Find vulnerabilities specific to session management

An interactive approach to verify vulnerabilities with IBM Security AppScan

Prabhangshu Adhyapak

January 26, 2017

Vulnerabilities that are specific to session management are great threats to any web application and are also among the most challenging to find and fix. Sessions are targets for malicious users because they can be used to gain access to a system without having to authenticate.

Vulnerabilities that are specific to session management are great threats to any web application and are also among the most challenging to find and fix. Sessions are targets for malicious users because they can be used to gain access to a system without having to authenticate.

This tutorial describes session management concepts and how you can use IBM Security AppScan® Standard to effectively find vulnerabilities specific to session management.

This tutorial also clearly describes methods to validate session-related vulnerabilities that are detected in IBM Security AppScan Standard, which eliminates false positives from the findings. We also describe a few of the concepts that are related to sessions and potential risks that are related to them by using an interactive approach.

Session management concepts

Session management scenario

First of all, what is a web session? To help you visualize, consider this common scenario where you are making a purchase online.

- As a registered customer, you log in once to the shopping portal with your valid login credentials to successfully purchase items.
- The application would not expect the user to provide credentials again and again unless the logged-in session times out or there is an execution of a privileged action (such as admin access) while it performs all of these activities. It would not be possible for a user to provide credentials with every click inside the application to authenticate his actions.
So how does the application recognize that the legitimate user is making the intended requests after that first session? Session management allows the application to involve the users in authenticating once and confirm that the user is executing a specific action is the user who provided the original credentials.

**Tracking a user**

The most common method of tracking a customer through a website is by assigning a unique session ID and having this information conveyed back to the web server with every request. Once a client successfully authenticates on the web application, the generated session ID can be used as a stored authentication token so that the client does not need to retype his/her login information with each page request.

Though there are several methods to both allocate and receive session ID information, the most commonly used method is cookies. Session information is stored on the server and the client is made to store the Session Identifier for its information. The session ID is stored by using a cookie (a small piece of data that is sent from a website and stored on the user's computer by the user's web browser while the user is browsing) and the server then uses the session ID to retrieve the information that it stored.

As session IDs are used to uniquely identify and track a web application user, an attacker is potentially able to submit this same information and impersonate someone else. This class of attack is commonly referred to as Session Hijacking. Because of the stateless nature of HTTP (or HTTPS) protocol, the attacker can very easily portray himself as a legitimate user using the hijacked session ID.

**Common failings**

While web-based session management is important for tracking users and their navigation throughout an application, its most serious use is to maintain the state information of an authenticated user as the user performs allowed functions. Some examples of state information include the identity of the current user, the contents of a shopping cart, and a database connection string. For online banking and retail environments, the use of an appropriately strong session management method is crucial to the success of the organization.

Based on investigations, some of the most common failings are:

- Predictable session IDs
- Insecure transmission (such as not using HTTPS)
- Length of session validity
- Session verification

Based on the common findings from IBM Security AppScan Standard, a few vulnerabilities that are related to these four listed failings were identified. In this tutorial, we go over the vulnerabilities and provide steps on how to verify them.
Configuring IBM Security AppScan Standard for session management

In this section, we describe how to configure IBM Security AppScan Standard to effectively find vulnerabilities that are specific to session management.

IBM Security AppScan Standard has features for effective session management; one such key feature is the Recorded Login option that is available in the 'Login Management configuration.

AppScan Standard provides an option to record the login sequence of the target web application, which can be done with the following steps:

1. Navigate to **Scan Configuration > Login Management**. Open the **Login/Logout** tab and select **Recorded**.
2. You can use the AppScan browser to record the login sequence. (Other browsers can also be used for recording the login sequence, if proper proxy settings are in place.)

**Figure 1: Scan configuration window**

3. The AppScan browser opens the login page of the target application, where you will log in with your credentials.
4. After you log in to the application, click "I am logged in to the site."

The tool will now try to analyze the login data and establish a Login Sequence to identify an "In-Session Detection Pattern" that occurs on the in-session page. During the scan, AppScan sends the in-session request repeatedly and checks that the response contains the In-Session Detection Pattern.
If AppScan does not find the pattern in the page's response, it assumes that it was logged out and attempts to log in again by replaying the login sequence to keep checking whether the tool is in-session during the scan. This method improves the Session Management efficiency. I recommend that you record a Login Sequence for improving the output of the scan in terms of the session.

For further understanding on this sequence, please refer to the document IBM Security AppScan Enterprise Edition – Scan configuration best practices.

The following sections cover all methods that you can use to verify session-related vulnerabilities.

- Session identifier not updated
- Session not invalidated after logout
- Inadequate account lockout
- Missing secure attribute in encrypted session (SSL) cookie
- MISSING HTTP-only attribute in session cookie

**Session identifier not updated: Medium severity**

Once a session ID is assigned to a user, it is normal for the SessionID to either invalidate or update after the active session ends. If you fail to invalidate an existing session identifier when you authenticate a user, or otherwise establish a new user session, attackers have the opportunity to steal authenticated sessions. These hijacked session IDs can be used to impersonate legitimate users. An attacker is also able to force a known session identifier on a user so that once the user authenticates, the attacker has access to the authenticated session.

**Verification steps**

1. When AppScan runs a scan, it automatically finds vulnerabilities. Look in your Issue List for detected vulnerabilities. Note the issue(s) and check the Request/Response tab.
2. From the Request/Response tab, we conclude that:

• During a Login (POST) request initiated by the client from the browser, a session cookie ASP.NET_SessionId is set by the server. The value is \( \text{ASP.NET_SessionId=pz4p3qu5mfbzij3z0uavta55} \) as seen in the following screenshot below.

**Figure 2.1: Request/Response when request is initiated**

- The request was redirected as it can be seen from the above screenshot in Figure 2.1.
- Finally, when the user logs in to his account, we assume that the session cookie value changed. But if we note the value of \( \text{ASP.NET_SessionId} \) after the login, it remains the same; this is an example of session mismanagement.
3. Another way that we can verify session management is through a browser add-on, Advanced Cookie Manager, which you can install in the browser. For this tutorial, we use Firefox and Advanced Cookie Manager, but similar add-ons are available for other browsers.

**Figure 2.3: Advanced Cookie Manager browser add-on**

4. Open the Test/Target application.
5. Open the Advanced Cookie Manager from the toolbar.
6. We can find the necessary application domain included in the list of domains, circled number 1 in the following image.
7. The cookies that are associated with this domain are in the Cookies Section, which is circled number 2.
8. Now we select a session cookie in order to check whether the session is updated or not. Select ASP.NET_SessionId (it is a session cookie as the isSession Flag is On (circle number 3). Note the Value (circle number 4).
9. Most likely, the value of this session cookie changes post-login. To confirm, follow these steps:
   • Ensure that you are logged in to the target application.
   • Click the refresh button in the Advanced Cookie Manager after you sign in (see the green square in Figure 2.5 below).
Note that the value of ASP.NET_SessionId remains the same after logon. Moreover, the value of this session cookie remains the same even after we log out. (This can be checked by using the same process after you log out). Therefore, we can conclude that there is no update in the value of the session ID. Unfortunately, if an attacker steals this session ID information, it is an easy enough exercise to hijack and manipulate another user's active session.

**Session not invalidated after logout: High severity**

This vulnerability exists in applications where the session IDs are not invalidated after a user logs out. As discussed previously, the session IDs will either invalidate/expire after logout or they will update after logout (the value of the session ID changes post-logout).

In certain cases where the session ID remains the same or valid post-logout, an attacker can easily access the account by using that session ID.

**Verification steps**

1. Mark the issue in the Issue List of AppScan and check the Request/Response.

   ![AppScan issue list with vulnerability](image)

2. To verify this vulnerability:
   a. Ensure that the user is logged out of the application.
b. Click the **Back** button in the browser. If the application does not prompt you to log in again and lands in the application, it means that the session did not expire and a user can still log in by using the previous session.

3. You can further verify the same issue by checking the session cookies and whether they are expired or updated to ensure that the previous session cannot be used again.

In some cases, if a cached version of the page is displayed, clicking the **Back** button means that you might need to refresh the page from the server. If the logout function causes session cookies to be set to a new value, the old value of the session cookies needs to be restored and a page will reload from the authenticated area of the application. If these tests don't show any vulnerabilities on a page, at least some further pages of the application that are considered as security-critical will test to ensure that session termination is recognized properly by these areas of the application.

If the server fails to invalidate the session identifiers, it might be possible to steal or manipulate customer sessions and cookies, which might be used to impersonate a legitimate user. This vulnerability allows the hacker to view or alter user records and to perform transactions as the original user.

**Inadequate account lockout: Medium severity**

In this case of inadequate account lockout, consider this common scenario: A hacker wants to log in to another user's account in a social networking site where he already knows the user name of the targeted user. He doesn't know the password, but based on the password policy that was used in the application he can try to guess the password.

This vulnerability could lead to an exploit by an attack that is usually known as brute force attacking. A brute force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until the attacker discovers the one correct combination that works.

Hackers launch brute force attacks by using widely available tools, like Burp Suite (Intruder) that utilize word lists and smart-rule sets to intelligently and automatically guess passwords. Although such attacks are easy to detect, they are not so easy to prevent. For example, many HTTP brute force tools can transmit requests through a list of open proxy servers. Since each request appears to come from a different IP address, these attacks cannot be blocked by simply blocking the IP address. To further complicate things, some tools try a different username and password on each attempt, so locking out a single account for failed password attempts is even not possible. Though there are several measures that can avoid similar future attacks.

A security best practice suggests that an account will get locked out after 3-5 invalid attempts and the application will display a proper error message ("Account has been locked") to prevent brute force attacks. The application that fails to employ this policy properly can fall prey to this brute force attack.

**Verification steps**

1. Mark the issue in the Issue List of AppScan and check the Request/Response.
Figure 4: AppScan issue list with vulnerability "Inadequate Account Lockout"

2. Check the Reasoning column on the right side of the Request/Response tab.

Figure 4.1: Reasoning from Request/Response tab

- The reasoning that is provided in the image above says the tool that was initially logged in to the account with a set of valid credentials. Beyond that, the tool tried to log in by using an invalid password several times (more than 3-5 attempts). Finally, the tool tried to log in once again with the valid password. But there was no message that stated the account was locked out, which shows that brute force attempts might be successful in this case.
- Read the Request/Response fully in order to check the difference between the successful and unsuccessful attempts and the pattern of the response in each case.

The same process can also be manually checked. You can try providing invalid passwords a number of times to check out the account lockout mechanism, provided it does not require a significant amount of work to unlock the accounts to allow users to attempt to authenticate again.

Missing secure attribute in encrypted session (SSL) cookie: Medium severity

Though the verification for this issue involves very little effort, it is highly effective at minimizing attacks. If an attacker is able to grab an authentication cookie, he can very easily impersonate the legitimate user.

Let's assume that HTTP is used as the protocol for communication and that the cookie is sent in plain text, which is ideal for the attacker who is eavesdropping on the communication channel between the browser and the server. The attacker can grab the cookie and impersonate the user. Now let's assume that HTTPS is used instead of HTTP. HTTPS provides encryption and the attacker will not see the cookie. The conclusion is to send the authentication cookie over a secure channel so that hackers cannot eavesdrop.

A common question is why do we need a secure flag if we use HTTPS that already provides an encryption channel? This section covers this question on adding a Secure Attribute in an encrypted session cookie.

In this instance, this scenario is typical: A site is available over both HTTP and HTTPS. An attacker is in the middle of the communication channel between the browser and the server. The cookie that is sent over HTTPS cannot be eavesdropped. However, the attacker can take advantage
of the fact that the site is also available over HTTP. The attacker can send the link of the HTTP version of the site to the user. The user clicks the link and the HTTP request is generated. Since HTTP traffic is sent in plain text, the attacker eavesdrops on the communication channel and reads the authentication cookie of the user.

So how can we avoid this vulnerability? Can we allow this cookie to be sent only over HTTPS? It turns out that it is possible, and a secure flag is used exactly for this purpose; the cookie with a secure flag will only be sent over an HTTPS connection.

**Verification steps:**

1. Mark the issue in the Issue List of AppScan and check the Request/Response.
   
   **Figure 5: AppScan issue list with vulnerability "Missing Secure attribute in Encrypted Session Cookie"**

2. Use the Advanced Cookie Manager to verify the vulnerability.
   - Open the target application (in this case, http://demo.testfire.net/)
   - Open the Advanced Cookie Manager from the Toolbar
   
   **Figure 5.1: Advanced Cookie Manager showing Secure Flag for the cookie**

3. Here, we are considering the cookie `ASP.NET_SessionId` as shown in Figure 5.1. The flag/attribute `isSecure` is shown as False. The secure flag is not associated with this cookie, which might lead to further exploits by attackers eavesdropping in the network.

**Missing HttpOnly attribute in session cookie: Low severity**

The concept behind HTTP-only cookies is to train a browser so that a cookie will never be accessible via JavaScript through the `document.cookie` property. To create an HTTP-only cookie,
add an HttpOnly flag to the cookie. Once this flag is set, there is no access via 'document.cookie' to this cookie.

To check whether an HttpOnly flag was added to a session cookie, we can check the Response Header from the AppScan Request/Response tab. The following image is an example of the added cookie attribute.

**Figure 6: Request/Response showing cookie attribute**

![Request/Response Image]

Further verification steps

1. Mark the issue in the Issue List of AppScan and check the Request/Response.
   
   **Figure 6.1: AppScan issue list with vulnerability "Missing HttpOnly Attribute in Session Cookie"

   ![AppScan Issue List](image)

2. You can also use the online tool [http://web-sniffer.net](http://web-sniffer.net) to verify sessions.
   
   **Figure 6.2: Web-Sniffer application interface

   ![Web-Sniffer Interface](image)

3. Click **Submit** to enter the HTTP(S) URL.
4. From the response header in Web-Sniffer, we can see that the HttpOnly flag is not added to the session cookie \texttt{amSessionId} as reported by AppScan in Figure 6.1. Whereas it is added to the other session cookie \texttt{ASP.NET_SessionId} and as such is not reported by AppScan.

5. You can also verify with Advanced Cookie Manager.

**Figure 6.4: Advanced Cookie Manager showing HttpOnly attribute**

6. Additionally, you can also verify with Firebug (Firefox Extension).
   a. Open the Firebug extension with Mozilla Firefox.
   b. Navigate to the **Cookies** tab.
   c. Check whether the HttpOnly flag is added to the cookie that needs to be verified.
   d. Figure 6.5 shows that the HttpOnly attribute is not added to the session cookie \texttt{amSessionId} as reported by AppScan. As such, the HttpOnly field is empty. However, it is added to the other session cookie \texttt{ASP.NET_SessionId} and the field displays HttpOnly.
Find vulnerabilities specific to session management

Besides the common vulnerabilities that are listed in the previous sections, there are still a few more issues that require attention to proper session management. One other major vulnerability is described in this section on password parameters in the query (or session IDs in the URL).

If a session ID is sent in the URL, it would be very easy for an attacker to get the session.

Figure 7 shows an application where the session ID is sent over URL in cleartext. Any hacker who eavesdrops on the network or any other person behind the current user’s system can easily use this ID and impersonate the current user. We recommend that these IDs or sensitive information are sent to the server in the body part of the request and in an encrypted connection (for example, SSL).

Figure 7: Session ID in URL
Conclusion

The identification of session management vulnerabilities requires a thorough understanding of session management concepts. By using IBM Security AppScan Standard with the appropriate configuration, we can detect the vulnerabilities that we covered in this article. However, the security analyst also needs to manually verify them to ensure that they are not false positives.

With the knowledge on how to address vulnerabilities from session IDs, you can ensure that your application has a good security posture from the perspective of session management.

Download a free trial of Security AppScan Standard

© Copyright IBM Corporation 2017
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)