THE BOOMERANG EFFECT

Using Session Puzzling to Attack Apps from the Backend

Shay Chen, CTO
@sectooladdict

Hacktics ASC, Ernst & Young

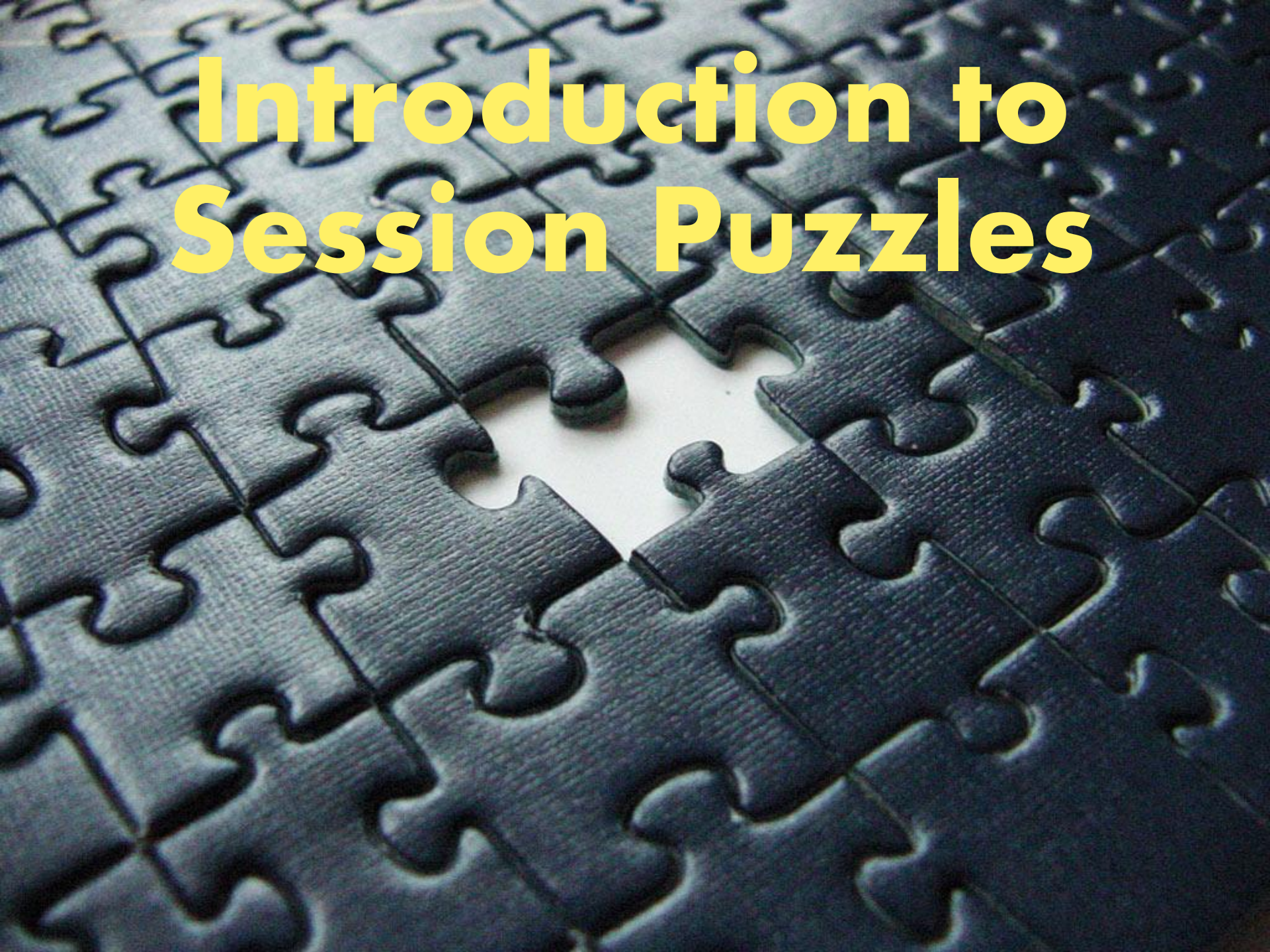
November 22nd, 2013

EY Building a better working world

DEEPSEC

About 

- Formerly a boutique company that provided information security services since 2004.
- As of 01/01/2011, Ernst & Young acquired Hacktics professional services practice, and the group joined EY as one of the firm's advanced security centers (ASC).


Introduction to
Session Puzzles

Session Puzzles – What's That?

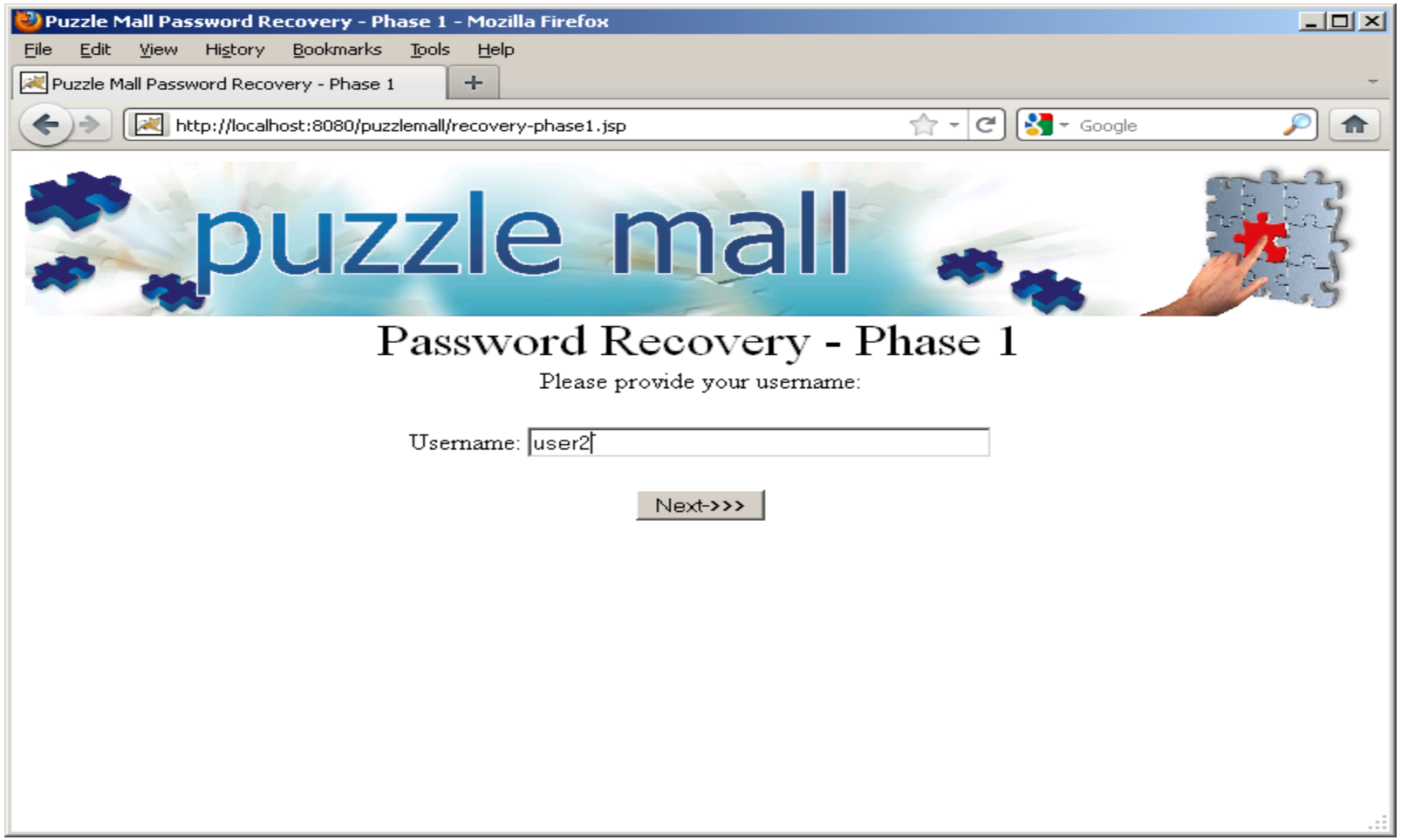
- **Session Puzzles** are application-level vulnerabilities that can be exploited by **overriding session attributes**
- The **"Session Puzzling"** exploitation process is referred as **"Session Variable Overloading"** by OWASP.
- Potential exploitation examples:
  - Bypass authentication and authorization enforcement
  - Elevate privileges
  - Impersonate legitimate users
  - Avoid flow enforcement restrictions
  - Execute "traditional attacks" in "safe" locations
  - Affect content delivery destination
  - Cause unexpected application behaviors

Indirect Session Attacks – Why Bother?

- Since the concept of indirect attacks suggests that the target is not attacked **directly**, the model itself has several benefits:
  - Low probability for code level mitigations.
  - Avoid detection by following a “valid” behavior pattern.
- Furthermore, since the exposure enables unique attack vectors, the attacker can exploit new exposures:
  - Gain control over a valid account or even an application without sending a single malicious input.
  - Perform new types of logical attacks.

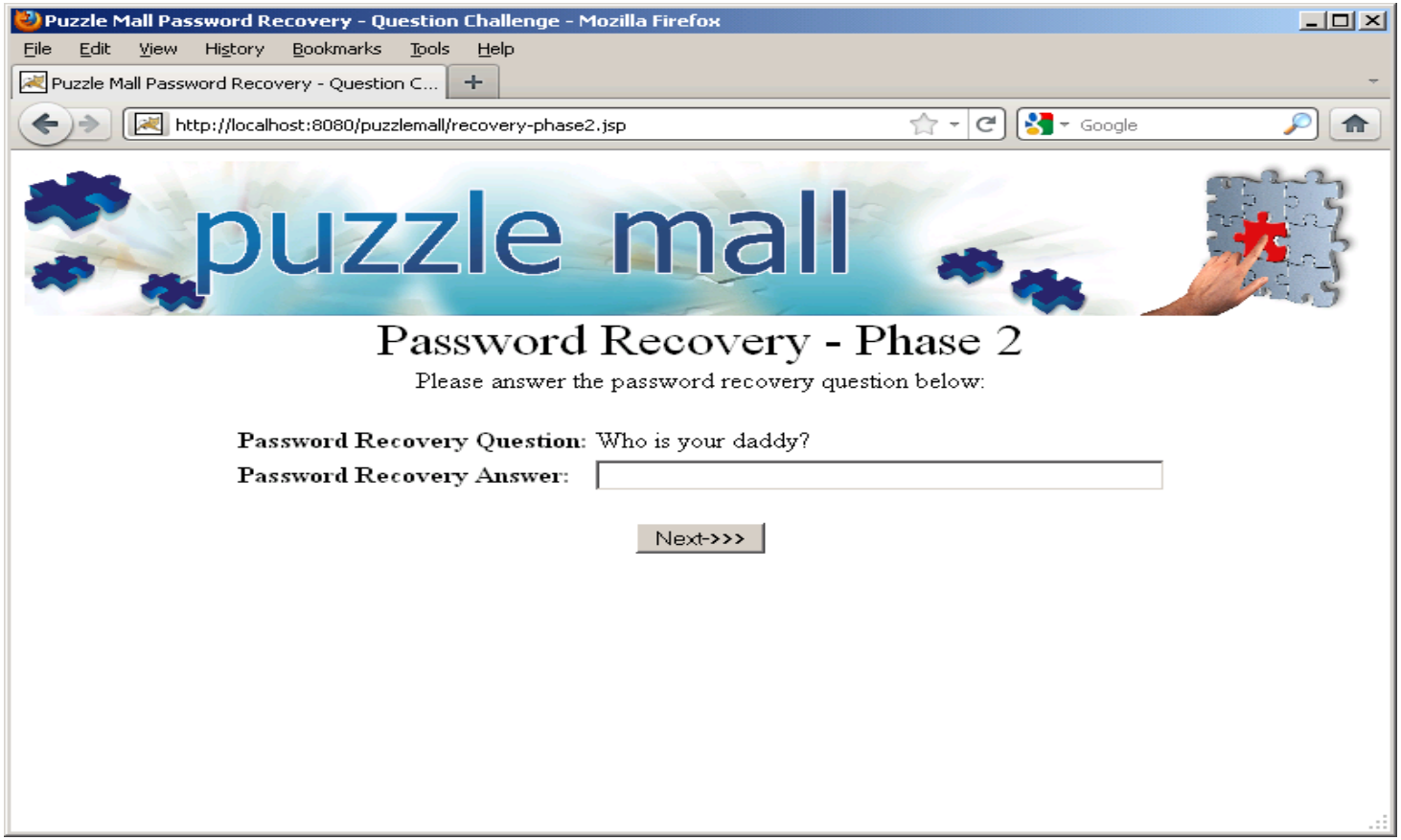
Session Puzzling - Example (1 of 3)

- Starting a password recovery process with a valid user



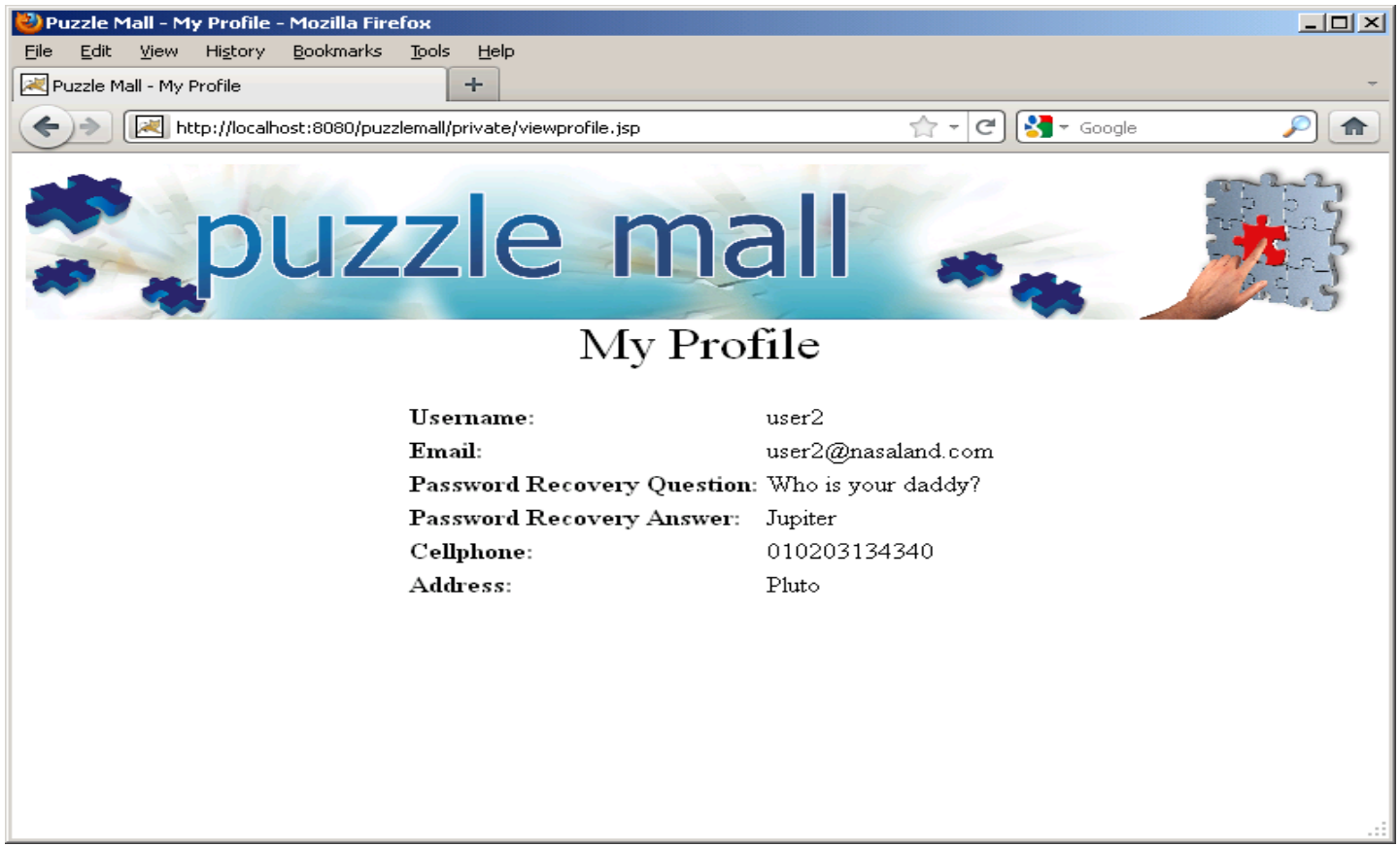
Session Puzzling - Example (2 of 3)

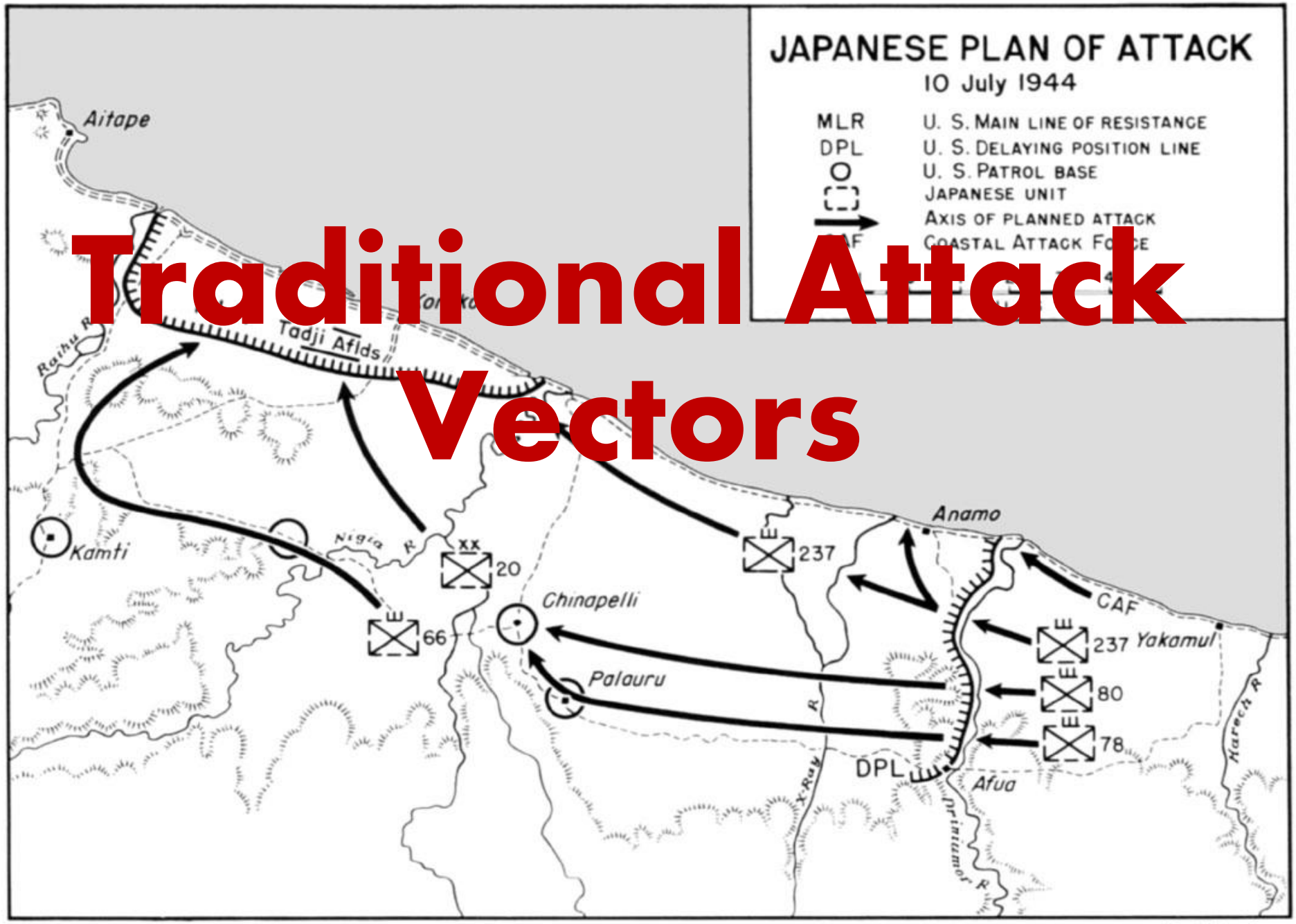
- The process populates the session memory with the username value…



Session Puzzling - Example (3 of 3)

- Tthe attacker directly access an internal page that relies on the session-stored username variable

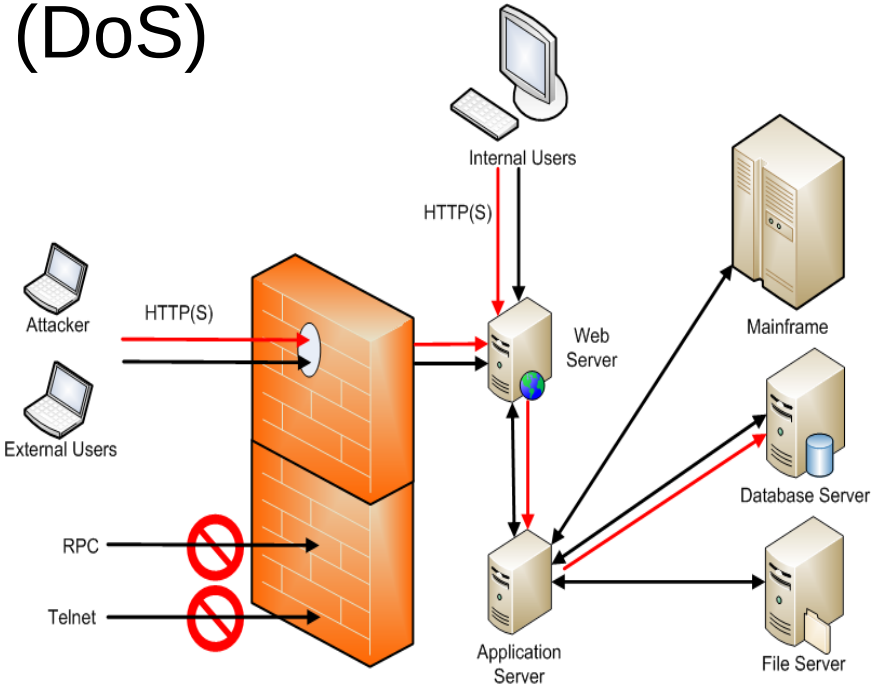


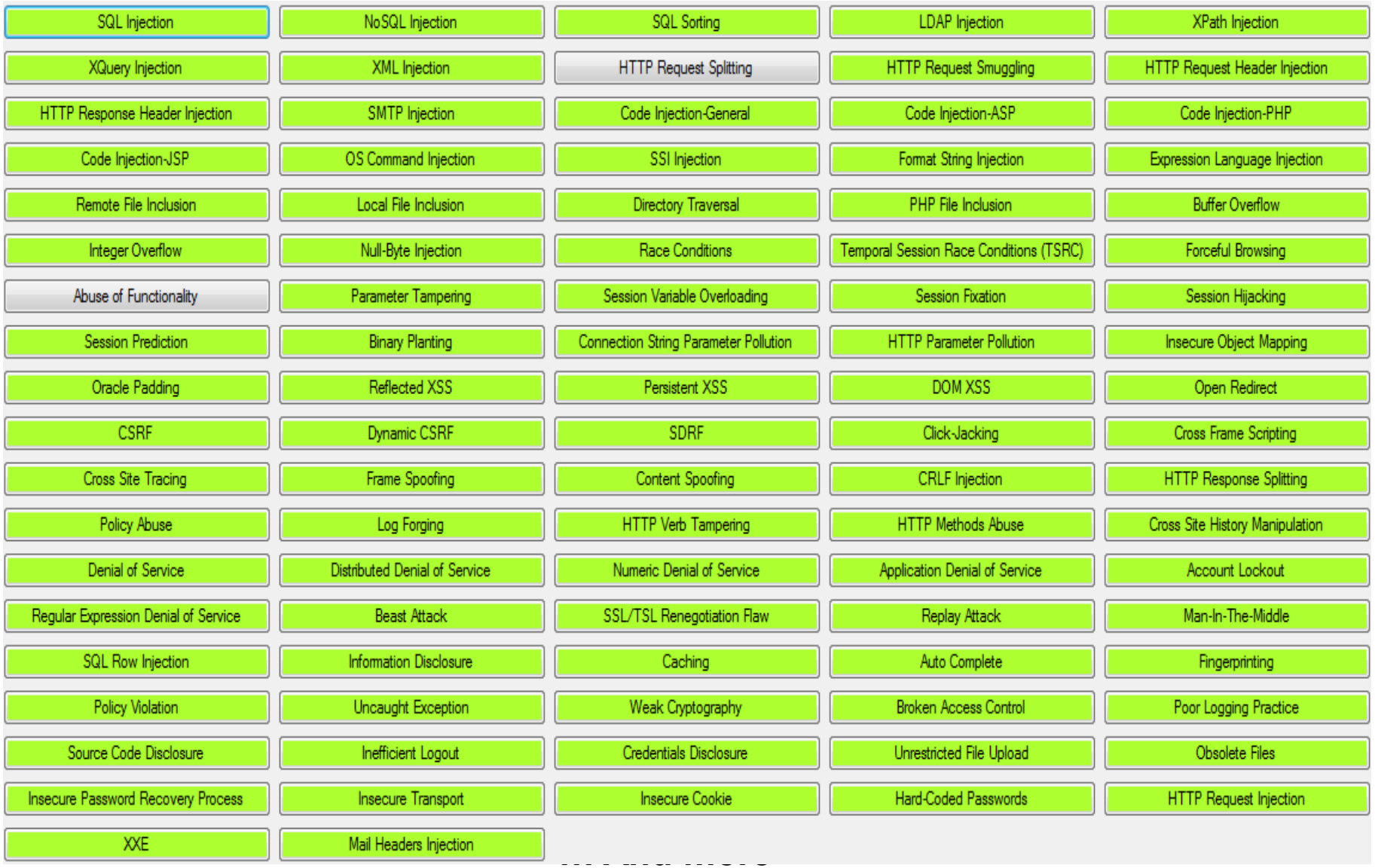

JAPANESE PLAN OF ATTACK
10 July 1944
MLR U. S. Main Line of Resistance
DPL U. S. Delaying Position Line
U. S. Patrol Base
Japanese Unit
Axis of Planned Attack
Coastal Attack Force
Traditional Attack Vectors
Aitape
Tadji Afids
Raihu R.
Kamti
Nigia R.
XX
20
66
237
Anamo
Chinapelli
Palauru
CAF
237
Yakamul
80
78
DPL
Afua
X-Ray R.
Driniumor R.
Karech R.
R Johnstone

MAP 6

“Traditional” Application Attack Vectors

- Malicious Inputs
- Forceful Access
- Consuming Resources (DoS)
- Enumeration
- Redirection
- Abusing Features
- Etc




SQL Injection
NoSQL Injection
SQL Sorting
LDAP Injection
XPath Injection
XQuery Injection
XML Injection
HTTP Request Splitting
HTTP Request Smuggling
HTTP Request Header Injection
HTTP Response Header Injection
SMTP Injection
Code Injection-General
Code Injection-ASP
Code Injection-PHP
Code Injection-JSP
OS Command Injection
SSI Injection
Format String Injection
Expression Language Injection
Remote File Inclusion
Local File Inclusion
Directory Traversal
PHP File Inclusion
Buffer Overflow
Integer Overflow
Null-Byte Injection
Race Conditions
Temporal Session Race Conditions (TSRC)
Forceful Browsing
Abuse of Functionality
Parameter Tampering
Session Variable Overloading
Session Fixation
Session Hijacking
Session Prediction
Binary Planting
Connection String Parameter Pollution
HTTP Parameter Pollution
Insecure Object Mapping
Oracle Padding
Reflected XSS
Persistent XSS
DOM XSS
Open Redirect
CSRF
Dynamic CSRF
SDRF
Click-Jacking
Cross Frame Scripting
Cross Site Tracing
Frame Spoofing
Content Spoofing
CRLF Injection
HTTP Response Splitting
Policy Abuse
Log Forging
HTTP Verb Tampering
HTTP Methods Abuse
Cross Site History Manipulation
Denial of Service
Distributed Denial of Service
Numeric Denial of Service
Application Denial of Service
Account Lockout
Regular Expression Denial of Service
Beast Attack
SSL/TSL Renegotiation Flaw
Replay Attack
Man-In-The-Middle
SQL Row Injection
Information Disclosure
Caching
Auto Complete
Fingerprinting
Policy Violation
Uncaught Exception
Weak Cryptography
Broken Access Control
Poor Logging Practice
Source Code Disclosure
Inefficient Logout
Credentials Disclosure
Unrestricted File Upload
Obsolete Files
Insecure Password Recovery Process
Insecure Transport
Insecure Cookie
Hard-Coded Passwords
HTTP Request Injection
XXE
Mail Headers Injection

Common Attack Vector Traits

- **Directly attack** the target through payloads, redirection or direct access to resources.
- **Straightforward** detection and exploitation methods.
- Potentially “**Noisy**”: might be detected by various mechanisms, due to abnormal and sometime intrusive behavior.

Session Puzzling Traits Comparison

- Access a **sequence** of entry points in a **pre-planned** order, **random** order or **timely** manner.
- "**Indirect**" - Attack a target indirectly by "composing" a back-end hosted "payload" that is delivered to it indirectly through a relatively trusted source – the session.
- "**Silent**" – ideal for stealth attacks and avoiding security mechanisms that validate input.
- "**Unknown**" – exploiting scenarios that are currently rarely mitigated.
- "**Obscure**" – inconsistent detection and exploitation methods.


Session Puzzle
Variants
In the Wild

A Couple of Prominent Examples

- Oracle E-Business Suite
  - Authentication Bypass
  - Privilege Escalation and Admin Takeover
- Sony Network Account Service System
  - Reset passwords of Sony Playstation users
- Undisclosed Vulnerabilities in Banks
  - Skip verification phases in multiphase transactions

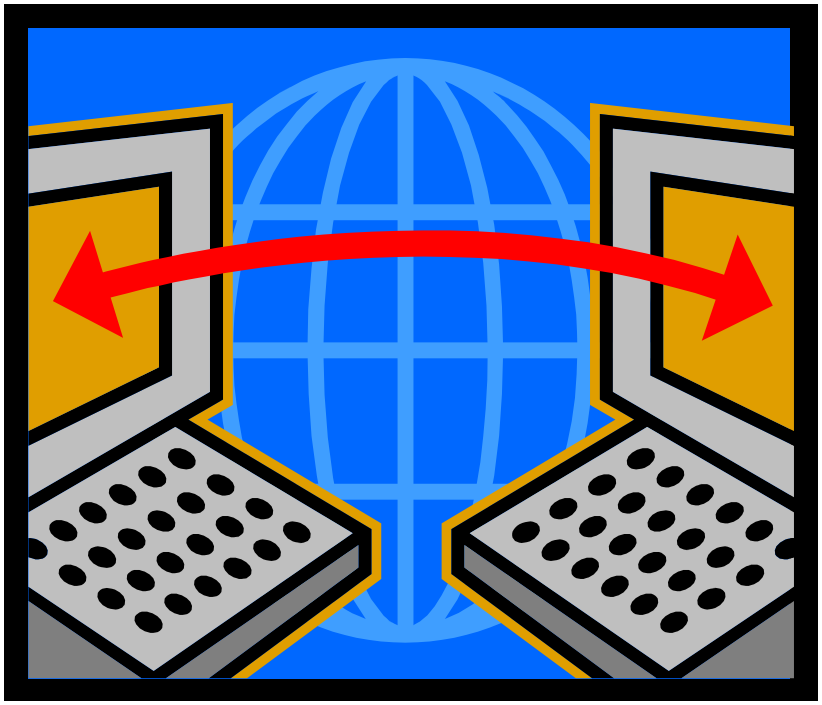
Insurance Company Site Corruption

- **2008**: An attacker gains remote control over the administrative interface of a European insurance company, and starts corrupting the web site content.
- An investigation performed revealed that the attacker gained control by crawling the entire application tree **twice**, using paros proxy, prior to accessing the administrative login page (which resided in a trivial URL address).
- The act of crawling automatically submitted contact-us forms, which populated the attacker's session with values that were used by the administrative application for authentication enforcement.

European Bank Back-Door Sequence

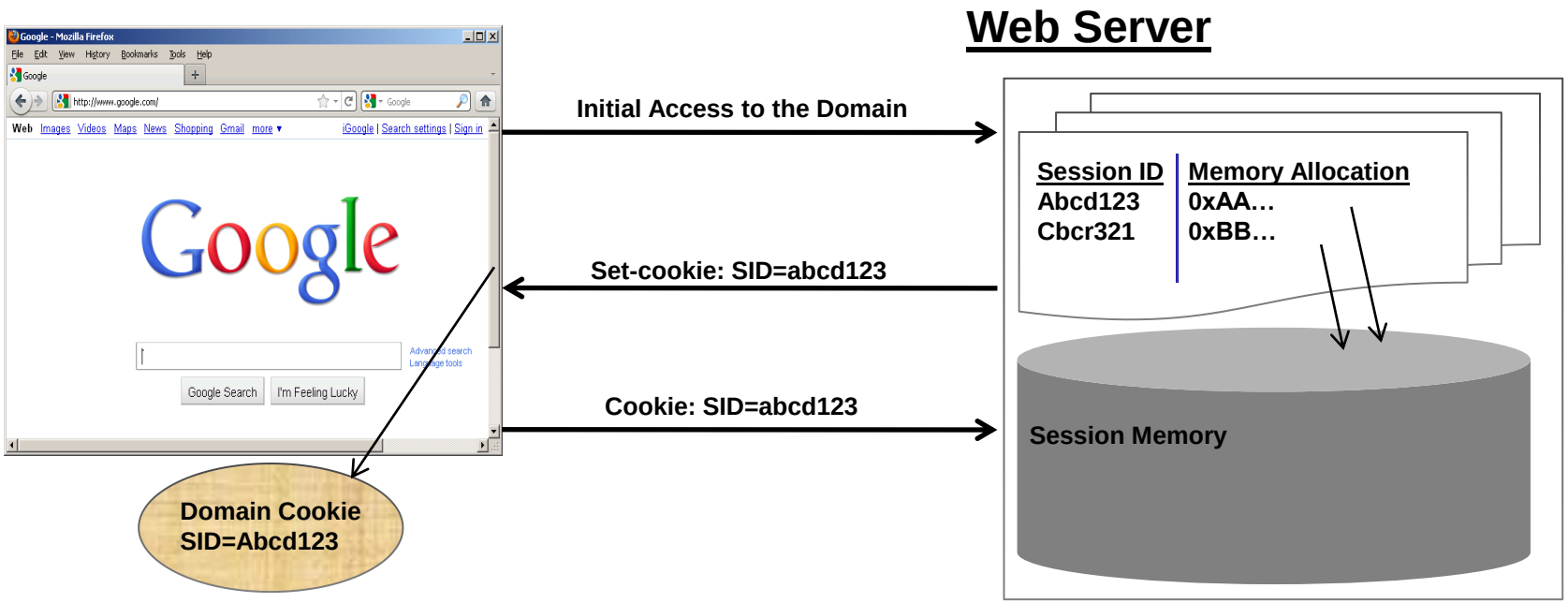
- **2007**: A session puzzle exposure was detected in a security assessment of a European banking application.
- The vulnerability enabled the attacker to gain **complete control** over the system (by activating a dormant feature), by accessing a sequence of seven different pages.

The Session Mechanism



The Session Mechanism

- The process of session identifier generation and association



Initial Browser Access → Session Identifier Generation → Session Memory Association → Session Identifier Storage → Session Identifier Reuse

The Session Lifespan in Web-Apps

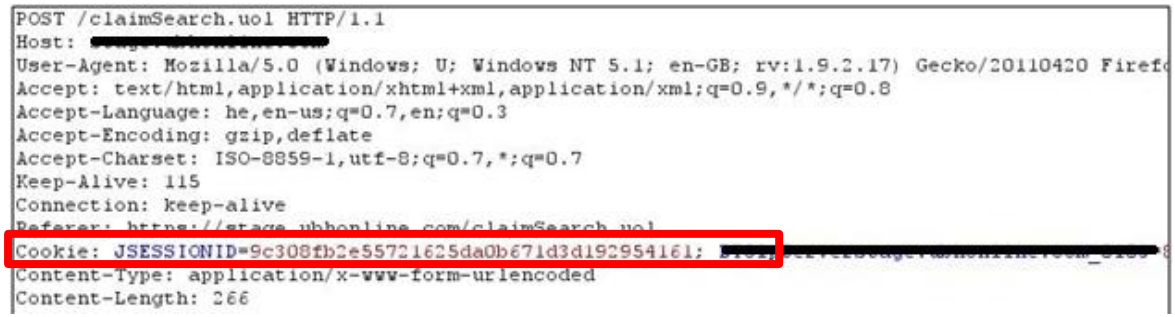
- Initial browser access to server -> generation of a new session identifier.

- The session identifier is returned to the browser, ***usually*** in a “set-cookie” response header.

```
Content-length: 0
Content-type: text/plain; charset=iso-8859-8
Set-cookie: JSESSIONID=963068b68d2c1d43e6d168178058 1e657c6c;secure;path=/
Location: https://stage.ubhonline.com/loginUser.uol
Content-Language: iw
```

The Session Lifespan in Web-Apps

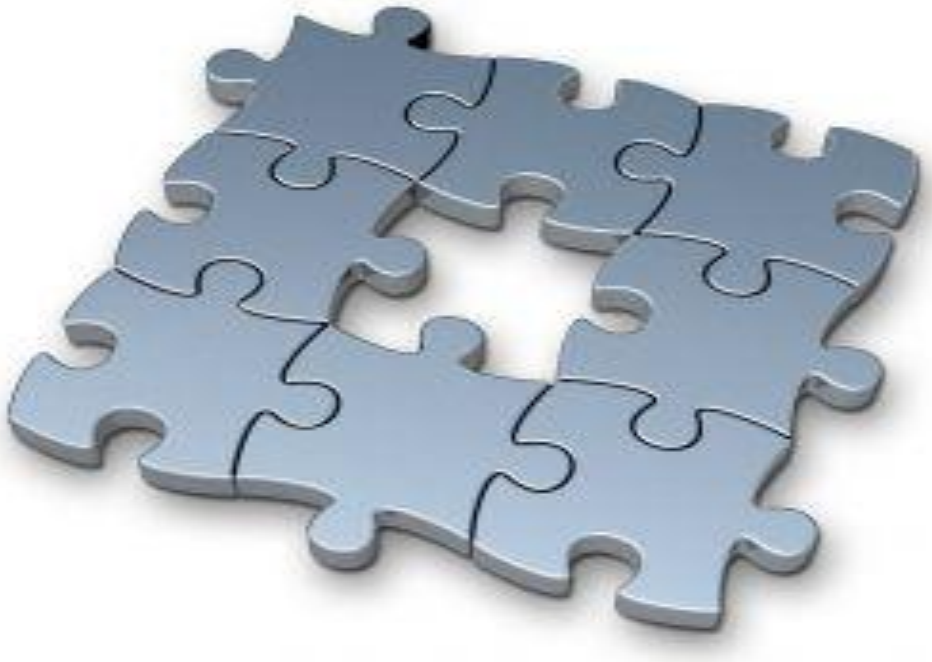
- The browser stores the identifier in a domain **cookie**,
- **Domain-specific cookies** are sent to the domain in every request (including the session identifier).
- The server uses the session identifier to "**associate**" the browser instance with the memory allocation
- Associated **memory** can store **flags**, **identities**, and browser instance specific data.



Session Stored Values

- Since sessions enable applications to “track” the state of browsers, they are used to store a variety of browser-instance related values:
  - ***User Identities*** (user identifiers, usernames, email addresses, social ID numbers, etc.)
  - ***Permissions*** (roles, resource lists, etc.)
  - **Flags** (Flow flags, State flags, etc.)
  - **Input** (Especially input from multiphase processes)
  - **Results of Operations, Queries, and Calculations**
  - **Etc.**

Session Puzzling Sequences

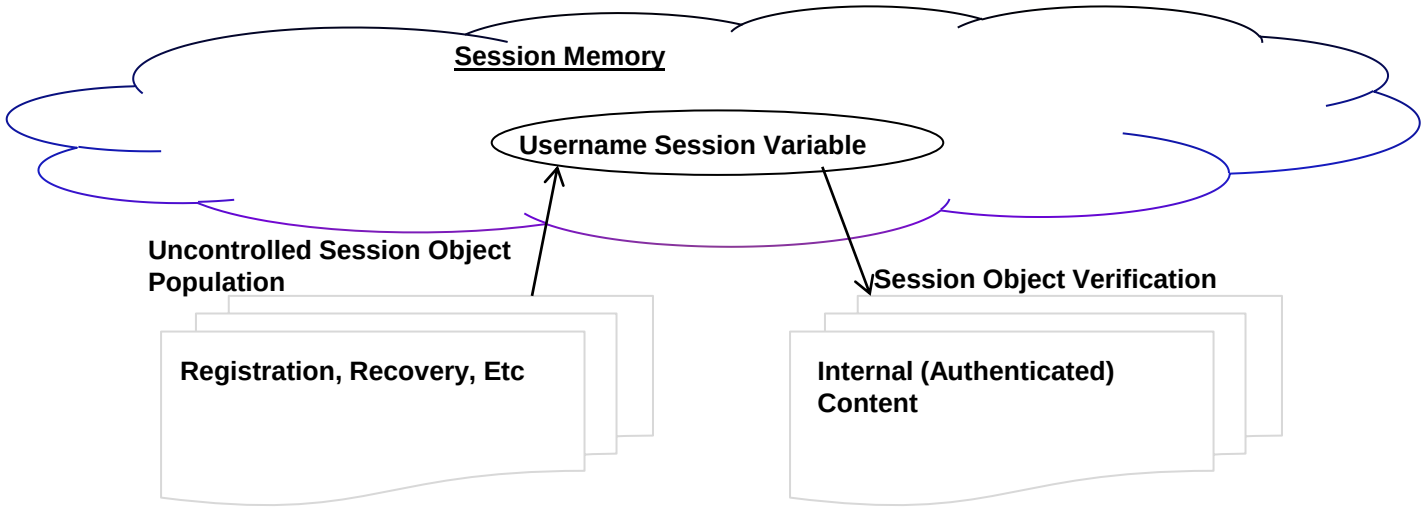


Session Puzzling Attack Sequences

- As mentioned earlier, session puzzles can be exploited in a variety of ways. Common instances include (but not limited to):
  - Authentication Bypass via Session Puzzling
  - Impersonation via Session Puzzling
  - Flow Bypass via Session Puzzling
  - Privilege Escalation via Session Puzzling
  - Content Theft via Session Puzzling
  - Indirect “Traditional” Attacks

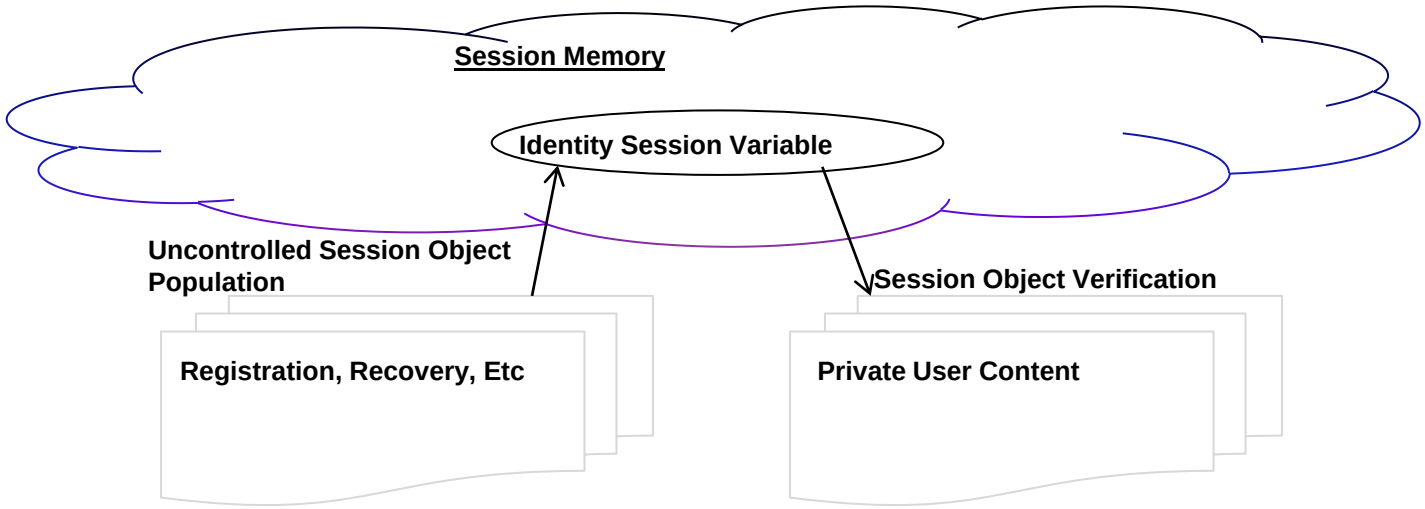
Authentication Bypass via Session Puzzling

- Authentication mechanisms that enforce authentication by validating the existence of identity-related session variables can be bypassed by accessing public entry points that might populate the session with identical values (registration modules, password recovery modules, contact-us forms, question challenges, etc.).



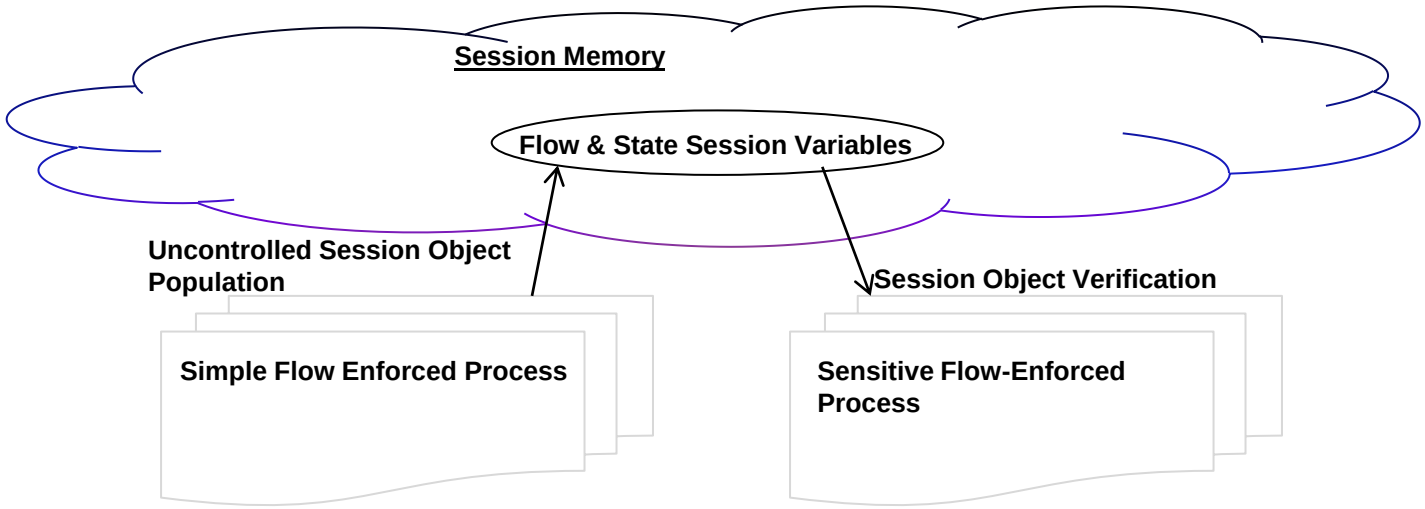
Impersonate Users via Session Puzzling

- Applications that rely on the session for storing user identities can be misled by malicious users that “overrun” their own identifying values with those of other users, through the use of modules that temporarily populate the session with client-originating identity values.



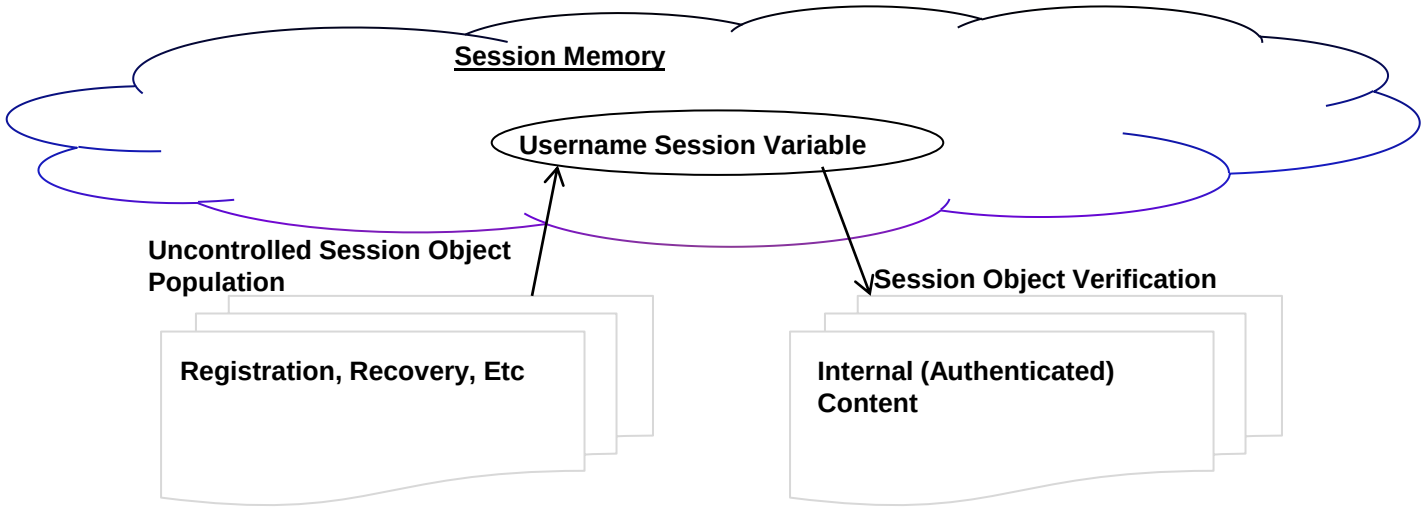
Flow Bypass via Session Puzzling

- Flow enforcement mechanisms (in processes such as password recovery, registration and transactions) that rely on identical session flags, can be bypassed by activating the processes simultaneously (for example, performing the registration process in parallel to the password recovery or transaction, to enable “skipping” phases).



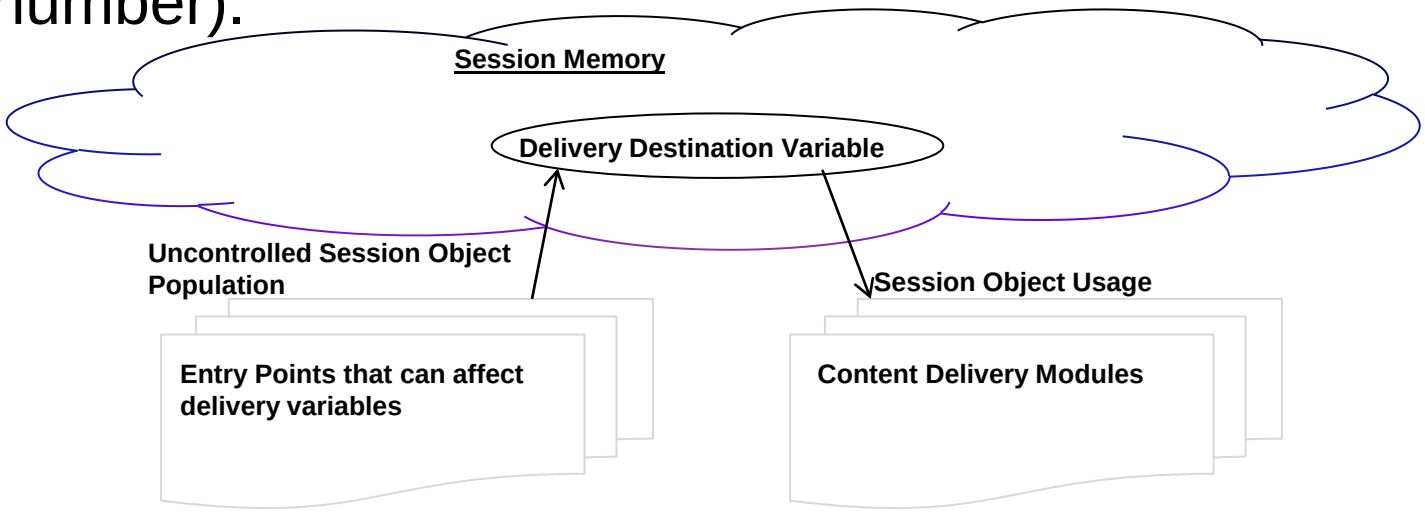
Privilege Escalation via Session Puzzling

- Attackers might be able to elevate their privileges in the application by accessing entry points that populate their session memory with additional values, permissions and flags, which might be required by other modules that were previously inaccessible.



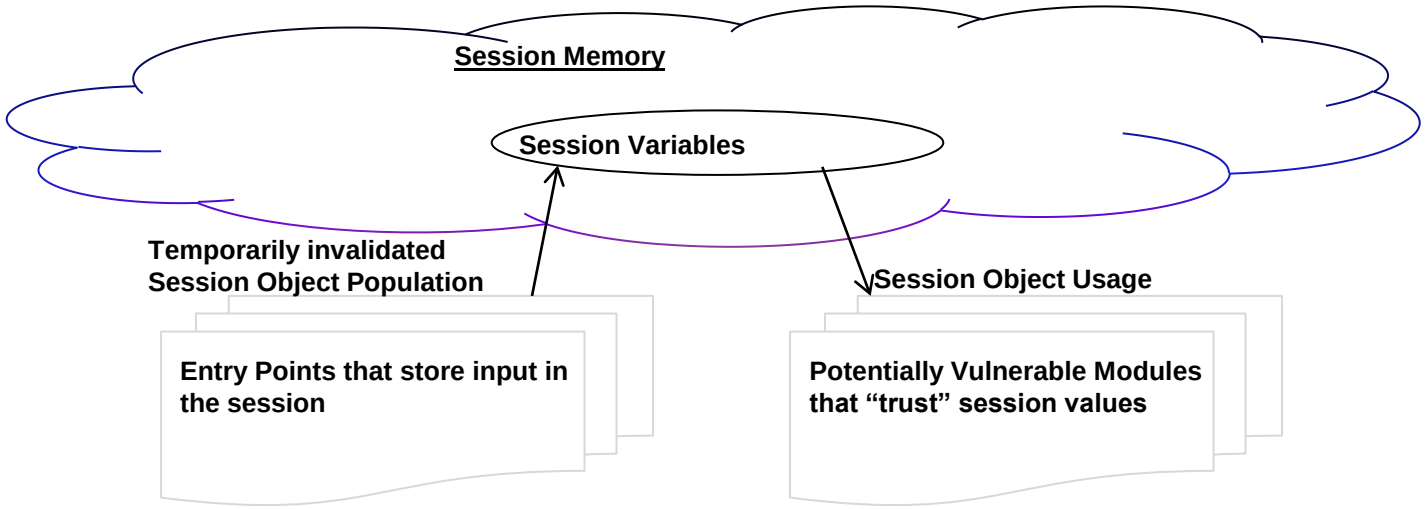
Content Theft via Session Puzzling

- Applications use a variety of content delivery methods to keep in touch with their consumers (SMS, email, etc.). Attackers can use session puzzles to initiate content delivery processes and affect their destination (for example, affect the destination of an SMS password recovery by simultaneously registering with a new number).



Indirect “Traditional” Attacks

- The same “indirect” method used in the previous instances can also be used to execute **injections**, **reflections**, **manipulations** and other “traditional” attacks in locations that were previously considered safe, simply by affecting session values which are used in entry points that treat their origin as trusted (and thus avoid validation).



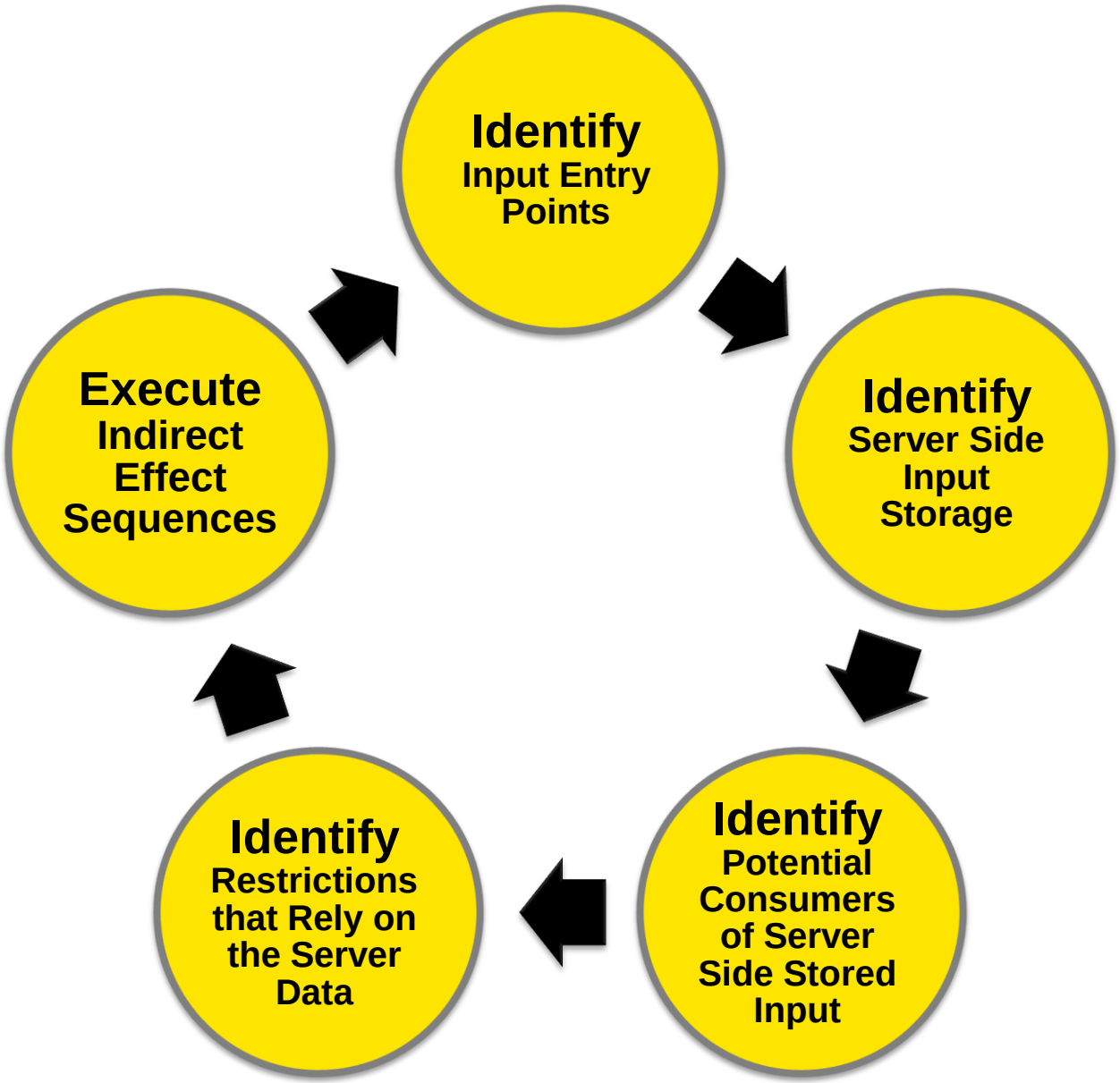
Potential Entry Points

- Login modules with premature session value population.
- Registration, password recovery and recovery challenge modules.
- Multiphase processes.
- Contact forms.
- Test pages and obsolete content.
- Security mechanisms.
- Any module that stores values in the session.
- Etc.

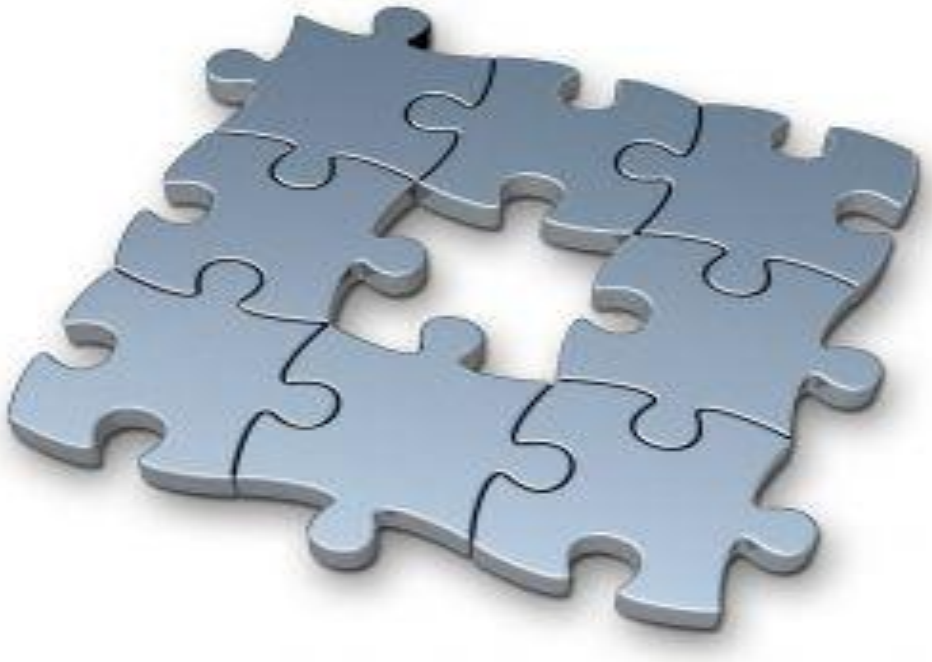
Session Puzzles FAQ

- **Should session puzzles be considered new vectors?**
  - **Yes** and **No**. It's a new way to perform unique logical attacks and an alternative method to execute traditional attack vectors.
- **How session puzzling differ from other methods?**
  - The testing perspective enables attackers to compose the attack pattern **in the back-end**.
  - The back-end stored data can be used to attack any entry point that relies on it, even if it is not affected by input.
- **Which applications might be vulnerable?**
  - Any application or system that tracks consumer "state", not just web applications.

Session Puzzling Walkthrough



Temporal Session Race Conditions



The Lifespan of Session "Leftovers"

- The lifespan of session variables might vary in the context of a module:
  - The content of the session might be initialized in the **beginning** of the module, a typical behavior in the following:
    - Logout modules
    - Login modules
  - The content of the session might be initialized at the end or the middle of the module:
    - Logout modules
    - The code sections of security mechanisms that deal with failures (including login failures, security events, etc.)
  - The entire session

The Lifespan of Session "Leftovers"

- Furthermore, in addition to the previously described scenarios, the lifespan of specific session variables might be limited in additional ways:
  - The content of a session variable might be initialized in certain phases of a multiphase process:
    - State flags
    - Variables used for calculation, identity storage, etc.
  - The content of a session variable might be initialized if a certain criteria is met (the process failed or successfully completed, exceptions did not occur, etc.).

TSRC Exploitation

- Definition: a combination of attacks meant to enhance the consistency of exploiting session-level race conditions.
  - In order to make the exploitation consistent, we will need to artificially create that which is missing… Latency.
  - Abusing the session variables will still require the exploitation request to be sent immediately after the request/s meant to populate the session and cause the latency.

Intentional Latency Increment

- The solution to exploiting session race conditions with consistency lies in extending the productive latency, artificially increasing the odds for the session manipulation success.

```
session.setAttribute(
        SessionConstants.USERNAME_VARIABLE,
        username);
        .
        .
        .
        .
        .
session.invalidate(); //invalidate session, erase all variables
```

Productive Latency

Intentional Latency Increment, Cont.

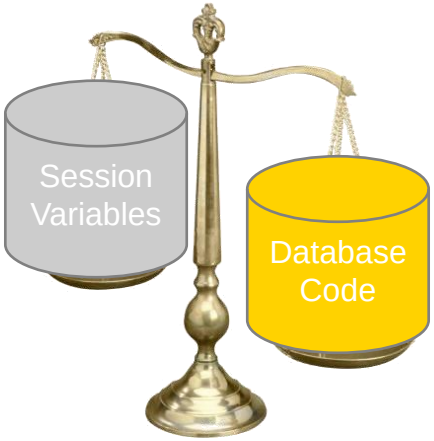
- An increment in the length of the session variable lifespan will directly increase the chances of abusing it…
- But how can we cause an increment in the execution latency of specific lines of code?

```
session.setAttribute(
        SessionConstants.USERNAME_VARIABLE,
        username);
        .
        .
        .
        .
        .
session.invalidate(); //invalidate session, erase all variables
```

++ ?

ADoS & Productive Latency

- The ADoS attack must affect the lines of code between the session population and the session invalidation **more** then it affects the rest of the code.
- For example, a denial of service attack that targets the web server is inefficient (since all the code is affected) while a denial of service attack that targets the database (and thus, the database access code) might be.



Temporal Session Race Conditions

- The unnecessary / premature session variable must be granted a lifespan long enough for bypassing the session-level validation.

```
if(session.getAttribute(
    SessionConstants.USERNAME_VARIABLE)
        == null) {
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(
        "/login.jsp");

    dispatcher.forward(request,response);
    return;
}
else {
    //perform actions
}
```

2

3

```
String username =
    request.getParameter("username");
session.setAttribute(
    SessionConstants.USERNAME_VARIABLE,
    username);
.
.
.
session.invalidate();
```

1

4

Productive Latency

Initial Samples of Layer Targeted ADoS

- RegEx DoS
  - Send Regular Expression DoS payloads to the target module, in order to increase the latency of validations that follow the session value population.
  - http://www.youtube.com/watch?v=3k_eJ1bcCro
- Connection Pool Consumption / Occupation
  - Intentionally “consume” all the available connections in the connection pool, in order to delay database operations in a target entry point.
  - http://www.youtube.com/watch?v=woWECWwrsSk

Increasing Latency with RegEx DoS

- RegEx Dos Payloads can increase the latency of validation and search mechanisms. For example:
  - RegEx: ([a-zA-Z0-9]+)*
  - Input: Admin, aaaaaaaaaaaaaaaaaaaaaaaaa!

```
String username = request.getParameter("username");
String password = request.getParameter("password");

session.setAttribute(SessionConstants.USERNAME_VARIABLE,username);

//input validation
if (!(username.matches(ValidationConstants.USERNAME_IV_REGEX)) ||
    !(password.matches(ValidationConstants.PASSWORD_IV_REGEX))){

session.invalidate(); //invalidate session, remove all variables
. . .
} else {
    ...
}
```

Occupying Connections to Increase Latency

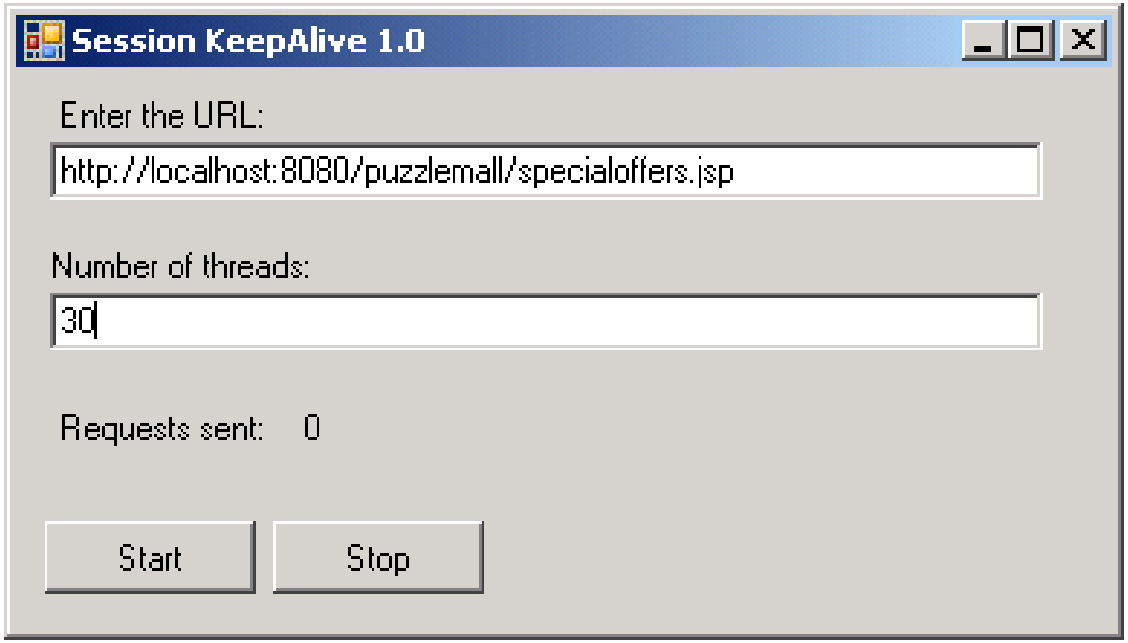
- Occupying connections will guarantee that code, which requires a database connection, will experience some latency.

```
String username =
    request.getParameter("username");
session.setAttribute(
    SessionConstants.USERNAME_VARIABLE,
    username);
.
.
Connection conn = ConnectionPoolManager.getConnection();
.
.
.
session.invalidate();
```

↑ **Delayed until a connection is released**

Occupying Connections to Increase Latency

- "Session KeepAlive" – a sample tool that can exhaust the connection pool:



Samples of Layer Targeted ADoS

- Intentional Execution of Complex Queries
  - Access entry points that execute resource-consuming queries, in order to delay the database responses.

- Shared Backend DoS
  - Perform ADoS on a web site that consumes services from a backend server shared by the target web site, effectively increasing the response time of the shared backend server.

Intentional Execution of Complex Queries

Login Module

```
String username =
    request.getParameter("username");
String password =
    request.getParameter("password");
session.setAttribute(
    SessionConstants.USERNAME_VARIABLE,
    username);

Connection conn =
    ConnectionPoolManager.getConnection();
String SqlString =
    "SELECT username FROM users " +
    "WHERE username = ? AND password = ?";

PreparedStatement pstmt =
    conn.prepareStatement(SqlString);
pstmt.setString(1, username);
pstmt.setString(2, password);

ResultSet rs = pstmt.executeQuery();
.
.
.
session.invalidate();
```

Productive Latency

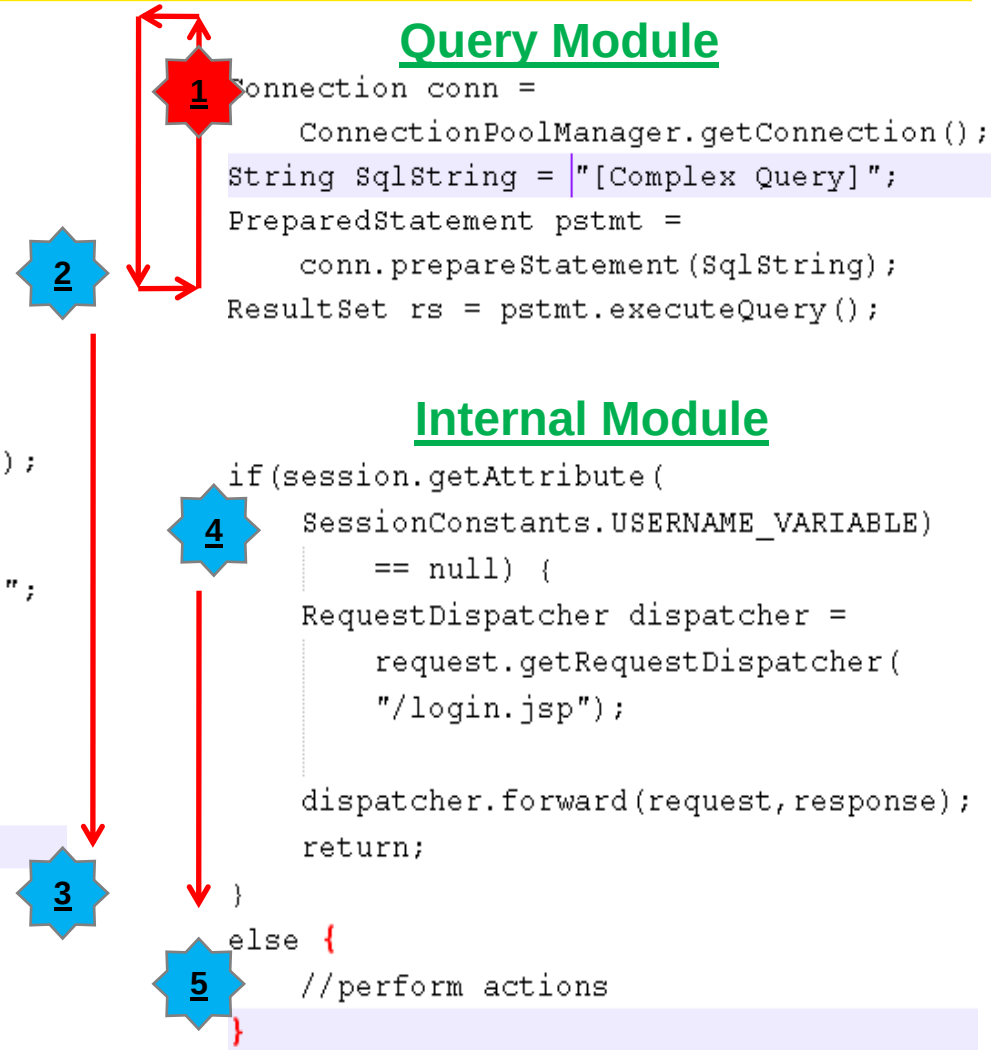
Query Module

```
Connection conn =
    ConnectionPoolManager.getConnection();
String SqlString = "[Complex Query]";
PreparedStatement pstmt =
    conn.prepareStatement(SqlString);
ResultSet rs = pstmt.executeQuery();
```

Internal Module

```
if(session.getAttribute(
    SessionConstants.USERNAME_VARIABLE)
        == null) {
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(
        "/login.jsp");

    dispatcher.forward(request,response);
    return;
}
else {
    //perform actions
}
```




The
Automation
Issue

The Numerous Potential Sequences

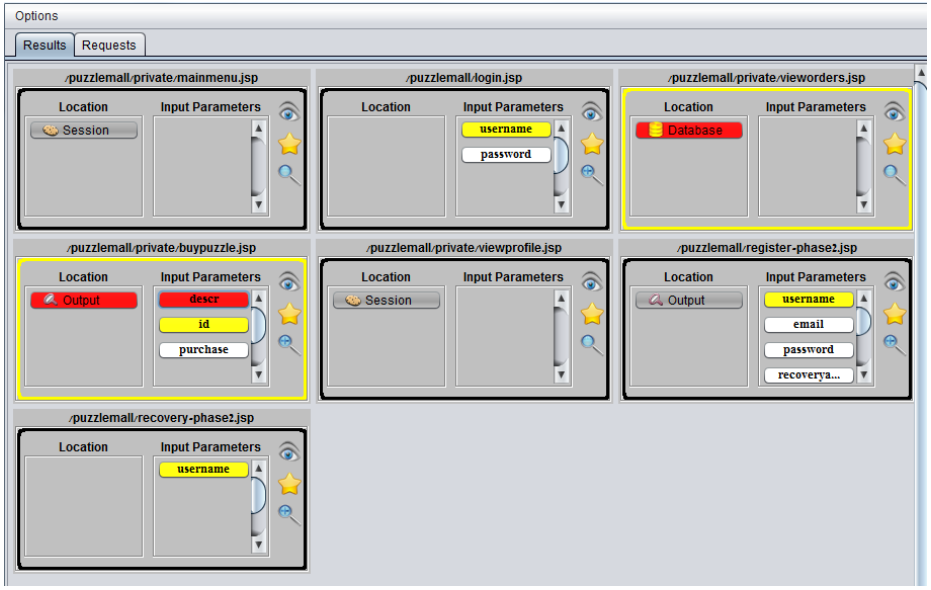
- The number of potential vectors to test can become overwhelming
  - Different Sequences
  - Different Inputs
  - Authentication Requirements
  - Token Requirements
  - Process Dependencies
  - Deprecated Values

Introducing

Diviner

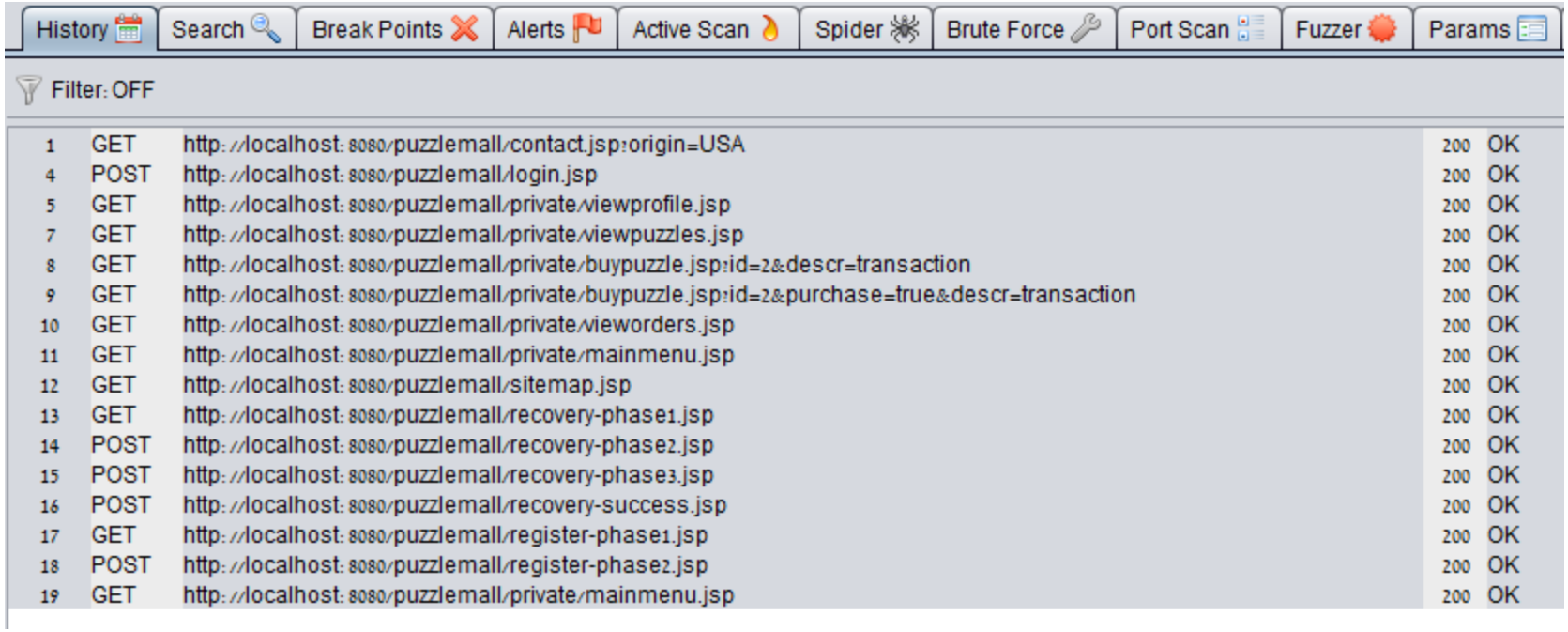
An Active Information Gathering Framework

Predicting Server-Side Content-Storage Structure and Effect



https://code.google.com/p/diviner/

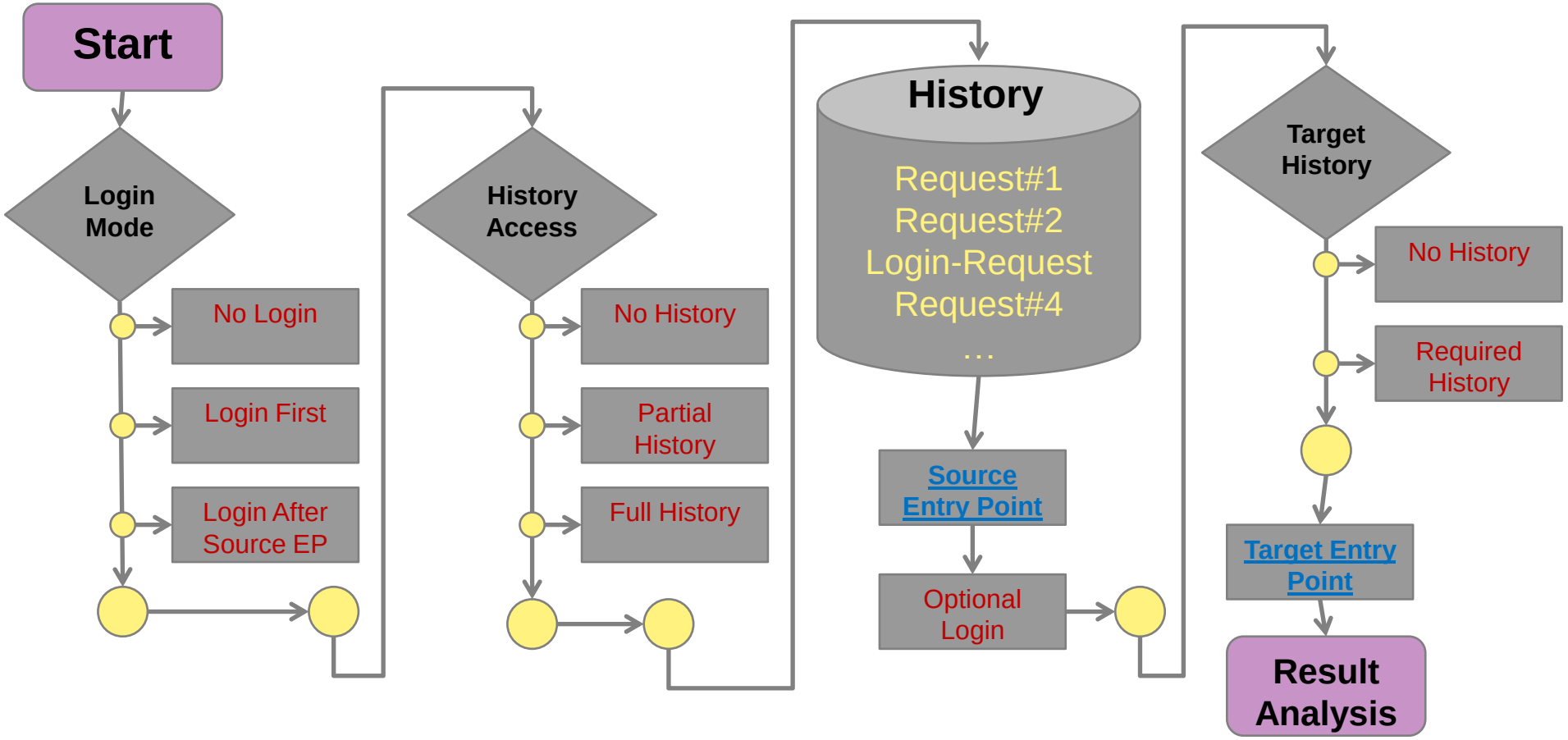
ZAP's Request History



1	GET	http://localhost:8080/puzzlemall/contact.jsp?origin=USA	200	OK
4	POST	http://localhost:8080/puzzlemall/login.jsp	200	OK
5	GET	http://localhost:8080/puzzlemall/private/viewprofile.jsp	200	OK
7	GET	http://localhost:8080/puzzlemall/private/viewpuzzles.jsp	200	OK
8	GET	http://localhost:8080/puzzlemall/private/buypuzzle.jsp?id=2&descr=transaction	200	OK
9	GET	http://localhost:8080/puzzlemall/private/buypuzzle.jsp?id=2&purchase=true&descr=transaction	200	OK
10	GET	http://localhost:8080/puzzlemall/private/vieworders.jsp	200	OK
11	GET	http://localhost:8080/puzzlemall/private/mainmenu.jsp	200	OK
12	GET	http://localhost:8080/puzzlemall/sitemap.jsp	200	OK
13	GET	http://localhost:8080/puzzlemall/recovery-phase1.jsp	200	OK
14	POST	http://localhost:8080/puzzlemall/recovery-phase2.jsp	200	OK
15	POST	http://localhost:8080/puzzlemall/recovery-phase3.jsp	200	OK
16	POST	http://localhost:8080/puzzlemall/recovery-success.jsp	200	OK
17	GET	http://localhost:8080/puzzlemall/register-phase1.jsp	200	OK
18	POST	http://localhost:8080/puzzlemall/register-phase2.jsp	200	OK
19	GET	http://localhost:8080/puzzlemall/private/mainmenu.jsp	200	OK

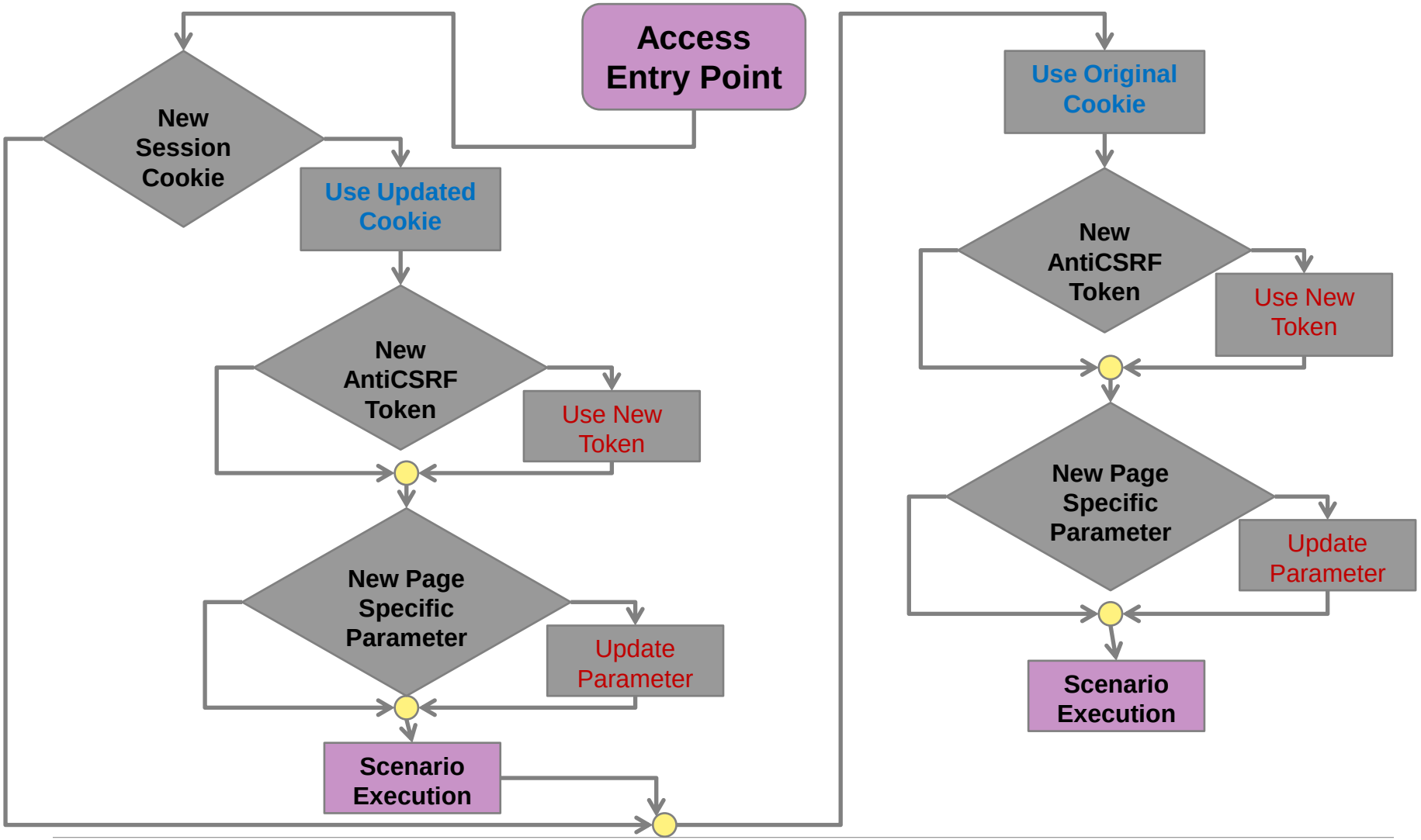
Exploring Different Paths of Execution

Behavior in Different Authentication Modes and History Perquisites



Exploring Different Paths of Execution

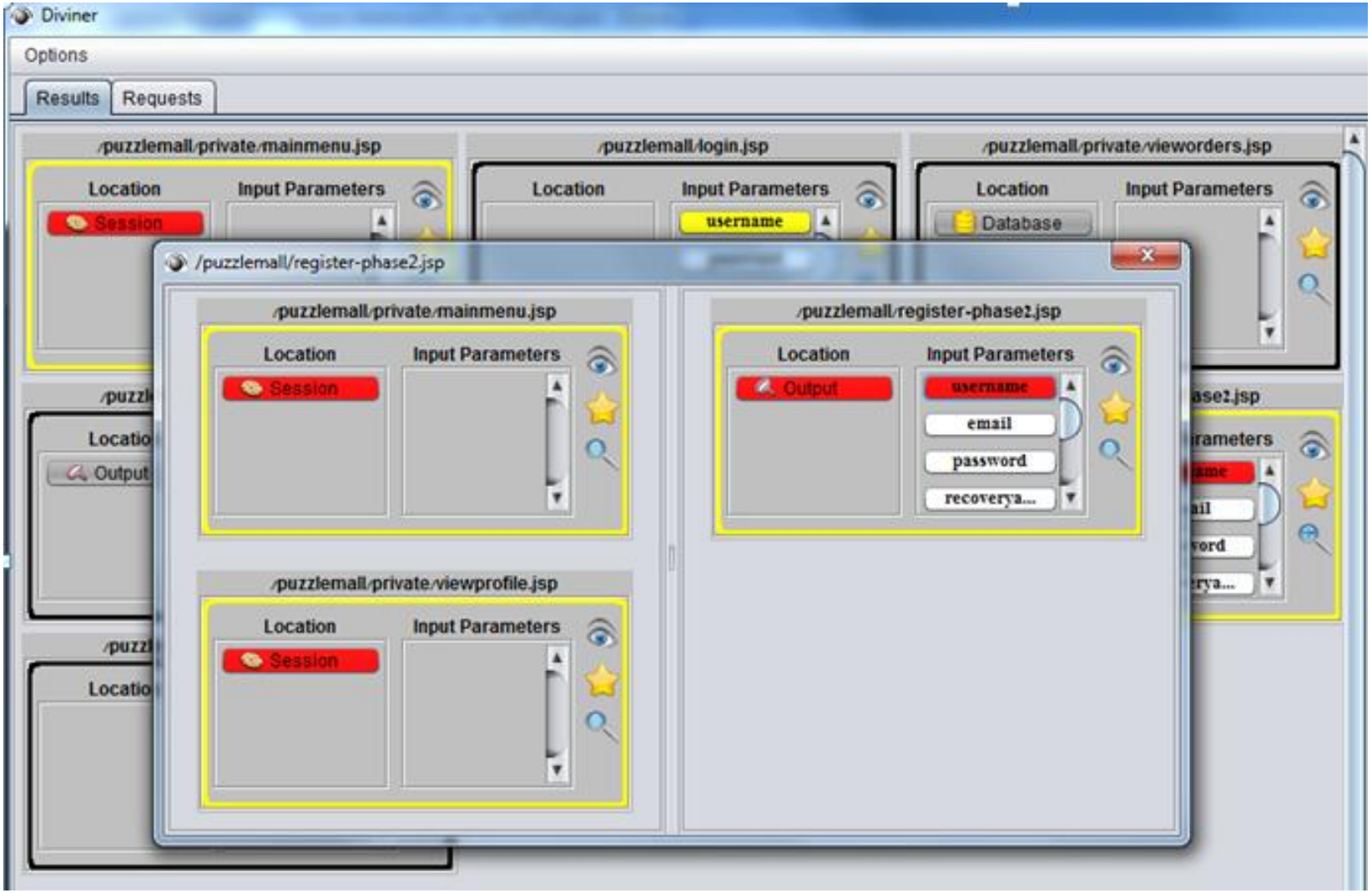
Behavior With Different Session Cookies, Identifiers and Tokens



Behavior Isolation

1	Input Reflected from Variable
2	Input Reflected from Session
3	Input Reflected from Database
4	Input Stored in Server Variable
5	Input Stored in Session Variable
6	Input Stored in Database Table
7	New Cookie Value
...	...

Visual Input/Output/Effect Correlation

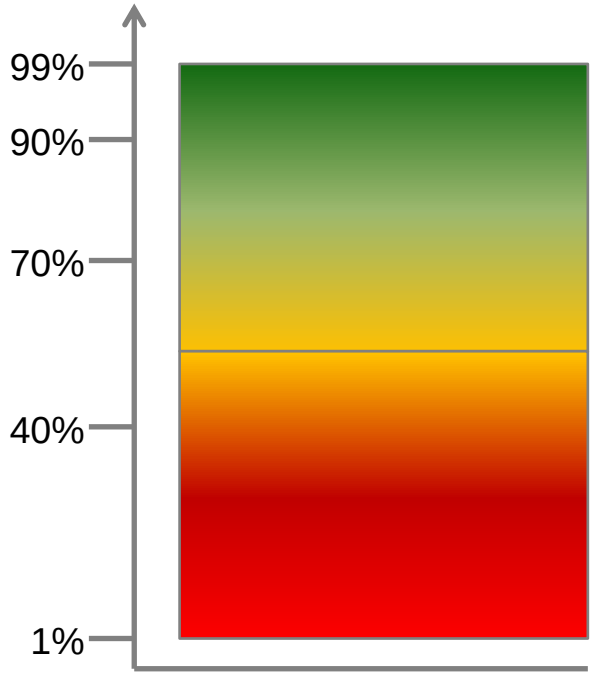


Source Code Divination Accuracy

1	Read Input to Variable	String input$$1$$ = request. getParameter(##1##);	String input$$1$$ = Request[“##1##”];	
2	Invalidate Session	session.invalidate();	Session.Abandon();	
3	New Session Identifier	request.getSession(true);	…	
4	New Cookie Value	Cookie cookie = new Cookie ("##1##",val); response.addCookie(cookie);	Response.Cookies("##1## ").Value = "val";	
5	Get Database Connection	Class.forName(DriverClassName); Connection conn = DriverManager.getConnection(X);	SqlConnection conn = new SqlConnection(X);	
…	…	…	…	…

Source Code Divination Accuracy

7	3	1	1010	50%
7	4	1	10040	70%
7	2	2	5550	40%
6	1	1	2010	90%
6	5	2	10000	80%
...	...	...	...	...

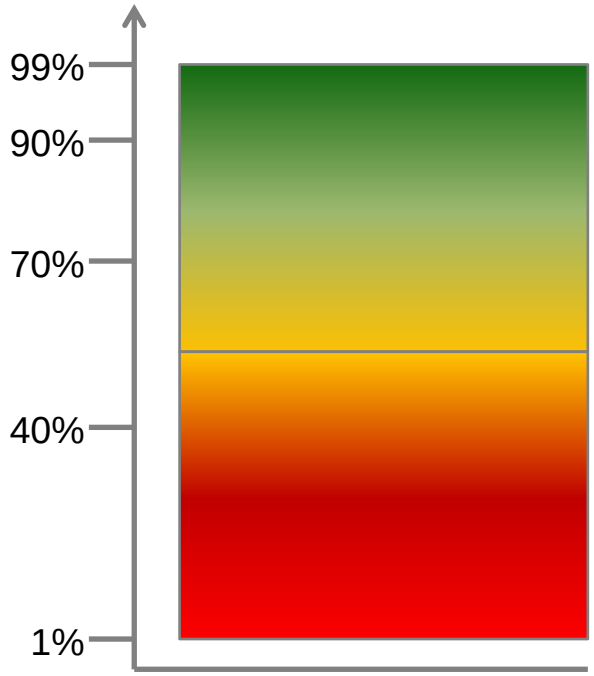


Verification Process and Probability

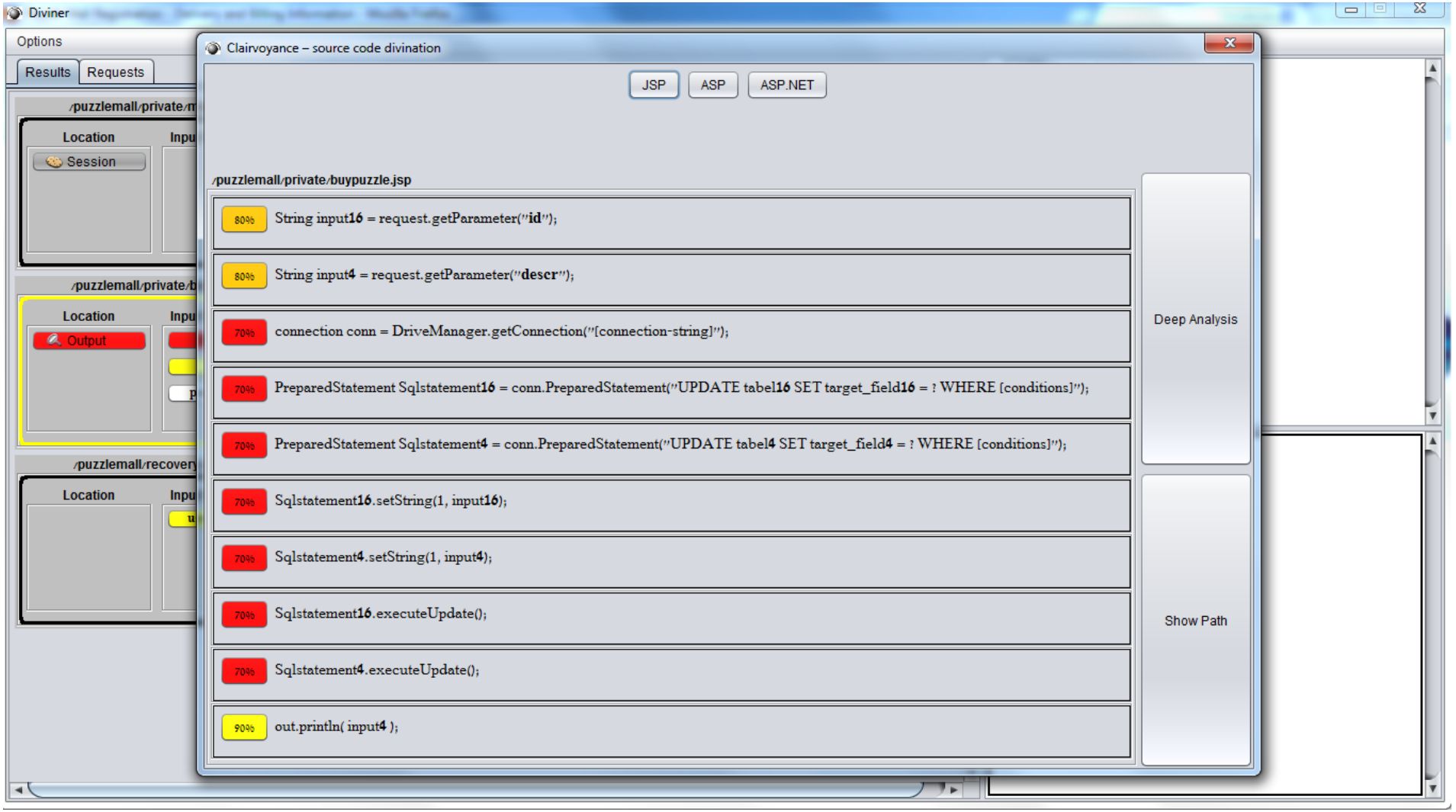
For **each** unique entry point / request, the probability for the existence of specific lines of code is adjusted according to the results of various behavior specific confirmation processes.

Previous session redirects to login after set-cookie instruction?
Behaviour7 -> CodeId2 +40%, CodeId3 +20%, CodeId4 -10%

7	3	1	1010	70%
7	4	1	10040	60%
7	2	2	5550	80%
6	1	1	2010	90%
6	5	2	10000	80%
...	...	...	...	...



Source/Target Code Correlation



Risk Mitigation

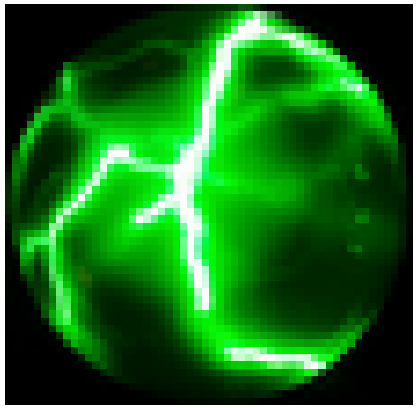
Session Puzzling & TSRC Mitigation

- Avoid storing unnecessary values in the session.
- Avoid using session variables with identical names in different modules, multiphase processes, and particularly in public vs. private entry points.
- Store **objects** in the session instead of variables. The name of the objects should include the origin process / module.
- Don’t use the session as a temporary container for values.
- Perform validations on session originating values before using them in the application code.

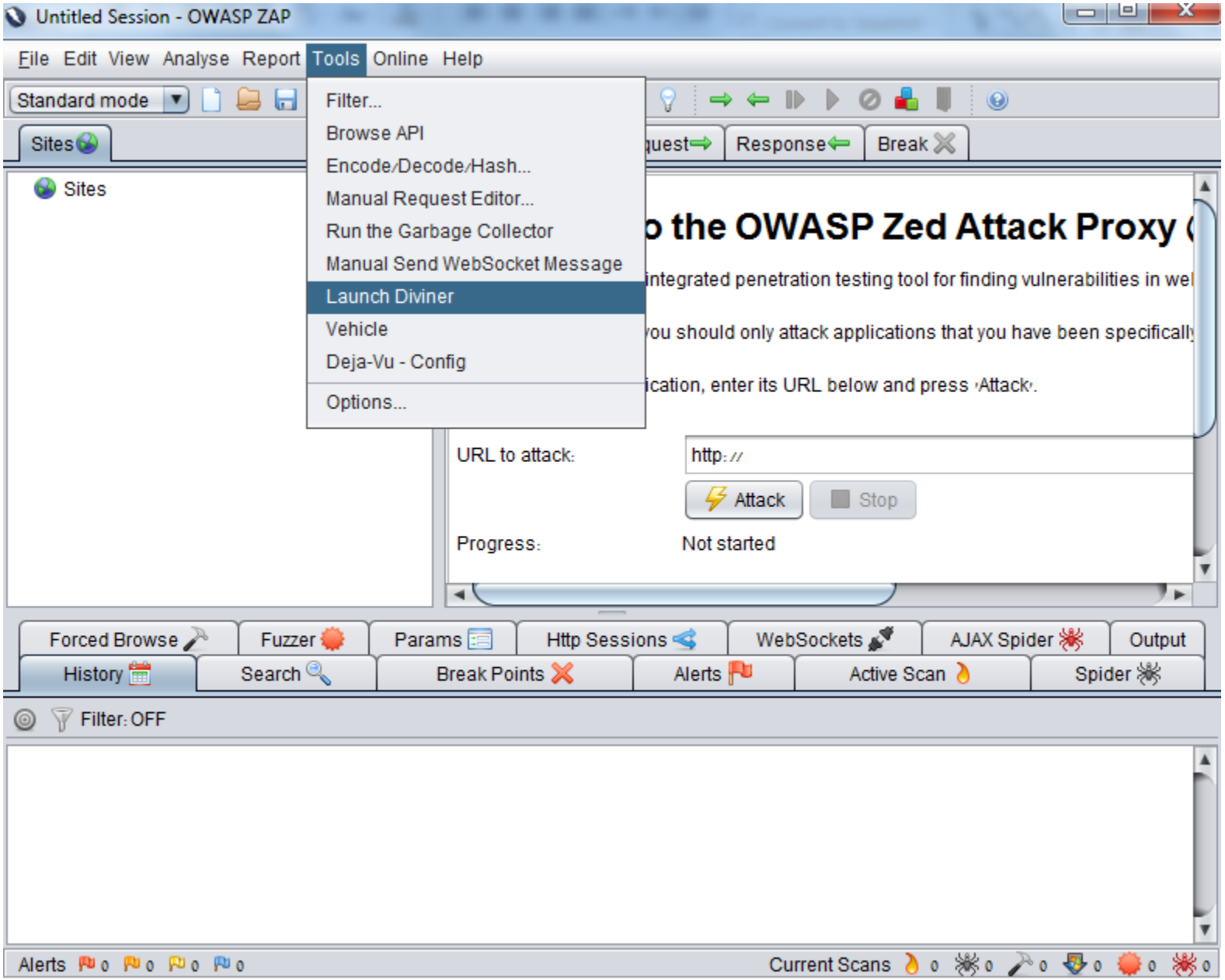
SUMMARY

The Diviner Project

- **Homepage:**https://code.google.com/p/diviner/
- OWASP ZAP extension (v2.0+), requires Java 1.7



Activating the Diviner Extension in ZAP

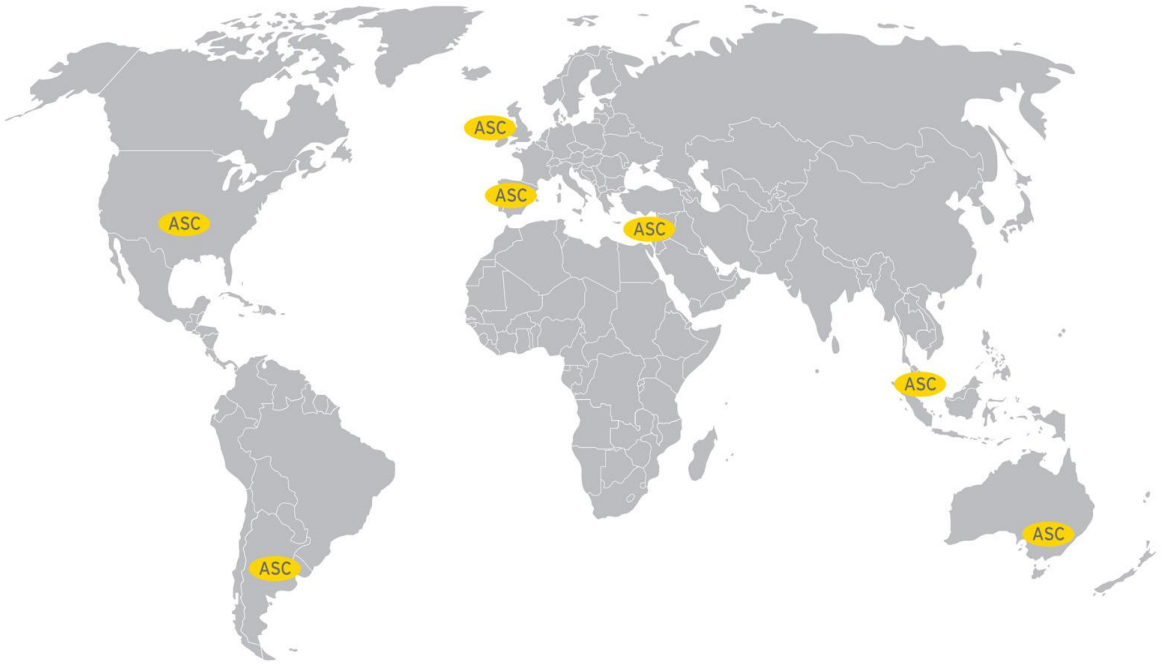


Additional Resources

- **Session Puzzling Original Concept:** Whitepaper
- **Session Puzzling Demo Videos:** Hacktics Youtube Channel, Oracle E-Business Suite SP Demo
- **OWASP ZAP:** https://code.google.com/p/zaproxy/
- **OWASP** Classification**:** Session Variable Overloading
- **Training/Testing Platforms:** Puzzlemall
- **Posts** on session puzzling / session race conditions: Articles, Presentation 1 (PHP), Presentation 2
- **Posts** on divination attacks and structure prediction: Articles, Presentations, Videos

EY Advanced Security Centers

- **Americas**
  - Hacktics IL
  - Houston
  - New York
  - Buenos Aires
- **Asia Pacific**
  - Melbourne
  - Singapore
- **EMEIA**
  - Dublin
  - Barcelona



QUESTIONS?

Shay Chen (@sectooladdict)

EY
Building a better working world