Hello. My name is Brennan Brazeau and I am a member of the Security Systems Ethical Hacking team. In this video, I will be talking about one of the more common weaknesses found in insecure code: Broken Authentication and Session Management.

This vulnerability category is currently ranked #2 on the OWASP Top 10 chart and is very commonly exploited.

What is broken authentication and session management?

These types of weaknesses can allow an attacker to either capture or bypass the authentication methods that are used by a web application.

The most commonly used authentication scheme is the use of a Username and Password that generates a Session ID. The web application should be protecting these values, and should not allow attackers to steal them to impersonate users.

According to a recent IBM X-Force Threat Intelligence report, approximately 23% of all applications tested are vulnerable to broken authentication and session management.

Let's take a look at what a web application does when you log in.

When visiting a website to access your information, you need to log in. To get past this point, you need to provide the correct Username and Password values.

Once you submit this information, a unique value called the Session ID is generated, which is linked to your credentials to keep track of you while you are logged into the application. This value is usually a string of random letters and numbers as seen at the bottom of the slide.

There are a number of ways a web application can fail to protect these resources. The list below categorizes the various types of weaknesses.

Letâ€™s go though each of these and see what they mean.

Unencrypted connections:

If the connection being used between you and the web application is not encrypted, anybody can see the data being transmitted. This means that ALL INFORMATION that you are sending and receiving between you and the site can be intercepted without your knowledge.

If the attacker can capture this data (using a man-in-the-middle attack for example), your username, password, Session ID, or any other sensitive data can be captured by the attacker.

Enabling encryption on requests that contain sensitive data can prevent this information from being intercepted by attackers.

Unencrypted connections: Demo

In this demo, I will demonstrate how an attacker would capture this information through an unencrypted connection. A man-in-the-middle application has been setup to capture the traffic as it is passed from the web browser to the application.

I will now access altoromutal.com and proceed to login.

Note that at no time while accessing the site was I warned that someone may be eavesdropping on my requests.

Now let's look at the captured traffic.

We can see from the requests that were captured that the username, password and Session ID are all visible to the attacker. They can then use these values to login to the web application to impersonate the user.

Predictable Login Credentials :

123456. Password. 12345678. qwerty. abc123. These are some of the top worst passwords that users choose to use. If a web application allows the use of these values as a password, this leaves an opening for attackers to guess the credentials of a user.

Users should not have the ability to set weak passwords or use default setting values. The web application needs to force users to use a strong password policy that does not allow for these types of values.

Predictable Login Credentials: Demo

In this demo, I will demonstrate how an attacker can use this weakness to get access to the application.

I will start to try and guess what username / password combinations will work with the altoromutual.com login.

Let's try to login with admin / 123456. That doesnâ€™t seem to work.

Let's try admin / Password. That doesnâ€™t seem to work either.

Let's try admin / admin. We are now in the administrator interface of altoromutal.com and have full access to its admin features.

Session ID value does not timeout or is not invalidated after logout.

If you are able to click the logout link in a web application, but still have access to it as if you were logged in, the Session ID value has not been invalidated.

This means if an attacker were to get hold of the Session ID value generated by the web application, they can impersonate you to access the application without having to provide login information. The same issue can occur if the Session ID never expires.

Session ID values should be invalidated when a user logs out of the web application or after a pre-determined amount of time to prevent this attack vector.

User authentication credentials are not protected when stored using hashing or encryption:

This is an issue if an attacker can get access to where the passwords are stored. If the values have not been encrypted, all the information will be available in plain-text and easily accessible. An attacker would immediately be able to use this information for further attacks.

To prevent this, stored Username and Password information should be encrypted, along with salting and hashing the values to make it more difficult to extract if the values fall into the wrong hands.

Session IDs are used in the URL.

Let's take a look at the example URL in the slides:

http://bank.com/login.jsp?sessionid=abcde

In our imaginary web application, that URL is created upon successfully login to the site. As the session Id is exposed in the URL, it is easier for an attacker to obtain this value, as it is more readily exposed.

A good example of why this is an issue is if you were to share the bank.com link with a friend. If your friend were to visit that URL, your account information would be displayed, as the web application uses the Session ID value to identify who is accessing the application.

This can be prevented by submitting any sensitive information to the web application via the body of a POST request.

AppScanâ€™s test policy includes a variety of ways to test for broken authentication and Session management. You can use the search dialogue in the test policy window to search for specific tests or variants you may want to test and then enable/disable them as desired.

You can set AppScan Standard to only use these tests using the following instructions:

In the Scan Configuration option, go to the Test Policy section. Select Enabled/Disabled from the dropdown. Unselect all the entries.

Now search for the test we wish to add. We will first search for Predictable Login Credentials.

Select Predictable Login Credentials and remove the text from the search dialog. You will now see the test under the Enabled section.

Repeat these steps for each test you wish to add. We will now add the rest of the tests.

Query parameter in SSL Request, Session Identifier Not Updated, Session Not Invalidated After Logout and Unencrypted Login Request.

All these tests will now be visible under the enabled section.

We will now scan the altoromutual.com test site for these vulnerabilities. We can see that AppScan Standard has found that the altoromutual web applications has the following vulnerabilities:

Predictable Login Credentials
Session Identifier Not Updated
Query parameter in SSL Request

This concludes our coverage of the 2nd OWASP Top 10 category. I hope you found this information useful. I wish you best of luck in writing and maintaining secure software!