Fortify Audit Workbench

Developer Workbook

SE-0019_20260115

Table of Contents

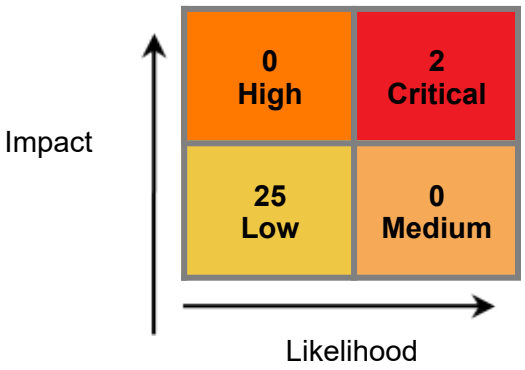
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the SE-0019_20260115 project audit. The information contained in this workbook is targeted at project managers and developers.

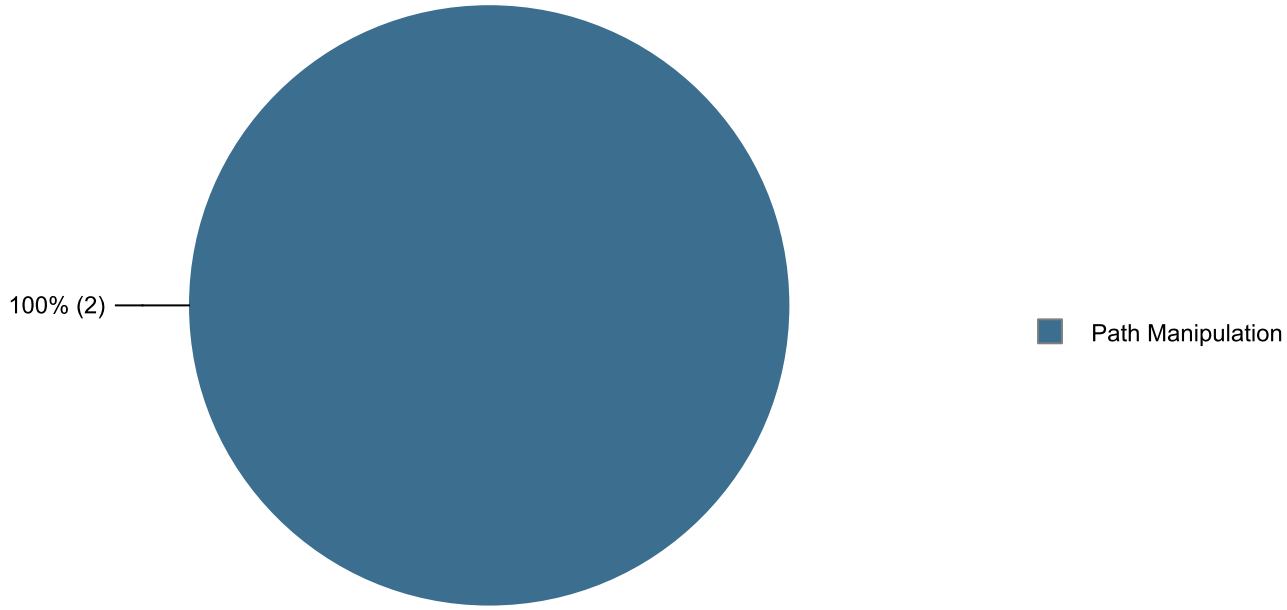
This section provides an overview of the issues uncovered during analysis.

Project Name:	SE-0019_20260115
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories



Project Description

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

SCA

Date of Last Analysis:	Jan 15, 2026 10:12 AM	**Engine Version:**	25.4.0.0135
Host Name:	Win2019-sca-new	**Certification:**	VALID
Number of Files:	47	**Lines of Code:**	597,189

Rulepack Name	Rulepack Version
Fortify Secure Coding Rules, Community, Cloud	2025.4.0.0009
Fortify Secure Coding Rules, Community, PHP	2025.4.0.0009
Fortify Secure Coding Rules, Community, Universal	2025.4.0.0009
Fortify Secure Coding Rules, Core, Cloud	2025.4.0.0009
Fortify Secure Coding Rules, Core, JavaScript	2025.4.0.0009
Fortify Secure Coding Rules, Core, PHP	2025.4.0.0009
Fortify Secure Coding Rules, Core, Universal	2025.4.0.0009
Fortify Secure Coding Rules, Extended, Configuration	2025.4.0.0009
Fortify Secure Coding Rules, Extended, Content	2025.4.0.0009
Fortify Secure Coding Rules, Extended, JavaScript	2025.4.0.0009

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	
Password Management: Password in Comment	0	0	0	0 / 21	0 / 21
Path Manipulation	0 / 2	0	0	0	0 / 2
Weak Cryptographic Hash	0	0	0	0 / 4	0 / 4

Results Outline

Password Management: Password in Comment (21 issues)

Abstract

Storing passwords or password details in plain text anywhere in the system or system code might compromise system security in a way that is not easy to remedy.

Explanation

It is never a good idea to hardcode a password. Storing password details within comments is equivalent to hardcoding passwords. Not only is the password visible to the project's developers, it also makes fixing the problem extremely difficult. After the code is in production, the password is now leaked to the outside world and cannot be protected or changed without patching the software. If the account protected by the password is compromised, the owners of the system must choose between security and availability.
Example 1: The following comment specifies the default password to connect to a database:

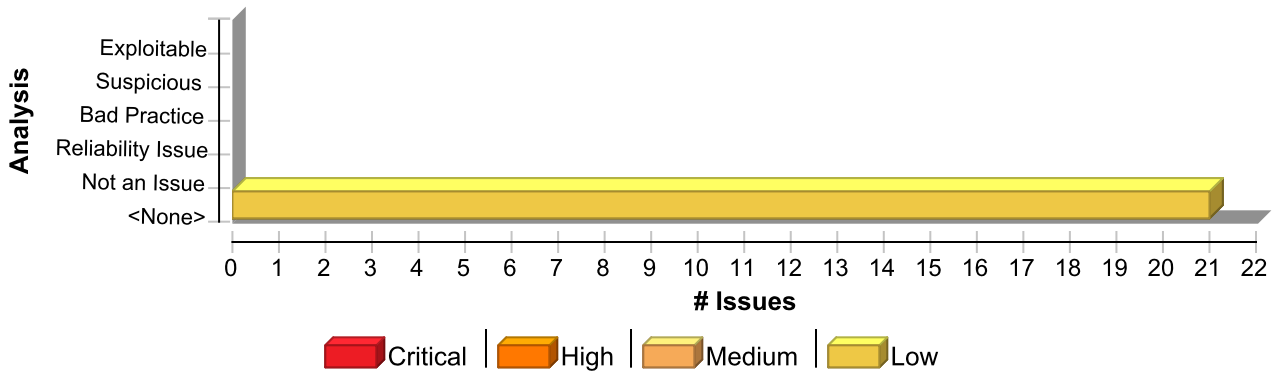
```
...
// Default username for database connection is "scott"
// Default password for database connection is "tiger"
...
```

This code will run successfully, but anyone who has access to it will have access to the password. An employee with access to this information can use it to break into the system.

Recommendation

Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Password Management: Password in Comment	21	0	0	21
Total	**21**	**0**	**0**	**21**

Password Management: Password in Comment — Low

Package: blog-url

blog-url/blog-url.php, line 1147 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:1147
Taint Flags:

```
1144  $this->url = sprintf(self::URL_TEMPLATE, $this->ip);
1145  }
1146
1147  /**
1148  * Reloads a peer server.
1149  *
1150  * @param string $password the password of this server
```

blog-url/blog-url.php, line 37 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:37
Taint Flags:

```
34  */
35  const SUDO_EXEC = "/usr/local/bin/wp-blog-url-sudo.php";
36
37  /**
38  * The password to run the script command
39  * @var string
40  */
```

blog-url/blog-url.php, line 1132 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:1132
Taint Flags:

Password Management: Password in Comment — Low

Package: blog-url

blog-url/blog-url.php, line 1132 (Password Management: Password in Comment)

```
1129  */
1130  private static array $PEERS;
1131
1132  /**
1133  * Constructs a new peer server.
1134  *
1135  * @param int $id the peer ID
```

blog-url/blog-url-script.php, line 20 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url-script.php:20
Taint Flags:

```
17 define("PATH", "/bin:/usr/bin");
18 define("IS_CERTBOT_DRY_RUN", false);
19
20 // The md5() hash of the password that must exist in the environment
21 // variable WP_BLOG_URL_PASSWORD, to verify that this script is
22 // run by no one else but the corresponding set-uid program.
23 define("MY_TOKEN_DIGEST", "358fac2e8e18a7fc29c0f7b0dca9f1f1");
```

blog-url/blog-url.php, line 1086 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:1086
Taint Flags:

```
1083  }
1084 }
1085
1086 /**
1087  * An RPC peer server
1088  *
1089  * @property int $id
```

Password Management: Password in Comment — Low

Package: blog-url

blog-url/blog-url.php, line 1246 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:1246
Taint Flags:

```
1243
1244 BlogUrlSettings::init();
1245
1246 /**
1247  * custom url website google apps login
1248  *
1249  * @param string $options client id and client secret
```

blog-url/blog-url.php, line 1114 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/blog-url.php:1114
Taint Flags:

```
1111  */
1112  private string $ip;
1113
1114  /**
1115  * The digest hash of the peer password
1116  * @var string
1117  */
```

Package: blog-url.rpc

blog-url/rpc/wp-blog-url-rpc.php, line 31 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment

Password Management: Password in Comment — Low

Package: blog-url.rpc

blog-url/rpc/wp-blog-url-rpc.php, line 31 (Password Management: Password in Comment)

File: blog-url/rpc/wp-blog-url-rpc.php:31
Taint Flags:

```
28  */
29  const SUDO_COMMAND = "/usr/local/bin/wp-blog-url-sudo.php";
30
31  /**
32  * The password to run the script command
33  * @var string
34  */
```

blog-url/rpc/wp-blog-url-rpc.php, line 153 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/rpc/wp-blog-url-rpc.php:153
Taint Flags:

```
150  */
151  private static array $PEERS;
152
153  /**
154  * Constructs a new peer server.
155  *
156  * @param string $ip the peer IP
```

blog-url/rpc/wp-blog-url-rpc.php, line 127 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/rpc/wp-blog-url-rpc.php:127
Taint Flags:

```
124  }
125 }
126
127 /**
```

Password Management: Password in Comment — Low

Package: blog-url.rpc

blog-url/rpc/wp-blog-url-rpc.php, line 127 (Password Management: Password in Comment)

```
128  * An RPC peer server
129  *
130  * @property string $ip
```

blog-url/rpc/wp-blog-url-rpc.php, line 141 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: blog-url/rpc/wp-blog-url-rpc.php:141
Taint Flags:

```
138  */
139  private string $ip;
140
141  /**
142  * The digest hash of the peer password
143  * @var string
144  */
```

Package: ilc-custom.includes

ilc-custom/includes/ilc-xmlrpc.php, line 18 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: ilc-custom/includes/ilc-xmlrpc.php:18
Taint Flags:

```
15  * Specs on http://plant.blogger.com/api and https://groups.yahoo.com/group/bloggerDev/
16  */
17
18  /**
19  * Retrieve blogs that user owns.
20  *
21  * Will make more sense once we support multiple blogs.
```

Password Management: Password in Comment — Low

Package: steps-disable-user-login.includes

steps-disable-user-login/includes/steps-disable-user-login.php, line 185 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-disable-user-login/includes/steps-disable-user-login.php:185
Taint Flags:

```
182  return true;
183  }
184
185  /**
186  * After login check to see if user account is disabled
187  *
188  * @param object $user
```

Package: steps-logger

steps-logger/steps-logger.php, line 376 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-logger/steps-logger.php:376
Taint Flags:

```
373  Logger::save('wpmu_delete_user',sprintf('使用者%s(%s)刪除',$user->user_email,$user-
>user_login));
374  }
375
376  /**
377  * 登入成功、登入失敗、登入錯誤訊息調整
378  *
379  * @param WP_User|WP_Error|null $user WP_User or WP_Error object if a previous
```

steps-logger/steps-logger.php, line 219 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Password Management: Password in Comment — Low

Package: steps-logger

steps-logger/steps-logger.php, line 219 (Password Management: Password in Comment)

Sink Details

Sink: Comment
File: steps-logger/steps-logger.php:219
Taint Flags:

```
216  foreach ($user_data as $key => $value) {
217  if ($old_user_data[$key]!=$value) {
218  if ($key=='user_pass') {
219  /*
220  $_POST['pass1']-後台編輯頁面修改密碼
221  $_POST['password']-User REST API 修改密碼
222  */
```

steps-logger/steps-logger.php, line 253 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-logger/steps-logger.php:253
Taint Flags:

```
250  }
251  }
252
253  /**
254  * Fires after the user's password is reset.
255  *
256  * @since 4.4.0
```

Package: steps-password-policy

steps-password-policy/steps-password-policy.php, line 217 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-password-policy/steps-password-policy.php:217
Taint Flags:

Password Management: Password in Comment — Low

Package: steps-password-policy

steps-password-policy/steps-password-policy.php, line 217 (Password Management: Password in Comment)

```
214  $pawd = !empty($_POST['pass1'])?$_POST['pass1']:$_POST['password'];
215  if ($pawd) {
216
217  // Get last password change timestamp
218  $last_password_change = (int) get_user_meta( $user->ID, 'steps_last_password_change_at', true );
219
220  // If last password change timestamp exists and is within 24 hours
```

steps-password-policy/steps-password-policy.php, line 133 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-password-policy/steps-password-policy.php:133
Taint Flags:

```
130  return true;
131  }
132
133  /**
134  * Fires before the password reset procedure is validated.
135  * 前台重設密碼的驗證
136  * @since 3.5.0
```

steps-password-policy/steps-password-policy.php, line 94 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-password-policy/steps-password-policy.php:94
Taint Flags:

```
91  return self::$steps_mima_policy;
92  }
93
94  /**
95  * Filters the text describing the site's password complexity policy.
```

Password Management: Password in Comment — Low

Package: steps-password-policy

steps-password-policy/steps-password-policy.php, line 94 (Password Management: Password in Comment)

```
96  *
97  * @since 4.1.0
```

steps-password-policy/steps-password-policy.php, line 220 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-password-policy/steps-password-policy.php:220
Taint Flags:

```
217  // Get last password change timestamp
218  $last_password_change = (int) get_user_meta( $user->ID, 'steps_last_password_change_at', true );
219
220  // If last password change timestamp exists and is within 24 hours
221  if ( $last_password_change && ( time() - $last_password_change ) < 24 * 60 * 60 ) {
222  $errors->add( 'password_change_too_soon', '錯誤：24 小時內您無法再次更改密碼。' );
223  return;
```

steps-password-policy/steps-password-policy.php, line 170 (Password Management: Password in Comment)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: Comment
File: steps-password-policy/steps-password-policy.php:170
Taint Flags:

```
167  }
168  }
169
170  /**
171  * Fires after the user's password is reset.
172  * 前台重設密碼後的動作
173  * @since 4.4.0
```

Path Manipulation (2 issues)

Abstract

Allowing user input to control paths used in file system operations could enable an attacker to access or modify otherwise protected system resources.

Explanation

Path manipulation errors occur when the following two conditions are met: 1. An attacker can specify a path used in an operation on the file system. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program might give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker. **Example 1:** The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "`../../tomcat/conf/server.xml`", which causes the application to delete one of its own configuration files.

```
$rName = $_GET['reportName'];
$rFile = fopen("/usr/local/apfr/reports/" . rName,"a+");
...
unlink($rFile);
```

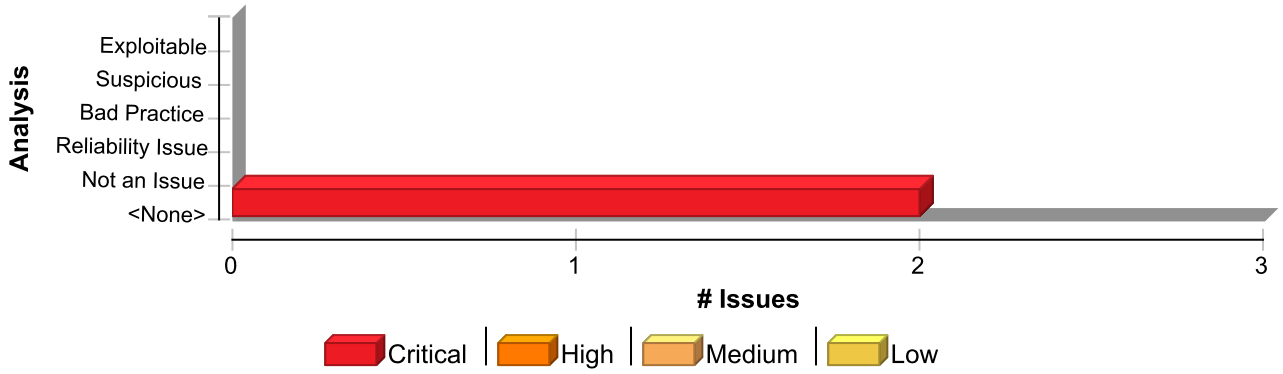
Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension `.txt`.

```
...
$filename = $CONFIG_TXT['sub'] . ".txt";
$handle = fopen($filename,"r");
$amt = fread($handle, filesize($filename));
echo $amt;
...
```

Recommendation

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate values from which the user must select. With this approach, the user-provided input is never used directly to specify the resource name. In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to maintain. Programmers often resort to implementing a deny list in these situations. A deny list is used to selectively reject or escape potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a list of characters that are permitted to appear in the resource name and accept input composed exclusively of characters in the approved set.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Path Manipulation	2	0	0	2
Total	**2**	**0**	**0**	**2**

Path Manipulation — Critical

Package: ilc-barrier-free

ilc-barrier-free/css-proxy.php, line 241 (Path Manipulation)

Issue Details

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

Source Details

Source: Read $_GET['css_url']
File: ilc-barrier-free/css-proxy.php:22

```
19  die('/* CSS URL parameter is required. */');
20 }
21
22 $css_url = urldecode($_GET['css_url']);
23
24 // 驗證URL（只允許WordPress 內部CSS）
25 $parsed_url = parse_url($css_url);
```

Sink Details

Sink: file_get_contents()
File: ilc-barrier-free/css-proxy.php:241
Taint Flags: NOVALIDATION, WEB, XSS

```
238  'user_agent' => 'WordPress CSS Proxy'
239  )
240  ));
241  $css_content = @file_get_contents($css_url, false, $context);
242  }
243 }
244
```

Path Manipulation — Critical

Package: ilc-barrier-free

ilc-barrier-free/css-proxy.php, line 241 (Path Manipulation)

ilc-barrier-free/css-proxy.php, line 241 (Path Manipulation)

Issue Details

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

Source Details

Source: Read $_SERVER['HTTP_HOST']
File: ilc-barrier-free/css-proxy.php:177

```
174  } else {
175  // 構建基本 URL
176  $protocol = (isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] === 'on') ? 'https' : 'http';
177  $host = isset($_SERVER['HTTP_HOST']) ? $_SERVER['HTTP_HOST'] : '';
178  $base = $protocol . '://' . $host;
179  $css_url = $base . $css_url;
180  }
```

Sink Details

Sink: file_get_contents()
File: ilc-barrier-free/css-proxy.php:241
Taint Flags: WEB, XSS

```
238  'user_agent' => 'WordPress CSS Proxy'
239  )
240  ));
241  $css_content = @file_get_contents($css_url, false, $context);
242  }
243 }
244
```

Weak Cryptographic Hash (4 issues)

Abstract

Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

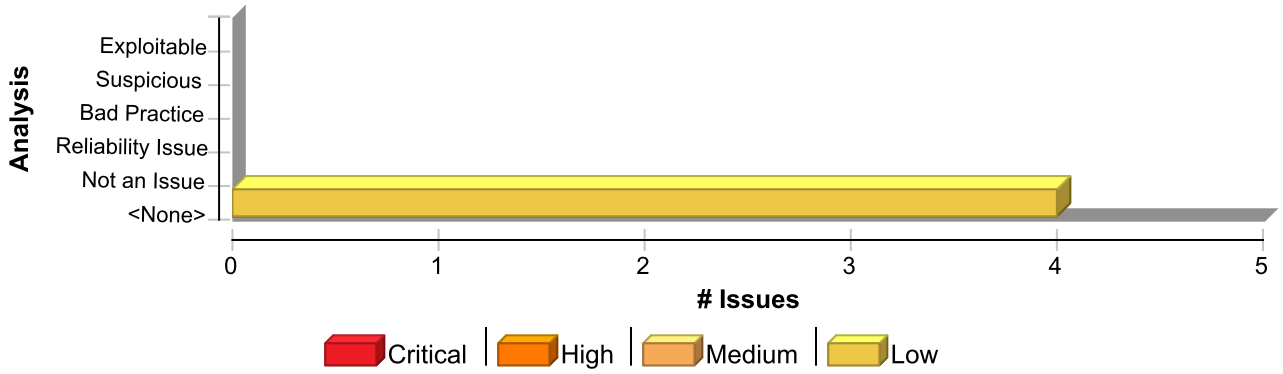
Explanation

MD2, MD4, MD5, RIPEMD-160, and SHA-1 are popular cryptographic hash algorithms often used to verify the integrity of messages and other data. However, as recent cryptanalysis research has revealed fundamental weaknesses in these algorithms, they should no longer be used within security-critical contexts. Effective techniques for breaking MD and RIPEMD hashes are widely available, so those algorithms should not be relied upon for security. In the case of SHA-1, current techniques still require a significant amount of computational power and are more difficult to implement. However, attackers have found the Achilles' heel for the algorithm, and techniques for breaking it will likely lead to the discovery of even faster attacks.

Recommendation

Discontinue the use of MD2, MD4, MD5, RIPEMD-160, and SHA-1 for data-verification in security-critical contexts. Currently, SHA-224, SHA-256, SHA-384, SHA-512, and SHA-3 are good alternatives. However, these variants of the Secure Hash Algorithm have not been scrutinized as closely as SHA-1, so be mindful of future research that might impact the security of these algorithms.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Weak Cryptographic Hash	4	0	0	4
Total	**4**	**0**	**0**	**4**

Weak Cryptographic Hash — Low

Package: blog-url

blog-url/blog-url-script.php, line 172 (Weak Cryptographic Hash)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Weak Cryptographic Hash — Low

Package: blog-url

blog-url/blog-url-script.php, line 172 (Weak Cryptographic Hash)

Sink Details

Sink: md5()
Enclosing Method: parse_args_real()
File: blog-url/blog-url-script.php:172
Taint Flags:

```
169  if (!array_key_exists("WP_BLOG_URL_PASSWORD", $_SERVER)) {
170  throw new ArgumentException("missing run key");
171  }
172  if (md5($_SERVER["WP_BLOG_URL_PASSWORD"]) != MY_TOKEN_DIGEST) {
173  throw new ArgumentException("incorrect run key");
174  }
175  if (posix_geteuid() != 0) {
```

Package: blog-url.rpc

blog-url/rpc/wp-blog-url-rpc.php, line 219 (Weak Cryptographic Hash)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: md5()
Enclosing Method: authenticate_peer()
File: blog-url/rpc/wp-blog-url-rpc.php:219
Taint Flags:

```
216  error_log("wp-blog-url-rpc: missing password");
217  return false;
218  }
219  if (md5($_POST["password"]) != $peer->password) {
220  header("HTTP/1.1 401 Unauthorized");
221  error_log("wp-blog-url-rpc: password incorrect");
222  return false;
```

Package: steps-password-policy

steps-password-policy/steps-password-policy.php, line 639 (Weak Cryptographic Hash)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: md5()
Enclosing Method: linear_search()

Weak Cryptographic Hash — Low

Package: steps-password-policy

steps-password-policy/steps-password-policy.php, line 639 (Weak Cryptographic Hash)

File: steps-password-policy/steps-password-policy.php:639
Taint Flags:

```
636  }
637
638  // 對輸入密碼進行 md5 雜湊處理
639  $target_hash = md5($target);
640
641  // 逐行讀取檔案進行比對
642  while (($line = fgets($file)) !== false) {
```

steps-password-policy/steps-password-policy.php, line 576 (Weak Cryptographic Hash)

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Semantic)

Sink Details

Sink: md5()
Enclosing Method: binary_search()
File: steps-password-policy/steps-password-policy.php:576
Taint Flags:

```
573  return false;
574  }
575
576  $target_hash = md5(stripslashes($target));
577
578  // 確認文件大小
579  fseek($file, 0, SEEK_END);
```

Description of Key Terminology

Likelihood and Impact

Likelihood
Likelihood is the probability that a vulnerability will be accurately identified and successfully exploited.

Impact
Impact is the potential damage an attacker could do to assets by successfully exploiting a vulnerability. This damage can be in the form of, but not limited to, financial loss, compliance violation, loss of brand reputation, and negative publicity.

Fortify Priority Order

Critical
Critical-priority issues have high impact and high likelihood. Critical-priority issues are easy to detect and exploit and result in large asset damage. These issues represent the highest security risk to the application. As such, they should be remediated immediately.

SQL Injection is an example of a critical issue.

High
High-priority issues have high impact and low likelihood. High-priority issues are often difficult to detect and exploit, but can result in large asset damage. These issues represent a high security risk to the application. High-priority issues should be remediated in the next scheduled patch release.

Password Management: Hardcoded Password is an example of a high issue.

Medium
Medium-priority issues have low impact and high likelihood. Medium-priority issues are easy to detect and exploit, but typically result in small asset damage. These issues represent a moderate security risk to the application. Medium-priority issues should be remediated in the next scheduled product update.

Path Manipulation is an example of a medium issue.

Low
Low-priority issues have low impact and low likelihood. Low-priority issues can be difficult to detect and exploit and typically result in small asset damage. These issues represent a minor security risk to the application. Low-priority issues should be remediated as time allows.

Dead Code is an example of a low issue.

About Fortify Solutions

Fortify is the leader in end-to-end application security solutions with the flexibility of testing on-premise and on-demand to cover the entire software development lifecycle. Learn more at www.microfocus.com/solutions/application-security.