

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Audit Report

Collaborative Research System

Web Application Security Audit

© SANS Institute 2004, Author retains full rights.

GSNA Practical Assignment, Version 3.2, Option 1

Submitted By

Dan Aiken, GSEC
December 15, 2004

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Abstract

I am auditing an Internet-based application used for human studies medical research at an academic medical center. I will call it the Collaborative Research System (CRS—a fictitious name for a live production application). Research is one of the three emphases at the medical center, along with teaching and_—_first and foremost_—_providing excellent patient care. This system was first implemented as an intranet-based system and later moved to the internet.

The information processed and stored by CRS is sensitive patient and research information. A high level of protection for CRS is vital to the hospital and its research organization, especially since CRS is accessible from the Internet. Therefore, I will audit CRS to assess its ability to protect patient and research information from outside attacks against the most common Web application vulnerabilities.

This audit will be specific to the application. The network, server, and workstations are referenced in the audit to define the environment, but they will not themselves be audited.

Auditing web applications is still in its infancy. Few freeware or shareware tools focus on this very important subject. Most security audit tools focus on the network and servers. However, applications are often vulnerable to attacks that will not be detected by network and server security controls, and could compromise not only the application and its data, but the network and servers as well.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Contents

Abstract ii

Figures iv

Tables v

1. Research in Audit, Measurement Practice, and Control 1

1.1. Description of the System 1

1.2. Evaluation of the Most Significant Risks 2

1.3. Current State of Practice – Secure Configurations or Audit Methods 4

1.4. Tools 5

2. Audit Checklist 9

2.1. Scope 9

2.2. Structure 9

2.3. Conventions 10

2.4. Unvalidated Input Checklist 10

2.5. Broken Access Control Checklist 12

2.6. Broken Authentication and Session Management Checklist 14

2.7. Cross-site Scripting (XSS) Flaws Checklist 19

2.8. Buffer Overflows Checklist 20

2.9. Injection Flaws Checklist 20

2.10. Improper Error Handling Checklist 21

2.11. Insecure Storage Checklist 23

2.12. Denial of Service Checklist 25

2.13. Hidden Content 26

3. Audit Testing, Evidence, and Findings 28

3.1. Unvalidated Input Checklist 28

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.2. Broken Access Control Checklist 32
3.3. Broken Authentication and Session Management Checklist 34
3.4. Cross-site Scripting (XSS) Flaws Checklist 39
3.5. Improper Error Handling Checklist 40
3.6. Insecure Storage Checklist 41
4. Audit Report 45
4.1. Executive Summary 45
4.1.1. Positive Findings 46
4.1.2. Negative Findings 46
4.2. Audit Findings 46
4.2.1. Terminology 46
4.2.2. Passed Audits 47
4.2.3. Failed Audits 48
4.3. Audit Recommendations 53
4.4. Compensating Controls 55
Appendix A – Top 10 Most Critical Web Application Vulnerabilities 58
Appendix B – Client-Side Validation 60
Appendix C – Cache Control HTTP Header Statement 62
Appendix D – Authentication and Session Management 63
Appendix E – Error Handling 64
Appendix F – Secure Storage 65
Appendix G – Additional Resources 67
References 69

Figures

Figure 1.1 – CRS Configuration 2

© SANS Institute 2004, Author retains full rights.
© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Figure 1.2 – Achilles HTTP / HTTPS Proxy 6

Tables

Table 1.1 – CRS Configuration 1
Table 1.2 – Summary of Risk Evaluation 3
Table 2.1 – Unvalidated Input Checklist (UI) 10
Table 2.2 – Broken Access Control Checklist (BA) 12
Table 2.3 – Broken Authentication and Session Management Checklist (AS) 15
Table 2.4 – Cross-site Scripting (XSS) Flaws Checklist (XS) 19
Table 2.5 – Buffer Overflows Checklist (BO) 20
Table 2.6 – Injection Flaws Checklist (IF) 21
Table 2.7 – Improper Error Handling Checklist (IE) 22
Table 2.8 – Insecure Storage Checklist (IS) 23
Table 2.9 – Denial of Service Checklist (DS) 25
Table 2.10 – Hidden Content Checklist (HC) 26
Table 3.1 – Unvalidated Input Checklist (UI) 28
Table 3.2 – Broken Access Control Checklist (BA) 32
Table 3.3 – Broken Authentication and Session Management Checklist (AS) 34
Table 3.4 – Cross-site Scripting (XSS) Flaws Checklist (XS) 39
Table 3.5 – Improper Error Handling Checklist (IE) 40
Table 3.6 – Insecure Storage Checklist (IS) 41
Table 4.1 – Error Handling 47
Table 4.2 – Missing Application Security Policies 48
Table 4.3 – Missing Application Security Design Specifications 48
Table 4.4 – Unvalidated Input 49
Table 4.5 – Broken Access Control 49

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table 4.6 – Broken Authentication and Session Management 50

Table 4.7 – Cross-site Scripting (XSS) Flaws 51

Table 4.8 – Improper Error Handling 52

Table 4.9 – Insecure Storage 52

Table 4.10 – Application Security Policies 53

Table 4.11 – Application Security Design Specifications 53

Table 4.12 – Input Validation 54

Table 4.13 – Access Control 54

Table 4.14 – Authentication and Session Management 54

Table 4.15 – Cross-site Scripting (XSS) Prevention 54

Table 4.16 – Error Handling 54

Table 4.17 – Secure Storage 55

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1. Research in Audit, Measurement Practice, and Control

1.1. Description of the System

I am auditing the Collaborative Research System (CRS), an Internet-based application that was developed to join workflow, research records, specimen data, and logistics information with the medical records of individual medical research patients involved in human subjects clinical trials. The goal was to create a secure, internet-based clinical research application to connect patients, doctors, and laboratory scientists. CRS was designed to enable researchers to collaborate externally, make study and patient data available within a research organization, and extend this capability to the broader research community.

CRS is a J2EE 3-tier Internet-based application using XSL and XSLT for the BEA Weblogic v.6.1 Presentation Layer, a Java business layer, and an Oracle v.9.0 database layer. Servers are located in open equipment racks in the hospital's main computer center.

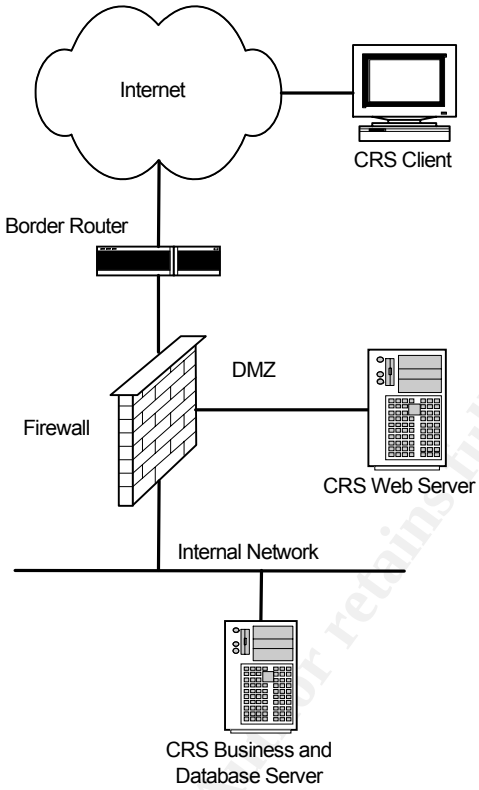
Table 1.1 – CRS Configuration

Application Tier	Hardware and Software
Client	Internet Explorer 5.5 or later required
Presentation	Dell PowerEdge 1550R BEA Weblogic 6.1 Microsoft Internet Information Server 5.0 Windows 2000 Server
Business and Database	Dell PowerEdge 4400R Oracle 9i Windows 2000 Server

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Figure 1.1 – CRS Configuration



1.2. Evaluation of the Most Significant Risks

Gartner analyst John Pescatore estimates that 75% of attacks against Web servers are entering through applications and not at the network level. And, adds Pescatore, when a company makes even subtle changes on its Web sites and applications, new vulnerabilities can arise. (Hulme)

Application vulnerabilities are often neglected, but they are as important to deal with as network issues. If every company eliminated [the OWASP Top 10 Critical Web Application Security Vulnerabilities], their work wouldn't be done, but they, and the Internet, would be significantly safer. (Beales)

By design,…the network must route legitimate traffic to the critical resources housing business logic in the form of applications. Network security protects the integrity and reliability of the traffic to critical resources, but application logic must determine what input or transactions are legitimate. Manipulation and corruption of application logic is an attacker's approach to compromising business data. (Levine, p.1)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

CRS, which stores and processes patient and research data generated from human subjects medical research clinical trials, is an attractive target for hacking and for corporate espionage.

Both patients and research sponsors depend on the effectiveness of the application's security controls. Inappropriate disclosure of patient data can have results ranging from patient irritation and embarrassment to social and financial loss. Likewise, disclosure of research data can result in competitive disadvantage and financial loss for research sponsors. In either case, patients or sponsors are likely to lose confidence and trust in the hospital and research organization.

The compromise of the security of CRS threatens not only patient and research data, but also the security of the hospital network. The consequences of a CRS security incident could extend to other medical and business processes and data.

The security of CRS becomes even more important because CRS is hosted on the Internet, thus having a high level of exposure to malicious attack. Considering the value and sensitivity of CRS and other hospital data, CRS should be carefully controlled and thoroughly secured.

Although risks are often divided into intentional or accidental, for the purposes of this audit the necessary protections are the same. For the most part, it also doesn't matter whether the attack is internal or external, although an internal attacker could have additional knowledge about the application logic and database structure that could make an attack more effective.

A most helpful web site is the Open Web Application Security Project (OWASP) at www.owasp.org. Among other resources, OWASP offers a list of the ten most critical web application vulnerabilities. My audit of CRS will be based on nine of the ten listed vulnerabilities.

I will also audit for the presence of Hidden Content, a vulnerability described in *Auditing Web Servers and Applications* (Rhoades), Day 3 of the SANS *Auditing Networks, Parameters, and Systems* course, under Web Application Audit—Higher Level Concepts. This does not seem to be included in the OWASP Top Ten list.

Table 1.2 – Summary of Risk Evaluation

Threat	Risk	Likelihood	Severity	I/P/O	Consequences
Unvalidated Input	Undetected malicious input	Moderate to High	High	Input	Disclosure of sensitive data or compromise of the web site
Broken Access Control	Improper access to restricted data	Moderate	High	Process	Disclosure of sensitive data or compromise

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Threat	Risk	Likelihood	Severity	I/P/O	Consequences
					of the web site
Broken Authentication and Session Management	Impersonation or session hijacking	Moderate	High	Process	Disclosure of sensitive data or compromise of the web site
XSS Flaws	Loss of user confidence and undetected malicious input	Moderate to High	High	Input	Disclosure of sensitive data or compromise of the web site
Buffer Overflows	Corruption of application execution	Low	Moderate	Input	Crashing the application
Injection Flaws	Undetected malicious input	Moderate	High	Input	Disclosure of sensitive data or the complete compromise of the web site
Improper Error Handling	Disclosure of application implementation details	Moderate	Moderate	Output	Provide useful implementation details to an attacker
Insecure Storage	Disclosure of sensitive data	Moderate	High	Output	Disclosure of sensitive data or compromise of the web site
DoS	Exhaustion of application resources	Moderate to High	Moderate	Process	Lack of application availability
Hidden Content	Disclosure of unnecessary system data	Moderate	Moderate	Output	Provide useful implementation details to an attacker

© SANS Institute 2004, Author retains full rights.

1.3. Current State of Practice – Secure Configurations or Audit Methods

Currently, security auditing of web applications is still in its infancy. Some tools exist that will assist in the audit, but there are many more tools available to audit the network, internet server, application server, and database server than the application itself. In many cases, the most effective approach, in my opinion, is to review the application development policies, design requirements, and application code. This may result in a less than ideal audit since many of the audit steps are subjective and depend heavily on the experience and development skills of the auditor. As tools

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

become more mature and more readily available, there may be less need for detailed code reviews. However, I recommend that regular reviews of policies, design requirements, and application code continue to be part of application development and maintenance processes for two reasons: to catch errors at the earliest possible stage of the development process and to prevent or uncover insider attacks by developers.

In addition to the Ten Most Critical Web Vulnerabilities list (see Appendix A), OWASP produces free, professional-quality, open-source documentation, tools, and standards for application developers and security professionals. The Ten Most Critical Web Vulnerabilities list also includes testing information and a checklist for these vulnerabilities. This is a very helpful site.

Another site targeted at the security of web applications is the Web Application Security Consortium (www.webappsec.org), which has produced a Threat Classification in an effort to promote industry standard terminology for describing web application security issues.

Microsoft has a helpful resource: “Improving Web Application Security: Threats and Countermeasures” in the Microsoft MSDN Library (msdn.microsoft.com/library/_en-us/dnnetsec/html/ThreatCounter.asp). This resource addresses security considerations during the design, development, implementation, and maintenance phases of a .NET web application. It is a practical guide complete with many useful examples.

The SANS InfoSec Reading Room (www.sans.org/rr/audittech/) has an excellent resource that addresses application security: *Auditing Web Applications for Small and Medium Sized Businesses* by Angela Loomis (Loomis).

Web Application Security – Layers of Protection, by William Fredholm, in the SANS Reading Room, includes helpful insights and references to sites and tools that are useful for auditing web applications. (Fredholm)

On November 16, 2004, the FDIC sent a letter to all FDIC-supervised banks (commercial and savings), *Guidance on Developing an Effective Computer Software Evaluation Program to Assure Quality and Regulatory Compliance*. In this letter, the FDIC provided several steps these institutions should take to assure product quality—including application security—and regulatory compliance. These steps would be helpful to any organization when developing or purchasing software. (FDIC)

1.4. Tools

> Unfortunately, most security products available today cannot adequately examine the applications that reside on your Web server! Yet these applications often provide backend access to confidential data! (SPI Dynamics)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Most security tools available today focus on network and server security. However, there are some useful tools available; some are freeware, others are commercial products.

SSLDigger – Foundstone has developed a freeware tool, SSLDigger v1.0, to assess the strength of SSL servers by testing the strength of the ciphers supported by the site (www.foundstone.com). An informative whitepaper on the need for strong SSL ciphers is also available on this site. (Araujo)

Achilles – Achilles is a publicly released general-purpose web application security assessment tool. Achilles acts as a HTTP / HTTPS proxy that sits between a browser and a web server and allows a user to intercept, log, and modify web traffic on the fly. (achilles.mavensecurity.com)

Figure 1.2 – Achilles HTTP / HTTPS Proxy



Webcracker – This program exploits a vulnerability in web site authentication methods. A password protected website can be easily brute-force hacked if there is no set limit on the number of times an incorrect password or User ID can be tried. (www.securityfocus.com/tools/706)

WebInspect – This is a commercial web application vulnerability assessment tool from SPI Dynamics. You can download a 15-day trial version that includes a key allowing you to scan their test web site. WebInspect comes with several policies, including policies to audit against GLBA, HIPAA, and SOX security standards and OWASP Top Ten vulnerabilities, a Full scan policy, and an aggressive Assault policy. (**Warning**: the Assault policy could bring down a web site.)

Application Firewalls – There are a number of application firewall products that could offer protection for existing and new web applications. Network World Fusion (www.nwfusion.com/_bg/2004/appsecurity/index.jsp) rated and compared ten application firewall products. Purchase prices for the reviewed products range from $1,295 to $35,000.

Several higher-end application firewall products are shown below. Considering the effort required to make an existing application secure, an application firewall may be the most cost-effective way to improve and maintain application security, particularly for sites that have multiple installed web applications that are not properly secured.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- **NC-1000 Application Security Gateway**
  $29,000
  ”The NetContinuum Application Security Gateway delivers the highest level of protection available for your critical applications — eliminating data theft, financial fraud and loss of customer confidence due to web application attacks.”
  www.netcontinuum.com/

- **SecureSphere G4 Gateway and SecureSphere MX Management Server**
  $30,000
  ”The SecureSphere Dynamic Profiling Firewall family of appliances — including the G4 Gateway and the MX Management Server — delivers a comprehensive security solution that proactively identifies and blocks attacks that threaten your mission-critical web-based enterprise applications and databases.”
  www.imperva.com/products/securesphere/

- **TrafficShield**
  $25-35,000
  ”TrafficShield is a Web Application Firewall that provides comprehensive, proactive, application-layer protection against both generalized and targeted attacks. TrafficShield employs a positive security model ('deny all unless allowed') to permit only valid and authorized application transactions, while automatically protecting critical Web applications from attacks such as Google hacking, cross-site scripting, and parameter tampering.”
  www.f5.com/f5products/products/TrafficShield/

- **AppShield**
  $20,000
  “AppShield, an automatic Web application firewall, provides enterprise-class Web intrusion prevention for a failsafe defense against all application level breaches. AppShield allows for easy application deployment in a secure environment by intelligently identifying the legitimate requests made of an ebusiness site and permitting only those actions to take place, enforcing the Web and business logic of the site. By preventing, logging and alerting administrators to any type of application manipulation through the browser, AppShield maintains application behavior 24/7 without the need for signatures or rules.”
  www.watchfire.com/products/appshield/default.asp

- **Teros Secure Application Gateway**
  $25,000
  ”The Teros Secure Application Gateway does for Web and Web Services applications what network firewalls do for the network. The Teros Gateway is a hardened security appliance that is deployed directly in the data path of application traffic and blocks attacks that are not detected by network-based firewalls and intrusion detection systems. The Teros Gateway enforces a positive security model that only permits correct application behavior, without relying on attack signatures. It provides defenses for vulnerabilities that may exist within custom applications, as well as the known weaknesses in commercially-

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

developed software."
www.teros.com/products/appliances/gateway/index.shtml

- **e-Gap Application Firewall**
  $23,000
  "The e-Gap® Application Firewall enables organizations to rapidly deploy secure web-based access to sensitive core applications. The System may be used to protect e-business applications for customers or partners (such as eCRM, supply chain integration or e-billing). It protects against known and unknown threats by isolating application servers - via Air Gap technology - and tightly controlling application layer access to them. It also significantly reduces the urgency to patch production web servers. It unites all of the application-protection components into a single application-centric appliance, and features automatic learning of the application to generate and enforce application-level rule sets. Encryption, authorization, authentication, PKI, HTTP payload screening, automatic rule-set generation and a physical air gap all reside within an integrated software/hardware platform."
  www.whalecommunications.com/site/Whale/Corporate/Whale.asp?pi=35

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2. Audit Checklist

A number of the following audit steps will involve code reviews. It is not always possible to provide a detailed explanation of "how to test" (for example, when looking for appropriate and consistent error handling). In these cases, the value of the audit is heavily dependant on the skills and experience of the auditor(s). Where possible, the code review should be followed by testing to show the results of vulnerabilities discovered in the code.

2.1. Scope

"The Gartner Group estimates that 70 percent of computer attacks are now aimed not at individual networks, but at the applications that run on them." (Roberts) Networks may be secure, but most network security controls will not prevent attacks on Internet applications because they see them as legitimate network traffic. It is up to the individual application to validate input values to be sure they are valid and appropriate.

Although the overall security of CRS is impacted by the effectiveness of the administrative, physical, and technical security controls of its network, servers, and clients, for the purposes of this audit, I will limit my examination to CRS' ability to protect itself, its data, and its environment against threats common to web-based applications. I will be using the first nine of the Open Web Application Security Project's *Top Ten Most Critical Web Application Vulnerabilities* list as my principal guide for vulnerabilities to be audited (OWASP). I will also audit for Hidden Content as suggested by *Advanced System and Network Auditing, the* course material for the SANS Track 7 course, *Auditing Networks, Perimeters, and Systems*. (Hoelzer)

2.2. Structure

The checklists are organized by the threats described in Section 1. Each checklist step has the following elements.

Identifier	A unique code for each checklist step
Title	A very brief description of the checklist item
Reference	The source or inspiration of the checklist item
Risk	Why this checklist step is important
Test Procedure	A detailed testing procedure for the checklist item
Testing Nature	Whether this checklist item is subjective or objective
Evidence	[A placeholder for Part 3: Audit Testing, Evidence, and Findings]
Findings	[A placeholder for Part 3: Audit Testing, Evidence, and Findings]

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.3. Conventions

System commands and the results returned that are used as part of the testing are referenced in `Courier` font. References are denoted by author name or organization acronym. A full citation may be found in the References list at the end of the report.

2.4. Unvalidated Input Checklist

> Web applications use input from HTTP requests (and occasionally files) to determine how to respond. Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms. Common names for common input tampering attacks include: forced browsing, command insertion, cross-site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation. (OWASP, p.5)

Before the application responds to the HTTP request, the input or file must be broken down into its simplest form and validated. Client side validation has no security value because attackers can bypass that processing and send whatever they want. If client side validation is used, it should mirror the server side validation logic.[1]

Cross-site scripting, buffer overflows, and injection flaws are discussed in more detail below.

Table 2.1 – Unvalidated Input Checklist (UI)

Identifier	UI-1
Title	Input Validation Policy Review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	Does a policy exist that requires the application to validate all input, including the URL query string, headers, cookies, form fields, and hidden fields?
Testing Nature	Objective
Evidence	
Findings	

[1] Unvalidated Input references:
- *OWASP Guide to Building Secure Web Applications and Web Services,* Chapter 10: Data Validation prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- modsecurity project (Apache module for HTTP validation) www.modsecurity.org
- How to Build an HTTP Request Validation Engine for Your J2EE Application (J2EE validation with Stinger) www.owasp.org/columns/jwilliams/jwilliams2.html
- Have Your Cake and Eat it Too (.NET validation) www.owasp.org/columns/jpoteet/jpoteet2.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Identifier	UI-2
Title	Input Validation Design Documentation Review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	Does a design document exist requiring that valid values be verified for all input, including the URL query string, headers, cookies, form fields, and hidden fields?
Testing Nature	Objective
Evidence	
Findings	
Identifier	UI-3
Title	Input validation code review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	The simplest way to find uses of unvalidated input (tainted parameters) is to have a detailed code review, searching for all the calls where information is extracted from an HTTP request. (In a J2EE application, these are the methods in the HttpServletRequest class.) Then follow the code to see where that variable gets used. If the variable is not checked before it is used, there is very likely a problem. Parameters should be validated against a "positive" specification that defines: • Data type (string, integer, real, etc…) • Allowed character set • Minimum and maximum length • Whether null is allowed • Whether the parameter is required or not • Whether duplicates are allowed • Numeric range • Specific legal values (enumeration) • Specific patterns (regular expressions)
Testing Nature	Subjective
Evidence	
Findings	

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.5. Broken Access Control Checklist

> Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized' users are allowed to do. Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, the users may fall into a number of groups or roles with different abilities or privileges. (OWASP, p.7)

Access control, or authorization, needs to be implemented in a careful, well-designed way. If access control logic is spread out throughout the application code, it can be very difficult to validate, test, and maintain.

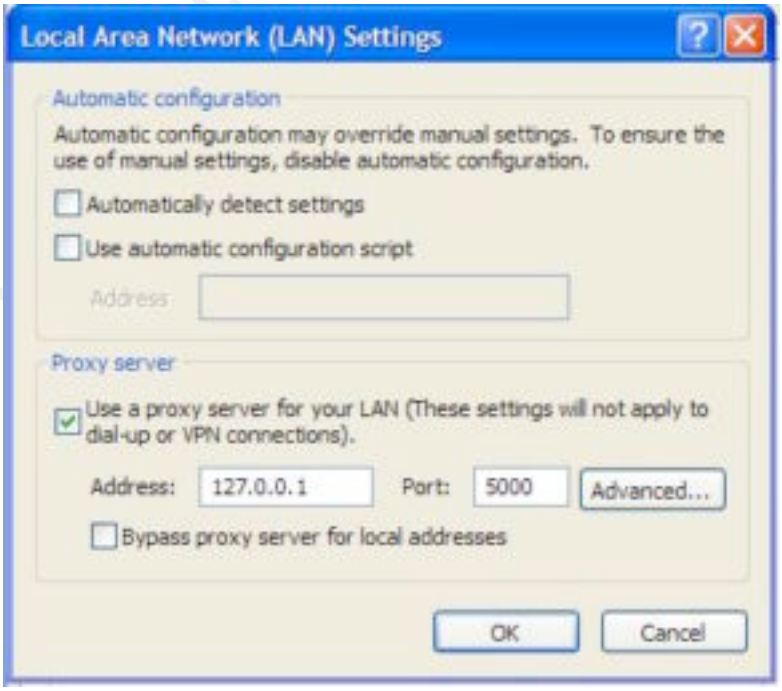
One particular type of access control problem is associated with remote administration interfaces. These interfaces could permit administrators to administer site users, data, and content from a remote location. These interfaces will be attractive targets for attackers.[2]

Table 2.2 – Broken Access Control Checklist (BA)

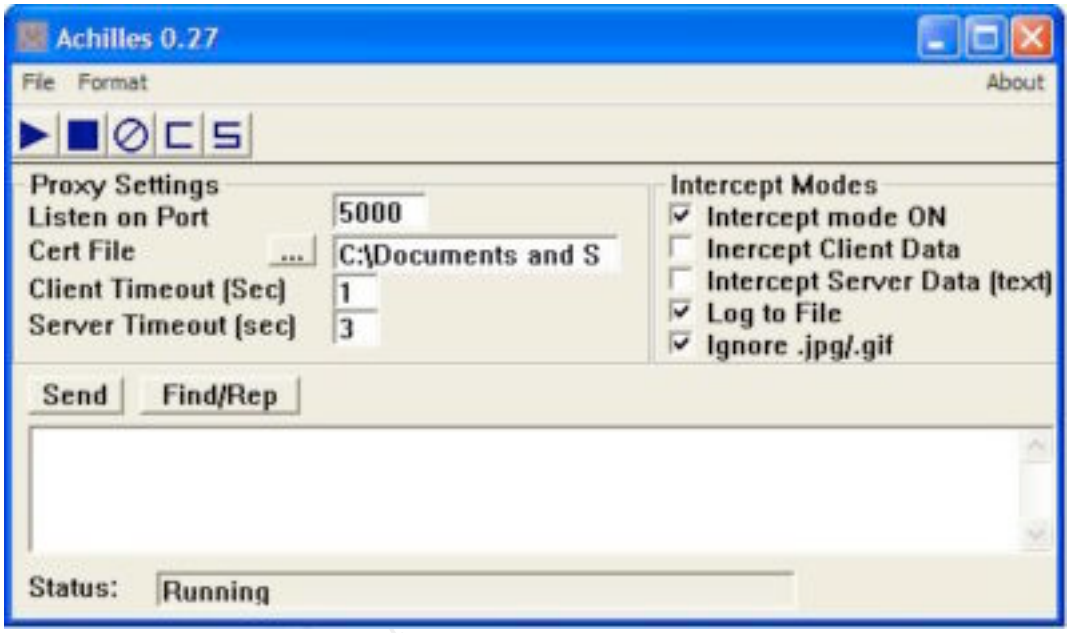
Identifier	BA-1
Title	Access Control Policy Review
Reference	OWASP Top Ten (A2)
Risk	Unauthorized access to data
Test Procedure	Does a policy exist requiring centralized access control logic and defining how it will be enforced?
Testing Nature	Objective
Evidence	
Findings	
Identifier	BA-2
Title	Access Control Design Documentation Review
Reference	OWASP Top Ten (A2)
Risk	Unauthorized access to data

[2] Broken Authentication and Session Management references:
- *OWASP Guide to Building Secure Web Applications and Web Services,* Chapter 8: Access Control and Authorization: prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- Access Control (aka Authorization) in Your J2EE Application www.owasp.org/columns/jeffwilliams/jeffwilliams3
- www.infosecuritymag.com/2002/jun/insecurity.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Test Procedure	Does a document exist that defines the centralized access control logic for the application and maintenance procedures? Do web-based administrative interfaces allow administrators to manage the application over the Internet?
Testing Nature	Objective
Evidence	
Findings	
Identifier	BA-3
Title	Check for Client Side Caching Controls
Reference	OWASP Top Ten (A2)
Risk	Unauthorized access to data
Test Procedure	Run Achilles to log HTTP and HTML for the application Set Internet Explorer's Internet Options: (images from IE 6.0) • Connections Tab: **LAN Settings** o Check **Use Proxy Server for your LAN** o **Address:** `127.0.0.1` o **Port:** `5000 (or any unused port if 5000 is in use)`  o Click **OK** twice to save settings Set Achilles options:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	• **Listen on Port:** `5000` (or whatever port you entered in the browser) • Check **Intercept mode ON** • Check **Log to File** and name the log file • Check **Ignore .jpg/.gif** • Click on **Run** ( )  Run the application as usual (the performance will be slower) • Log in • Select a screen with sensitive data • Log out Open the Log file Check for HTTP `Expires` header with a past date Check for `Pragma: no-cache` statement
Testing Nature	Objective
Evidence	
Findings	

2.6. Broken Authentication and Session Management Checklist

Authentication and session management includes all aspects of handling user authentication and managing active sessions. Authentication is a critical aspect of this process, but even solid authentication mechanisms can be undermined by flawed credential management functions, including password change, forgot

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

> my password, remember my password, account update, and other related functions. Because "walk by" attacks are likely for many web applications, all account management functions should require reauthentication even if the user has a valid Session ID. (OWASP, p.9)
>
> It has long been assumed that as long as a website or service is protected using the Secure Sockets Layer (SSL), it is secure. However, we now know that this is not true and now more than ever, it is critically important to recognize that SSL is by no means a panacea. Security researchers have long shown that the strength of any cryptographic approach is dependent on the algorithms and key lengths used by the underlying primitives. Consequently, the security of an SSL protected service is strongly correlated to the cipher suite in use as part of the protocol. (Araujo)

Most web applications rely on User ID and Password for authentication. Stronger authentication methodologies are available, but they are frequently cost prohibitive.

Maintaining state is necessary for web applications. Otherwise, the user would have to re-authenticate with each web site request. Often the method chosen to maintain state is implemented improperly, opening the site to session hijacking.

Authentication credentials and session tokens must be protected with SSL and protected against disclosure from other flaws.[3]

Table 2.3 – Broken Authentication and Session Management Checklist (AS)

Identifier	AS-1
Title	Authentication and Session Management Policy Review
Reference	OWASP Top Ten (A3)
Risk	Improper access to restricted data
Test Procedure	Does a policy exist defining how the application securely manages user credentials and session management?
Testing Nature	Objective
Evidence	
Findings	
Identifier	AS-2

[3] Broken Authentication and Session Management references:
- *OWASP Guide to Building Secure Web Applications and Web Services,* Chapter 6: Authentication and Chapter 7: Managing User Sessions: prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- White paper on the Session Fixation Vulnerability in Web-based Applications: www.acros.si/papers/session_fixation.pdf
- White paper on Password Recovery for Web-based Applications - fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

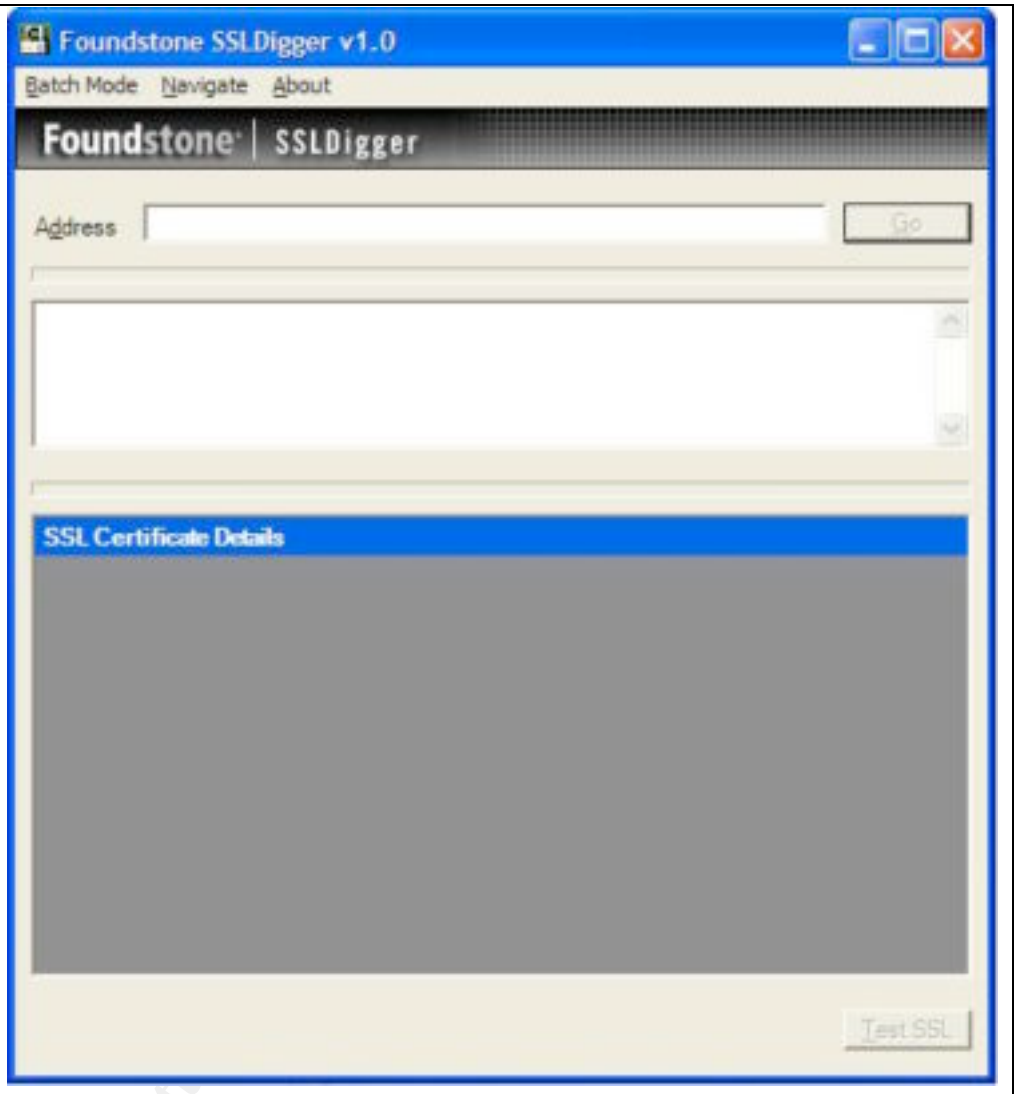
Title	Authentication and Session Management Design Review
Reference	OWASP Top Ten (A3)
Risk	Improper access to restricted data
Test Procedure	Does a design document exist defining how the application securely manages user credentials and session management? The design document should cover: • Password Strength (minimum length, complexity, periodic changes, reuse) • Password Use (lockout, logging failed attempts, failure error messages, display of last login date and time) • Password Change Controls (single mechanism; old and new passwords required; if forgotten passwords are e-mailed to users, reauthentication must be required for changing e-mail address) • Password Storage (stored in hashed or encrypted form, no hard-coded passwords in code) • Protecting Credentials in Transit (encrypt entire login transaction using SSL) • Session ID Protection (entire session protected by SSL; session IDs never included in URLs; long, complicated, random numbers for Session IDs that cannot be easily guessed) • Account Lists (not displaying account names to users) • Browser Caching (authentication and session data never submitted as part of a GET, authentication pages should be marked with all varieties of the no cache tag, autocomplete flag set to false)
Testing Nature	Objective
Evidence	
Findings	
Identifier	AS-3
Title	Authentication and Session Management Code Review
Reference	OWASP Top Ten (A3)
Risk	Improper access to restricted data
Test Procedure	Does the CRS code securely manage application user credentials and session management? Assess how the code addresses: • Password Strength (minimum length, complexity, periodic changes, reuse) • Password Use (lockout, logging failed attempts, failure error

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	messages, display of last login date and time) • Password Change Controls (single mechanism; old and new passwords required; if forgotten passwords are e-mailed to users, reauthentication must be required for changing e-mail address) • Password Storage (stored in hashed or encrypted form, no hard-coded passwords in code) • Protecting Credentials in Transit (encrypt entire login transaction using SSL) • Session ID Protection (entire session protected by SSL; session IDs never included in URLs; long, complicated, random numbers for Session IDs that cannot be easily guessed) • Account Lists (not displaying account names to users) • Browser Caching (authentication and session data never submitted as part of a GET, authentication pages should be marked with all varieties of the no cache tag, autocomplete flag set to false)
Testing Nature	Subjective
Evidence	
Findings	
Identifier	AS-4
Title	Check SSL Strength
Reference	www.foundstone.com
Risk	Disclosure of sensitive data
Test Procedure	Run SSLDigger against the web site • Open SSLDigger application

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	 • Enter URL of the site to be audited in the Address field • Click Go • Answer Yes to save and view the report • Specify where report file should be stored • Answer Yes again to view the report Display **File** | **Properties** for the site home page • Check the Connection information • 128-bit encryption or better is required
Testing Nature	Objective
Evidence	
Findings	

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.7. Cross-site Scripting (XSS) Flaws Checklist

> Cross-site scripting (sometimes referred to as XSS) vulnerabilities occur when an attacker uses a web application to send malicious code, generally in the form of a script, to a different end user. These flaws are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating it.
>
> An attacker can use cross-site scripting to send malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page. (OWASP, p.11)

In addition to the validation audits using the Unvalidated Input checklists, one of the most effective protections is to convert certain characters to prevent executable script from being sent to the browser.

Table 2.4 – Cross-site Scripting (XSS) Flaws Checklist (XS)

Identifier	XS-1
Title	Output Encoding
Reference	OWASP Top Ten (A4)
Risk	Loss of user confidence and undetected malicious input
Test Procedure	Check output routines for the presence of output encoding routines converting the following characters: **From** → **To** < → < > → > (→ ( ) →) # → # & → &
Testing Nature	Objective
Evidence	
Findings	

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.8. Buffer Overflows Checklist

> Attackers use buffer overflows to corrupt the execution stack of a web application. By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine. Buffer overflows are not easy to discover and even when one is discovered, it is generally extremely difficult to exploit. Nevertheless, attackers have managed to identify buffer overflows in a staggering array of products and components. (OWASP, p.13)

Buffer Overflow flaws can be found in the web and application server software and these flaws become widely known, but they are outside of the scope of this audit. We are concerned here with buffer overflow flaws that may be found in CRS. However, Java and J2EE applications are not vulnerable to buffer overflow attacks.[4]

Table 2.5 – Buffer Overflows Checklist (BO)

Identifier	BO-1
Title	Input Length Edits
Reference	OWASP Top 10 (A5)
Risk	Corruption of application execution
Test Procedure	If the application is not a Java or J2EE application (Java and J2EE applications are not vulnerable to buffer overflow attacks), review all code that accepts input using HTTP requests for size checking for all such input
Testing Nature	Objective
Evidence	
Findings	

2.9. Injection Flaws Checklist

> Injection flaws allow attackers to relay malicious code through a web application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). Whole scripts written in perl, python, and other languages can be injected into poorly designed web applications and executed. Any time a web application uses an interpreter of any type there is a danger of an injection attack. (OWASP, p.14)

[4] Buffer Overflow references:
- *OWASP Guide to Building Secure Web Applications and Web Services,* Chapter 10: Data Validation prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- Aleph One, "Smashing the Stack for Fun and Profit", www.phrack.com/show.php?p=49&a=14
- Mark Donaldson, "Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention", rr.sans.org/code/inside_buffer.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

If a web application uses operating system functions or external programs to perform its functions, and passes http input to the external function, the http input must be carefully scrubbed. Otherwise, the application may be open to injection flaws. The consequences could range from the trivial to complete system compromise or destruction. Since the use of external functions is common, the likelihood of the web application having an injection flaw should be considered very high.[5]

Table 2.6 – Injection Flaws Checklist (IF)

Identifier	IF-1
Title	Injection Flaws Code Review
Reference	OWASP Top Ten (A6)
Risk	Disclosure of sensitive data to complete takeover of the web site
Test Procedure	1. Review all code that calls outside programs or interpreters • Verify that all supplied data is properly validated to protect against malicious input • Verify that all calls to the backend database use stored procedures 2. Check to see that application is not running at a higher privilege level than necessary • For J2EE applications, verify that the application is using the Java sandbox.
Testing Nature	Subjective
Evidence	
Findings	

2.10. Improper Error Handling Checklist

Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker). These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on

[5] Injection Flaws references:
- **Examples:** A malicious parameter could modify the actions taken by a system call that normally retrieves the current user's file to access another user's file (e.g., by including path traversal "../" characters as part of a filename request). Additional commands could be tacked on to the end of a parameter that is passed to a shell script to execute an additional shell command (e.g., "; rm –r *") along with the intended command. SQL queries could be modified by adding additional 'constraints' to a where clause (e.g., "OR 1=1") to gain access to or modify unauthorized data.
- *OWASP Guide to Building Secure Web Applications and Web Services,* Chapter 10: Data Validation prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- *How to Build an HTTP Request Validation Engine* (J2EE validation with Stinger) www.owasp.org/columns/jeffwilliams/jeffwilliams2
- *Have Your Cake and Eat it Too* (.NET validation) www.owasp.org/columns/jpoteet/jpoteet2

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

> potential flaws in the site and such messages are also disturbing to normal users. (OWASP, p.16)

There are many error conditions that occur unavoidably, such as out of memory or resource unavailable conditions. In these cases, the error messages should follow a carefully thought out format, with helpful information to the user, useful information to the administrator or developer, and *no* meaningful information to an attacker.

One common error-handling problem occurs when the application "fails open." Access to system and application resources should always be denied until specifically granted (application will "fail closed"), access should never be granted until specifically denied (i.e., "fail open").[6]

Table 2.7 – Improper Error Handling Checklist (IE)

Identifier	IE-1
Title	Error Handling Policy Review
Reference	OWASP Top Ten (A7)
Risk	Disclosure of application implementation details
Test Procedure	Is there a policy that documents how errors should be handled? Does it require that error messages reveal only necessary information?
Testing Nature	Objective
Evidence	
Findings	
Identifier	IE-2
Title	Sign On Error Testing
Reference	OWASP Top Ten (A7)
Risk	Disclosure of valid User IDs – useful for harvesting User IDs
Test Procedure	Enter bad User ID and Password, valid User ID and bad password • Time the responses to see if they are the same • Check error responses to see if they are identical, including the HTML code
Testing Nature	Objective
Evidence	
Findings	

[6] Improper Error Handling references:
- OWASP discussion on generation of error codes: www.owasp.org/documentation/guide/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Identifier	IE-3
Title	Consistent Error Handling Code Review
Reference	OWASP Top Ten (A7)
Risk	Disclosure of application implementation details
Test Procedure	Check the code for consistent and appropriate error handling and error messages
Testing Nature	Subjective
Evidence	
Findings	

2.11. Insecure Storage Checklist

> Most web applications have a need to store sensitive information, either in a database or on a file system somewhere. The information might be passwords, credit card numbers, account records, or proprietary information. Frequently, encryption techniques are used to protect this sensitive information. While encryption has become relatively easy to implement and use, developers still frequently make mistakes while integrating it into a web application. Developers may overestimate the protection gained by using encryption and not be as careful in securing other aspects of the site. (OWASP, p.18)

Some common instances of insecure storage are:

- Failure to encrypt critical data
- Insecure storage of keys, certificates, and passwords
- Improper storage of secrets in memory
- Poor sources of randomness
- Poor choice of algorithm
- Attempting to invent a new encryption algorithm
- Failure to include support for encryption key changes and other required maintenance procedures

Encryption is normally used to protect a site's most sensitive assets. A failure here can have devastating consequences.[7]

Table 2.8 – Insecure Storage Checklist (IS)

[7] Insecure Storage references:
- *OWASP Guide to Building Secure Web Applications and Web Services* prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download
- Bruce Schneier, "Applied Cryptography", 2nd edition, John Wiley & Sons, 1995.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Identifier	IS-1
Title	Secure Storage Policy Review
Reference	OWASP Top Ten (A8)
Risk	Disclosure of sensitive data
Test Procedure	Review the policy to see if it requires appropriate protection of sensitive information. • Does it require the use of encryption algorithms that are strong and that have been publicly tested?
Testing Nature	Objective
Evidence	
Findings	
Identifier	IS-2
Title	Secure Storage Design Specification Review
Reference	OWASP Top Ten (A8)
Risk	Disclosure of sensitive data
Test Procedure	Check the design specification for the requirement to protect sensitive information. • Verify that it specifies encryption algorithms that are strong and that have been publicly reviewed.
Testing Nature	Objective
Evidence	
Findings	
Identifier	IS-3
Title	Secure Storage Code Review
Reference	OWASP Top Ten (A8)
Risk	Disclosure of sensitive data
Test Procedure	Review source code to see how the cryptographic functions are implemented. Check how passwords, keys, and other sensitive information is: • Stored • Protected • Loaded • Processed • Cleared from memory

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	Information must be protected using strong encryption methods
Testing Nature	Subjective
Evidence	
Findings	

2.12. Denial of Service Checklist

> Web applications are particularly susceptible to denial of service (DoS) attacks._…_A web application can't easily tell the difference between an attack and ordinary traffic. There are many factors that contribute to this difficulty, but one of the most important is that, for a number of reasons, IP addresses are not useful as an identification credential. Because there is no reliable way to tell where an HTTP request is from, it is very difficult to filter out malicious traffic. For distributed attacks, how would an application tell the difference between a true attack, multiple users all hitting reload at the same time (which might happen if there is a temporary problem with the site), or getting "slashdotted"[8]? (OWASP, p.20)

It is not difficult to overwhelm application resources (e.g., bandwidth, disk storage, database connections, CPU, memory, or threads). There is a wide variety of these attacks, and they can be easily launched. While it may not always be possible to prevent such an attack, the application should make a successful attack as difficult as possible.

DoS attacks against the network, such as SYN flood attacks, are beyond the scope of this audit.[9]

Table 2.9 – Denial of Service Checklist (DS)

Identifier	DS-1
Title	Denial of Service Code Review
Reference	OWASP Top Ten (A9)
Risk	Exhaustion of application resources
Test Procedure	Check to see if requests are synchronized on the Session ID • Are there limits on the resources that can be allocated for a single session? • Are there limits on the number of sessions per user?

[8] "The Slashdot effect is the huge influx of Internet traffic to a website as a result of its being mentioned on Slashdot, a popular technology news and information site." (en.wikipedia.org/wiki/Slashdotted)

[9] DoS references:
- *OWASP Guide to Building Secure Web Applications and Web Services* prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	• Is a session limited to one request at a time? • Does the application cancel prior requests when it receives a new request from the same session?
Testing Nature	Objective
Evidence	
Findings	

2.13. Hidden Content

The primary danger with hidden content is revealing information that could be useful to an attacker.

Table 2.10 – Hidden Content Checklist (HC)

Identifier	HC-1
Title	Hidden Content Policy Review
Reference	Auditing Web Servers and Applications (Rhoades, pp.90–103)
Risk	Disclosure of unnecessary system information
Test Procedure	Review development policies for the requirement to limit hidden content
Testing Nature	Objective
Evidence	
Findings	
Identifier	HC-2
Title	Hidden Content Application Specifications Review
Reference	Auditing Web Servers and Applications (Rhoades, pp.90–103)
Risk	Disclosure of unnecessary system information
Test Procedure	Review application specifications for the requirement to limit hidden content • Are comments required to be removed from production code? • Is unnecessary information restricted?
Testing Nature	Objective
Evidence	
Findings	

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Identifier	HC-3
Title	Hidden Content Code Review
Reference	Auditing Web Servers and Applications (Rhoades, pp.90–103)
Risk	Disclosure of unnecessary system information
Test Procedure	Review the output from audit checklist step BA-3 • Check for comments or meta tags in client-side code • Check for unnecessary information in server HTTP headers and custom headers
Testing Nature	Subjective
Evidence	
Findings	

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3. Audit Testing, Evidence, and Findings

The CRS application was audited using the checklists provided above. In order to keep this document to a reasonable length, I have included only ten checklist items. I have attempted to select those items that provide the most useful information. Seven of the included checklist items involve code reviews. This is a time consuming and tedious auditing method, but, because of the limited number of tools available to me to audit web applications, I found it necessary to examine the code to determine the security state of the application.

If it had been available to me, I would have run WebInspect using the OWASP Top Ten policy to try tests against each of the listed vulnerabilities (see Appendix A). The use of this or a similar commercial tool would have made this a stronger audit.

I have included findings for the following checklist items:

- UI-1 – Input Validation Policy Review
- UI-2 – Input Validation Design Documentation Review
- UI-3 – Input Validation Code Review
- BA-3 – Check for Client Side Caching Controls
- AS-3 – Authentication and Session Management Code Review
- AS-4 – Check SSL Strength
- XS-1 – Output Encoding
- IE-2 – Sign On Error Testing
- IE-3 – Consistent Error Handling Code Review
- IS-3 – Secure Storage Code Review

3.1. Unvalidated Input Checklist

Table 3.1 – Unvalidated Input Checklist (UI)

Identifier	UI-1
Title	Input Validation Policy Review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	Does a policy exist that requires the application to validate all input, including the URL query string, headers, cookies, form fields, and hidden fields?
Testing Nature	Objective
Evidence	No overall development or coding policies of any kind were found.
Findings	No input validation policy exists for CRS.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Identifier	UI-2
Title	Input Validation Design Documentation Review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	Does a design document exist requiring that valid values be verified for all input, including the URL query string, headers, cookies, form fields, and hidden fields?
Testing Nature	Objective
Evidence	Project documentation was reviewed, including the Standard Operating Procedures used to develop CRS. The SOP on Standard Development Life Cycle Planning mentions Security, but its focus was limited to the functional ability to limit data and program access to authorized personnel and to protect the integrity of CRS data by logging all changes. No mention was made of the necessity to validate input data. The SOP on Security requires User ID and Password authentication. Authorization is required to be "user/role/context based." The security interests are confined to data privacy and integrity. Data validity is not mentioned. The SOP on Physical and Network Security simply says that CRS must comply with the proposed HIPAA security regulations. Security procedures are simply, "To comply with all [hospital] procedures for security and the privacy of patient information."
Findings	No input validation design specifications exist for CRS.
Identifier	UI-3
Title	Input Validation Code Review
Reference	OWASP Top Ten (A1)
Risk	Undetected malicious input
Test Procedure	The simplest way to find uses of unvalidated input (tainted parameters) is to have a detailed code review, searching for all the calls where information is extracted from an HTTP request. (In a J2EE application, these are the methods in the HttpServletRequest class.) Then follow the code to see where that variable gets used. If the variable is not checked before it is used, there is very likely a problem. Parameters should be validated against a "positive" specification that defines: • Data type (string, integer, real, etc…) • Allowed character set • Minimum and maximum length

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	• Whether null is allowed • Whether the parameter is required or not • Whether duplicates are allowed • Numeric range • Specific legal values (enumeration) • Specific patterns (regular expressions)
Testing Nature	Subjective

Evidence	CRS validates input, using client-side Java Script, for:
	• Data Type
	• Allowed character set – edits use the following regular expressions:
	o Alphabetic: (`/[^A-Za-z]/`)
	o Alphanumeric: (`/[^A-Za-z0-9]/`)
	o Alphanumeric and underscore: (`/[^A-Za-z0-9_]/`)
	o Alphanumeric and dash: (`/[^A-Za-z0-9-]/`)
	o Alphanumeric, period, single quote, and space: (`/[^A-Za-z0-9\.'\s]/`)
	o E-mail (alphanumeric, @, period, underscore, and required format: (`/[a-z0-9_\-\.]+@[a-z0-9_\-\.]+\.[a-z]{2,3}$/i`)
	• Minimum and maximum length
	• Whether the field is required or not
	• Where feasible, the field values are selected from drop-down lists
	For example, the following client-side code calls edit routines based on the element type of each form element.
	
	`Switch (strElementType) {`
	`case "typeAlpha" : strReturnValue = IsAlphabet(elElement); break;`
	`case "typeNum" : strReturnValue = IsNumber(elElement); break;`
	`case "typeString" : strReturnValue = IsString(elElement); break;`
	`case "typeDate" : strReturnValue = IsValidDate(elElement, false); break;`
	`case "typeDob" : strReturnValue = IsValidDob(elElement); break;`
	`case "typeName" : strReturnValue = IsValidName(elElement); break;`
	`case "typeSSN" : strReturnValue = IsValidSSN(elElement); break;`

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
          case "typeEMail" : strReturnValue =
IsValidEMail(elElement); break;
          case "typeRadio" : strReturnValue =
IsRadioSelected(elElement); break;
          case "typeUrl" : strReturnValue =
IsValidUrl(elElement); break;
          case "typeAtLeastOne" : strReturnValue =
AtLeastOne(elElement, frmForm); break; }
```

Examples of the called edit routines are:

```
function IsValidName(elElement) {
var strReturnValue = "";
var strValue = elElement.value;
var strExp = /[^A-Za-z0-9.\-'\s]/;
if (strValue.search(strExp) >= 0) {
     var DisplayName = elElement.displayName;
     if (!DisplayName)
          strReturnValue = "Field";
     else
          strReturnValue = DisplayName;
     strReturnValue += " can only contain alphanumeric
characters, period(.), hyphen(-), space or single
quotes."; }
return strReturnValue; }

function IsAlphabet(elElement) {
var strReturnValue = "";
var strValue = elElement.value;
var strExp = /[^A-Za-z]/;
if (strValue.search(strExp) >= 0)
     strReturnValue = "Field can only contain
alphabets.";
return strReturnValue; }
```

Both by interview and by code inspection, I verified that there is no server-side data validation except for User ID and Password.

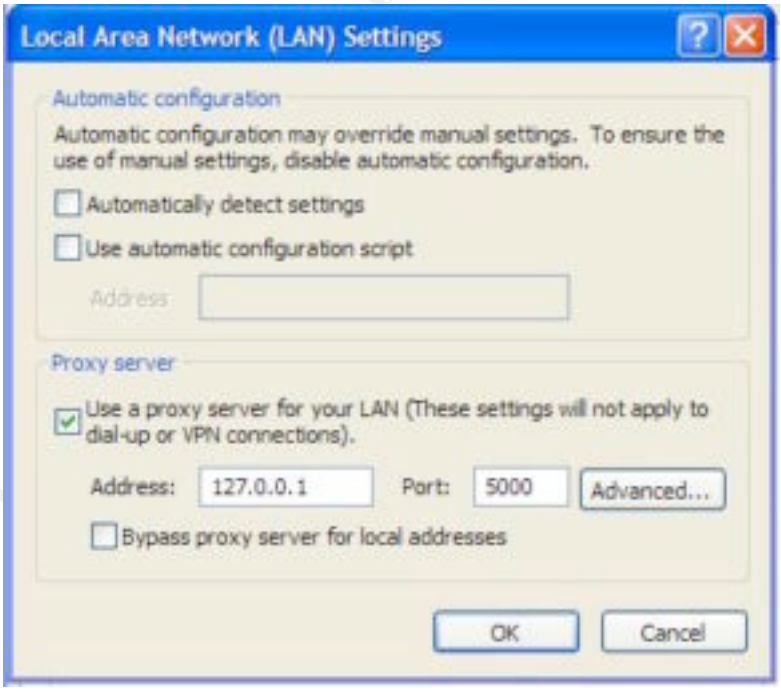
Findings	Client-side data validation offers no security benefits because it is very easy to bypass. Since there is no server-side data validation, there is no effective protection against malicious input.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

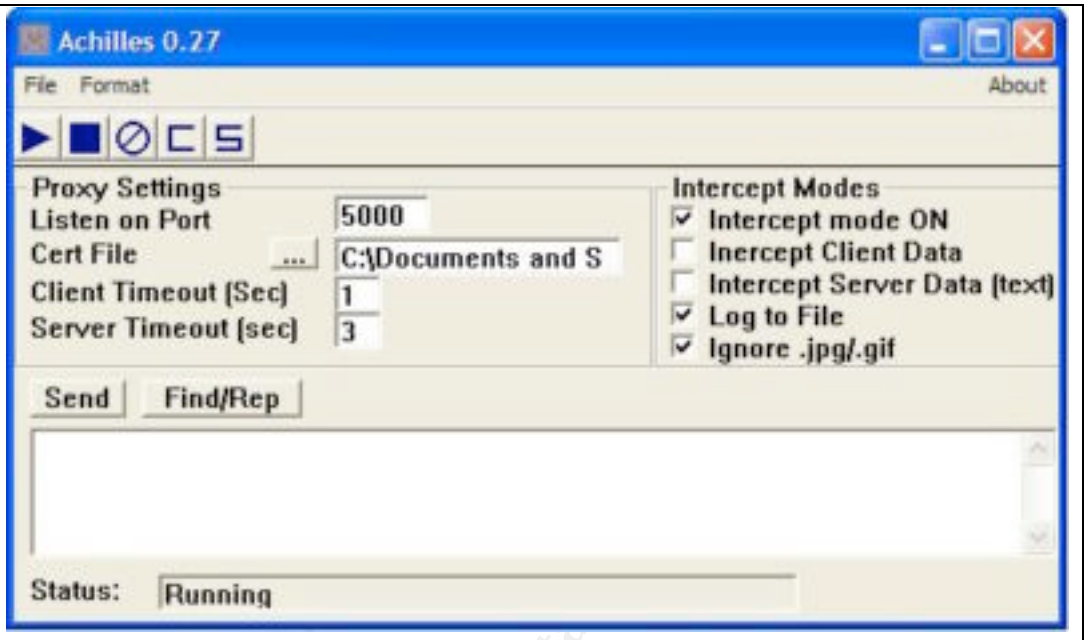
3.2. Broken Access Control Checklist

Table 3.2 – Broken Access Control Checklist (BA)

Identifier	BA-3
Title	Check for Client Side Caching Controls
Reference	OWASP Top Ten (A2)
Risk	Unauthorized access to data

Test Procedure	Run Achilles to log HTTP and HTML for the application
	Set Internet Explorer's Internet Options: (images from IE 6.0)
	• Connections Tab: **LAN Settings**
	  o Check **Use Proxy Server for your LAN**
	  o **Address:** `127.0.0.1`
	  o **Port:** `5000 (or any unused port if 5000 is in use)`
	
	
	  o Click **OK** twice to save settings
	Set Achilles options:
	• **Listen on Port:** `5000` (or whatever port you entered in the browser)
	• Check **Intercept mode ON**
	• Check **Log to File** and name the log file
	• Check **Ignore .jpg/.gif**
	• Click on **Run** (▶)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

<table>
<tr><td></td><td>


Run the application as usual (the performance will be slower)
• Log in
• Select a screen with sensitive data
• Log out
Open the Log file
Check for HTTP <code>Expires</code> header with a past date
Check for <code>Pragma: no-cache</code> statement</td></tr>
<tr><td>Testing Nature</td><td>Objective</td></tr>
<tr><td>Evidence</td><td>In a response from the server that could contain sensitive data, the following HTTP header was received:
<pre>

POST . . . HTTP/1.0
Accept: . . .
Referer: . . .
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
User-Agent: . . .
Host: . . .
Content-Length: 140
Cache-Control: no-cache
Cookie: . . .
HEADER_SELECTION= . . .
</pre></td></tr>
</table>

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	`----------`
Findings	No `Pragma: no-cache` statement was found in HTML. One POST HTTP submission contained a `Pragma: no-cache` statement, but no HTTP response header contained it. CRS sent a `Cache-Control: no-cache` response header when replying to the POST transaction with the `Pragma: no-cache` statement. According to Mark Nottingham, a `Cache-Control: no-cache` statement has the following effect: > **no-cache** – forces caches to submit the request to the origin server for validation before releasing a cached copy, every time. This is useful to assure that authentication is respected (in combination with public), or to maintain rigid freshness, without sacrificing all of the benefits of caching. (Nottingham) Since the cache only validates the page before redisplaying it, and does not obtain new content, the sensitive patient content must still be cached on the local disk, and thus is susceptible to being viewed by others. The CRS cache control is not as secure as it would be if the header contained `Cache-Control: no-store` or an `Expires` header with a past date.

3.3. Broken Authentication and Session Management Checklist

Table 3.3 – Broken Authentication and Session Management Checklist (AS)

Identifier	AS-3
Title	Authentication and Session Management Code Review
Reference	OWASP Top Ten (A3)
Risk	Improper access to restricted data
Test Procedure	Does the CRS code securely manage application user credentials and session management? Assess how the code addresses: • Password Strength (minimum length, complexity, periodic changes, reuse) • Password Use (lockout, logging failed attempts, failure error messages, display of last login date and time) • Password Change Controls (single mechanism; old and new passwords required; if forgotten passwords are e-mailed to users, reauthentication must be required for changing e-mail address) • Password Storage (stored in hashed or encrypted form, no hard-coded passwords in code) • Protecting Credentials in Transit (encrypt entire login transaction using SSL)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

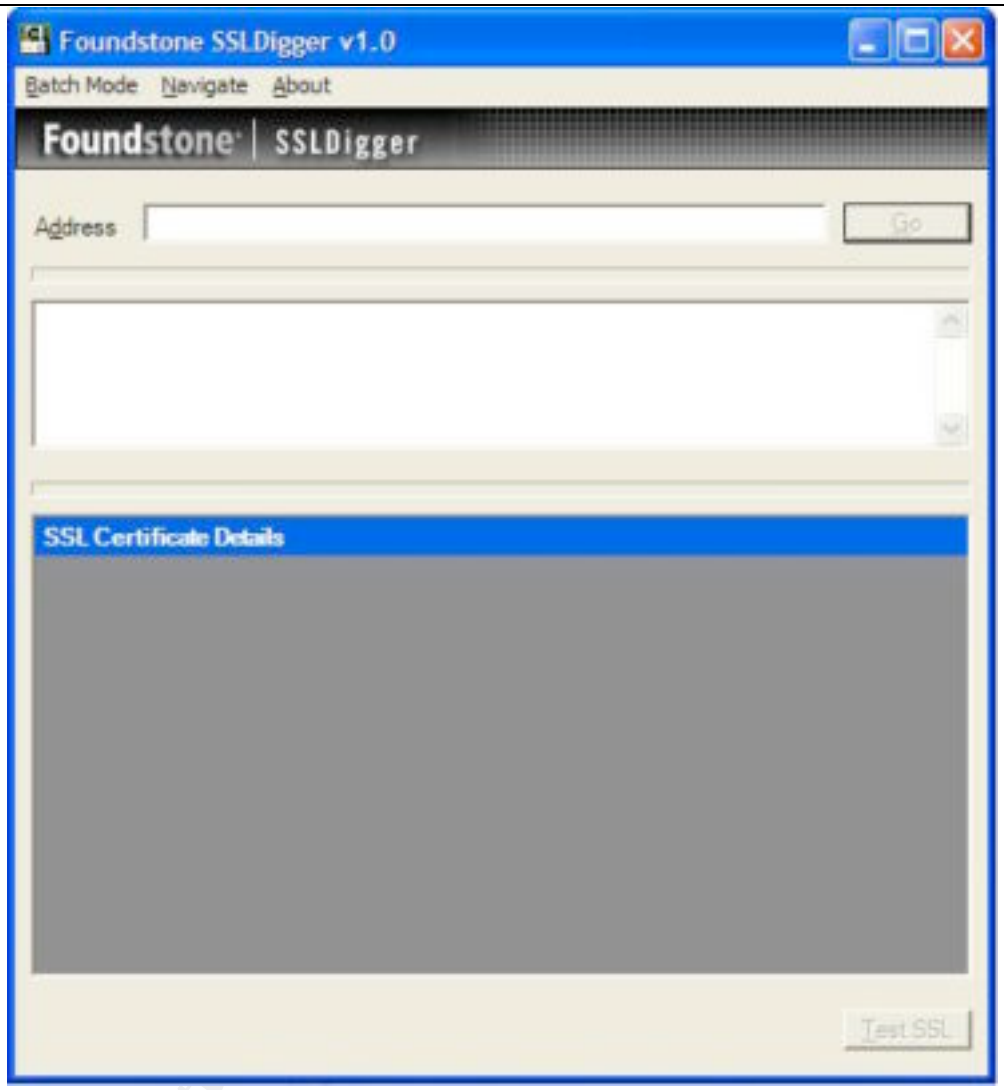
	• Session ID Protection (entire session protected by SSL; session IDs never included in URLs; long, complicated, random numbers for Session IDs that cannot be easily guessed) • Account Lists (not displaying account names to users)
Testing Nature	Subjective

Evidence	Some password issues are user configurable in a configuration file (sample values are shown):
	`PASSWORD_DURATION=90`
	`LOGIN_ATTEMPTS_ALLOWED=3`
	`ALPHA_CHARACTER_REQUIRED=yes`
	`SPECIAL_CHARACTER_REQUIRED=yes`
	`SESSION_INACTIVITY_THRESHOLD=500`
	`USERNAME_LENGTH_MIN=8`
	`USERNAME_LENGTH_MAX=99`
	`PASSWORD_LENGTH_MIN=7`
	`PASSWORD_LENGTH_MAX=20`
	CRS uses DESEncrypt from Oracle's DBMS Obfuscation Toolkit to store passwords. Unfortunately, Oracle's DBMS Obfuscation Toolkit only supports DES(56) encryption.
	
	`function desencrypt`
	`(p_input_string varchar2) return varchar2 is`
	`v_input_string varchar2(2048);`
	`v_encrypted_string varchar2(2048);`
	`v_multiple number;`
	`begin`
	`v_multiple := trunc((length(p_input_string) / 8) + 1);`
	`v_input_string := rpad(p_input_string, 8 * v_multiple, '@');`
	`dbms_obfuscation_toolkit.DESEncrypt(`
	`input_string => v_input_string,`
	`key_string => key_string,`
	`encrypted_string => v_encrypted_string );`
	`return v_encrypted_string;`
	`end desencrypt;`
	
	Any user with administrative access (Data Administrator or Study Administrator roles) can display a list of all users on CRS, including the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	User ID (*User History Report* and *Role User History Report*). 
Findings	The password change process is protected by the use of SSL (but see AS-4 below). DES(56) encryption of passwords on the server in not secure enough to protect this sensitive data. > NIST has determined that the strength of the Data Encryption Standard (DES) algorithm, as specified in Federal Information Processing Standard (FIPS) 46-3, (including Triple-DES keying option 3 (1 key Triple-DES)) is no longer sufficient to adequately protect Federal government information. (NIST) In 1999, NIST recommended moving from DES to Triple-DES as an interim measure while the new Advanced Encryption Standard (AES) was being developed. In 2004, DES is inadequate for all uses. User ID harvesting is easy for any user with Data Administrator or Study Administrator roles. It is potentially available for any attacker who upgrades their rights to an Administrator role. Authentication and Session Management are vulnerable to attack.
Identifier	AS-4
Title	Check SSL Strength
Reference	www.foundstone.com
Risk	Disclosure of sensitive data
Test Procedure	Run SSLDigger against the web site • Open SSLDigger application

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	 • Enter URL of the site to be audited in the Address field • Click Go • Answer Yes to save and view the report • Specify where report file should be stored • Answer Yes again to view the report Display **File** | **Properties** for the site home page • Check the Connection information • 128-bit encryption or better is required
Testing Nature	Objective
Evidence	**SSL Cipher Strength Report** **Saturday, October 30, 2004 11:46:45 AM** ***Summary***

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Number of Servers Tested	1
Number of Ciphers Used in Testing	26
No Security	2
Weak Security	10
Strong Security	9
Excellent Security	5

Ciphers Supported

Server URL	No Security	Weak Security	Strong Security	Excellent Security	Grade
https://xxx.xxxxxx.xxx:9999	0	3	0	0	C

Detailed Results (only Supported rows are shown)

Server: https://xxx.xxxxxx.xxx:9999

Grade	C	
Certificate Details	Server Gated Cryptography	Netscape SGC

Ciphers

OpenSSL Name	Display Name	Export Grade?	Strength	Supported?
EXP-DES-CBC-SHA	Key Exchange: RSA(512); Authentication: RSA; Encryption: DES(40); MAC: SHA1	true	Weak Security	true
EXP-RC4-MD5	Key Exchange: RSA(512); Authentication: RSA; Encryption: RC4(40); MAC: MD5	true	Weak Security	true
DES-CBC-SHA	Key Exchange: RSA; Authentication: RSA; Encryption: DES(56); MAC: SHA1	false	Weak Security	true

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	
Findings	The CRS SOP on Security requires 128-bit security. Although CRS appears to support only 3 weak ciphers, the site has a Server Gated Cryptography or Global ID server certificate. Use of such a certificate generally implies that a minimum of 128-bit encryption will be used. However, when I displayed the page properties, the connection information was: SSL 3.0, DES with 56 bit encryption (Medium); RSA with 512 bit exchange CRS appears to be using 56-bit DES, a weak encryption level. (See the note on DES in AS-3 Findings.) The DES(56) cipher is far below the specified 128-bit minimum encryption level for CRS. The audit finding is that weak SSL encryption is used by CRS.

3.4. Cross-site Scripting (XSS) Flaws Checklist

Table 3.4 – Cross-site Scripting (XSS) Flaws Checklist (XS)

Identifier	XS-1
Title	Output Encoding
Reference	OWASP Top Ten (A4)
Risk	Loss of user confidence and undetected malicious input
Test Procedure	Check output routines for the presence of output encoding routines converting the following characters: **From** → **To** < → < > → > (→ ( ) →) # → # & → &

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Testing Nature	Objective
Evidence	A search of all application code turned up one service routine that converted "<" and ">" characters into "`<`" and "`>`" respectively. However, this service routine was never called by any other code. No other conversion was found in CRS code.
Findings	No protection is provided for the user from embedded executable code in responses sent to the client.

3.5. Improper Error Handling Checklist

Table 3.5 – Improper Error Handling Checklist (IE)

Identifier	IE-2
Title	Sign On Error Testing
Reference	OWASP Top Ten (A7)
Risk	Disclosure of valid User IDs – useful for harvesting User IDs
Test Procedure	Enter bad User ID and Password, valid User ID and bad password • Time the responses to see if they are the same • Check error responses to see if they are identical, including any hidden content
Testing Nature	Objective
Evidence	With a bad User ID and Password, the response took 91 seconds. The error message was,  With a valid User ID and a bad Password, the response took 91 seconds. The error message was, 
Findings	While I might quibble with a 91-second response for an invalid logon attempt as being unfriendly to users, there was no difference in timing between the bad User ID and Password and the valid User ID and bad Password. In addition, no information was provided that would assist an attempt to harvest User IDs.
Identifier	IE-3
Title	Consistent Error Handling Code Review
Reference	OWASP Top Ten (A7)
Risk	Disclosure of application implementation details

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Test Procedure	Check the code for consistent and appropriate error handling and error messages
Testing Nature	Subjective
Evidence	The Error Page includes the system logo. OnMouseOver, debug error information is displayed. OnMouseOut the debug error information is hidden. `<td class="FormText" align="center" height="32"><img src="/web_crs/images/Logor=Crs.gif" width="256" height="20" onMouseOver="MM_showHideLayers('errortext','', 'show')" alt="Crs Logo" onMouseOut="MM_showHideLayers('errortext','', 'hide')">`
Findings	Too much information is displayed to the user if they move the mouse over the system logo. The information is of no use to a legitimate user, but it provides too much system information to a malicious user. This capability should be removed from production code.

3.6. Insecure Storage Checklist

Table 3.6 – Insecure Storage Checklist (IS)

Identifier	IS-3
Title	Secure Storage Code Review
Reference	OWASP Top Ten (A8)
Risk	Disclosure of sensitive data
Test Procedure	Review source code to see how the cryptographic functions are implemented. Check how passwords, keys, and other sensitive information is: • Stored • Protected • Loaded • Processed • Cleared from memory Information must be protected using strong encryption methods
Testing Nature	Subjective
Evidence	CRS uses SSL to protect authentication information in transit. Unfortunately, the encryption level used is DES(56).

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

CRS uses DESEncrypt and DESDecrypt from Oracle's DBMS Obfuscation Toolkit to store and retrieve passwords. Unfortunately, Oracle's DBMS Obfuscation Toolkit only supports DES(56) encryption.

```
procedure add_user (p_app_user users.username%type
 ,p_username users.username%type
 ,p_last_name users.last_name%type
 ,p_first_name users.first_name%type
 ,p_middle_name users.middle_name%type
 ,p_title users.title%type
 ,p_display_name users.display_name%type
 ,p_email_address users.email_address%type
 ,p_password users.password%type) is
v_default_study_id number;
begin
    util_pkg.check_privilege(p_app_user,
'COMMUNITY', 1, null, 'Add-Edit Users');
    select default_study_id
    into v_default_study_id
    from users
    where username = p_app_user;

    insert into users (
        username,
        last_name,
        first_name,
        middle_name,
        title,
        display_name,
        email_address,
        password,
        default_study_id,
        created_by,
        date_created)
    values (
        p_username,
        p_last_name,
        p_first_name,
        p_middle_name,
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	p_title,
	p_display_name,
	p_email_address,
	util_pkg.desencrypt(p_password),
	v_default_study_id,
	p_app_user,
	sysdate);
	end add_user;
	
	function desencrypt (p_input_string varchar2)
	return varchar2 is
	v_input_string varchar2(2048);
	v_encrypted_string varchar2(2048);
	v_multiple number;
	begin
	v_multiple := trunc((length(p_input_string) / 8) + 1);
	v_input_string := rpad(p_input_string, 8 * v_multiple, '@');
	
	dbms_obfuscation_toolkit.DESEncrypt(
	input_string => v_input_string,
	key_string => key_string,
	encrypted_string => v_encrypted_string);
	return v_encrypted_string;
	end desencrypt;
	There is no code to clear sensitive information from memory after use. The cache direction, Cache-Control: no-cache does not offer enough protection against caching sensitive information.

Findings	DES(56) encryption for SSL and the protection of passwords on the server in not secure enough to protect this sensitive data. NIST has determined that the strength of the Data Encryption Standard (DES) algorithm, as specified in Federal Information Processing Standard (FIPS) 46-3, (including Triple-DES keying option 3 (1 key Triple-DES)) is no longer sufficient to adequately protect Federal government information. (NIST) In 1999, NIST recommended moving from DES to Triple-DES as an interim measure while the new Advanced Encryption Standard (AES) was being developed. In 2004, DES is inadequate for all uses. No code clears sensitive information from memory after use, and the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	cache control is inadequate.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4. Audit Report

4.1. Executive Summary

"The Gartner Group estimates that 70 percent of computer attacks are now aimed not at individual networks, but at the applications that run on them." (Roberts) Networks may be secure, but most network security controls will not prevent attacks on Internet applications because they see them as legitimate network traffic. The application itself must determine if input data is valid and safe.

This is an audit of the Collaborative Research System (CRS), an Internet-based application used for human studies medical research. The primary objective was to assess the ability of CRS to protect patient and research information from outside attacks against the most common Web application vulnerabilities.

The focus of this audit is the application itself. The security of the server and network are critical to the security of the application, but they are outside the scope of this audit.

The audit was conducted using available freeware or shareware tools and visual inspection of the application's development policies, design documentation, and source code. The tools used were:

- **SSLDigger** – Foundstone developed SSLDigger to assess the strength of SSL ciphers. (www.foundstone.com – look under Resources | Free Tools | S3i Tools)
- **Achilles** – Available from achilles.mavensecurity.com, Achilles is a publicly released general-purpose web application security assessment tool. Achilles acts as a HTTP/HTTPS proxy that allows a user to intercept, log, and modify web traffic on the fly.
- **Site Inspector** – Available from www.paessler.com, Site Inspector is a Web site and Web page analysis tool. It is an extension to IE 5 or later. Right-clicking on a web page opens a context menu which displays Site Inspector options, including:
    - Show all forms (including hidden form fields)
    - Show all scripts
    - Show HTTP Header
    - Show Complete Page Analysis
    - Show Source based on DOM
- **Multi-Edit** – I used Multi-Edit as a code editor to view and search the source code. The code was J2EE Java and JavaScript. I also used Multi-Edit to view the Achilles log output because it formatted the log file, making it easy to follow.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Multi-Edit is available for a free 30-day trial from Multi Edit Software, Inc., at www.multiedit.com.

In planning this audit, I decided to use the OWASP Top Ten Most Critical Web Application Vulnerabilities list as a primary guide (Appendix A). All of the listed vulnerabilities were within the scope of this audit except for **Insecure Configuration Management**, which focused on the server configuration. I added one item that was not in the OWASP list: **Hidden Content**.

Although it was necessary to perform a detailed review of CRS source code, which required technical skills that will not exist for all auditors, the audit was successful in determining the state of CRS' security.

4.1.1. Positive Findings

CRS properly handled authentication errors.

- The response was identical for both a bad User ID and Password and for a good User ID and bad Password.
- The user's account is disabled after an administrator-specified number of failed authentication attempts (currently set to three failed attempts).

4.1.2. Negative Findings

CRS failed most security audits. CRS does not provide satisfactory:

- Security policies and specifications
- Validation of input values
- Protection of sensitive information to prevent it from being accessible after it is displayed on the user's computer
- Protection of User ID values
- Encryption to protect passwords and other sensitive information while it is stored by the application or in transit between the client's browser and the application
- Protection against malicious scripts contained in information displayed to the user
- Prevention of unnecessary, sensitive information being displayed to the user

4.2. Audit Findings

4.2.1. Terminology

To understand of this report, it is necessary to understand the following terms.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

HTTP – *HyperText Transfer Protocol* – Used to transfer data between the web browser and the web server or to wrap the web server's response. The key word here is "text". HTTP can be manipulated with any text editor. Web browsers send requests to web servers using HTTP. The content of form fields (User ID, Password, application information, etc.) are contained in the body of the HTTP request.

HTML – *HyperText Markup Language* – Used to control how the text is displayed by the web browser. Once again, HTML is text that can be manipulated by any text editor. Web servers respond with HTML, usually wrapped in HTTP.

SSL – *Secure Socket Layer* – Used to protect information in transit by establishing encryption between the client's web browser and the application's web server.

While conducting this audit, I had access only to a test study with Data Administrator and Study Administrator authority. In my access, I did not have access to any production patient or study information.

4.2.2. Passed Audits

Table 4.1 – Error Handling

Background/Risk	**Background:** Improper error handling can introduce a variety of security problems for a web site. The most common problem is offering unnecessary information when errors occur; for example, displaying the processing stack or call trace. **Risk:** This information offers data that could help an attacker and is unhelpful and irritating to users.
Root Cause	Error handling should be well thought out and consistent with an application. This most effectively occurs during the design phase. CRS offers no effective design specifications to assure that errors are properly handled.
Test Results	• When I entered a bad User ID and a bad Password, and when I entered a good User ID and a bad password, the response took 91 seconds and displayed this error box:  There was no information that would be conveyed to an attacker identifying when the User ID was good, and therefore would not help the attacker who attempts to harvest User IDs as part of an attack. • CRS has a user setting specifying how many invalid logon messages may be received before locking the user's account. The setting is currently set to three invalid logon

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	messages. This is acceptable protection against brute force attacks repeatedly trying different User ID and Password combinations to break into the application. However, this could result in a Denial of Service (DoS) attack by locking user accounts following three invalid logon attempts. Administrator action is required to reset locked accounts.

4.2.3. Failed Audits

Table 4.2 – Missing Application Security Policies

Background/Risk	**Background:** Policies should exist to require that web applications are secure against the entire OWASP Top Ten list of vulnerabilities. **Risk:** It is unlikely that web applications will be secure in the absence of attention to security in all stages of the system's life cycle, beginning with the underlying development policies and continuing through all stages until system retirement.
Root Cause	CRS only gives cursory attention to security and no attention at the policy level.
Test Results	CRS includes no overall development or coding policies of any kind.

Table 4.3 – Missing Application Security Design Specifications

Background/Risk	**Background:** It is vital to have design documents that require that the web application include good security practices to be protected from the OWASP Top Ten list of vulnerabilities. **Risk:** In the absence of good security requirements, web applications are unlikely to be coded in a secure fashion.
Root Cause	CRS includes security requirements for: • Authentication • Authorization • Transmission security (128-bit SSL encryption) • Database create, update, and delete logging to assure data integrity • HIPAA compliance There are many more topics that should be included in web application security requirements: • Input validation of all HTTP input, including requirements to protect against cross-site scripting, injection, and buffer

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	overflow attacks • Centralized authentication and session management processing • Centralized authorization processing • Consistent error handling • Secure storage, including 128-bit or better encryption of passwords or password hashes • Limits on session-related resource usage to protect against denial of service attacks • No display of unnecessary information in HTTP headers, custom headers, code comments, URLs, cookies, hidden fields, error messages, or reports
Test Results	CRS has inadequate security requirements.

Table 4.4 – Unvalidated Input

Background/Risk	**Background:** All input from HTTP requests must be validated for appropriate content. Any input from and HTTP request can be manipulated, including the URL, query string, headers, cookies, and form fields, including hidden form fields. **Risk:** Malicious input from any part of the HTTP request can result in the compromise of the application, server, and network. All sensitive information used and stored by the application is at risk, as is any information stored by any other application or server on the network.
Root Cause	CRS validates input on the client using JavaScript executed by the web browser. Since any part of the HTTP request from the web browser can be manipulated by a text editor after client-side validation, validation on the client offers no security. Validation of data from the HTTP request must be done server-side to provide security. If client-side validation is used, it must be mirrored on the server. See Appendix B for a diagram showing how one tool (Achilles) can be used to manipulate text after it leaves the browser and before it arrives at the server.
Test Results	Both by interview and by code inspection, I verified that there is no server-side data validation except for User ID and Password. CRS is at risk of being compromised by unvalidated malicious input. An example of the client-side validation code is shown in Appendix B.

Table 4.5 – Broken Access Control

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Background/Risk	**Background:** Access Control, sometimes called Authorization, is the granting of access to information and functions to some users and not to others. If Access Control is not done properly, a user could gain inappropriate access by simply modifying the HTTP request to ask for unauthorized content or functions, or permit another user of the computer to gain access to sensitive information that was cached inappropriately on the client computer. **Risk:** The consequences of a breakdown in Access Control can be devastating. In addition to viewing inappropriate content, inappropriate access might permit the user to modify or delete content, perform inappropriate functions, or even gain site administration privileges. Inappropriate caching could permit another user of the computer to gain access to sensitive information on the client computer.
Root Cause	CRS does not provide appropriate caching control to prevent sensitive information from being stored on the client computer. CRS uses an HTTP `Cache-Control: no-cache` statement in an attempt to control caching. However, the `no-cache` value only forces caches to submit a request to validate the content *before releasing a cached copy*. That means that—even though the `Cache-Control: no-cache` statement sounds like the browser should not cache the content—sensitive content *is cached* by the browser. (Nottingham)
Test Results	In response to a client request for sensitive information, CRS sent a `Cache-Control: no-cache` statement in the HTTP header. As explained above, the sensitive information is cached anyway. See Appendix C for the HTTP header text.

Table 4.6 – Broken Authentication and Session Management

Background/Risk	**Background:** This audit looks at all aspects of handling user authentication and managing active sessions; among these are password strength, password use, password change, password storage, protecting credentials in transit, session ID protection, and account lists. **Risk:** Improper authentication and session management can result in granting access rights inappropriately. This can open the application, server, and network to improper access or to attack.
Root Cause	CRS has problems in two areas: • **Password storage** – See Insecure Storage below. • **Protecting credentials in transit** – CRS only supports three

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	weak ciphers for the protection of information in transit, including user authentication credentials (DES(40), RC4(40), and DES(56)). As noted above, none of these ciphers provides sufficient protection for credentials in transit. See Appendix D for a portion of the SSLDigger CRS report and the page properties showing the use of weak DES(56) SSL encryption. • **Account lists** – Anyone who is assigned a Data Administrator or Study Administrator role has access to reports that display all CRS User IDs, including application administrator IDs. This is helpful for anyone harvesting User IDs for an attack against the system. See Appendix D for the report selection list that will display CRS User IDs.
Test Results	CRS uses DES 56-bit encryption for encrypting information in transit. DES is insufficient protection for sensitive information. CRS allows Data and Study Administrators to display all CRS User IDs. This offers valuable information to anyone attempting to harvest User IDs to attack the application.

Table 4.7 – Cross-site Scripting (XSS) Flaws

Background/Risk	**Background:** Cross-site scripting, or XSS, occurs when an attacker sends malicious code, usually script code, as input to an application that subsequently sends it out unchanged, to an end user. The end user's browser has no way to know that this text should not be trusted since it comes from a trusted source—the application—and the malicious code is executed on the user's system. **Risk:** The malicious code could do anything the user has authority to do, or take advantage of other system and network flaws to promote itself to higher levels of authority. Potentially, the attacker could gain full rights to the network and all its data.
Root Cause	All HTTP input, including all headers, cookies, query strings, form fields, and hidden fields could potentially be used for an XSS attack. All HTTP input must be verified to assure that it contains only expected values. In addition, all text sent to a user must have character values translated as shown to prevent it from being executed as code by the receiving browser.

From	To	From	To
<	<	)	)
>	>	#	#
(	(	&	&

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Test Results	As stated previously, no server-side input validation is performed by CRS. Although code exists in CRS to translate the "<" and ">" characters as shown above, I could find no routines that called that code. The other translations are not performed by CRS.

Table 4.8 – Improper Error Handling

Background/Risk	**Background:** Improper error handling can introduce a variety of security problems for a web site. The most common problem is offering unnecessary information when errors occur; for example, displaying the processing stack or call trace. **Risk:** This information offers data that could help an attacker and is unhelpful and irritating to users.
Root Cause	Error handling should be well thought out and consistent with an application. This most effectively occurs during the design phase. CRS offers no effective design specifications to assure that errors are properly handled.
Test Results	When a processing error occurs, CRS displays an error page that includes the CRS logo. When the user moves the mouse over the logo, debug error information is displayed. This information is helpful to CRS support staff when investigating the error, but could also be helpful to an attacker by displaying internal application information. In addition, it is not information that would be useful or helpful to a valid user. This information should be written to an error log file for the use of the support staff. See Appendix E for code from the error page that displays the information when the mouse is moved over the CRS logo.

Table 4.9 – Insecure Storage

Background/Risk	**Background:** Web applications often have a need to store sensitive information; for example, passwords or proprietary information. Encryption is not difficult to use, but often it is implemented improperly; for example, the encryption could be weak or keys could be stored insecurely. **Risk:** Sensitive information could be accessed and decrypted by an attacker.
Root Cause	Encryption must be implemented with care: • Used to protect sensitive information • Well established and strong ciphers must be used • Keys must be stored securely
Test Results	CRS uses tools from Oracle's DBMS Obfuscation Toolkit to

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	encrypt passwords. Unfortunately, the DBMS Obfuscation Toolkit only supports DES(56) encryption, which, as noted above, is inadequate today to protect sensitive information. See Appendix F for an example of the code that encrypts passwords. Also, as noted above, a weak DES(56) cipher is used to protect information in transit, and the cache is not properly controlled to prevent the caching of sensitive information by the user's browser.

4.3. Audit Recommendations

Estimated costs are based on the following hourly rates. The estimated hours assume the availability of analysts, designers, and coders who are already familiar with CRS policies, specifications, functionality, and code.

Position	Hourly Rates
Analyst	$125
Designer	$100
Coder	$75

The OWASP Stinger project should prove particularly helpful for improving the input validation of CRS. Stinger is a J2EE validation mechanism that can be downloaded at no cost from the OWASP web site[10].

Table 4.10 – Application Security Policies

Recommendation	Develop the appropriate policies to define appropriate application security policies to cover the OWASP Top Ten list of vulnerabilities.
Estimated Costs	Two weeks of one analyst's time = $10,000

Table 4.11 – Application Security Design Specifications

Recommendation	Develop the appropriate design specifications to meet the application security policies. Input validation design specifications should require the use of the OWASP Stinger J2EE validation mechanism.
Estimated Costs	Four weeks of one designer's time = $16,000

[10] *OWASP Stinger — A J2EE HTTP Validation Engine*. OWASP. URL: www.owasp.org/software/validation/stinger.html.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table 4.12 – Input Validation

Recommendation	Develop server-side input validation routines using the OWASP Stinger J2EE validation mechanism.
Estimated Costs	Three weeks of one coder's time = $9,000

Table 4.13 – Access Control

Recommendation	Develop code to make use of appropriate HTTP header statements to prevent caching of sensitive information. One or both of the following HTTP Header statements should be used whenever sensitive information is sent to the user. • `Cache-Control: no-store` • `Expires` with a past date
Estimated Costs	Two days of one coder's time = $1,200

Table 4.14 – Authentication and Session Management

Recommendation	Make the following enhancements to CRS code: • Upgrade SSL to support 128-bit ciphers. Remove support for DES(40), RC4(40), and DES(56) ciphers. • Change report menus or report formats so that User IDs are only displayed to the CRS Administrator.
Estimated Costs	Three weeks of one coder's time = $9,000

Table 4.15 – Cross-site Scripting (XSS) Prevention

Recommendation	Upgrade all routines sending text to the User to make the following character translations to eliminate the possibility of sending executable script code to the user's browser.
Estimated Costs	One week of one coder's time = $3,000

From	To	From	To
<	<	)	)
>	>	#	#
(	(	&	&

Table 4.16 – Error Handling

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Recommendation	Write error information to an error log that is accessible to CRS support staff. Remove the onMouseOver and onMouseOut events from the production code for the CRS Logo displayed on the Error Page.
Estimated Costs	One day of one coder's time = $600.

Table 4.17 – Secure Storage

Recommendation	Replace Oracle's DBMS Obfuscation Toolkit with a third-party tool that uses 128-bit or better cipher strength for encrypting passwords
Estimated Costs	One week of one coder's time = $3,000

4.4. Compensating Controls

While the most effective protection against the OWASP Top Ten vulnerabilities is to make the individual web application secure, it is an expensive process to retrofit security into a production application. In the case of CRS, the conservative estimated costs total $51,800.

In addition, if two or more web applications need to be secured, that cost can increase rapidly.

Another security control that could offer satisfactory protection to an insecure web application is an application firewall. Application firewalls inspect the information to and from web applications looking for malicious or dangerous input and unnecessary or dangerous output. Only safe, valid information is allowed to be sent to or from the application.

There are a number of application firewall products that could offer protection for existing and new web applications. Ten application firewall products have been rated and compared by Network World Fusion (www.nwfusion.com/bg/2004/appsecurity/_index.jsp). Prices for the reviewed products range from $1,295 to $35,000.

Several higher-end application firewall products are shown below. There are additional application firewalls available on the market that should also be considered. In any case, it is easy to see that it could be far less expensive to implement an application firewall than to retrofit security for CRS, without even considering any other web applications that might also need to retrofit security.

- **NC-1000 Application Security Gateway**
  $29,000

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

"The NetContinuum Application Security Gateway delivers the highest level of protection available for your critical applications — eliminating data theft, financial fraud and loss of customer confidence due to web application attacks."
www.netcontinuum.com/

- **SecureSphere G4 Gateway and SecureSphere MX Management Server**
  $30,000
  "The SecureSphere Dynamic Profiling Firewall family of appliances — including the G4 Gateway and the MX Management Server — delivers a comprehensive security solution that proactively identifies and blocks attacks that threaten your mission-critical web-based enterprise applications and databases."
  www.imperva.com/products/securesphere/

- **TrafficShield**
  $25-35,000
  "TrafficShield is a Web Application Firewall that provides comprehensive, proactive, application-layer protection against both generalized and targeted attacks. TrafficShield employs a positive security model ('deny all unless allowed') to permit only valid and authorized application transactions, while automatically protecting critical Web applications from attacks such as Google hacking, cross-site scripting, and parameter tampering."
  www.f5.com/f5products/products/TrafficShield/

- **AppShield**
  $20,000
  "AppShield, an automatic Web application firewall, provides enterprise-class Web intrusion prevention for a failsafe defense against all application level breaches. AppShield allows for easy application deployment in a secure environment by intelligently identifying the legitimate requests made of an ebusiness site and permitting only those actions to take place, enforcing the Web and business logic of the site. By preventing, logging and alerting administrators to any type of application manipulation through the browser, AppShield maintains application behavior 24/7 without the need for signatures or rules."
  www.watchfire.com/products/appshield/default.asp

- **Teros Secure Application Gateway**
  $25,000
  "The Teros Secure Application Gateway does for Web and Web Services applications what network firewalls do for the network. The Teros Gateway is a hardened security appliance that is deployed directly in the data path of application traffic and blocks attacks that are not detected by network-based firewalls and intrusion detection systems. The Teros Gateway enforces a positive security model that only permits correct application behavior, without relying on attack signatures. It provides defenses for vulnerabilities that may exist within custom applications, as well as the known weaknesses in commercially-developed software."
  www.teros.com/products/appliances/gateway/index.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- **e-Gap Application Firewall**
  $23,000
  "The e-Gap® Application Firewall enables organizations to rapidly deploy secure web-based access to sensitive core applications. The System may be used to protect e-business applications for customers or partners (such as eCRM, supply chain integration or e-billing). It protects against known and unknown threats by isolating application servers - via Air Gap technology - and tightly controlling application layer access to them. It also significantly reduces the urgency to patch production web servers. It unites all of the application-protection components into a single application-centric appliance, and features automatic learning of the application to generate and enforce application-level rule sets. Encryption, authorization, authentication, PKI, HTTP payload screening, automatic rule-set generation and a physical air gap all reside within an integrated software/hardware platform."
  www.whalecommunications.com/site/Whale/Corporate/Whale.asp?pi=35

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix A – Top 10 Most Critical Web Application Vulnerabilities

Ref.	Vulnerability	Description
A1	Unvalidated Input	Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.
A2	Broken Access Control	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.
A3	Broken Authentication and Session Management	Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.
A4	Cross-site Scripting (XSS) Flaws	The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.
A5	Buffer Overflows	Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
A6	Injection Flaws	Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
A7	Improper Error Handling	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.
A8	Insecure Storage	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Ref.	Vulnerability	Description
		protection.
A9	Denial of Service	Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.
A10	Insecure Configuration Management	Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

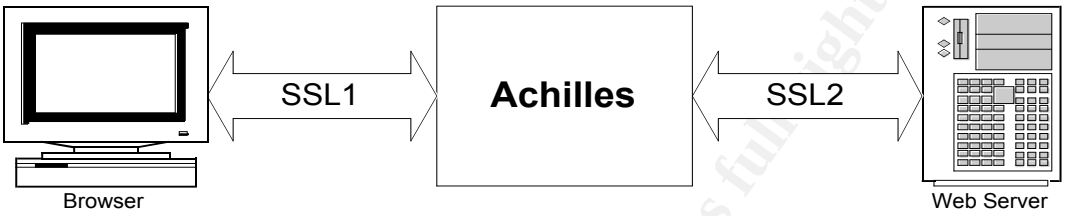
(OWASP, pp.4)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix B – Client-Side Validation

Achilles, by Maven Security, is an example of tools that can be used to intercept and manipulate text between the web browser and the web server, even if the web browser establishes a "secure" SSL link with the web server. The web browser thinks it is connected to the web server, and the web server thinks it is connected to the web browser. In fact, both are connected to Achilles. Achilles is a legitimate audit tool, but it—and products like it—can be used maliciously to attack a web application.



The following code is an example of CRS client-side input validation.

```
Switch (strElementType) {
      case  "typeAlpha" : strReturnValue =      IsAlphabet(elElement); break;
      case  "typeNum" : strReturnValue = IsNumber(elElement); break;
      case  "typeString" : strReturnValue =      IsString(elElement); break;
      case  "typeDate" : strReturnValue =  IsValidDate(elElement, false);
break;
      case  "typeDob" : strReturnValue =   IsValidDob(elElement); break;
      case  "typeName" : strReturnValue =  IsValidName(elElement); break;
      case  "typeSSN" : strReturnValue =   IsValidSSN(elElement); break;
      case  "typeEMail" : strReturnValue =       IsValidEMail(elElement);
break;
      case  "typeRadio" : strReturnValue =       IsRadioSelected(elElement);
break;
      case  "typeUrl" : strReturnValue =   IsValidUrl(elElement); break;
      case  "typeAtLeastOne" : strReturnValue =  AtLeastOne(elElement,
frmForm); break; }
```

Examples of the called edit routines are:

```
function IsValidName(elElement) {
      var strReturnValue = "";
      var strValue = elElement.value;
      var strExp = /[^A-Za-z0-9.\-'\s]/;
      if (strValue.search(strExp) >= 0) {
            var DisplayName = elElement.displayName;
            if (!DisplayName)
                  strReturnValue = "Field";
            else
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
                  strReturnValue = DisplayName;
            strReturnValue += " can only contain alphanumeric characters,
period(.), hyphen(-), space or single quotes."; }
      return strReturnValue; }

function IsAlphabet(elElement) {
      var strReturnValue = "";
      var strValue = elElement.value;
      var strExp = /[^A-Za-z]/;
      if (strValue.search(strExp) >= 0)
            strReturnValue = "Field can only contain alphabets.";
      return strReturnValue; }
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix C – Cache Control HTTP Header Statement

```
POST . . . HTTP/1.0
Accept: . . .
Referer: . . .
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
User-Agent: . . .
Host: . . .
Content-Length: 140
Cache-Control: no-cache
Cookie: . . .
HEADER_SELECTION= . . .
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix D – Authentication and Session Management

SSLDigger CRS report fragment:

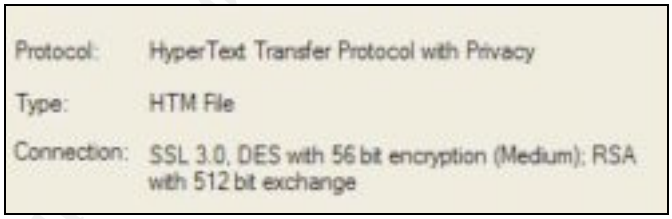
Summary

Number of Servers Tested	1
Number of Ciphers Used in Testing	26
No Security	2
Weak Security	10
Strong Security	9
Excellent Security	5

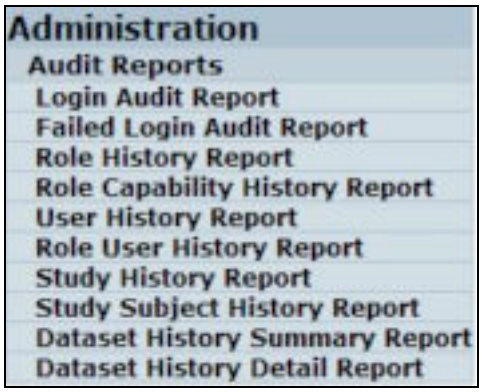
Ciphers Supported

Server URL	No Security	Weak Security	Strong Security	Excellent Security	Grade
https://xxx.xxxxxx.xxx:9999	0	3	0	0	C

Page properties fragment showing the use of DES 56-bit encryption for SSL.



CRS report selection list for Data Administrators and Study Administrators. Note the User History Report and the Role User History Report: both display CRS User IDs.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix E – Error Handling

The Error Page includes the system logo. When the mouse is moved over the cursor, debug error information is displayed. When the mouse is moved off of the cursor, the debug error information is hidden.

```
<td class="FormText" align="center" height="32"><img
src="/web_crs/images/LogoCrs.gif" width="256" height="20"
onMouseOver="MM_showHideLayers('errortext','',
'show')" alt="Crs Logo"
onMouseOut="MM_showHideLayers('errortext','',
'hide')">
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix F – Secure Storage

```
procedure add_user (p_app_user users.username%type
 ,p_username users.username%type
 ,p_last_name users.last_name%type
 ,p_first_name users.first_name%type
 ,p_middle_name users.middle_name%type
 ,p_title users.title%type
 ,p_display_name users.display_name%type
 ,p_email_address users.email_address%type
 ,p_password users.password%type) is v_default_study_id number;
begin
    util_pkg.check_privilege(p_app_user, 'COMMUNITY', 1, null,
'Add-Edit Users');
    select default_study_id
    into v_default_study_id
    from users
    where username = p_app_user;

    insert into users (
        username,
        last_name,
        first_name,
        middle_name,
        title,
        display_name,
        email_address,
        password,
        default_study_id,
        created_by,
        date_created)
    values (
        p_username,
        p_last_name,
        p_first_name,
        p_middle_name,
        p_title,
        p_display_name,
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        p_email_address,
        util_pkg.desencrypt(p_password),
        v_default_study_id,
        p_app_user,
        sysdate);
end add_user;

function desencrypt (p_input_string varchar2) return varchar2 is
    v_input_string varchar2(2048);
    v_encrypted_string varchar2(2048);
    v_multiple number;
begin
    v_multiple := trunc((length(p_input_string) / 8) + 1);
    v_input_string := rpad(p_input_string, 8 * v_multiple, '@');

    dbms_obfuscation_toolkit.DESEncrypt(
        input_string => v_input_string,
        key_string => key_string,
        encrypted_string => v_encrypted_string );
    return v_encrypted_string;
end desencrypt;
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix G – Additional Resources

Bayuk, Jennifer. "Introducing Security at the Cradle, not the Grave." April 6, 2003. URL: www.sans.org/rr/audittech/Jennifer_Bayuk_WP.pdf (August 16, 2004)

Curphey, Mark, et. al. "A Guide to Building Secure Web Applications." The Open Web Application Security Project. September 11, 2002. URL: aleron.dl.sourceforge.net/sourceforge/owasp/OWASPGuideV1.1.1.pdf (September 12, 2004)

Elmenshawy, Maged, and David Meeh. "Systems and Controls to Make Information Security Measurable and Relevant in a Global Enterprise." SANS InfoSec Reading Room. 2003. URL: www.sans.org/rr/audittech/Magid_Elmenshawy_WP.pdf (August 28, 2004)

Graff, Mark G., and Kenneth R. van Wyk. *Secure Coding: Principles and Practices*. O'Reilly & Associates: Sebastopol, CA. 2003.

Ihrer, Ken. "Database Security: Securing Oracle." Information Security Magazine. September 2000. URL: infosecuritymag.techtarget.com/articles/september00/_features1.shtml (November 21, 2004)

Jaquith, Andrew. "The Security of Applications: Not All Are Created Equal." February 2002. URL: www.atstake.com/research/reports/acrobat/_atstake_app_unequal.pdf (August 28, 2004)

Landwehr, Carl E., Alan R. Bull, John P. McDermott, and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples." September, 1994. URL: www.cs.mdx.ac.uk/research/SFC/Papers/1994landwehr-acmcs.pdf (September 12, 2004)

Meier, J.D., et. al. "Improving Web Application Security: Threats and Countermeasures." Microsoft MSDN Library. June 2003. URL: msdn.microsoft.com/_library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp (September 16, 2004)

OWASP2. "The OWASP Testing Project." Draft Version 1.0. The Open Web Application Security Project. July 2004. URL: prdownloads.sourceforge.net/owasp/_TheOWASPTestingProjectPart1Draft.pdf?download (October 3, 2004)

OWASP3. "OWASP Web Application Penetration Checklist." Version 1.1. The Open Web Application Security Project. July 14, 2004. URL:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

prdownloads.sourceforge.net/_owasp/OWASPWebAppPenTestList1.1.pdf?download (October 3, 2004)

SiegeWorks. "Application Audits." *Application Security.* 2004. URL: www.siegeworks.com/app-audits.html (September 8, 2004)

Soo Hoo, Kevin, Andrew Jaquith, and Dan Geer. "The Security of Applications, Reloaded." July 2003. URL: www.atstake.com/research/reports/acrobat/_atstake_app__reloaded.pdf (August 28. 2004)

Valois, Kim. "Making the Business Case for Security Controls in Very Tough Economic Times." 2003. URL: www.sans.org/rr/audittech/Kim_Valois_WP.pdf (August 16, 2004)

WASC. "Web Application Security Consortium: Threat Classification." Web Application Security Consortium. Last update: July 27, 2004. URL: www.webappsec.org/tc/_WASC-TC-v1_0.pdf (September 19, 2004)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

References

Araujo, Rudolph. "The Need for Strong SSL Ciphers." Foundstone. July 2004. URL: www.foundstone.com/resources/whitepapers_registration.htm?file=wp_ssldigger.pdf (October 30. 2004)

Beales, J. Howard, III. "OWASP Updates its Top 10 List of Critical Web Application Security Vulnerabilities." EiP. January 2004. URL: www.eipdistribution.com/_news.php?news_id=32 (September 19, 2004)

FDIC. *Guidance on Developing an Effective Computer Software Evaluation Program to Assure Quality and Regulatory Compliance*. FDIC. November 16, 2004. URL: www.fdic.gov/news/news/_financial/2004/fil12104.html (December 4, 2004)

Fredholm, William. *Web Application Security – Layers of Protection*. SANS Reading Room. January 26, 2003. URL: www.sans.org/rr/papers/index.php?id=965 (November 27, 2004)

Hoelzer, David. *Advanced System and Network Auditing.* SANS. 2004.

Hulme, George V. "New Software May Improve Application Security." Information Week. February 9, 2001. URL: www.informationweek.com/story/_IWK20010209S0003 (September 13, 2004)

Levine, Matthew. "The Importance of Application Security." April, 2002 (updated January, 2003). URL: www.atstake.com/research/reports/acrobat/_atstake_application__security.pdf (August 25, 2004)

Loomis, Angela. "Auditing Web Applications for Small and Medium Sized Businesses." SANS InfoSec Reading Room, 2003. URL: www.sans.org/rr/audittech/Angela__Loomis_WP.pdf (August 16, 2004)

NIST. "DES Transition – Request for Public Comment." National Institute for Standards and Technology, October 7, 2004. URL: csrc.nist.gov/cryptval/notices.htm (November 27, 2004)

Nottingham, Mark. "Caching Tutorial." Mark Nottingham's home page. February 15, 2004. URL: www.mnot.net/cache_docs/. (November 21, 2004)

OWASP. "The Ten Most Critical Web Application Security Vulnerabilities." The Open Web Application Security Project. January 27, 2004. URL: voxel.dl.sourceforge_.net/sourceforge/owasp/OWASPTopTen2004.pdf (September 15, 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Rhoades, David. “Auditing Web Servers and Applications.” v. 1.8. Advanced System and Network Auditing. SANS. 2004.

Roberts, Timothy. “Hackers Getting New Foe.” Silicon Valley / San Jose Business Journal. February 16, 2004. URL: sanjose.bizjournals.com/sanjose/stories/2004/02/_16/story2.html (September 13, 2004)

SPI Dynamics. Description of “Security at the Next Level: Are Your Web Applications Vulnerable?” 2004. URL: www.spidynamics.com/support/whitepapers/index.html (November 13, 2004)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.