Introduction

- Social sign-on: users use social networks to sign on to a third-party site
  - Microsoft, Facebook (OAuth), Google (OpenID)
- Example: Yelp
  - Yelp allows you to log on via Facebook
  - Clicking login will open a window with Facebook login
  - Facebook will prompt for permission for Yelp to access certain data
  - Returns to Yelp with successful authentication
- Security goals
  - User doesn't want unauthorized access to personal info
  - Yelp wants to ensure that user is correct
  - Facebook wants to ensure Yelp and user are who they say they are
  - Need to be secure over insecure web!
- Social Login CSRF
  - Assumptions
    - User must have previously logged into third party client via OAuth, e.g. Facebook
    - Facebook only asks for authorization the first time
  - Attacker controlled site can log into third party client and access third-party APIs on behalf of user without user's knowledge

OAuth

- OAuth 2.0 involved parties
  - Resource server
  - trusted authorization server
  - Resource owner
  - user agent
  - client
- Comparison with first OAuth
  - flows are protocol configurations
  - lightweight and flexible
- User Agent flow
  - Used to perform auth from a user-agent running a client application
  - Figure below details flow
- Web Server Flow
  - Used between client server and resource server/auth server
  - Figure below details flow

- Protocol parameters
- **redirection URI**: tells where the authorization server should send its response (the access token) to. This in effect determines where the user agent navigates to after authorization.
- **state**: a nonce that is strongly bound to a resource owner's session. It is attached to the request when client app tells user agent to redirect to auth server.
  - prevents Social CSRF
- **Threat model**
  - network attackers
  - malicious sites
  - malicious redirectors
  - existing CSRF vulns in clients
- **Security goals are modeled in Datalog-like policies**

**WebSpi**

- **A library of idioms useful in modeling web app security**
- **Model constituents**
  - **Principals**: users and servers
    - own credentials / pub-priv keys
    - WebSurfer
  - **Processes**: Receives and responds over channels; does actions
    - CredentialFactory
    - HttpClient (e.g. browsers)
    - HttpServer
  - **Channels**: communication pathways
    - net: HttpClient communicates with HttpServer
    - httpClientRequest: user submits http request & uri to HttpClient over this private channel
    - httpClientResponse: HttpClient returns response through this private channel
- **Security policies**
  - **Assume(...)**: Adds given statement to global knowledge
    - Assume(UserSends(user, message))
  - **Expect(...)**: Must be able to prove statement at this point in the code
    - Expect(ServerAuthorizes(s, u, d)): prove that server s is willing to authorize user u to get data d
  - "security policies are defined as Horn clauses extending a predicate fact"
    - "A Horn clause with exactly one positive literal is a **definite clause**; a definite clause with no negative literals is sometimes called a **fact**; and a Horn clause without a positive literal is sometimes called a **goal clause** (note that the empty clause consisting of no literals is a goal clause)."
Says(...): if a fact e is true, then it is assumed to be said by any principal. If a principal is compromised, then the principal is untrusted and can say anything.

- **Modeling**
  - Three processes: process running on HttpServer, client process on HttpClient, user process using a user-agent
  - LoginUserAgent() process
    - `in(httpClientResponse, (...))`
    - gets a login page info in (...) from channel httpClientResponse, which is the response received by httpClient when requesting login page
  - LoginApp() process

- **Attacker model**
  - Dolev-Yao
  - All channels and tables in WebSpi are private by default
  - AttackerProxy
    - receives commands on public chan "admin", and sends results out on public chan "result".
    - Manage principals
    - Network attacks
    - Start malicious client sites / client webapps
    - Inject malicious javascript to HttpClient

### OAuth 2.0: ProVerif Analysis

- **OAuth 2.0 Model**
  - Login application
    - process LoginApp + user-agent LoginUserAgent
    - Models form-based login
  - Data Server
    - Models resource servers
  - user-agent flow
    - OAuthImplicitServerApp: auth servers
    - OAuthUserAgent: resource owners
  - web server flow
    - OAuthExplicitClientApp: clients
    - OAuthExplicitServerApp: auth servers

- **Positive ProVerif verification**
  - attacker model: network, malicious resource owners, malicious clients, malicious sites, malicious javascript, OAuth implicit + explicit flows
    - no http redirectors, no CSRF on honest apps

- **Attacks**
  - Social CSRF
- Token redirection