OAuth2.0:
the Promise and Pitfalls

Sergey Ozernikov – Security Consultant
OWASP NZ Day
4th February 2016
Company Overview

Company
- Lateral Security (IT) Services Limited
- Founded in April 2008 by Nick von Dadelszen and Ratu Mason (both Directors)
- Auckland, Wellington, Christchurch: ~20 highly specialised security consultants

Services
- Security testing (design & architecture, penetration testing, configuration, code reviews, security devices & controls, mobile apps)
- Security advisory (Lifecycle compliance & audit – ISO, PCI-DSS, NZISM, policy process development, threat modeling and risk assessment)
- Regular ongoing technical testing and assurance programs
Outline

• Introduction – Why it matters?
• OAuth2.0 basics – Terms and flows
• Threat model
• Examples of vulnerabilities
• Resources
• Conclusion
Introduction

- The OAuth 2.0 authorisation framework enables a resource owner (User) to allow a third-party application (Client) to obtain a certain level of access to an HTTP service (Provider).
- Published in October, 2012 as an IETF standard.
- Facebook supports OAuth 2.0. Google supports OAuth 2.0. Microsoft supports OAuth 2.0. It’s everywhere.
- Twitter is one of the few exceptions still using OAuth 1.0a for sensitive API.
- It’s an authorisation framework, not authentication! But often used to outsource authentication too.
Introduction

• What happens if things go wrong?

• Why? Because it’s an authorisation framework!
Introduction

• Have things ever gone wrong?

Facebook said to fix OAuth-based account hijacking flaw

The vulnerability could have allowed attackers to steal OAuth tokens and access Facebook account, a researcher says

Serious security flaw in OAuth, OpenID discovered

Attackers can use the "Covert Redirect" vulnerability in both open-source log-in systems to steal your data and redirect you to unsafe sites

FIVE OAUTH BUGS LEAD TO GITHUB HACK

by Chris Brook

February 11, 2014, 10:53 am

Security

Outlook.com had classic security blunder in authentication engine

Redmond pays $25k to hacker who spotted flaw allowing anyone to own your email
Introduction

• What should we do?
Basic terms

- **User** (Resource owner) - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

- **Client** - An application making protected resource requests on behalf of the user and with their authorisation.

- **Authorisation Server** - The server issuing access tokens to the client after successfully authenticating the user and obtaining authorisation.

- **Resource Server** - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

- **Authorisation Server + Resource Server = Provider**
OAuth2.0 Flow (Example)

• You came across a great web app to help you post cute puppies on your Facebook page.
• Of course you instantly click “I want it now” on the app’s web site and it redirects you to Facebook.
• Facebook checks that you’re logged in and asks “Really? This app wants the right to post puppies on your behalf”.
• You say “Sure!” and approve it.
• You get redirected back to the app’s web site.
• Puppy magic happens.
OAuth2.0 Flow (Another example)

- You came across a great web site with puppies.
- Unfortunately, to be able to ‘like’ puppies, you’re required to sign up.
- You don’t want to sign up.
- You luckily notice a “Sign in with Facebook” button and instantly click on it.
- You’re getting redirected to Facebook and it asks: “Really? This app wants to access your profile, email and friend list”.
- You say “Sure!” and approve it.
- You get redirected back to the web site.
- You can now ‘like’ puppies using your Facebook user id.
Authorisation Code flow

Initiating the process

Redirecting the User to the Provider

Following the redirect and authenticating if needed

Checking authentication and showing approval UI

Approving Client’s access to Users’ resources

Redirecting the User to redirect_uri with the access code

Following the redirect

Providing access code

Responding with access token

Accessing User’s resources
Implicit flow

Initiating the process

Redirecting the User to the Provider

Following the redirect and authenticating if needed

Checking authentication and showing approval UI

Approving Client’s access to Users’ resources

Redirecting the User to redirect_uri with the access token

Following the redirect

Accessing User’s resources
Pros and Cons

• **Pros:**
  • Simple specification
  • Easy to implement
  • It’s everywhere!

• **Cons:**
  • Simple specification
  • Easy to implement
  • It’s everywhere!

(To Do: List Proper Cons Here!)
Pros and Cons

**Pros:**
- Simple specification
- Easy to implement
- It’s everywhere!

**Cons:**
- Hard to implement securely
- No crypto by default
- A lot of security controls now become implementation/environment-specific
Threat Model

- Section 10 of **RFC6749** “The OAuth 2.0 Authorization Framework”. (7 pages out of 75)
- **RFC6819** “OAuth 2.0 Threat Model and Security Considerations”. (69 pages)

- Various flow types: Authorisation Code flow vs. Implicit flow
- Various attack scenarios: are you a Client or Provider?

- Bearer tokens are worse than session cookies: longer lifespan and less secure storage/management.
- Tokens are not bound to a Client by default. A few big providers rolled out their own mitigations (Google, Facebook, Github).
Vulnerability #1 – CSRF on “Approve” button

- Malicious **Client** is trying to access **User’s** data on the **Provider’s** side.

- Vulnerability is on the **Provider’s** side - cross-site request forgery on **User’s** approving **Client’s** access to their data.

- As a result, malicious **Client** obtains unauthorised access to **User’s** resources.

- References:
Vulnerability #1 – CSRF on “Approve” button

User

- Initiating the process and redirecting the User to the Provider

Client = Attacker

- Following the redirect. User is currently authenticated.
- Checking authentication and showing approval UI
- Approving Client’s access to Users’ resources
- Redirecting the User to redirect_uri with the access code

Provider

- Following the redirect
- Providing access code
- Responding with access token
- Accessing User’s resources
Vulnerability #2 – CSRF on Adding a Client

• An attacker is trying to bind their **Provider** account to **User’s Client** account.

• Vulnerability is on the **Client’s** side – CSRF again. ‘State’ parameter is not properly checked.

• As a result, the attacker can:
  • Trick the **User** to update/upload their sensitive data to the resource controlled by the attacker (in the first scenario).
  • Log in as the **User** on the **Client** application using attacker’s **Provider** account (in the second, “Social login” scenario).

• References:
  • [http://tools.ietf.org/html/rfc6819#section-4.4.1.8](http://tools.ietf.org/html/rfc6819#section-4.4.1.8)
Vulnerability #2 – CSRF on Adding a Client

 iniciating the process

 redirecting to the Provider

 following the redirect and authenticating as an attacker if needed

 checking authentication and showing approval UI

 approving Client’s access to Attacker’s resources

 redirecting to redirect_uri with the access code

 user

 following the redirect

 Attacker

 provider

 providing access code

 responding with access token

 accessing Attackers’s resources
Vulnerability #3 – Open Redirect On the Client

- Attacker tries to obtain access token from a vulnerable **Client** to access **User’s** resources on the **Provider**.
- Caused some media hype as “Covert redirect” in 2014.
- Several prerequisites:
  - Open redirect on the **Client**
  - **Provider** should allow dynamic ‘redirect_uri’
  - **Provider** should allow ‘response_type=token’
- Could be used with ‘response_type=code’, but exploitation would depend on other things as well.

- References:
  - [http://tools.ietf.org/html/rfc6819#section-4.1.5](http://tools.ietf.org/html/rfc6819#section-4.1.5)
Vulnerability #3 – Open Redirect On the Client

User → Client 
Initiating the process and redirecting the User to the Provider
Following the redirect with a crafted redirect_uri and authenticating if needed

Client → Provider
Checking authentication and showing approval UI
Approving Client’s access to Users’ resources
Redirecting the User to the crafted redirect_uri with the access token
Following the redirect
https://client.com/redirector?
target=attacker.com#token=123

Open redirector leaks the access token to
http://attacker.com#token=123

Provider → Attacker
Attacker Accessing the resource
Attacker

https://client.com/redirector?
target=attacker.com#token=123
Vulnerability #3 – Open Redirect On the Client

A HINT:
If you’re using in your web application, make sure you specify an exact ‘redirect_uri’ parameter within the settings of your client application on Facebook.
In case it’s left blank – it’s allowed to be dynamic!
Resources

- [http://www.oauthsecurity.com/](http://www.oauthsecurity.com/)
- Major OAuth providers have their own security check-lists, e.g.:
  - [https://developers.facebook.com/docs/facebook-login/security](https://developers.facebook.com/docs/facebook-login/security)
Conclusion

- Understand the threat model and how it applies to you.
- Use additional crypto. E.g.:
  - [https://developers.facebook.com/docs/graph-api/securing-requests](https://developers.facebook.com/docs/graph-api/securing-requests)
- Don’t use implicit flow where possible – use authorisation codes.
- Use out-of-the box, popular solutions and libraries (and don’t forget to update them).
- Do a pentest after you’re done!
Questions and Contacts

Lateral Security (IT) Services Limited

Wellington
69 The Terrace (level 5, Gleneagles House)
PO Box 8093, Wellington 6011, New Zealand
Phone: +64 4 4999 756
Email: sas@lateralsecurity.com

Auckland
53 High Street (level 1)
PO Box 7706, Auckland, New Zealand
Phone: +64 9 3770 700
Email: sas@lateralsecurity.com

Christchurch
36 Byron Street (level 1)
Sydenham 8023, Christchurch, New Zealand
Phone: +64 35950387
Email: sas@lateralsecurity.com

Presentation Download
www.lateralsecurity.com/presentations