University of Stuttgart
Institute of
Information Security

Formal Security Analysis of the OpenID Financial-grade API

Daniel Fett, Pedram Hosseyni, Ralf Küsters

To appear at IEEE Security and Privacy 2019

2019-03-20

Motivation

- OpenID Financial-grade API
  - Profile of OAuth 2.0 Authorization Framework
- "Financial-grade":
  - "highly secured OAuth profile"
  - "to be used in write access to financial data [...] and other similar higher risk access"
  - "higher risk use cases"
- Such situations are extremely interesting for attackers ...

Some Recent Attacks ...

Security >

Date
Kredi

07.09.2018

Forbes Billionaires Innovation Leadership Money Consumer Industry Lifestyle

Aug 28, 2018, 01:57am

Laughing All The Way To The Bank: Cybercriminals Targeting U.S. Financial Institutions

FINANCIAL TIMES *my*FT

ons Carphone PLC + Add to myFT

s Carphone reveals attempt to hack almost 6m ecords

retailer fall 6% after attack on payments processing systems

Business

North Korea Hackers Tried to Take $1.1 Billion in Bank Attacks

By Yalman Onaran
October 8, 2018, 2:00 PM GMT+2 *Updated on October 9, 2018, 1:35 AM GMT+2*

Mexico central bank says hackers siphoned $15 million from five companies

Menü

Bankräuber erbeuten 1 Milliarde US-Dollar

Gemeinsam haben Interpol, Europol, Kaspersky Lab und andere Institutionen den bisher größten Cyber-Raubzug aufgedeckt. Seit dem Jahr 2013 habe die "Carbanak"-Gang Angriffe auf Banken in über 20 Ländern gestartet.

Objectives of our Work

- Create a Model of the Financial-grade API
  - Including: PKCE, mTLS, OAUTB, ...
- Capturing Security Goals and Assumptions
- Proof of Security Properties

Outline

- Model-based Approach
- Financial-grade API
  - Key Mechanisms
  - Attacker Model
- Security Properties
- Attacks Found through the Analysis

Outline

- Model-based Approach
- Financial-grade API
  - Key Mechanisms
  - Attacker Model
- Security Properties
- Attacks Found through the Analysis

WIM Previously Analyzed Protocols

Mozilla BrowserID

- Discovered severe attacks against authentication
- After fixes: Proof of authentication
- Special feature privacy: broken beyond repair

[S&P2014]
[ESORICS2015]

SPRESSO

- Designed from scratch
- First formalized in *WIM*, then implemented
- First SSO with proven privacy and security

[CCS2015]

OAuth 2.0

- Found several new attacks
- Developed fixes and implementation guidelines
- Proof of security

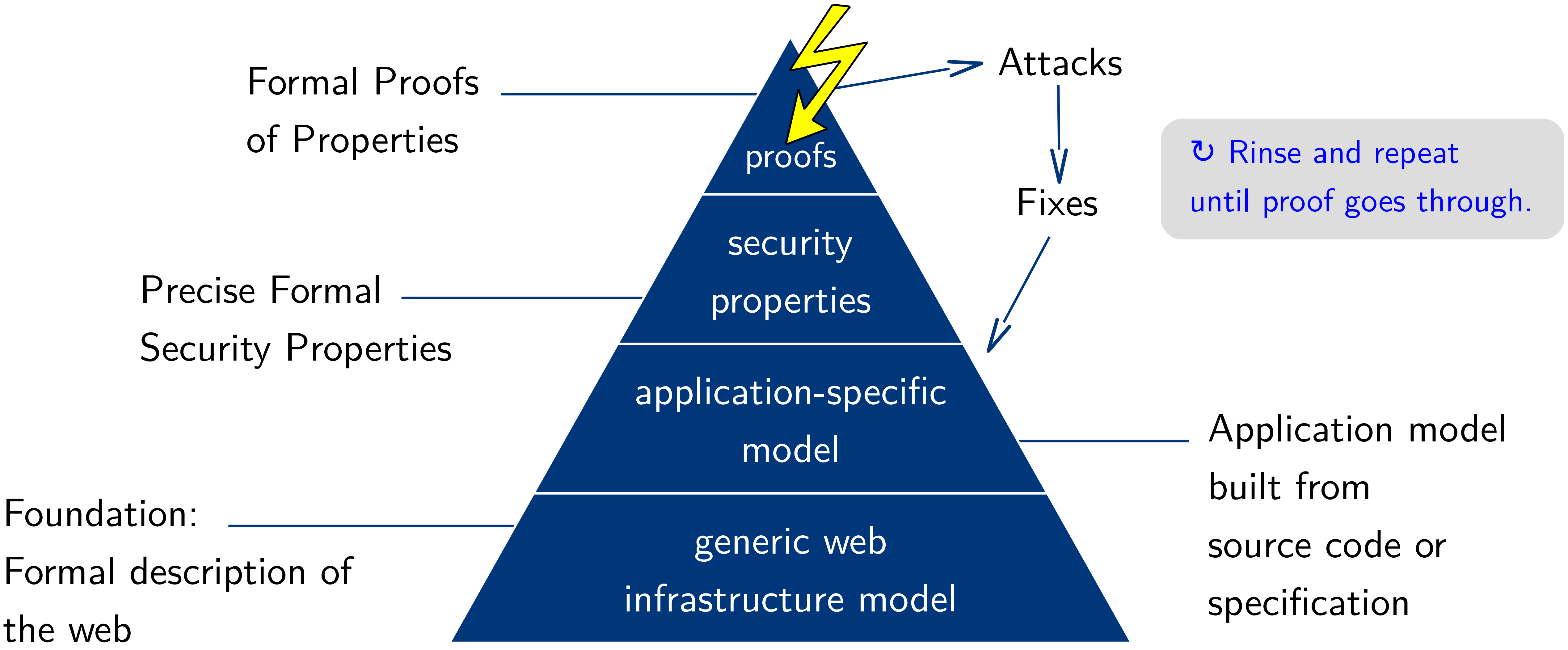
[CCS2016]

OpenID Connect

- Including extensions
- Developed best practices against known attacks
- Proof of security

[CSF2017]

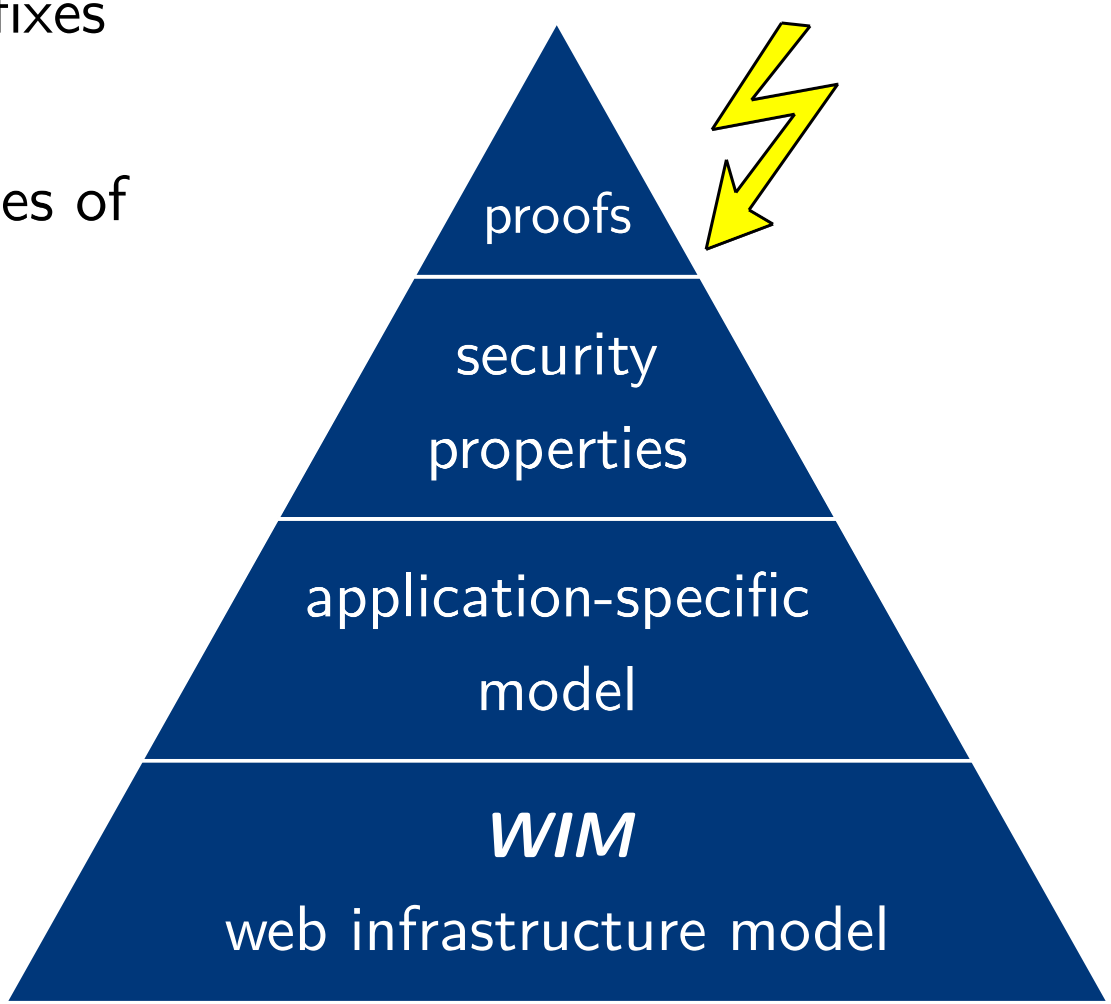
Our Model-Based Approach



Advantages

This approach can yield...

- new attacks and respective fixes
- strong security guarantees excluding even unknown types of attacks



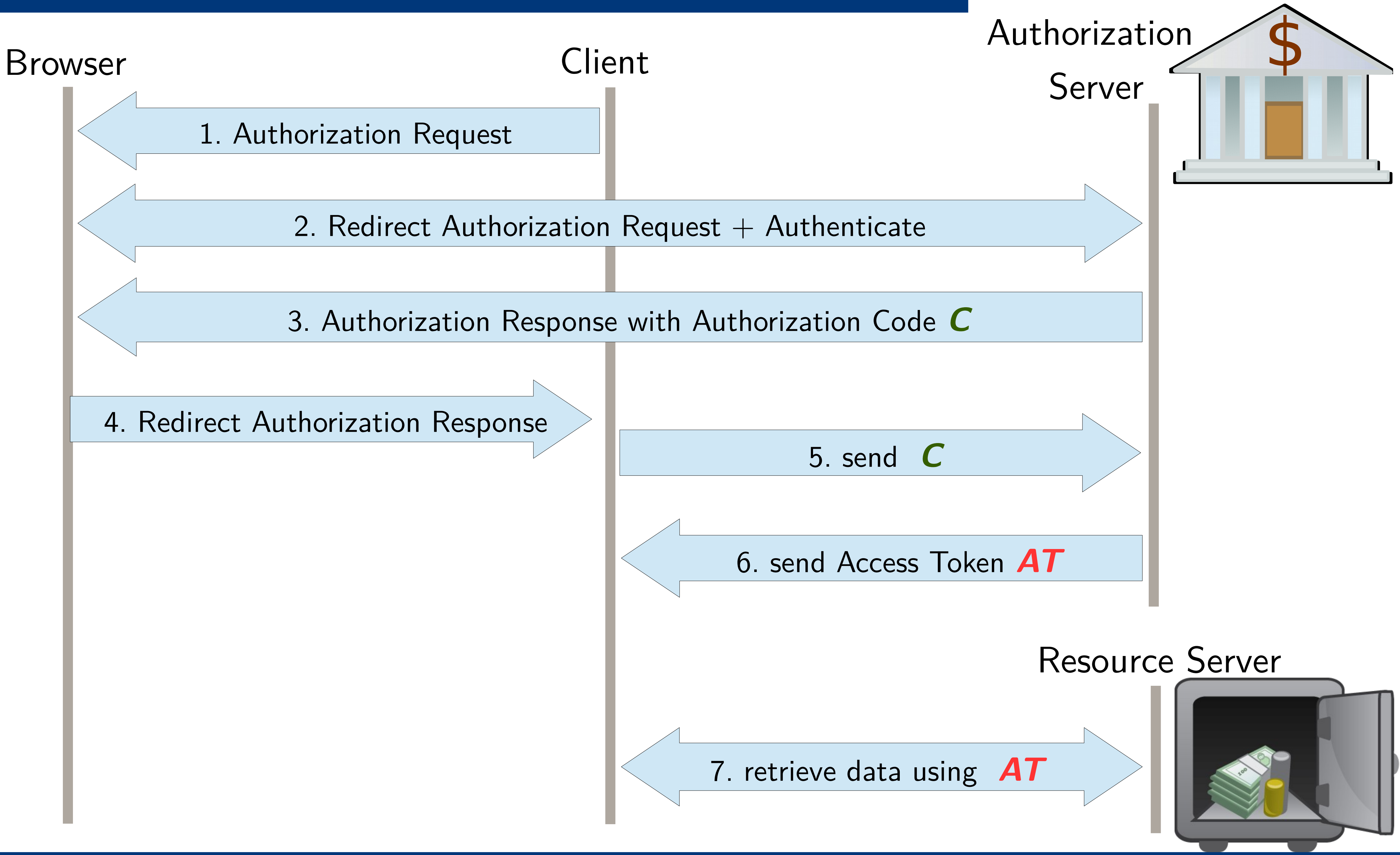
Outline

- Model-based Approach
- Financial-grade API
  - Key Mechanisms
  - Attacker Model
- Security Properties
- Attacks Found through the Analysis

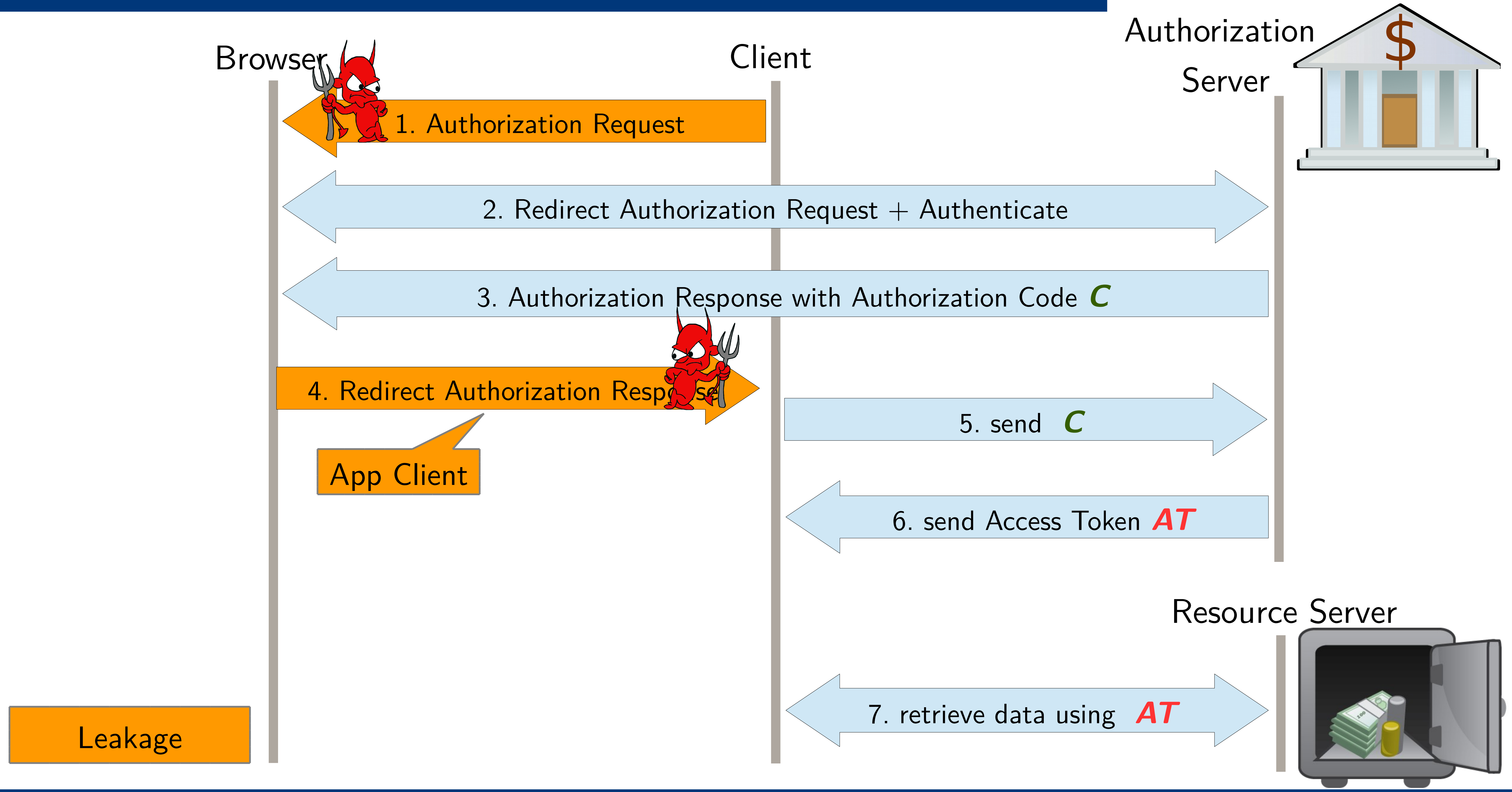
OpenID Financial-grade API (FAPI)

- Profile of the OAuth 2.0 Authorization Framework
- Utilizes mechanisms of OpenID Connect
- Different Profiles
  - Read-Only Profile
    - Authorization Code Flow
  - Read-Write Profile
    - OIDC Hybrid Flow
    - Authorization Code Flow with JARM

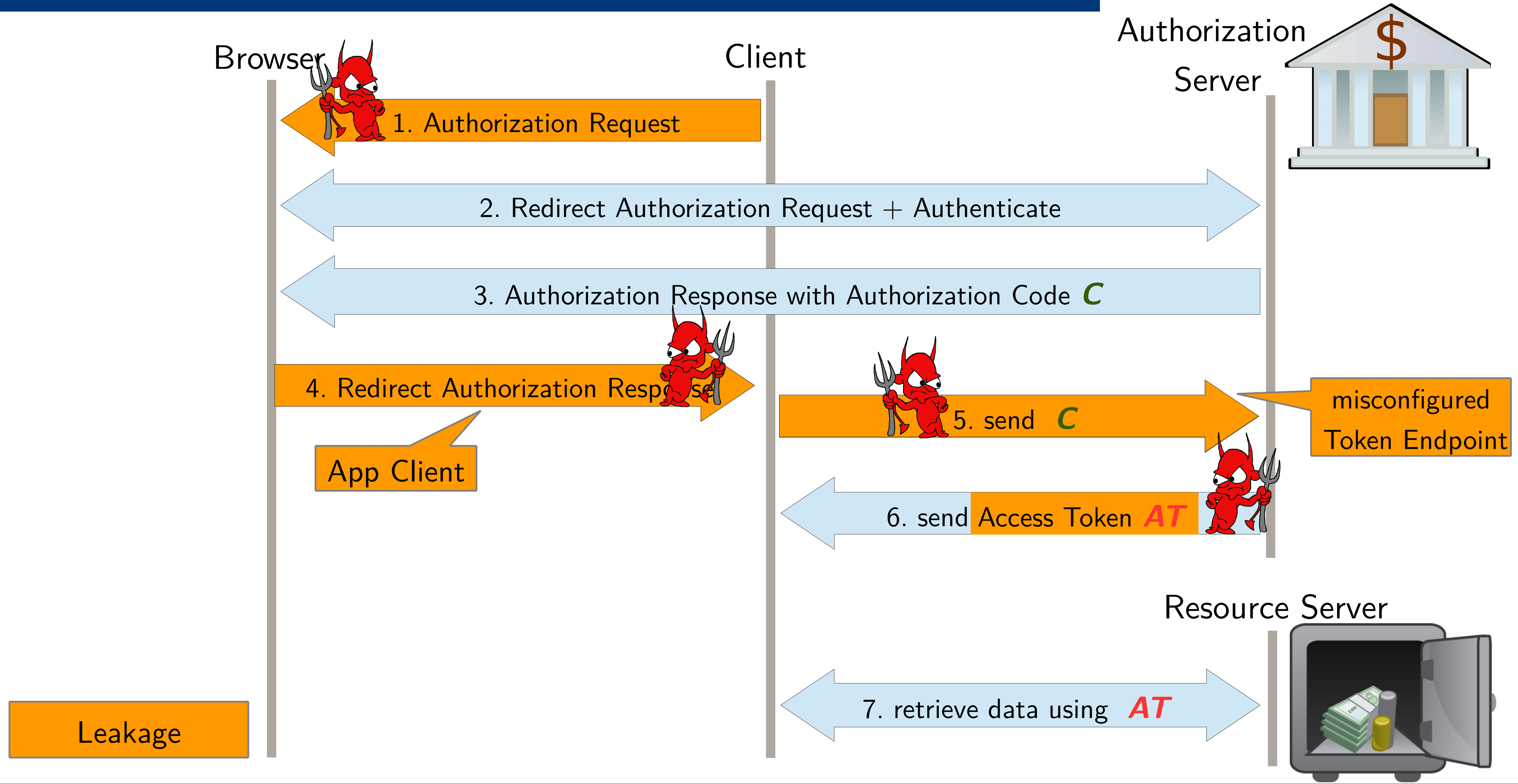
OAuth 2.0 Authorization Code Mode



Attacker Model (Read-Only Profile)



Attacker Model (Read-Write Profile)



FAPI: Key Mechanisms

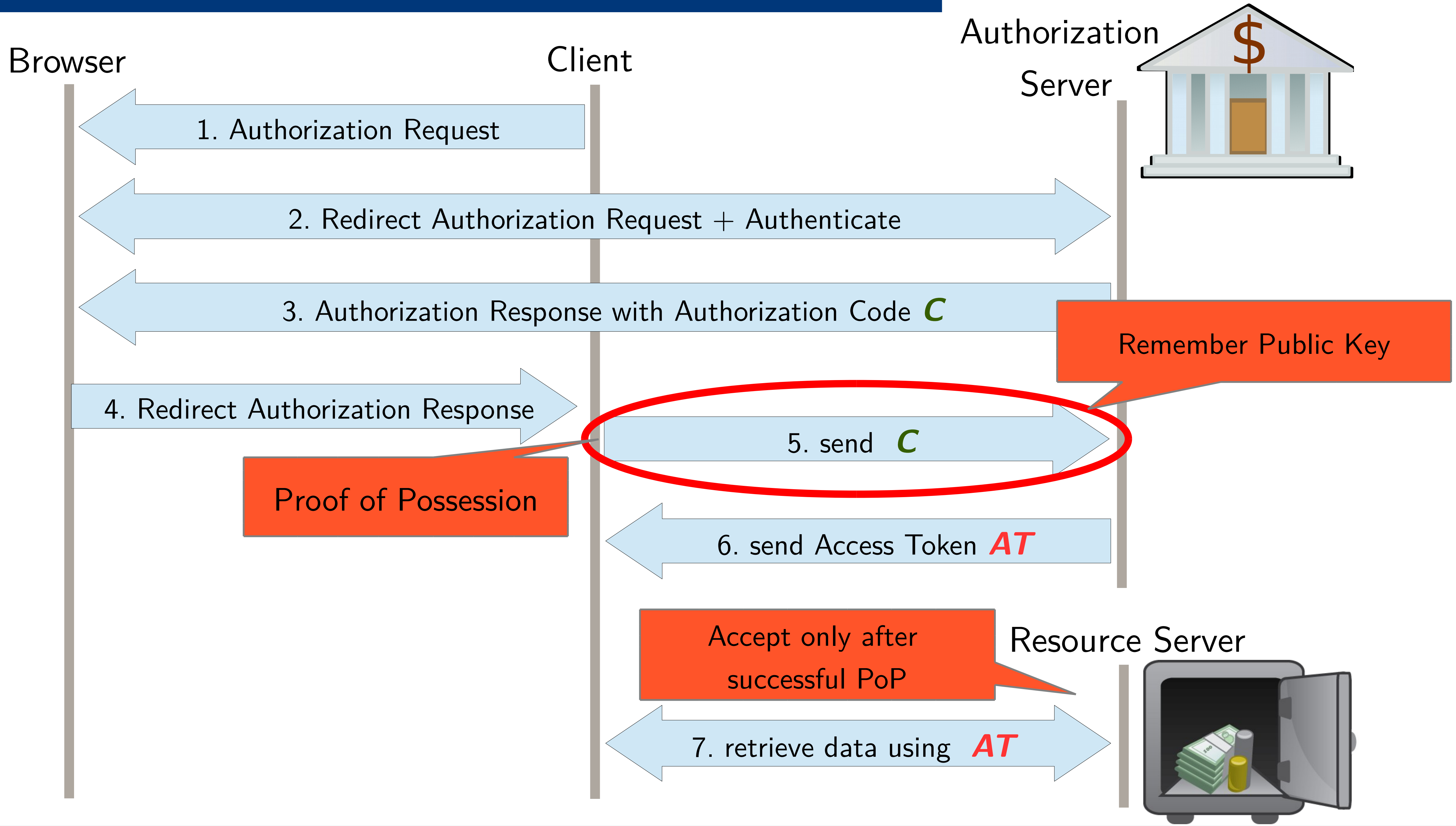
New Defense Mechanisms

- Token Binding
- Proof Key for Code Exchange (PKCE)
- Signed Authorization Response (JARM)
- Improved Client Authentication
- Signed Authorization Request

Token Binding

- Two Methods:
  - OAuth 2.0 Token Binding
  - Mutual TLS
- Goal: Bind Authorization Code and Access Token to Client

Binding Access Tokens: Idea



FAPI: Attacker Model

Attacker Model

As of 23-10-2018,
(including JARM)

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

Attacker Model: More Details

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

Part 1: Read-Only API Security Profile

5.2.2 Authorization server:

7. shall require RFC7636 with S256 as the code challenge method;

Attacker Model: More Details

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

```
+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
| End Device (e.g., Smartphone)    |
|                                  |
| +-------------+   +----------+   | (6) Access Token  +----------+
| |Legitimate   |   | Malicious|<--------------------|          |
| |OAuth 2.0 App|   | App      |-------------------->|          |
| +-------------+   +----------+   | (5) Authorization |          |
|        |    ^          ^         |        Grant      |          |
|        |     \         |         |                   |          |
|        |      \   (4)  |         |                   |          |
|    (1) |       \  Authz|         |                   |          |
|   Authz|        \ Code |         |                   |  Authz   |
| Request|         \     |         |                   |  Server  |
|        |          \    |         |                   |          |
|        |           \   |         |                   |          |
|        v            \  |         |                   |          |
| +----------------------------+   |                   |          |
| |                            |   | (3) Authz Code    |          |
| |     Operating System/      |<--------------------|          |
| |         Browser            |-------------------->|          |
| |                            |   | (2) Authz Request |          |
| +----------------------------+   |                   +----------+
|
```

Figure: https://tools.ietf.org/html/rfc7636

Attacker Model: More Details

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

Part 1: Read-Only API Security Profile

5.2.2 Authorization server:

7. shall require RFC7636 **with S256 as the code challenge method**;

Attacker Model: More Details

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

```
4b. A more sophisticated attack scenario
allows the attacker to observe requests
(in addition to responses) to the authorization
endpoint. [...]
This was caused by leaking http log information in
the OS. To mitigate this, "code_challenge_method"
value must be set either to "S256" or a value
defined by a cryptographically secure
"code_challenge_method" extension.
```

Attacker Model: More Details

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

8.3.2 Client credential and authorization code phishing at token endpoint

In this attack, the client developer is social engineered into believing that the token endpoint has changed to the URL that is controlled by the attacker.

As the result, the client sends the code and the client secret to the attacker, which will be replayed subsequently.

When the FAPI client uses MTLS or OAUTB, the authorization code is bound to the TLS channel, any phished client credentials and authorization codes submitted to the token endpoint cannot be used since the authorization code is bound to a particular TLS channel.

Attacker Model: More Details

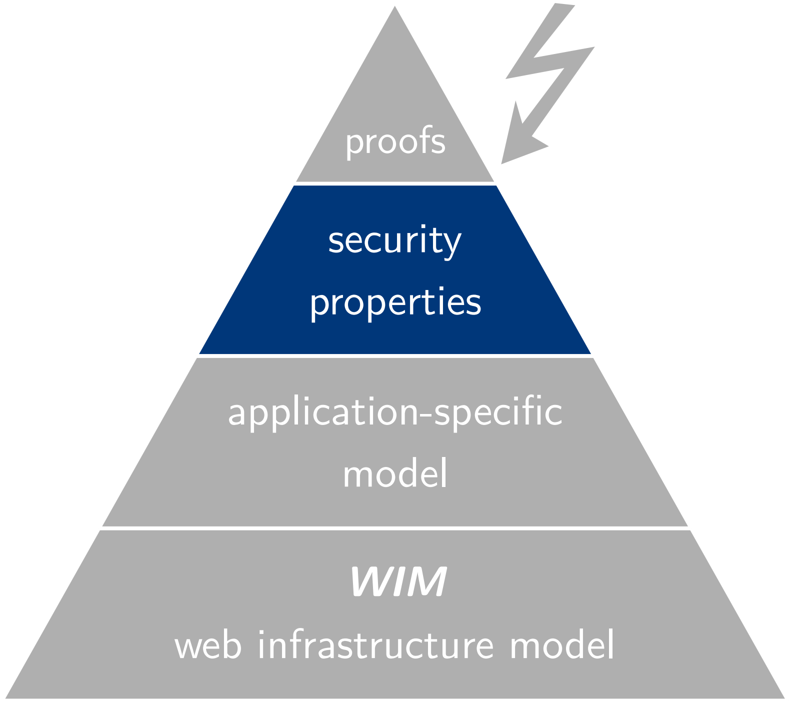
- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

8.3.5 Access token phishing

When the FAPI client uses MTLS or OAUTB, the access token is bound to the TLS channel, it is access token phishing resistant as the phished access tokens cannot be used.

Security Definitions

FAPI: Security Definitions



- Authentication

  Attacker cannot log in at client with honest identity

- Authorization

  Attacker cannot access resources of honest identity

- Session Integrity

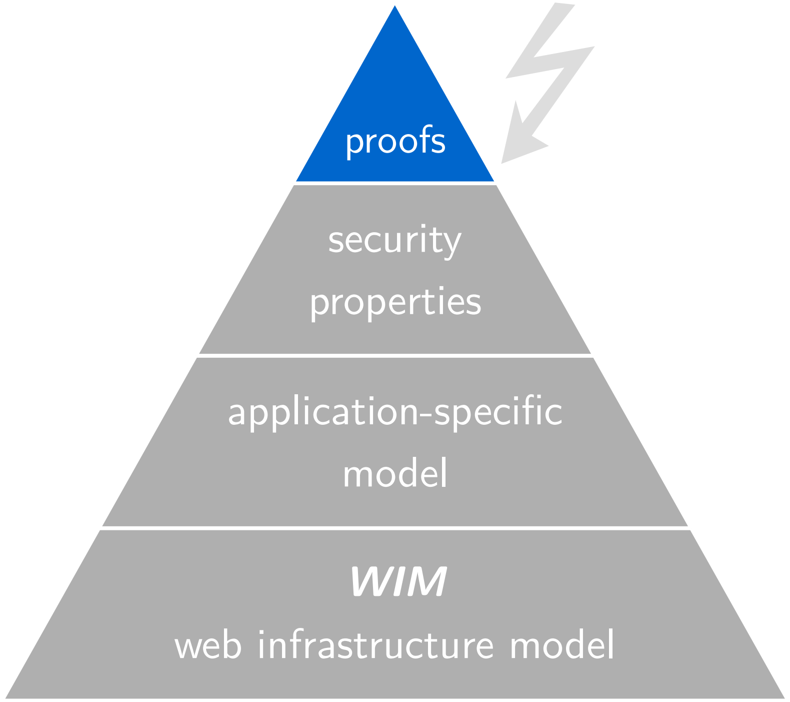
  Honest user is logged in under their own account and using their own resources

Authorization-Property

Definition 17 (Authorization Property). We say that the FAPI web system with a network attacker $\mathcal{FAPI}^n$ *is secure w.r.t. authorization* iff for every run ρ of $\mathcal{FAPI}^n$, every configuration (S,E,N) in ρ, every authorization server $as \in \mathsf{AS}$ that is honest in S with $s_0^{as}.\mathtt{resource\_servers}$ being domains of honest resource servers, every identity $id \in \mathsf{ID}^{as}$ with $b = \mathsf{ownerOfID}(id)$ being an honest browser in S, every client $c \in \mathsf{C}$ that is honest in S with client id *clientId* issued to c by as, every resource server $rs \in \mathsf{RS}$ that is honest in S such that $id \in s_0^{rs}.\mathtt{ids}$, $s_0^{rs}.\mathtt{authServ} \in \mathsf{dom}(as)$ and with $dom_{rs} \in s_0^{as}.\mathtt{resource\_servers}$ (with $dom_{rs} \in \mathsf{dom}(rs)$), every access token t associated with c, as and id and every resource access nonce $r \in s_0^{rs}.\mathtt{rNonce}[id] \cup s_0^{rs}.\mathtt{wNonce}[id]$ it holds true that:

If r is contained in a response to a request m sent to rs with $t \equiv m.header[\mathtt{Authorization}]$, then r is not derivable from the attackers knowledge in S (i.e., $r \not\in d_{\emptyset}(S(\mathsf{attacker}))$).

FAPI: Security Definitions



- Authentication

  Attacker cannot log in at client with honest identity

- Authorization

  Attacker cannot access resources of honest identity

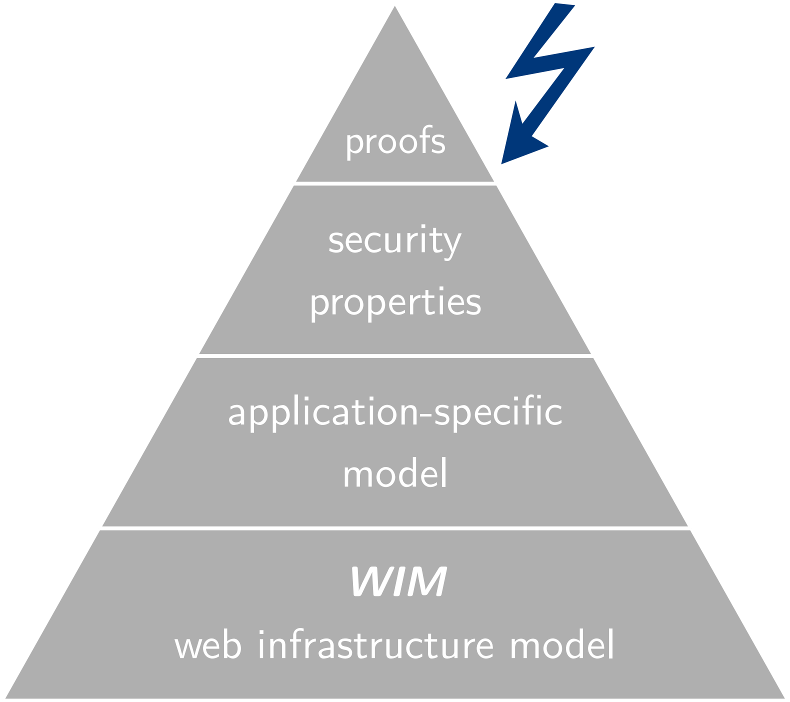
- Session Integrity

  Honest user is logged in under their own account and using their own resources

Attacks

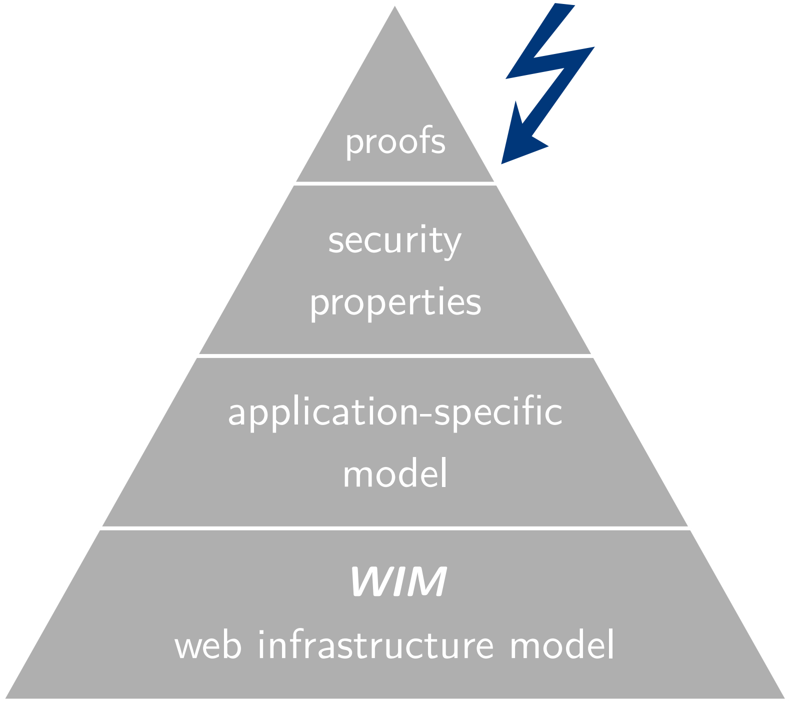
Attacks Found Through Our Formal Analysis

- Cuckoo's Token Attack
- Access Token Injection
- PKCE Chosen Challenge Attack
- Authorization Request Leak Attacks

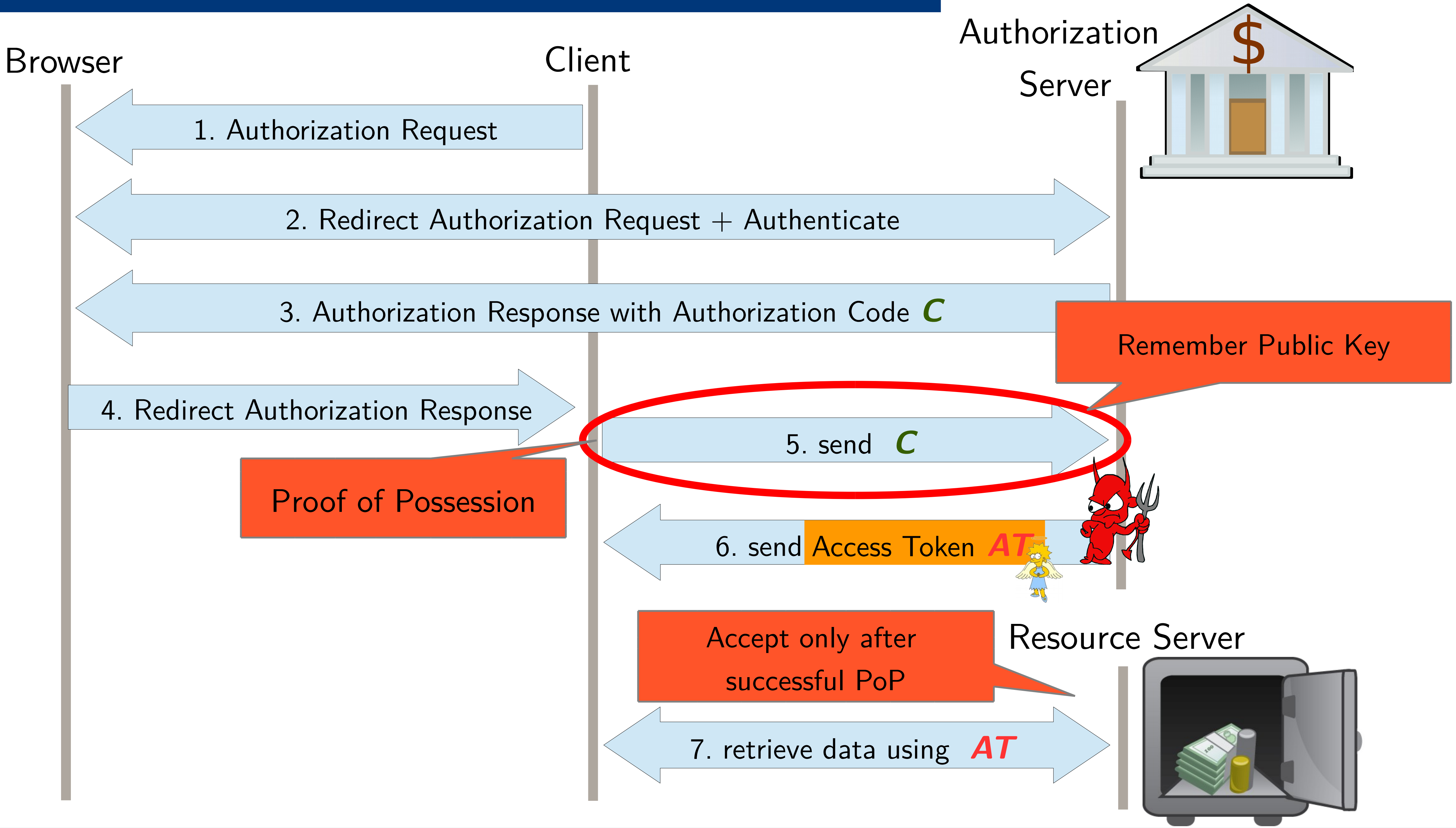


Attacks Found Through Our Formal Analysis

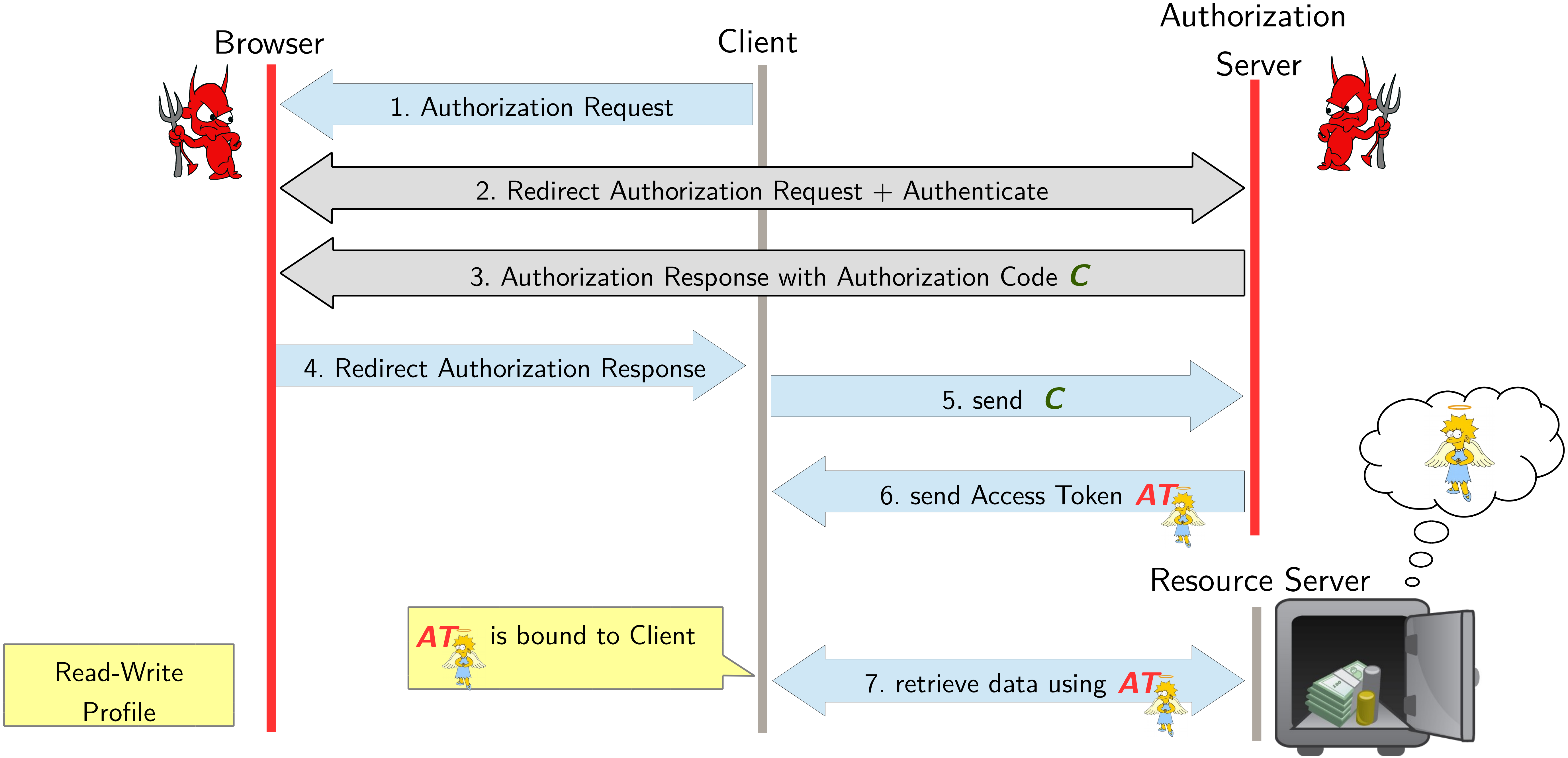
- Cuckoo's Token Attack
- Access Token Injection
- PKCE Chosen Challenge Attack
- Authorization Request Leak Attacks



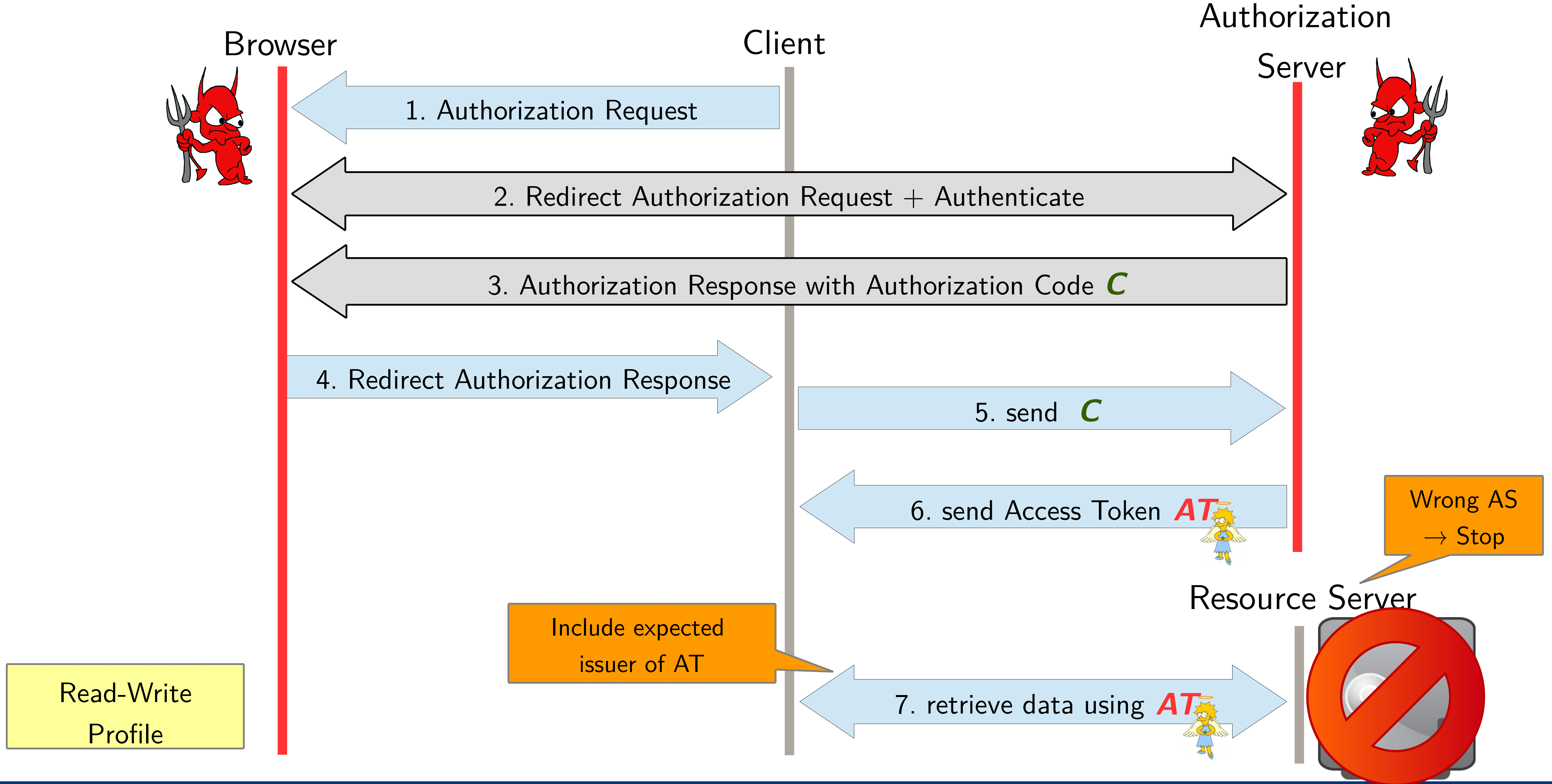
Recap: Binding Access Tokens



Cuckoo's Token Attack

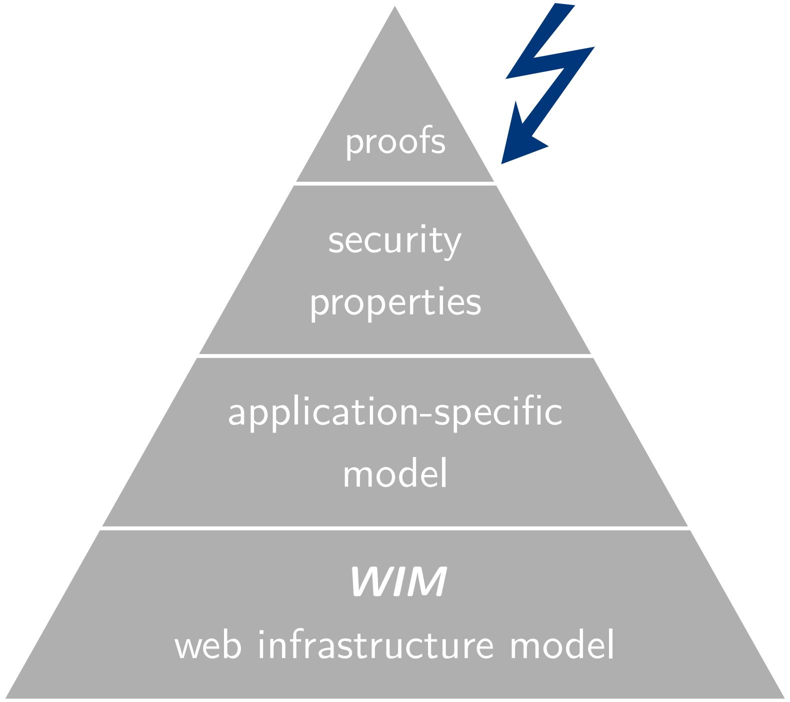


Mitigation


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Wrong AS → Stop
Resource Server
Include expected issuer of AT
7. retrieve data using AT
Read-Write Profile

Attacks Found Through Our Formal Analysis

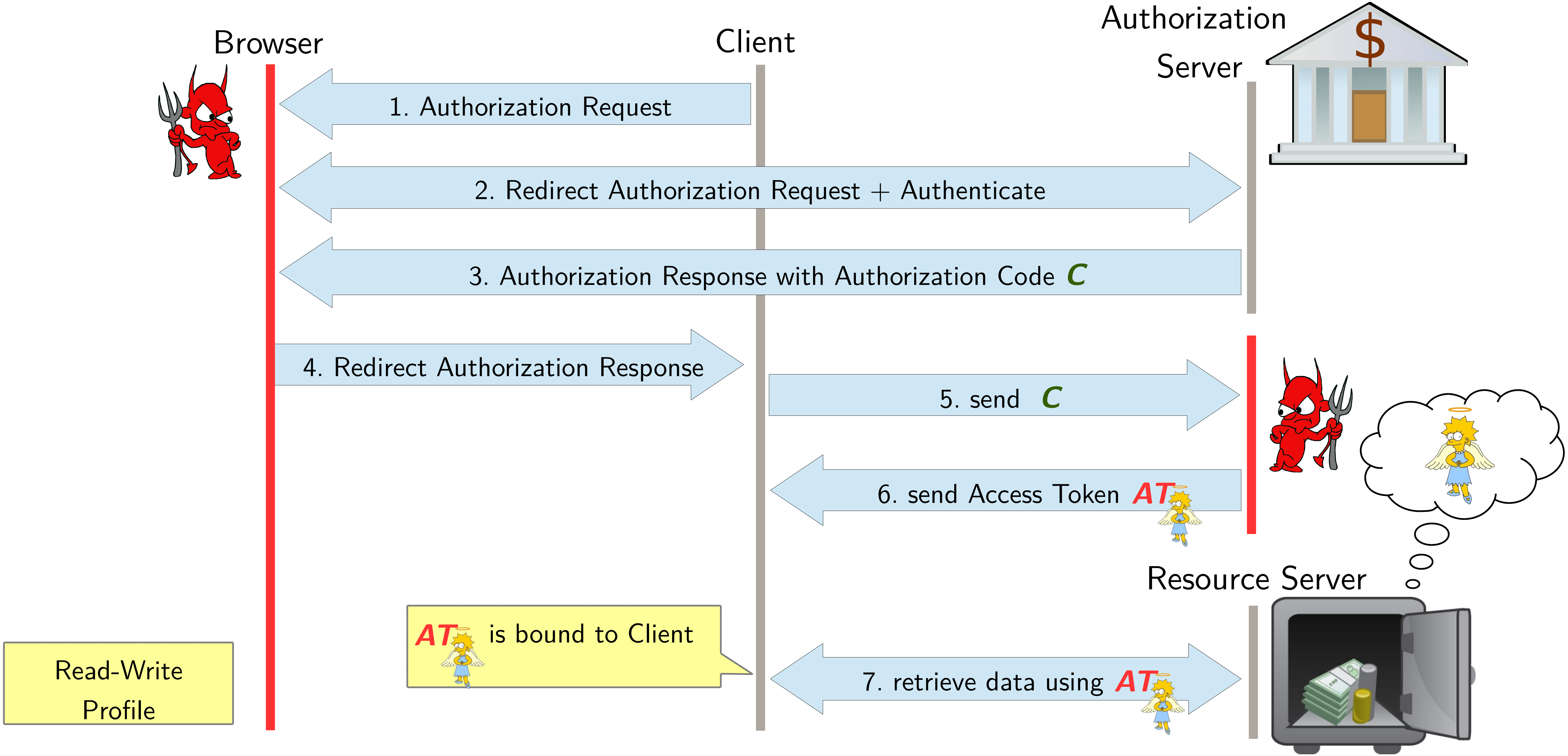
- Cuckoo's Token Attack
- Access Token Injection
- PKCE Chosen Challenge Attack
- Authorization Request Leak Attacks



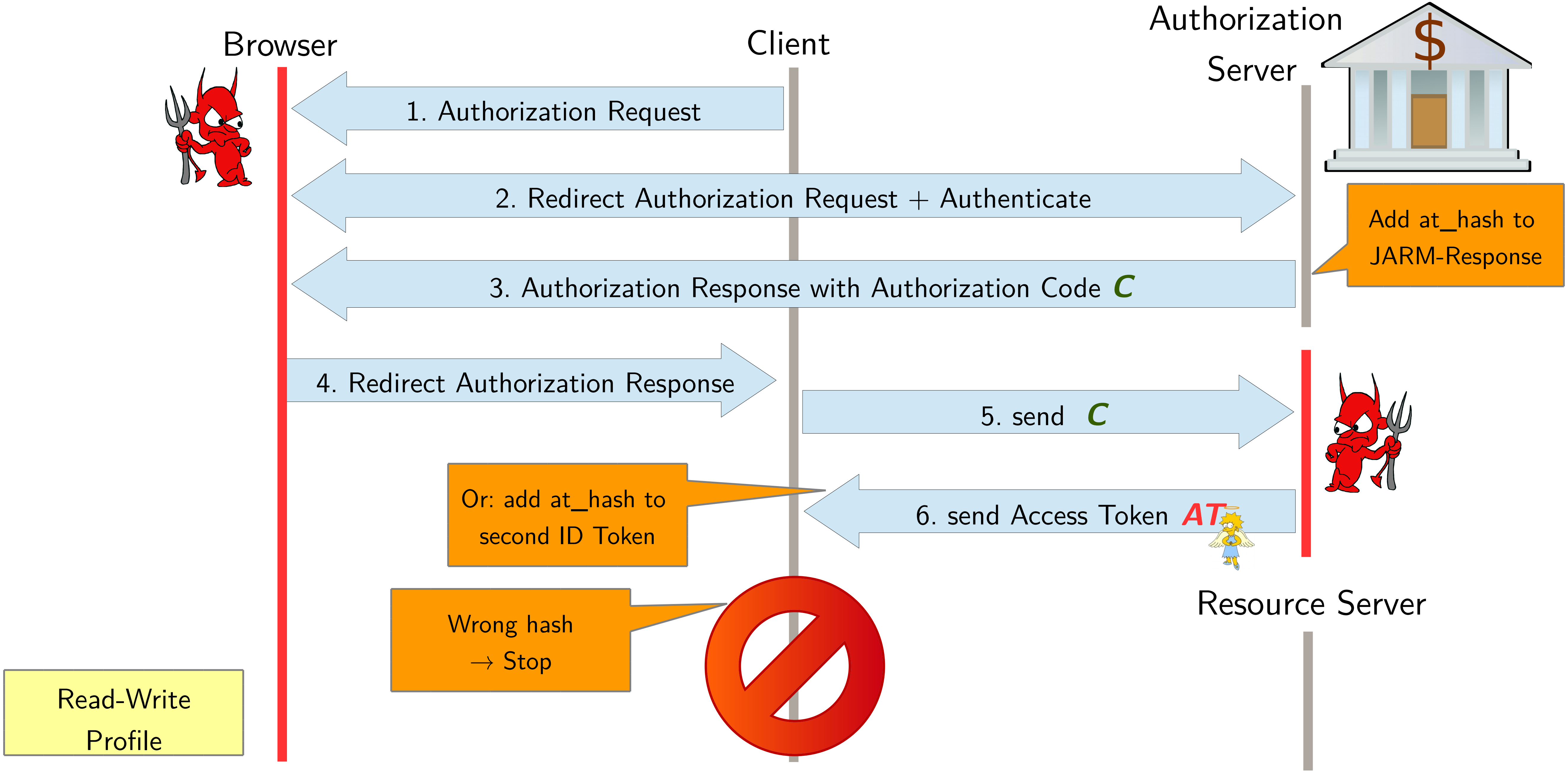
Recap: Attacker Model

- Read-Only Profile:
  - Authorization Response leaks
  - Authorization Request leaks
- Read-Write Profile
  - Token Endpoint controlled by Attacker
  - Access Tokens leaks

Access Token Injection



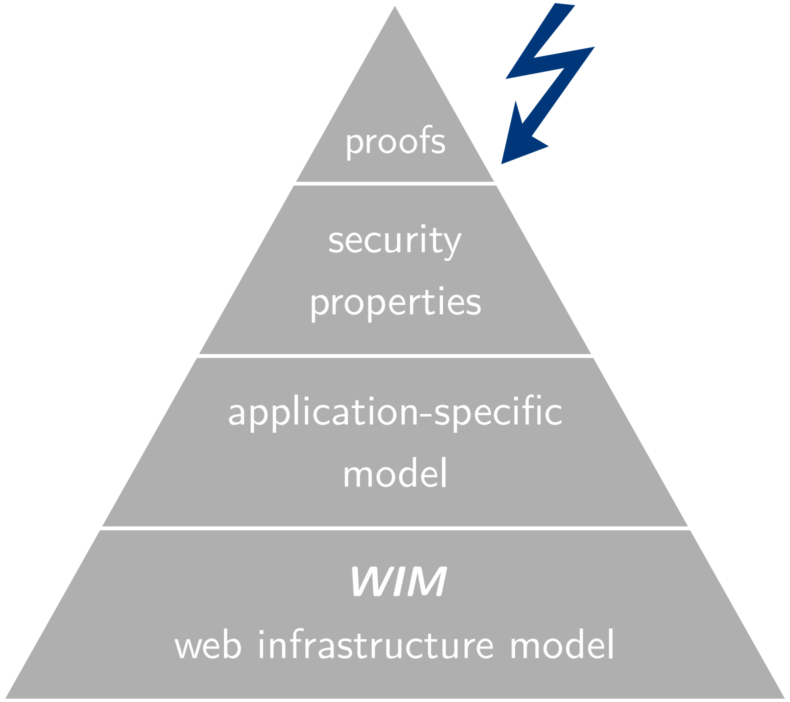
Mitigation



Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
Add at_hash to JARM-Response
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
Or: add at_hash to second ID Token
6. send Access Token *AT*
Wrong hash → Stop
Resource Server
Read-Write Profile

Attacks Found Through Our Formal Analysis

- Cuckoo's Token Attack
- Access Token Injection
- PKCE Chosen Challenge Attack
- Authorization Request Leak Attacks



Fixes and Security Proof

- Fixes proposed for all attacks
- Proved security
  - Authentication ✓
    - Attacker cannot log in at client with honest identity
  - Authorization ✓
    - Attacker cannot access resources of honest identity
  - Session Integrity ✓
    - Honest user is logged in under their own account and using their own resources

Only for Webserver
Clients using OAUTB

Conclusion

- First formal security analysis of the OpenID Financial-grade API
- Found several attack scenarios
- Suggested fixes
- Proved security under strong attacker model

Thanks!