Automatic Analysis of Browser-based Security Protocols

Avinash Sudhodanan
Alessandro Armando (FBK, coordinator)
Roberto Carbone (FBK, tutor)
Luca Compagna (SAP, tutor)

FP7-PEOPLE-2012-ITN
Outline

• Context
• Problem Overview
• State of the art
• Proposed Approach
• Conclusion and Future Work
Web Authentication Schemes & Single Sign-On

**Web Authentication**
- Username or email
- Password
- Remember me
- Login
- I forgot username or password
- OR
- Login with Twitter
- Login with Facebook
- Login with LinkedIn
- Login with Google

**Single Sign-On (SSO)**
- Login with PayPal
- Sign in with LinkedIn
- Facebook Login
  - 250+ Million users, 2,000,000 websites
- OpenID
  - One billion users, 50,000
Integration of third-party Web services
Analysis of Security Protocols

• Current protocol analysis technique: Verification of design-level protocol specification
• But.. security relies on the IMPLEMENTATION
Secure Implementation

• Provide secure implementation guidelines
  – Sign in with LinkedIn
    - FAQ
      Q: Why bother with signature validation? What's the point?
      A: The main purpose of the credentials_cookie feature is to communicate, securely, two pieces of information from the user's browser to your application: an access_token and LinkedIn member_id for that user.
  – Facebook Login
    - https://developers.facebook.com/docs/facebook-login/security/
    - Security Checklist
      This list below should be considered the absolute minimum that all apps using Facebook Login should implement. Other features will be unique to your app and you will need to always think about how to make your app as secure as possible. Apps that are not secure will lose the trust of their audience and people will stop using them.
      - Never include your App Secret in client-side or decompilable code.
      - Sign all server-to-server Graph API calls with your App Secret.
Web Security: Current solutions

• Follow secure implementation guidelines
• Use penetration testing tools (ZAP, Burp, VERA…)
  – Mainly focus on injections vulnerabilities, e.g., XSS, SQLi, …
  – Attack patterns highly dependent on application
  – Logic vulnerabilities out of the scope
• Rely on security knowledge of developer/pen-tester
SAML-based SSO for Google Apps

Sequence diagram:

1. Alice (Client) sends Alice, Google, URI to Google (Service Provider).
2. Google (Service Provider) sends FBK, AuthReq(ID, Google), URI to FBK (Identity Provider).
3. FBK (Identity Provider) sends FBK, AuthReq(ID, Google), URI to Google (Service Provider).
4. Google (Service Provider) sends Alice, Google, URI to Alice (Client).
5. Alice (Client) sends Login and consent to Google (Service Provider).
6. Google (Service Provider) sends Google, URI, {AA}\textsuperscript{-1}\textsubscript{FBK} to FBK (Identity Provider).
7. FBK (Identity Provider) sends Builds and authenticated assertion AA=AuthAssert(Alice, FBK) to Google (Service Provider).
8. Google (Service Provider) sends Resource to Alice (Client).
Attack: SAML-based SSO for Google Apps

[Diagram showing the interaction between Alice, Malice, FBK, and Google Calendar.]
State of the art

- BRM analyzer [8], WebSpi[2], AuthScan[3], SPaCloS(SATMC SAT-based model checker)[6], VERA (SPaCloS module)[15], WEMM (Giancarlo Pellegrino, Davide Balzarotti) [5]
  - **Good**: Evaluates protocol against 3 attacker scenarios and classifies parameters in the communication. Helpful for expert pen-tester
  - **Bad**: Identifying attack depends on pen-tester’s skill
  - **Good**: Library of ProVerif for modelling Web specific protocols, use power of model checking to discover vulnerabilities
  - **Bad**: Requires programs to be written in a subset of PHP and Javascript for automatic model extraction
  - **Good**: Possibility to automatically extract protocol model and test the attack trace discovered by model checker
  - **Bad**: Difficult to verify the correctness of the model, False positives
  - **Good**: Nice starting point: combine testing/model checking
  - **Bad**: Inability to extract the model from the specification
  - **Good**: Separates attack from attack payloads
  - **Bad**: Need to manually model the attack sequence
  - **Good**: Automatically generating test cases for a wide range of modern applications
  - **Bad**: No provision for adding new attack patterns
Proposed Approach

• Automatically extracting the protocol model from the implementation
  – Extending state of the art techniques
• Applying attack patterns on the extracted protocol model
• Attack patterns that are applicable for wide range of security protocols
• Possibility to add
  – New attack patterns
  – New attack scenarios
• Automatic testing of the implementation
## Model Inference: Syntactic Labeling

<table>
<thead>
<tr>
<th>Syntactic Label</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST [8]</td>
<td>$scop=(a,b,c)$</td>
</tr>
<tr>
<td>BLOB [8]</td>
<td>$access_token=e72e16c7e42f292c6912e77$</td>
</tr>
<tr>
<td>WORD [8]</td>
<td>$type=code$</td>
</tr>
<tr>
<td>UNKNOWN [5]</td>
<td>$#a$</td>
</tr>
<tr>
<td>EMAIL [5]</td>
<td>$user_email= <a href="mailto:example@example.com">example@example.com</a>$</td>
</tr>
<tr>
<td>EMPTY</td>
<td>$acope=$</td>
</tr>
<tr>
<td>NUMBER</td>
<td>$id=25$</td>
</tr>
<tr>
<td>BOOL</td>
<td>$member=True$</td>
</tr>
</tbody>
</table>
## Model Inference: Semantic Label

<table>
<thead>
<tr>
<th>Label</th>
<th>User 1, Application 1</th>
<th>User 1, Application 2</th>
<th>User 2, Application 1</th>
<th>User 2, Application 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU (user-unique) [8]</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>SU (session-unique) [8]</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>App Unique (AU)</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC (secret) [8]</td>
<td>Parameter is necessary for the authentication</td>
</tr>
<tr>
<td>SIG (signature) [8]</td>
<td>Signature</td>
</tr>
<tr>
<td>BG (browser-generated) [8]</td>
<td>Element present in a request but not included in preceding responses</td>
</tr>
<tr>
<td>REDURI(redirection url)</td>
<td>URL which was passed as a request parameter and later found in the Location header of a redirection response</td>
</tr>
</tbody>
</table>
User: Test, Application: GoogleApp

Client

Test

Service Provider

Google

Identity Provider

FBK

REDURI
Test, Google, appURI

FBK, AuthReq(ID, Google), appURI

FBK, AuthReq(ID, Google), appURI

Login and consent

Google, appURI, \{AA\}K^{-1}_{FBK}

SIG

Sec, Sec, SIG, Sec

Google, appURI, \{AA\}K^{-1}_{FBK}

Resource

Builds and authenticated assertion AA=AuthAssert(Test, FBK)
User: Alice, Application: TestApp

Client

Alice

Service Provider

Bob

Identity Provider

FBK

REDURI

Alice, TestApp, URI

FBK, AuthReq(ID, TestApp), URI

FBK, AuthReq(ID, TestApp), URI

Login and consent

Bob, URI, \{AA\}K^{-1}_{FBK}

Sec, Sec, SIG, Sec
Bob, URI, \{AA\}K^{-1}_{FBK}

Resource

Builds and authenticated assertion
AA=AuthAssert(Alice, FBK)

Sec, Sec, SIG, Sec
Bob, URI, \{AA\}K^{-1}_{FBK}
Replace the value with that of Alice in TestApp
Attack Pattern: Replay attack

- **Goal:** Replay session parameters in order to gain unauthorized access to at least one User Unique element in U1C1
- **Preconditions:** There is at least one element with semantic type as SEC in U2C1
- **Actions:**
  AND 1. Initialize test with baseurl of U2C1 & useractions of U2C1
  2. Set variable sec_list as all elements in U2C1 that has semantic type as SEC
  3. Start executing test
     AND 3.1. For each combination of elements in sec_list, replace their value in the Requests of test with corresponding value in U1C2
- **Post conditions:** There are elements of U1C1 with semantic type as User Unique & origin as responsebody in trace of test
Conclusions

• Existing testing methods are insufficient for automatically testing security protocols
• We discovered a number of security issues in the implementation of widely used SSO protocols (LinkedIn, Yahoo)
• We propose a system that can automatically generate test cases for evaluating the security of protocol implementations
  – Current status: Identifying design patterns for representing protocol, attacks and threat model
Future Work

• Refine the proposed approach and provide a prototype of the tool
• Testing security protocol implementations
• Integrate with a legacy penetration testing tool
• Application of model checking for improving the effectiveness of the vulnerability detection technique
References 1/3


References 2/3


References 3/3

• [16] https://github.com/mozilla/zest/wiki
Thank You

sudhodanan@fbk.eu
Architecture Diagram

Browser
  Selenium Plugin

User Actions
  User 1 Actions
  User 2 Actions

Selenium sessions

Input collector
  Browser

Web Application
  SSO Login
  Sign in with LinkedIn

www.globalsources.com

Model Inference
  Protocol Model
  User 1 Session 1
  User 1 Session 2
  User 2 Session 1

Extended Zest Scripts

ZAP
  Proxy
  API

Input Traces
  User i, Session j

ZAP Session
Architecture Diagram cont.

Web Application
  SSO Login
  Sign in with LinkedIn

www.globalsources.com

Protocol Model
  User 1 Session 1
  User 1 Session 2
  User 2 Session 1

Extended Zest Scripts

Test Case Generation
  Oracle

Attack Patterns
  OAuth attacks
  Online shop Attacks

Get pattern

Is Applicable?
  Yes
  No

Is Testable?
  Yes
  No

High Priority
Low Priority