A Pentester's Guide to Source Code Review

This blog post guides how to conduct a source code review project, focusing on advice for those new to the task. The post covers the purpose of a source code review, the process for conducting one, and the information needed to conduct a proper assessment.

MAY 15, 2023 • EST READ TIME: 17 MIN

NILESH SAPARIYA

Everyone who has achieved mastery began as a novice. This blog post guides how to manage and conduct a source code review project, particularly if you are new to the task and have to do it. It offers advice on approach and execution to help readers complete the source code review assessment successfully.

Source code review is the process of thoroughly examining and evaluating the source code of an application to identify any potential security vulnerabilities at the code level. From a security engineer's perspective, the goal of a source code review is to identify and mitigate any issues that could be exploited by an attacker, such as SQL injection, cross-site scripting, or other types of code-level vulnerabilities.

What should I do if assigned a project to conduct a source code review assessment?

- What is the process for conducting a source code review?
- What information should I ask the client to conduct a source code review assessment properly?

- How can a source code review be conducted?

If you are still getting familiar with conducting a source code review, this guide will assist you in preparing and conducting the assessment effectively. Before discussing the specific context of a client, let's first gain a general understanding of code review.

Understanding Secure Code Review in Secure Software Development

Secure Code Review is a process used to identify potential security vulnerabilities in software source code. It is an important part of a secure SDLC process and can be used to detect common coding flaws such as buffer overflows, SQL injection, and cross-site scripting.

Secure code review aims to produce secure and reliable software by identifying and correcting security vulnerabilities before threat actors exploit them. During the review, trained security professionals review the application source code and look for potential security flaws. They then document their findings and provide recommendations for remediation. Secure code review is essential for ensuring secure software development and should be part of any organization's security program.

Benefits of Secure Code Review in Secure Software Development

Most obvious from the pentester's perspective would be to find security flaws!

One of the primary reasons for conducting a secure code review is to identify and address security flaws at the code level. This includes identifying vulnerabilities such as SQL injection, cross-site scripting, and other types of security risks that could be exploited by attackers.

Protection of information and assets

Secure code review helps to protect the organization's sensitive information and assets by identifying and addressing vulnerabilities in the code. This can help to prevent data breaches and protect the organization's reputation. Secure code review can also help find hard-coded secrets & API Keys, saving tons of money.

Effort

Conducting a secure code review can save a lot of effort in the long run. Fixing vulnerabilities early in the development process is much easier and less expensive than fixing them after the software has been deployed in production.

Cost

Secure code review can help save costs by identifying and fixing vulnerabilities early in development. This can prevent costly security breaches and reduce the need for expensive remediation efforts.

Compliance

Many organizations are required to comply with regulations and standards, such as PCI-DSS, and SOC II, that mandate secure code review as a requirement. Conducting a secure code review can help organizations meet these compliance requirements.

Reputation

A data breach can profoundly affect an organization's standing. Secure code review can help to protect an organization's reputation by identifying and addressing vulnerabilities in the code before attackers can exploit them.

Preparing for a Source Code Review: Methods and Approach

If you have a task or project involving a source code review. To ensure the review is done well and thoroughly, there are a few things you need to do before the engagement even begins. Let's look at its methods and approach.

Identify Security Code Review Objectives:

- Identifying the objectives for the security code review is the first step in the process.
- These objectives will determine what specific code areas will be reviewed and what specific security vulnerabilities will be looked for.
- Examples of objectives could include identifying and fixing SQL injection vulnerabilities, ensuring compliance with industry standards, or reducing the overall attack surface of the application.
- Establishing a clear and mutual understanding of the goals and objectives of a source code review with the client is essential before starting the engagement. This will ensure that both parties are on the same page and

have a common understanding of the purpose and expectations of the review.

Prerequisites for Code Review:

Gathering specific information from the client is essential before beginning a source code review engagement. This includes

- **Access**: Obtain access to the source code for the module under review. This can be access on GitHub/GitLab instance or any internal code hosting service.
- **LOC (Lines of Code)**: A critical step in preparation for a source code review is determining the total number of lines of code (LOC) in the module under review. This information will aid in calculating the estimated number of work hours necessary to complete the activity.
- **Hardware Requirement for license tool installation**: One of the prerequisites for a source code review is to confirm with the client whether the review will be conducted on a machine provided by the client or on the pentester’s own machine. This is important to plan for as it will affect any enterprise tools requiring a license.
- **Walk-through**: Have a session with the developer team to discuss the code and any concerns. The most significant aspect of this section is the Walk-through with the development team. Meeting with the application’s developer to discuss the code’s goals is the key step of a secure code review.

Sr. No	Area of focus	What to ask
1	Authentication	Are application users authenticated, or are they all treated as anonymous users? What factors are used for authentication (such as passwords, tokens, and certificates)?

		If passwords are being used, are there any policies regarding complexity or age in place?
2	Authorization	Are there different roles that users can be given depending on the applications' function? Is the authorization data cached checked for each incoming request? Are there any private, sensitive data files stored in the web root that is not authorized for the regular user?
3	Data Validation	Is the user-submitted data validated? Is data validated as soon as it comes in from the user or when it is used by the code? How is the data validation accomplished (whitelisting, blacklisting, min/max, etc.)? Are you using a database? If so, are you using prepared statements?
4	Exception/Error	What approach(s) for error handling is

	Handling	being used? What kind of details about an error are displayed to the user? Are error traces ever sent back to the user? Or are they sent to logs only? Is the sensitive data in error logged to the log files? If the database throws an error, is the error message sent to the user, or is it passed to a log?
5	Session Management	Is there any way the application manages or stores session state, and if so, how? How is the session id being generated? Is the previous session deleted when a user logs into the site and creates a new session? Are tokens used for session management? If yes, what algorithm is used? Any timeouts for sessions?

		If cookies are used, are there path and domain restrictions in the cookie?
6	Logging	Is any type of logging being used within the code? Where are generated log messages sent? Are users who shouldn't have access to the log files able to access them (any or all employees)? Are you logging any input that is not validated first or data that has failed validation? Are log messages time-stamped? Is any sensitive data written to a log (e.g., password, API key, etc.)?
7	Encryption	Are there any encryption algorithms used within the code at all? (SSL, TLS, RSA?) Where did you get the library's implementation, and what version is it using? If using 3DES or AES (any block cipher), what encryption mode is used?

		Is there a central function in the code that handles encryption? Where is it?

This is just a high-level overview of the checks you can request; feel free to add or modify any other checks as needed.

Perform Preliminary Scan

But why use Tools for Source Code Analysis?

The purpose of source code review tools is to automate and streamline the process of reviewing large amounts of code, typically measured in thousands or millions of lines. This is necessary as performing a manual line-by-line analysis of such a vast amount of code is virtually impossible. Source code review tools aid in the efficient identification and resolution of potential security vulnerabilities, coding errors, and performance issues.

- After the objectives and necessary information have been gathered, the next step is to perform a preliminary scan of the source code.
- Based on the total number of lines of code (LOC), the approach for conducting the test will be determined.
- This is often the case when the number of lines of code is high. This scan can be done using automated tools or manual review methods to identify potential vulnerabilities or areas of the code that may require further examination.
- Benefits of SAST tools:

    - Reduction in manual effort
    - Time efficient
    - Find all the instances of vulnerabilities
    - Source to sink analysis
    - Exhaustive coverage of vulnerability patterns
    - Elaborate reporting format
- Limitations of SAST tools
    - Unable to detect Business Logic Flaws
    - Limited Scope
    - Custom Validations
    - Design Flaws
    - Application Specific Recommendations

Source Code Review – Free Tools

- VCG - VisualCodeGrepper
- Yasca - Yet Another Source Code Analyzer
- Findbugs
- RIPS Scanner
- OWASP Orizon

Note: Many open-source tools are available for examining and evaluating source code during a review process. Some examples have been highlighted. However, it is crucial to remember that each tool has its specific programming language compatibility. It is recommended to thoroughly examine and assess the options before deciding which tool to utilize.

Source Code Review – Commercial Tools

- Checkmarx Static Application Security Testing (SAST)
- AppScan Source - HCL
- Veracode

Note: Many commercial tools are available for examining and evaluating source code during a review process. Selecting the correct source code review tool for your project is essential. When choosing a tool, be sure to consider its programming language compatibility and features. Consider your project's needs and evaluate the various enterprise tools available to determine which one best meets those needs.

Key Points to Consider When Reviewing Small Code Bases Manually

If you have a relatively small amount of code to review, such as several thousand lines, and you plan to conduct a manual assessment, there are several key points to remember when examining the source code.

The following are high-level findings that can be easily identified in the code: while there may be more, the points covered below provide a good starting point.

Basics: One of the most important prerequisites for conducting a code review is understanding at least one object-oriented framework, such as J2EE or .NET.

Application Details: The next critical step in code review is to comprehend the workings of the application and the settings it employs. The deployment descriptor, such as web.xml in J2EE or web.config in .NET, is the entry point for this process. In web service testing, it's essential to understand the application's purpose, business logic, and data flow. Examining the application can reveal vital information, including its APIs and third-party libraries.

Following are a few terminologies of which a code reviewer should be aware:

- **Source**: The "source" refers to the location in the code where malicious input was introduced, such as using the "request.getParameter()" method.
- **Sink**: The "sink" is the location in the code where a vulnerability is exploited, such as where XSS alerts are reflected.
- **Taint**: "Taint" refers to malicious data provided by the user.
- **Taint propagator**: The "taint propagator" function takes malicious data as input and then passes it out without any validation.

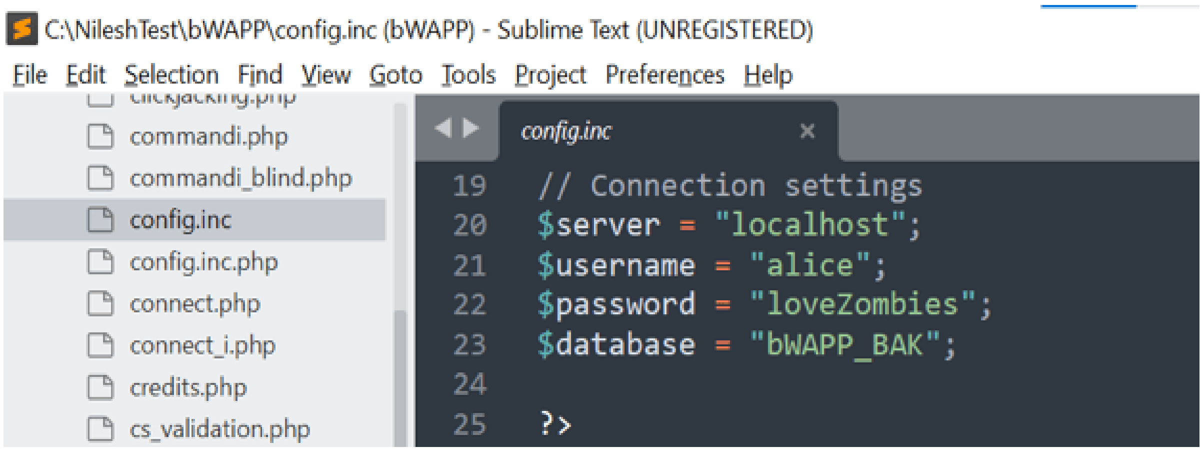
Area of focus during code review:

- Input validation and output encoding
- Authentication & Authorization
- Cryptography
- Session management
- Threat Modeling Terminology

Some high-level evergreen findings can be easily discovered during a manual code review assessment. Note that this list is not exhaustive, and it's meant to provide a general idea. You can create your own checklist based on these findings.

1. Hard-Coded Secrets in source code

- **Description**: One of the most straightforward issues to identify during a source code review is the presence of hard-coded secrets such as API Keys, tokens, passwords, etc. By using the find ("Ctrl+F") function and searching for specific keywords such as "token" or "password," it is easy to determine if the source code contains any hard-coded credentials.
- **Recommendation**: To address the issue of hard-coded secrets in the source code, it is recommended to remove the secret from the code and instead use a secure secret management solution, aka KMS(Key Management Service), such HashiCorp Vault, to store the secrets.
- The diagram below illustrates that the "config.inc" file contains a hard-coded password & database name.


C:\NileshTest\bWAPP\config.inc (bWAPP) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
clickjacking.php
commandi.php
commandi_blind.php
config.inc
config.inc.php
connect.php
connect_i.php
credits.php
cs_validation.php
config.inc

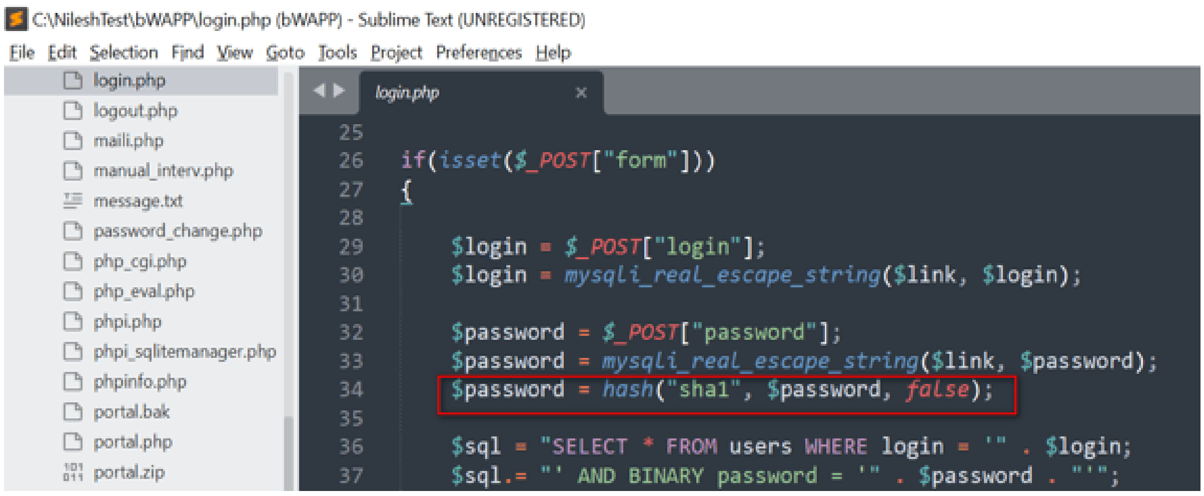
```
// Connection settings
$server = "localhost";
$username = "alice";
$password = "loveZombies";
$database = "bWAPP_BAK";

?>
```

2. Insecure Algorithm Used

- **Description**: To quickly find a list of insecure algorithms that have been used, you can again use the find ("Ctrl+F") function and search for specific keywords such as SHA-1 (or sha1), md5, DES, and RC4. These keywords will help you easily locate information on algorithms considered to be insecure.
- **Recommendation**: To ensure secure cryptography, it's important to choose the right algorithms for the task at hand. For symmetric key cryptography, use AES or Twofish, while RSA is recommended for asymmetric key encryption with a minimum key length of 2048 bits. SHA-256 is now the

generally recommended hash function, and SHA-3 algorithms are preferred for new development. For applications with high-security needs, consider AES-256, RSA with larger key sizes, and SHA-512. Stay updated with best practices to ensure maximum security.

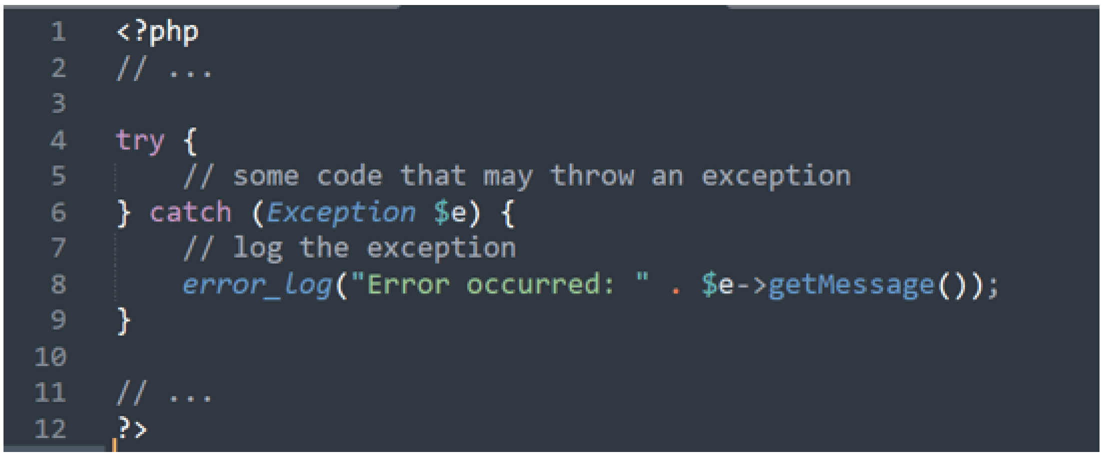
- The diagram below illustrates the code $password = hash("sha1", $password, false), using the SHA-1 hashing algorithm to create a hash of the password. However, SHA-1 is no longer considered a secure hashing algorithm as it is vulnerable to collision attacks. This means that an attacker could create two different inputs that produce the same hash value, making it easier to break the hash and gain unauthorized access. Therefore, it is not safe to use SHA-1 for hashing passwords.



```
if(isset($_POST["form"]))
{

    $login = $_POST["login"];
    $login = mysqli_real_escape_string($link, $login);

    $password = $_POST["password"];
    $password = mysqli_real_escape_string($link, $password);
    $password = hash("sha1", $password, false);

    $sql = "SELECT * FROM users WHERE login = '" . $login;
    $sql.= "' AND BINARY password = '" . $password . "'";
```

3. Improper Exception Handling

- **Description**: The diagram below illustrates the catch block catches all exceptions of the Exception type, which is too general and can lead to unexpected behavior. An attacker could exploit this by intentionally throwing an exception that the application is unprepared to handle, causing the application to crash or behave unexpectedly.
- **Recommendation**: To fix this issue, the application should catch only specific exception types that it expects and can handle appropriately. Additionally, the application should provide informative error messages and log relevant information, such as the stack trace, to help diagnose issues and improve security.



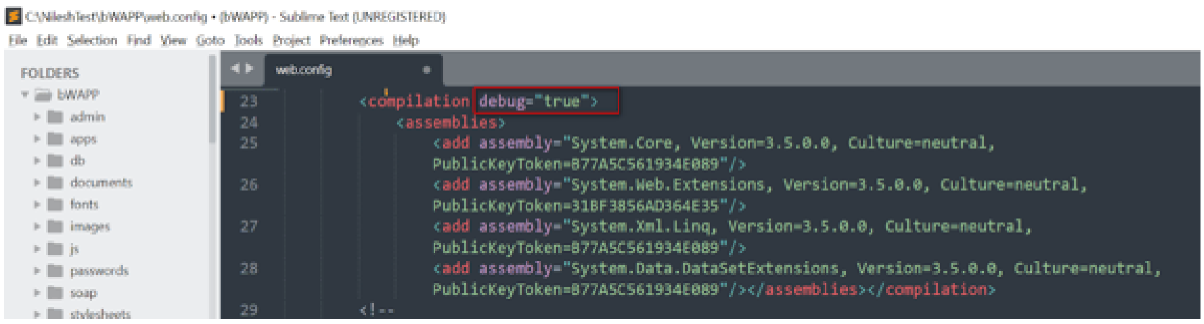
```php
<?php
// ...

try {
    // some code that may throw an exception
} catch (Exception $e) {
    // log the exception
    error_log("Error occurred: " . $e->getMessage());
}

// ...
?>
```

4. Application Logs in Clear Text

- **Description**: During source code review, searching for log statements that may write sensitive information in clear text is crucial. Firstly, locate log files created by the application, which may be in various forms, such as text files, databases, or other storage mechanisms. Next, scrutinize the log messages to identify any sensitive information like passwords, usernames, credit card numbers, API keys, or other PII (personally identifiable information) that may be logged. It is important to ensure that sensitive information is not logged in clear text as it can lead to sensitive data exposure, and thus reviewing application logs is an essential security measure.
- **Recommendation**: It is advisable not to log sensitive data, encrypt the logs and limit access to authorized users with appropriate privileges.

5. DEBUG is enabled in the application

- **Description**: Enabling DEBUG mode in an application provides detailed information about its processes and any errors that may occur, which is helpful for developers to identify and resolve issues. However, it can be a security risk if the information is accessed by unauthorized parties. DEBUG mode should only be used during development and testing environments, not in a production where it can reveal sensitive information and potentially provide attackers with an opportunity for further exploitation.
- **Recommendation**: If DEBUG is enabled in the source code, the recommended fix is disabling it before deploying it to a production environment.
- The following diagram depicts that the source code has its debug mode set to true.


C:\NileshTest\bWAPP\web.config • (bWAPP) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bWAPP
admin
apps
db
documents
fonts
images
js
passwords
soap
stylesheets
web.config

```
<compilation debug="true">
  <assemblies>
    <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
    <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=31BF3856AD364E35"/>
    <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
    <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/></assemblies></compilation>
<!--
```

Via Enterprise License Tool

- **Tool Name**: I will be using AppScan Source for the Analysis tool
- **Practical Example**: By using the source code of bWAPP – buggy web application.
- **Programming Language**: PHP
- **Assumption**: When conducting the source code review for this engagement, it's important to consider that if client has provided a large codebase that cannot be manually reviewed due to its LOC (Lines of code), automated tools or scripts may need to be employed to assist with the analysis and ensure thorough coverage of the code.

AppScan Source Tool

When you open the tool, it looks something like this

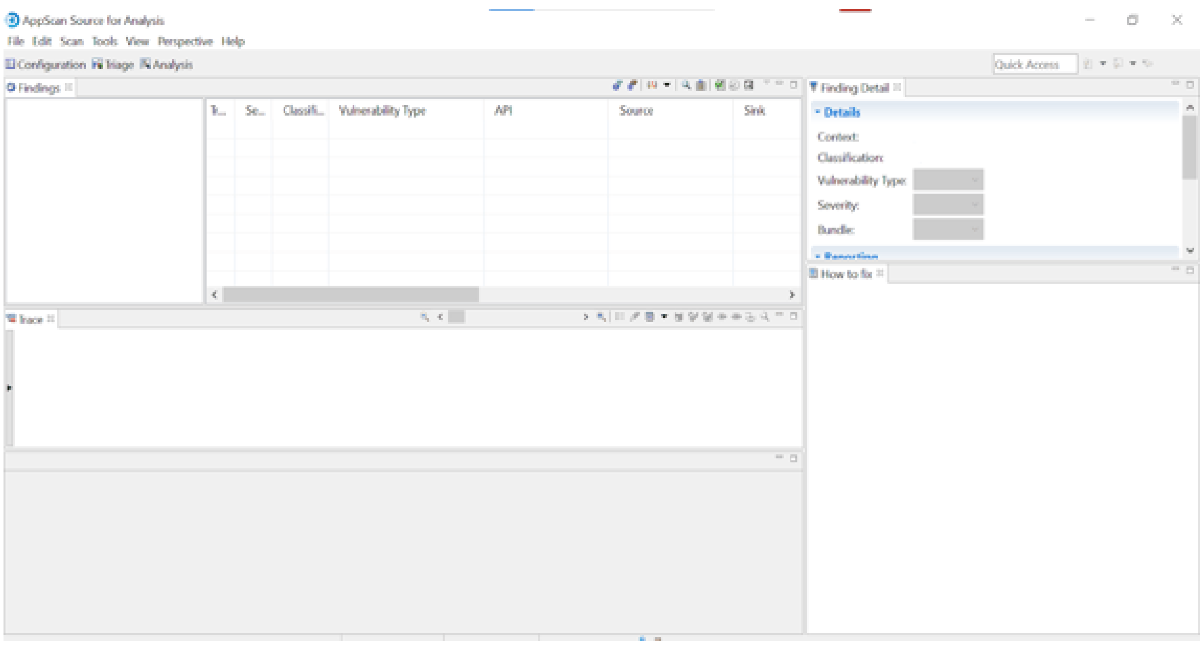
AppScan Source

Version 10.0.8

July 2022

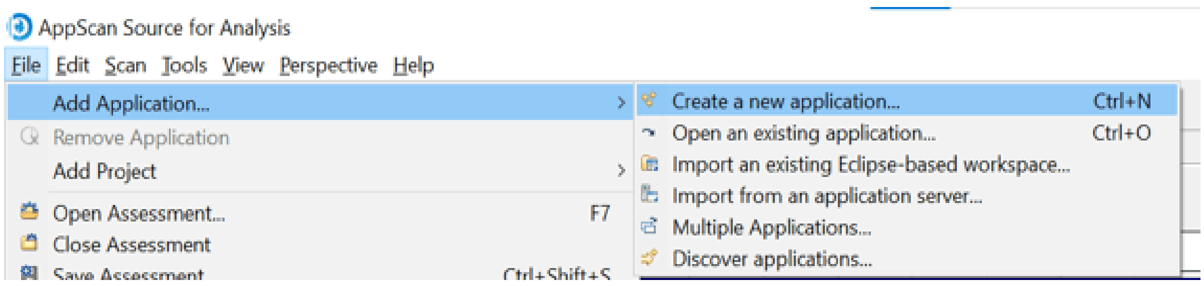
Licensed Materials- Property of HCL Technologies Ltd. Copyright HCL Technologies Ltd.2003, 2022. All rights reserved. See product license for details. US Government Users Restricted Rights. Use, duplication, or disclosure restricted by GSA ADP Schedule Contract with HCL Technologies Ltd. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and / or its affiliates. HCL and the HCL logo are registered trademarks of HCL Technologies in the United States, other countries, or both.

GUI Representation

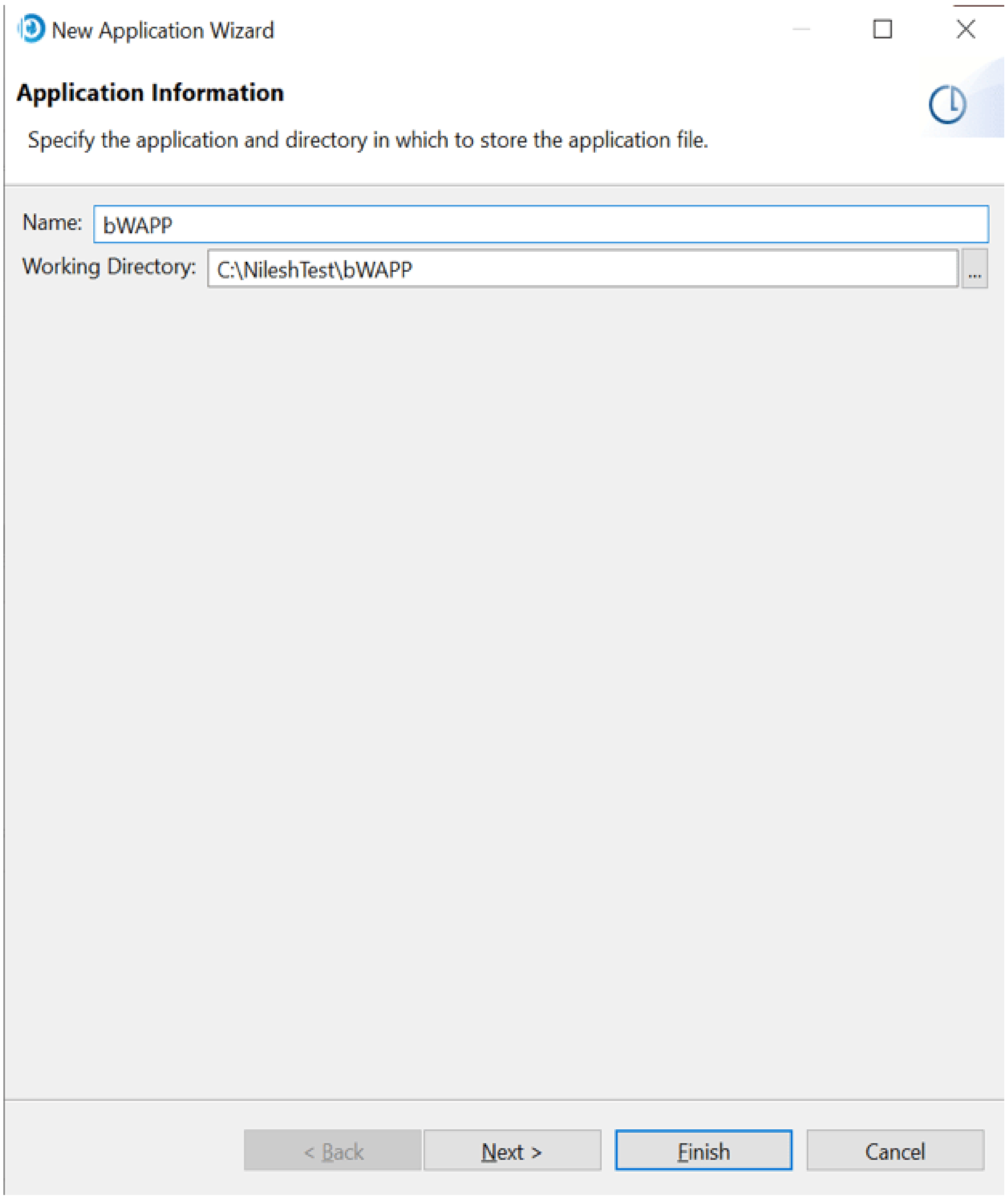

AppScan Source for Analysis
File Edit Scan Tools View Perspective Help
Configuration Triage Analysis
Quick Access
Findings
T... Se... Classifi... Vulnerability Type API Source Sink
Finding Detail
Details
Context:
Classification:
Vulnerability Type:
Severity:
Bundle:
Reporting
How to fix
Trace

You can Start running the scan by going in

```
File => Add Application => Create a new
application
```


AppScan Source for Analysis
File Edit Scan Tools View Perspective Help
Add Application...
Remove Application
Add Project
Open Assessment... F7
Close Assessment
Save Assessment Ctrl+Shift+S
Create a new application... Ctrl+N
Open an existing application... Ctrl+O
Import an existing Eclipse-based workspace...
Import from an application server...
Multiple Applications...
Discover applications...

Give the Name and the working directory of the code and click on Next

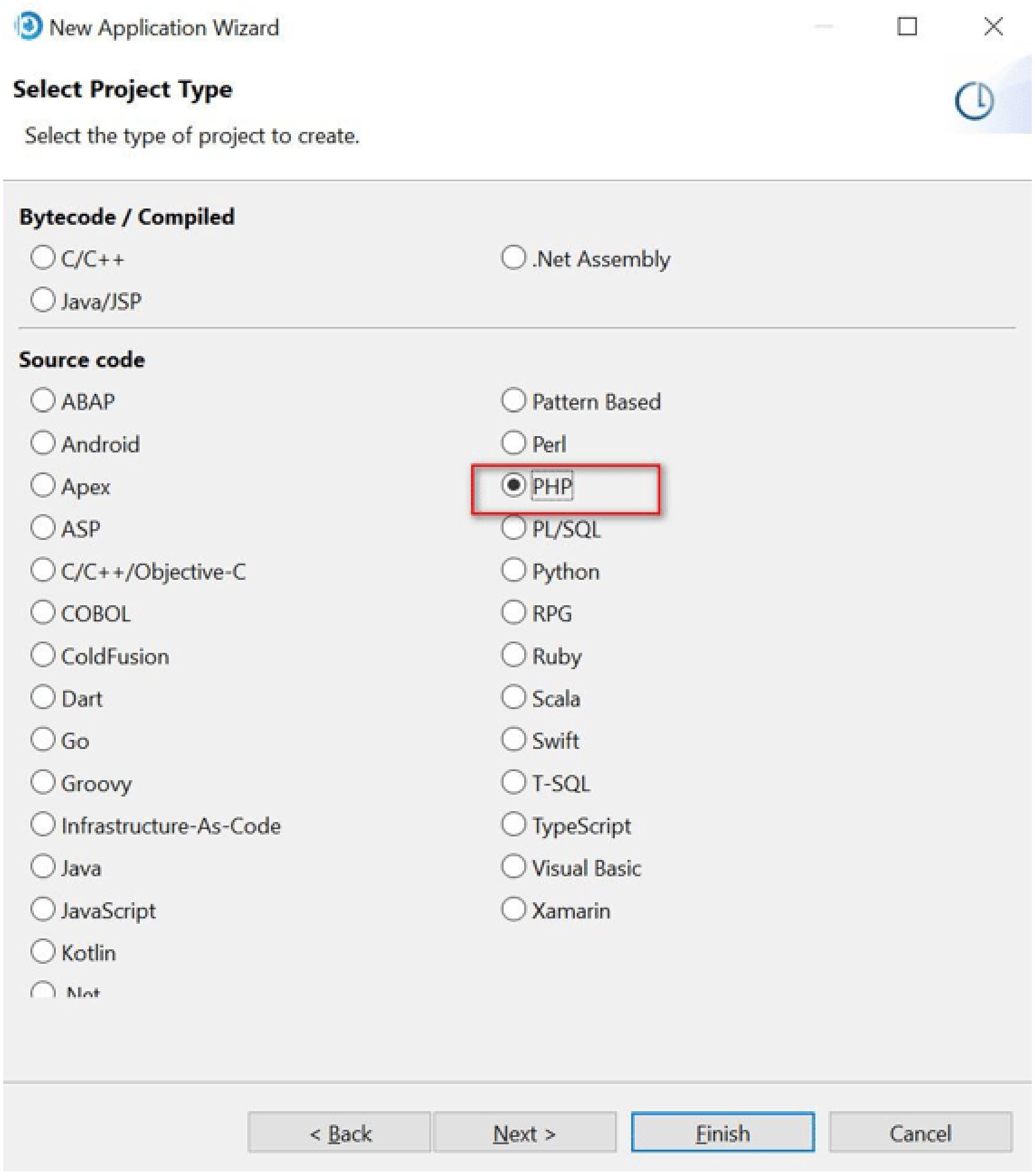

New Application Wizard
Application Information
Specify the application and directory in which to store the application file.

Name: bWAPP

Working Directory: C:\NileshTest\bWAPP

...
< Back
Next >
Finish
Cancel

Now Select the language.


New Application Wizard

Select Project Type

Select the type of project to create.

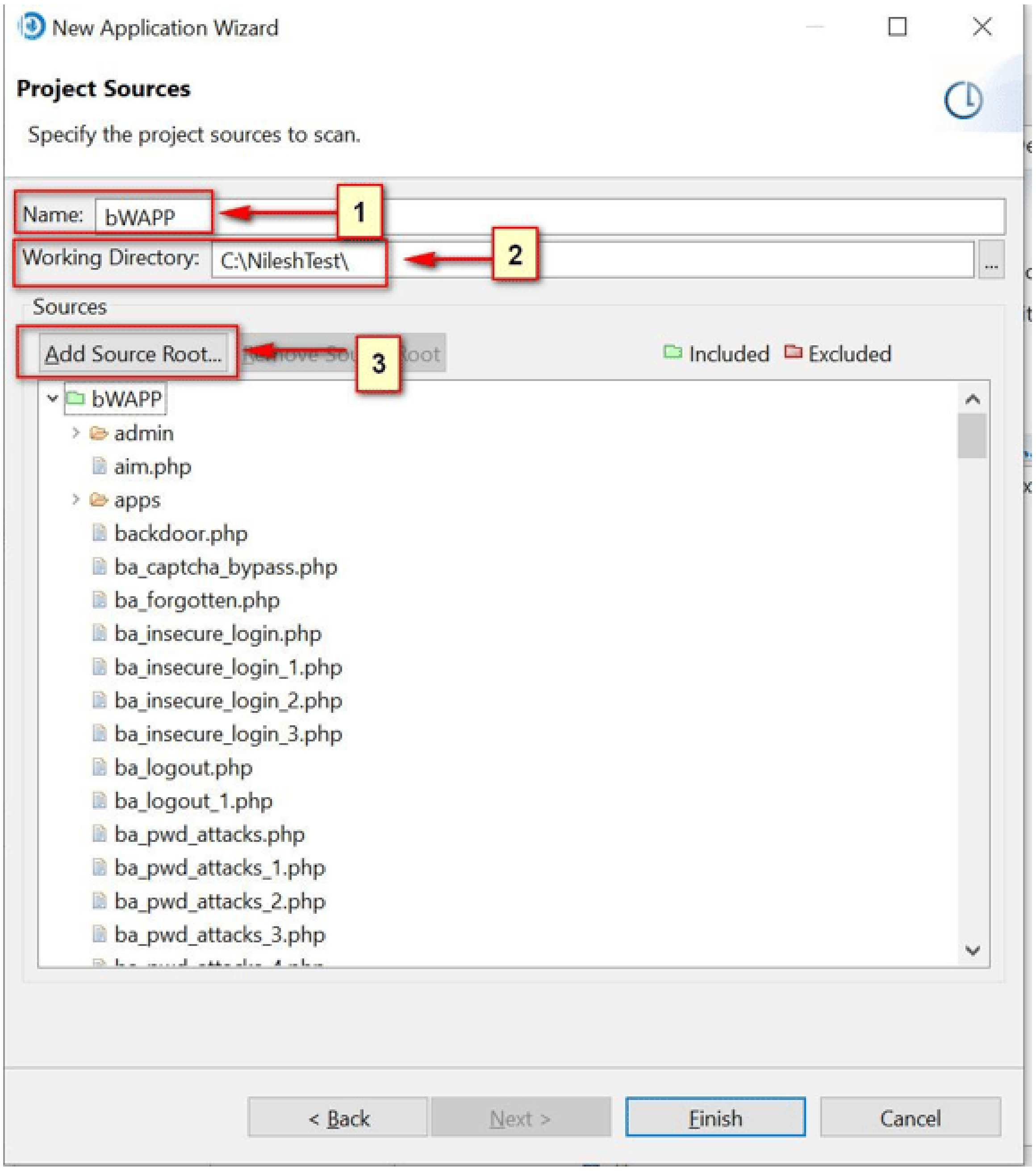
Bytecode / Compiled

- ( ) C/C++
- ( ) .Net Assembly
- ( ) Java/JSP

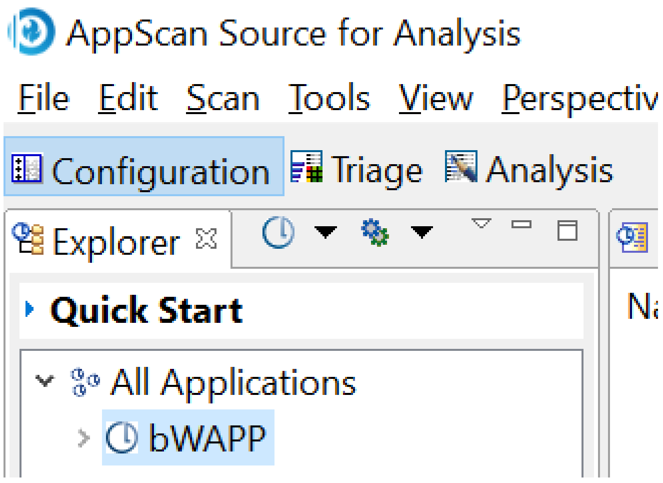
Source code

- ( ) ABAP
- ( ) Android
- ( ) Apex
- ( ) ASP
- ( ) C/C++/Objective-C
- ( ) COBOL
- ( ) ColdFusion
- ( ) Dart
- ( ) Go
- ( ) Groovy
- ( ) Infrastructure-As-Code
- ( ) Java
- ( ) JavaScript
- ( ) Kotlin
- ( ) Net
- ( ) Pattern Based
- ( ) Perl
- [✓ PHP]
- ( ) PL/SQL
- ( ) Python
- ( ) RPG
- ( ) Ruby
- ( ) Scala
- ( ) Swift
- ( ) T-SQL
- ( ) TypeScript
- ( ) Visual Basic
- ( ) Xamarin

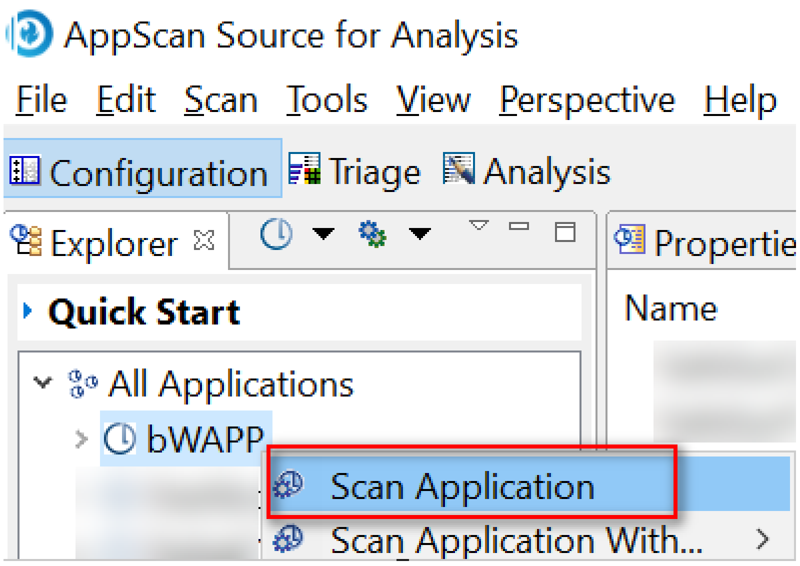
Click Next and Give Name, specify the Working directory, and click on Add Source Root Folder



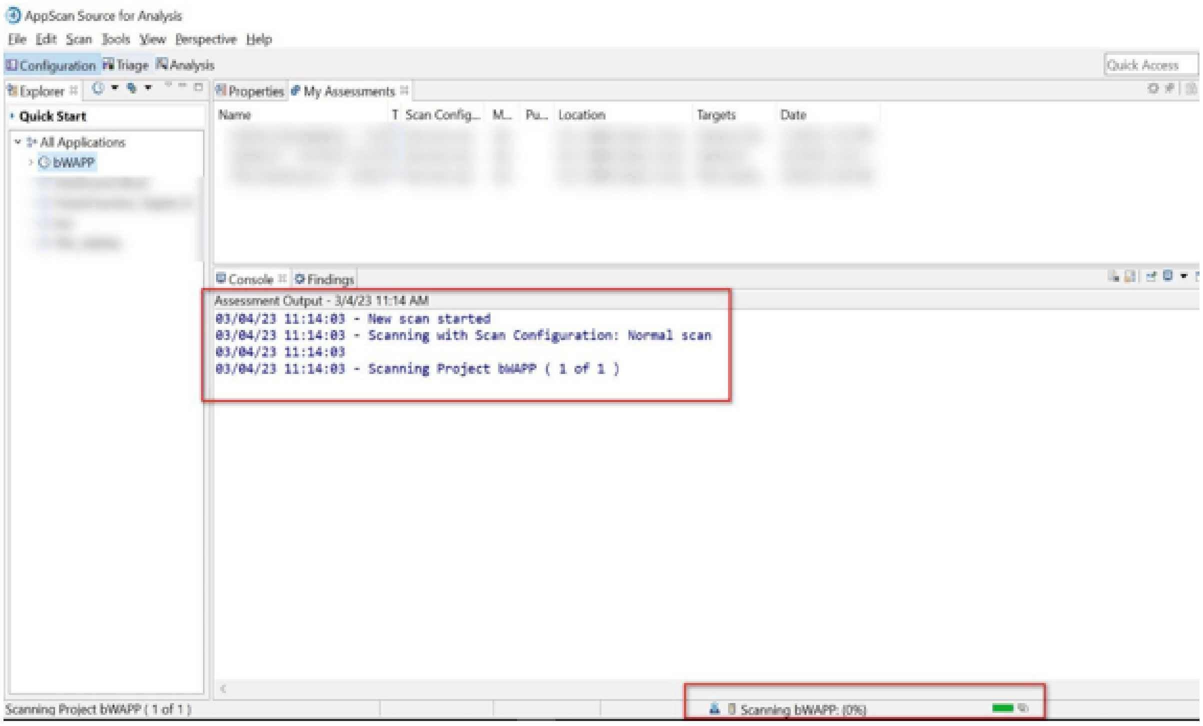
Click on Finish and Go to Configuration



Right-click on your project and Scan Application



After the scan, the result looks something like this

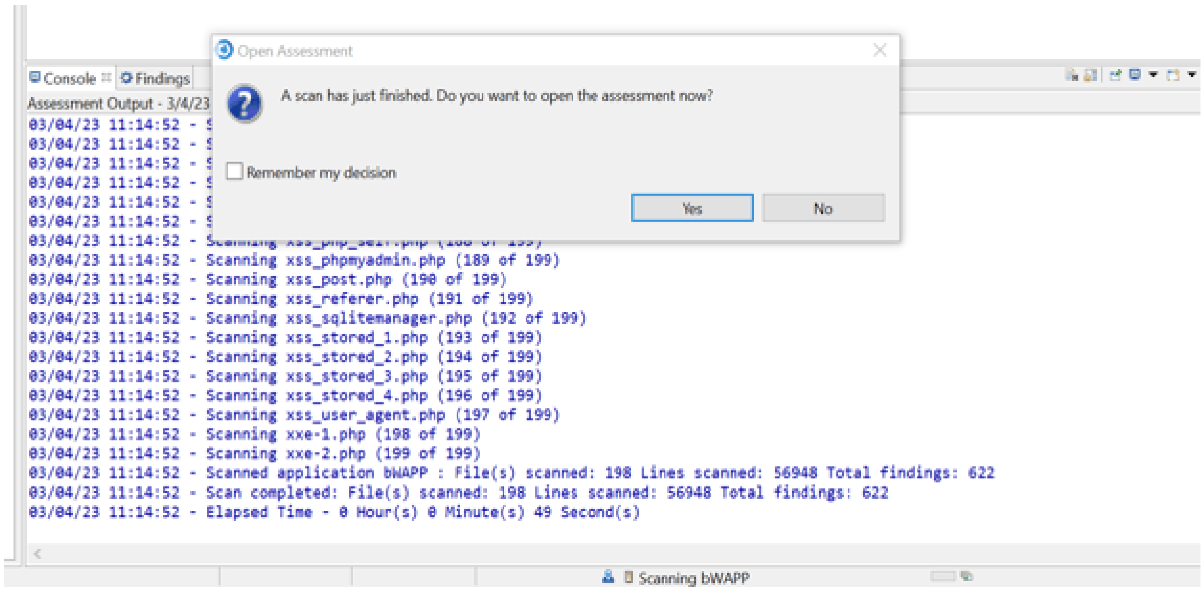


```
Assessment Output - 3/4/23 11:14 AM
03/04/23 11:14:03 - New scan started
03/04/23 11:14:03 - Scanning with Scan Configuration: Normal scan
03/04/23 11:14:03
03/04/23 11:14:03 - Scanning Project bWAPP ( 1 of 1 )
```

Scanning Project bWAPP (1 of 1)

Scanning bWAPP: (0%)

Click Yes



Open Assessment

A scan has just finished. Do you want to open the assessment now?

☐ Remember my decision

Yes No

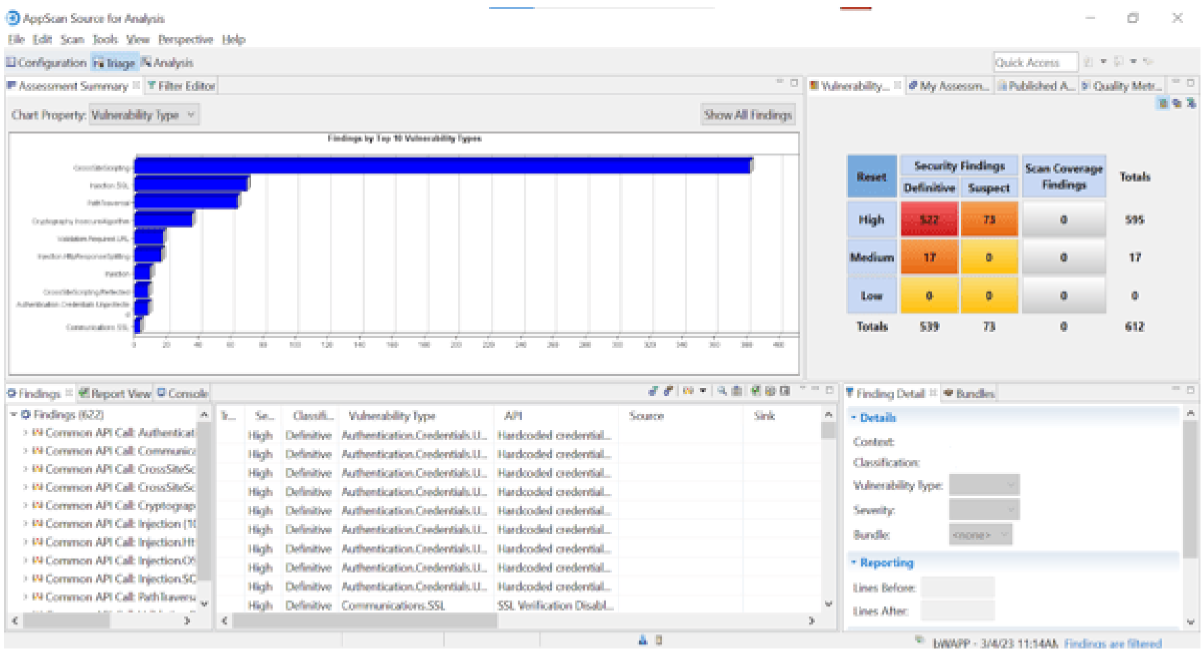
Console Findings

Assessment Output - 3/4/23

```
03/04/23 11:14:52 - Scanning xss_phpmyadmin.php (189 of 199)
03/04/23 11:14:52 - Scanning xss_post.php (190 of 199)
03/04/23 11:14:52 - Scanning xss_referer.php (191 of 199)
03/04/23 11:14:52 - Scanning xss_sqlitemanager.php (192 of 199)
03/04/23 11:14:52 - Scanning xss_stored_1.php (193 of 199)
03/04/23 11:14:52 - Scanning xss_stored_2.php (194 of 199)
03/04/23 11:14:52 - Scanning xss_stored_3.php (195 of 199)
03/04/23 11:14:52 - Scanning xss_stored_4.php (196 of 199)
03/04/23 11:14:52 - Scanning xss_user_agent.php (197 of 199)
03/04/23 11:14:52 - Scanning xxe-1.php (198 of 199)
03/04/23 11:14:52 - Scanning xxe-2.php (199 of 199)
03/04/23 11:14:52 - Scanned application bWAPP : File(s) scanned: 198 Lines scanned: 56948 Total findings: 622
03/04/23 11:14:52 - Scan completed: File(s) scanned: 198 Lines scanned: 56948 Total findings: 622
03/04/23 11:14:52 - Elapsed Time - 0 Hour(s) 0 Minute(s) 49 Second(s)
```

Scanning bWAPP

After the scan, the result looks something like this


AppScan Source for Analysis
File Edit Scan Tools View Perspective Help
Configuration Triage Analysis
Assessment Summary Filter Editor
Chart Property: Vulnerability Type
Show All Findings
Findings by Top 10 Vulnerability Types
Vulnerability... My Assessm... Published A... Quality Metr...
Finding Detail Bundles
Details
Context:
Classification:
Vulnerability Type:
Severity:
Bundle:
Reporting
Lines Before:
Lines After:
bWAPP - 3/4/23 11:14AM Findings are filtered

Reset	Security Findings		Scan Coverage Findings	Totals
	Definitive	Suspect		
High	522	73	0	595
Medium	17	0	0	17
Low	0	0	0	0
Totals	539	73	0	612

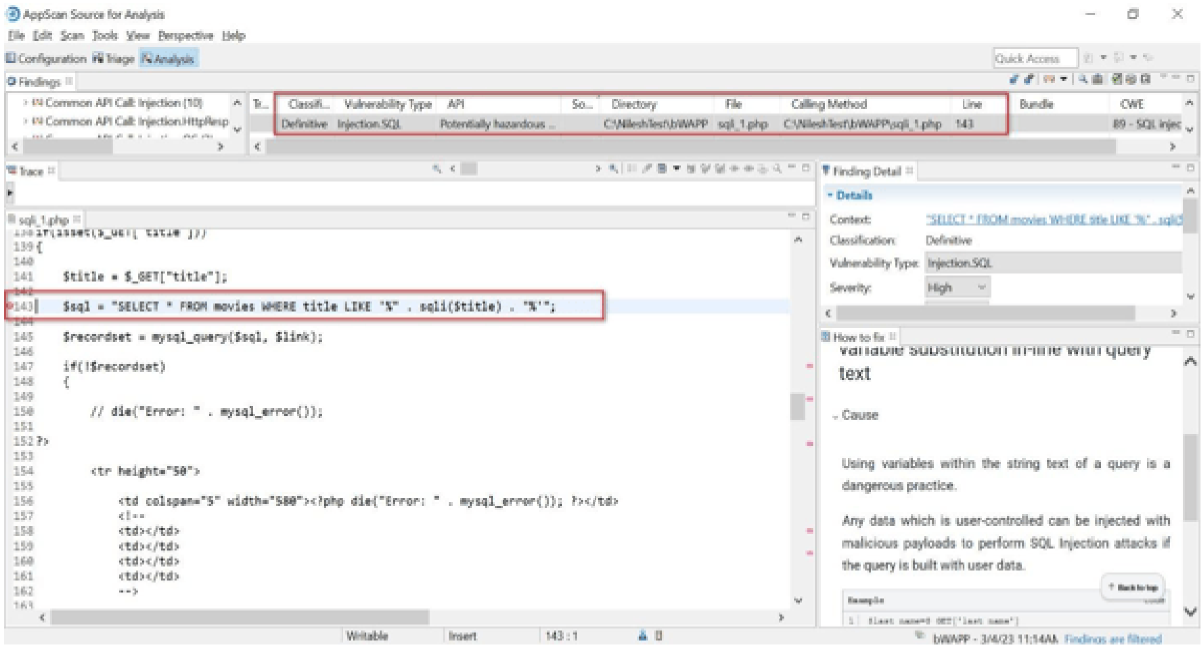
Findings Report View Console

Findings (622)
- Common API Call: Authenticat
- Common API Call: Communic
- Common API Call: CrossSiteSc
- Common API Call: CrossSiteSc
- Common API Call: Cryptograp
- Common API Call: Injection (1
- Common API Call: Injection.H
- Common API Call: Injection.O
- Common API Call: Injection.SC
- Common API Call: PathTravers

Tr...	Se...	Classifi...	Vulnerability Type	API	Source	Sink
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Authentication.Credentials.U...	Hardcoded credential...		
	High	Definitive	Communications.SSL	SSL Verification Disabl...		

From the above findings, we will take 3 vulnerabilities - SQL Injection, XSS, and OS Command Injection (Blind).

SQL Injection


AppScan Source for Analysis

Classifi...	Vulnerability Type	API	So...	Directory	File	Calling Method	Line	Bundle	CWE
Definitive	Injection.SQL	Potentially hazardous ...		C:\NileshTest\bWAPP	sqli_1.php	C:\NileshTest\bWAPP\sqli_1.php	143		89 - SQL injec

```
139 {
140
141     $title = $_GET["title"];
142
143     $sql = "SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'";
144
145     $recordset = mysql_query($sql, $link);
146
147     if(!$recordset)
148     {
149
150         // die("Error: " . mysql_error());
151
152 ?>
153
154         <tr height="50">
155
156             <td colspan="5" width="580"><?php die("Error: " . mysql_error()); ?></td>
157             <!--
158             <td></td>
159             <td></td>
160             <td></td>
161             <td></td>
162             -->
```

Finding Detail

Context:	"SELECT * FROM movies WHERE title LIKE '%" . sqli(
Classification:	Definitive
Vulnerability Type:	Injection.SQL
Severity:	High

How to fix

text

Cause

Using variables within the string text of a query is a dangerous practice.

Any data which is user-controlled can be injected with malicious payloads to perform SQL Injection attacks if the query is built with user data.

bWAPP - 3/4/23 11:14AM Findings are filtered

So, the tool detected SQL Injection vulnerability at line 143. Let's first understand the code block that tool reported to us

`Source of the file C:\USER\bWAPP\sqli_1.php`

Code Explanation:

- This code uses GET method to retrieve a movie title based on the user input, and then it searches the "movies" table in the database using the retrieved title with a SQL LIKE statement. Note: The user input $_GET["title"] is used to build a SQL query without proper input sanitization.
- The code is vulnerable to SQL injection attacks since it directly includes the user input into the SQL statement.
- An attacker can craft a malicious input to execute arbitrary SQL statements, bypass authentication, or access sensitive data.

Remediation:

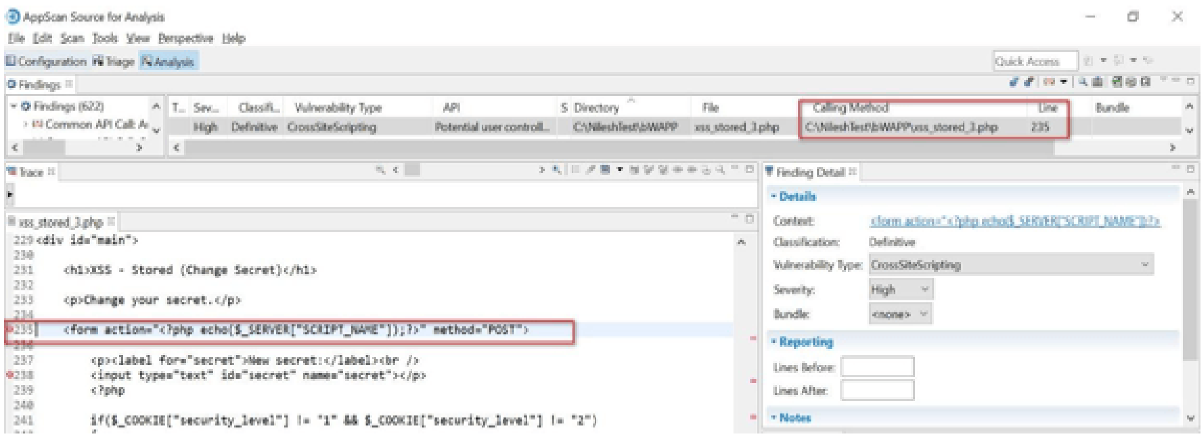
- To fix the vulnerability, the code needs to use parameterized queries, prepared statements, or escape characters to sanitize user input before using it in SQL statements.
- Moreover, the code should not reveal detailed error messages to the user, as it may provide clues for attackers to exploit the vulnerability.

Fixed Code:

```
if(isset($_GET["title"])) {

    $title = $_GET["title"];

        // Create a prepared statement

    $stmt = mysqli_prepare($link, "SELECT *
FROM movies WHERE title LIKE ?");

    // Bind the parameter

    mysqli_stmt_bind_param($stmt, "s",
$title);

    // Execute the query

    mysqli_stmt_execute($stmt);

    // Get the results

    $result = mysqli_stmt_get_result($stmt);

    // Check if any rows were returned

    if(mysqli_num_rows($result) > 0) {

        while($row =
mysqli_fetch_array($result)) {

            // Do something with the row data

        }
    } else {
        // No rows were returned
    }
}
```

In this modified code, a prepared statement is created using mysqli_prepare(), and the query is parameterized by replacing the variable with a question mark ?. The parameter is bound using mysqli_stmt_bind_param(), which ensures that any user input is properly sanitized and escaped from any malicious payload. Then, the query is executed using mysqli_stmt_execute(), and the results are retrieved using mysqli_stmt_get_result().

Cross-Site Scripting (XSS) attack


AppScan Source for Analysis
File Edit Scan Tools View Perspective Help
Configuration Triage Analysis
Quick Access
Findings
Findings (622)
Common API Call: A
Trace
Finding Detail
Details
Reporting
Notes

T...	Sev...	Classifi...	Vulnerability Type	API	S	Directory	File	Calling Method	Line	Bundle
	High	Definitive	CrossSiteScripting	Potential user controll...		C:\NileshTest\bWAPP	xss_stored_3.php	C:\NileshTest\bWAPP\xss_stored_3.php	235	

xss_stored_3.php

```
<div id="main">

    <h1>XSS - Stored (Change Secret)</h1>

    <p>Change your secret.</p>

    <form action="<?php echo($_SERVER["SCRIPT_NAME"]);?>" method="POST">

        <p><label for="secret">New secret:</label><br />
        <input type="text" id="secret" name="secret"></p>
        <?php

        if($_COOKIE["security_level"] != "1" && $_COOKIE["security_level"] != "2")
```

Context: <form action="<?php echo($_SERVER["SCRIPT_NAME"]);?>
Classification: Definitive
Vulnerability Type: CrossSiteScripting
Severity: High
Bundle: <none>
Lines Before:
Lines After:

The vulnerability in this code is that it's vulnerable to Cross-Site Scripting (XSS) attacks. The `$_SERVER["SCRIPT_NAME"]` variable is not properly sanitized, which means that an attacker can inject a malicious payload into the form action attribute and execute it in the victim's browser.

Remediation:

There are various ways to protect against cross-site scripting attacks within PHP.

- htmlspecialchars: Converts characters &, ", ', < and > to their HTML entity representation, preventing XSS from using those characters. This will not stop any user code javascript replacements from running on the system
- htmlentities: Converts all characters with an HTML entity replacement, including those with htmlspecialchars. This is more complete protection from cross-site scripting vulnerability but can lead to display bugs if certain characters are not replaced.

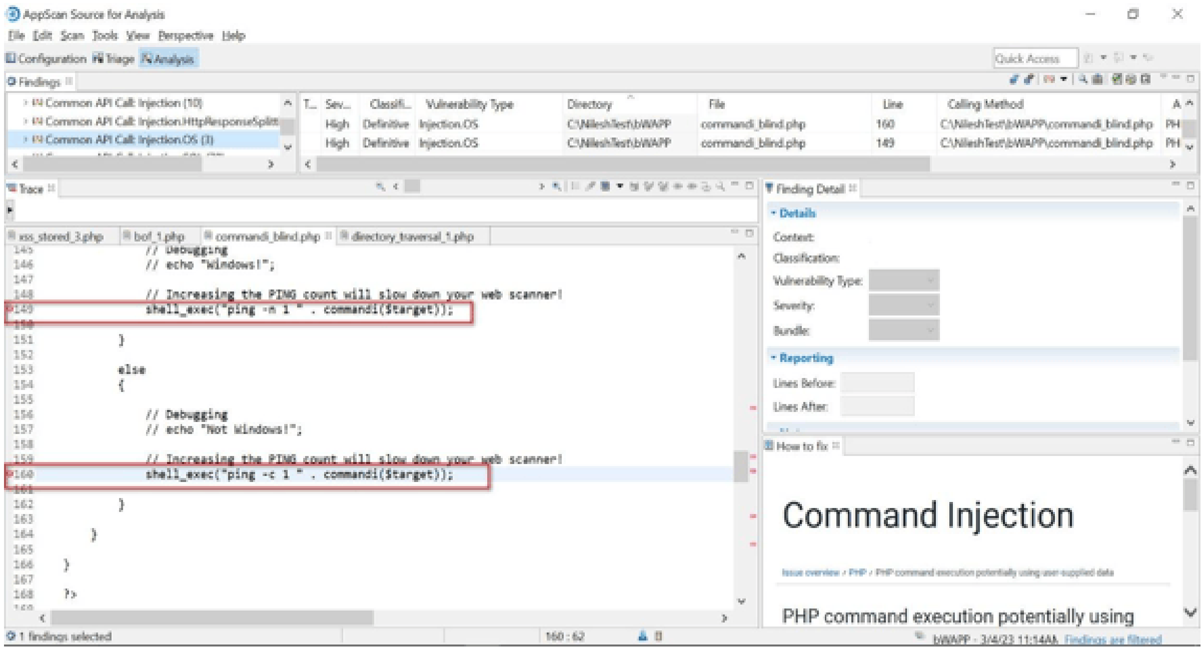
Example:

```
echo '<div>' .
htmlspecialchars($_GET['input']) .
'</div>';
```

This line of code takes a user input passed through the GET request, then wraps it with an HTML div element after encoding it using the htmlspecialchars() function.

htmlspecialchars() is a PHP function that converts special characters to their HTML entities. This helps prevent XSS attacks by preventing malicious scripts from being executed when the user input is displayed on the webpage.

OS Command Injection (Blind)


AppScan Source for Analysis
File Edit Scan Tools View Perspective Help
Configuration Triage Analysis
Findings
Common API Call: Injection (10)
Common API Call: Injection.HttpResponseSplit
Common API Call: Injection.OS (3)
Trace
Finding Detail
Details
Context:
Classification:
Vulnerability Type:
Severity:
Bundle:
Reporting
Lines Before:
Lines After:
How to fix
Command Injection
Issue overview / PHP / PHP command execution potentially using user-supplied data
PHP command execution potentially using
1 findings selected
160 : 62
bWAPP - 3/4/23 11:14AM Findings are filtered

T...	Sev...	Classifi...	Vulnerability Type	Directory	File	Line	Calling Method	A
	High	Definitive	Injection.OS	C:\NileshTest\bWAPP	commandi_blind.php	160	C:\NileshTest\bWAPP\commandi_blind.php	PH
	High	Definitive	Injection.OS	C:\NileshTest\bWAPP	commandi_blind.php	149	C:\NileshTest\bWAPP\commandi_blind.php	PH

xss_stored_3.php | bof_1.php | commandi_blind.php | directory_traversal_1.php

```
                // Debugging
                // echo "Windows!";

                // Increasing the PING count will slow down your web scanner!
                shell_exec("ping -n 1 " . commandi($target));

            }

            else
            {

                // Debugging
                // echo "Not Windows!";

                // Increasing the PING count will slow down your web scanner!
                shell_exec("ping -c 1 " . commandi($target));

            }

        }

    }

    ?>
```

This code's vulnerability lies in using the shell_exec function to execute the ping command without proper input validation or sanitization. This can be exploited by an attacker who can control the value of the $target variable, which is passed as an argument to the ping command.

An attacker can manipulate this variable to inject arbitrary commands and execute them on the target system with the privileges of the PHP script user. For example, an attacker can use command injection to execute a reverse shell or to delete files from the target system (where the application is running).

Recommendation: To mitigate this vulnerability, the input to the $target variable should be validated and sanitized before passing to the shell_exec() to prevent any injection of malicious commands.

One way to do this is by using a whitelist of allowed characters or a regular expression to validate the input. Additionally, it's recommended to use PHP's escapeshellarg function to escape any special characters before passing the argument to the ping command. Avoid using command line calls. Disable executable os command APIs through php.ini disable_functions entry.

Conclusion

Conducting a source code review project can seem like a daunting task, especially for those who are new to it. However, with proper planning, execution, and a thorough understanding of the code being reviewed, the process can be completed successfully. By following the advice outlined in this guide, consultants and security professionals can confidently approach source code review projects, knowing they have the tools and strategies needed to complete the task effectively.

Remember always to prioritize security, accuracy, and thoroughness when reviewing code, and don't be afraid to ask for help or advice from colleagues or industry experts when needed. With these tips in mind, you can successfully conduct a source code review project and contribute to your organization's software development efforts.

References:

- OWASP Code Review Guide
- Source Code Analysis Tools

Back to Blog

About Nilesh Sapariya

Nilesh is a highly experienced Penetration Tester with over 10 years of experience in the field. He specializes in testing web applications, mobile applications, cloud infrastructure, thick client systems, conducting source code reviews, and assessing APIs. He has a proven track record of identifying and reporting on zero-day vulnerabilities. Nilesh has successfully led, executed, and managed numerous security assessment projects, from initial planning and testing to final reporting and client deliverables. In his free time, Nilesh participates in bug bounties to stay up to date with the latest vulnerabilities.

MORE BY NILESH SAPARIYA →