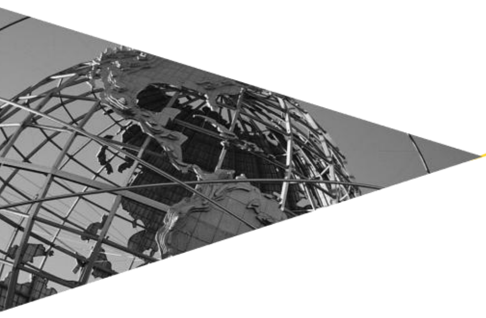


Session Puzzles

Indirect Application Attack Vectors

Shay Chen
Senior Manager, HASC CTO
Hacktics ASC, Ernst & Young
May 23, 2011

Table of Contents

1. Introduction to Session Puzzles

1.1. Background

It began with a live hacking incident, sometime in 2008. When an unidentified attacker managed to corrupt the content of an insurance company web site, we were called to investigate, and found that the attacker managed to gain control over the administrative interface, by using a rather unique attack vector…

The "malicious" attacker used the spider feature of "paros proxy" to automatically crawl the web site twice in a row, and that action alone, allowed him to gain administrative access.

No malicious input, not even a brute force attack, but the results were undeniable, and very similar to a rare and complex vulnerability that we detected in 2007, while auditing the source code of a European bank…

1.2. A Big Fat Claim – New Indirect Application Attack Vectors

Pen-testers often struggle to justify security requirements, secure development best practices and proposed mitigations, by locating vulnerabilities that could be prevented using these mitigations; furthermore, with each mitigation the programmer implements, partial and incomplete as it may be, this task becomes harder and harder.

Mitigations often include input validations, access control mechanisms and other solutions…

But what if there was a simple way to bypass **all** those restrictions?

What if there was an attack vector that could execute a new breed of logical attacks, while enhancing the capabilities of traditional attacks vectors, to the point they could **completely avoid** "traditional" security mechanisms, and deliver malicious payloads from a seemingly trusted location?

. . .

Then the task at hand would probably become much easier… and I believe that in this white paper, lies a method that could do just that.

This document describes a new/underemphasized application-level attack vector (and in a sense, ***a different approach to penetration testing***) that could enable attackers to perform a variety of indirect attacks, in a way that cannot be prevented by using common code level mitigations:

- Bypass efficient authentication enforcement mechanisms, and impersonate legitimate users.
- Elevate the privileges of a malicious user account, in an environment that would otherwise be considered foolproof.
- Skip over qualifying phases in multiphase processes, even if the process includes all the commonly recommended code level restrictions.
- Manipulate server-side values in indirect methods that cannot be predicted or detected.
- Execute traditional attacks in locations that were previously unreachable, or even considered secure.

Sounds exaggerated?

You may think so if you wish… in fact, I could hardly blame you. If I were in your place, I would probably think the same thing… but if these claims got you curious, try and delay your judgment, and keep reading, just a little bit longer…

1.3. Session Puzzles – What's That?

The term ***Session Puzzle*** refers to ***a new vulnerability classification***, which can be detected and exploited using ***a new application-level attack vector*** (or at the very least, ***an underemphasized*** attack vector).

Session Puzzles are exposures that are caused by uncontrolled creation/population of session objects, which are used or relied on by various application entry points.

While exploiting Session Puzzles, the session objects creation can be indirectly initiated, and later exploited, by accessing ***a sequence of entry points*** (web pages, web services, remote procedure calls, etc.) in a certain order.

A successful "construction" of a Session Puzzle enables the attackers to bypass authentication enforcement mechanisms, impersonate legitimate users, elevate privileges, bypass flow restrictions, and even execute additional attacks in locations that were previously considered "safe" (Injections, Parameter manipulations, etc.).

As opposed to traditional attack vectors, in which the attacker either accesses or sends malicious payloads to a single entry point, the attack vector of a session puzzle **requires** the attacker to access two (or more) entry points, using the same session identifier.

1.4. Indirect Session Population Attacks – Why Now?

Although traditional exposures are still common, the mitigation methods for attack vectors used to exploit them are widely known, and in many cases, implemented properly.

Session puzzles can be used to ***bypass*** these security mechanisms in order to exploit "traditional" exposures, as well as for exploiting new types of logical exposures, which are described in the following sections.

1.5. Session Puzzling – How does it Work?

The purpose of a session puzzling attack is to access application entry points (web pages, services, etc.) that populate the session memory with objects and values, in order to "compose" a collection of session objects that enables the attacker to impersonate valid users, bypass security restrictions, and cause unexpected behaviors.

The following schema describes a ***simple*** session puzzle vulnerability that can enable an attacker to impersonate valid users, by accessing a public entry point that stores input in a temporary session variable named "username" (the password recovery page), and then directly accessing internal application pages that rely on the "username" session variable for authentication enforcement / privileges validation:

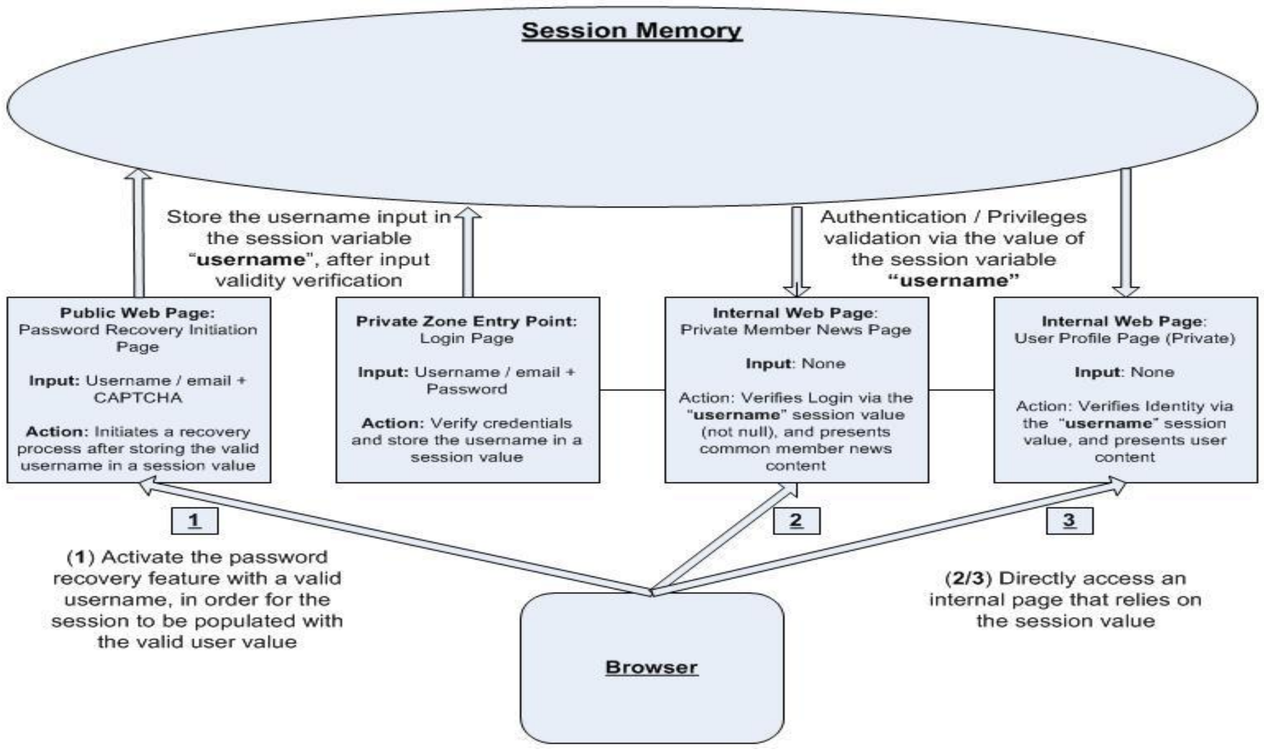

Session Memory
Store the username input in the session variable "username", after input validity verification
Authentication / Privileges validation via the value of the session variable "username"
Public Web Page: Password Recovery Initiation Page
Input: Username / email + CAPTCHA
Action: Initiates a recovery process after storing the valid username in a session value
Private Zone Entry Point: Login Page
Input: Username / email + Password
Action: Verify credentials and store the username in a session value
Internal Web Page: Private Member News Page
Input: None
Action: Verifies Login via the "username" session value (not null), and presents common member news content
Internal Web Page: User Profile Page (Private)
Input: None
Action: Verifies Identity via the "username" session value, 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

Figure 1. – Simple Session Puzzle and Exploitation Scenario

The following sections list a variety of different and complex session puzzles, which require a variety of different detection and exploitation sequences.

1.6. Session Puzzling - Detecting Session Puzzles

Session Puzzles can be detected and exploited using black box methodologies, even though it's **much easier** to detect instances using code reviews.

This document includes several tested "black box" methods for detecting different types of session puzzles, as well as simple guidelines that assist in detecting instances in code reviews.

2. Background – Attack Vectors & Sessions

2.1. Terminology

The following table provides an interpretation for the common terms used in this article, in order to clarify the intention of the author when using these terms:

Term	**Interpretation**
Entry Points / Access Points	Modules in the application that can be accessed or initiated directly from the client side. The list of modules consists of (but not limited to) web pages, web service methods, services, remote procedure calls, filters, individual events, etc.
Attack Vector	The method, process or means used by an attacker to perform an attack or cause a malicious outcome.
Exposure	Security Exposure / Vulnerability.
Entity	An entity that uses the application services or provides services to the application (usually external to the system), including different users, remote servers, and internal servers.
Session Puzzling	Attempting to locate or exploit session puzzle exposures.
Session Puzzles	The application level vulnerabilities described in this document.
Indirect Attack	An attack that affect the system without initially targeting the vulnerable access point.

2.2. Attack Vectors – Something New Under the Sun?

Attackers use various methods to perform malicious operations.

In general, an attacker typically uses one of the following attack vectors to perform an application-level attack:

- ***Manipulate the protocol structure***
- ***Send malicious input to the application*** *(injections, memory attacks, output attacks, parameter manipulations, etc.)*
- ***Directly access restricted resources*** *(forceful browsing, flow bypassing, etc.)*
- ***Flood the application with requests*** *(DoS, ADoS,, DDoS)*
- ***Abuse legitimate features*** *(for spam, privilege escalation, etc.)*
- ***Enumerate sensitive values*** *(credentials, files, identifiers, etc.)*
- ***Gather sensitive information*** *(user data, system data, etc.)*
- ***Redirect users to entry points or access entry points on behalf of users*** *(CSRF, Clickjacking, phishing via redirection, etc.)*

In comparison, the ***session puzzling process*** is performed using a new attack vector (or as mentioned earlier; at the very least, an underemphasized attack vector), and requires a totally different approach then the one used in "traditional" attack vectors.

Session puzzles can be detected and exploited ***by accessing a sequence of entry points,*** prior to ***directly accessing restricted resources*** or ***to executing other attack*** **vectors**. This sequence varies according to the result that the attacker desires, and according to the specific implementation flaws in the vulnerable application.

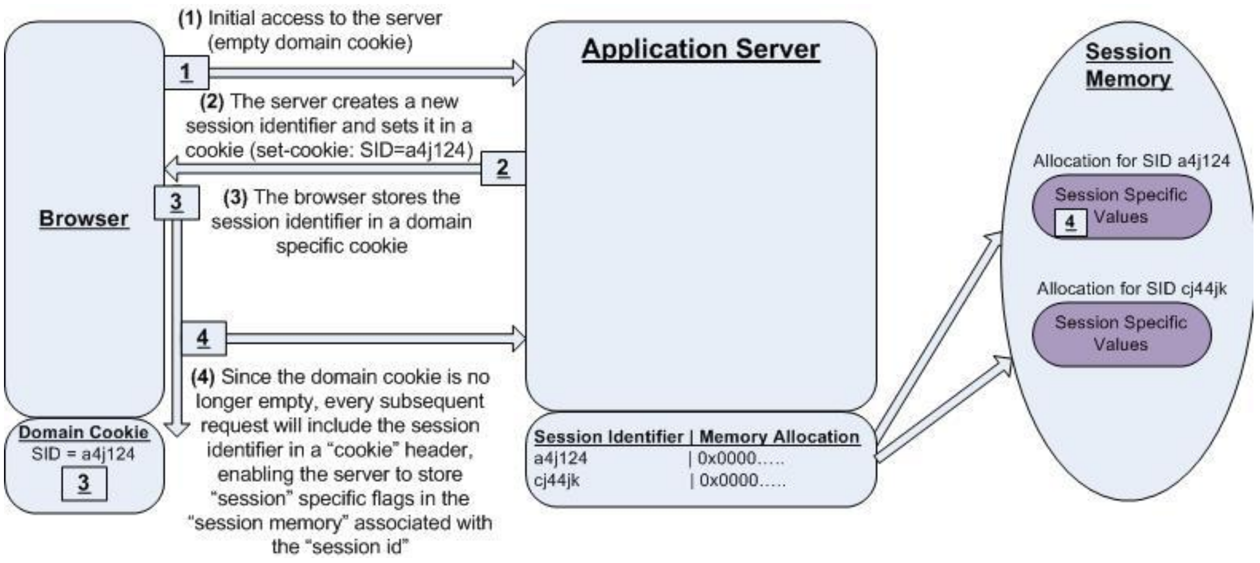
This attack vector requires detailed analysis, consistent usage of the same session identifier, intentional redirection prevention, testing a number of different sequences, and in the case of black box tests, luck.

2.3. Session – Definition, Process and Common Uses

If you are familiar with the concept of sessions, you can skip directly to section 3.

In order to properly understand session puzzles, it is crucial to understand the interpretation of the concept "session" in applications, and to differentiate between the concepts of "session", "session identifier" and "session memory allocation".

Figure 2 demonstrates the process of session generation in web applications (the process might vary in different implementations and technologies):



(1) Initial access to the server (empty domain cookie)
1
Application Server
Session Memory
(2) The server creates a new session identifier and sets it in a cookie (set-cookie: SID=a4j124)
2
Allocation for SID a4j124
Session Specific Values
4
3
(3) The browser stores the session identifier in a domain specific cookie
Browser
Allocation for SID cj44jk
Session Specific Values
4
(4) Since the domain cookie is no longer empty, every subsequent request will include the session identifier in a "cookie" header, enabling the server to store "session" specific flags in the "session memory" associated with the "session id"
Domain Cookie
SID = a4j124
3
Session Identifier | Memory Allocation
a4j124 | 0x0000…..
cj44jk | 0x0000…..

Figure 2. The Session: Generation, Storage, and Usage

In applications, the term session is defined as "an interactive information interchange (also, dialog) between two (or more) entities".

The following list describes the terms used in Figure 2:

- The term "***session***" refers to the entire dialog.
- The term "***session identifier***" (session id / SID) refers to a unique value, which is generated by the server, returned in the response, stored in a ***client side*** container (typically the cookie) and is used as a unique session "key" by the browser.
- The term "***cookie***" refers to a client side container in the browser (memory or file) which can be used to store **domain-specific** flags and values.
- The term "***session memory allocation***" refers to a memory allocation in the server side, which is associated to a specific session identifier, and is used by the server to store identifier specific values and flags (and since a browser is associated with a specific session identifier, those values are de-facto, browser instance specific flags).

The session process is typically composed of the following phases:

(1) Browsers automatically include the content of ***cookies*** in requests to their origin domain (the domain that created them), as long as the content of the cookie isn't empty. As a result, any **initial** access to a domain (or access after a previous session has "expired" or deleted) causes the browser to access the domain without any session identifiers / cookie containers (or with identifiers that "expired").

(2) When an application server is accessed without a session identifier (or with an identifier that "expired"), it generates **a new unique session identifier**, associates it to a designated server side memory allocation (in the server side "session memory"), and returns it in the response, typically through a "set-cookie:" HTTP Response Header (although additional session association methods exists).

(3) When the browser receives a "set-cookie" header, it populates the domain-specific cookie with the values defined within the header, which includes the session identifier in the currently described process.

(4) Since the domain specific cookie is no longer empty (and the session identifier defined in the cookie is not yet expired), the browser automatically sends the content of the cookie (including the session identifier) in **every subsequent** request to the server (typically in a "cookie:" HTTP Request Header).

(5) When the server receives a request that contains a valid session identifier, it is able to lookup or store session specific values in the memory associated to the session identifier.

(6) From this point on, any logical process in the application can be enforced through the use of session specific flags. For example:

   a. After a successful login process, the server typically populates the browser-specific session memory with the identity and/or permissions of the authenticated user, and use those "flags" to verify the identity (or permissions) in any other application entry point (e.g. internal authenticated entry points).

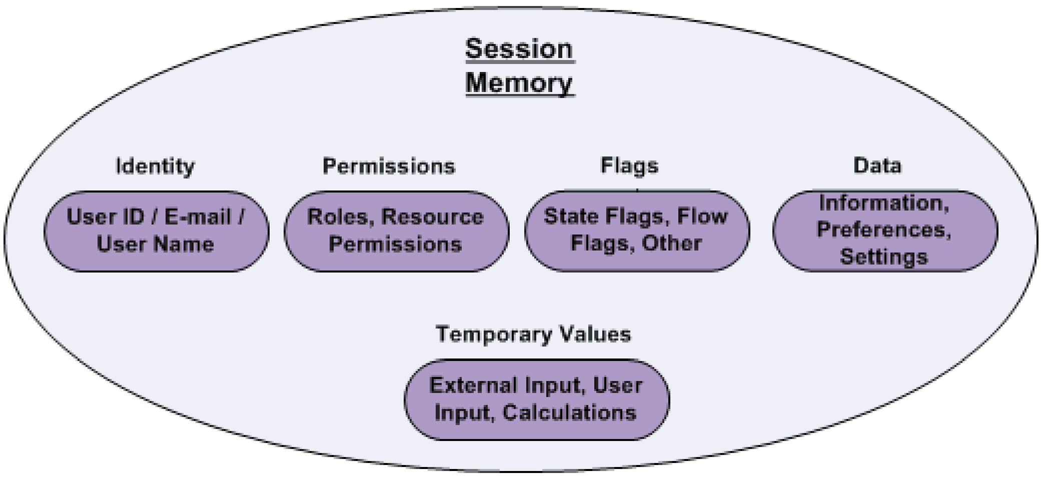
   b. When performing a process that requires several phases (such as a password recovery through a question challenge), the server could enforce the user to perform each phase through the usage of session specific flags; each phase only functions if the session memory contains a temporary confirmation flag that was assigned after the successful completion of the previous phase, and sometimes even erase that flag to prevent reuse.

(7) The session can "expire" after a predefined period of inactivity (typically 20-30 minutes), when the server "erases" the identifier and the memory allocation (logout), or when the cookie is deleted (and as a result, the session expires after a period of inactivity).

(*) The processes of session generation, management and expiration may vary in many parameters, from the storage of the session identifiers (cookie, URL, HTML parameter, etc) to the actual flow of generation, usage and expiration processes.

The main purpose of the session identifier and server side session specific memory allocation is to enable the server to differentiate between clients (or rather, browsers, in the case of a web application), and to store client specific flags in the server side.

Session flags can be used by application to identify users, enforce restrictions and gather information on the activities of a specific user, while using a memory container that's outside of a malicious user reach, since unlike the cookie (which is in the full control of the user), the session memory can't be directly affected by the client side (unless the programmer made **a critical** mistake, or included a deliberate back-door).



Session Memory
Identity
User ID / E-mail / User Name
Permissions
Roles, Resource Permissions
Flags
State Flags, Flow Flags, Other
Data
Information, Preferences, Settings
Temporary Values
External Input, User Input, Calculations

Figure 3. Typical Content Stored in the Session Memory

3. Practical Uses for Session Puzzles

3.1. Overview

Session puzzle exposures can only work on applications that use a session mechanism, which is a mechanism for storing the temporary activity state of an entity.

The actual implementation of session memory allocations could be in the RAM memory, databases, files, or in other data repositories (although memory tables and database tables are by far the most common implementations).

This document demonstrates detection and exploitation of session puzzles in web applications, ***but* it's important** to remember that these exposures can be found in **any** application that use a session mechanism, and since session mechanisms are implemented in a myriad of different technologies, the scope of this exposure is not limited to the examples provided in this document.

The following sections describe and explain different instances of session puzzle exposures, while providing methods for detecting and exploiting them.

3.2. Authentication Bypass via Session Puzzling

The authentication process can be enforced in multiple methods, while relying on various "identifiers" (tokens, certificates, credentials, etc.).

A session puzzling attack can be used to ***bypass*** authentication enforcement mechanisms of applications that enforce authentication by ***validating the existence of session variables that contain identity–related values***, which are usually stored in the session after a successful authentication process.

The authentication bypass attack vector could be executed by accessing a ***publically accessible entry point*** (e.g., web page, service, etc.) that ***populates the session with an identical session variable***, based on fixed values or on user originating input.

This "abnormal" behavior (public entry points that populate the session object with identity variables regardless of the authentication phase) can be found in many locations, but is common in the following entry points:

- **Password recovery initiation entry point/s** – developers often populate the session with identity values in the initiation phase of the password recovery process (an entry point that requires the username and/or email in order send a recovery email or present a recovery question). These entry points might populate the session with identifying values, which are received directly from the client side (username, email address, social ID number, etc), or with indentifying values that are obtained from queries or calculations based on input values (user ID, identity token, username, etc.).

- **The registration entry point/s** – developers often store in the session input values which are received during the registration process, prior to the process completion. The values "loaded" to the session usually include usernames, emails and other identifying values. This behavior is particularly common in registration processes that include multiple phases, which are implemented in several entry points (since the previous values received must be temporarily stored until the end of the process).
- **Password recovery question challenge entry point/s** – many password recovery processes operate by sending a "recovery link" to the user's email, which is embedded with a temporary token associated with the user account. When the user accesses the recovery link, he is usually presented with a password recovery question challenge, which was defined in the registration process. In some instances, the access to the challenge-question entry point and the token-to-account association verification eventually populates the session memory with identity objects, which enable malicious entities that obtained the link to use forceful browsing to access internal application modules.
- **Contact forms** – certain instances of contact forms require the user to provide identifying values, such as email addresses and social ID numbers. These values might be temporarily stored in a session variable, in order to support various logical processes.
- **Testing modules** – obsolete, duplicate and test features are in high risk of automatically populating the session memory with default hard-coded identifying values.
- **Login entry point/s with premature session population** – although login entry points are supposed to populate the session with identifying values only ***after*** a successful authentication attempts, in some rare cases, the programmer might load user originating values into the session prior to the actual validation, and thus, enable attackers to abuse this behavior (in some instances, the developer invalidates the session memory in the initial phases of the module, allocates a new identifier and session memory, populates the session with the user-originating values and only then validates the identity, while redirecting to the login page upon failure, relying on the initial "cleansing" behavior to get rid of session remains).

In general, any public (unauthenticated) entry point that accepts identifying values from external input can be abused to serve as the initial entry point for executing the attack vector, assuming it stores the values it receives in the session memory.

As soon as the collection of necessary session variables was "composed", attackers will be able to bypass the authentication enforcement using direct access to internal components, regardless of the attack vector used.

Attack Vector Assumptions:

- The authentication process is properly enforced in internal entry points, by validating the existence of an identity-related session variable.
- Due to an implementation flaw, the validated identity-related session variable can be "created" ***prior*** to the actual authentication validation, by accessing a public entry point that "populates" the session variable with **a default value** or overruns it with **a custom value**, due to a flaw, testing feature or legitimate feature requirement.

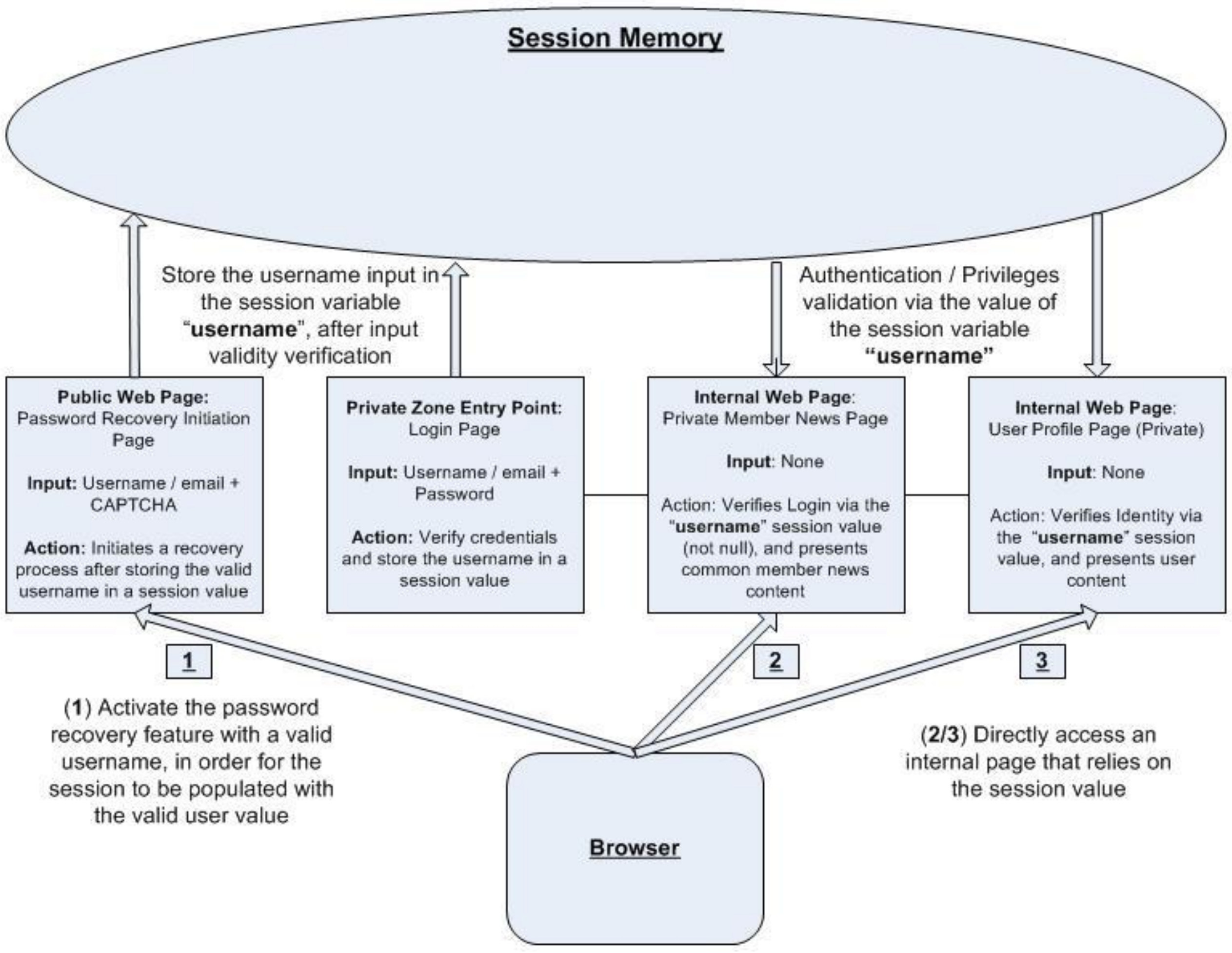

Session Memory
Store the username input in the session variable "username", after input validity verification
Authentication / Privileges validation via the value of the session variable "username"
Public Web Page: Password Recovery Initiation Page
Input: Username / email + CAPTCHA
Action: Initiates a recovery process after storing the valid username in a session value
Private Zone Entry Point: Login Page
Input: Username / email + Password
Action: Verify credentials and store the username in a session value
Internal Web Page: Private Member News Page
Input: None
Action: Verifies Login via the "username" session value (not null), and presents common member news content
Internal Web Page: User Profile Page (Private)
Input: None
Action: Verifies Identity via the "username" session value, 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

Figure 4. Sample Flow for Authentication Bypass via Session Puzzling

3.3. User Impersonation via 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.*

As mentioned in the previous section, session puzzling attacks can be used to populate the session memory of an unauthenticated entity with identity values which enables him to ***bypass*** the authentication enforcement mechanism of application, but a "side affect" of this exposure is the ability to impersonate specific users.

<u>This Scenario Differs from the Authentication Bypass Scenario Because:</u>

- ***The Identity "populated" must derive from input*** – while authentication bypassing attacks usually enables some access, even if the identity values do not represent a valid user (in cases in which a hardcoded identity is populated, for example), in order to impersonate specific users the attacker must have a method to affect the specific identity populated by sending input values containing the identity to a session "populating" entry point (usernames, user identifiers, emails, etc), or by sending values that are interpreted into the actual identity (tokens, national identification numbers, etc.).
- ***The session "population" entry point can be internal (private/authenticated) as well as external (public/unauthenticated)*** – Unlike authentication bypassing attacks in which the attacker attempts to bypass the authentication enforcement, malicious users might desire to impersonate other users as well, and in the latter case, the authentication enforcement does not need to be bypassed. As a result, the user impersonation scenario expands the range of access points that can populate and/or alter the session-stored identity, and includes 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 object with input originating identity values) can be found in many locations, but is common in the following entry points:

- All the entry points described in the authentication bypass scenario, including registration, password recovery and question challenge modules, in addition to login modules with premature session population.

- **Profile related entry points**, including entry points that enable viewing, updating, deleting or contacting other accounts.

Attack Vector Assumptions:

- The authentication process might or might not be properly enforced. However, access control is enforced when accessing private user resources and this access control relies on identity-related session variables.
- Due to an implementation flaw, the validated identity-related session variables can be "altered" or "overrun" by accessing public and private entry points that "populate" the session with an appropriate value (or "overrun" the values with user originating input).

3.4. Privilege Escalation via Session Puzzling

The RBAC security model (role based access control) is a common authorization enforcement model in applications, in which users are associated to **roles**, and roles are granted **permissions** to perform operations or view information.

The actual validation of permission is ***usually*** performed in the following events:

- Upon access to a restricted entry point.
- Prior to performing operations for the accessing entity.
- Prior to presenting information to the accessing entity.

The role of the authorization enforcement mechanism is to prevent unauthorized entities from performing restricted operations or viewing classified data, and the task of enforcing permissions is usually performed by validating the required privilege level of the operation/data/entry-point in front of the privilege level of the user;

Since the user roles, permissions and privileges are ***usually*** stored in session variables, session puzzling attacks might be able to alter those values, and thus, elevate the user's privileges and enable him to affect content that he was previously unable to access.

In order to use session puzzling for elevating privileges, the attacker should authenticate to the system (or bypass the authentication mechanism), and then access entry points that populate the session memory with role-related variables.

(*) Note – in some systems, the identity itself might be used for role verification (for example, the username "admin"), so impersonating to a specific user using session puzzling sequences might also "grant" the attacker all the associated privileges.

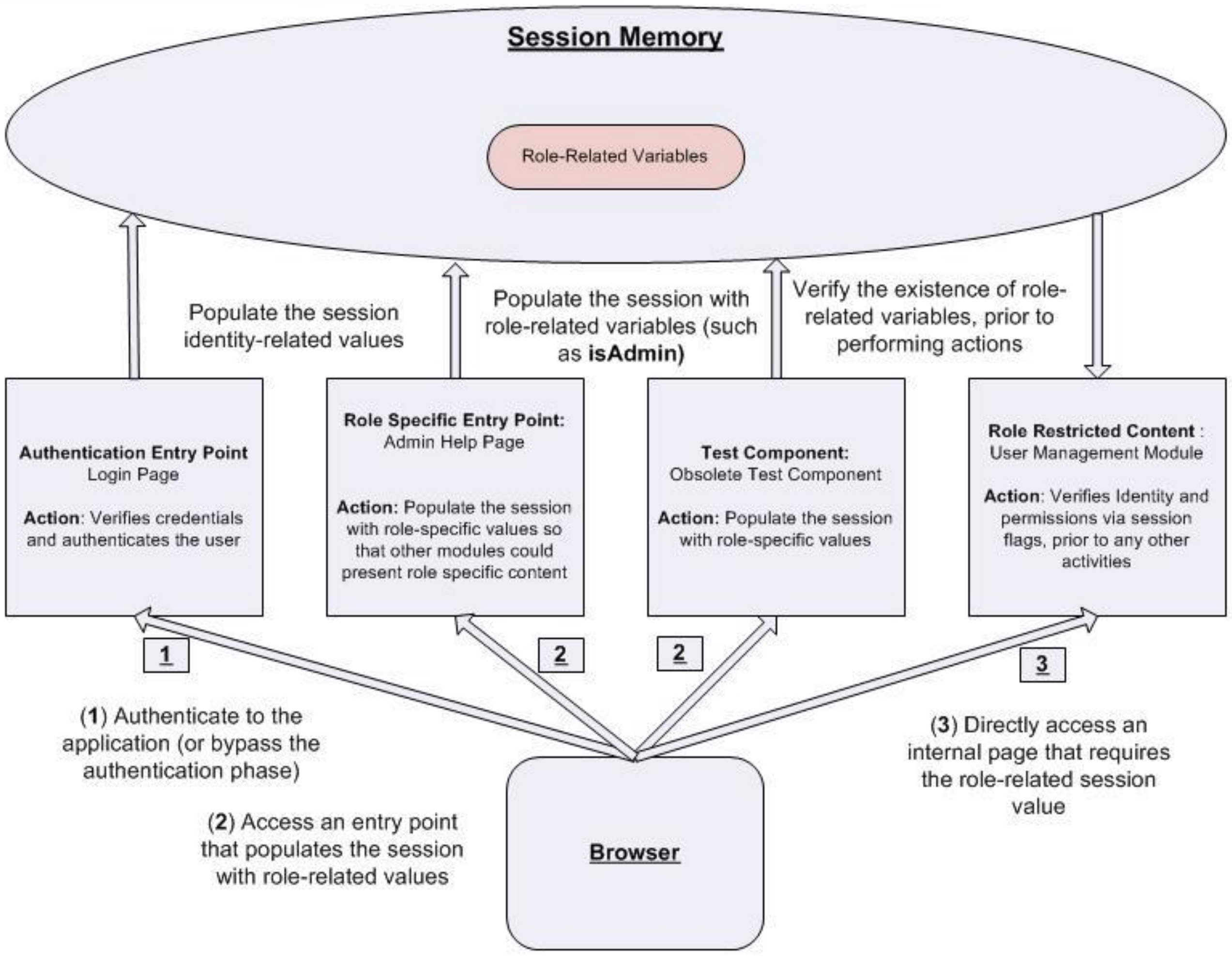

Session Memory
Role-Related Variables
Populate the session identity-related values
Populate the session with role-related variables (such as **isAdmin**)
Verify the existence of role-related variables, prior to performing actions
Authentication Entry Point
Login Page
Action: Verifies credentials and authenticates the user
Role Specific Entry Point:
Admin Help Page
Action: Populate the session with role-specific values so that other modules could present role 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

Figure 5. Sample Flow for Privilege Escalation via Session Puzzling

Since permissions, unlike **identifiers**, are rarely received from the client side, it is rather difficult to locate entry points that enable the attacker to alter role-related session variables; however, there are several locations that might serve as a good starting point:

- **Test modules and obsolete modules:** these modules might populate the session with various variables, including variables that are permission-related.
- **Role specific content** – help pages for high privileged users, menus that present role specific content, role specific entry points that fail to enforce permissions, etc.

Attack Vector Assumptions:

- The authorization enforcement mechanism is properly enforced in the **target** role-specific restricted entry points, and operates by verifying the existence of role related session variables in the user's session memory.
- Due to an implementation flaw, the validated role-related session variables can be "populated" in the session by accessing entry points available to low privileged user

accounts (including low privileged users), or by accessing "restricted" entry points that fail to enforce permissions.

3.5. Flow Enforcement Bypass via Session Puzzling

One of the most interesting capabilities of session puzzling attack sequences is the ability to bypass restrictions of sensitive multiphase processes.

Sensitive multiphase processes are modules that perform sensitive operations in the system or account, and include qualifying phases meant to verify that the initiating entity has the proper identity, permissions, target, etc. Common examples include:

- Password recovery processes (which usually require the user to access a link sent to his email, answer password recovery questions, prove his identity, etc).
- Financial transactions (which might require the user to provide a password or a token to complete the process).
- Permission grant processes (which might require the user to re-authenticate to complete the operation)

In order to prevent users from "skipping" qualifying phases in multiphase processes ("traditional" flow bypass attacks), applications use a variety of methods:

- Storing state and flow flags in the session after each successful phase, and verifying the existence of those flags in subsequent events.
- Limiting the usage of the state and flow flags in order to prevent re-use for multiple processes of the same type (each phase erases the flag stored by the previous phase, after verifying its existence, in order to prevent the flag from enabling the user to perform the same process twice).
- Using different flags in each phase.

A session puzzling attack can be used to bypass flow enforcement mechanisms that could otherwise be considered "foolproof", by exploiting the **simultaneous execution** of several multiphase processes that use **identical flow flags**.

For example, if an attacker would have wanted to bypass the qualifying phase of a multiphase financial transaction (since the phase requires the user to re-enter a password that the attacker does not have), he could have achieved that goal by starting a simultaneous multiphase registration process, which would have populated his session memory with flags that could enable him to skip phases in the transaction multiphase process (and thus, avoid the password challenge).

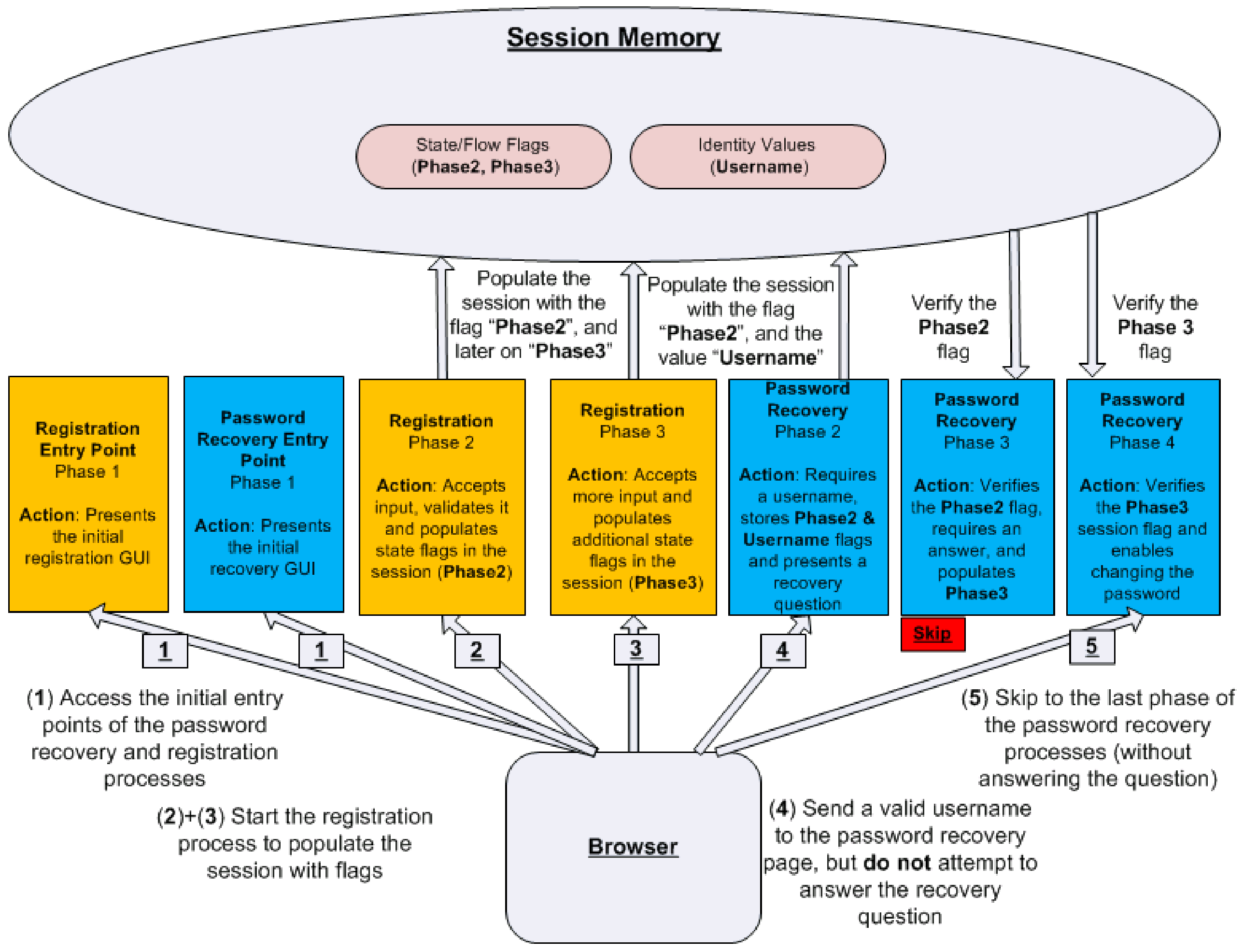

Session Memory
State/Flow Flags (Phase2, Phase3)
Identity Values (Username)
Populate the session with the flag "Phase2", and later on "Phase3"
Populate the session with the flag "Phase2", and the value "Username"
Verify the Phase2 flag
Verify the Phase 3 flag
Registration Entry Point Phase 1 Action: Presents the initial registration GUI
Password Recovery Entry Point Phase 1 Action: Presents the initial recovery GUI
Registration Phase 2 Action: Accepts input, validates it and populates state flags in the session (Phase2)
Registration Phase 3 Action: Accepts more input and populates additional state flags in the session (Phase3)
Password Recovery Phase 2 Action: Requires a username, stores Phase2 & Username flags and presents a recovery question
Password Recovery Phase 3 Action: Verifies the Phase2 flag, requires an answer, and populates Phase3
Password Recovery Phase 4 Action: Verifies the Phase3 session flag and enables changing the password
Skip
1
1
2
3
4
5
(1) Access the initial entry points of the password recovery and registration processes
(2)+(3) Start the registration process to populate the session with flags
Browser
(4) Send a valid username to the password recovery page, but do not attempt to answer the recovery question
(5) Skip to the last phase of the password recovery processes (without answering the question)

Figure 6. Sample Scenario for Flow Bypass via Session Puzzling

As a general rule, both of the multiphase processes should be accessed simultaneously, **but** the attacker should perform the process without the qualifying phase **first**, and then abuse the flags stored by the first process to bypass the qualifying phase in the sensitive multiphase process

Common non-sensitive multiphase processes that could be used for this attack vector include:

- Multiphase registration processes.
- Multiphase transactions without a qualifying phase (or with a phase that the attacker can pass/bypass).
- Initial password-change processes (after the first login, every several months).
- Etc.

Attack Vector Assumptions:

- The flow enforcement mechanism is properly enforced in the target sensitive multiphase process (at least against traditional flow bypass attacks), and operates by verifying the existence of session-stored flow flags in each and every crucial phase.

- The flags used for flow enforcement by the target sensitive multiphase process are used by **another** multiphase process **that does not include a qualifying phase** or **contains a qualifying phase that the attacker can pass, skip, delay or bypass**.

3.6. Content Theft via Session Puzzling

Session puzzles can be used to cause various abnormal behaviors in applications, which could be abused for various purposes, including a ***unique attack vector*****;**

Applications can deliver content to different external targets, including:

- Mobile phones (SMS, Video, Audio)
- Email
- Physical Mail
- External Servers.

These means are used to deliver content such as password recovery notifications, passwords, personal information, reminders, refunds, etc.

Session puzzles enable an attacker to alter the target of a content delivery module, and "redirect" private user content back to the attacker, instead of the user.

In order to accomplish the task, the attacker will need to initiate the process in the content delivery module for another user identity, while simultaneously activating another process that can overrun the content delivery target in the session memory (processes such as registration, profile updates, contact us forms, failed login attempts, etc).

For example, a password recovery mechanism which populates the session with the email address of the user whose password is being recovered could be manipulated to send the password to another email address, if a registration process is performed simultaneously; the registration process could overrun the email address in the session memory **after** the user's identity is populated to the session memory:

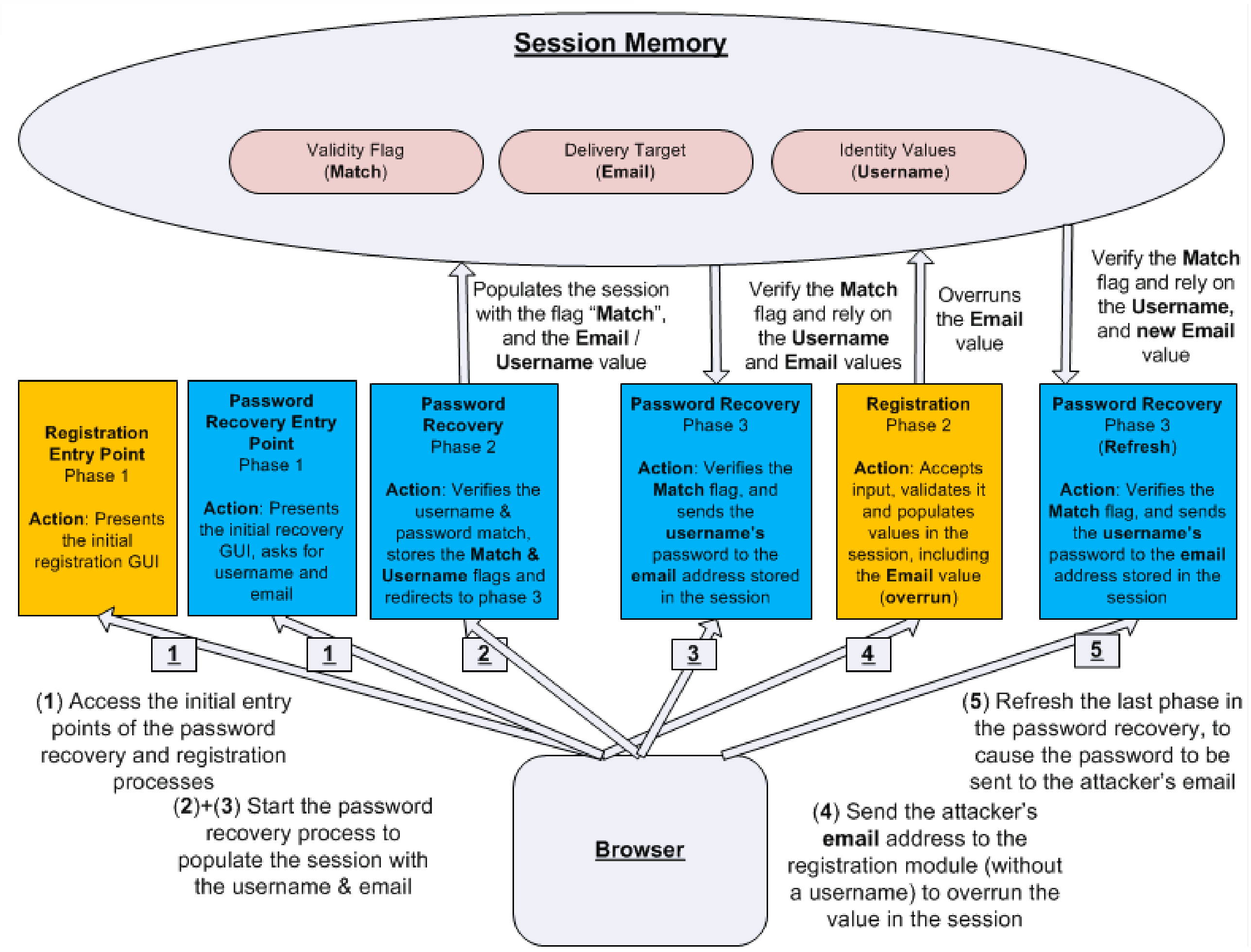

Session Memory
Validity Flag (Match)
Delivery Target (Email)
Identity Values (Username)
Populates the session with the flag "Match", and the Email / Username value
Verify the Match flag and rely on the Username and Email values
Overruns the Email value
Verify the Match flag and rely on the Username, 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

Figure 7. Sample Scenario for Content Theft via Session Puzzling

Attack Vector Assumptions:

- The content delivery target is stored in a session variable.
- Other entry points can overrun the content delivery target session variable.

3.7. Indirect Injections, Reflections and Manipulations

Session puzzling attack vectors can be used to execute "traditional" attacks in locations that were previously considered "secure". The main idea behind this attack vector is to construct the attack "payload" in a location that is considered trusted (the session), in order to bypass potential validations, or to access vulnerable locations that are **not** directly affected by input.

As a result, the payloads of attacks such as SQL Injection, Cross Site Scripting and Parameter Tampering could be delivered through a seemingly trusted location, without undergoing validation, and while expanding the **attack surface** to locations that were previously "**unreachable**".

For Example, if a module constructs SQL queries by concatenating session values into the query string, it can't be directly affected by input; however, session puzzling sequences can enable attackers to populate the session variables used in the query with malicious SQL payloads that will affect the query structure just like any input-based SQL Injection attack.

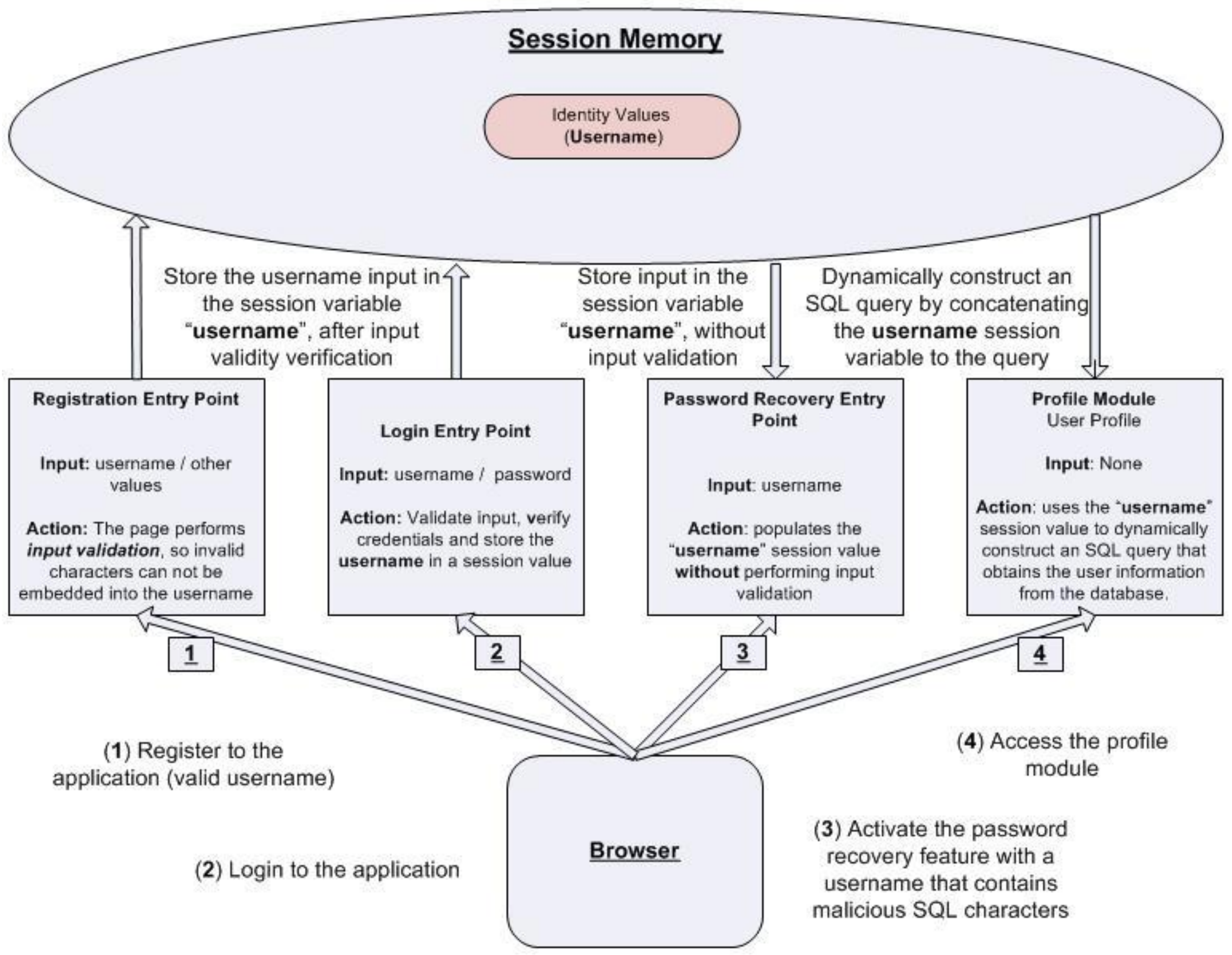

Session Memory
Identity Values (Username)
Store the username input in the session variable "username", after input validity verification
Store input in the session variable "username", without input validation
Dynamically construct an SQL query by concatenating the username session variable to the query
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
Browser
(3) Activate the password recovery feature with a username that contains malicious SQL characters
(4) Access the profile module

Figure 8. Sample Scenario for an Indirect SQL Injecion via Session Puzzling

The same model can be used to execute a variety of attacks, including LDAP Injection, XSS, Logical Manipulations, etc.

Attack Vector Assumptions:

- The application contains an entry point that performs an insecure operation based on a session variable, without performing any validations to the session variable.
- Other entry points can overrun the session variable.

3.8. Blind Session Puzzling

Since the programmer can populate the session in any entry point and in various conditions, finding all the instances of session puzzle exposures can be very difficult in large scale applications.

However, an **automated tool** can easily detect non-logical session puzzles in an automated manner, and even provide leads as to where logical puzzles may exist.

In order to achieve that goal, the automated tool/script should attempt all the possible entry point **access sequences**, with as many valid and invalid values as possible; by comparing the "normal" authenticated and unauthenticated responses of entry points to their response after a certain sequence, it is possible to locate anomalies in the behavior of entry points, and determine if the access sequence altered or added session variables that affect the response.

The anomalies can then be analyzed to determine if they are consistent, and if the sequence that caused them could be abused.

3.9. Untested Ideas & Insights

Session puzzling attack sequences can potentially be used to trigger additional events, including:

- Race conditions on session variables (relevant for situations in which session values are temporarily populated - very difficult to exploit)
- Deadlocks
- DoS
- Data corruption
- Etc.

It's important to note that none of these sequences had ever been attempted (or even properly planned), and thus, they remain as an idea that should be tested in the future.

3.10. Session Puzzling on Multiple Sessions

Session puzzling attacks can exceed the scope of a single session, and work in applications that use multiple session mechanisms (for example, an application that uses a local session identifier and an SSO session identifier, or technologies such as ASP.net); in the case of multiple sessions, the tester **might be required to combine different session identifiers from different processes**, in order to execute the attack**.**

4. Black-box & White-box Testing Methods

4.1. General Session Puzzling Guidelines

Regardless of the required attack vector, most session puzzling attacks should be performed while paying attention to the following guidelines:

- While session puzzling, the same session identifier should be used to access all the application entry points, unless this identifier was deleted or expired.
- It's best to ***avoid*** entry points that might "***erase***" the session identifier and memory allocation, cause the session identifier to ***expire***, ***delete objects*** from the session memory allocation, or populate it with ***conflicting values*** (for example: logout access points, session expiration notification access points, certain implementations of login entry points, etc.).
- As a general rule, it is best to **ignore** redirections, in order to avoid access points that erase session memory allocations or session variables. It's important to note that some access points, which populate the session themselves, might ***initially*** "***erase***" session content prior to populating it with new values, and thus, even redirection to the same access point should be avoided.
- Test different sequences for exploiting each instance; some session puzzles require specific input, while some work only in hardcoded values. Some require a simple sequence, while some work only in rare conditions. When the source code is not available, persistence can fill the gap.
- Use a "guided" automated crawling process to attempt and populate the session, especially if there is a large number of publically accessible entry points.

4.2. Black-box Testing Methods

When testing for session puzzles, the following guiding questions can help the tester achieve insights on the session usage and potential of each entry point:

- Does the entry point need to populate the session with values? If it does, then for what purpose?
- Which values might be stored in the session by the entry point?
- Which entry point might use the session-stored values and when?
- Which additional entry points might store, rely on or use the same session-stored values?
- Can one or more values be replaced? What is the effect?

- Do these session values provide access to additional entry points which store additional values in the session?

Although there are endless possibilities as to where a session puzzling sequence can start from, the following locations can serve as a good starting point:

Common authentication bypass and impersonation sequences:

- Register with a username (existing and new), while trying to access internal entry points during the process.
- Start a password recovery process with valid and invalid usernames, while trying to access internal entry points during the process.
- Access password recovery question-challenge email links, and try to access internal entry points before answering the recovery question.
- Activate login entry points with valid usernames, avoid any redirection instructions, and try to access internal entry points afterwards.

Common flow-bypass sequences:

- Attempt to perform multiphase processes simultaneously, while trying to skip phases in some, after advancing others (registration, password recovery and transactions are all good choices).

Common privilege escalation sequences:

- Authenticate, access obsolete content and test pages, and eventually access entry points that were previously inaccessible (restricted).
- Authenticate, access every entry point that does not initialize the session (login, logout, etc.), and then attempt to access entry points that were previously restricted.

Common content hijack sequences:

- Attempt to affect the target of content delivery and credential recovery processes during the initiation of the processes by attempting to activate features such as registration and profile update (which might overrun the session variable that contains the original delivery target – e.g., SMS, email, home address, etc).

4.3. Code Review Guidelines

The best method to detect session puzzles is during a security code review.

While reviewing the code, it's important to keep track of the following subjects and objects, for each and every entry point:

- Which session values are created by the entry point?
- Which input values have affected the session stored values?
- Which input values are sent to the entry point?
- Which session values are being relied on?
- Which session values are used by the entry point? Are those values used in a way that could have enabled an attack if these values originated from the client side?
- Does the entry point require authentication?
- Is the access to the entry point restricted to a specific role?
- Is the entry point implementing a part of a multiphase process?
- Is the entry point responsible for delivering content via SMS, email, or similar methods?
- Does the entry point enable access to private user content?

At the end of the code review process, the auditor will be able to detect different sequences due to the documentation of each entry point variables.

5. Mitigations

Session puzzles are vulnerabilities that require a specific code level mitigation, and can't be easily blocked in a centralized mechanism, due to the various sequences and behaviors that can be used in this attack vector.

However, the following guidelines should prevent most occurrences, if implemented properly and thoroughly throughout the code:

5.1.1. Store Objects Instead of Variables

As a general guideline, it's better to store objects in the session, instead of individual variables, since it enhances the ability of the developer to track the session content, makes maintenance tasks easier and reduces the probability for session leftovers.

This model can also help prevent entry points from overrunning identical values of other processes (for example, the registration object could not override the identical values of the password recovery object).

5.1.2. Use Different Objects for Authenticated / Unauthenticated Zones

It's important to make sure that the session objects used in the unauthenticated section of the application will differ from those used in the authenticated entry points, in order to prevent public entry points from overrunning sensitive session variables.

5.1.3. The Login Module Should Populate the Identity and Privilege Values

As a general rule, the login module should be the only module that populates the session with identity and privilege values; public modules that need to temporarily store the user's identity should use a different temporary session object that is not used by any other process.

5.1.4. Use Different Flow-tracking Flags for Each Multiphase Process

When implementing multiphase processes, it's crucial to use **unique flow/state variables** in different processes (password recovery, registration, transactions, etc). Naming flags according to the process that uses them is an action that should be adopted as a best practice.

5.1.5. Only Populate the Session with Values after Validations

In case the session needs to be populated with values, make sure those values are populated after all the validations take place. Avoid storing values in the session in advance.

5.1.6. Do Not Store Unnecessary Values in the Session

Entry points should not populate the session with variables that don't have an explicit role in the current implemented functionality.

6. Resources

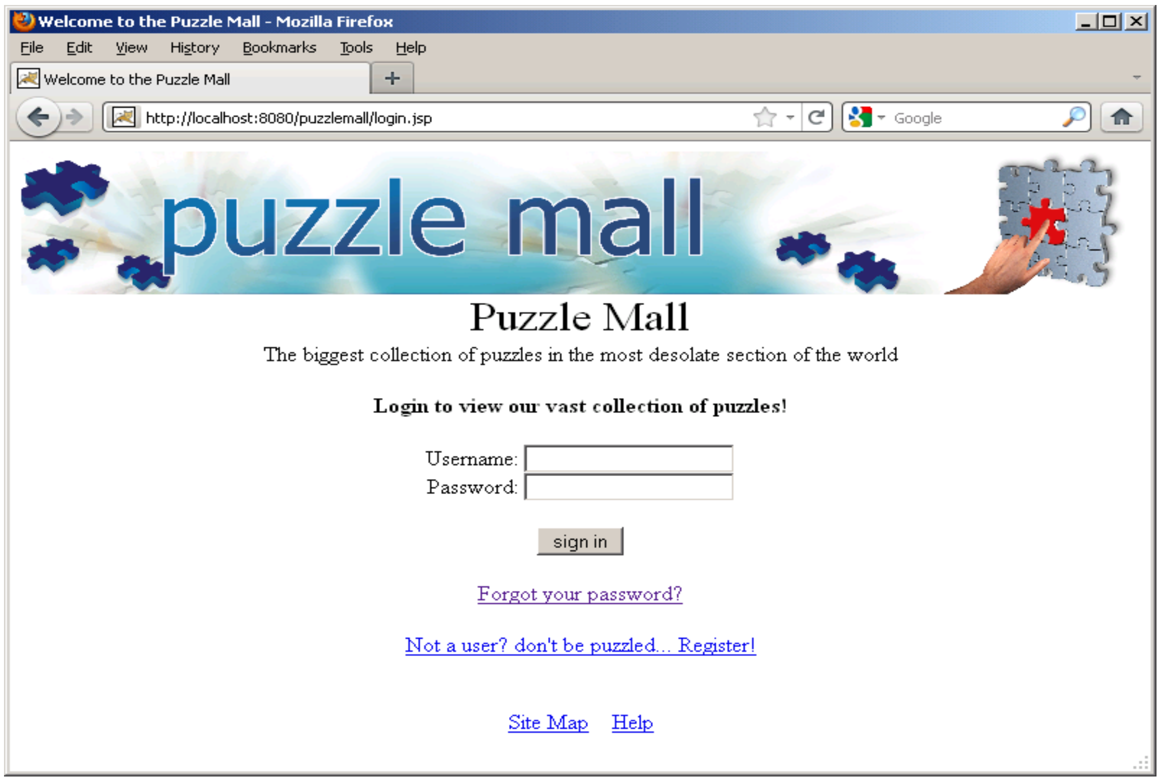
6.1. PuzzleMall – A Sample Vulnerable Web Application

In order to help pen-testers fine-tune their Session Puzzle detection skills, I developed a dedicated web application named "**PuzzleMall**".

This application is vulnerable to several Session Puzzle exposures which can be exploited using different sequences, and are meant to simulate the most common cases. The application also includes a detailed walkthrough (the pen-tester help page), which covers ***most*** of the Session Puzzle attack sequences in the application, and lists the credentials of default users in the system.

It's simple to install, and can be obtained from the following address (the installation instructions are provided in the web site, no configuration or database installation required):

http://code.google.com/p/puzzlemall/



http://localhost:8080/puzzlemall/login.jsp

Puzzle Mall

The biggest collection of puzzles in the most desolate section of the world

Login to view our vast collection of puzzles!

6.2. Presentation and Online Resources

The idea behind Session Puzzling was first introduced in a local OWASP chapter meeting, at May 17, 2011 (although we have been using it internally for a few years). The meeting also included live demonstrations of Session Puzzle attack vectors, a description of real hacking incidents that were performed using some exposure instances, and in addition, vulnerabilities in products that match the session puzzle category.

The original presentation can be obtained in the following address:

http://code.google.com/p/puzzlemall/downloads/detail?name=Session%20Puzzles%20-%20Indirect%20Application%20Attack%20Vectors%20-%2017%20May%202011%20EY%20HASC%20-%20Presentation.pptx&can=2&q=

In order to understand how realistic the sample attack vectors are, it's important to clarify that we have personally witnessed the detection of each and every instance in live systems. Some instances have even been exploited in real hacking incidents.

A good example for exploiting a "real" session puzzle is presented in a demonstration movie that shows the exploitation processes of an authentication bypass (via session puzzling) vulnerability in Oracle E-Business Suite (reported by Hacktics in 2009):

http://www.hacktics.com/content/advisories/AdvORA20091214.html

6.3. Additional Publications

An attack similar to session puzzling is mentioned under the name "session poisoning", but the session puzzling sequences differ from this attack mainly by the lack of direct input dependency (see the **flow bypass scenario** and the **exception scenario**), and expand the attack tool-set in the aspect of methodology, scope of modules and complementary methods.

Furthermore, the session poisoning attack have been "mistreated", and due to awareness issues and a debate whether or not it should get its own classification, was not included in any of the OWASP or WASC vulnerability lists.

7. Credits

7.1. About the Author

Shay Chen is the CTO of Hacktics Advanced Security Center (HASC), the security excellence center of Ernst & Young. In his current position in HASC, Shay is in charge of research, training, optimization, quality assurance and the constant improvement of HASC security services.

7.2. Additional Contribution

I'd like to thank the following individuals for their contribution to the release of this white paper:

- ***Ernst & Young*** – for letting me invest the time to write this paper.
- ***Oren Hafif, Hacktics ASC*** – for additional ideas, ***brilliant insights*** and spectacular video editing skills, used to create numerous session puzzle POC movies.
- ***James Philippe, Huston ASC*** – for providing guidance, support and documentation.
- **Oren Ofer, Hacktics ASC** – for additional Session Puzzle samples & demonstration movies.

7.3. About Ernst & Young

About Ernst & Young

Ernst & Young is a global leader in assurance, tax, transaction and advisory services. Worldwide, our 130,000 people are united by our shared values and an unwavering commitment to quality. We make a difference by helping our people, our clients and our wider communities achieve potential.

About Ernst & Young's Technology Risk and Security Services

Information technology is one of the key enablers for modern organizations to compete. It gives the opportunity to get closer, more focused and faster in responding to customers, and can redefine both the effectiveness and efficiency of operations. But as opportunity grows, so does risk. Effective information technology risk management helps you to improve the competitive advantage of your information technology operations, to make these operations more cost efficient and to manage down the risks related to running your systems. Our 6,000 information technology risk professionals draw on extensive personal experience to give you fresh perspectives and open, objective advice – wherever you are in the world. We work with you to develop an integrated, holistic approach to your information technology risk or to deal with a specific risk and security issue. And because we understand that, to achieve your potential, you need a tailored service as much as consistent methodologies, we work to give you the benefit of our broad sector experience, our deep subject matter knowledge and the latest insights from our work worldwide. It’s how Ernst & Young makes a difference.

For more information, please visit http://www.ey.com.