OAuth 2.0 Redirect URI Validation Falls Short, Literally

Tommaso Innocenti
Northeastern University
Boston, MA, USA

Matteo Golinelli
University of Trento
Trento, Italy

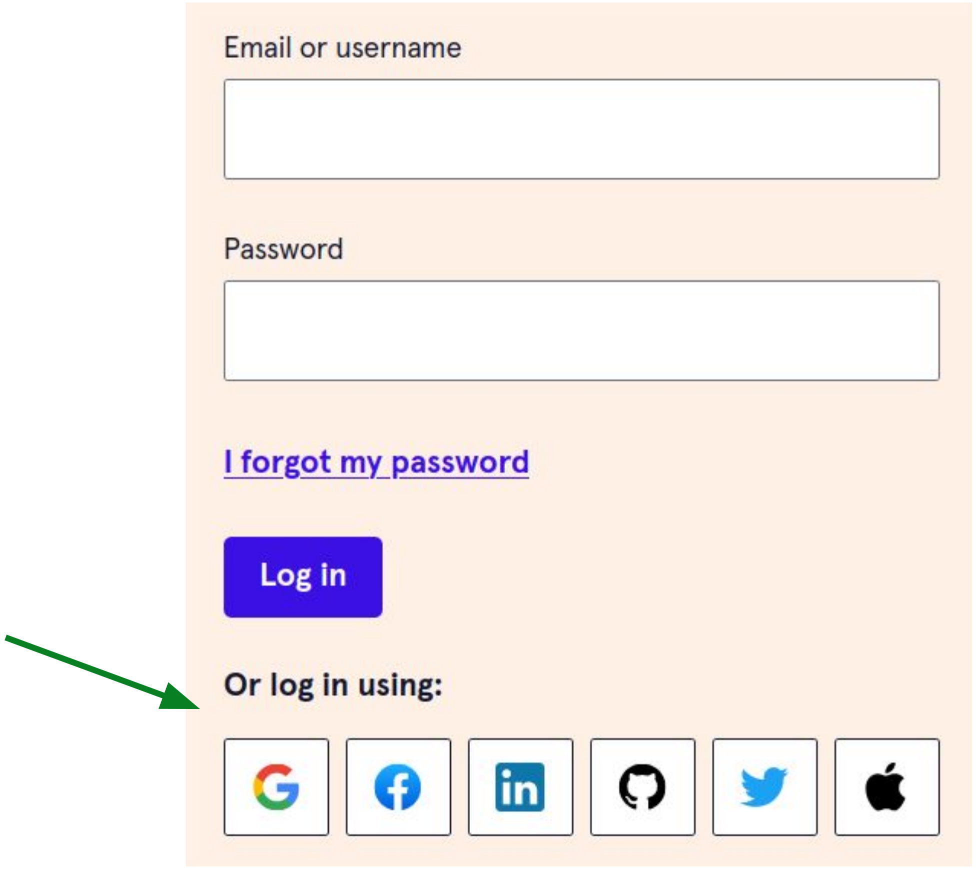
Kaan Onarlioglu
Akamai Technologies and
Northeastern University*
Cambridge, MA, USA

Ali Mirheidari
Independent Researcher
Austin, TX, USA

Bruno Crispo
University of Trento
Trento, Italy

Engin Kirda
Northeastern University
Boston, MA, USA

OAuth 2.0


Email or username

Password

Or log in using:

Background: OAuth 2.0

Secure **delegated access framework**

- Enables *Resource Owners* to grant a *Client* website access to their data hosted on a third-party *Resource Server*
- **Authorization** is granted via an *Authorization Server*, instead of sharing the Resource Owner credentials with the Client

Defines 4 grant types

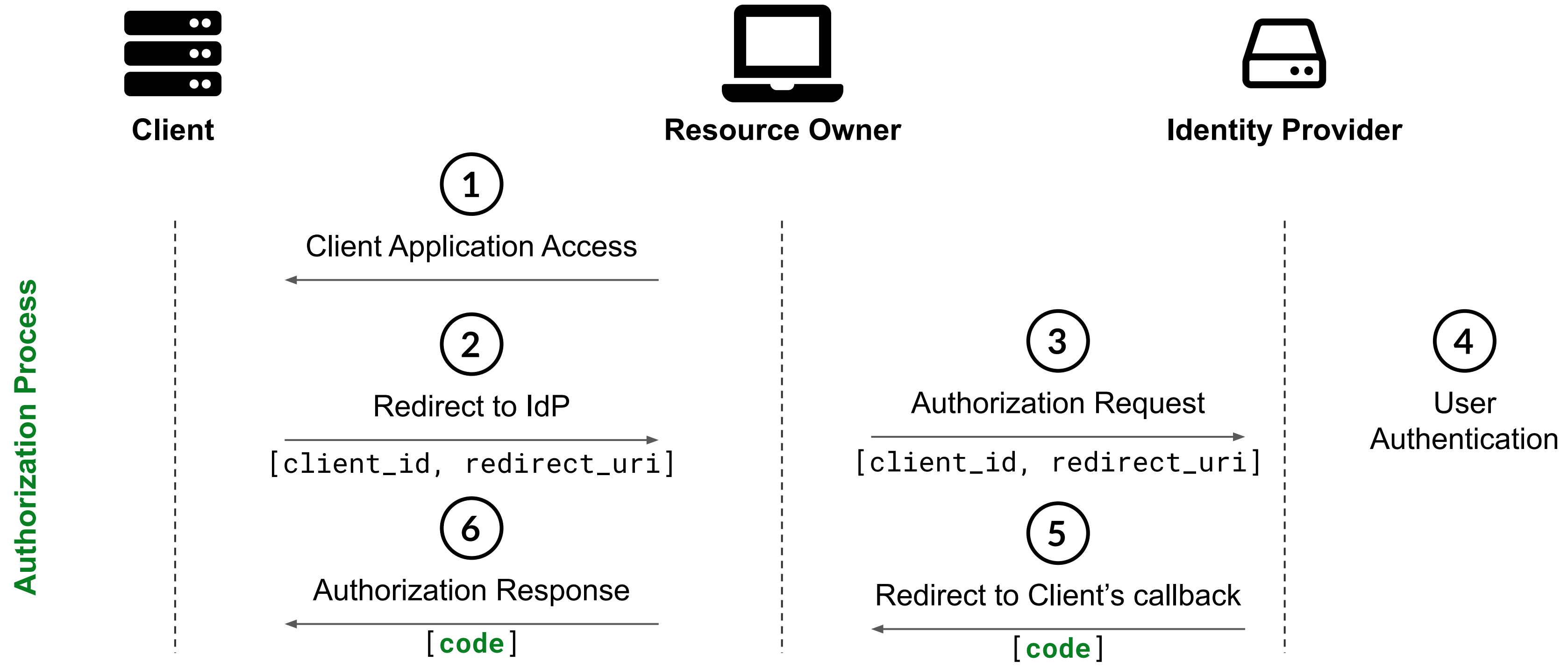
- **Authorization Code Grant** is the most common

Background: Authorization Code Grant

Web applications (*Client*) access the data of **internet users** (*Resource Owner*) by authenticating to an **Identity Provider** (*IdP*: generally a combination of *Authorization Server* and *Resource Server*)

- The Client must first **establish a trust relationship** with the IdP by registering their application
  - Set up a callback endpoint called **`redirect URI`**
  - Receive a **`client ID`** and **`client secret`**

Background: Authorization Code Grant



Research Statement: RFC 6749 & RFC 3986

RFC 6749 Section 3.1.2.3 *The authorization server MUST compare the two URIs using simple string comparison as defined in RFC 3986 Section 6.2.1.*

RFC 3986 Section 6.2.1 *Testing strings for equality is normally based on pair comparison of the characters that make up the strings, starting from the first and proceeding until both strings are exhausted, and all characters are found to be equal, until a pair of characters compares unequal, or until one of the strings is exhausted before the other.*

What if two URIs have a **matching prefix**, but **different lengths**?
The string comparison is a failure or a success?

How to interpret the RFC?

Should IdPs interpret this ambiguity as an **intentional flexibility**?

- e.g., support *dynamic path components* or *query parameters* in `redirect URI`

This validation scheme prevents tampering with the **host** or **domain** name included in a redirect URI

- Falls short of detecting potentially **malicious additions** to the path and query string that follow

Background: Path Confusion

Attacks that abuse **URI parsing discrepancies** within complex system interactions

- Appending **maliciously crafted path components** to a URL
  - ↪ Confuse modern URL parsers designed to accommodate complex URL rewriting and routing mechanisms
  - ↪ Induce discrepancies between multiple parsers in a complex system (e.g., on an origin server and on a CDN cache)

```
https://client.com/callback%2Frandom%2F%2e%2e
```

Is OAuth 2.0 vulnerable to Path Confusion attacks?

Attack: replace the legitimate redirect URI parameter in OAuth 2.0 flows with path confusion payloads

- Determine which *IdPs fail to detect this malicious modification* through validation

Impact: the IdP redirects the victim's user agent to an unintended endpoint on the Client site

Path Confusion Attack Consequences

The authorization code is delivered to a maliciously modified callback endpoint on the Client

- The **code** remains unused by the Client
    - If stolen, can be used to get access to the Resource Owner’s data

- An attacker can steal the code by redirecting to an **arbitrary enpoint vulnerable to data exfiltration**
    - XSS, open redirect, third-party code inclusion
    - Multi-tenant websites

Methodology: Path Confusion vulnerabilities detection

On each website to test:

1. Identify the HTML elements that start an OAuth flow
2. Trigger an OAuth flow
3. Use a proxy to intercept the flows and **inject our path confusion payloads** into the redirect URI
4. Collect all the network traffic
5. Analyse the data collected and **check if the redirection was hijacked**

Results: Path Confusion

We tested the websites in the **Tranco Top 15k**

- 728 websites supporting OAuth
- Selected the IdPs used by at least 3 Clients and that do not require personal information when registering an account: **22 IdPs in scope**

6 IdPs tested did not correctly validate the *redirect URI* and were **vulnerable to path confusion**

Facebook, Microsoft, GitHub, Atlassian, NAVER, and VK

Mitigations

Redirect URI validation should use **strict string equality check**

IdPs should **never sanitize** the redirect URI to avoid introducing discrepancies, instead they should **validate** it

Responsible Disclosure

We contacted all the impacted *Identity Providers*

- Microsoft acknowledge our report and fixed their validation procedure
- GitHub is tracking the problem internally and is actively working on a fix
- Naver fixed the issue and rewarded us with a bug bounty
- The other acknowledged our reports but did not provide information on fixes

We reported our finding to the *OAuth Working Group*:

- They **update to the OAuth 2.0 Security Best Current Practice**, clarifying the requirement for an exact string match during redirect URI validation

  https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics#name-countermeasures

- The *OpenID foundation* modified the conformance test suite to include our attack

UNIVERSITY OF TRENTO

Conclusions

Northeastern University

The current "best practice" is not good enough, leaving IdPs, Clients, and Internet users exposed to attacks

- Path confusion
- OAuth Parameter Pollution (OPP): find details in the full paper

The vulnerabilities we discovered are **not implementation bugs**

- They are **rooted in the OAuth 2.0 specification** where language is not prescriptive enough
- IdPs that follow the RFCs still risk exposing redirect URI validation vulnerabilities

Extra slides

Research Statement: RFC 6749

RFC 6749 Section 3.1 *The endpoint URI MAY include an "application/x-www-form-urlencoded" formatted [...] query component [...], which MUST be retained when adding additional query parameters.*

RFC 6749 Section 10.14 *A code injection attack occurs when an input or otherwise external variable is used by an application unsanitized and causes modification to the application logic. This may allow an attacker to access the application device or its data, cause a denial of service, or introduce a wide range of malicious side-effects. The authorization server and Client MUST sanitize (and validate when possible) any value received–in particular, the value of the "state" and "redirect_uri" parameters.*

Background: Parameter Pollution