THE BOOMERANG EFFECT

Using Session Puzzling to Attack Apps from the Backend

Shay Chen, CTO
@sectooladdict

Hacktics ASC, Ernst & Young

November 22nd, 2013
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
The process populates the session memory with the username value…
The attacker directly access an internal page that relies on the session-stored username variable.
Traditional Attack Vectors
“Traditional” Application Attack Vectors

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

**DeepSec**
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.
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
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

Web Server

Initial Access to the Domain

Set-cookie: SID=abcd123
Cookie: SID=abcd123

Domain Cookie
SID=Abcd123

Session Memory

Session ID
Abcd123
Cbcr321

Memory Allocation
0xAA...
0xBB...

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.
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 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.).
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.
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

1. Identify Input Entry Points
2. Identify Server Side Input Storage
3. Identify Potential Consumers of Server Side Stored Input
4. Identify Restrictions that Rely on the Server Data
5. Execute Indirect Effect Sequences
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
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.
The solution to exploiting session race conditions with consistency lies in extending the productive latency, artificially increasing the odds for the session manipulation success.

```java
session.setAttribute(
    SessionConstants.USERNAME_VARIABLE,
    username);

// invalidate session, erase all variables
session.invalidate();
```
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?

```java
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 \textbf{more} than 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.
The unnecessary / premature session variable must be granted a lifespan long enough for bypassing the session-level validation.

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

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

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

// productive latency

session.invalidate();
```
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, aaaaaaaaaaaaaaaaaaaaaaaaaa!

```java
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.

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

Connection conn = ConnectionPoolManager.getConnection();

↑ Delayed until a connection is released

session.invalidate();
```
“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 strSqlString = "SELECT username FROM users " +
    "WHERE username = ? AND password = ?";

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

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

Query Module

Connection conn = ConnectionPoolManager.getConnection();
String strSqlString = "[Complex Query]";
PreparedStatement pstmt =
    conn.prepareStatement(strSqlString);
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

[Image of Diviner framework]

https://code.google.com/p/diviner/
ZAP’s Request History

<table>
<thead>
<tr>
<th></th>
<th>Method</th>
<th>URL</th>
<th>Status Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/contact.jsp?origin=USA">http://localhost:8080/puzzlemail/contact.jsp?origin=USA</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>POST</td>
<td><a href="http://localhost:8080/puzzlemail/login.jsp">http://localhost:8080/puzzlemail/login.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/viewprofile.jsp">http://localhost:8080/puzzlemail/private/viewprofile.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>7</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/viewpuzzles.jsp">http://localhost:8080/puzzlemail/private/viewpuzzles.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>8</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/buypuzzle.jsp?id=2&amp;descr=transaction">http://localhost:8080/puzzlemail/private/buypuzzle.jsp?id=2&amp;descr=transaction</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>9</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/buypuzzle.jsp?id=2&amp;purchase=true&amp;descr=transaction">http://localhost:8080/puzzlemail/private/buypuzzle.jsp?id=2&amp;purchase=true&amp;descr=transaction</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>10</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/vieworders.jsp">http://localhost:8080/puzzlemail/private/vieworders.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>11</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/mainmenu.jsp">http://localhost:8080/puzzlemail/private/mainmenu.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>12</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/sitemap.jsp">http://localhost:8080/puzzlemail/sitemap.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>13</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/recovery-phase1.jsp">http://localhost:8080/puzzlemail/recovery-phase1.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>14</td>
<td>POST</td>
<td><a href="http://localhost:8080/puzzlemail/recovery-phase2.jsp">http://localhost:8080/puzzlemail/recovery-phase2.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>17</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/register-phase1.jsp">http://localhost:8080/puzzlemail/register-phase1.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>18</td>
<td>POST</td>
<td><a href="http://localhost:8080/puzzlemail/register-phase2.jsp">http://localhost:8080/puzzlemail/register-phase2.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>19</td>
<td>GET</td>
<td><a href="http://localhost:8080/puzzlemail/private/mainmenu.jsp">http://localhost:8080/puzzlemail/private/mainmenu.jsp</a></td>
<td>200</td>
<td>OK</td>
</tr>
</tbody>
</table>
Exploring Different Paths of Execution

Behavior in Different Authentication Modes and History Perquisites

Start

Login Mode
- No Login
- Login First
- Login After Source EP

History Access
- No History
- Partial History
- Full History

History
- Request#1
- Request#2
- Login-Request
- Request#4
- ...

Source Entry Point

Optional Login

Target History
- No History
- Required History

Target Entry Point

Result Analysis
Exploring Different Paths of Execution

Behavior With Different Session Cookies, Identifiers and Tokens

- **New Session Cookie**
  - Use Updated Cookie
    - New AntiCSRF Token
      - Use New Token
        - New Page Specific Parameter
          - Update Parameter
            - Scenario Execution
- **Access Entry Point**
  - Use Original Cookie
    - New AntiCSRF Token
      - Use New Token
        - New Page Specific Parameter
          - Update Parameter
            - Scenario Execution

- **Use New Token**
- **Update Parameter**
### Behavior Isolation

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input Reflected from Variable</td>
</tr>
<tr>
<td>2</td>
<td>Input Reflected from Session</td>
</tr>
<tr>
<td>3</td>
<td>Input Reflected from Database</td>
</tr>
<tr>
<td>4</td>
<td>Input Stored in Server Variable</td>
</tr>
<tr>
<td>5</td>
<td>Input Stored in Session Variable</td>
</tr>
<tr>
<td>6</td>
<td>Input Stored in Database Table</td>
</tr>
<tr>
<td>7</td>
<td>New Cookie Value</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Visual Input/Output/Effect Correlation
### Source Code Divination Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>ID</th>
<th>Description</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read Input to Variable</td>
<td>String input$$1$$ = request.getParameter(&quot;##1##&quot;);</td>
<td>String input$$1$$ = Request[&quot;##1##&quot;];</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Invalidate Session</td>
<td>session.invalidate();</td>
<td>Session.Abandon();</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>New Session Identifier</td>
<td>request.getSession(true);</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>New Cookie Value</td>
<td>Cookie cookie = new Cookie(&quot;##1##&quot;, val); response.addCookie(cookie);</td>
<td>Response.Cookies(&quot;##1##&quot;).Value = &quot;val&quot;;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Get Database Connection</td>
<td>Class.forName(DriverClassName); Connection conn = DriverManager.getConnection(X);</td>
<td>SqlConnection conn = new SqlConnection(X);</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Source Code Divination Accuracy

<table>
<thead>
<tr>
<th>Code</th>
<th>ID</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>50%</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>70%</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>40%</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>90%</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>80%</td>
</tr>
</tbody>
</table>

*Note: The code ID and probability are shown in the table.*
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? **Behaviour 7 -> Codeld2 +40%, Codeld3 +20%, Codeld4 -10%**

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>CodeId</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1010</td>
<td>70%</td>
</tr>
<tr>
<td>7</td>
<td>10040</td>
<td>60%</td>
</tr>
<tr>
<td>7</td>
<td>5550</td>
<td>80%</td>
</tr>
<tr>
<td>6</td>
<td>2010</td>
<td>90%</td>
</tr>
<tr>
<td>6</td>
<td>10000</td>
<td>80%</td>
</tr>
</tbody>
</table>

...
Source/Target Code Correlation

```
670b  String input16 = request.getParameter("id");
670b  String input4, request.getParameter("desc");
700b  connection conn = DriverManager.getConnection("(connection-string)");
700b  PreparedStatement Sqlstatement16 = conn.prepareStatement("UPDATE tbl16 SET target_field16 = ? WHERE (conditions)");
700b  PreparedStatement Sqlstatement4  = conn.prepareStatement("UPDATE tbl4 SET target_field4 = ? WHERE (conditions)");
700b  Sqlstatement16.setString(1, input16);
700b  Sqlstatement4.setString(1, input9);
700b  Sqlstatement16.executeUpdate();
700b  Sqlstatement4.executeUpdate();
700b  out.println(input5);
```
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/](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/](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)