

# **Web Application Security Testing with the Security Center and Nessus**

January 2, 2004  
(Updated February 7, 2007)

**Stephen Schwing**  
Tenable Sales

**John Lampe**  
Senior Security Engineer

---

## Table of Contents

TABLE OF CONTENTS .....	2
INTRODUCTION .....	3
WHY DETECT WEB APPLICATION VULNERABILITIES?.....	3
DETECTING WEB APPLICATION VULNERABILITIES USING NESSUS .....	3
DDI_DIRECTORY_SCANNER.NASL.....	5
TORTURECGIS.NASL.....	5
SQL INJECTION (AND SQL_INJECTION.NASL).....	5
ADVANTAGES OF WEB APPLICATION SCANNING WITH NESSUS .....	6
CONFIGURING NESSUS FOR A WEB APPLICATION SCAN.....	6
<i>ABOUT TENABLE NETWORK SECURITY.....</i>	<i>8</i>

## Introduction

The detection of Application-layer flaws within HTTP (or web) applications has become a major source of activity for many enterprise security groups. Source code audits are one means of finding these security holes. Many of these efforts to detect application bugs within web applications can be automated by using the Nessus security scanner in conjunction with the Security Center (formerly Lightning Console). The Security Center allows the security administrator to schedule and report on scans in an automated and distributed fashion. This paper will discuss the techniques used by Nessus and the Security Center to efficiently scan for application layer bugs. This paper assumes that the reader is familiar with operating a Nessus scan, web basics, SQL basic syntax and Active scripting (or CGI) applications.

## Why Detect Web Application Vulnerabilities?

A Web application bug can be every bit as devastating as a network-level bug. To further exacerbate the problem, many companies and institutions **allow** web traffic to/from their secured network segments (DMZ). An attacker that gains access to a web server may have the ability to modify or create web content. The attacker may also gain access to business-critical data (such as business logic, passwords, etc.). Furthermore, the attackers can use their existing access to “escalate” their privileges on the internal network. While the firewall may block access from the Internet into the internal network, most firewalls must, necessarily, allow their web servers to communicate with internal machines and other DMZ machines.

## Detecting Web Application Vulnerabilities Using Nessus

**NOTE:** *Nessus checks for many web/CGI vulnerabilities (literally, hundreds). The scope of this paper is the detection of “unknown” or “undocumented” vulnerabilities within a web application. That is, the methodologies denoted below will find inherent weaknesses within “homegrown” or “custom” web applications. This is significant in that it is a departure from “signature based” vulnerability scanning.*

Security auditors attempting to identify web application bugs should consider using the Nessus vulnerability scanner available from <http://www.nessus.org/>. Tenable Network Security, Inc. is the author and manager of the Nessus Security Scanner which is available for Unix, Windows and OS X operation systems. Nessus performs many security checks for the hosts it tests and these checks are written predominantly in a language named NASL (Nessus Attack Scripting Language). The checks are often referred to as “plugins” or Nessus modules. The first step that Nessus automates during a vulnerability test is the mirroring of the entire web server. This allows Nessus to find (and save within the knowledge base) all active scripting (or CGI) directories, and all web forms. The latter is a very important component in web application testing.

The plugin which mirrors the website is webmirror.nasl with a Plugin ID of 10662. Webmirror.nasl does more than just “mirror” the website and populate the knowledge base. Webmirror.nasl also looks for inherent weaknesses within the active scripts (or CGI scripts) that it finds. Furthermore, as webmirror.nasl runs, it finds all active (or CGI) scripts. Nessus notes the location of these scripts and ferrets away the information for further testing. So, for example, if there was a link to a file called /my-secret-cgi/foo.cgi, Nessus would make

---

note of the directory /my-secret-cgi and include this directory in **all** future CGI directory checks.

After webmirror.nasl runs, the knowledge base is populated with all known active scripts and forms (among other things). The following scripts take advantage of this information and run individual checks.

- apache\_dir\_listing.nasl
- asp\_source\_data.nasl
- asp\_source\_dot.nasl
- asp\_source\_space.nasl
- bakfiles.nasl
- BEA\_weblogic\_Reveal\_Script\_Code\_2.nasl
- BEA\_weblogic\_Reveal\_Script\_Code\_3.nasl
- BEA\_weblogic\_Reveal\_Script\_Code.nasl
- db4web\_dir\_trav.nasl
- DDI\_IIS\_Compromised.nasl
- domino5\_overflows.nasl
- domino6\_overflows.nasl
- ibm\_server\_code.nasl
- iis\_asp\_overflow.nasl
- iis\_codebrws.nasl
- iisprotect\_bypass.nasl
- imp\_mime\_viewer\_html\_xss.nasl
- iplanet\_app\_server\_detection.nasl
- mailreader.nasl
- mod\_auth\_any.nasl
- mod\_gzip\_running.nasl
- mod\_ntlm.nasl
- mod\_survey\_sql\_injection.nasl
- ms\_index\_server.nasl
- multitech\_proxy\_default\_pwd.nasl
- no404.nasl
- office\_files.nasl
- oracle9i\_globals\_dot\_jsa.nasl
- oracle9i\_jsp\_source.nasl
- osX\_apache\_finder\_content.nasl
- osX\_apache\_finder.nasl
- php\_4\_2\_x\_malformed\_POST.nasl
- php\_split\_mime.nasl
- psynch\_multiple\_vulns.nasl
- servletExec\_File\_Reading.nasl
- sql\_injection.nasl
- tomcat\_source\_exposure.nasl
- torturecgis.nasl
- webgais.nasl
- webstores\_browseitemdetails\_sql\_injection.nasl
- www\_hosting\_copyrighted\_material.nasl

A description of each of these NASL modules is outside the scope of this paper. We will, however, go into more detail on DDI\_Directory\_Scanner.nasl, torturecgis.nasl and sql\_injection.nasl.

## DDI\_Directory\_Scanner.nasl

This NASL module will find directories (both visible and “hidden”). It does this by using a brute-force technique against the web server. So, for example, if the Web Administrator created a directory named “backup” which he/she uses to store old password files, DDI\_Directory\_Scanner.nasl would find this directory and report on it. This NASL module will find directories which are not “linked” to the actual web server content. As many Administrators rely on security through obfuscation, this NASL module is a critical piece of any web application penetration test.

## Torturecgis.nasl

This NASL module takes the output from webmirror.nasl and tests each of the active scripts (or CGI scripts). Torturecgis.nasl includes checks for:

- Cross-site-scripting (XSS). Cross-site-scripting allows a malicious external user to potentially run active code against unsuspecting browsers (or email clients) within the security scope that is applied to the target web server. An example is probably in order. Consider a site which has a form that is vulnerable to cross-site-scripting. A malicious individual can craft a URL that looks valid (<http://www.vulnerable.site/vulnerable.asp?myinfo=<script>...</script>>). Now, in this example, everything between the <script> and </script> would be executed on the client browser with the security rights of the site [www.vulnerable.site](http://www.vulnerable.site). Of course, this could be used to retrieve and forward cached credentials, steal cookies, besmirch company image, etc.
- Directory escape techniques. For example, a “../” within a directory path could force the web server to access files outside of the web directories. A common example is the use of “../../../../etc/passwd” to obtain the Unix password file.
- Hidden form values. Many sloppy applications will pass or store credentials as a “hidden” value.
- Many other application-layer checks.

## SQL Injection (and sql\_injection.nasl)

### What is SQL Injection?

SQL injection is an Application layer (or layer 7) attack. At this time, it is one of the more popular attack methods employed by professional penetration testers as well as hackers. SQL injection attacks exploit flaws that are caused by the failure to properly validate input data by the programmer, and are a common root cause of many of the popular Application layer attacks.

Within SQL statements, certain characters have a special meaning. For instance, the semicolon tells the SQL engine to stop processing the statement, and the keyword OR changes the logic of the SQL statement. When an application fails to strip out one of these special characters or strings, the logic of the SQL statement can be compromised. This can lead to unauthorized system or data access. SQL injection attacks commonly use a “front-end” (or ingress) machine as their attack vector. Typically Web sites are connected to a back-end SQL server. The attacker typically tests the Web server for CGI scripts or forms that return error codes regarding SQL. Once an attacker finds a script or form which does

not validate the user input, it is a short time before the attacker gains full access to the SQL server.

You can test for the SQL injection manually; however, this can be very time consuming. To manually check for SQL injection flaws (not recommended), you must first identify the ingress point of the SQL statement (i.e., the Web form, CGI, etc.) and map out each of the input parameters. For example, `www.thisNEwidget.com/orderform.asp` is a Web form which accepts four input variables (text boxes named `box`, `box2`, `box3` and `box4`). A typical browser address bar may look like:

```
http://www.thisNEwidget.com/orderform.asp?box=Joe%20Smith&box2=Atlanta%20GA&box3=widget%20order&box4=654333
```

(Note that `%20` is just how the browser represents a space.)

In this imaginary order, a user named Joe Smith from Atlanta, Georgia, is placing a widget order for item number 654333. In the browser window, you can tamper with the values of the four variables. By changing the URL to:

```
http://www.thisNEwidget.com/orderform.asp?box=Joe%20Smith&box2=Atlanta%20GA&box3=widget%20order&box4=;
```

(Note the trailing semicolon.) You may receive a SQL or database error. The site is probably vulnerable to SQL injection if this occurs. Once a SQL injection flaw is found, it is typically manually exploited. Common exploit methods include calling a stored procedure (such as `xp_cmdshell`, a Microsoft SQL default stored procedure), creating a SQL query which dumps all table data to an HTML table, and copying a file (like `cmd.exe`) into the Web root directory.

### **How Does Nessus Detect SQL Injection Vulnerabilities?**

Nessus automates the process denoted above.

*Sql\_injection.nasl* takes the output from *webmirror.nasl* and queries each form using an invalid SQL statement. Nessus then looks at the reply from the web server. So, for example, if the web server returned an error which contained the string "You have an error in your SQL syntax", Nessus would flag the application as being vulnerable to SQL injection.

## **Advantages of Web Application Scanning with Nessus**

Manual code audits are tedious and time consuming. Nessus can scan every day and find the obvious flaws within an application. Even after a manual code audit, the programs will be edited and changed over time. Automated Nessus scanning (a feature within the Security Center) will detect the application-layer vulnerability at the next scan window.

## **Configuring Nessus for a Web Application Scan**

To conduct a Nessus scan for Web/CGI applications, perform the following steps:

1. Perform an update of the Nessus plugins to make sure you have the latest version of the plugins denoted above.

- 
2. Configure a new scan by selecting all plugins within the family "CGI ABUSES"
  3. Enable a port scan for ports 80 and 443 (NOTE: for completeness, you should scan web servers for all TCP ports 1-65535 on a regular basis)
  4. Make sure that "Enable Dependencies at Runtime" is ENABLED.
  5. Run the scan.

For those who are using the Security Center to manage their scans, you would simply create a "Template" scan with the settings from above. This scan would then be available to authorized users.

---

## ***About Tenable Network Security***

*Tenable, headquartered in Columbia, Md., USA, is the world leader in Unified Security Monitoring. Tenable provides agent-less solutions for continuous monitoring of vulnerabilities, configurations, data leakage, log analysis and compromise detection. For more information, please visit us at <http://www.tenablesecurity.com>.*

**TENABLE** Network Security, Inc.  
7063 Columbia Gateway Drive  
Suite 100  
Columbia, MD 21046  
TEL: 410-872-0555  
<http://www.tenablesecurity.com>