OAuth: Where are we going?

- What is OAuth?
- OAuth and CSRF
- Redirection
- Token Reuse
- OAuth Grant Types

OAuth v1 and v2

"OAuth 2.0 at the hand of a developer with deep understanding of web security will likely result [in] a secure implementation. However, at the hands of most developers ... 2.0 is likely to produce insecure implementations."

Eran Hammer, Original Founder of OAuth

http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/
http://hueniverse.com/2010/09/29/oauth-bearer-tokens-are-a-terrible-idea/

OAuth History

- Nov 2006: Blaine Cook begins working on OAuth while at Twitter
- 2007: Ma.gnolia, Google and others join the discussion
- 2007: Eran Hammer joins and soon leads the specification
- Dec 2007 : OAuth 1.0 final draft
- 2008 : Google OAuth 1.0 support begins
- 2010 : Twitter forces all third party apps to use OAuth 1.0
- June 2012: Eran ragequits the OAuth 2.0 body after the shift from crypto to bearer tokens
- October 2012: OAuth 2.0 framework published

OAuth v1 and v2:
What are the primary security differences?

OAuth v2 is *Transport-Dependent*
Most security defenses are delegated to HTTPS/TLS

A typo, an improper TLS configuration, or a failure to properly validate a certificate can lead to a man-in-the-middle attack, compromising all OAuth communications.

OAuth v1 is *Transport-Independent*
Security is not delegated to HTTPS/TLS

OAuth v1 messages are each individually cryptographically signed. If a single message within the communication is constructed or signed improperly, the entire transaction will be invalidated.

OAuth v1 and v2: Signatures vs Bearer Tokens

OAuth v2 Authorizes Messages with Bearer Tokens

Bearer Tokens do not provide internal security mechanisms. They can be copied or stolen.

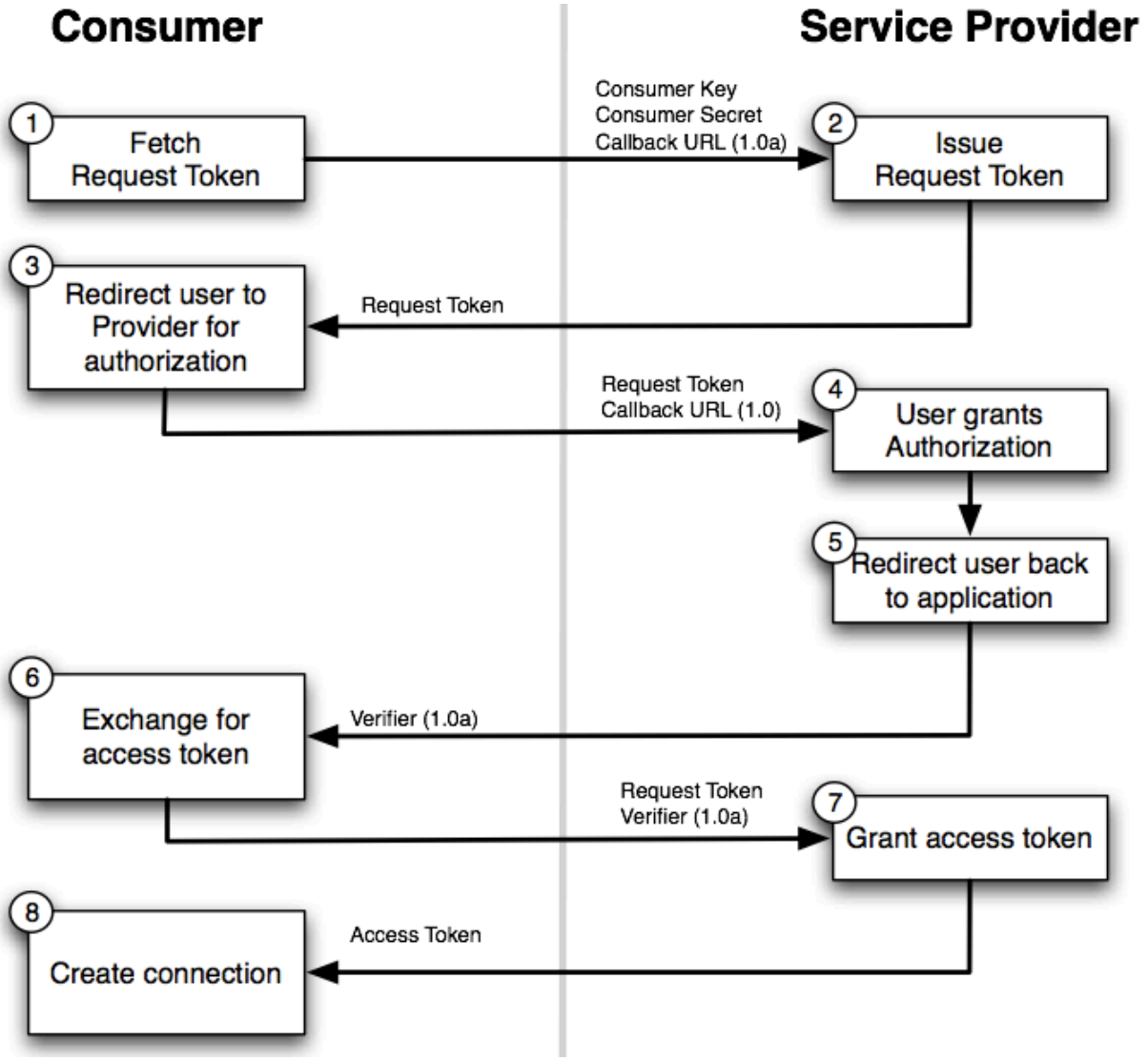
OAuth v1 Authorizes Messages with Digital Signatures

A signed message is tied to it's origin. It cannot be tampered with or copied to another source.

OAuth v1 and v2: Which should you use?

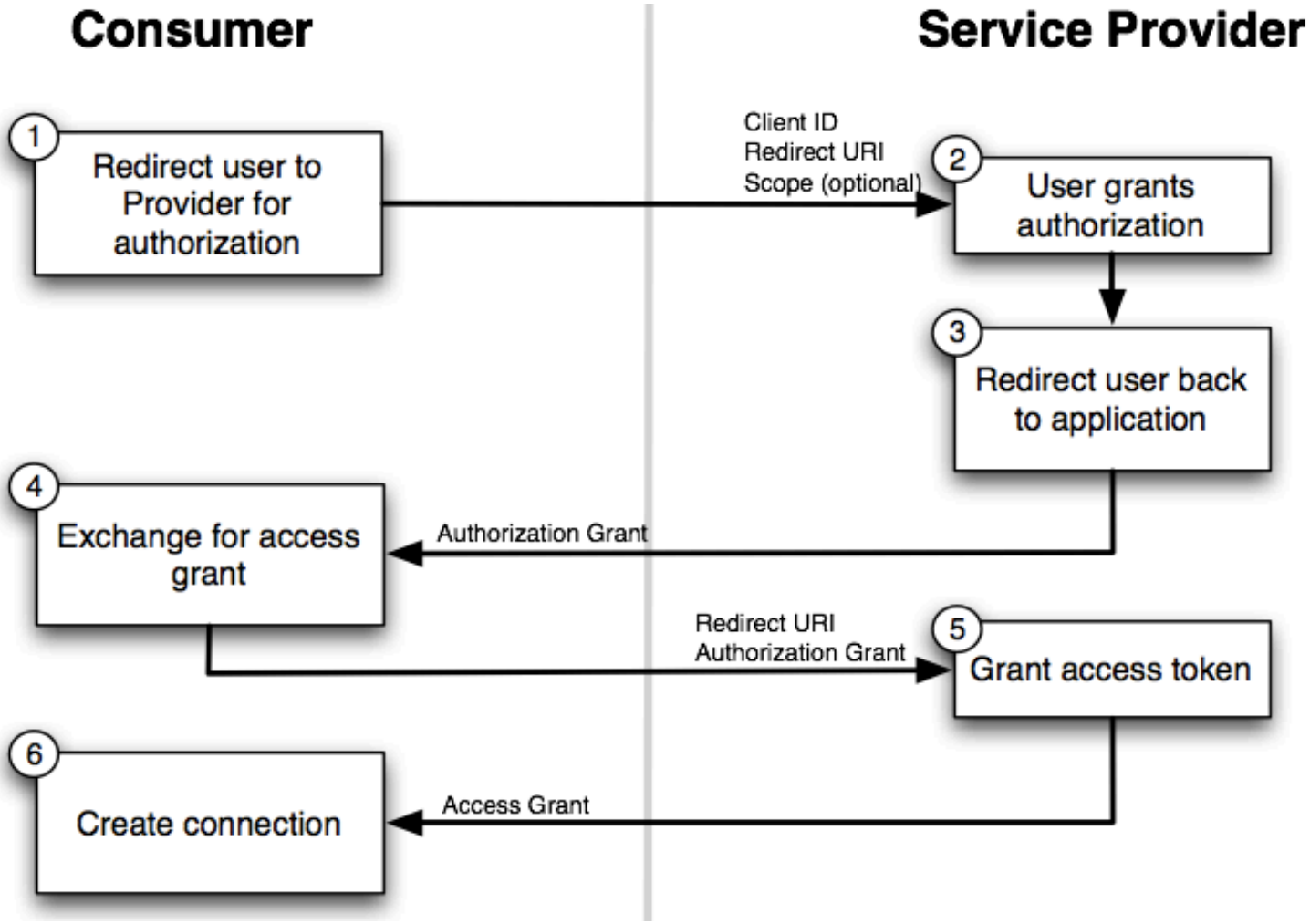
- **Google moved away from OAuth 1.0 in April 2012.**
- **Twitter still supports OAuth 1.0.**
- **It's rare for new server implementations to support OAuth 1.0.**
- **Plenty of OAuth 2.0 "add-on" RFC's to support crypto if needed.**
- **So yea, 2.0 in almost all situations in 2015.**

OAuth v1 Workflow



http://docs.spring.io/spring-social/docs/1.0.0.RC1/reference/html/serviceprovider.html#service-providers-oauth1

OAuth v2 Workflow



http://docs.spring.io/spring-social/docs/1.0.0.RC1/reference/html/serviceprovider.html#service-providers-oauth2

OAuth Security in Products is Stabilizing (osvdb.org)

Search Query: text_type: alltext vuln_title: oauth		
ID	**Disc Date**	**Title**
119704	2015-03-18	IBM WebSphere Application Server Full / Liberty Profile OAuth Grant Password Functionality Unspecified Remote Privilege Escalation
119124	2015-01-09	GNOME librest OAuth Implementation Invalid Pointer Dereference Arbitrary Code Execution
113959	2014-10-29	OAuth1 Server Plugin for WordPress lib/class-wp-json-authentication-oauth1.php Token / Signature Verification Timing Attack Weakness
113396	2014-10-08	OAuth2 Client Module for Drupal Client Class API Error Message XSS
121176	2014-09-03	Oauth 2.0 Protocol Scope Handling Open Redirect Weakness
109037	2014-07-11	Katello API OAuth Authentication consumer_key Value Handling Remote Memory Exhaustion DoS
108326	2014-06-12	WP Microblogs Plugin for WordPress get.php oauth_verifier Parameter Reflected XSS
108328	2014-06-12	WP-RESTful Plugin for WordPress html_api_authorize.php oauth_callback Parameter Reflected XSS
106567	2014-05-02	OAuth / OpenID Protocol Unspecified Application Redirect Weakness
107306	2014-04-13	Phabricator Anchor Reattachment Oauth Token Disclosure
107308	2014-04-11	Phabricator Login 302 Redirect Anchor Preservation Oauth Token Disclosure
106246	2014-04-02	IBM SmartCloud Analytics Log Analysis OAuth Authorization Endpoint Invalid Query Parameter Response XSS
107316	2014-02-24	Phabricator Oauth Flow Arbitrary Attacker-controlled Twitter Account Authentication CSRF
102193	2014-01-17	Open-Xchange (OX) AppSuite oAuth API Call Unspecified Parameters Reflected XSS
101897	2014-01-07	OAuth Library for PHP Example Page Unspecified XSS

Sample OAuth Workflow

- Using OAuth, your eCommerce server can now tweet on behalf of the user even when the user is not logged on. How does this happen?
- First, the user logs into the eCommerce server with his account and edits his account profile.
- Next, the eCommerce server redirects the user to Twitter.
- The user logs onto Twitter and authorizes the eCommerce server to tweet on her behalf.
- Then, whenever orders are complete the eCommerce tweets a little note about how awesome the eCommerce company is - even when the user is not logged onto the eCommerce server.

High Level Concepts

- **OAuth Roles (**Resource owner, resource server, client app, authorization server)
- **OAuth Grant Types** (authorization code, implicit, resource owner password credentials, client credentials, extensions)
- **Token Types** (refresh token, access token)
- **Endpoint Types** (resource server, authorization server, client registration, client authorization)

Terms

- **Resource Owner or End-User:** User and account owner of resource (the end-user)
- **Resource Server/Service Provider:** Server hosting protected resources owned by the end-user. Accepts access tokens for protected resources.
- **Authorization Server/Service Provider.** Server issuing access tokens to provide other clients access to protected resources. Often same server as resource server. One authorization server may issue access tokens to many resource servers.

Terms

- **Client/Consumer:** Application requesting access to protected resource on behalf of resource owner (typical clients are mobile applications, web browsers, desktop applications or other web applications). Depending on the OAuth workflow, the browser may use an access token directly (implicit grant type) or redirect the user to another web application that acts as the client of the service (authorization code grant type).

Terms

- **Access Token:** OAuth token used to directly access protected resources on behalf of a user or service.
- **Refresh Token:** Refresh tokens, when given to the authorization server, will provide a new active access token. Refresh tokens themselves cannot access resources. While access tokens should be short lived, refresh tokens are long lived or simply never expire until the user revokes them. Refresh tokens also provide more scalable patterns.
- **Client Identifier:** Unique ID of each client given to client by authorization server.
- **Bearer Token:** "A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession)." - https://tools.ietf.org/html/rfc6750#section-1.2

Authorization Server Security

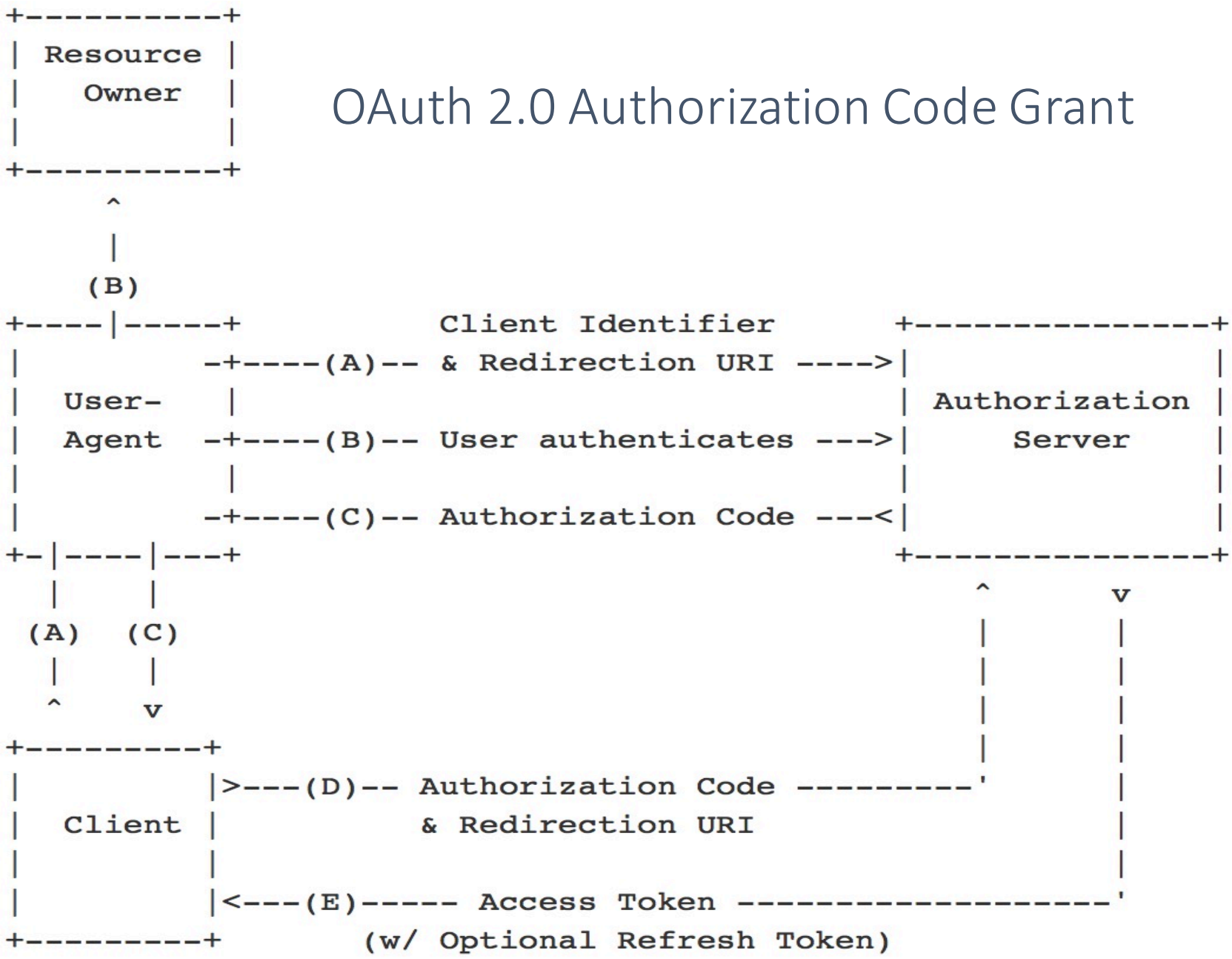
https://tools.ietf.org/html/rfc6819

- TLS for everything (Authenticity, Confidentiality, Integrity)
- Authorization servers should not automatically process repeat authorizations to public clients unless the client is validated using a pre-registered redirect URI (Section 5.2.3.5).
- Authorization servers can mitigate the risks associated with automatic processing by limiting the scope of access tokens obtained through automated approvals (Section 5.1.5.1).
- Explain the scope (resources and the permissions) the user is about to grant in an understandable way (Section 5.2.4.2).
- Narrow the scope as much as possible (Section 5.1.5.1).
- Don't redirect to a redirect URI if the client identifier or redirect URI can't be verified (Section 5.2.3.5).

OAuth 2.0 Grant Types

- You can hide long lived tokens token from the user (**authorization code grant**)
- You can only activate a short-lived token in the browser when the user is currently logged on (implicit grant)
- You can grant and expose a long-lived tokens directly to the user via a trusted client (password grant).
- You can grant and expose a long-lived token directly to other services that need to access data not associated with a specific user (client credentials grant)

OAuth 2.0 Authorization Code Grant



Reference: https://tools.ietf.org/html/rfc6749

OAuth 2.0 Authorization Code Grant

- The User (Resource Owner) Credentials are never exposed to Client
- The User (Resource Owner) never has access to actual access token
- The Client application can use the access token even when the resource owner is not present
- Authorization Code Refresh Tokens are often long lived or permanent until the User (Resource Owner) revokes this access through the Client UI.

Authorization Code Variables

- The client starts the "authorization code" workflow by redirecting the user to the authorization server with the right request data. This initial client request includes:
- response_type : this is required by "authorization code" grant type and should contain the value "code"
- client_id: this is the client identifier assigned to the client at client registration time. This is unique for every client for authorization code grants.
- scope: level of access requested, domain specific
- redirection URI: Where the authorization server redirects the user after access is granted or denied

Authorization Code Grant Security

https://tools.ietf.org/html/rfc6819#section-4.4.1

- 4.4.1.1 Threat: Eavesdropping or Leaking Authorization "codes" (Referrer header leakage, logs, open redirect, browser history)
  - **Use TLS, require strong authentication between client and server, expiration time for access tokens, only allow one use per token, consider revoking client sending lots of bad codes, reduce scope of tokens, flush browser cache**
- 4.4.1.2 Threat: Obtaining Authorization "codes" from Authorization Server Database
  - **Parameterize your F#$KING QUERIES, store access tokens with a one-way methodology, good database security**
- 4.4.1.3 Threat: Online Guessing of Authorization "codes"
  - **High entropy tokens, strong client authentication, short expiration time**
- 4.4.1.5 Threat: Authorization "code" Phishing
  - **Standard Phishing defense. Good luck! One phish and game over.**

CSRF Attacks against OAuth Part 1

1. **Attacker** assumes that **Victim** is currently logged in at **Consumer** site https://consumer-site.example/ (The OAuth ***Client*** Application)
2. **Attacker** goes through registration and login workflow at **Consumer Site** https://consumer-site.example/login and uses that account to trigger a Oauth workflow with the Provider Service (The OAuth authorization/resource server)
    1. **Consumer Site** redirects attacker to **Provider Site** login interface. This is called the *Authorization Request.*

       https://provider-site.example/login
    2. **Attacker** successfully logs in with **Provider Site**
    3. **Provider Site** responds with redirect URL which contains the authorization code in the *code* parameter. this is called the *Authorization Grant*. http://consumer-site.example/auth?code=1a2s3d4f5g6h

CSRF Attacks against OAuth Part 2

3. Instead of visiting or redirecting to the *Authorization Grant* redirect URL, **Attacker** copies the URL and places a reference to it in an image tag on a web page () (https://evil-page.example/)
4. **Attacker** gets **Victim** to visit https://evil-page.example/.
    1. This in turn gets **Victim** to request the Authorization Grant URL (http://consumer-site.example/auth?code=1a2s34f5g6h)
    2. By visiting the Authorization Grant URL, the **Victim** has now authorized the **Attacker** to have full authorized access to **Victim's** account on **Consumer Site** (https://consumer-site.example/).

OAuth 2 and avoiding CSRF

1	**Consumer** generates unique random state value, and stores it in server side session variable. JSON Web Tokens are good for state values.
2	**Consumer** sends "state" parameter with Authorization Request
3	On successful authorization, **Provider Site** includes "state" parameter in Authorization Grant redirect URI
4	When **Victim** visits redirect URI, the "state" parameter is compared against the "state" parameter stored in server side session variable.

Use the "state" parameter!
It is essentially a CSRF token

OAuth 2 Authorization Code Flow Open Redirector: Attack

1. **Victim** goes through login workflow at **Consumer Site** (https://consumer-site.example/login) using **Provider Site** for authorization.
2. **Attacker** constructs an Authorization Request URL for **Provider Site**
    1. **redirect_uri** is set to https://evil-site.example/
3. **Attacker** either embeds evil URL in an image tag or constructs a clickable link at **Consumer Site**.
4. When the evil URL is loaded, the provider will 302 redirect back to **redirect_uri** since **user** was already logged in.
5. When the redirect occurs, the evil site can read the HTTP Referrer to get the Authorization Code.
6. Using this Authorization Code, Attacker can login as **user**

OAuth 2 Authorization Code Flow Open Redirector: Remediation

1. **Whitelist redirect_uri!**
2. **There is no need for a Provider to require the redirect_uri param!**

OAuth 2 Implicit Flow
Access Token Reuse: Attack

1. **Victim** authorizes with **Evil Consumer Site** for **Provider Site** using **access_token**
2. **Acme Widgets Consumer Site** uses Implicit Flow for authentication.
3. **Attacker** authenticates as **Victim** with the **Evil Consumer Site access_token** using https://acme-widgets.example/callback#access_token=**access_token**

"One Token to Rule Them All"

OAuth 2 Implicit Flow
Access Token Reuse: Remediation

1. **OAuth should be used for authorization, not authentication!**
2. **Validate that access_token belongs to your client_id via provider API**

OAuth 2.0: Summary

1. **Holy crap this is crazy**
2. **It takes massive efforts to build secure OAuth 2 solutions**
3. **The core standard barely addresses security**
4. **Major providers with PHD's to spare are overall doing a reasonable job of build secure solutions**
5. **Clients are at risk because they are likely to build less security implementations than providers**
6. **Buckle up, read the threat model several times and follow it's many many many recommendations**

OAuth: Summary

- What is OAuth?
- OAuth and CSRF
- Redirection
- Token Reuse
- OAuth Grant Types

Thank You!

jim@manicode.com