



## Web Site Security Audit (WSSA) Whitepaper

As web applications become an integral part of more and more business activities, the need to perform security checks on self-developed web applications is a necessity.

To answer this need, users of Beyond Security's Automated Scanning server can now perform web application level security audits, in a very similar manner to the network level and OS level security scans.

Beyond Security's web application checks detect with great accuracy whether the remote web application is vulnerable to web-level attacks. These checks are conducted on all server side scripts, including custom-made web applications.

As custom web applications are usually built in-house and are not commercial applications, vulnerabilities in those web applications may not be detected by the "known vulnerability" checks in the regular network-level scan. Therefore, a separate scan is necessary.

The **crawling** process - detecting what web application components exist (CGI's, ASP's, JSP servlets, etc) is partially manual, and the most time consuming aspect of the Layer 7 checks.

Custom-made web application are, by definition, unpredictable in their response to attack. However, certain similar characteristics exist in all vulnerable products. A web application attacked using several different types of SQL injection techniques, will respond with a predictable fingerprint, allowing **detection** of vulnerable web applications.

The Layer 7 checks consist of the following groups of vulnerabilities:

1. SQL Injections
2. JSP/ASP/PHP Code Injections
3. Command Execution (through piping)
4. File disclosure (Windows and UNIX style)
5. Cross Site Scripting (HTML and JavaScript injection)

Before any of the above tests are conducted, the Layer 7 check tries to create an initial **fingerprint** of what would be a normal behavior of the file being tested. This is done by accessing the scripts, and sending the web form complete, partial, and malformed content



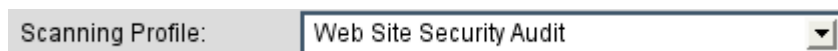
for each of the parameters that are provided as input to the page.

After this initial fingerprinting phase, the check goes on to sending for each of the group of vulnerabilities, predefined strings that are known to trigger a specific vulnerability that is part of the group. As the tool has already gathered what the normal behavior of the page is (using the **fingerprinting** process described previously) , it can now conclude very accurately that the application misbehaved when it was sent the attack string.

At this point, it would have been very easy to simply print out a list of vulnerable pages, and how we triggered these misbehaviors. However, this may result in many false positives (reporting vulnerabilities that do not exist). To avoid this, the tool sends special predefined strings that are known to trigger certain behavior in specific back-end programs while not triggering other back-end programs (for example, MS SQL Server vs. MySQL). This allows us to better ascertain whether the misbehavior is due to an actual vulnerability or due to bad fingerprinting. This dramatically reduces the number of false positives generated by the test.

## How to perform the test

In order to perform the web application tests, add a new scan entry, and select the “Web Site Security Audit” profile:



***NOTE: This profile will be available only if your server is equipped with the “Web Site Security Audit” license.***



Click “Add” and continue to enter the scan entry into the database.  
You will then be prompted by a new screen, that will enable you to configure the crawling functionality.

**Crawler configuration**

The crawler will begin from the following URL: **http://192.168.1.61**.

Protocol:

Crawler starting URL:

Connect on port:

Maximum traversal depth:

Use the following cookie:

Use the following username and password (HTTP Authentication):

\* Username:

\* Password:

Use the following for URL-based authentication:

\* URL:

\* URL Method:

\* Username Parameter:  (i.e. 'sys\_username')

\* Username Value:  (i.e. 'bar')

\* Password Parameter:  (i.e. 'sys\_password')

\* Password Value:  (i.e. 'foo')

\* Any additional values and paramters :  (i.e. 'id=1&from=login.asp')

\* Process reponse:  , response parameter:

Skip URLs with the string (for example: logout):

On this screen you can define the URL where the crawling will start from, the depth of the



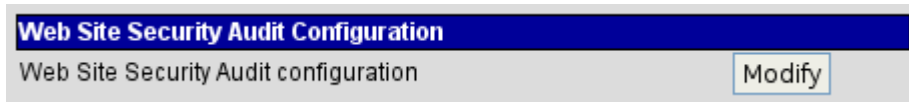
crawl, the cookie configuration and custom authentication, if the web site is hidden behind web authentication (“Basic Authentication”)

After configuring the crawler you will see the result of the crawl:

```
The following CGI(s) was/were found (with its parameters/values):  
  
1. CGI parameters: /cgi-bin/login2.cgi - username [Enter%20Username] password []  
2. CGI parameters: /cgi-bin/register2.cgi - username [] password [] desc [] pin []  
3. CGI parameters: /cgi-bin/search2.cgi - username [Username%20Here]  
  
We gathered (at least) 6 link(s), we went through 6 link(s).
```

***NOTE: The crawling process may be time consuming, and on large or complicated web sites may present a heavy load on the server. Make sure you run the crawling when the server is not busy, or during off-peak hours.***

You will then be able to change the list of server-side-scripts that were detected, in order to add/delete/modify them accordingly. To do this, modify the scan entry and click on the “change audit” button:





You will be presented with the configuration options, and in addition, see the list of server side script pages that were detected:

Path: /cgi-bin/login2.cgi	Method: POST	<input type="checkbox"/> Remove
Name: username	Value: Enter Username	
Name: password	Value:	
Path: /cgi-bin/register2.cgi	Method: POST	<input type="checkbox"/> Remove
Name: username	Value:	
Name: password	Value:	
Name: desc	Value:	
Name: pin	Value:	
Path: /cgi-bin/search2.cgi	Method: POST	<input type="checkbox"/> Remove
Name: username	Value: Username Here	
<input type="button" value="Update"/>		



In some cases you will need to manually add CGIs to the interface this is easily done via any of the below methods:

### URL Parser

To insert new CGIs provide a URL, which will be parsed out for CGIs. URLs containing CGI reference look like: <http://www.google.com/search?num=100&q=sample>

A screenshot of a web interface titled "URL Parser". It features a text input field labeled "URL:" and a button labeled "Parse URL" below it.

### Form Parser

Alternatively, you can provide an HTML code section containing a reference to one or more CGIs you are interested in testing. In addition to providing the HTML code, you need to provide a base URL to allow any relative directories referenced in the HTML code to be properly located during the actual testing phase:

A screenshot of a web interface titled "Form Parser". It features a text input field labeled "Base URL:" and a large text area labeled "HTML:" below it. A button labeled "Parse HTML" is located at the bottom right of the interface.



### Manual Addition

The third and last method of adding new CGIs is by manually providing the path where the CGI is located, the method of accessing it (GET/POST) followed by all the parameter's names and their corresponding values.

A screenshot of a web application interface titled "Manual Add". It contains a "Path:" text box, a "Method:" dropdown menu set to "GET", a "Name:" text box, and a "Value:" text box. Below these are two buttons: "Add another CGI" and "Add another Name/Value combination". At the bottom center is an "Add" button.

### Test Scheduling

The tests will be performed just like the network-level tests, on a pre-scheduled basis with the possibility of differential reporting. All other options are the same as the regular scanning.

### Test Results

Results generated by the Web Site Security Audit module are separated into two parts first a list of all the CGIs and their corresponding names and parameters is provided:

A screenshot of a security report window titled "2. Layer 7 Security List". The window has a close button in the top right corner. The text inside reads: "Hosts affected: 192.168.1.49 (port: http (80/tcp))". Below this, it says "The following CGIs were tested:" followed by a list of three CGI paths and their parameters: 1. /cgi-bin/search2.cgi - username [Username%20Here], 2. /cgi-bin/register2.cgi - username [] password [] desc [] pin [], 3. /cgi-bin/login2.cgi - username [Enter%20Username] password []. At the bottom, there is an "Impact:" section stating: "This is an intelligence gathering test only. The information found here will be used in layer 7 (application level) checks."



This is followed by another list of all the CGIs that have been discovered to contain a vulnerability:

**1. Layer 7 Security Checks**  
**Hosts affected: 192.168.1.49 (port: http (80/tcp))**

The following CGIs with their values, seem to contain a vulnerability:

SQL Injection - via POST - /cgi-bin/login2.cgi?password=l7checks&username='  
SQL Injection - via POST - /cgi-bin/login2.cgi?password=l7checks&username=%27  
SQL Injection - via POST - /cgi-bin/login2.cgi?username=Enter%20Username&password='  
SQL Injection - via POST - /cgi-bin/login2.cgi?username=Enter%20Username&password=%27

In those cases where the method used was POST, a proper HTTP POST request must be used. In those cases where the method used was GET, a simple HTTP URL can be used to verify the vulnerability.

**Impact:** Attackers can take control over your database, and in some cases over the operating system (using master..xp\_cmdshell, CREATE LIBRARY, etc).

**Possible Solution:** Filter out any user provided data from inappropriate data (especially ' ) | , etc).

**For More Information:**  
See <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>,  
<http://www.securiteam.com/securityreviews/5UP010A6AA.html>, <http://www.securiteam.com/securityreviews/5IP030K8AA.html>,  
and <http://www.securiteam.com/securityreviews/5GP0E2K7FO.html>