Identify and avoid false positives with IBM AppScan

Minimize significant development time and effort when searching for false positives

Akash Shetty
Mohammed Yacoob
Shrivathsa H Bhat

August 31, 2016

IBM Security AppScan® is a tool that performs dynamic security scanning of web applications and services to identify the security vulnerabilities that are present in applications. Along with valid vulnerabilities, an automated scanning tool can also report vulnerabilities that turn out to be invalid upon further manual analysis. These "vulnerabilities" are commonly known as false positives. In this article, we discuss some common false positives reported by AppScan and provide guidance on how a tester can validate whether the reported issue is a false positive or not. Additionally, the article explains how to avoid such false positives from being reported by making the proper configuration changes to the AppScan tool.

IBM Security AppScan® is an automated web application security assessment tool that identifies prominent security vulnerabilities, including OWASP Top Ten and SANS 25 vulnerabilities. The tool also provides detailed reports on security issues along with advisory and fix recommendations.

A typical automated application security scanner will have a list of policies that comprise different test scenarios, which are running against the target to get the security posture of the application.

For each specific vulnerability type, an expected set of tests are run and the responses are analyzed by the tool. Based on the rules and patterns that are used for this analysis, a vulnerability is reported.

“In this article, we discuss some common false positives reported by AppScan and provide guidance on how a tester can validate whether a reported issue is a false positive and how to evade it being reported if it is.”

The rules and patterns that are used by scanners to analyze the results of test scenarios not only catch actual issues but also sometimes return false positives. False positives can occur
because of various factors, such as misconfigured scans, network instability, or issues in the scan environment, target application design, or target technology. Also, depending on the nature of the application, the received responses might trick the tool into believing that there is a vulnerability when actually none exists.

Overview

This article discusses the commonly found false positives reported by AppScan after dynamic scans are run against several different applications and technologies. For each vulnerability type, the article addresses the following three questions:

- Why does AppScan report false positives?
- How can I verify false positives?
- How can I evade false positives in the scan results in the future?

Typical false positive issues by AppScan

In this section, we discuss the types of false positives that can occur:

1. Authentication bypass by using verb tampering
2. Authentication bypass by using SQL injection
3. Blind SQL injection and MongoDB NoSQL injection
4. Link injection and phishing through frames
5. Alternative version of file detected
6. Archive file download
7. Compressed directory found
8. Temporary directory found
9. Temporary file found

1. Authentication bypass by using verb tampering

Why AppScan reports this issue

This false positive can result from a misconfigured login scenario and target request sets for the scan. For example, if a URL can be accessed without logging in; contains no sensitive data, such as a help page URL; if it is recorded in the Manual Explorer tool; and is included in the scan, then in the test phase AppScan changes the method to "BOGUS" and sends the same request. Some web applications consider any requests to the server as GET requests by default when no authentication is needed, and they respond with the same content for all methods. In such a case, AppScan analyzes the similarity between the original and test responses and reports an issue.

How to verify the false positive

Check whether the test response given by AppScan has a valid response that contains sensitive information and whether it needs an application session. Check whether the reported URLs are supposed to be accessed without a login.
Figure 1. Reported URLs that can be accessed without logging in

How to evade this false positive in the results

Below are some tips that can be used to evade such false positives:

- Capture only those requests for current privileges in AppScan.
- Do not scan session pages and public pages together. We recommend that you initiate different scans for such pages.
- If the application has one user privilege, then use one scan for those requests that are only accessed after a successful login and another scan with no authentication.

2. Authentication bypass using SQL injection

Why AppScan reports this issue

Often, this issue is a false positive if the recorded login sequence is incorrect. For testing this issue, AppScan compares the original recorded login response to the login response from the current test scenario. If the originally recorded login itself is incorrect, then both the original and the current test response will be the same (that is, they will be failed responses). AppScan considers this as a successful test for bypass of authentication and reports an issue.

How to verify the false positive

Check whether the test response has a successful login to the application. If it is a false positive, then the response usually will be a generic error page or a redirect to a login page or to any other non-sensitive page.

How to evade this false positive in the results

- Record the login sequence properly.
• Validate the recorded login and the "in-session detection" before the scan to ensure that the
login sequence is correct.
• When the login scenario involves multiple requests (such as SAML or OAuth), record the
complete scenario and carefully examine the in-session detection in the endpoints of the
target domain (not of the domain that provides SAML or OAuth). Below is the "in-session
detection" example for the console.stage1.ng.bluemix.net target.

Figure 2. Example of "in-session detection"

3. Blind SQL injection (BSQLi) and MongoDB NoSQL injection

Why AppScan reports this issue

The idea behind Blind SQL injection is to inject special characters and functions (which can be
used in an SQL query) into user input and observe the output for any strange behavior, such as
an error response, delay in response. AppScan tries to send some predefined sets of Blind SQL
payloads to a single target request and examines all responses from these requests for any kind
of change. This compared percentage can be set in Configuration > Advanced Configuration
> Test: Difference Threshold > Increase (in %). If there is any change in the comparison of
responses or any error response, then AppScan reports an issue.

How to verify the false positive

Often, right-clicking on the issue and retesting will remove the issue if it is a false positive. If BSQLi
or NoSQLi vulnerabilities show up, always right-click and retest them individually after the end of
the scan (when the load is low) to validate that the site returns the same response under optimal
conditions.

In other cases, the false positive can be verified by using test request-response that is given by
AppScan. Observe the test requests and responses for any deviations from the original request.
If there is any change in the test responses, then try to make the same request again (by using a
suitable tool such as Burp Suite), and make sure that you have a valid session for the request. If
the new response received is not the same as the one given by AppScan, then it must be a false
positive because of incorrect application behavior at the time of scan.
How to evade this false positive in the results

A trick to evade false positives in this case is to make the AppScan request valid requests all the time and analyze valid responses for issues. Below are some points to consider:

- If the site uses a login, always record the login and enable proper in-session detection so that AppScan won't come across any invalid session responses while comparing for Blind SQL injection.
- Scan at a time when network activity is low to reduce network congestion as a factor.
- If the site has other state-sensitive pages (such as surveys, check outs, and registrations), always record these in a multistep operation to ensure the state when they are tested.
- If the target application has large dynamic values (for example, session ID, SSO tokens, and view state), in **Configuration > Advanced Configuration > Test: Difference Threshold**, increase the threshold percentage by 15–20%.
- On target servers with lower hardware or resources, always reduce the number of threads to 1; this will minimize the scan impact on the application server and reduce inconsistent responses due to instability.

To illustrate a false BSQLi, we analyze a situation where such a result is caused by a heavy load on the server. We show three screenshots of the analysis, which are illustrated in Figures 3, 4, and 5.

Figures 3 and 4 show scans in which numerous tests, in addition to Blind SQL injection, are enabled in the Test policy. Many requests are sent to the server from AppScan in a short span of time, causing the load on the server to be high. This heavy load results in the server behaving erratically and responding incorrectly at times, which in turn makes AppScan falsely assume that there is a Blind SQL injection.

**Figure 3. Screenshot of numerous tests and heavy server load**
Figure 4 shows one such response from the server, where an incorrect 400 Bad result is returned by AppScan.

Figure 4. Screenshot of an incorrect 400 Bad result that is caused by heavy server load

Figure 5 shows the result when the scan is run at a time when there is no load on the server. You can see that, in this case, a Blind SQL injection is not reported. For this scan, we enabled only a Blind QSI Injection test in the Test policy. Therefore, the number of HTTP requests sent to the server were relatively few, causing the load on the server to be light.

Figure 5. Screenshot of a scan result under light server load
4. Link injection and phishing through frames

Why AppScan reports this issue

Link injection and phishing through frames are security issues that allow attackers to inject malicious links and frames into target applications. To test this, AppScan tries to inject anchor and frame tags into the target application. Usually, these two issues will be identified wherever an XSS issue exists (with `<script>` tag injection).

How to verify the false positive

This is not a false positive, but it is an unnecessary response. If the vulnerable parameters that are identified in the issue are the same as the parameters identified in XSS issues from the same application, then ignore such issues since a fix for XSS (HTML/URL encoding user inputs in response) will automatically fix these two issues also.

5. Alternative version of file detected

Why AppScan reports this issue

For this scenario, AppScan tries to get any backup or old version of a file from the target application by altering all explored URLs. For example, if the application has a file like http://test.com/test.html, then AppScan tries to fetch a file such as:

- http://test.com/_test.html

It then analyzes the response for 200 OK.

If the application is configured to respond with 200 OK for all pages (including 4xx and 5xx error pages) with generic content, then all test scenarios that were conducted by AppScan will return with a 200 OK response, and a false positive will be reported.

How to verify the false positive

Load the test URL given by AppScan in a browser with the proper session. Verify whether it is an alternative version of any application file or a generic error page.

How to evade this false positive in the results

The reason for a false positive in this issue is the inability to identify the generic error page of a target application by AppScan (because error pages are application-specific and AppScan can't generalize it). This false positive can be evaded by configuring generic error pages in AppScan before the scan.

Here's a sample comparison of results with and without configuring the generic error page. Configuring the generic error page in AppScan gives the best results for most of the test scenarios, but here in this sample I selected five issues that occur in any list of common false positives:
• Alternative versions of file
• Archive file download
• Compressed directory found
• Temporary directory found
• Temporary file found

The sample app here has a generic error page and it responds with 200 OK for all error pages.

Figure 6. Sample app

Figure 7. Generic error page

The following figure is the AppScan result that was configured to scan for the five issues that are listed above.
Figure 8. AppScan result when configured to scan for five common issues

The AppScan reports all five issues because of 200 OK in the response for test scenario.

The generic error page of the sample application has Apache/2.2.17 (Win32) in all of the error pages. By configuring this in Configuration > Error Pages, AppScan identifies these error pages and does not report these false positives in the results.
6. Archive file download

Why AppScan reports this issue

Web servers usually associate Common Gateway Interface (CGI) file name extensions, such as .pl, with a handler, such as Perl. When a URL path ends with .pl, the file name that is designated in the path is sent to Perl for execution; the file contents are not returned to the browser. However, when the script files are archived, an ARC file might be created. The web server usually does not have a specific handler for this extension. If the attacker requests this file, the file contents are sent directly to the browser. For this scenario, AppScan tries to get archive files from the target application by altering all explored URLs. For example, if the application has a file like http://test.com/test.html, then AppScan tries to fetch files, such as:

- http://test.com/test.html.arc
- http://test.com/test.html.ARC

It then analyzes the response for 200 OK.

If an application is configured to respond with 200 OK for all pages (including 4xx and 5xx error pages) with generic content, then all test scenarios that were conducted by AppScan will return with a 200 OK response, and a false positive will be reported.

How to verify the false positive

Load the test URL given by AppScan in a browser with the proper session. Verify whether it is an archive file or a generic error page.
**How to evade this false positive in the results**

This can be evaded by setting a generic error page. See the instructions above for *Alternative version of file detected*.

### 7. Compressed directory found

**Why AppScan reports this issue**

For this scenario, AppScan tries to get compressed files from the target application in each identified directory by altering all explored URLs. For example, if the application has a file like http://test.com/testfolder/test.html, then AppScan tries to fetch files, such as:

- http://test.com/testfolder.zip
- http://test.com/testfolder.war

It then analyzes the response for **200 OK**.

If the application is configured to respond with **200 OK** for all pages (including 4xx and 5xx error pages) with generic content, then all test scenarios that were conducted by AppScan will return with a **200 OK** response, and a false positive will be reported.

**How to verify the false positive**

Load the test URL given by AppScan in a browser with the proper session. Verify whether it is a compressed file or a generic error page.

**How to evade this false positive in the results**

This can be evaded by setting a generic error page. See the instructions above for *Alternative version of file detected*.

### 8. Temporary directory found

**Why AppScan reports this issue**

For this scenario, AppScan tries to get temporary folders from the target application in each identified directory by altering all explored URLs. Temporary directories can be a copy of an existing folder or a configuration folder.

For example, if an application has a file like http://test.com/test/testfolder/test.html, then AppScan tries to fetch files, such as:

- http://test.com/test/Copy of testfolder/
- http://test.com/test/testfolder(copy)/

It then analyzes the response for **200 OK**.
If the application is configured to respond with 200 OK for all pages (including 4xx and 5xx error pages) with generic content, then all test scenarios that were conducted by AppScan will return with a 200 OK response, and a false positive will be reported.

**How to verify the false positive**

Load the test URL given by AppScan into a browser with the proper session. Verify whether it is a real folder or a generic error page.

**How to evade this false positive in the results**

This false positive can be evaded by setting a generic error page. See the instructions above for [Alternative version of file detected](#).

### 9. Temporary file found

**Why AppScan reports this issue**

Web servers usually associate CGI file name extensions, such as .pl, with a handler, such as Perl. When a URL path ends with .pl, the file name that is designated in the path is sent to Perl for execution; the file contents are not returned to the browser. However, when the script files are edited in place, the editor might save a backup copy of the edited script with a new file extension, such as .bak, .sav, and .old files. The web server usually does not have a specific handler for these extensions. If the attacker requests one of these files, the file contents are sent directly to the browser.

For this scenario, AppScan tries to get temporary files from the target application in each identified endpoint by altering all explored URLs.

For example, if the application has a file like http://test.com/test/testfolder/test.html, then AppScan tries to fetch files, such as:


It then analyzes the response for 200 OK.

If the application is configured to respond with 200 OK for all pages (including 4xx and 5xx error pages) with generic content, then all test scenarios that were conducted by AppScan will return with a 200 OK response and a false positive will be reported.

**How to verify the false positive**

Load the test URL given by AppScan in a browser with the proper session. Verify whether it is a real file or a generic error page.

**How to evade this false positive in results**

This can be evaded by setting a generic error page. See the instructions above for [Alternative version of file detected](#).
Conclusion

False positives and "noise" are annoying and, if not identified early, can consume considerable development time and effort. AppScan has many configuration options that a tester can explore to ensure a successful scan with the target application and environment. With the help of this article, AppScan users can significantly reduce the number of false positives reported.
Related topics

- IBM Security AppScan
- OWASP Top Ten
- SANS 25
- Burp Suite

© Copyright IBM Corporation 2016
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)