RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

SESSION ID: IDY-W04

OAUTH 2.0 THREAT LANDSCAPES

Prabath Siriwardena

Senior Director of Security Architecture
WSO2
@prabath

NEWS

Google Docs phishing attack underscores OAuth security risks

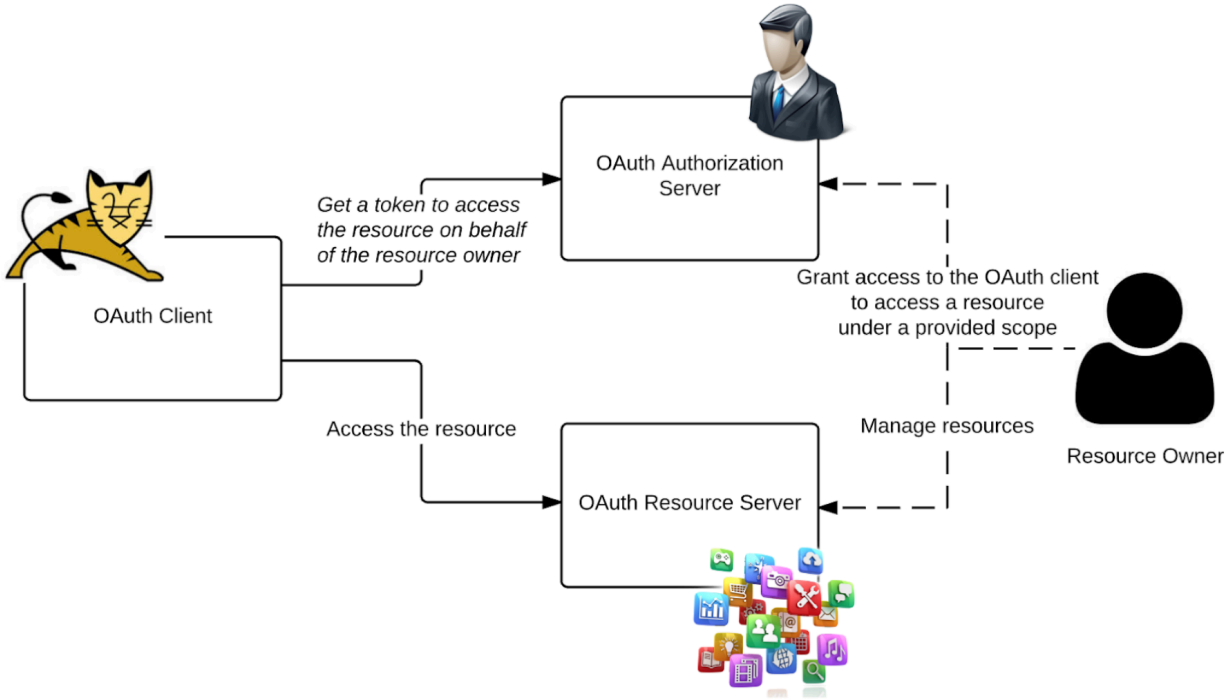
One security researcher easily managed to replicate Wednesday's phishing attack.

By Michael Kan

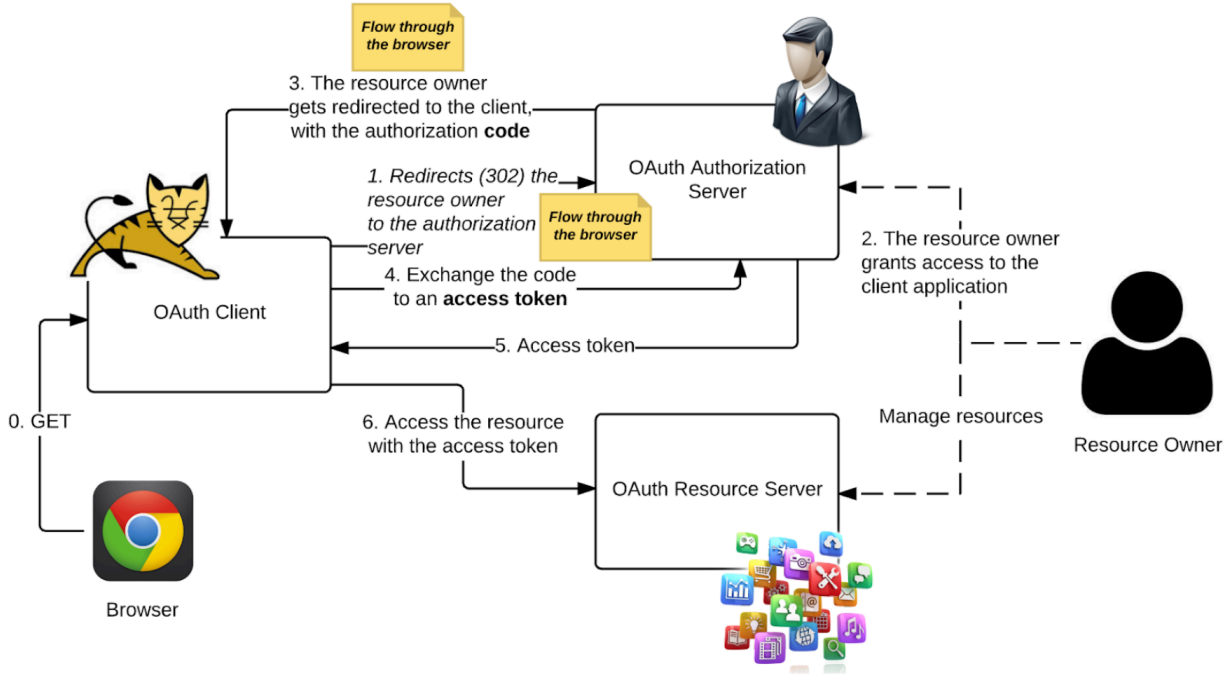
U.S. Correspondent, IDG News Service | MAY 5, 2017 4:30 AM PT

OAUTH 2.0 - A QUICK OVERVIEW

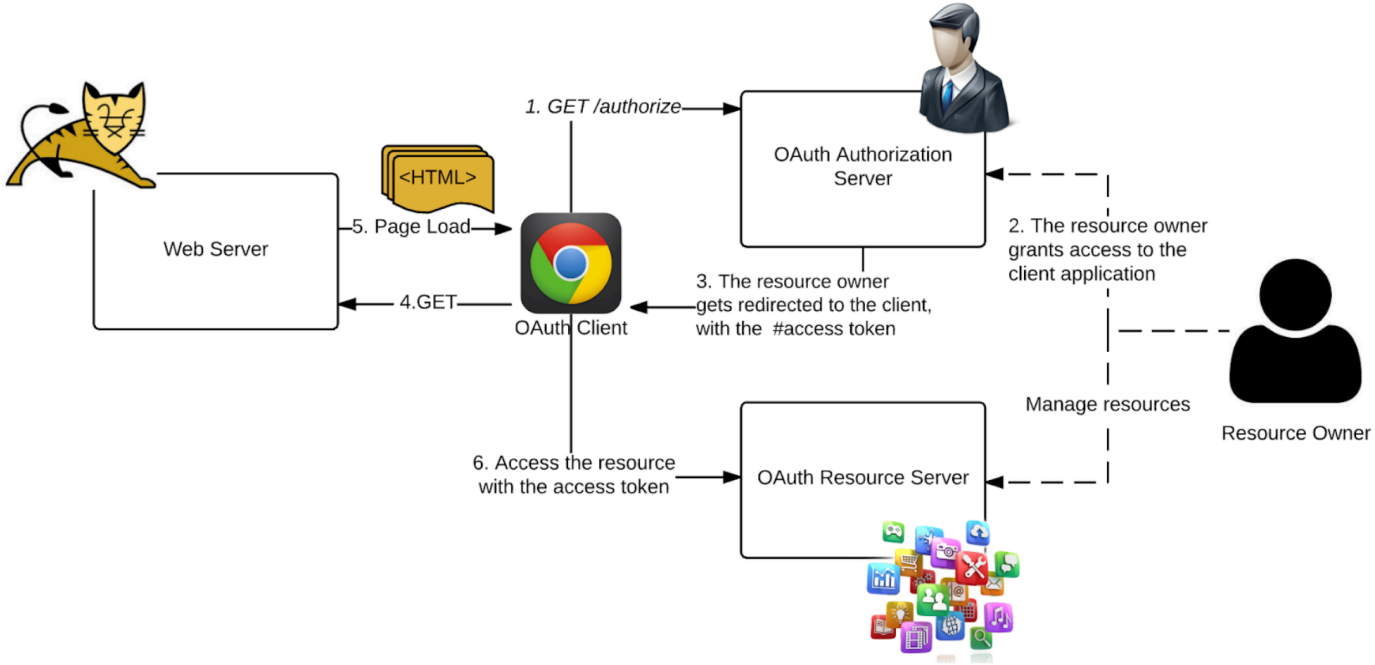
OAuth 2.0


OAuth Authorization Server
Get a token to access the resource on behalf of the resource owner
OAuth Client
Grant access to the OAuth client to access a resource under a provided scope
Resource Owner
Access the resource
Manage resources
OAuth Resource Server

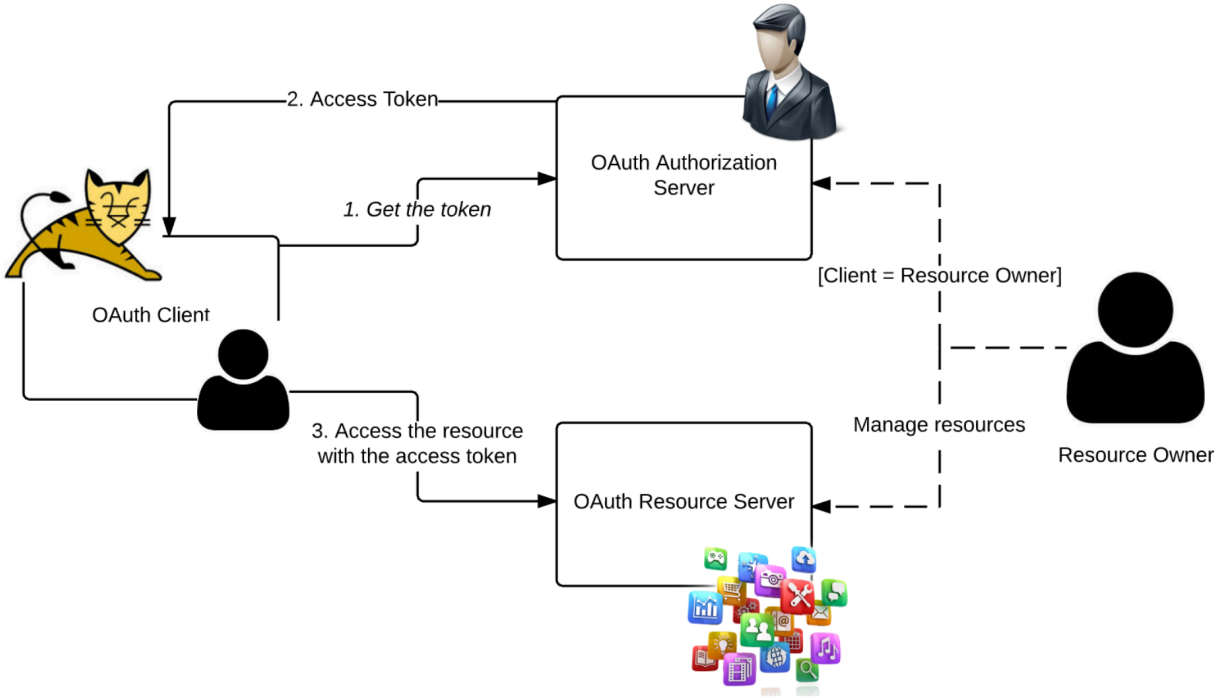
Authorization Code Grant Type



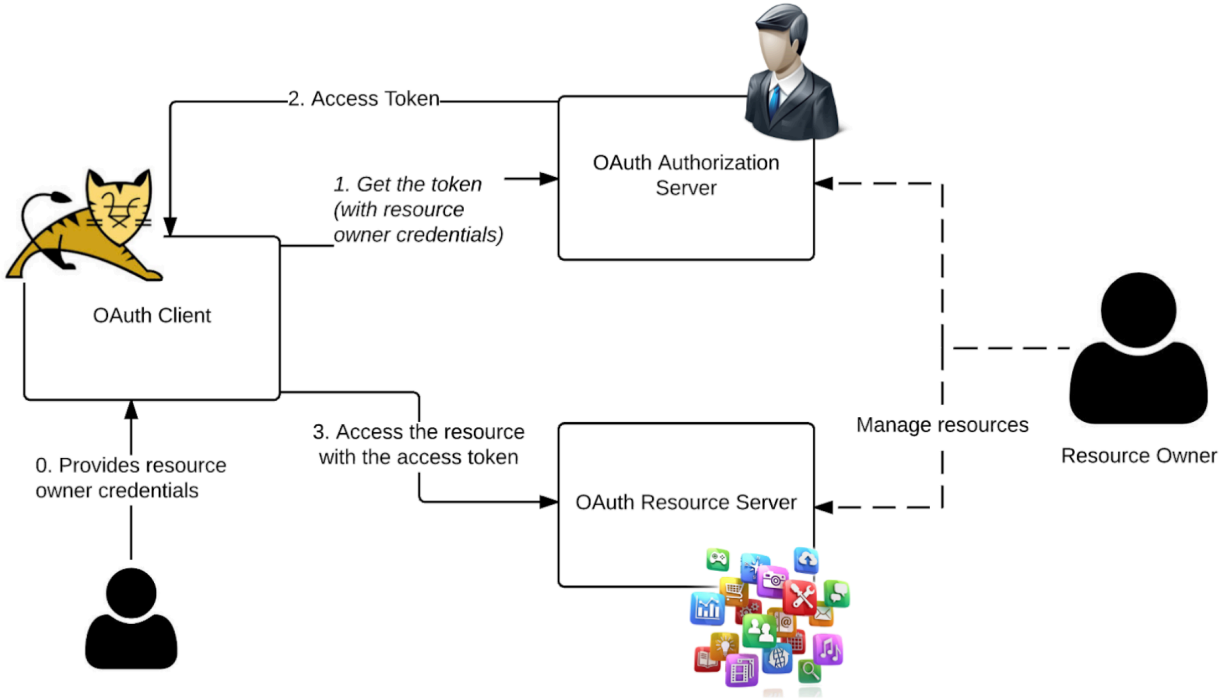
Implicit Grant Type



Client Credentials Grant Type

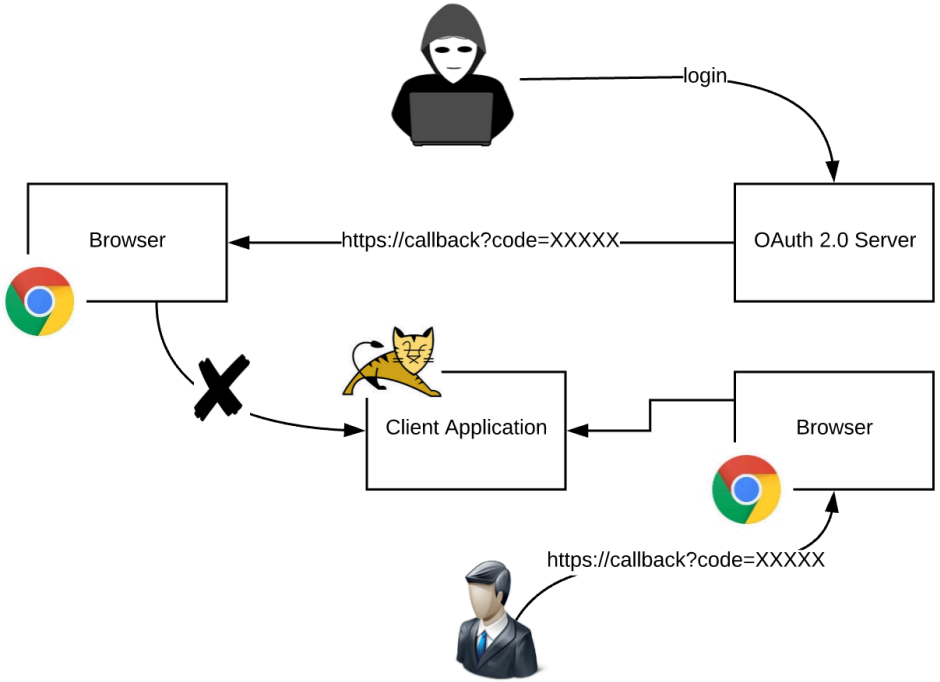


Password Grant Type



THREATS / MITIGATIONS / BEST PRACTICES

Session Injection with CSRF (Threats)



Session Injection with CSRF (Victims)

- Web / Mobile application users

Session Injection with CSRF (Mitigation / Best Practices)

- Short-lived authorization code
- Use ***state*** parameter
- Proof-key-for-code-exchange (PKCE)

Token Leakage (Threats)

- Attacker may attempt to eavesdrop authorization code/access token/ refresh token in transit.
- Authorization Code Flow Open Redirector
- OAuth 2.0 native apps can get hold of the authorization code.

Token Leakage (Threats)

- Attacker may attempt a brute force attack to crack the authorization code/access token.
- Attacker may attempt to steal the authorization code/access token/ refresh token stored in the authorization server.
- IdP Mix-Up / Malicious Endpoint.

Token Leakage (Victims)

- Web/Mobile application users
- Web/Mobile application owners

Token Leakage (Mitigation / Best Practices)

- Always on TLS (use TLS 1.2 or later)
- Address all the TLS level vulnerabilities both at the client, authorization server and the resource server.
- The token value should be >=128 bits long and constructed from a cryptographically strong random or pseudo-random number sequence.

Token Leakage (Mitigation / Best Practices)

- Never store tokens in clear text - but the salted hash.
- Short-lived tokens.
  - LinkedIn has an expiration of 30 seconds for its authorization codes.
- The token expiration time would depend on the following parameters.
  - Risk associated with token leakage
  - Duration of the underlying access grant
  - Time required for an attacker to guess or produce a valid token

Token Leakage (Mitigation / Best Practices)

- One-time authorization code
- One-time access token (implicit grant type)
- Use PKCE (proof key for code exchange) to avoid authorization code interception attack.
  - Have S256 as the code challenge method
- Enforce standard SQL injection countermeasures

Token Leakage (Mitigation / Best Practices)

- Avoid using the same client_id/client_secret for each instance of a mobile app - rather use the Dynamic Client Registration API to generate a key pair per instance.
  - Most of the time the leakage of authorization code becomes a threat when the attacker is in hold of the client id and client secret.
- Restrict grant types by client.
  - Most of the authorization servers do support all core grant types. If unrestricted, leakage of client id/client secret gives the attacker the opportunity obtain an access token via client credentials grant type.

Token Leakage (Mitigation / Best Practices)

- Enable client authentication via a much secured manner.
  - JWT client assertions
  - TLS mutual authentication
  - Have a key of size 2048 bits or larger if RSA algorithms are used for the client authentication
  - Have a key of size 160 bits or larger if elliptic curve algorithms are used for the client authentication

Token Leakage (Mitigation / Best Practices)

- White-list callback URLs (redirect_uri)
  - The absolute URL or a regEx pattern
- IdP-Mixup
  - Use different callback URLs by IdP
  - https://tools.ietf.org/html/draft-ietf-oauth-mix-up-mitigation-01

Token Reuse/Misuse (Threats)

- A malicious resource could reuse an access token used to access itself by a legitimate client to access another resource, impersonating the original client

Token Reuse/Misuse (Threats)

- An evil web site gets an access token from a legitimate user, can reuse it at another web site (which trusts the same authorization server) with the implicit grant type
  - https://target-app/callback?access_token=<access_token>
- A legitimate user misuses an access token (issued under implicit grant type/SPA) to access a set of backend APIs in a way, exhausting server resources.

Token Reuse/Misuse (Victims)

- Web/Mobile application users
- Web/Mobile application owners

Token Reuse/Misuse (Mitigations / Best Practices)

- A malicious resource (an API / Microservice) could reuse an access token used to access itself by a legitimate client to access another resource, impersonating the original client.
- An evil web site gets an access token from a legitimate user, can reuse it at another web site (which trusts the same authorization server) with the implicit grant type
  - https://target-app/callback?access_token=<access_token>

Token Reuse/Misuse (Mitigations / Best Practices)

- A legitimate user misuses an access token (issued under implicit grant type/SPA) to access a set of backend APIs in a way, exhausting server resources.
- To avoid exhausting resources at the server side, enforce throttle limits by user by application. In case an attacker wants to misuse a token - the worst he/she can do is to eats his/her own quota.

Token Export (Threats)

- An attacker could export an access token from its originating channel and use somewhere else.
- A common attack vector for SPAs (Single Page Applications)
- A major concerns with bearer tokens.

Token Export (Victims)

- Web/Mobile application users
- Web/Mobile application owners

Token Export (Mitigations / Best Practices)

- The use of Token Binding protects access tokens from man-in-the-middle and token export and replay attacks.
- https://tools.ietf.org/html/draft-jones-oauth-token-binding-00

Apply What You Have Learned Today!

- Review and test current OAuth 2.0 client applications against the threats we discussed – probably build a test suite!
- Build a security check-list for an OAuth 2.0 Authorization Server, and make sure what you build or what you buy, adheres to it.
- Be the security champion of your team!

THANK YOU!