a dictionary file, and “The Jargon File”²³, and contained slightly more than 30,000 entries. The scan was performed against the development server and database rather than the live site. Before starting the test the site owner and I verified that the user list on the development database was identical to the one on the live database.

3.9.1 Evidence

The results are shown below:



The “Positive Authentication Results” section is empty, which indicates that Brutus was unable to find a password that successfully logged into the site.

3.9.2 Findings

Brutus was unable to guess the password to the administrative account. Therefore, the site passes this test.

²³ <http://www.catb.org/jargon/>

As the site expands and additional users become administrators, a method of checking their accounts for weak passwords may be desired. This could be an extra check when the user signs up, or could be off-line password cracking against the MD5 password hashes that are stored in the database.

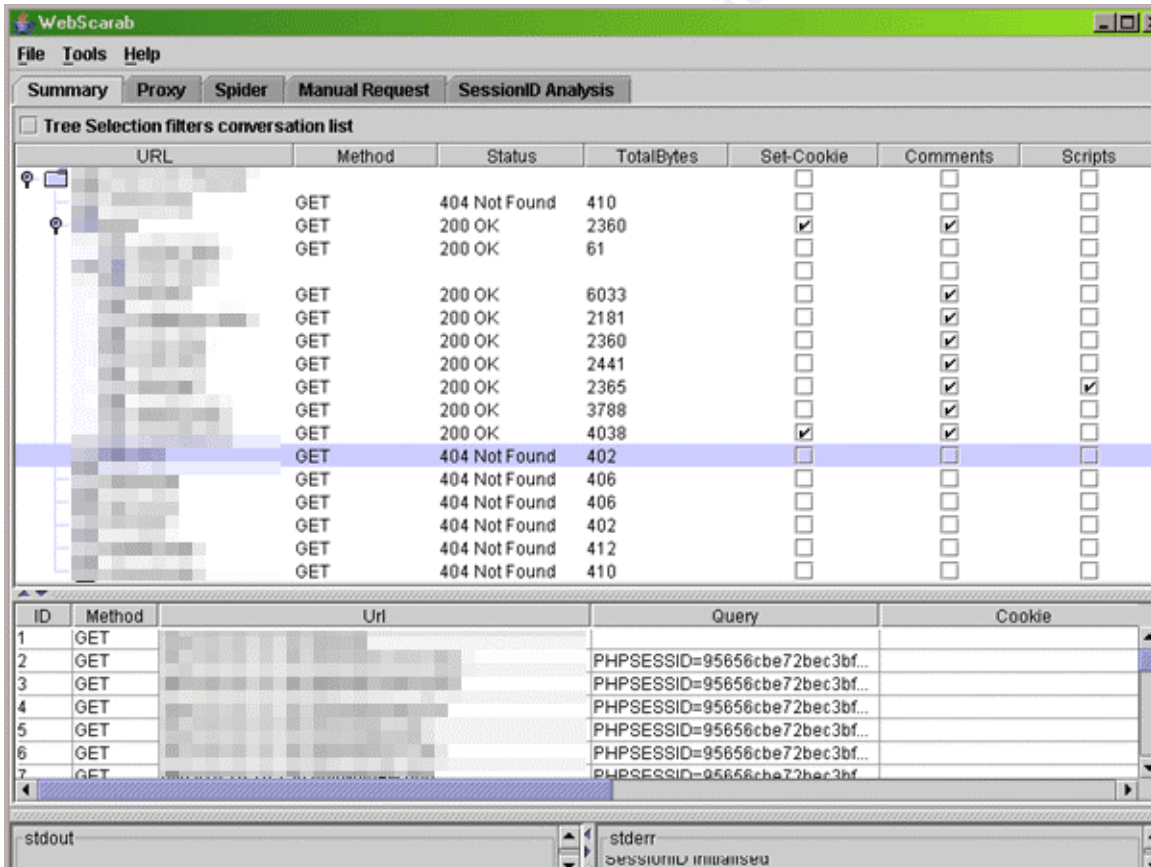
RESULT: PASS

3.10 Verify Security of any Client-Side Javascript

For this test, I once again used the WebScarab listing obtained in section 3.1, test item 001.

3.10.1 Evidence

The WebScarab listing is shown again below:



This shows a single file on the site that is using client-side scripting. The script contained in that file is included below:

```
function setFocus ()
{
    if (!formInUse) {
        document.newitem.title.focus ();
    }
}
```

The HTML code also has an “onload” trigger to call that function, and several “onfocus” triggers to set the formInUse variable. There is no other Javascript on the page.

3.10.2 Findings

The Javascript above only sets the cursor focus to a form field, so that a user can begin typing in that field immediately, rather than having to click on the field before typing. As this is purely an ease-of-use enhancement and unrelated to security, no further analysis needs to be done.

RESULT: PASS

© SANS Institute 2004, Author retains full rights.

4 Audit Report

4.1 Executive Summary

This audit report is a result of an audit of AuditApp, performed by Herschel Gelman in April 2004. As AuditApp is hosted with a web hosting company, this audit only examined the portions of the application that are under the control of the site owner. This includes the code that powers the site and the configuration options that the web hosting company makes available to the site owner.

All vulnerabilities tested were in the medium risk range; some were on the low end of medium, and others on the high end of medium. The audit checklist we created for the site contained ten items to test, and all ten were successfully tested.

The site passed nine of the tests, and failed one. The failed test is item number 004, and was assessed a risk of medium-low.

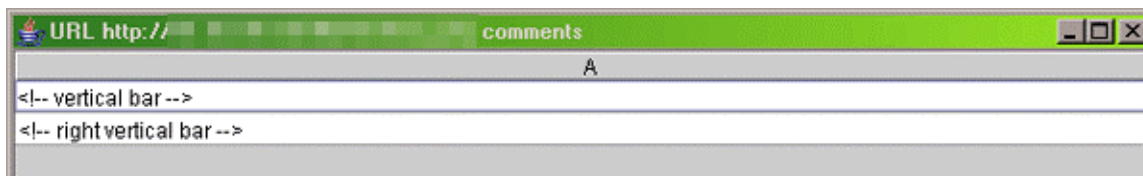
4.2 Audit Findings

This audit had an unusual subject, as it was a one-person operation. Some of the usual requirements that an auditor would be assessing, such as comprehensive security policy and procedures, are not applicable in this case. However, if the site grows to the point of requiring additional staff, a new assessment may be needed to ensure best practices are being followed by all involved.

The findings for each test are detailed below. More detailed descriptions of the testing procedures and results are available in sections 2 and 3 of this document. All but one of these tests were passed; the report on the one failed test is in section 4.2.4 below.

4.2.1 Check For Hidden Comments in HTML

This site had only minimal HTML comments, none of which leaked any information that could be used by an attacker. An example of typical comments in use on this site is below:



The screenshot shows a web browser window with a green title bar. The address bar contains "URL http://". The page title is "comments". The main content area displays two lines of HTML comments: "`<!-- vertical bar -->`" and "`<!-- right vertical bar -->`".

4.2.2 Session Hijacking Via Cookie Manipulation

This site uses session cookies securely, based on PHP's session management functions. In addition, the web hosting company is properly securing session data on their servers.

4.2.3 SQL Injection

This site is not vulnerable to SQL injection attacks, because of the use of PHP's `magic_quotes_gpc` setting.

However, if that setting were disabled, the site has many pages with vulnerable code, and SQL injection attacks could easily be carried out. A sample of vulnerable code—taken from `authenticate.php`—is included here:

```
$auth = $_GET['auth'];  
  
$result = @mysql_query("SELECT * FROM user_auth WHERE  
auth='$auth'");
```

While the site is safe as is, you can greatly improve this code. Please see the recommendations in section 4.3.2.1, as well as the findings in section 3.3.2, for more information

4.2.4 Test for Adequate Safeguards Against Bandwidth Theft

The site failed this test; no safeguards are currently in place to protect against bandwidth theft. The risk is that anyone can create web pages on their own site that link to images stored on AuditApp. The images could also be used in HTML e-mail messages, web-based forum postings, etc. While this is not a problem itself, if the site receives a large number of visitors, AuditApp's bandwidth limit may be exceeded. That will cause the web hosting company to disable access to the site until they receive payment for the extra bandwidth usage.

While this is not a very high risk item—there's no possibility of lost or changed data, and the likelihood of this happening is low—it still has the possibility of creating a denial of service against the site. It is therefore my recommendation that this be corrected.

4.2.5 Scan for Sample Files or Scripts

The site passes this test. No sample files or scripts were present on this site.

4.2.6 Test Backup Procedures

The site passes this test. The site owner performs automated nightly backups of the PHP and HTML code on the development server, as well as automated nightly backups of the database on the production server. The database backups are manually copied to the development server once a day, on average, according to the site owner.

The manual step in there is a concern, but is not a large one, as automating that process would raise additional security issues.

4.2.7 Unsafe Hidden Form Elements

The site had no unsafe hidden form elements, and therefore passed this test. However, there were a number of hidden form elements in use to perform functions that could have been coded in better ways.

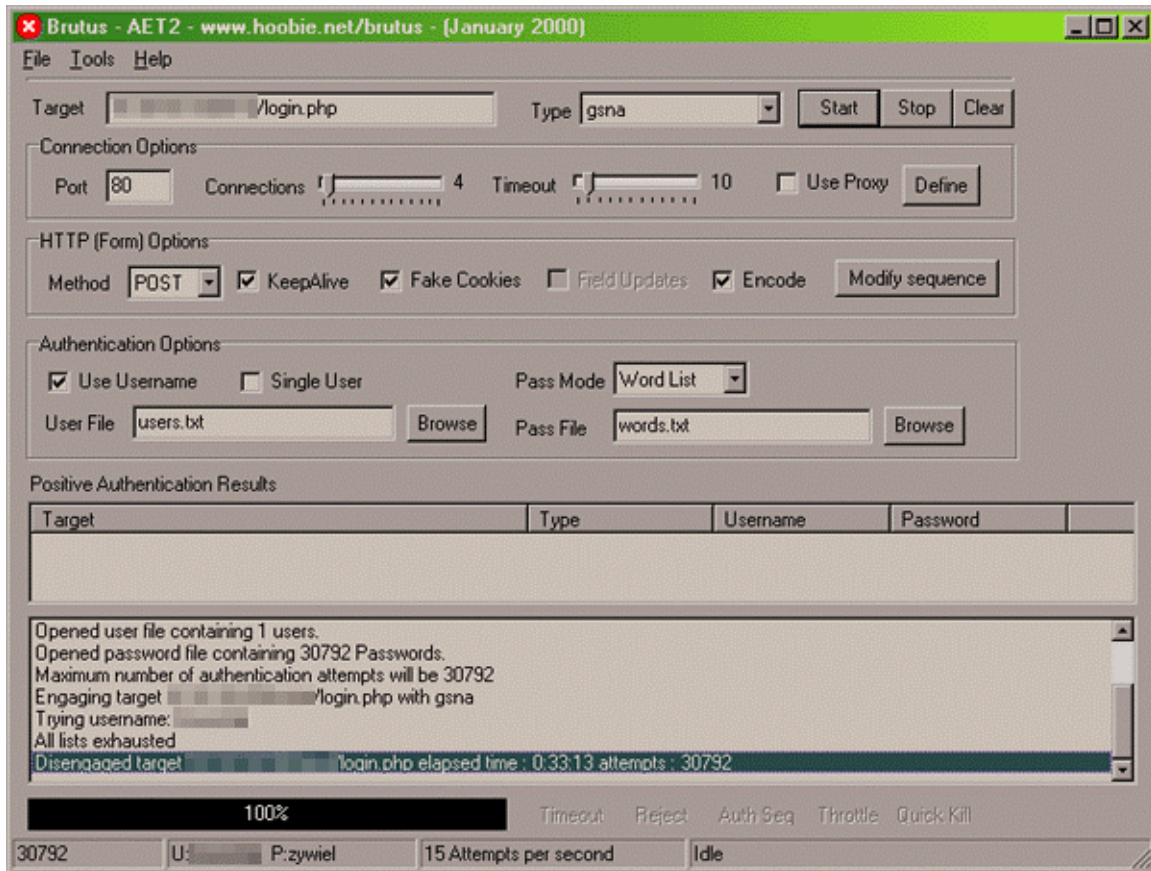
4.2.8 Ensure Directory Browsing Settings Are Correct

I tested the five subdirectories that exist on the web site, and all either asked for authentication—for example, to get to the MySQL administration scripts—or refused to generate a directory listing. This is the expected result, and the site therefore passed this test.

4.2.9 Attempt to Brute Force Administrative Account

My attempt to crack the password for the administrator's account via a brute force attack was unsuccessful. The empty "Positive Authentication Results" box on the results screen below shows that no working username/password combinations were found:

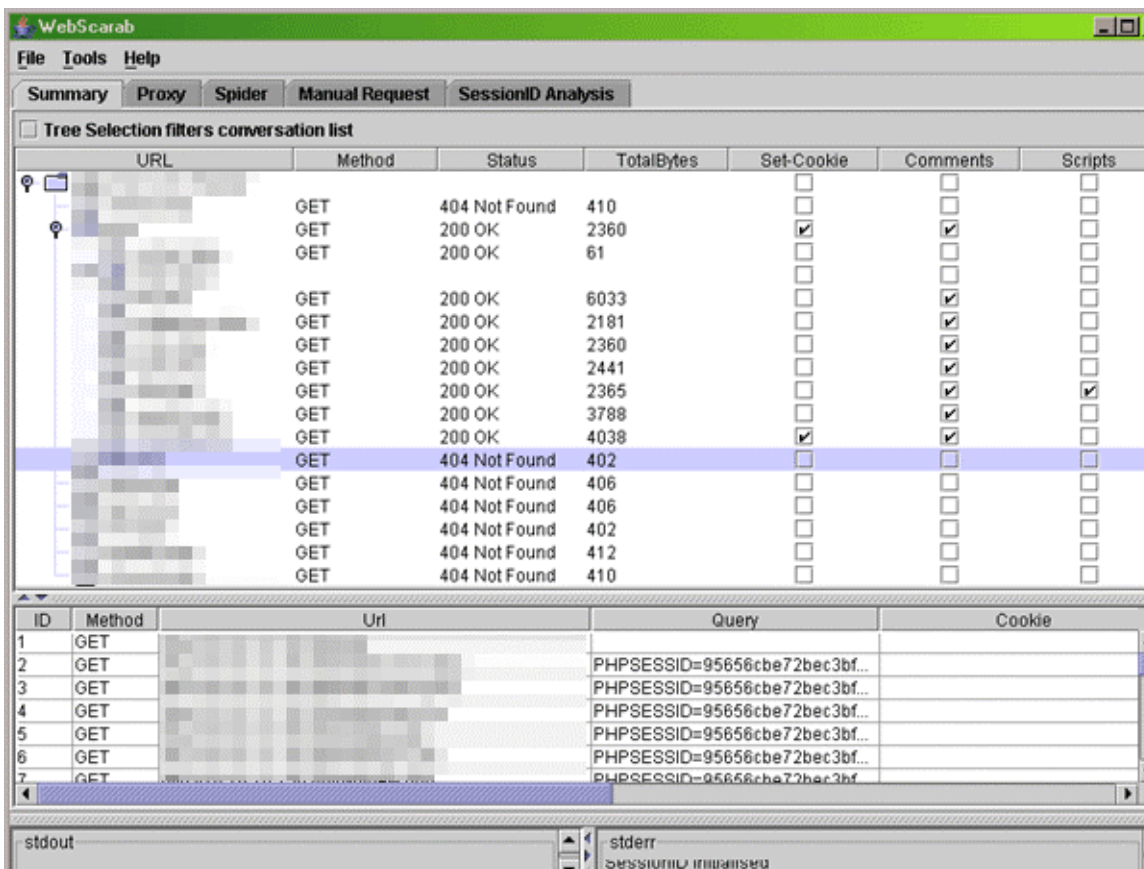
© SANS Institute 2004. All rights reserved.



However, as there is only one administrative account at the moment, the site owner should pay additional attention to this item when more administrative accounts are created.

4.2.10 Verify Security of any Client-Side Javascript

Only one instance of client-side scripting was found in use on this site, as shown below:



This script was a simple cursor focus script that has no security implications, and therefore the site passed this test.

4.3 Audit Recommendations

4.3.1 Highly Recommended Actions

4.3.1.1 Protect Against Bandwidth Theft

4.3.1.1.1 Description

While this is the lowest risk item that I tested for, and it is possible that this would never be an issue for this web site, I still recommend implementing some form of protection against bandwidth theft. As this vulnerability creates the possibility of a denial of service—whether accidental or intentional—it should be corrected.

4.3.1.1.2 Costs

The cost to correct this is minimal. I would estimate at most one hour of the site owner’s time to research the site settings to protect against this, implement it, and test it.

It does create the possibility for additional complexity later. For example, if a problem arises with some visitors viewing images on the site, the protection mechanism that was added would be another item that would need to be analyzed. However, my personal experience is that this is a very common configuration option that is currently in use on many web sites. Therefore, this additional complexity should be minimal, if any.

4.3.1.1.3 Compensating Controls

As the cost to eliminate this risk is so low, no compensating controls are needed.

4.3.2 Lower Priority Recommendations

These recommendations are simply suggestions to improve the potential security posture of AuditApp, and do not reflect any existing vulnerability in the web site.

4.3.2.1 SQL Injection

As the code is now, the site's security against SQL injection attacks depends on PHP's `magic_quotes_gpc` setting being enabled. If this is accidentally disabled, or if the site owner switches to a different web host that does not have this option enabled, the site would be extremely vulnerable to injection attacks.

I recommend that the code be strengthened so that it is safe regardless of the `magic_quotes_gpc` setting. It is possible to write a short function that is called at the beginning of every page that checks the server's `magic_quotes_gpc` setting. If it is enabled, the function does nothing. If it is disabled, the function escapes all dangerous characters, doing the job that `magic_quotes_gpc` would have done otherwise. This would allow for guaranteed safety against SQL injection attacks regardless of PHP's configuration settings on the site.

4.3.2.2 Hidden form elements

AuditApp has a number of hidden form elements: 11 total, seven of which are available to the public viewing the site, and four of which are only seen by users with administrative access.

While all of the publicly accessible items were audited and do not pose a security risk, I recommend replacing them with session variables that can accomplish the same job. There are no security issues here, but that would be a much cleaner and preferred implementation.

4.3.2.3 Future password safety

The current single administrative account proved to be safe from the brute force attack I launched against it. However, as additional administrator-level accounts

are created on this site, the potential for an attacker to successfully guess or brute-force a valid login increases. Therefore, I would recommend incorporating password checks into the PHP code when the account is created, and/or performing offline password cracking attempts against the password's MD5 hash that is stored in the database. The site owner would thereby ensure that he is adequately protected against password guessing attacks against any administrative account on AuditApp.

© SANS Institute 2004, Author retains full rights.

References

- Atkinson, K. (2003). "Kevin's word list page." <<http://wordlist.sourceforge.net/>> (28 Apr. 2004).
- "Brutus: the remote password cracker." <<http://www.hoobie.net/brutus/>> (24 Apr. 2004).
- Curphey, M., Endler, D., Hau, W., Taylor, S., Smith, T., et al. (2002). "A guide to building secure web applications: the open web application security project." Version 1.1.1. <<http://www.owasp.org/documentation/guide>> (1 May 2004).
- Fredholm, W. (2003). "Web application security: layers of protection." *SANS InfoSec Reading Room: Security White Papers*. <<http://www.sans.org/rr/papers/index.php?id=965>> (24 Apr. 2004).
- Harper, M. (2002). "SQL injection attacks: are you safe?" *Sitepoint*. <<http://www.sitepoint.com/article/794>> (28 Apr. 2004).
- Hendrickx, Michael (2004). "Lilith: web application auditing." <<http://users.pandora.be/0xfffffice/scanit/tools/lilith/>> (1 May 2004).
- Ollmann, Gunter (2003). "Application assessment questioning." <<http://www.technicalinfo.net/papers/AssessmentQuestions.html>> (28 Apr. 2004).
- Pisetsky, A. (2002). "Securing e-commerce web sites." *SANS InfoSec Reading Room: Security White Papers*. <<http://www.sans.org/rr/papers/index.php?id=303>> (24 Apr. 2004).
- Rafail, J. (2001). "Cross-site scripting vulnerabilities." <http://www.cert.org/archive/pdf/cross_site_scripting.pdf> (24 Apr. 2004).
- Shiarla, M. (2002). "Cross-sight scripting vulnerabilities [sic]." *SANS InfoSec Reading Room: Security White Papers*. <<http://www.sans.org/rr/papers/index.php?id=478>> (24 Apr. 2004).
- "SQL injection walkthrough." *SecuriTeam*. <<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>> (1 May 2004).
- "Web application security archive." *SecurityFocus*. <<http://www.securityfocus.com/archive/107>> (28 Apr. 2004).

“Web application security mailing list charter v1.0.” *SecurityFocus*.
<http://www.securityfocus.com/popups/forums/web_application_security/intro.shtml> (28 Apr. 2004).

“Webscarab.” *The Open Web Application Security Project*.
<<http://www.owasp.org/development/webscarab>> (1 May 2004).

© SANS Institute 2004, Author retains full rights.

Upcoming SANS IT Audit Training

CLICK HERE TO
{REGISTER NOW}



AUDIT CHECKLIST
 Audit Satisfactory
 Nonconformances Found
 Observations Made