Hewlett Packard Enterprise

HPE Security Fortify Audit Workbench

Developer Workbook

testcms-final-anon

FORTIFY

Table of Contents

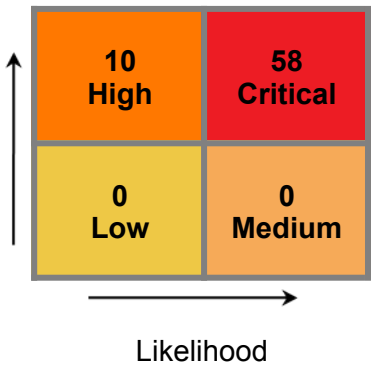
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the testcms-final-anon project audit. The information contained in this workbook is targeted at project managers and developers.

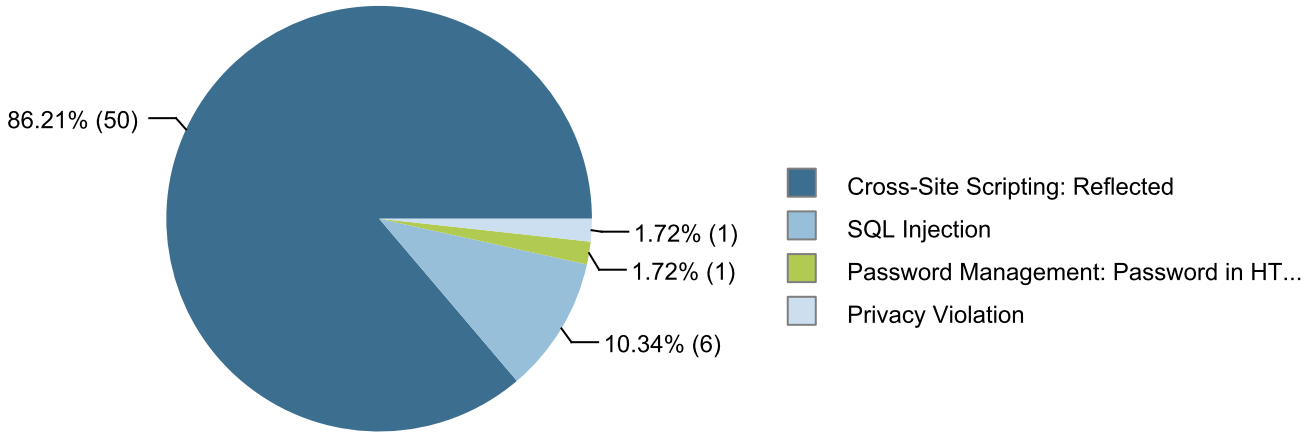
This section provides an overview of the issues uncovered during analysis.

Project Name:	testcms-final-anon
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
SecurityScope:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories



Project Description

This section provides an overview of the HPE Security Fortify scan engines used for this project, as well as the project meta-information.

SCA

Date of Last Analysis:	Nov 9, 2016, 1:19 PM	**Engine Version:**	16.10.0095
Host Name:	mrl-PC	**Certification:**	VALID
Number of Files:	92	**Lines of Code:**	3,731

Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
Cookie Security: HTTPOnly not Set	0	0 / 1	0	0	0 / 1
Cross-Site Scripting: Reflected	0 / 50	0	0	0	0 / 50
Key Management: Empty Encryption Key	0	0 / 1	0	0	0 / 1
Password Management: Empty Password	0	0 / 1	0	0	0 / 1
Password Management: Password in HTML Form	0 / 1	0	0	0	0 / 1
Privacy Violation	0 / 1	0	0	0	0 / 1
Privacy Violation: Autocomplete	0	0 / 2	0	0	0 / 2
SQL Injection	0 / 6	0	0	0	0 / 6
Weak Encryption	0	0 / 5	0	0	0 / 5

Results Outline

Cookie Security: HTTPOnly not Set (1 issue)

Abstract

The program creates a cookie, but fails to set the `HttpOnly` flag to `true`.

Explanation

All major browsers support the `HttpOnly` cookie property that prevents client-side scripts from accessing the cookie. Cross-site scripting attacks often access cookies in an attempt to steal session identifiers or authentication tokens. When `HttpOnly` is not enabled, attackers may more easily access user cookies.

Example 1: The code in the example below creates a cookie without setting the `HttpOnly` property.

```
setcookie("emailCookie", $email, 0, "/", "www.example.com", TRUE);  //Missing
7th parameter to set HttpOnly
```

Recommendation

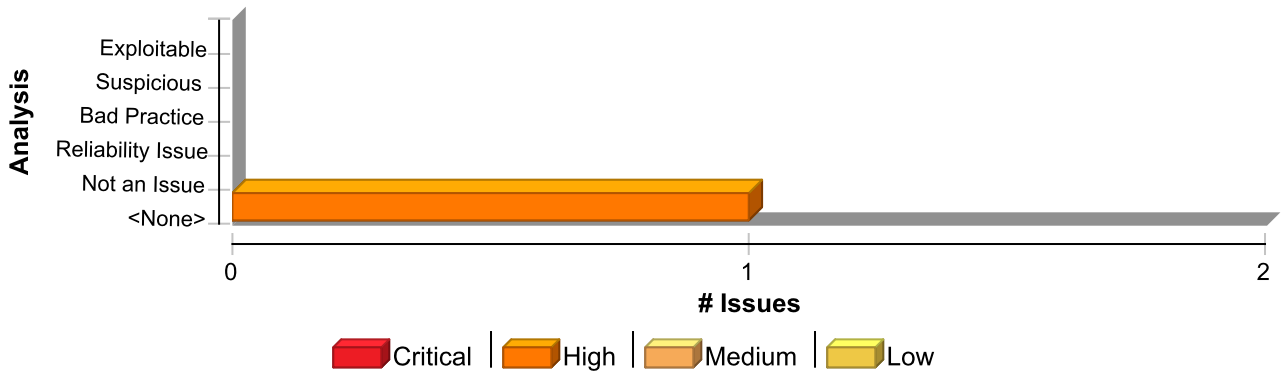
Enable the `HttpOnly` property when creating cookies. This can be done by setting the `HttpOnly` parameter in the `setcookie()` call to `true`.

Example 2: The code in the example below creates the same cookie as the code in Example 1, but this time sets the `HttpOnly` parameter to `true`.

```
setcookie("emailCookie", $email, 0, "/", "www.example.com", TRUE, TRUE);
```

Do not be lulled into a false sense of security by `HttpOnly`. As several mechanisms for bypassing it have been developed, it is not completely effective.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Cookie Security: HTTPOnly not Set	1	0	0	1
Total	**1**	**0**	**0**	**1**

Cookie Security: HTTPOnly not Set — High

Package: system.classes

system/classes/cookie.php, line 19 (Cookie Security: HTTPOnly not Set) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: setcookie
Enclosing Method: write()
File: system/classes/cookie.php:19
Taint Flags:

```
16
17  public static function write($name, $data, $expire, $path, $domain) {
18  if(headers_sent() === false) {
19  return setcookie($name, $data, $expire, $path, $domain, false);
20  }
21
22  return false;
```

Cross-Site Scripting: Reflected (50 issues)

Abstract

Sending unvalidated data to a web browser can result in the browser executing malicious code.

Explanation

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of Reflected XSS, the untrusted source is typically a web request, while in the case of Persisted (also known as Stored) XSS it is typically a database or other back-end datastore.

2. The data is included in dynamic content that is sent to a web user without being validated.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash or any other type of code that the browser may execute. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following PHP code segment reads an employee ID, `eid`, from an HTTP request and displays it to the user.

```
...
```

The code in this example operates correctly if `eid` contains only standard alphanumeric text. If `eid` has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

Example 2: The following PHP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

As in Example 1, this code functions correctly when the values of `name` are well-behaved, but it does nothing to prevent exploits if they are not. Again, this code can appear less dangerous because the value of `name` is read from a database, whose contents are apparently managed by the application. However, if the value of `name` originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker may execute

malicious commands in the user's web browser. This type of exploit, known as Persistent (or Stored) XSS, is particularly insidious because the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in Example 1, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that may include session information, from the user's machine to the attacker or perform other nefarious activities.

- As in Example 2, the application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

Recommendation

The solution to XSS is to ensure that validation occurs in the correct places and checks for the correct properties.

Since XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate their input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application may accept input through a shared data store or other trusted source, and that data store may accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application destined for the user.

The most secure approach to validation for XSS is to create a whitelist of safe characters that are allowed to appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alpha-numeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters

that have special meaning to the browser should still be considered valid input once they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using the input. In order to form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines what characters have special meaning, many web browsers try to correct common mistakes in HTML and may treat other characters as special in certain contexts, which is why we do not encourage the use of blacklists as a means to prevent XSS. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed with double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed with single quote, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything above 128 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C %6F" becomes "hello" when it appears on the web page in question.

Within the body of a :

- Semicolons, parentheses, curly braces, and new line characters should be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

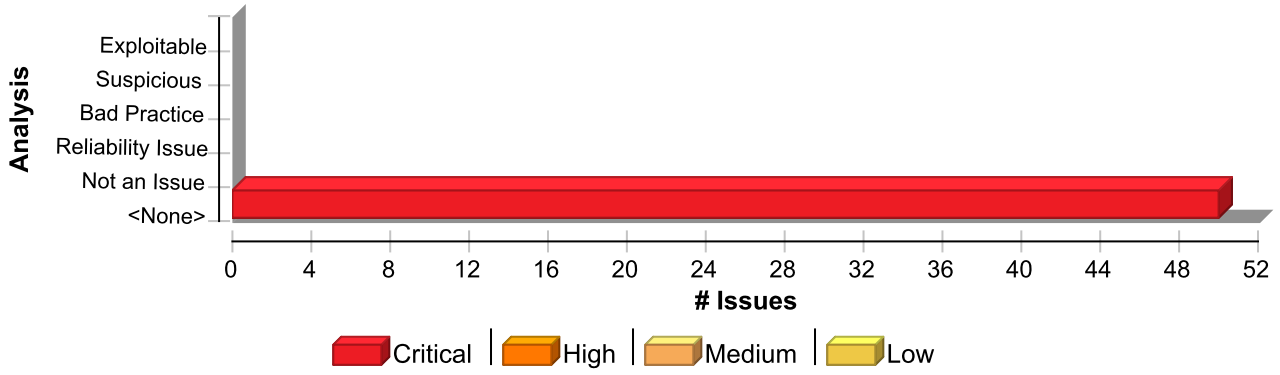
- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and may bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

Once you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option in this situation is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and may be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. When an application is developed there are no guarantees about what application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will also stay in sync.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Cross-Site Scripting: Reflected	50	0	0	50
Total	**50**	**0**	**0**	**50**

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme

system/admin/theme/error_php.php, line 67 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
File: system/admin/theme/error_php.php:67

```
64  <li>Operating System: <?php echo php_uname(); ?></li>
65  <li>Server Software: <?php echo $_SERVER['SERVER_SOFTWARE']; ?></li>
66  <li>User Agent: <?php echo $_SERVER['HTTP_USER_AGENT']; ?></li>
67  <li>Request Uri: <?php echo $_SERVER['REQUEST_URI']; ?></li>
68  </ul>
69  </div>
70  </body>
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/error_php.php:67
Taint Flags: WEB, XSS

```
64  <li>Operating System: <?php echo php_uname(); ?></li>
65  <li>Server Software: <?php echo $_SERVER['SERVER_SOFTWARE']; ?></li>
66  <li>User Agent: <?php echo $_SERVER['HTTP_USER_AGENT']; ?></li>
67  <li>Request Uri: <?php echo $_SERVER['REQUEST_URI']; ?></li>
68  </ul>
69  </div>
70  </body>
```

system/admin/theme/error_php.php, line 66 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['HTTP_USER_AGENT']
File: system/admin/theme/error_php.php:66

```
63  <li>PHP Version: <?php echo phpversion(); ?></li>
64  <li>Operating System: <?php echo php_uname(); ?></li>
65  <li>Server Software: <?php echo $_SERVER['SERVER_SOFTWARE']; ?></li>
66  <li>User Agent: <?php echo $_SERVER['HTTP_USER_AGENT']; ?></li>
67  <li>Request Uri: <?php echo $_SERVER['REQUEST_URI']; ?></li>
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme

system/admin/theme/error_php.php, line 66 (Cross-Site Scripting: Reflected) — Critical

```
68  </ul>
69  </div>
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/error_php.php:66
Taint Flags: WEB, XSS

```
63  <li>PHP Version: <?php echo phpversion(); ?></li>
64  <li>Operating System: <?php echo php_uname(); ?></li>
65  <li>Server Software: <?php echo $_SERVER['SERVER_SOFTWARE']; ?></li>
66  <li>User Agent: <?php echo $_SERVER['HTTP_USER_AGENT']; ?></li>
67  <li>Request Uri: <?php echo $_SERVER['REQUEST_URI']; ?></li>
68  </ul>
69  </div>
```

Package: system.admin.theme.metadata

system/admin/theme/metadata/index.php, line 53 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/metadata/index.php:53
Taint Flags: WEB, XSS

```
50
51  <p>
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.metadata

system/admin/theme/metadata/index.php, line 53 (Cross-Site Scripting: Reflected) — Critical

```
52  <label for="posts_per_page">Posts per page:</label>
53  <input id="posts_per_page" name="posts_per_page" value="<?php echo
Input::post('posts_per_page', $metadata->posts_per_page); ?>">
54
55  <em>The number of posts to display per page.</em>
56  </p>
```

system/admin/theme/metadata/index.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/metadata/index.php:11
Taint Flags: WEB, XSS

```
8  <fieldset>
9  <p>
10  <label for="sitename">Site name:</label>
11  <input id="sitename" name="sitename" value="<?php echo Input::post('name', $metadata-
>sitename); ?>">
12
13  <em>Your site&rsquo;s name.</em>
14  </p>
```

system/admin/theme/metadata/index.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.metadata

system/admin/theme/metadata/index.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/metadata/index.php:18
Taint Flags: WEB, XSS

```
15
16   <p>
17   <label for="description">Site description:</label>
18   <textarea id="description" name="description"><?php echo Input::post('description',
$metadata->description); ?></textarea>
19
20   <em>A short paragraph to describe your site.</em>
21   </p>
```

system/admin/theme/metadata/index.php, line 80 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.metadata

system/admin/theme/metadata/index.php, line 80 (Cross-Site Scripting: Reflected) — Critical

```
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/metadata/index.php:80
Taint Flags: WEB, XSS

```
77
78  <p>
79  <label for="twitter">Twitter:</label>
80  <input id="twitter" name="twitter" value="<?php echo Input::post('twitter', $metadata->twitter); ?>">
81
82  <em>Your twitter account. Displayed as @<span id="output"><?php echo $metadata->twitter; ?></span>.</em>
83  </p>
```

Package: system.admin.theme.pages

system/admin/theme/pages/add.php, line 33 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/add.php, line 33 (Cross-Site Scripting: Reflected) — Critical

File: system/admin/theme/pages/add.php:33
Taint Flags: WEB, XSS

```
30
31  <p>
32  <label for="content">Content:</label>
33  <textarea id="content" name="content"><?php echo Input::post('content'); ?></textarea>
34
35  <em>Your page's content. Accepts valid HTML.</em>
36  </p>
```

system/admin/theme/pages/add.php, line 12 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/add.php:12
Taint Flags: WEB, XSS

```
9  <fieldset>
10  <p>
11  <label for="name">Name:</label>
12  <input id="name" name="name" value="<?php echo Input::post('name'); ?>">
13
14  <em>The name of your page. This gets shown in the navigation.</em>
15  </p>
```

system/admin/theme/pages/add.php, line 19 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/add.php, line 19 (Cross-Site Scripting: Reflected) — Critical

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/add.php:19
Taint Flags: WEB, XSS

```
16
17   <p>
18   <label>Title:</label>
19   <input id="title" name="title" value="<?php echo Input::post('title'); ?>">
20
21   <em>The title of your page, which gets shown in the <code>&lt;title&gt;</code>.</em>
22   </p>
```

system/admin/theme/pages/edit.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/edit.php, line 18 (Cross-Site Scripting: Reflected) — Critical

```
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/edit.php:18
Taint Flags: WEB, XSS

```
15
16  <p>
17  <label>Title:</label>
18  <input id="title" name="title" value="<?php echo Input::post('title', $page->title); ?>">
19
20  <em>The title of your page, which gets shown in the <code>&lt;title&gt;</code>.</em>
21  </p>
```

system/admin/theme/pages/add.php, line 28 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['HTTP_HOST']
File: system/admin/theme/pages/add.php:28

```
25  <label for="slug">Slug:</label>
26  <input id="slug" autocomplete="off" name="slug" value="<?php echo
Input::post('slug'); ?>">
27
28  <em>The slug for your post (<code><?php echo $_SERVER['HTTP_HOST']; ?>/
<span id="output">slug</span></code>).</em>
29  </p>
30
31  <p>
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/add.php:28
Taint Flags: WEB, XSS

```
25  <label for="slug">Slug:</label>
26  <input id="slug" autocomplete="off" name="slug" value="<?php echo Input::post('slug'); ?>">
27
28  <em>The slug for your post (<code><?php echo $_SERVER['HTTP_HOST']; ?>/<span
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/add.php, line 28 (Cross-Site Scripting: Reflected) — Critical

```
id="output">slug</span></code>).</em>
29  </p>
30
31  <p>
```

system/admin/theme/pages/add.php, line 26 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/add.php:26
Taint Flags: WEB, XSS

```
23
24  <p>
25  <label for="slug">Slug:</label>
26  <input id="slug" autocomplete="off" name="slug" value="<?php echo Input::post('slug'); ?>">
27
28  <em>The slug for your post (<code><?php echo $_SERVER['HTTP_HOST']; ?>/<span
id="output">slug</span></code>).</em>
29  </p>
```

system/admin/theme/pages/edit.php, line 25 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/edit.php, line 25 (Cross-Site Scripting: Reflected) — Critical

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/edit.php:25
Taint Flags: WEB, XSS

```
22
23   <p>
24   <label for="slug">Slug:</label>
25   <input id="slug" autocomplete="off" name="slug" value="<?php echo Input::post('slug', $page->slug); ?>">
26
27   <em>The slug for your page (<code id="output">slug</code>).</em>
28   </p>
```

system/admin/theme/pages/edit.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.pages

system/admin/theme/pages/edit.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/edit.php:11
Taint Flags: WEB, XSS

```
8  <fieldset>
9  <p>
10  <label for="name">Name:</label>
11  <input id="name" name="name" value="<?php echo Input::post('name', $page->name); ?>">
12
13  <em>The name of your page. This gets shown in the navigation.</em>
14  </p>
```

system/admin/theme/pages/edit.php, line 32 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/pages/edit.php:32
Taint Flags: WEB, XSS

```
29
30  <p>
31  <label for="content">Content:</label>
32  <textarea id="content" name="content"><?php echo Input::post('content', $page->content); ?
></textarea>
33
34  <em>Your page's content. Accepts valid HTML.</em>
35  </p>
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 77 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:77
Taint Flags: WEB, XSS

```
74
75   <p>
76   <label for="css">Custom CSS:</label>
77   <textarea id="css" name="css"><?php echo Input::post('css'); ?></textarea>
78
79   <em>Custom CSS. Will be wrapped in a <code>&lt;style&gt;</code> block.</em>
80   </p>
```

system/admin/theme/posts/add.php, line 84 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 84 (Cross-Site Scripting: Reflected) — Critical

```
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:84
Taint Flags: WEB, XSS

```
81
82  <p>
83  <label for="js">Custom JS:</label>
84  <textarea id="js" name="js"><?php echo Input::post('js'); ?></textarea>
85
86  <em>Custom Javascript. Will be wrapped in a <code>&lt;script&gt;</code> block.</em>
87  </p>
```

system/admin/theme/posts/add.php, line 21 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:21
Taint Flags: WEB, XSS

```
18  <fieldset>
19  <p>
20  <label for="title">Title:</label>
21  <input id="title" name="title" value="<?php echo Input::post('title'); ?>">
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 21 (Cross-Site Scripting: Reflected) — Critical

```
22
23  <em>Your post&rsquo;s title.</em>
24  </p>
```

system/admin/theme/posts/edit.php, line 117 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:117
Taint Flags: WEB, XSS

```
114  <?php foreach(Input::post('field', array()) as $data => $value): ?>
115  <?php list($key, $label) = explode(':', $data); ?>
116  <p>
117  <label><?php echo $label; ?></label>
118  <input name="field[<?php echo $key; ?>:<?php echo $label; ?>]" value="<?php echo $value; ?>">
119  </p>
120  <?php endforeach; ?>
```

system/admin/theme/posts/edit.php, line 49 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/edit.php, line 49 (Cross-Site Scripting: Reflected) — Critical

From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:49
Taint Flags: WEB, XSS

```
46
47   <p>
48   <label for="html">Content:</label>
49   <textarea id="html" name="html"><?php echo Input::post('html', $article->html); ?></
textarea>
50
51   <em>Your post's main content. Enjoys a healthy dose of valid HTML.</em>
52   </p>
```

system/admin/theme/posts/edit.php, line 28 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/edit.php, line 28 (Cross-Site Scripting: Reflected) — Critical

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:28
Taint Flags: WEB, XSS

```
25  <fieldset>
26  <p>
27  <label for="title">Title:</label>
28  <input id="title" name="title" value="<?php echo Input::post('title', $article->title); ?>">
29
30  <em>Your post&rsquo;s title.</em>
31  </p>
```

system/admin/theme/posts/edit.php, line 42 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:42
Taint Flags: WEB, XSS

```
39
40  <p>
41  <label for="description">Description:</label>
42  <textarea id="description" name="description"><?php echo Input::post('description', $article->description); ?></textarea>
43
44  <em>A brief outline of what your post is about. Used in the post introduction, RSS feed, and <code>&lt;meta name="description" /&gt;</code>.</em>
45  </p>
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/edit.php, line 35 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:35
Taint Flags: WEB, XSS

```
32
33   <p>
34   <label for="slug">Slug:</label>
35   <input type="url" id="slug" autocomplete="off" name="slug" value="<?php echo
Input::post('slug', $article->slug); ?>">
36
37   <em>The slug for your post (<code id="output">slug</code>).</em>
38   </p>
```

system/admin/theme/posts/add.php, line 35 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 35 (Cross-Site Scripting: Reflected) — Critical

```
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:35
Taint Flags: WEB, XSS

```
32
33  <p>
34  <label for="description">Description:</label>
35  <textarea id="description" name="description"><?php echo Input::post('description'); ?></
textarea>
36
37  <em>A brief outline of what your post is about. Used in the post introduction, RSS feed,
and <code>&lt;meta name="description" /&gt;</code>.</em>
38  </p>
```

system/admin/theme/posts/add.php, line 28 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:28
Taint Flags: WEB, XSS

```
25
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 28 (Cross-Site Scripting: Reflected) — Critical

```
26  <p>
27  <label for="slug">Slug:</label>
28  <input id="slug" autocomplete="off" name="slug" value="<?php echo Input::post('slug'); ?>">
29
30  <em>The slug for your post (<code id="output">slug</code>).</em>
31  </p>
```

system/admin/theme/posts/edit.php, line 84 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:84
Taint Flags: WEB, XSS

```
81
82  <p>
83  <label for="css">Custom CSS:</label>
84  <textarea id="css" name="css"><?php echo Input::post('css', $article->css); ?></textarea>
85
86  <em>Custom CSS. Will be wrapped in a <code>&lt;style&gt;</code> block.</em>
87  </p>
```

system/admin/theme/posts/add.php, line 103 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 103 (Cross-Site Scripting: Reflected) — Critical

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:103
Taint Flags: WEB, XSS

```
100  <?php list($key, $label) = explode(':', $data); ?>
101  <p>
102  <label><?php echo $label; ?></label>
103  <input name="field[<?php echo $key; ?>:<?php echo $label; ?>]" value="<?php echo $value; ?>">
104  </p>
105  <?php endforeach; ?>
106  </div>
```

system/admin/theme/posts/edit.php, line 91 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/edit.php, line 91 (Cross-Site Scripting: Reflected) — Critical

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:91
Taint Flags: WEB, XSS

```
88
89  <p>
90  <label for="js">Custom JS:</label>
91  <textarea id="js" name="js"><?php echo Input::post('js', $article->js); ?></textarea>
92
93  <em>Custom Javascript. Will be wrapped in a <code>&lt;script&gt;</code> block.</em>
94  </p>
```

system/admin/theme/posts/edit.php, line 118 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/edit.php:118
Taint Flags: WEB, XSS

```
115  <?php list($key, $label) = explode(':', $data); ?>
116  <p>
117  <label><?php echo $label; ?></label>
118  <input name="field[<?php echo $key; ?>:<?php echo $label; ?>]" value="<?php echo $value; ?>">
119  </p>
120  <?php endforeach; ?>
121  </div>
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/edit.php, line 118 (Cross-Site Scripting: Reflected) — Critical

system/admin/theme/posts/add.php, line 102 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:102
Taint Flags: WEB, XSS

```
99  <?php foreach(Input::post('field', array()) as $data => $value): ?>
100  <?php list($key, $label) = explode(':', $data); ?>
101  <p>
102  <label><?php echo $label; ?></label>
103  <input name="field[<?php echo $key; ?>:<?php echo $label; ?>]" value="<?php echo $value; ?>">
104  </p>
105  <?php endforeach; ?>
```

system/admin/theme/posts/add.php, line 42 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.posts

system/admin/theme/posts/add.php, line 42 (Cross-Site Scripting: Reflected) — Critical

```
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/posts/add.php:42
Taint Flags: WEB, XSS

```
39
40  <p>
41  <label for="html">Content:</label>
42  <textarea id="html" name="html"><?php echo Input::post('html'); ?></textarea>
43
44  <em>Your post's main content. Enjoys a healthy dose of valid HTML.</em>
45  </p>
```

Package: system.admin.theme.users

system/admin/theme/users/amnesia.php, line 12 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/amnesia.php:12

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/amnesia.php, line 12 (Cross-Site Scripting: Reflected) — Critical

Taint Flags: WEB, XSS

```
9
10  <p>
11  <label for="email">Email:</label>
12  <input autocapitalize="off" name="email" id="email" value="<?php echo
Input::post('email'); ?>">
13  </p>
14
15  <p class="buttons">
```

system/admin/theme/users/edit.php, line 59 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/edit.php:59
Taint Flags: WEB, XSS

```
56
57  <p>
58  <label for="username">Username:</label>
59  <input id="username" name="username" value="<?php echo Input::post('username', $user-
>username); ?>">
60
61  <em>The desired username. Can be changed later.</em>
62  </p>
```

system/admin/theme/users/add.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/add.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/add.php:11
Taint Flags: WEB, XSS

```
8  <fieldset>
9  <p>
10  <label for="real_name">Real name:</label>
11  <input id="real_name" name="real_name" value="<?php echo Input::post('real_name'); ?>">
12
13  <em>The user&rsquo;s real name. Used in author bylines (visible to public).</em>
14  </p>
```

system/admin/theme/users/add.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/add.php, line 18 (Cross-Site Scripting: Reflected) — Critical

```
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/add.php:18
Taint Flags: WEB, XSS

```
15
16  <p>
17  <label for="bio">Biography:</label>
18  <textarea id="bio" name="bio"><?php echo Input::post('bio'); ?></textarea>
19
20  <em>A short biography for your user. Accepts valid HTML.</em>
21  </p>
```

system/admin/theme/users/edit.php, line 11 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/edit.php:11
Taint Flags: WEB, XSS

```
8  <fieldset>
9  <p>
10  <label for="real_name">Real name:</label>
11  <input id="real_name" name="real_name" value="<?php echo Input::post('real_name', $user->real_name); ?>">
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/edit.php, line 11 (Cross-Site Scripting: Reflected) — Critical

```
12
13  <em>The user&rsquo;s real name. Used in author bylines (visible to public).</em>
14  </p>
```

system/admin/theme/users/add.php, line 59 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/add.php:59
Taint Flags: WEB, XSS

```
56
57  <p>
58  <label for="username">Username:</label>
59  <input id="username" name="username" value="<?php echo Input::post('username'); ?>">
60
61  <em>The desired username. Can be changed later.</em>
62  </p>
```

system/admin/theme/users/add.php, line 73 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/add.php, line 73 (Cross-Site Scripting: Reflected) — Critical

File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/add.php:73
Taint Flags: WEB, XSS

```
70
71  <p>
72  <label for="email">Email:</label>
73  <input id="email" name="email" value="<?php echo Input::post('email'); ?>">
74
75  <em>The user&rsquo;s email address. Needed if the user forgets their password.</em>
76  </p>
```

system/admin/theme/users/edit.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/edit.php, line 18 (Cross-Site Scripting: Reflected) — Critical

File: system/admin/theme/users/edit.php:18
Taint Flags: WEB, XSS

```
15
16  <p>
17  <label for="bio">Biography:</label>
18  <textarea id="bio" name="bio"><?php echo Input::post('bio', $user->bio); ?></textarea>
19
20  <em>A short biography for your user. Accepts valid HTML.</em>
21  </p>
```

system/admin/theme/users/edit.php, line 73 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/edit.php:73
Taint Flags: WEB, XSS

```
70
71  <p>
72  <label for="email">Email:</label>
73  <input id="email" name="email" value="<?php echo Input::post('email', $user->email); ?>">
74
75  <em>The user&rsquo;s email address. Needed if the user forgets their password.</em>
76  </p>
```

system/admin/theme/users/login.php, line 12 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/login.php, line 12 (Cross-Site Scripting: Reflected) — Critical

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
36   }
37
38   public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/login.php:12
Taint Flags: WEB, XSS

```
9
10   <p>
11   <label for="user">Username:</label>
12   <input autocapitalize="off" name="user" id="user" value="<?php echo Input::post('user'); ?>">
13   </p>
14
15   <p>
```

system/admin/theme/users/reset.php, line 14 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32   }
33
34   public static function post($key, $default = false) {
35   return static::fetch_array($_POST, $key, $default);
```

Cross-Site Scripting: Reflected — Critical

Package: system.admin.theme.users

system/admin/theme/users/reset.php, line 14 (Cross-Site Scripting: Reflected) — Critical

```
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: builtin_echo()
File: system/admin/theme/users/reset.php:14
Taint Flags: WEB, XSS

```
11
12  <p>
13  <label for="password">Password:</label>
14  <input name="password" id="password" type="password" value="<?php echo
Input::post('password'); ?>">
15  </p>
16
17  <p class="buttons">
```

Package: themes.default

themes/default/posts.php, line 15 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/posts.php:15
Taint Flags: WEB, XSS

```
12  <?php endwhile; ?>
```

Cross-Site Scripting: Reflected — Critical

Package: themes.default

themes/default/posts.php, line 15 (Cross-Site Scripting: Reflected) — Critical

```
13  </ul>
14
15  <p><?php echo posts_prev(); ?> <?php echo posts_next(); ?></p>
16
17  <?php else: ?>
18  <p>Looks like you have some writing to do!</p>
```

themes/default/search.php, line 3 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/search.php:3
Taint Flags: WEB, XSS

```
1 <section class="content">
2
3  <h1>You searched for &ldquo;<?php echo search_term(); ?>&rdquo;.</h1>
4
5  <?php if(has_search_results()): ?>
6  <p>We found <?php echo total_search_results(); ?> <?php echo
pluralise(total_search_results(), 'result'); ?> for &ldquo;<?php echo search_term(); ?
>&rdquo;</p>
7  <ul class="items wrap">
```

themes/default/search.php, line 6 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Cross-Site Scripting: Reflected — Critical

Package: themes.default

themes/default/search.php, line 6 (Cross-Site Scripting: Reflected) — Critical

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/search.php:6
Taint Flags: WEB, XSS

```
3  <h1>You searched for &ldquo;<?php echo search_term(); ?>&rdquo;.</h1>
4
5  <?php if(has_search_results()): ?>
6  <p>We found <?php echo total_search_results(); ?> <?php echo
pluralise(total_search_results(), 'result'); ?> for &ldquo;<?php echo search_term(); ?
>&rdquo;</p>
7  <ul class="items wrap">
8  <?php while(search_results()): ?>
9  <li>
```

themes/default/search.php, line 21 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
```

Cross-Site Scripting: Reflected — Critical

Package: themes.default

themes/default/search.php, line 21 (Cross-Site Scripting: Reflected) — Critical

```
21   }
```

Sink Details

Sink: builtin_echo()
File: themes/default/search.php:21
Taint Flags: WEB, XSS

```
18  <p><?php echo search_prev(); ?> <?php echo search_next(); ?></p>
19
20  <?php else: ?>
21  <p>Unfortunately, there's no results for &ldquo;<?php echo search_term(); ?>&rdquo;. Did
you spell everything correctly?</p>
22  <?php endif; ?>
23
24 </section>
```

themes/default/search.php, line 18 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/search.php:18
Taint Flags: WEB, XSS

```
15  <?php endwhile; ?>
16  </ul>
17
18  <p><?php echo search_prev(); ?> <?php echo search_next(); ?></p>
19
```

Cross-Site Scripting: Reflected — Critical

Package: themes.default

themes/default/search.php, line 18 (Cross-Site Scripting: Reflected) — Critical

```
20  <?php else: ?>
21  <p>Unfortunately, there's no results for &ldquo;<?php echo search_term(); ?>&rdquo;. Did
you spell everything correctly?</p>
```

themes/default/404.php, line 5 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/404.php:5
Taint Flags: WEB, XSS

```
2 <section class="content">
3  <h1>Oh no, this page can&rsquo;t be found.</h1>
4
5  <p>Unfortunately, the page at <code><?php echo current_url(); ?></code>
6  can't be found, but don't give up hope yet! You can always try going back to
7  the homepage, or searching.</p>
8 </section>
```

Package: themes.default.includes

themes/default/includes/comment_form.php, line 21 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Cross-Site Scripting: Reflected — Critical

Package: themes.default.includes

themes/default/includes/comment_form.php, line 21 (Cross-Site Scripting: Reflected) — Critical

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Sink Details

Sink: builtin_echo()
File: themes/default/includes/comment_form.php:21
Taint Flags: WEB, XSS

```
18  </ul>
19  <?php endif; ?>
20
21  <form id="comment" class="commentform" method="post" action="<?php echo current_url(); ?
>#comment">
22  <legend>Add your comments</legend>
23
24  <?php echo comment_form_notifications(); ?>
```

themes/default/includes/header.php, line 35 (Cross-Site Scripting: Reflected) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_SERVER['REQUEST_URI']
From: request.uri
File: system/classes/request.php:18

```
15  // try request uri
16  elseif(isset($_SERVER['REQUEST_URI'])) {
17  // make sure we can parse URI
18  if(($uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)) === false) {
19  throw new Exception('Malformed request URI');
20  }
21  }
```

Cross-Site Scripting: Reflected — Critical

Package: themes.default.includes

themes/default/includes/header.php, line 35 (Cross-Site Scripting: Reflected) — Critical

Sink Details

Sink: builtin_echo()
File: themes/default/includes/header.php:35
Taint Flags: WEB, XSS

```
32  <body>
33
34  <form id="search" action="<?php echo search_url(); ?>" method="post">
35  <input type="search" name="term" placeholder="To search, type and hit enter&hellip;"
value="<?php echo search_term(); ?>">
36  </form>
37
38  <header id="top">
```

Key Management: Empty Encryption Key (1 issue)

Abstract

Empty encryption keys may compromise system security in a way that cannot be easily remedied.

Explanation

It is never a good idea to use an empty encryption key. Not only does using an empty encryption key significantly reduce the protection afforded by a good encryption algorithm, but it also makes fixing the problem extremely difficult. Once the offending code is in production, the empty encryption key cannot be changed without patching the software. If an account protected by the empty encryption key is compromised, the owners of the system will be forced to choose between security and availability.

Example: The code below initializes an encryption key variable to an empty string.

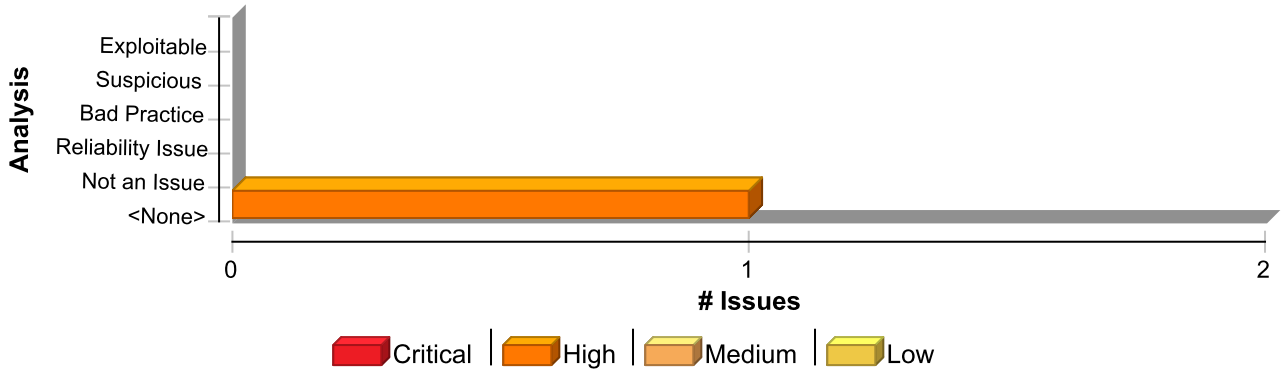
```
...
$encryption_key = '';

$filter = new Zend_Filter_Encrypt($encryption_key);

$filter->setVector('myIV');

$encrypted = $filter->filter('text_to_be_encrypted');
print $encrypted;
...
```

Not only will anyone who has access to the code be able to determine that it uses an empty encryption key, but anyone employing even basic cracking techniques is much more likely to successfully decrypt any encrypted data. Once the program has shipped, there is no way to change the empty encryption key unless the program is patched. An employee with access to this information could use it to break into the system. Even if attackers only had access to the application's executable, they could extract evidence of the use of an empty encryption key.

Recommendation

Encryption keys should never be empty and should generally be obfuscated and managed in an external source. Storing encryption keys in plaintext, empty or otherwise, anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key.
Starting with Microsoft(R) Windows(R) 2000, Microsoft(R) provides Windows Data Protection Application Programming Interface (DPAPI), which is an OS-level service that protects sensitive application data, such as passwords and private keys [1].

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Key Management: Empty Encryption Key	1	0	0	1
Total	**1**	**0**	**0**	**1**

Key Management: Empty Encryption Key — High

Package: <none>

config.default.php, line 28 (Key Management: Empty Encryption Key) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: ArrayAccess
File: config.default.php:28
Taint Flags:

```
25  'admin_folder' => 'admin',
26
27  // your unique application key used for signing passwords
28  'key' => ''
29  ),
30
31  // Session details
```

Password Management: Empty Password (1 issue)

Abstract

Empty passwords may compromise system security in a way that cannot be easily remedied.

Explanation

It is never a good idea to assign an empty string to a password variable. If the empty password is used to successfully authenticate against another system, then the corresponding account's security is likely compromised because it accepts an empty password. If the empty password is merely a placeholder until a legitimate value can be assigned to the variable, then it can confuse anyone unfamiliar with the code and potentially cause problems on unexpected control flow paths.

Example: The code below attempts to connect to a database with an empty password.

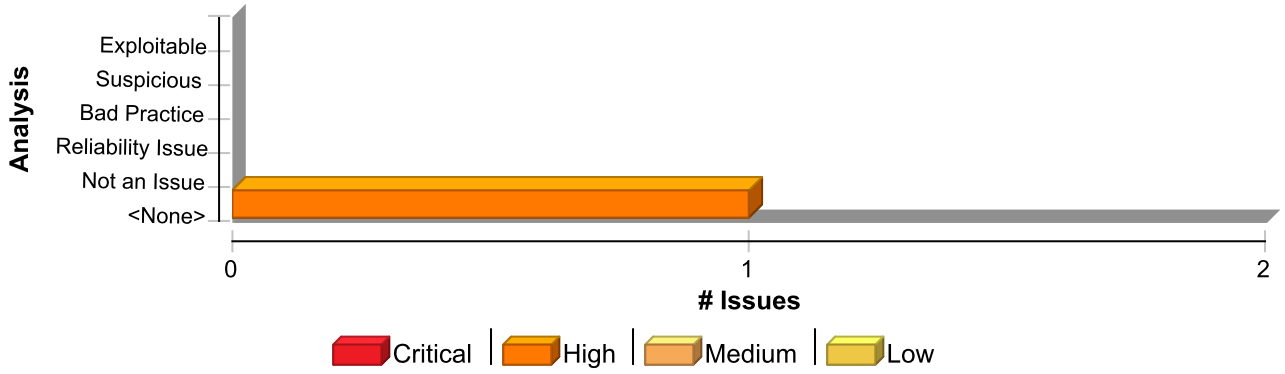
If the code in the Example succeeds, it indicates that the database user account "scott" is configured with an empty password, which can be easily guessed by an attacker. Even worse, once the program has shipped, updating the account to use a non-empty password will require a code change.

Recommendation

Always read stored password values from encrypted, external resources and assign password variables meaningful values. Ensure that sensitive resources are never protected with empty or null passwords.

Starting with Microsoft(R) Windows(R) 2000, Microsoft(R) provides Windows Data Protection Application Programming Interface (DPAPI), which is an OS-level service that protects sensitive application data, such as passwords and private keys [1].

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Password Management: Empty Password	1	0	0	1
Total	**1**	**0**	**0**	**1**

Password Management: Empty Password — High

Package: <none>

config.default.php, line 11 (Password Management: Empty Password) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: ArrayAccess
File: config.default.php:11
Taint Flags:

```
8  'database' => array(
9  'host' => '127.0.0.1',
10  'username' => 'root',
11  'password' => '',
12  'name' => 'testcms'
13  ),
14
```

Password Management: Password in HTML Form (1 issue)

Abstract

Populating password fields in an HTML form could result in a system compromise.

Explanation

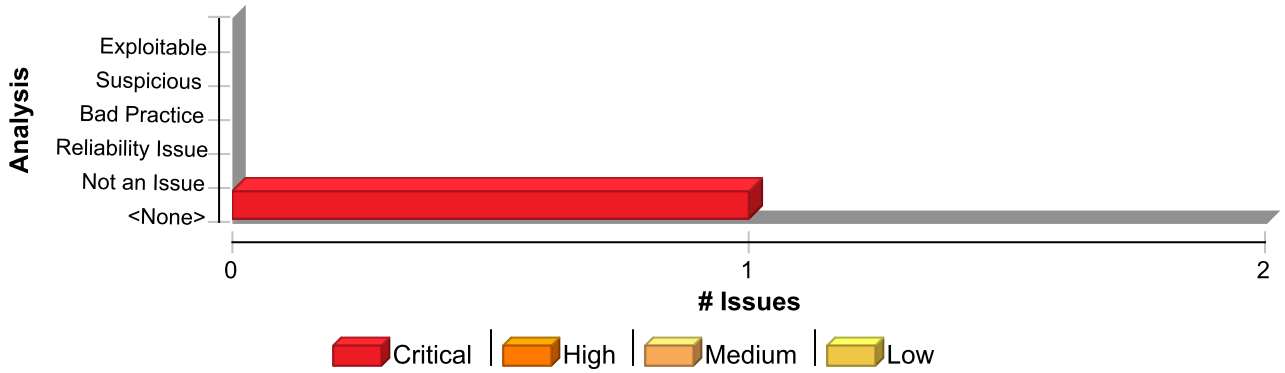
Populating password fields in an HTML form allows anyone to see their values in the HTML source. Furthermore, sensitive information stored in password fields may be cached by proxies or browsers.

Recommendation

Do not populate password-type form fields.

Example: In HTML forms, do not set the `value` attribute of sensitive inputs.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Password Management: Password in HTML Form	1	0	0	1
Total	**1**	**0**	**0**	**1**

Password Management: Password in HTML Form	**Critical**
Package: system.admin.theme.users	
system/admin/theme/users/reset.php, line 14 (Password Management: Password in HTML Form)	**Critical**

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Content)

Sink Details

Password Management: Password in HTML Form	Critical
Package: system.admin.theme.users	
system/admin/theme/users/reset.php, line 14 (Password Management: Password in HTML Form)	**Critical**

File: system/admin/theme/users/reset.php:14
Taint Flags:

```
11
12  <p>
13  <label for="password">Password:</label>
14  <input name="password" id="password" type="password" value="<?php echo
Input::post('password'); ?>">
15  </p>
16
17  <p class="buttons">
```

Privacy Violation (1 issue)

Abstract

Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

Explanation

Privacy violations occur when:

1. Private user information enters the program.

2. The data is written to an external location, such as the console, file system, or network.

Example: The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored is the return value from the `getPassword()` function that returns user-supplied plaintext password associated with the account.

The code in the example above logs a plaintext password to the application eventlog. Although many developers trust the eventlog as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information

- Accessed from a database or other data store by the application

- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

Despite these regulations, privacy violations continue to occur with alarming frequency.

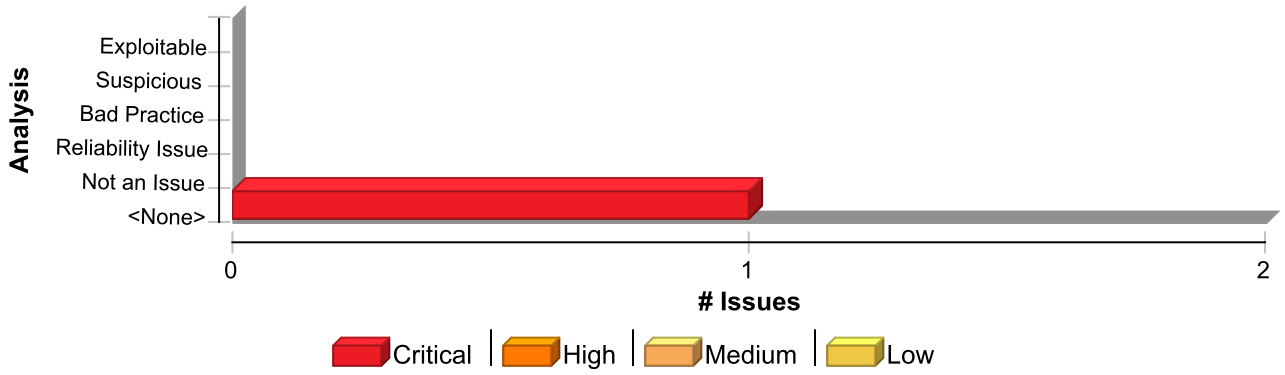
Recommendation

When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.

To enforce good privacy management, develop and strictly adhere to internal privacy guidelines. The guidelines should specifically describe how an application should handle private data. If your organization is regulated by federal or state law, ensure that your privacy guidelines are sufficiently strenuous to meet the legal requirements. Even if your organization is not regulated, you must protect private information or risk losing customer confidence.

The best policy with respect to private data is to minimize its exposure. Applications, processes, and employees should not be granted access to any private data unless the access is required for the tasks that they are to perform. Just as the principle of least privilege dictates that no operation should be performed with more than the necessary privileges, access to private data should be restricted to the smallest possible group.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Privacy Violation	1	0	0	1
Total	**1**	**0**	**0**	**1**

Privacy Violation — Critical

Package: install

install/installer.php, line 159 (Privacy Violation) — Critical

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $password
File: install/installer.php:150

```
147 if(empty($errors)) {
148   //no errors we're all gooood
149   $response['installed'] = true;
150   $response['password'] = $password;
151   $response['warnings'] = $warnings;
152 } else {
153   $response['installed'] = false;
```

Sink Details

Sink: builtin_echo()
File: install/installer.php:159
Taint Flags: PRIVATE

```
156 }
157
158 // output json formatted string
159 echo json_encode($response);
160
161 undefined
162 undefined
```

Privacy Violation: Autocomplete (2 issues)

Abstract

Autocompletion of forms allows some browsers to retain sensitive information in their history.

Explanation

With autocompletion enabled, some browsers retain user input across sessions, which could allow someone using the computer after the initial user to see information previously submitted.

Recommendation

Explicitly disable autocompletion on forms or sensitive inputs. By disabling autocompletion, information previously entered will not be presented back to the user as they type. It will also disable the "remember my password" functionality of most major browsers.

Example 1: In an HTML form, disable autocompletion for all input fields by explicitly setting the value of the `autocomplete` attribute to `off` on the `form` tag.

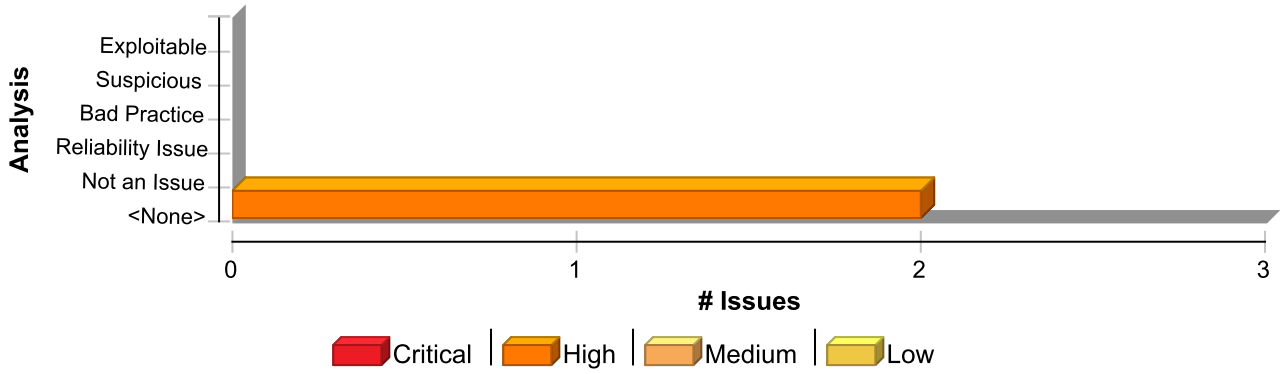
Address:
Password:

Example 2: Alternatively, disable autocompletion for specific input fields by explicitly setting the value of the `autocomplete` attribute to `off` on the corresponding tags.

Address:
Password:

Note that the default value of the `autocomplete` attributed is `on`. Therefore do not omit the attribute when dealing with sensitive inputs.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Privacy Violation: Autocomplete	2	0	0	2
Total	**2**	**0**	**0**	**2**

Privacy Violation: Autocomplete — High

Package: system.admin.theme.users

system/admin/theme/users/login.php, line 17 (Privacy Violation: Autocomplete) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Content)

Sink Details

File: system/admin/theme/users/login.php:17
Taint Flags:

```
14
15  <p>
16  <label for="pass">Password:</label>
17  <input type="password" name="pass" id="pass">
18
19  <em><a href="<?php echo admin_url('users/amnesia'); ?>">Forgotten your password?</a></em>
20  </p>
```

system/admin/theme/users/reset.php, line 14 (Privacy Violation: Autocomplete) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Content)

Sink Details

File: system/admin/theme/users/reset.php:14
Taint Flags:

```
11
12  <p>
13  <label for="password">Password:</label>
```

Privacy Violation: Autocomplete	High

Package: system.admin.theme.users

system/admin/theme/users/reset.php, line 14 (Privacy Violation: Autocomplete)	High

```
14  <input name="password" id="password" type="password" value="<?php echo
Input::post('password'); ?>">
15  </p>
16
17  <p class="buttons">
```

SQL Injection (6 issues)

Abstract

Constructing a dynamic SQL statement with input coming from an untrusted source might allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Explanation

SQL injection errors occur when:

1. Data enters a program from an untrusted source.

2. The data is used to dynamically construct a SQL query.

Example 1: The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user.

```
    ...
    $userName = $_SESSION['userName'];
 $itemName = $_POST['itemName'];
    $query = "SELECT * FROM items WHERE owner = '$userName' AND itemname =
'$itemName';";
  $result = mysql_query($query);
 ...
```

The query that this code intends to execute follows:

```
    SELECT * FROM items
    WHERE owner =
   AND itemname = ;
```

However, because the query is constructed dynamically by concatenating a constant query string and a user input string, the query only behaves correctly if `itemName` does not contain a single-quote character. If an attacker with the user name `wiley` enters the string "`name' OR 'a'='a`" for `itemName`, then the query becomes the following:

```
  SELECT * FROM items
    WHERE owner = 'wiley'
  AND itemname = 'name' OR 'a'='a';
```

The addition of the `OR 'a'='a'` condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
    SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the `items` table, regardless of their specified owner.

Example 2: This example examines the effects of a different malicious value passed to the query constructed and executed in Example 1. If an attacker with the user name `wiley` enters the string "`name'; DELETE FROM items; --`" for `itemName`, then the query becomes the following two queries:

```
SELECT * FROM items
   WHERE owner = 'wiley'
 AND itemname = 'name';

   DELETE FROM items;

   --'
```

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed [4]. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in Example 1. If an attacker enters the string "`name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a`", the following three valid statements will be created:

```
SELECT * FROM items
  WHERE owner = 'wiley'
AND itemname = 'name';

  DELETE FROM items;

  SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers may:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Recommendation

The root cause of a SQL injection vulnerability is the ability of an attacker to change context in the SQL query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead. When a SQL query is constructed, the programmer knows what should be interpreted as part of the command and what should be interpreted as data. Parameterized SQL statements can enforce this behavior by disallowing data-directed context changes and preventing nearly all SQL injection attacks. Parameterized SQL statements are constructed using strings of regular SQL, but where user-supplied data needs to be included, they include bind parameters, which are placeholders for data that is subsequently inserted. In other words, bind parameters allow the programmer to explicitly specify to the database what should be treated as a command and what should be treated as data. When the program is ready to execute a statement, it specifies to the database the runtime values to use for each of the bind parameters without the risk that the data will be interpreted as a modification to the command.

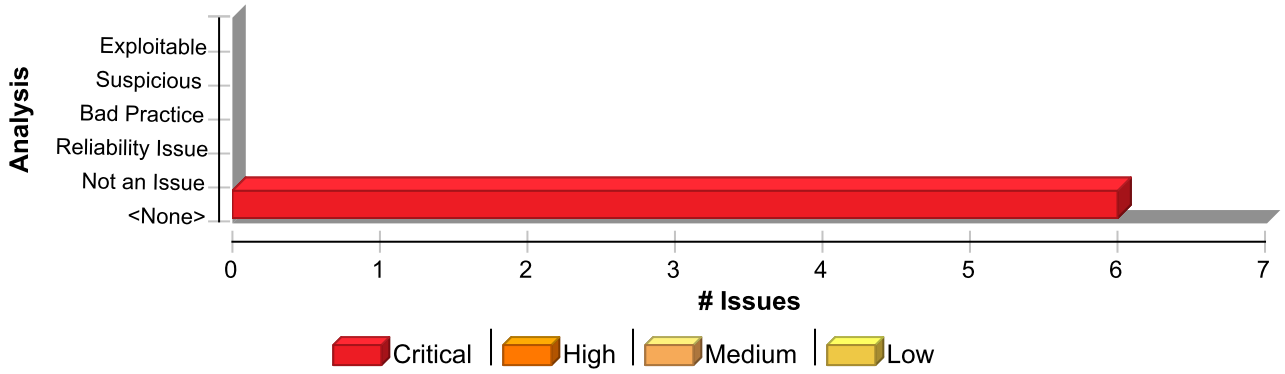
When connecting to MySQL, the previous example can be rewritten to use parameterized SQL statements (instead of concatenating user supplied strings) as follows:

```
...
   $mysqli = new mysqli($host,$dbuser, $dbpass, $db);
$userName = $_SESSION['userName'];
$itemName = $_POST['itemName'];
   $query = "SELECT * FROM items WHERE owner = ? AND itemname = ?";
  $stmt = $mysqli->prepare($query);
 $stmt->bind_param('ss',$username,$itemName);
  $stmt->execute();
 ...
```

The MySQL Improved extension (mysqli) is available for PHP5 users of MySQL. Code that relies on a different database should check for similar extensions.

More complicated scenarios, often found in report generation code, require that user input affect the structure of the SQL statement, for instance by adding a dynamic constraint in the `WHERE` clause. Do not use this requirement to justify concatenating user input to create a query string. Prevent SQL injection attacks where user input must affect command structure with a level of indirection: create a set of legitimate strings that correspond to different elements you might include in a SQL statement. When constructing a statement, use input from the user to select from this set of application-controlled values.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
SQL Injection	6	0	0	6
Total	**6**	**0**	**0**	**6**

SQL Injection — Critical

Package: install

install/installer.php, line 127 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST['?']
File: install/installer.php:27

```
24 $errors = array();
25
26 foreach($fields as $field) {
27  $post[$field] = isset($_POST[$field]) ? $_POST[$field] : false;
28 }
29
30 if(empty($post['db'])) {
```

Sink Details

Sink: PDO.exec()
File: install/installer.php:127
Taint Flags: WEB, XSS

```
124
125  try {
126  $dbh->beginTransaction();
127  $dbh->exec($sql);
128
129  $sql= "INSERT INTO `meta` (`key`, `value`) VALUES ('sitename', ?), ('description', ?),
('theme', ?);";
```

SQL Injection — Critical

Package: install

install/installer.php, line 127 (SQL Injection) — Critical

```
130  $statement = $dbh->prepare($sql);
```

Package: system.classes

system/classes/db.php, line 70 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

Sink: PDO.prepare()
Enclosing Method: query()
File: system/classes/db.php:70
Taint Flags: WEB, XSS

```
67  }
68
69  // prepare
70  $sth = static::$dbh->prepare($sql);
71
72  // bind params
73  $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

system/classes/db.php, line 70 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_GET

SQL Injection — Critical

Package: system.classes

system/classes/db.php, line 70 (SQL Injection) — Critical

From: input.get
File: system/classes/input.php:39

```
36  }
37
38  public static function get($key, $default = false) {
39  return static::fetch_array($_GET, $key, $default);
40  }
41
42  public static function put($key, $default = false) {
```

Sink Details

Sink: PDO.prepare()
Enclosing Method: query()
File: system/classes/db.php:70
Taint Flags: WEB, XSS

```
67  }
68
69  // prepare
70  $sth = static::$dbh->prepare($sql);
71
72  // bind params
73  $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

system/classes/db.php, line 116 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read $_POST
From: input.post
File: system/classes/input.php:35

```
32  }
33
34  public static function post($key, $default = false) {
35  return static::fetch_array($_POST, $key, $default);
36  }
37
38  public static function get($key, $default = false) {
```

Sink Details

SQL Injection — Critical

Package: system.classes

system/classes/db.php, line 116 (SQL Injection) — Critical

Sink: PDO.prepare()
Enclosing Method: exec()
File: system/classes/db.php:116
Taint Flags: WEB, XSS

```
113  }
114
115  // prepare
116  $sth = static::$dbh->prepare($sql);
117
118  // bind params
119  $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

system/classes/db.php, line 70 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: PDOStatement.fetchcolumn()
File: upgrade/migrations.php:43

```
40
41 // make posts_page the home page
42 if(Schema::has('meta', 'key', 'home_page') === false) {
43  $posts_page = Db::query("select `value` from meta where `key` =
'show_posts'")->fetchColumn();
44
45  $sql = "insert into `meta` (`key`, `value`) values ('home_page', '" .
$posts_page . "')";
46  $migration->query($sql);
```

Sink Details

Sink: PDO.prepare()
Enclosing Method: query()
File: system/classes/db.php:70
Taint Flags: DATABASE, XSS

```
67  }
68
69  // prepare
70  $sth = static::$dbh->prepare($sql);
71
72  // bind params
73  $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

SQL Injection — Critical

Package: system.classes

system/classes/db.php, line 70 (SQL Injection) — Critical

system/classes/db.php, line 70 (SQL Injection) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: PDO.prepare()
From: db.query
File: system/classes/db.php:70

```
67   }
68
69   // prepare
70   $sth = static::$dbh->prepare($sql);
71
72   // bind params
73   $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

Sink Details

Sink: PDO.prepare()
Enclosing Method: query()
File: system/classes/db.php:70
Taint Flags: DATABASE, XSS

```
67  }
68
69  // prepare
70  $sth = static::$dbh->prepare($sql);
71
72  // bind params
73  $reflector = new ReflectionMethod('PDOStatement', 'bindValue');
```

Weak Encryption (5 issues)

Abstract

The identified call uses a weak encryption algorithm that cannot guarantee the confidentiality of sensitive data.

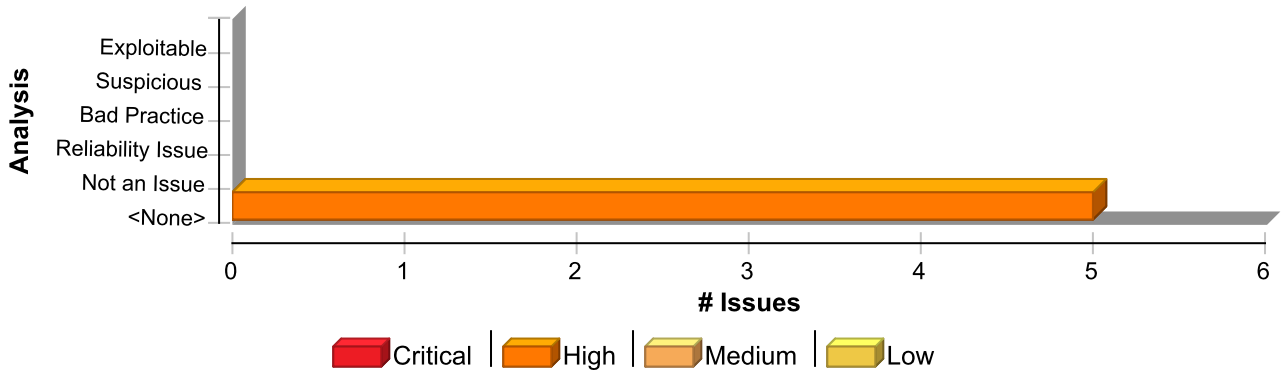
Explanation

Antiquated encryption algorithms such as DES no longer provide sufficient protection for use with sensitive data. Encryption algorithms rely on key size as one of the primary mechanisms to ensure cryptographic strength. Cryptographic strength is often measured by the time and computational power needed to generate a valid key. Advances in computing power have made it possible to obtain small encryption keys in a reasonable amount of time. For example, the 56-bit key used in DES posed a significant computational hurdle in the 1970's when the algorithm was first developed, but today DES can be cracked in less than a day using commonly available equipment.

Recommendation

Use strong encryption algorithms with large key sizes to protect sensitive data. Examples of strong alternatives to DES are Rijndael (Advanced Encryption Standard or AES) and Triple DES (3DES). Before selecting an algorithm, first determine if your organization has standardized on a specific algorithm and implementation.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Weak Encryption	5	0	0	5
Total	**5**	**0**	**0**	**5**

Weak Encryption	**High**
Package: install	
install/installer.php, line 122 (Weak Encryption)	**High**

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Weak Encryption — High

Package: install

install/installer.php, line 122 (Weak Encryption) — High

Sink Details

Sink: crypt()
File: install/installer.php:122
Taint Flags:

```
119  $password = random(8);
120
121  $sql = str_replace('[[now]]', time(), file_get_contents('test.sql'));
122  $sql = str_replace('[[password]]', crypt($password), $sql);
123  $sql = str_replace('[[email]]', strtolower(trim($post['email'])), $sql);
124
125  try {
```

Package: system.classes

system/classes/users.php, line 142 (Weak Encryption) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: crypt()
Enclosing Method: reset_password()
File: system/classes/users.php:142
Taint Flags:

```
139  return false;
140  }
141
142  $password = crypt($post['password']);
143
144  $sql = "update users set `password` = ? where id = ?";
145  Db::query($sql, array($password, $id));
```

system/classes/users.php, line 73 (Weak Encryption) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: crypt()
Enclosing Method: login()
File: system/classes/users.php:73
Taint Flags:

```
70  // find user
```

Weak Encryption — High

Package: system.classes

system/classes/users.php, line 73 (Weak Encryption) — High

```
71  if($user = Users::find(array('username' => $post['user']))) {
72  // check password
73  if(crypt($post['pass'], $user->password) != $user->password) {
74  $errors[] = 'Incorrect details';
75  }
76  } else {
```

system/classes/users.php, line 190 (Weak Encryption) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: crypt()
Enclosing Method: update()
File: system/classes/users.php:190
Taint Flags:

```
187
188  if(strlen($post['password'])) {
189  // encrypt new password
190  $post['password'] = crypt($post['password']);
191  } else {
192  // remove it and leave it unchanged
193  unset($post['password']);
```

system/classes/users.php, line 247 (Weak Encryption) — High

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: crypt()
Enclosing Method: add()
File: system/classes/users.php:247
Taint Flags:

```
244  }
245
246  // encrypt password
247  $post['password'] = crypt($post['password']);
248
249  // format email
250  $post['email'] = strtolower(trim($post['email']));
```