

TECHNICAL BRIEF

QUALYSGUARD PCI 3.0 — SUPPORT FOR PCI REQUIREMENT 6.6

Overview of PCI Requirement 6.6

The Payment Card Industry Data Security Standard (PCI DSS) requirement 6.6 deals with security of Web applications. The requirement calls for securing Web-facing applications against known attacks: SQL Injection, Cross-Site Scripting, etc. In PCI DSS 1.2 this is a mandatory requirement starting October 1st, 2008.

The PCI Council published a clarification to requirement 6.6 in PCI DSS 1.2, which states that either of the following procedures performed by a person trained in that area can be used to satisfy requirement 6.6:

- Regular use of automated or manual application vulnerability assessment tools or methods (can be performed by trained, independent internal team or specialized external tools/organizations).
- Properly configured web application firewall with capabilities in line with the special 6.6 information supplement released by the council.

To further elaborate, the intent of requirement 6.6 is to ensure that Web applications are built and maintained in a secure way. And a crucial part of those efforts is making sure Web applications, whether built in-house or purchased from a vendor, are tested for potential vulnerabilities. It's also important to use automated tools that will evaluate Web applications before they're deployed in production. And once they are, regular scans should be conducted not only as part of a normal security and compliance program, but every time applications and/or the servers hosting the applications are changed or updated. Finally, the council also clarified that source code reviews are no longer an option in PCS DSS 1.2 to satisfy requirement 6.6.

QualysGuard PCI Solution for Requirement 6.6

Qualys is expanding its QualysGuard PCI solution to help customers meet the new PCI requirement in order to maintain Web applications securely. The intent of this document is to describe this solution in detail and highlight its features.

Web applications are structured in three layers. Typically, the first layer would be a Web server, the second would be a content generation technology tool such as Java servlets or ASP (Active Server Pages), and the third layer would be one or more compatible databases.

Web Application Scanning looks for a variety of vulnerability types within customized code that can consist of shopping carts, forms, login pages, and other types of dynamic content – all examples of Web Applications.

QualysGuard PCI 3.0 provides an automated Web Application Scanning (WAS) module that allows customers to crawl Web applications, spot cross-site scripting vulnerabilities, spot SQL injection, and conduct authenticated and unauthenticated scanning to capture the perspective of both authorized and unauthorized users. The WAS module works with standard HTML and also supports technologies like JavaScript and AJAX. Results of such an automated scan need to be reviewed and interpreted for remediation steps by a security professional inside or outside your organization. QualysGuard PCI provides a workflow for end users to be able to request such a review. This solution is delivered via Qualys' Software-as-a-Service (SaaS) subscription model, which offers economies of scale and allows customers to scan external web applications without having to deploy additional infrastructure, software, or manage it.

QualysGuard PCI Web Application Scanning Module: Architecture and Features

The QualysGuard PCI WAS solution automates the techniques used to identify most web vulnerabilities such as those in the OWASP Top 10 and WASC-TC, including SQL Injection and Cross-Site Scripting. The WAS module combines pattern recognition and observed behaviors to accurately identify and verify vulnerabilities. Automation is a core feature of the WAS module that leads to a consistent, repeatable test framework. It also reduces the overhead of managing a scan configuration. The module identifies login forms, error pages, and other customized features of the target application on its own. This helps the web application scanner adapt to changes as a web site matures. Adaptability also enables the scanner to be used against unknown or legacy web applications for which little may be known about their error pages or other behavior.

The QualysGuard PCI WAS solution provides comprehensive capabilities to assess and track web application vulnerabilities across distinctly different web sites. The module includes the following features:

- Profiles the target application to determine custom error behaviors and login forms.
- Crawling algorithm that balances breadth and depth of links in order to obtain wide coverage of the target application.
- Works with multiple character sets and internationalized text within HTML content.
- Automatically authenticates to HTML forms and monitors its session state.
- Combines pattern and behavior analysis to improve accuracy and reduce false positives.
- Accurate identification of SQL Injection and Cross-Site Scripting (XSS) vulnerabilities.
- Workflow for expert review and approval.
- Tightly integrated with QualysGuard PCI; no additional hardware or software resources necessary.

QualysGuard PCI Web Application Scanning Module: Phase I (October 2008)

Features of QualysGuard PCI WAS module will be released in three phases. Phase I includes the following features:

1. Crawling & Link Discovery

- The embedded web crawler parses HTML and extracts static links. It also extracts some JavaScript-based links and has the capability to find custom links. The crawler automatically

balances the breadth and depth of the site. This enables the crawler to obtain a high degree of site coverage while avoiding redundant or recursive links.

- The crawler crawls up to 5000 links per web application (the number of links include form submissions and links requested as an anonymous user and links requested as an authenticated user).
- The crawler crawls an application under a single host name or IP address. The application can consist of a single physical host or multiple identical hosts behind a single load-balanced IP address.

2. Authentication

The following authentication techniques will be supported:

- HTTP Basic and HTTP NTLM authentication: Server based authentication using either of these protocols.
- Simple Form authentication: Simple forms contain a username and password field. The scanner automatically identifies and populates these forms.
- Login forms that use JavaScript to modify properties or values upon form submission are not handled automatically. Users can manually provide a session cookie to bypass the authentication process.
- CAPTCHA and one-time passwords authentication are not supported. These are either designed to block automated scans or have time limits that inhibit automated scans.

3. Black List

- This feature prevents the crawler from visiting certain links in your application. On a production Web application, this lets you specify pages that should not be crawled to prevent them from possibly sending out too many emails or potentially submitting a 'delete all' button.

4. Performance Tuning

- This feature allows the user to pick the bandwidth level as to how many parallel scans can be conducted against the host and the delay between scan processing requests. This helps control the impact of the scan on your web application.
- Crawl only option that allows the scanner to crawl the application and catalog the links without performing security checks.

5. Workflows for Setting Scans and Reviewing Reports

PCI WAS scans are treated independently of vulnerability scans and hence they come with new workflows for setting up Web applications, configuring WAS scans and reviewing WAS scan results. The following screen shots for the product illustrate the new workflows:

- QualysGuard PCI dashboard and catalog for managing web applications



Figure 1 – New QG PCI Dashboard to Support WAS Scanning for PCI Req. 6.6

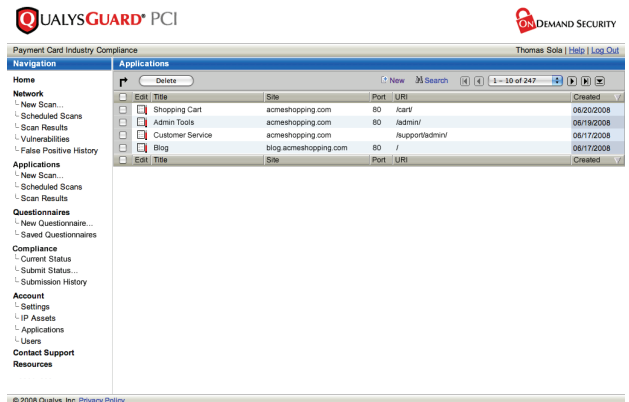


Figure 2 – Web Applications Catalog Per Account

- QualysGuard PCI workflows to create web applications and define scans

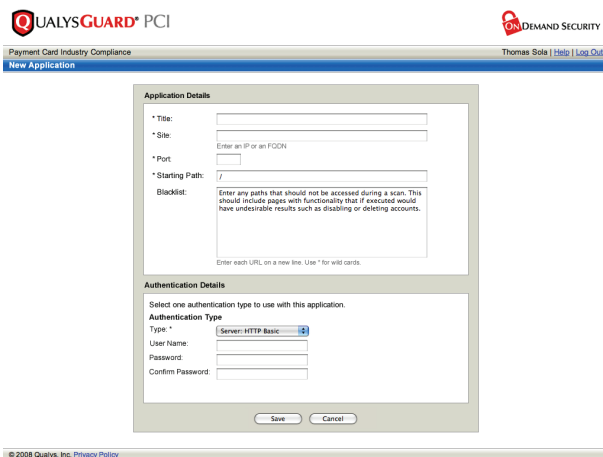


Figure 3 – Configuring a PCI Web Application: Form, HTTP or NTLM Authentication

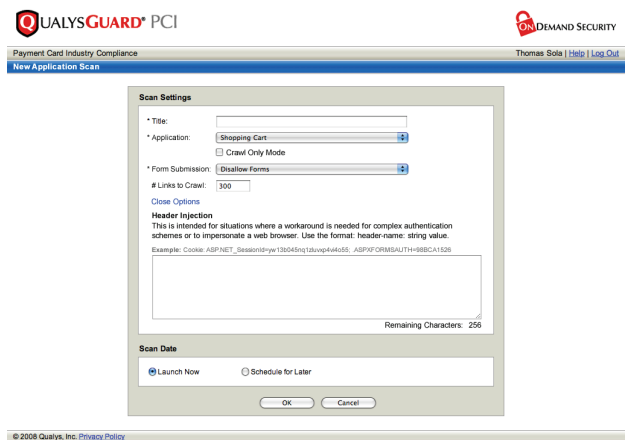



Figure 4 – Setting Up & Launching a PCI WAS Scan

- QualysGuard PCI WAS results. Appendix A provides additional details for WAS QIDs

Scan Results

File ▾ View ▾



Scan Results

Mike Shema
quays_ms2
Manager

July 08, 2008

Created: 07/08/2008 at 12:22:25 (GMT-0700)

Qualys, Inc.
1600 Bridge Pkwy
Redwood Shores, California 94065
United States of America

Report Summary

Date: 07/08/2008 at 10:30:20 (GMT-0700)

Active Hosts: 1

Total Hosts: 1

Type: On demand

Status: Finished

Reference: scan/1215538221.29335

Scanner Appliance: 10.10.21.22 (Scanner 4.11.60-1, Web 6.0 FR5 [build 6.3.45-1], Vulnsigs 1.19.183-1)

Duration: 00:29:26

Title: PCI test

Asset Groups: -

IPs: 10.10.25.61

Option Profile: [Web Application - Attack](#)

Details

10.10.25.61 (-, -) Linux 2.4-2.6 / Embedded Device

Vulnerabilities (2) pcitest.vuln.qa.qualys.com:80/tcp

▼ SQL Injection

QID: 150003

Category: Web Application

CVE ID: -

Vendor Reference: -

Bugtraq ID: -

Modified: 10/18/2007

Edited: No

THREAT:

SQL injection enables an attacker to modify the syntax of a SQL query in order to retrieve, corrupt or delete data. This is accomplished by manipulating query criteria in a manner that affects the query's logic. The typical causes of this vulnerability are lack of input validation and insecure construction of the SQL query.

Queries created by concatenating strings with SQL syntax and user-supplied data are prone to this vulnerability. If any part of the string concatenation can be modified, then the meaning of the query can be changed.

Examples:

These two lines demonstrate an insecure query that is created by appending the user-supplied data (`userid`):

```
dim strQuery as String
strQuery = "SELECT name,email FROM users WHERE userid=" + Request.QueryString("userid")
```

If no checks are performed against the `userid` parameter, then the query may be arbitrarily modified as shown in these two examples of a completed query:

```
SELECT name,email FROM users WHERE userid=42
SELECT name,email FROM users WHERE userid=42; SHUTDOWN WITH NOWAIT
```

IMPACT:

The scope of a SQL injection exploit varies greatly. If any SQL statement can be injected into the query, then the attacker has the equivalent access of a database administrator. This access could lead to theft of data, malicious corruption of data, or deletion of data.

SOLUTION:

SQL injection vulnerabilities can be addressed in three areas: input validation, query creation, and database security.

All input received from the Web client should be validated for correct content. If a value's type or content range is known beforehand, then stricter filters should be applied. For example, an email address should be in a specific format and only contain characters that make it a valid address; or numeric fields like a U.S. zip code should be limited to five digit values.

Prepared statements (sometimes referred to as parameterized statements) provide strong protection from SQL injection. Prepared statements are precompiled SQL queries whose parameters can be modified when the query is executed. Prepared statements enforce the logic of the query and will fail if the query cannot be compiled correctly. Programming languages that support prepared statements provide specific functions for creating queries. These functions are more secure than string concatenation for assigning user-supplied data to a query.

Stored procedures are precompiled queries that reside in the database. Like prepared statements, they also enforce separation of query data and logic. SQL statements that call stored procedures should not be created via string concatenation, otherwise their security benefits are negated.

SQL injection exploits can be mitigated by the use of Access Control Lists or role-based access within the database. For example, a read-only account would prevent an attacker from modifying data, but would not prevent the user from viewing unauthorized data. Table and row-based access controls potentially minimize the scope of a compromise, but they do not prevent exploits.

Example of a secure query created with a prepared statement:

```
PreparedStatement ps = "SELECT name,email FROM users WHERE userid=?"; ps.setInt(1, userid);
```

COMPLIANCE:

Not Applicable

RESULTS:

url: http://10.10.25.61:80/recipe/recipe_view.php?intid=%22%3e%3cqs%3e

variants: 119

matched: aseException' with message 'MySQL Error: You have an error in your SQL syntax; check the manual that corresponds to you

► Blind SQL Injection pcitest.vuln.qa.qualys.com:80/tcp

Information Gathered (7) pcitest.vuln.qa.qualys.com:80/tcp

- Operating System Detected
- DNS Host Name
- Host Scan Time
- Open TCP Services List
- Web Application Authentication Failed
- Links Crawled
- External Links Discovered

Figure 5 – PCI WAS Scan Report Dedicated for SQL Injections and XSS Vulnerabilities

Road Map

Additional advanced features for QualysGuard PCI WAS module will be delivered in Phases II and III per the following schedule:

Phase II Feature (target GA January 2009):

1. White List

- Ability to list specific URLs to scan directly and not depend on the crawler to identify them.
- This feature enables the user to specify a small, specific set of links that should be scanned by the crawler. This is an alternative to using a Black List where it may be more convenient to define explicit links to request rather than create a list of links that should not be requested.

2. Multi-Site Support

- This provides the ability to crawl a logically-defined web application that consists of multiple sites. This feature enables the user to define scans across sites that use multiple host names or sub-domains. For example, a site may require users to authenticate to one host in the domain, then carry the session cookie to a different host within the same domain. (Note that this distinction is for a web application whose business logic and capabilities are spread across multiple sites. Any web application that uses load balancers or other traffic distribution methods can be crawled by the scanner's default behavior.)

3. Additional Authentication

- Complex Form authentication: Complex forms contain a username, password, and one or more fields. The additional fields are not automatically populated by the scanner. The user can supply the name/value pairs when defining scan settings.

4. Additional Reporting

- Filter vulnerabilities based on URL and parameter name.
- Vulnerabilities can be filtered based on QID (e.g. cross-site scripting, SQL injection) or link (e.g. each different QID associated with a link). Within each link the vulnerable parameter is identified, along with the payload used to indicate the vulnerability, and a small sample of the web server's response that highlights the specifics of the vulnerability. This will let customers group together all vulnerabilities by link.

5. Review Workflow

- This embedded workflow allows a customer to review the results with a security professional inside or outside their organization.
- The reviewer has access to the scan report, the credentials used by the crawler and the target (this assumes the target is an Internet-facing application which is the case for applications in scope for PCI).
- The reviewer can identify and remove false positives from scan results.

- The reviewer can group results into common themes or areas not already identified by the scanner. For example, the reviewer can be informed that input is not properly sanitized across several pages and can identify that several vulnerabilities are due to a single underlying cause.
- The reviewer can interpret the relative risk of each vulnerability and assign a priority for remediation.

Phase III Feature Include (target GA Q2 2009):

1. Advanced JavaScript Support

- Crawl sites that rely on the XMLHttpRequest object (AJAX) and use complex JavaScript for navigation and form submission.
- Extract links created via JavaScript using string concatenation and variable substitution.

2. Sensitive Content

- The feature allows the user to specify a regular expression search for sensitive content in HTML. This is useful to customers to be able to scan HTML for things like Credit Card Number or Social Security Numbers or any other patterns that may be security violations.

Appendix A: QualysGuard WAS QIDs

Qualys has created specific QualysGuard QIDs that allow the user to examine Web applications with an eye toward discovering common vulnerability types. These QIDs allow the user the flexibility to run Web application scans separately from vulnerability scans.

The QIDs and a brief description of each are listed below:

150000	Persistent Cross-Site Scripting (XSS) Vulnerabilities
XSS attacks insert malicious content into the HTML response from the Web application in order to attack the victim's browser. This XSS attack is stored by the Web application and could affect any user that visits the vulnerable page.	
150001	Reflected Cross-Site Scripting (XSS) Vulnerabilities
XSS attacks insert malicious content into the HTML response from the Web application in order to attack the victim's browser. The victim must be tricked into submitting a request with the malicious payload.	
150002	Header Cross-Site Scripting (XSS)
XSS attacks insert malicious content into the HTML response from the Web application in order to attack the victim's browser. In this case, the Web application does not properly filter the browser's header fields.	
150003	SQL Injection
XSS attacks insert malicious content into the HTML response from the web application in order to attack the victim's browser. The victim must be tricked into submitting a request with the malicious payload.	
150004	Path-based Vulnerability
A sensitive file, source code, or browseable directory exists on the Web server.	

150005	Nonspecific Web Application Vulnerability
The web application returned an error in response to a security test. The root cause of the error (typically an HTTP 500 response code) cannot be determined.	
150006	Web Application Authentication Not Attempted
Authentication credentials were provided; however, the authentication record forbids credentials from being sent via cleartext protocols and the login form is submitted via HTTP.	
150007	Web Application Authentication Method
Authentication succeeded.	
150008	Web Application Authentication Failed
Authentication failed.	
150009	Links Crawled
This list represents all of the links crawled by the Web application scanner.	
150010	External Links Discovered
This represents all of the external links discovered by the Web application scanner. These links were present in the target Web application, but were not crawled.	
150011	Local File Inclusion
Local file inclusion can expose sensitive files through the Web application. Typically, any file that is readable by the user privileges associated with the Web service can be exposed. This can divulge source code, database configurations, and other sensitive information inside or outside the Web document root.	
150012	Blind SQL Injection
Blind SQL injection is a specialized type of SQL injection. It enables an attacker to modify the syntax of a SQL query in order to retrieve, corrupt or delete data. This is accomplished by manipulating query criteria in a manner that affects the query's logic. The typical causes of this vulnerability are lack of input validation and insecure construction of the SQL query.	
150013	Browser-Specific Cross-Site Scripting (XSS)
XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contains characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.	
150014	External Form Actions Discovered
The external form actions discovered by the Web application scanner are provided in the Result section. These links were present within HTML forms on the target Web application, but were not crawled.	