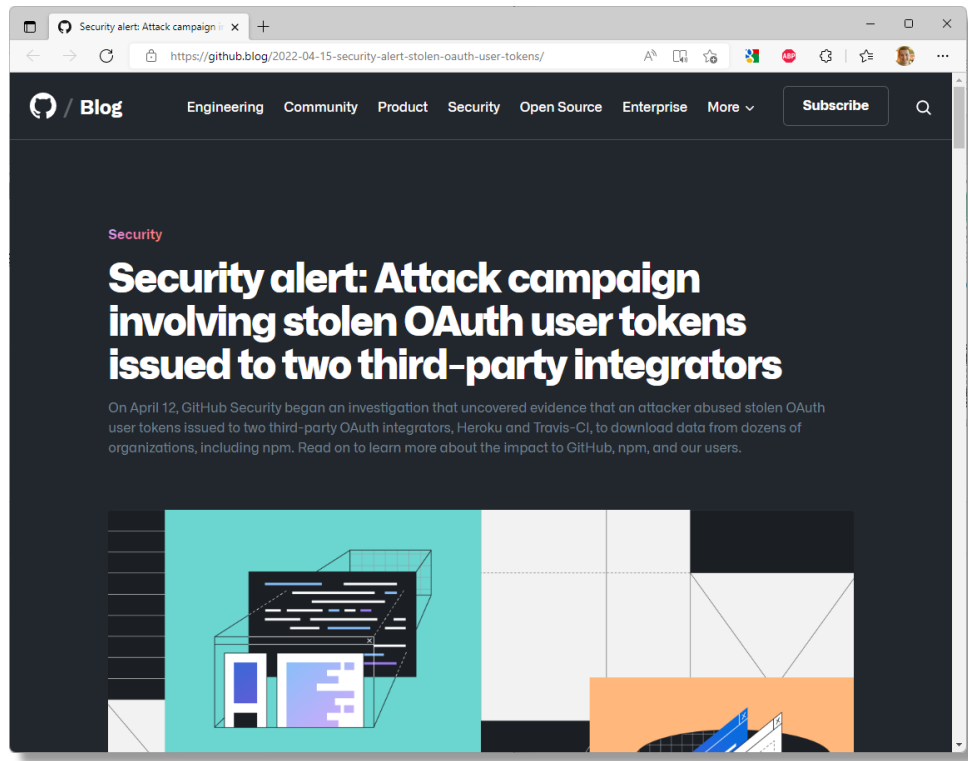


Analyzing the Compliance of OAuth 2.0 Implementations

Pieter Philippaerts



"We do not believe the attacker obtained these tokens via a compromise of GitHub [...] because the tokens in question are not stored by GitHub in their original, usable formats"

- Mike Hanley, chief security officer, GitHub

"Once you have implemented OAuth2, how do you know you have implemented it correctly?"

SSL Server Test: www.google.com

https://www.ssllabs.com/ssltest/analyze.html?d=www.google.com&s=172.217.6.68&hideResul...

Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > [www.google.com](#) > 172.217.6.68

SSL Report: [www.google.com](#) (172.217.6.68)

Assessed on: Mon, 20 Jul 2020 06:35:49 UTC | [HIDDEN](#) | [Clear cache](#) [Scan Another »](#)

Summary

Overall Rating

B

Certificate 100
Protocol Support 70
Key Exchange 90
Cipher Strength 90

Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This server supports TLS 1.0 and TLS 1.1. Grade capped to B. [MORE INFO »](#)

This server supports TLS 1.3.

Static Public Key Pinning observed for this server.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO »](#)

DNS Certification Authority Authorization (CAA) Policy found for this domain. [MORE INFO »](#)

Scan results for www.facebook.com

https://securityheaders.com/?q=www.facebook.com&foll...

Security Headers

Sponsored by [Report URI](#)

Home About Donate

Scan your site now

☐ Hide results ☒ Follow redirects

Security Report Summary

A

Site: <https://www.facebook.com/>

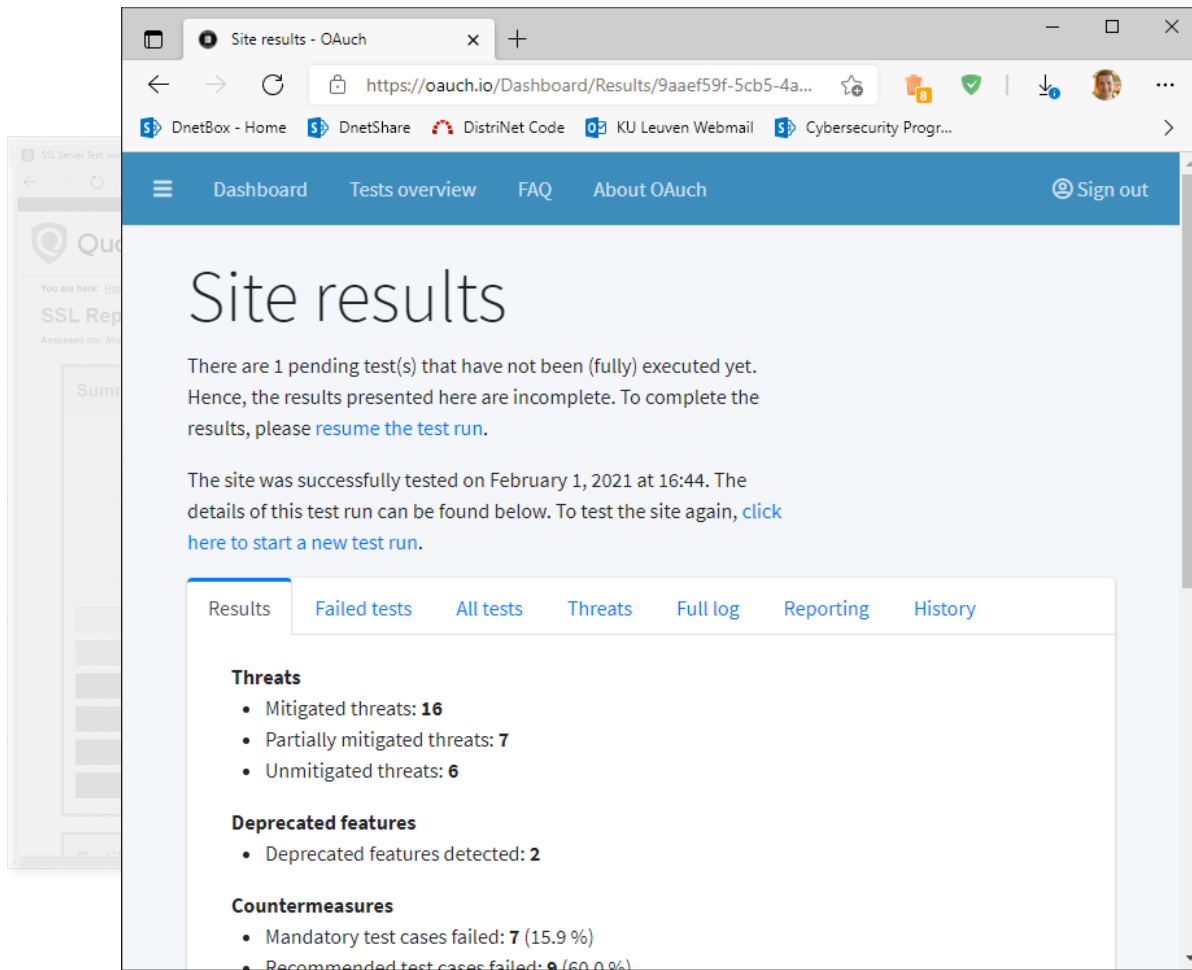
IP Address: 2a03:2880:f131:83:face:b00c:0:25de

Report Time: 20 Jul 2020 10:51:59 UTC

Headers: ☒ Strict-Transport-Security ☒ Content-Security-Policy
☒ X-Content-Type-Options ☒ X-Frame-Options ☒ Referer-Policy
☒ Feature-Policy

Warning: Grade capped at A, please see warnings below.

Supported By



Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: 5849
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft
October 2012

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc6749>.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

Authorization servers **MAY** issue refresh tokens to web application clients and native application clients.

Refresh tokens **MUST** be kept confidential and only shared among the authorization server and the client that issued them. The authorization server **MUST** ensure the binding between a refresh token and the client that issued it. Refresh tokens **MUST** only be used as described in Section 1.6 with servers that support [RFC2818].

The authorization server **MUST** verify the token and client identity whenever a client requests a refresh token. When a client authenticates, the authorization server **SHOULD** deploy measures to prevent token abuse.

A complex maze representing the OAuth 2.0 specification. The maze starts at a cat icon at the bottom center, with an arrow pointing upwards. The path through the maze leads to the 'TOKEN EXCHANGE' box. Various boxes are labeled with OAuth 2.0 terms and RFC numbers, including: POP, TOKEN BINDING, RFC7662, UMA 2, HTTP SIGNING, RFC8414, CSRF, STATE PARAM, CLIENT TYPE, RFC6750, RFC6749, RFC7518, RFC7517, RFC7516, RFC7515, FAPI, OIDC, CIBA, SECURITY BCP, TLS, GRANT TYPE, JARM, JAR, and MUTUAL TLS.

Building a test case

The client MUST NOT use the authorization code more than once.

- » OAuch tries to use the same authorization code two times and keeps track of the server's response

Test case coverage

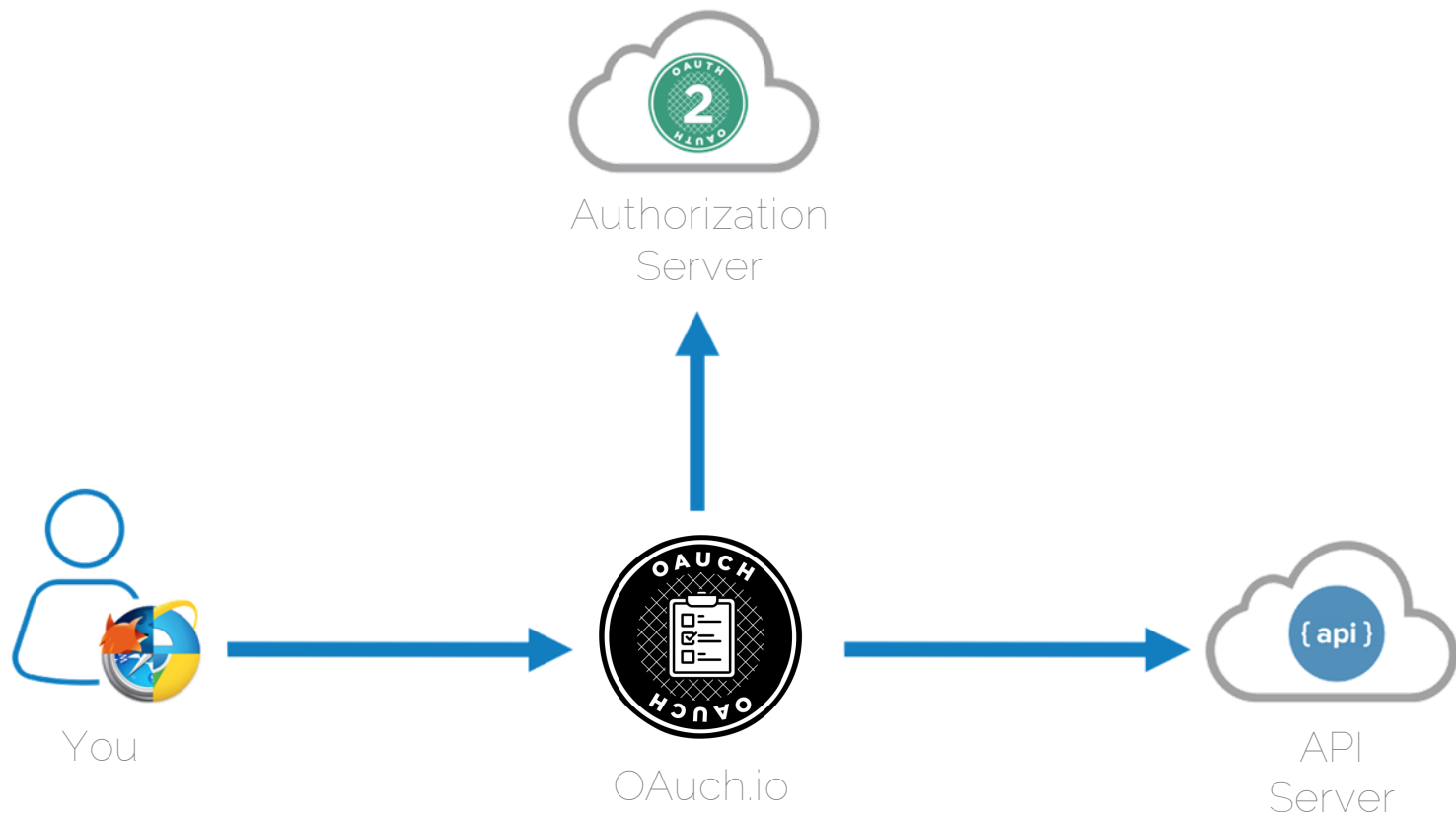
- › OAuch implements 112 unique test cases from 10 documents
 - » Many documents contain the same requirements
 - » If a requirement has varying requirement levels, OAuch picks the strictest one
- › Not all security requirements can be converted to test cases

Testing Process

- › OAuch is set up like any other client
 - » ... but acts like a malicious client!
- › Access token validation requires an API endpoint
 - » **HTTP 2xx** → access token is valid
 - » **HTTP 4xx/5xx** → access token is invalid

Testing Process

- › OAuch detects which features are enabled on the server
 - » The relevant test cases are selected and run
 - » OAuch keeps a detailed log, that can be inspected by the user
- › Result: a full overview of which countermeasures are enabled on the server



Testing Process

- › OAuch detects which features are enabled on the server
 - » The relevant test cases are selected and run
 - » OAuch keeps a detailed log, that can be inspected by the user
- › Result: a full overview of which countermeasures are enabled on the server

But what does that mean?

OAuth Threat Model

Internet Engineering Task Force (IETF)
Request for Comments: 6819
Category: Informational
ISSN: 2070-1721

T. Lodderstedt, Ed.
Deutsche Telekom AG
M. McGloin

Name →

4.4.2.2. Threat: Access Token Leak in Browser History

Description →

An attacker could obtain the token from the browser's history. Note that this means the attacker needs access to the particular device.

List of countermeasures →

Countermeasures:

- o Use short expiry time for tokens (see Section 5.1.5.3). Reduced scope of the token may reduce the impact of that attack (see Section 5.1.5.1).
- o Make responses non-cacheable.

Information and how to provide feedback may be obtained at <http://www.rfc-editor.org/info/rfc6819>.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review the

OAuth Threat Model

- › OAuth integrates this threat model (+BCP) into the analysis
 - » 42 server-side threats are evaluated
 - » A threat can be full mitigated, partially mitigate or not mitigated
- › OAuth gives clear advice to a site owner
 - » Which threats your site might be vulnerable to
 - » Which countermeasures must be implemented to mitigate them

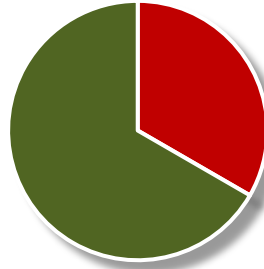
Limitations of OAuch

- › Only tests the authorization server
 - » Assumes no client-side mitigations
- › Only unintrusive tests
 - » No validation of DDoS countermeasures
- › The threat model assumes a powerful attacker
 - » Nation-state attackers

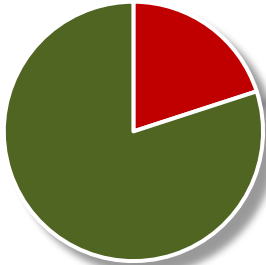
What we did

- › We tested 100 OAuth implementations
 - » 80 API providers, 20 OIDC providers
 - » 75 sites from Top 10000
 - » All publicly available (so they *should* be secure)
- › We drew statistics over the sites and over the individual countermeasures/threats

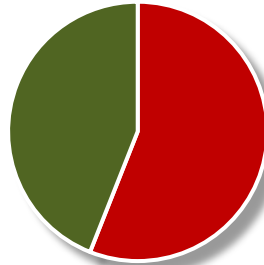
Results – Failure Rates



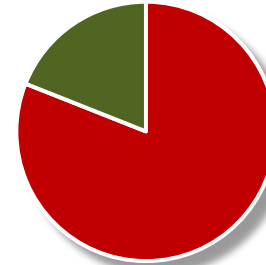
Overall: **33% FR**



Must: **20% FR**

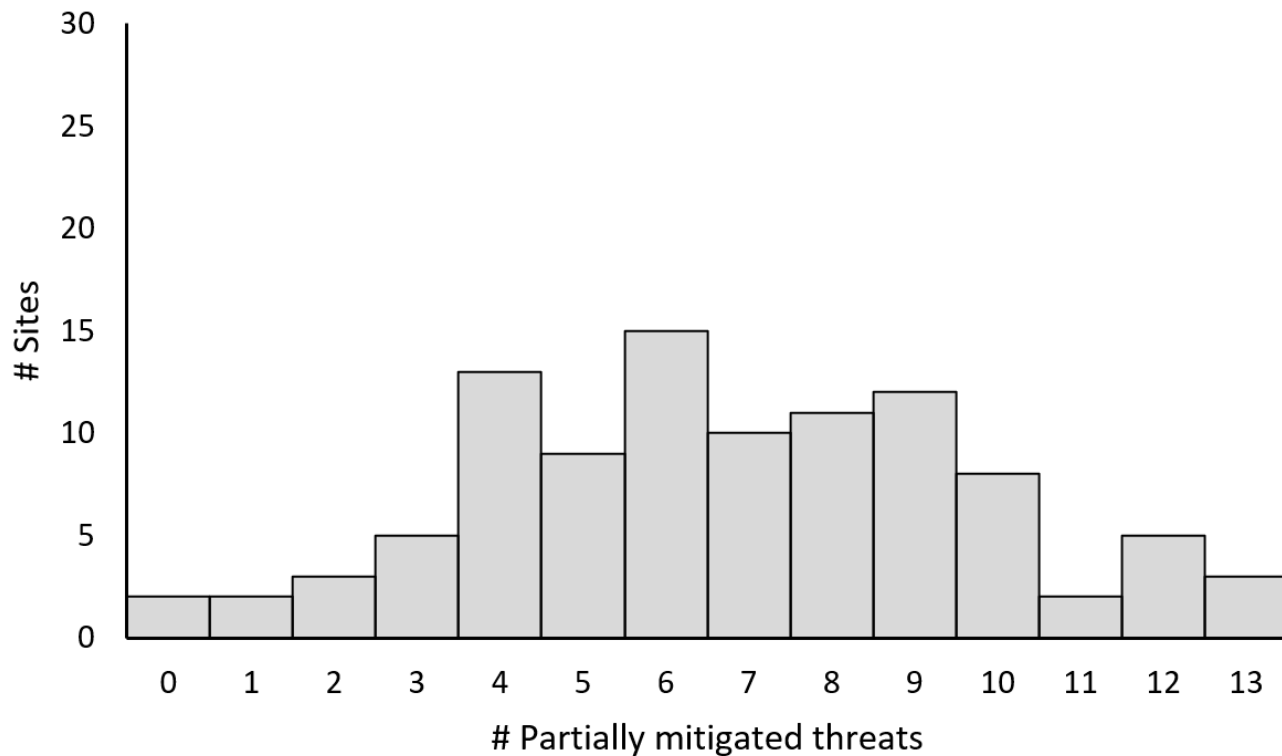


Should: **56% FR**

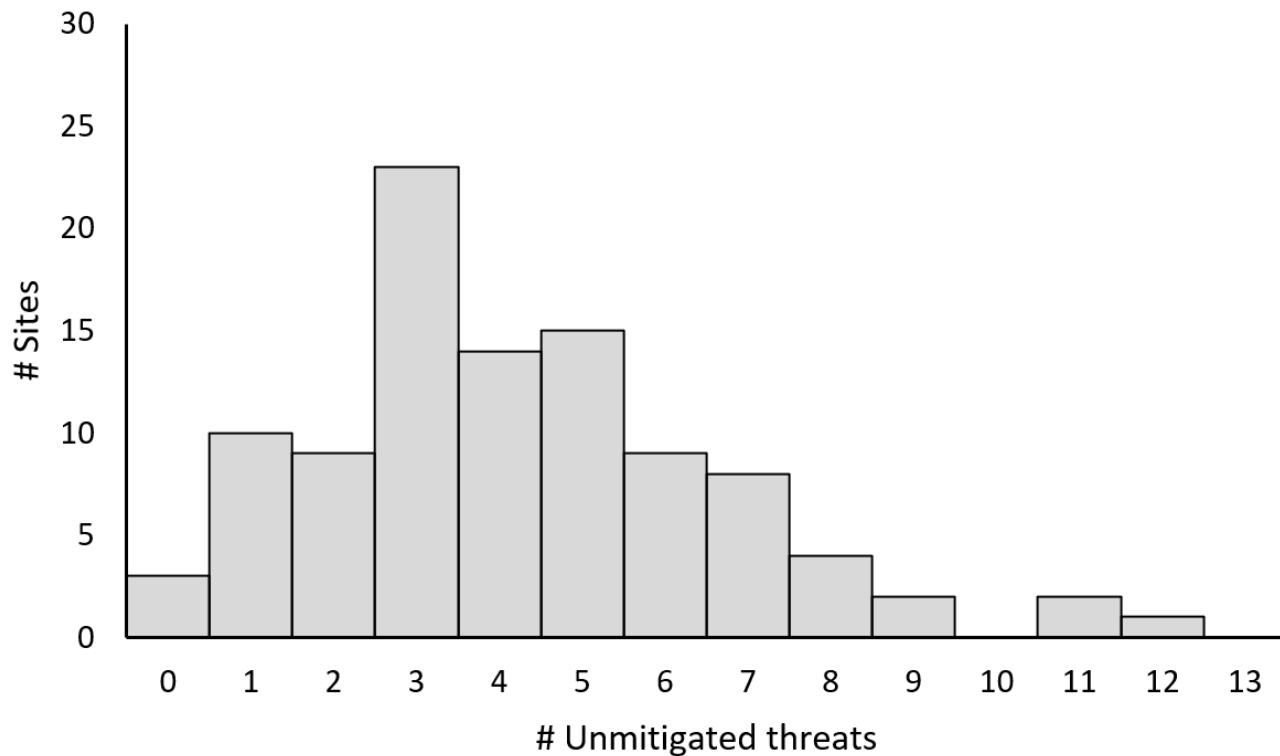


May: **81% FR**

Results – Partially Mitigated Threats



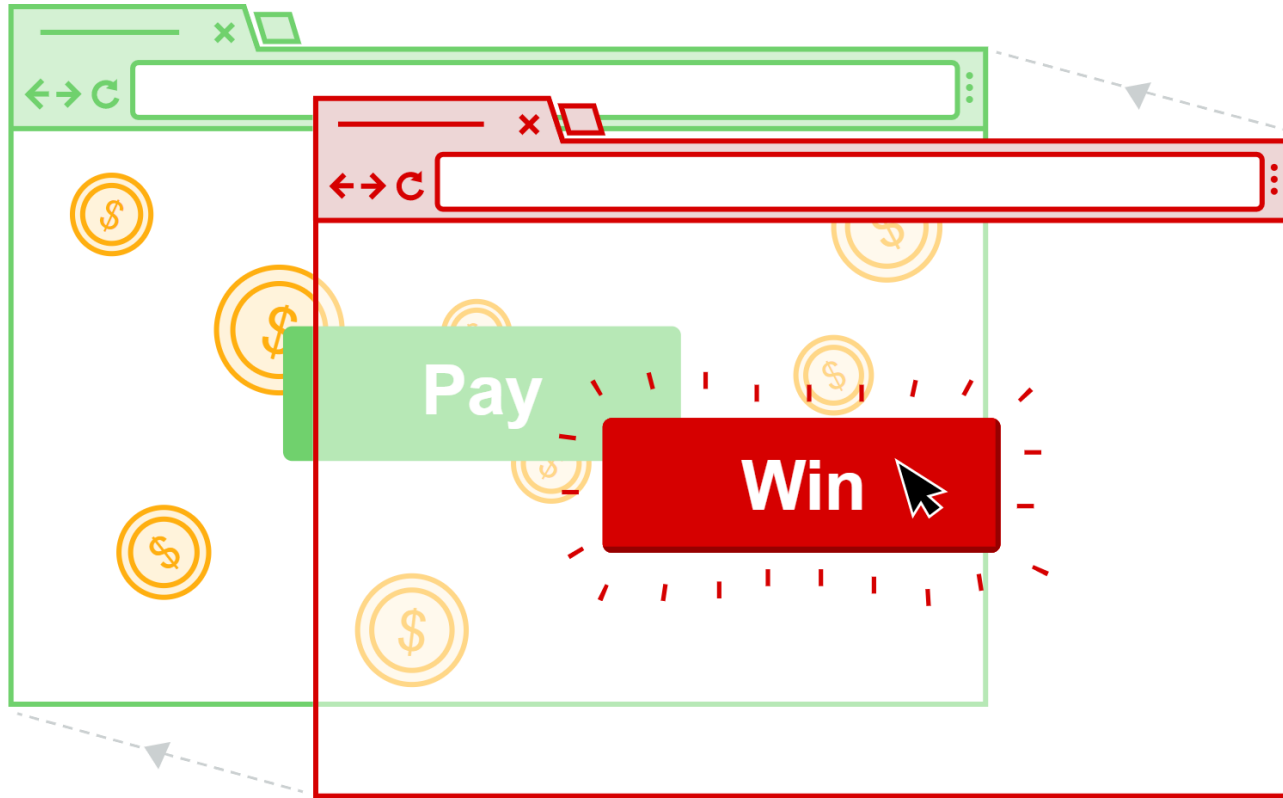
Results – Unmitigated Threats



Confirming the Results

- › To validate the results, we used OAuch as an offensive tool
 1. Choose an attack vector
 2. Use OAuch to list all vulnerable sites
 3. Try to write a proof-of-concept exploit

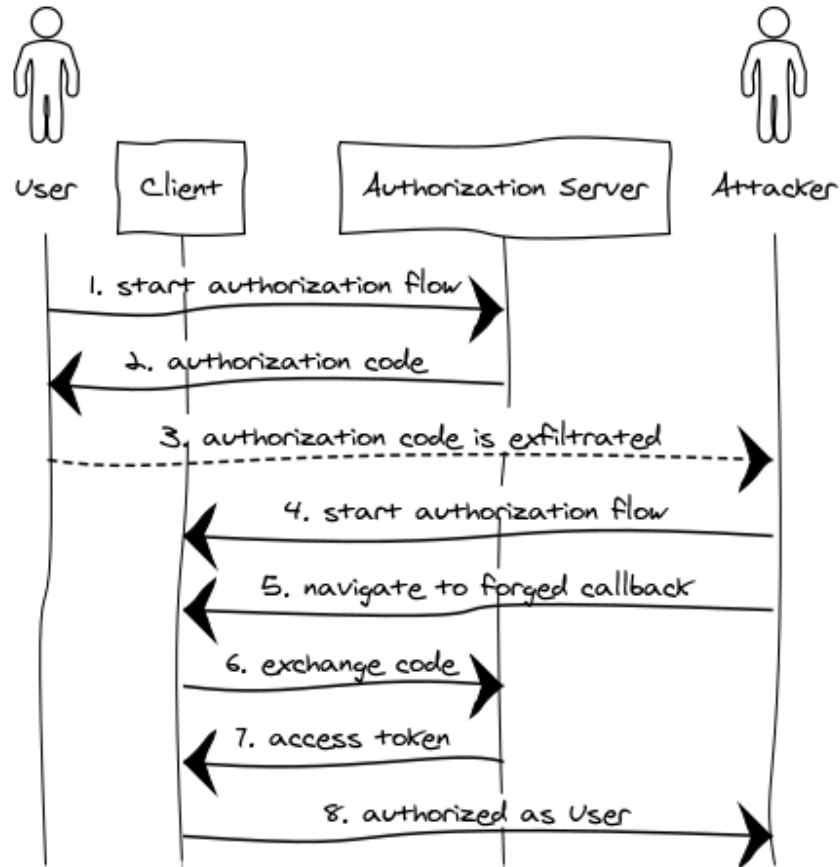
Confirming the Results – Clickjacking Attack



Confirming the Results – Clickjacking Attack

- › OAuch identified 22 sites that could be vulnerable to this threat
 - » After manual verification, 19 could be exploited (86% success rate)
 - » 2 sites used JavaScript to redirect to a secure page
 - » 1 site used frame-busting JavaScript

Confirming the Results – Authorization Code Injection



Confirming the Results – Authorization Code Injection

- › Focus on the OIDC providers
 - » Found clients for 12 OIDC providers
 - » These clients were tested for this vulnerability
 - »» Most clients were vulnerable
 - »» For each provider, at least one vulnerable client was found (100% success rate)

"Is it really that bad?"

Is it really that bad?

- › Yes and no.
 - » Yes, the servers do not (fully) mitigate certain threats
 - » No, the threat model assumes a powerful attacker
 - ›› Often complex exploitability
 - » No, OArch assumes no client mitigations

*"Why are OAuth implementations
lacking so many counter-
measures?"*

Why are implementations non-compliant?

- › The provider knows about it, but...
 - » ... wants to maintain backward compatibility
 - » ... some countermeasures cannot be efficiently implemented
 - » ... they have other development priorities
 - » ... doesn't care, because "it can be mitigated on the client side"

Why are implementations non-compliant?

- › The provider may not know about it, because...
 - » ... the original OAuth standard is outdated
 - » ... they make invalid assumptions
 - » ... they assume the OAuth library handles everything
 - » ... OAuth looks deceptively easy to implement

Lessons Learned?

- › It's hard to use these results to create generally applicable advice
 - » Everyone makes different mistakes
 - » OAuch gives tailor-made advice per site

Lessons Learned

- › Do not assume that a library is safe. Verify that it is.
- › Update your packages regularly. Security protocols evolve.
- › Do not rely on clients making great security decisions. Enforce them.

Lessons Learned

- › Having a formal verification of the OAuth2 protocol is great
 - » ... but we also need tools to verify practical implementations
- › A lot of sites can benefit from implementing missing countermeasures

Try it!

- › The tool is available on <https://oauch.io/>
 - » Let us know if we can improve something





Thank you!

<https://distrinet.cs.kuleuven.be/>

Pieter.Philippaerts@kuleuven.be