Web cookies are a kind of data that a website saves to your browser for later use. The cookie has basically two-part **Variable: VALUE, Session: admin**.

In a session puzzle attack, we manipulate the **Session** variable to get an unexpected response.

Session Puzzling enables a wide variety of attacks that includes:

- Authentication Bypass
- User Impersonation
- Privilege Escalation
- Flow Enforcement Bypass
- Content Theft
- Indirect Injections
- Reflections and Manipulations

Purpose of the Session Variable

The Session variable is mainly used to store the roles/privileges of different users.

For example, In WordPress sites, admin is a role where the user will get the full privilege to modify almost anything they want.

While, the author is a role where the user will be only posting and modifying the blogs that he created and he cannot add another user, change the admin's password or add some new plugins to the site.

This is what the session variable does, it stores different roles for different users which are provided via the web server.

The root cause of Session Puzzling

This vulnerability takes place when a web application uses the same session variable for more than one purpose.

Let's imagine an example, a web application has 2 functions/pages, one is the login panel where different users can log in, and the other one forgets passwords where the users can reset their passwords.

Now on the login page when a user logs in the web application sends the session variable as **Session: USER** to the browser. And the password page also takes one input of the username to reset the password.

Now, where is the vulnerability, the vulnerability occurs when the web application uses the same **Session** variable for both login and forget password functions. Below video will clarify your doubts.

Practical Use of Session Puzzling

This vulnerability can be found in any web application that uses a session mechanism, and the scope of the attack is not limited to the examples provided in this blog.

- Authentication bypass using session puzzling
- User impersonation using session puzzling
- Privilege escalation using session puzzling
- Content theft using session puzzling
- Indirect injections, reflections, and manipulations

Authentication Bypass Using Session Puzzling

The authentication can be exploited in many methods while relying on various “Session cookies” like tokens, credentials, etc.

In this attack, authentication mechanisms can be bypassed by modifying those session variables that contain privilege-related values, which are usually stored after a successful authentication process.

This vulnerability behavior can be found in many locations, but the following are some common entry points:

Password recovery page

This is a common page where the developer makes a mistake and populates the session identity values in the initial phase of the password recovery process(like an entry point that requires the username/email in order to send a recovery email or present a recovery question page).

Those entry points might populate the session with identifying values, which can be controlled from the client-side(username, email, etc) or values that are obtained from queries or calculations based on the input values (user ID, identity token, username, etc).

Registration page

Developers often store the input values (which are received during the registration process) to the session variable, before completing the registration process.

The value stored in the session usually includes usernames, emails, and other identifying values like admin = 1 in base64 or using other encodings. Sometimes it might take to send several requests to bypass the authentication.

Contact page

In a contact form, there is a field where the user provides identifying values, such as email, username, numbers.

These values might be temporarily stored in a session variable, for supporting various logical processes.

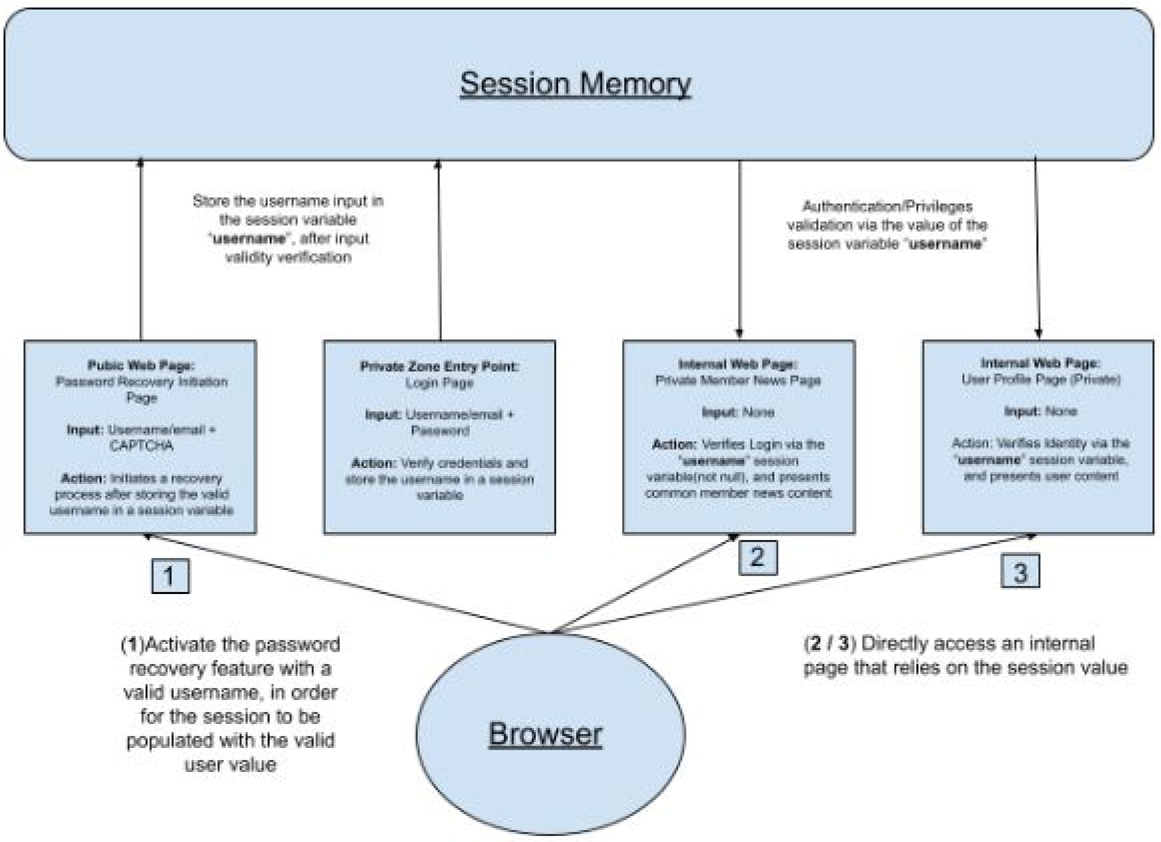
Login entry point

The login entry points are supposed to populate the session value only after a successful authentication attempt.

But in some cases, the programmer might load user originating values into the session prior to the actual validation. This enables attackers to abuse this behavior.

In general, any unauthenticated public entry point that accepts values from external input can be abused to serve as the initial entry point for executing the attack vector.

Here is the practical video of Authentication Bypass using Session Puzzling.


Session Memory
Store the username input in the session variable "**username**", after input validity verification
Authentication/Privileges validation via the value of the session variable "**username**"
Pubic Web Page: Password Recovery Initiation Page
Input: Username/email + CAPTCHA
Action: Initiates a recovery process after storing the valid username in a session variable
Private Zone Entry Point: Login Page
Input: Username/email + Password
Action: Verify credentials and store the username in a session variable
Internal Web Page: Private Member News Page
Input: None
Action: Verifies Login via the "**username**" session variable(not null), and presents common member news content
Internal Web Page: User Profile Page (Private)
Input: None
Action: Verifies Identity via the "**username**" session variable, and presents user content
1
2
3
(**1**)Activate the password recovery feature with a valid username, in order for the session to be populated with the valid user value
(**2 / 3**) Directly access an internal page that relies on the session value
Browser

SAMPLE FLOW FOR AUTHENTICATION BYPASS VIA SESSION PUZZLING

User Impersonation using Session Puzzling

Applications enforce access control on private user resources using a variety of methods, but as a general rule, the validations are performed while relying on the logged-in user identity, or rather, based on the identifying values stored in his session.

If a malicious user will be able to find a way to alter the identifying values in his session, he could, potentially, impersonate other users, view their content, and perform operations on their behalf.

Session puzzling attacks provide a method to do just that.

The difference between "*Authentication Bypass*" and "*User*

***Impersonation”*:**

In an authentication bypass attack sometimes the web application enables some access, even if the provided value doesn’t represent a valid user. But to impersonate a specific user, the attacker must have a method to affect the specific session value populated by sending user input containing the value to a session populating entry point (username, user identifiers, emails, etc), or by sending some values that are interpreted into the actual identify like tokens.

In an authentication bypass attack where the attacker attempts to bypass the authentication enforcement, but in user impersonation, the attacker might desire to impersonate other users as well, and in the latter case, the authentication enforcement does not need to be bypassed.

Because the user impersonation scenario expands the range of access points that can be used to alter the session-stored value, including access points that require authentication (such as profile update-modules, public profile modules, etc.).

As with any session puzzling scenario, the required abnormal behavior (entry points that populate the session variable with user-provided input) can be found in many locations, but here are some common entry points:

- **All the entry points described in the authentication bypass** scenario (registration, password recovery, login).
- **Profile-related entry points:** These kinds of entry points enable viewing, updating, deleting, adding content to other user accounts.

Privilege Escalation Using Session Puzzling

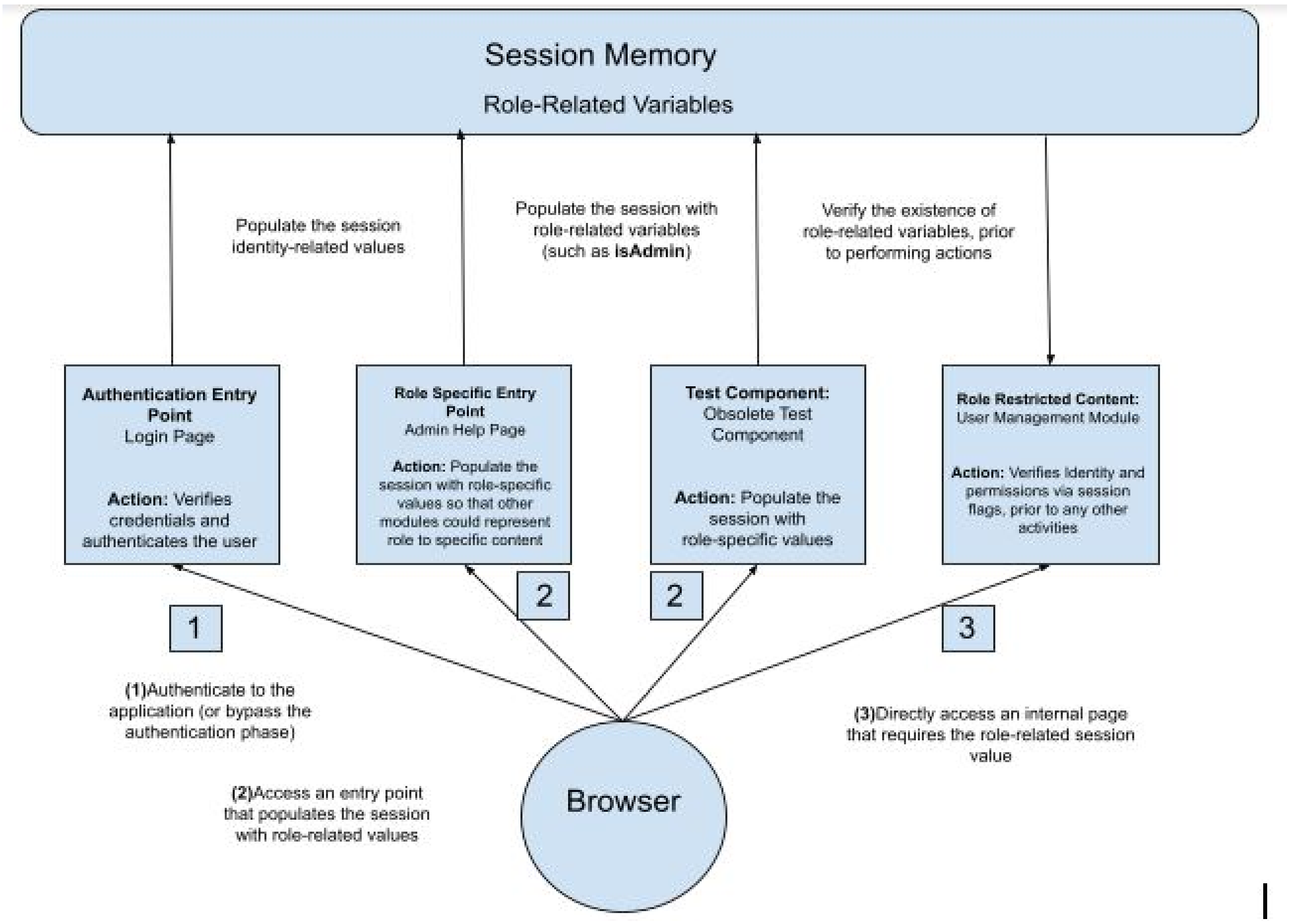
The RBAC (Role-Based Access Control) is a common authorization enforcement model in web applications, where users are associated with different roles, and those roles are granted permissions to perform operations or view information.

For example, admin and author roles and according to those roles, each user has different permission.

The role of the authorization mechanism is to prevent unauthorized users from performing restricted operations or viewing classified data, and this is done by validating the required privilege level of the action/data/entry points in front of the privilege level of the user.

Because the user roles, permissions, and privileges are usually stored in session variables.

So, using a session puzzling attack might be able to modify those values in the session variable, and by this attacker can elevate the privileges and enable him to affect content that was previously inaccessible.


Session Memory
Role-Related Variables
Populate the session identity-related values
Populate the session with role-related variables (such as **isAdmin**)
Verify the existence of role-related variables, prior to performing actions
Authentication Entry Point Login Page
Action: Verifies credentials and authenticates the user
Role Specific Entry Point Admin Help Page
Action: Populate the session with role-specific values so that other modules could represent role to specific content
Test Component: Obsolete Test Component
Action: Populate the session with role-specific values
Role Restricted Content: User Management Module
Action: Verifies Identity and permissions via session flags, prior to any other activities
1
2
2
3
(1) Authenticate to the application (or bypass the authentication phase)
(2) Access an entry point that populates the session with role-related values
(3) Directly access an internal page that requires the role-related session value
Browser

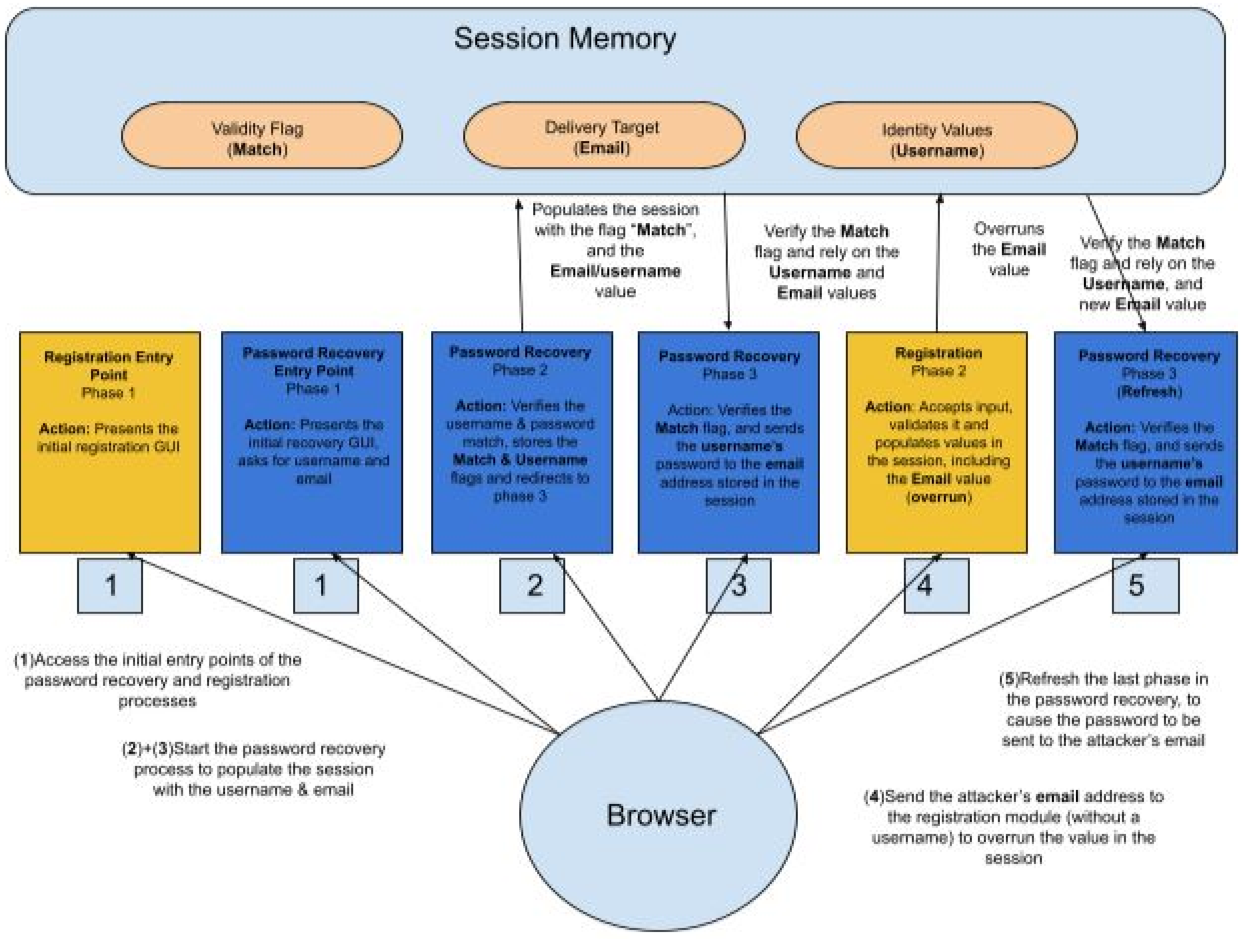
Content Theft Using Session Puzzling

In a web application, there are many functions such as sending password recovery, personal information updates, reminders, refunds, etc.

Session puzzling enables an attacker to modify the target content during delivery and redirect the private user’s content back to the attacker, instead of the user.

In order to accomplish this, the attacker will need to initiate the process during the content delivery by changing the session value with another user's identity.

For example, a password recovery mechanism that populates the session value with the username/email address of the user whose password is being recovered could be manipulated to send the password to another email address by modifying the web request.



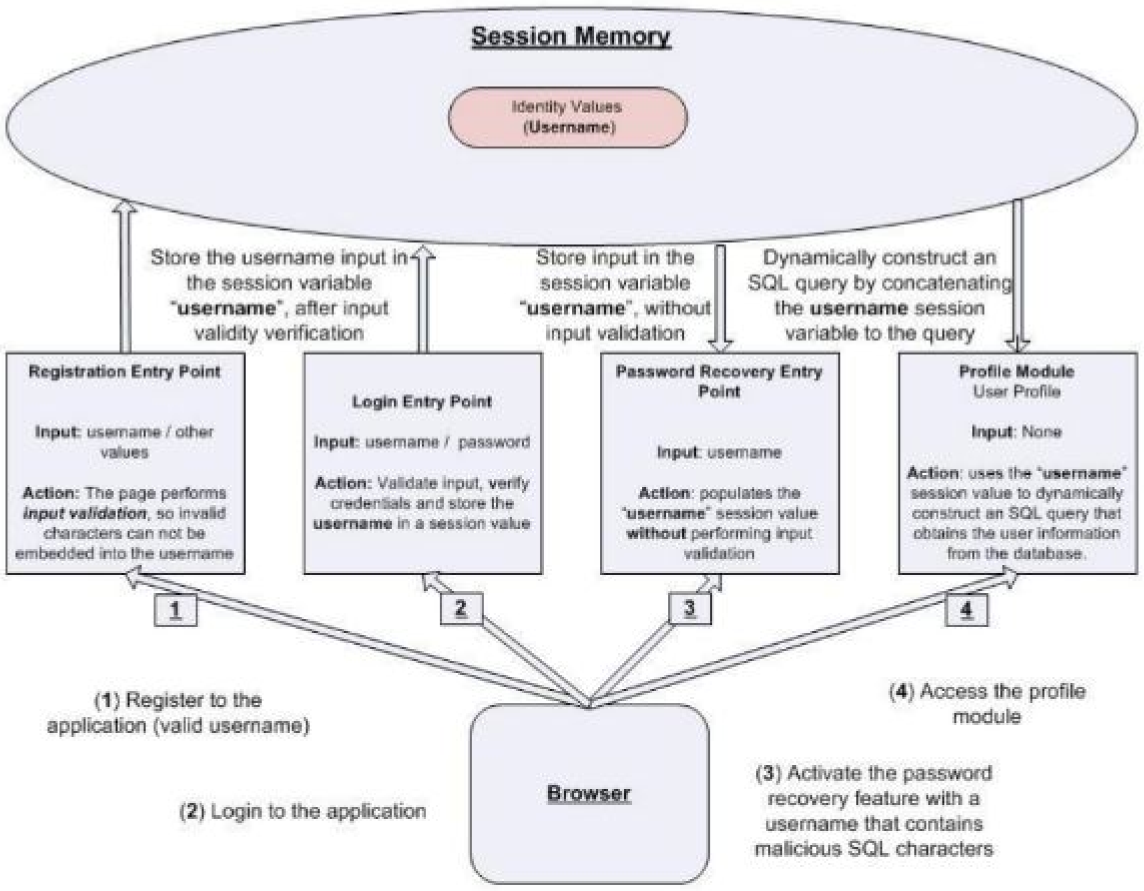
Session Memory
Validity Flag (Match)
Delivery Target (Email)
Identity Values (Username)
Populates the session with the flag "Match", and the Email/username value
Verify the Match flag and rely on the Username and Email values
Overruns the Email value
Verify the Match flag and rely on the Username, and new Email value
Registration Entry Point Phase 1 Action: Presents the initial registration GUI
Password Recovery Entry Point Phase 1 Action: Presents the initial recovery GUI, asks for username and email
Password Recovery Phase 2 Action: Verifies the username & password match, stores the Match & Username flags and redirects to phase 3
Password Recovery Phase 3 Action: Verifies the Match flag, and sends the username's password to the email address stored in the session
Registration Phase 2 Action: Accepts input, validates it and populates values in the session, including the Email value (overrun)
Password Recovery Phase 3 (Refresh) Action: Verifies the Match flag, and sends the username's password to the email address stored in the session
1
1
2
3
4
5
(1)Access the initial entry points of the password recovery and registration processes
(2)+(3)Start the password recovery process to populate the session with the username & email
(5)Refresh the last phase in the password recovery, to cause the password to be sent to the attacker's email
(4)Send the attacker's email address to the registration module (without a username) to overrun the value in the session
Browser

Indirect Injections, Reflections, and Manipulations

Common injection attacks such as SQL Injection, Cross-Site Scripting, and Parameter Tampering could be delivered through a seemingly trusted location, without undergoing bypass and validation, and it also expands the attack surface to the location that was previously unreachable.

For Example, if a module constructs the SQL queries by concatenating session values into query strings, it can't be directly affected by the user input.

But using session puzzling sequences the attacker can populate the session variable used in the query with malicious SQL payloads that will affect the query structure just like any input-based SQL Injection attack.



Session Memory
Identity Values (Username)
Store the username input in the session variable "username", after input validity verification
Store input in the session variable "username", without input validation
Dynamically construct an SQL query by concatenating the username session variable to the query
Registration Entry Point
Input: username / other values
Action: The page performs input validation, so invalid characters can not be embedded into the username
Login Entry Point
Input: username / password
Action: Validate input, verify credentials and store the username in a session value
Password Recovery Entry Point
Input: username
Action: populates the "username" session value without performing input validation
Profile Module
User Profile
Input: None
Action: uses the "username" session value to dynamically construct an SQL query that obtains the user information from the database.
1
2
3
4
(1) Register to the application (valid username)
(2) Login to the application
(3) Activate the password recovery feature with a username that contains malicious SQL characters
(4) Access the profile module
Browser

Final Words

Thanks for your time! I hope, now you know what to do when you encounter any session puzzling attack vector. I will update his blog from time to time. So, if you don't want to miss it, keep a bookmark of this.

If you like this, make sure to share it with others so they can leverage this information. As always, for any doubts or questions, please leave a comment below, or reach me on

Instagram.

Pranjal Singhal

I am a freelancer Cybersecurity researcher and a digital marketer. I have already helped Top IT Giants to secure their web applications and maintain a safe environment for their users. Sharing and Caring is my motive. I love to guide beginners about making a successful career in the cybersecurity industry.