Web Application Security

Rajendra Kachhwaha
rajendra1983@gmail.com

August 19, 2015

Outline

Attacking Authentication

1. Bad Passwords
2. Brute-Forcible Login
3. Password Change Functionality
4. Forgotten Password Functionality
5. "Remember Me" Functionality
6. Incomplete Validation of Credentials
7. Non-Unique Usernames
8. Predictable Usernames
9. Predictable Initial Passwords
10. Fail-Open Login Mechanisms
11. Defects in Multistage Login

Attacking Authentication:Incomplete Validation of Credentials:

1. Using an account you control, attempt to log in with variations on your own password: removing the last character, changing the case of a character, and removing any special typographical characters. If any of these attempts is successful, continue experimenting to try and understand what validation is actually occurring.
2. Feed any results back into your automated password guessing attacks, to remove superfluous test cases and improve the chances of success.

Attacking Authentication:Non-Unique Usernames:

1. If self-registration is possible, attempt to register the same username twice with different passwords.
2. If the application blocks the second registration attempt, you can exploit this behavior to enumerate existing usernames even if this is not possible on the main login page or elsewhere. Make multiple registration attempts with a list of common usernames to identify the already registered names that the application blocks.
3. If the registration of duplicate usernames succeeds, attempt to register the same username twice with the same password, and determine the applications behavior.

cont.

Attacking Authentication:Non-Unique Usernames:

4. If an error message results, you can exploit this behavior to carry out a brute-force attack. Target a guessed username, and attempt to register this username multiple times with a list of common passwords. When the application rejects one specific password, you have probably found the existing password for the targeted account.
5. If no error message results, log in using the credentials you specified and see what happens. You may need to register several users, and modify different data held within each account, to understand whether this behavior can be used to gain unauthorized access to other users accounts.

Attacking Authentication:Predictable Usernames:

1. If usernames are generated by the application, try to obtain several usernames in quick succession and determine whether any sequence or pattern can be discerned.
2. If so, extrapolate backwards to obtain a list of possible valid usernames. This can be used as the basis for a brute-force attack against the login and other attacks where valid usernames are required, such as the exploitation of access control flaws.

Attacking Authentication:Predictable Initial Passwords:

1. If passwords are generated by the application, try to obtain several passwords in quick succession and determine whether any sequence or pattern can be discerned.
2. If so, extrapolate the pattern to obtain a list of passwords for other application users.
3. If passwords demonstrate a pattern that can be correlated with usernames, you can try to log in using known or guessed usernames and the corresponding inferred passwords.

Attacking Authentication:Fail-Open Login Mechanisms:

1. Perform a complete, valid login using an account you control. Record every piece of data submitted to the application, and every response received, using your intercepting proxy.
2. Repeat the login process numerous times, modifying pieces of the data submitted in unexpected ways. For example, for each request parameter or cookie sent by the client:
   - Submit an empty string as the value.
   - Remove the name/value pair altogether.
   - Submit very long and very short values.
   - Submit strings instead of numbers and vice versa.
   - Submit the same item multiple times, with the same and different values.

cont.

Attacking Authentication:Fail-Open Login Mechanisms:

3. For each malformed request submitted, review closely the applications response to identify any divergences from the base case.
4. Feed these observations back into framing your test cases. When one modification causes a change in behavior, try to combine this with other changes to push the applications logic to its limits.

Attacking Authentication:Defects in Multistage Login:

1. Perform a complete, valid login using an account you control. Record every piece of data submitted to the application using your intercepting proxy.
2. Identify each distinct stage of the login and the data that is collected at each stage. Determine whether any single piece of information is collected more than once or is ever transmitted back to the client and resubmitted, via a hidden form field, cookie, or preset URL parameter.
3. Repeat the login process numerous times with various requests:
   - Try performing the login steps in a different sequence.
   - Try proceeding directly to any given stage and continuing from there.
   - Try skipping each stage and continuing with the next.

cont

Attacking Authentication:Defects in Multistage Login:

4. If any data is submitted more than once, try submitting a different value at different stages, and see whether the login is still successful. It may be that some of the submissions are superfluous and are not actually processed by the application. It might be that the data is validated at one stage and then trusted subsequently in this instance, try to provide the credentials of one user at one stage, and then switch at the next to actually authenticate as a different user. It might be that the same piece of data is validated at more than one stage, but against different checks in this instance, try to provide (for example) the username and password of one user at the first stage, and the username and PIN number of a different user at the second stage.

cont.

Attacking Authentication:Defects in Multistage Login:

5. Pay close attention to any data being transmitted via the client that was not directly entered by the user. This may be used by the application to store information about the state of the login progress, and may be trusted by the application. For example, if the request for stage three includes the parameter "stage2complete=true" then it may be possible to advance straight to stage three by setting this value. Try to modify the values being submitted and determine whether this enables you to advance or skip stages.

Securing Authentication

Need to counterbalance defense against threats with actual user-friendliness.

1. Using strong Credentials
2. Validate Credentials Properly
3. Prevent Information Leakage
4. Prevent Brute-Force Attacks
5. Prevent Misuse of the Password Change Function
6. Log, Monitor, and Notify

Importance of strong Credentials

1. Our goal is to make it harder for attackers to guess our password.
2. Increase the complexity.
3. Increasing the minimum length required.
4. Increasing the number of possible characters.
5. For example: It would take longer to guess all the 10-element alphanumeric password than it would to guess only the 10-digit password.

Using strong Credentials:

1. Require minimum password length
2. Enforce minimum password complexity (Uppercase, Lowercase, Numbers, Special Characters)
3. Require password uniqueness
4. Password cannot Equal Username
5. Allow Accounts to be disabled
6. Properly Store Password:
   - Don't store in plaintext
   - Don't Encrypt
   - Use a Strong Hash (SHA-256 or SHA-512)
   - Use a Salt
   - Multiple Rounds of Hashing

Validate Credentials Properly:

1. All authentication logic should be closely code-reviewed to identify logic errors such as fail-open conditions.
2. Multistage logins should.
   - Not allows users to enter the same information more than once or change it once the page was been processed.
   - The first task carried out at every stage should be to verify that all prior stages have been correctly completed.
3. To prevent information leakage about which stage of the login failed, the application should always proceed through all stages of the login, even if the user has failed to complete earlier stages correctly, and even if the original username was invalid.
4. Ensure that an attacker is not able to effectively choose his own question in the random varying question process.

Prevent Information Leakage:

1. An attacker should have no means of determining which piece of the various items submitted has caused a problem.
2. A single code component should be responsible for responding to all failed login attempts, with a generic message.

Prevent Brute-Force Attacks:

1. Measures should be taken to prevent attacks via automation such as the use of unpredictable usernames.
2. A more balanced policy, suitable for most security-aware applications, is to suspend accounts for a short period following a small number of failed login attempts, slows down password guessing attacks.
3. The application should never indicate that any specific account has been suspended.
4. Instead, a message advising that accounts are suspended if multiple failures occur and that the user should "try again later."

cont.

Prevent Brute-Force Attacks:

5. Login attempts should be rejected without even checking the credentials, when accounts are suspended to prevent attackers from continuing to attempt to login.
6. An application can specifically protect itself against this kind of attack through the use of CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart") challenges on every page that may be a target for brute-force attacks.

Prevent Misuse of the Password Change Function:

1. Use of password change functions. Periodic password expiration and user controlled changes.
2. Password can only be changed during a session.
3. New password should be entered twice to prevent mistakes, compare the "new password" and "confirm new password" fields.

Log, Monitor, and Notify:

1. All authentication-related events should be logged by the application, including login, logout, password change, password reset, account suspension, and account recovery.
2. Anomalies in authentication events should be processed by the applications real-time alerting and intrusion prevention functionality.
3. Users should be communicated with via out-of-band such as email if changes to their accounts are made.