OAuth 2.0 and the Road to XSS: attacking Facebook Platform

Andrey Labunets — @isciurus
Who is @isciurus

• Security researcher, occasional reverse-engineer
• Student at the Tyumen State University
• Frequent guest to Facebook vulnerability submission form
OAuth

• An open framework for web authorization
  • Resource owner authorizes Client to access owner’s data on Resource server
  • Password never given to a client

• Known attacks on OAuth variations
  • Facebook JS SDK bugs by K. Bhargavan, C. Bansal in 2012
  • Flash bug on Facebook by R. Wang, S. Chen, L. Xing, X. Wang in 2012
  • ...

• Fundamental problems
  • Session fixation for OAuth 1.0 in 2009
  • Bearer tokens for OAuth 2.0
  • ...
OAuth 2.0 in 60 seconds

Implicit Grant Flow

- **A.** Client Identifier & Redirection URI
- **B.** User Authenticates
- **C.** Redirection URI with Access Token in Fragment
- **D.** Redirection URI Without Fragment
- **E.** Script
- **G.** Access Token

Authorization Code Flow

- **A.** Client Identifier & Redirection URI
- **B.** User Authenticates
- **C.** Authorization Code
- **D.** Authorization Code & Redirection URI
- **E.** Access Token (w/ Optional Refresh Token)
OAuth 2.0 Case Study: Facebook Platform

Motivation:

• OAuth 2.0 — proposed RFC standard
• Facebook — largest platform for web-developers (1b users, 9m apps)
• Poorly explored, huge attack surface
Assumptions and threat model

• A victim has an account on Facebook, and he uses some apps

• An attacker is able to create a malicious web site and a malicious Facebook app

• An attacker can convince a victim to click a specially crafted malicious link

• Attacker wants to:
  • Access victim’s private data
  • Invoke some actions on behalf of a victim
  • Sign into his account on a third-party web site (authentication bypass)
  • Execute its code on *facebook.com* client-side (XSS)
Legacy authorization flow
Legacy authorization flow

- `extern/login_status.php` returns token in query string

Exploitation:

- Insert a picture from your server somewhere inside the Client site
- Tamper `redirect_uri` to point this page
- Let the user click the link

- Resource owner’s access token leaked via HTTP Referrer
Legacy authorization flow


HTTP 302
Legacy authorization flow

- Lots of external developers depend on this flow, not easy to patch
- Still works for some apps (bing, etc)

Impact:
- Access token stealing

Lessons:
- Design it carefully
- If not, don’t mix legacy/latest auth flows
Javascript SDK issues
Normal JS SDK workflow

- Trusted Client
- JS SDK
- Facebook proxy iframe
  - `xd_arbiter.php?version=18#...origin=app.com...`
- Facebook auth window
  - `xd_arbiter.php?version=18#...origin=client.com...&signed_request=...&token=...`

`postMessage()`

`proxyMessage()`
Flaw in JS SDK proxy

Evil Facebook Client

JS SDK

Facebook proxy iframe

\texttt{xd\_arbiter.php#...origin=evil.com...}

Facebook auth window

\texttt{xd\_arbiter.php#...origin=client.com...&signed\_request=...&token=...}

dst/src origins were never compared
Flaw in JS SDK proxy

• Exploited by setting `redirect_uri` to an old-versioned `xd_proxy` without origin checks

• Impact:
  • Code, access token, signed_request stealing

• Lessons:
  • If this is out of specs, implement in twice carefully

• Suggestion:
  • Make JS SDK `xd_arbiter` open-source
URL fragment tricks
Hash-bang (#!) + Referrer exploitation

• Facebook QuicklingPrelude (or hash-bang feature):
  • Fills `location` with value from `location.hash`
  • Redirect: `facebook.com/#!/whatever` → `facebook.com/whatever`
  • Abused to pull sensitive data from URL fragment

• Generic idea of all hash-bang + Referrer exploits:
  • Redirect to a permitted page at `facebook.com`
  • Pull access token from fragment and redirect to another Facebook page
  • Redirect to your own domain
  • Pick the Referrer from the request and extract the token
App RPC getHash trick

- Facebook app controller implemented a special `getHash` method (possibly, for app navigation or parameter passing)

- `top.location.hash` could be disclosed to a malicious app iframe

- No need to authorize the malicious app

- Exploitation:
  - Utilize hash-bang feature to bypass filters on `redirect_uri`
  - Redirect to your app canvas page
  - Invoke `FB_RPC` call `getHash` from your app
  - Get a full URL fragment with access token
App RPC getHash trick

Facebook.com

1. `postMessage("FB_RPC:...method:getHash...",...)`

App.com Canvas iframe

2. `XdArbiter.handleMessage()`

3. `proxyMessage("FB_RPC:...result:"access_token=AAA...",...)`

4. `postMessage("FB_RPC:...result:"access_token=AAA...",...)`

Facebook proxy iframe
URL fragment tricks

- Fragment-based navigation is an excellent vector for OAuth 2.0

  - Impact:
    - code, access token, signed_request stealing

  - Lessons:
    - Avoid navigation with URL fragment on your authorization endpoint domain
    - If not, deny any redirect_uri containing URL fragment
    - If not, think twice how you integrate your fragment navigation with OAuth 2.0
PHP SDK issues
PHP SDK issues

- **OAuth 2.0**: stealing code via `redirect_uri` tampering gives nothing
- **Facebook JS/PHP SDK**: code is issued with an empty `redirect_uri`:

  ```php
  src/base_facebook.php#L426
  
  protected function getUserAccessToken() {
    ...
    // the JS SDK puts a code in with the redirect_uri of ''
    if (array_key_exists('code', $signed_request)) {
      $code = $signed_request['code'];
      ...
      $access_token = $this->getAccessTokenFromCode($code, '');
      ...
  ```

- **`redirect_uri` tampering-based attacks are invisible**
PHP SDK issues

*signed_request* takes priority over code-based authentication:

src/base_facebook.php#L525

protected function getUserFromAvailableData() {
  // if a signed request is supplied, then it solely determines
  // who the user is.
  $signed_request = $this->getSignedRequest();
  if ($signed_request) {
    if (array_key_exists('user_id', $signed_request)) {
      $user = $signed_request['user_id'];
  
*signed_request* parsed also from $_REQUEST, no CSRF checks:

src/base_facebook.php#L489

public function getSignedRequest() {
  if (!$this->signedRequest) {
    if (!empty($_REQUEST['signed_request'])) {
      $this->signedRequest = $this->parseSignedRequest($_REQUEST['signed_request']);
PHP SDK issues

• PHP SDK compromises OAuth 2.0 authorization code grant flow

• Still not patched

• Impact:
  • Downgrade attack (from code grant to signed_request-based flow)
  • Session fixation (CSRF) with signed_request
  • redirect_uri tampering and stolen signed_request means authentication bypass

• Lessons:
  • Facebook PHP SDK is not for secure authentication
  • Don’t trust code from external SDK
RPC issues
Facebook RPC showDialog workflow

- App communicate with Facebook RPC controller through FB_RPC messages
- App can invoke a special RPC method showDialog
- To render the dialog, Facebook controller makes an XHR request and parses the JSON payload
- XHR endpoint uiserver.php also serves as OAuth 2.0 endpoint
- We control most of query parameters for uiserver.php (redirect_uri)
Facebook RPC showDialog workflow

1. postMessage() / Flash
2. XdArbiter.handleMessage()
3. XHR
4. XHR handler
Facebook RPC showDialog workflow

Guess, how is JSON payload parsed?
Facebook RPC showDialog workflow

We could trick the Facebook app controller with OAuth 2.0 redirects and submit malicious payload to the XHR handler:

```javascript
_handleXHRResponse: function(ka) {
  var la;
  if (this.getOption('suppressEvaluation')) {
    la = {asyncResponse: new h(this, ka)};
  } else {
    var ma = ka.responseText, na = null;
    try {
      var pa = this._unshieldResponseText(ma);
      try {
        var qa = (eval)('(' + pa + ')');
      }
    }
  }
}
```

XHR cross-domain redirects are not permitted, but let's knock it down up to cross-site scripting anyway.
Yet another JS SDK issue: Flash XD transport

- `redirect_uri` parameter of `showDialog` method must belong to app’s own domain, which is defined in `xd_arbiter` proxy url

- Two flaws in Flash cross-domain transport allowed to hijack the origin and to send FB_RPC messages on behalf of `facebook.com`:
  - Controllable Flash channel names
  - Absence of secret nonce validation

- Exploitation:
  - Inject two `xd_arbiter` proxies with transport=flash
  - Connect them by setting the same Flash channel name
  - Inject the third `xd_arbiter` and let him initiate the flow with `proxyMessage()`
Yet another JS SDK issue: Flash XD transport

- Secret nonce validation — insufficient, another nonce check here is missing
- Adobe LocalConnection() channel names are controlled by xd_arbiter url
- proxyMessage()
XSS with OAuth 2.0

- Now we send FB_RPC message on behalf of facebook.com and invoke showDialog method.
- `redirect_uri` parameter in FB_RPC message is `http://facebook.com/…something`, and it passes all checks.
- Wrapping a small stage-0 malicious payload inside a picture.
- Proxying the picture from our site through `facebook.com/safe_image.php`.
XSS with OAuth 2.0

1. proxyMessage()

2. connection.send()

3. redirect_uri = gif

4. eval()

Facebook.com

App.com Canvas iframe

FB proxy iframe

Proxy swf

FB proxy iframe

Proxy swf

FB proxy iframe
XSS with OAuth 2.0

• Lessons:
  • XSS is not only about $q=<script>alert($, design flaws are unique
  • eval is still evil, nothing new
  • OAuth redirects can be abused for taint propagation in your javascript apps
Conclusion
Endless attack vectors for Facebook OAuth

- `redirect_uri` tampering
- Sensitive data leakage through Referrer
- Token transport (JS SDK and xd_arbiter.php)
- `#!` (hash-bang) and redirect_uri filtering bypasses
- SDK authentication (PHP SDK, … SDK)
- App RPC exploiting
Endless attack vectors for Facebook OAuth

- **redirect_uri tampering**
  - Sensitive data leakage through Referrer
  - `#!` (hash-bang) and redirect_uri filtering bypasses
  - Token transport (JS SDK and xd_arbiter.php)
  - SDK authentication (PHP SDK, ... SDK)
  - App RPC exploiting
Q&A

Thanks!