Architectural Design Patterns for SSO (Single Sign On)
Design and Use Cases for Financial Web Applications

Wei Zhang & Marco Morana
OWASP Cincinnati, U.S.A.

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

http://www.owasp.org
Presentation outline

- The needs/challenges for securing SSO architectures in financial web applications
- Secure SSO design patterns
- Security consideration and risk for SSO design
- Secure architecture principles
- Architectural diagrams
- Data flow diagrams
- Sequence diagrams
- Threat models of SSO architectures
- Attack trees and misuse cases
- Security risk framework for secure design of SSO architectures
Single Sign On (SSO) needs for financial web systems

- Different systems serving different functions
  - ATM cash withdraw
  - Branch deposit
  - Monthly statements
  - Make a payment with a check
  - Wire transfer

- Different systems have different type of user’s information
  - Personal and sensitive information
  - Financial transactions
  - Bank statements

- Different business features are available for each system
  - Saving/checking accounts
  - Credit card accounts
  - Mortgage applications or loans
  - Cash rewards
  - Mileage rewards
Requirements for SSO

- User friendly is the key
- Single view is the goal
- Eliminate additional sign on is the approach
- Security is the foundation
SSO Use Case

- User can sign on site A to do function B about product C.
- User can sign on site X to do function Y about product Z.
- For users with both product C and Z, user should be able to sign on from Site A to access function B and function Y without additional authentication.
- Site X will not be sunset to support users with only product Z.
Design options for SSO

- Duplicate function Y on site A and access information on site X
  - Pros:
    - Make it possible to sunset site X
  - Cons:
    - Introduce duplicated function on two sites
    - Needs to maintain the function and processing rules on two sites.

- Build SSO for user to access function Y on site X
  - Pros:
    - No need to maintain two sets of function and processing rules on two sites
    - Enable the possibility to fully leverage functions on site X
  - Cons:
    - Make site A depends on site X
    - Introduce security complexity:
      - Authentication
      - Authorization
      - Session coordination
      - UI
SSO Design Patterns

- **Ad-hoc Encrypted Token:**
  Use symmetric and public key cryptography to encrypt the application data that used for SSO.

- **Standard Secure Token Service (STS):**
  Central Security Token Service to respond with standard SAML token that supports federation across different sites upon request.
SSO - Ad-Hoc Encrypted Token

■ Pros:
  ▸ Easy to set up and implement
  ▸ Supports PKI and SAML
  ▸ No dependency on other system.

■ Cons:
  ▸ Not a unified solution
  ▸ Each federated site has to manage cryptographic key
SSO - Ad-hoc encrypted token

Customer

Internet

SSO with encrypted token

Decrypt token and authenticate

Site A

Application Server

Sign on

Secure token for SSO

Key store

Credential store

Site X

Application Server

Key store

User information

OWASP 9
SSO - STS

**Pros:**
- Drives to unified solution for both internal and external communication.
- Centralized cryptographic key management approach.
- Supports SAML and SOAP

**Cons:**
- Introduces dependences on STS for federated sites
- Introduces additional internet hop for communication
SSO - STS

Customer

Internet

SSO with SAML token

Site A

Application Server

Credential store

Key store

STS

Site X

Application Server

Key store

User information

Sign on

SAML token for SSO

Varify SAML token/authenticate

OWASP 11
SSO design and security considerations

- Secure authentication and authorization
  - Ad-hoc encrypted token
  - STS

- Secure session management: one dies, both die
  - Session initiation
  - Session termination
  - Session recovery
  - Keep alive

- Secure web page wrapping: look and feel of the site
  - iFrames
Potential Security Issues with SSO Design

- In-Secure Session Management:
  - Sessions are not sync: one dies, one left open
  - Session Replay
  - Session Riding (CSRF)
  - Session hijacking
  - Sessions un-protected/in clear, cached, logged

- Malicious Data Injections:
  - XSS, SQL Injection

- Elevation of Privileges, Bypass of Authentication
  - Bypass authorizations
  - Forceful browsing
Secure Architecture Design

General Security Design Principles

1. Implement Authentication With Adequate Strength
2. Enforce Least Privilege
3. Protect Data In Storage, Transit And Display
4. Enforce Minimal Trust
5. Trace and Log User Actions And Security Events
6. Fail Securely And Gracefully
7. Apply Defense in Depth
8. Apply Security By Default
9. Design For Simplicity, KISS Principle
10. Secure By Design, Development and Deployment
11. Secure As The Weakest Link
12. Avoid Security By Obscurity
Security Controls Design Guidelines: Authentication and Authorization

■ Authentication, What, Where and How
  ▸ Mandatory to restrict access to validated users
  ▸ Strength depends on application risk/data classification
  ▸ Compliant with regulations/standards
  ▸ Provide for secure password and account management
  ▸ Mitigates brute forcing and credentials harvesting
  ▸ Mitigates Man In The Middle Attacks (MiTM)
  ▸ Provides for user and host to host authentication

■ Authorization Most Common Flaws
  ▸ Flaws in Role Base Access Controls (RBAC)
  ▸ Flaws allow for horizontal and vertical privilege escalation
  ▸ Forceful browsing
Security Controls Design Guidelines: Session Management

Avoid common session management flaws:

- Session cookies and authentication tokens unprotected (e.g. clear text) between client and server
- Missing session invalidation at idle-time out and user logout
- Missing re-issuance of new session token to prevent re-use for authentication
- Un-secure storage in a session store in clear text
- Lack of strong random generation of session cookies/identifiers (e.g. >128 bit)
- Lack of coordinated session between application tiers
Security Controls Design Guidelines: Input Validation

- What and where to validate
  - Type, format, range and length of input data
  - Wherever data crosses system or trust boundaries
  - Input and output
  - Client validation vs. server validation

- How to validate
  - Constraint, Reject, Sanitize
  - Canonical Validation
  - Encoding
  - Integrity Checks
Sequence diagram of SAML SSO

1. Request target resource
2. Redirect to SSO Service
3. Request SSO Service
4. (Identify the user)
5. Request Assertion Consumer Service
6. Redirect to target resource
7. Request target resource
8. Respond with requested resource
Threat Modeling Web Applications

Attack Trees - Online Banking Applications

Analyzing the Security of Internet Banking Authentication Mechanisms:
http://www.itgi.org/Template.cfm?Section=Home&CONTENTID=35743&TEMPLATE=/ContentManagement/ContentDisplay.cfm
Use and Misuse Case of Authentication

From OWASP Security Testing Guide
### Security risk framework for secure design of SSO architectures

<table>
<thead>
<tr>
<th>Threat Agents</th>
<th>Misuses and Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Security Controls/Countermeasures</th>
<th>Technical Impacts</th>
<th>Business Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users, Customers/ Employees</td>
<td>User logs out from one application and forget to log out to another application that SSOs into it</td>
<td>Inherent weaknesses in synchronizing sessions among applications</td>
<td>Single Logout Among Applications, Keep-alives</td>
<td>Loss of sensitive/confidential data</td>
<td>Reputation loss. Unlawful compliance fines</td>
</tr>
<tr>
<td>Malicious Users,Fraudsters</td>
<td>Victim is targeted by phishing, download of malware</td>
<td>Social Engineering, Web Application Vulnerabilities, XSS</td>
<td>Consumer Education, Data Filtering, escape all un-trusted data based on HTML content</td>
<td>Execute JS on client, install malware</td>
<td>Fraud, financial losses, reputation loss/defacements</td>
</tr>
<tr>
<td>Malicious Users, Fraudsters</td>
<td>Attacker sends malicious data to the application</td>
<td>Input Validation Vulnerabilities: XSS, SQL Injection</td>
<td>Filtering, parameterized API, ESAPI filtering APIs, white-list validations</td>
<td>Loss of data, data alteration, denial of service/access</td>
<td>Public disclosure of vulnerability Reputation damage</td>
</tr>
<tr>
<td>Malicious Users, Fraudsters</td>
<td>Attacker target design flaws in the SSO/authentication or session management functions</td>
<td>Weak Auth and Session Mgmt Vulnerabilities</td>
<td>Follow Security Requirements For Secure Password Policies, Implement Account Locking, Disable “Auto-logons”</td>
<td>Unauthorized access to data, functions</td>
<td>Legal and financial implications</td>
</tr>
<tr>
<td>Fraudsters</td>
<td>Attacker creates forged HTTP requests and tricks a victim into submitting them</td>
<td>Cross Site Request Forgery Vulnerabilities</td>
<td>Include the unique token in a hidden field.</td>
<td>Can change data and functions on behalf of the user</td>
<td>Fraud, revenue loss because of denial of access</td>
</tr>
<tr>
<td>Automated Scripts/Spam Bots</td>
<td>Attacker uses a bot/script to attack the application for denial of service and harvesting</td>
<td>Insufficient Anti-Automation protection</td>
<td>Include CAPTCHA, ESAPI intrusion detection APIs</td>
<td>Can overflow/deny service to process spam data, harvest accounts/data</td>
<td>Loss due to business Disruptions/losses, reputational damage</td>
</tr>
</tbody>
</table>
References

- OpenSAML:
  https://spaces.internet2.edu/display/OpenSAML/Home
- ESAPI:
- Improving Web Application Security: Threats and Countermeasures
- Analyzing the Security of Internet Banking Authentication Mechanisms
  http://www.itgi.org/Template.cfm?Section=Home&CONTENTID=35743&TEMPLATE=/ContentManagement/ContentDisplay.cfm
- OWASP Security Testing Guide
THANK YOU!

■ Q&A

■ Contact:
  ▶ 张炜: zhangwei@hotmail.com
  ▶ Marco Morana: marco.m.morana@gmail.com