University of Stuttgart
Institute of
Information Security

Web Security Model and Applications

Prof. Dr. Ralf Küsters

SummerSOC 2018

In this Tutorial

- Motivation: formal security analysis of web applications and standards
- Our Model of the Web Infrastructure
- Single Sign-On Case Studies
- Formal Security Analysis of OAuth 2.0
  - Introduction to OAuth 2.0
  - Attacks on OAuth 2.0

Motivation

Formal Security Analysis of Web Applications and Standards

The web is complex ...

- Interaction of different components
- Large number of complex standards developed at a high pace by many separate organizations

Some examples:

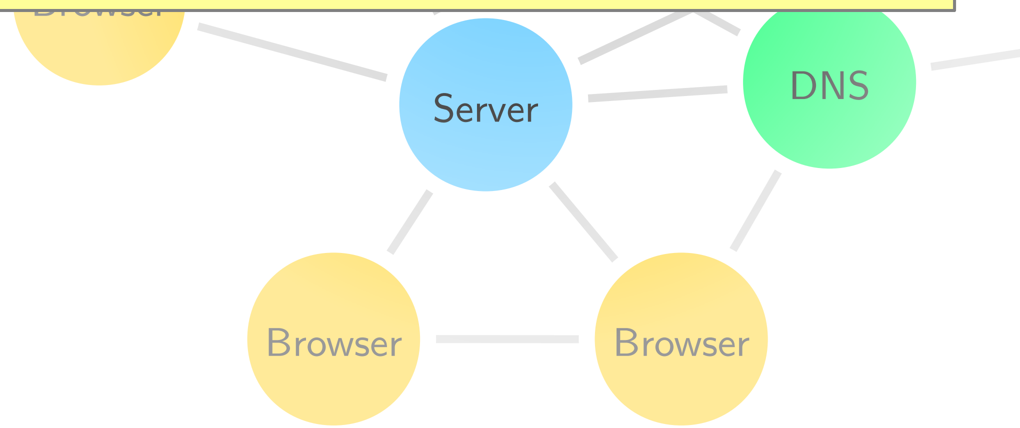
IETF:
- HTTP/1.1 and HTTP/2
- RFC 6265
- RFC 6797
- RFC 6454

W3C:
- HTML5
- Web Storage
- Cross-Origin Resource Sharing

WHATWG:
- Fetch

... and web applications as well ...

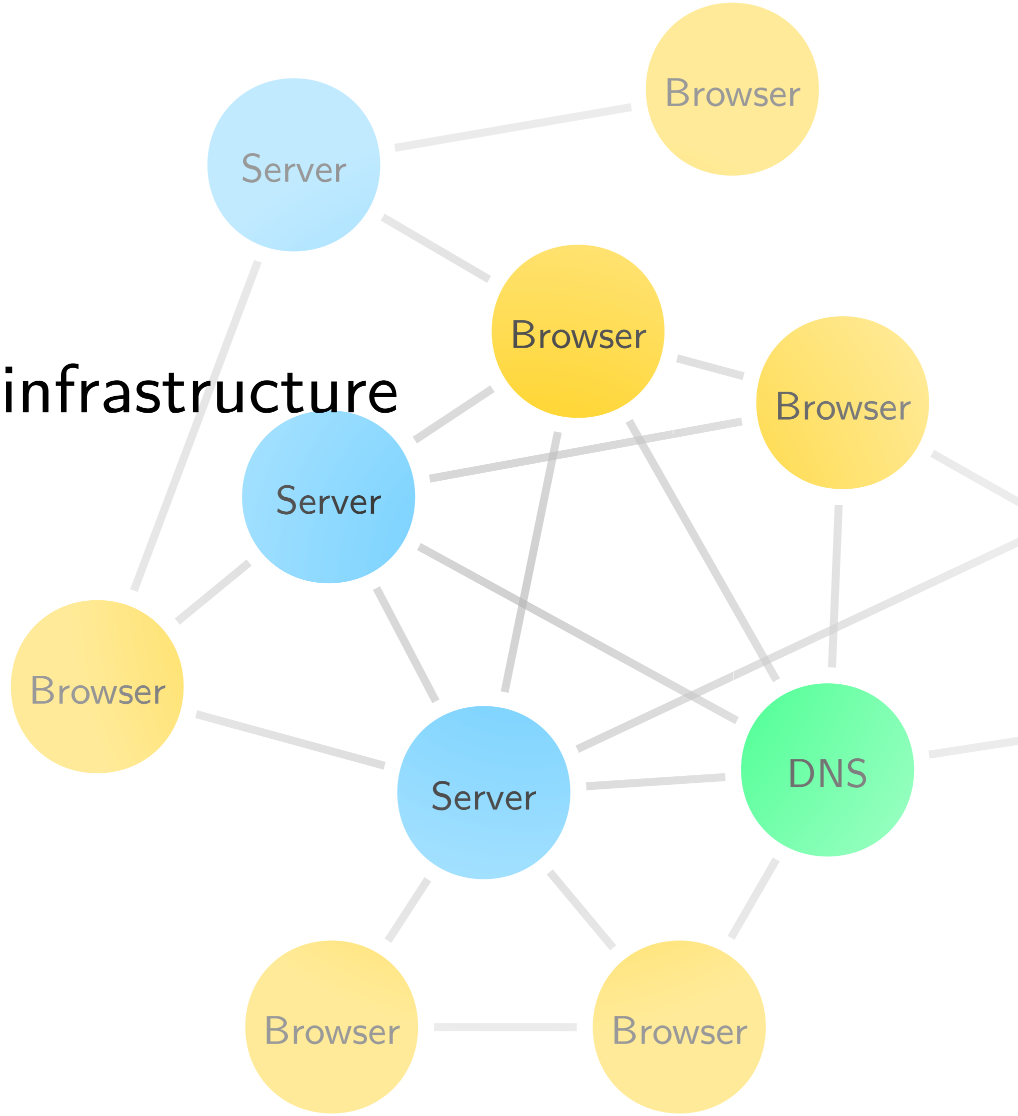
- Increasing complexity of web applications
- Many vulnerabilities



Formal Security Analysis of Web Applications and Standards

Formal security analysis:

- Based on a comprehensive model of the web infrastructure
- Precisely specify security properties
- Carry out security proofs



The Story So Far

Previous work:

- Akhawe, Barth, Lam, Mitchell, Song (2010): Alloy Model
- Bansal, Bhargavan, Delignat-Lavaud, Maffeis (2012, 2013): WebSpi model in ProVerif

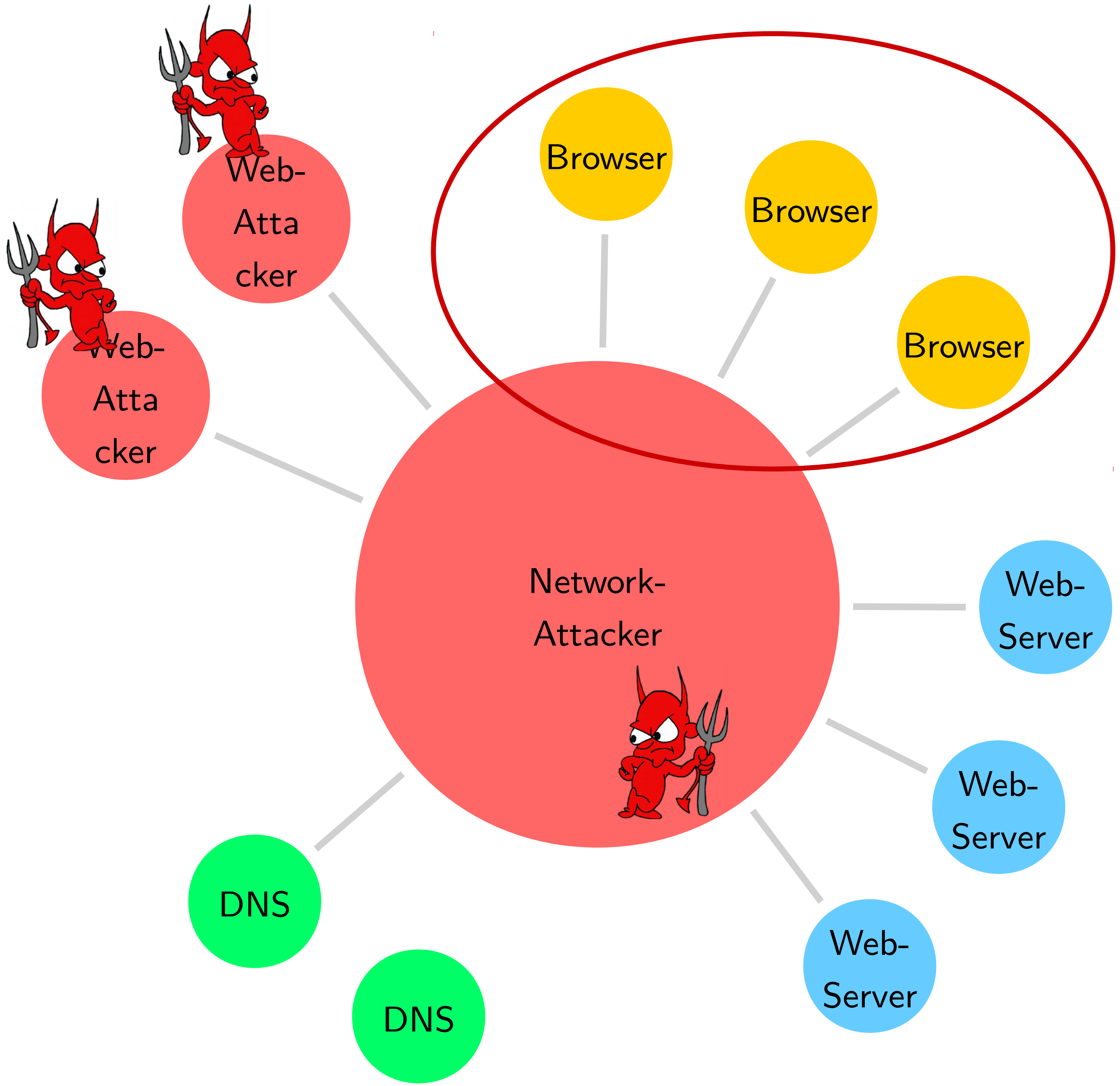
Our approach:

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016, CSF 2017]

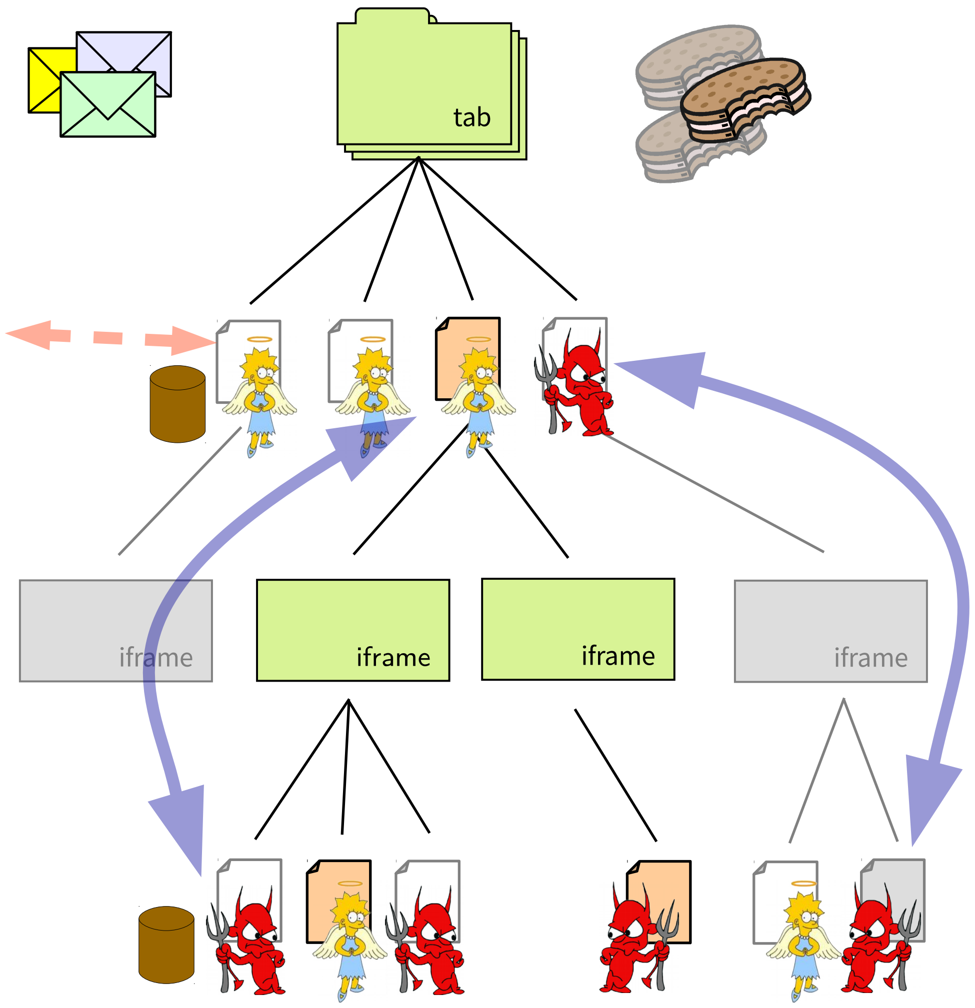
- Very close to the standards
- More comprehensive
- Manual model (so far)

Our Model of the Web Infrastructure

Network Model



Web Browser Model



Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts
- web storage & cookies
- web messaging & XHR
- message headers
- redirections
- security policies
- dynamic corruption
- ...

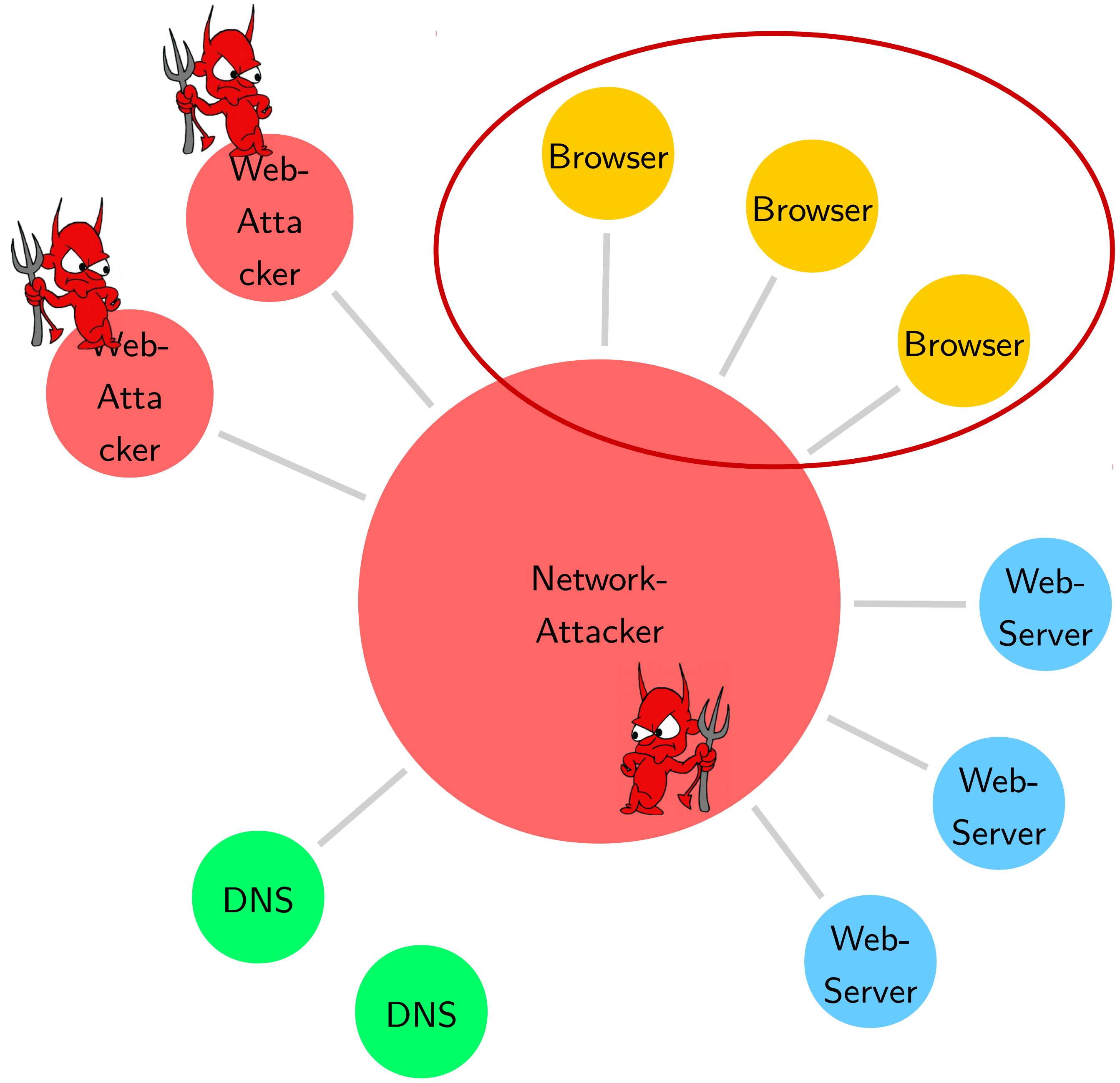
Origin: https://example.com

Browser Model - Example

Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE(response, reference, request, requestUrl, key, f, s′)
    if Set-Cookie ∈ response.headers then
        for each c ∈⟨⟩ response.headers[Set-Cookie], c ∈ Cookies do
            let s′.cookies[request.host]
              ↪ := AddCookie(s′.cookies[request.host], c)
    if Strict-Transport-Security ∈ response.headers ∧ requestUrl.protocol ≡ S then
        let s′.sts := s′.sts +⟨⟩ request.host
    if Referer ∈ request.headers then
        let referrer := request.headers[Referer]
    else
        let referrer := ⊥
    if Location ∈ response.headers ∧ response.status ∈ {303, 307} then
        let url := response.headers[Location]
        if url.fragment ≡ ⊥ then
            let url.fragment := requestUrl.fragment
        let method′ := request.method
        let body′ := request.body
        if Origin ∈ request.headers then
            let origin := ⟨request.headers[Origin], ⟨request.host, url.protocol⟩⟩
        else
            let origin := ⊥
        if response.status ≡ 303 ∧ request.method ∉ {GET, HEAD} then
            let method′ := GET
            let body′ := ⟨⟩
```

Network Model



Limitations

- No language details
- No user interface details
- No byte-level attacks (e.g, buffer overflows)
- Abstract view on cryptography and TLS

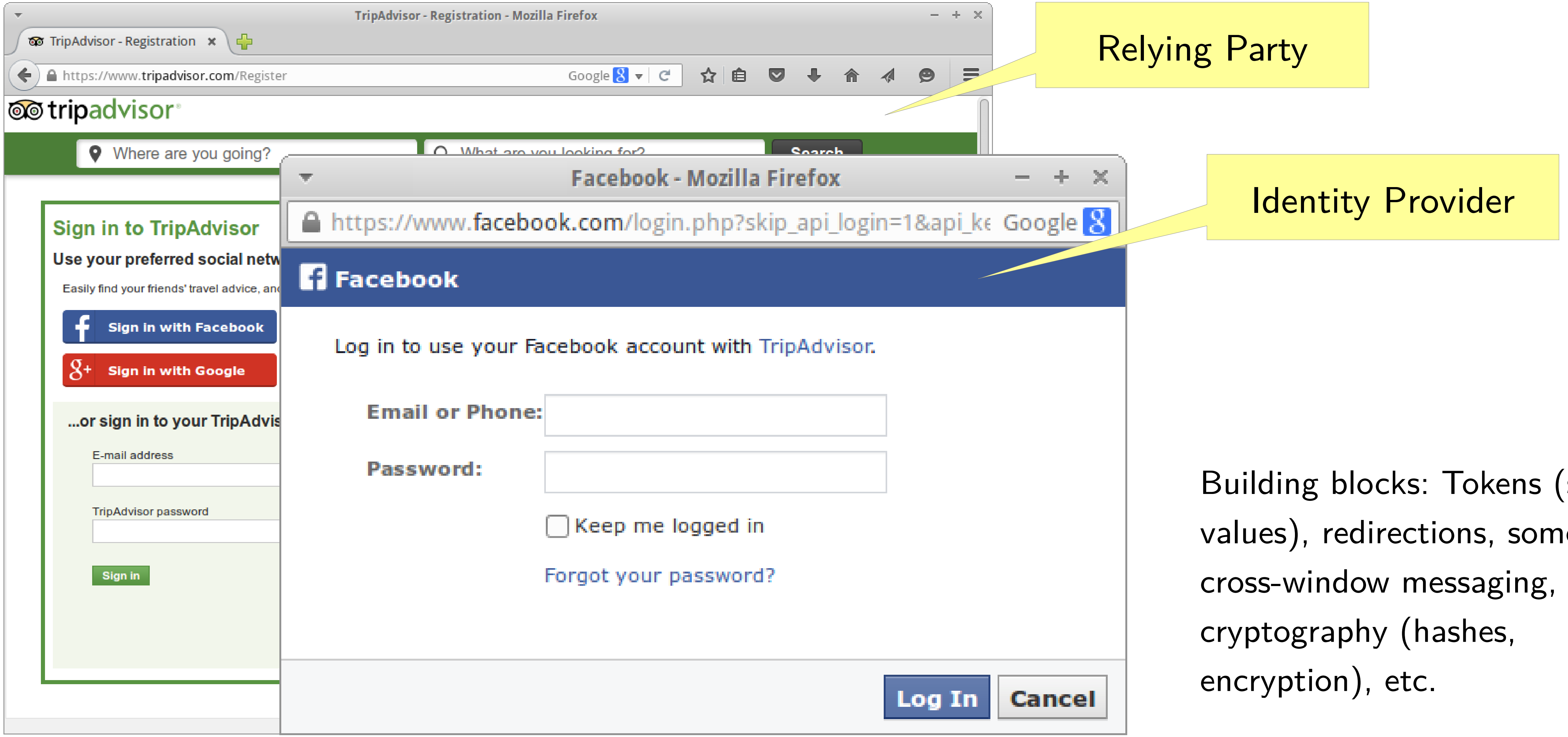
Our Model of the Web Infrastructure

- Detailed formal model
- Comprehensive and precise
- Summarizes and condenses relevant standards
- Solid basis for analysis
- Reference model

  for tool-based analysis, developers, researchers, teaching

Single Sign-On Case Studies

Single Sign-On (SSO)



Building blocks: Tokens (secret values), redirections, sometimes: cross-window messaging, cryptography (hashes, encryption), etc.

Previous Work

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016, CSF 2017]

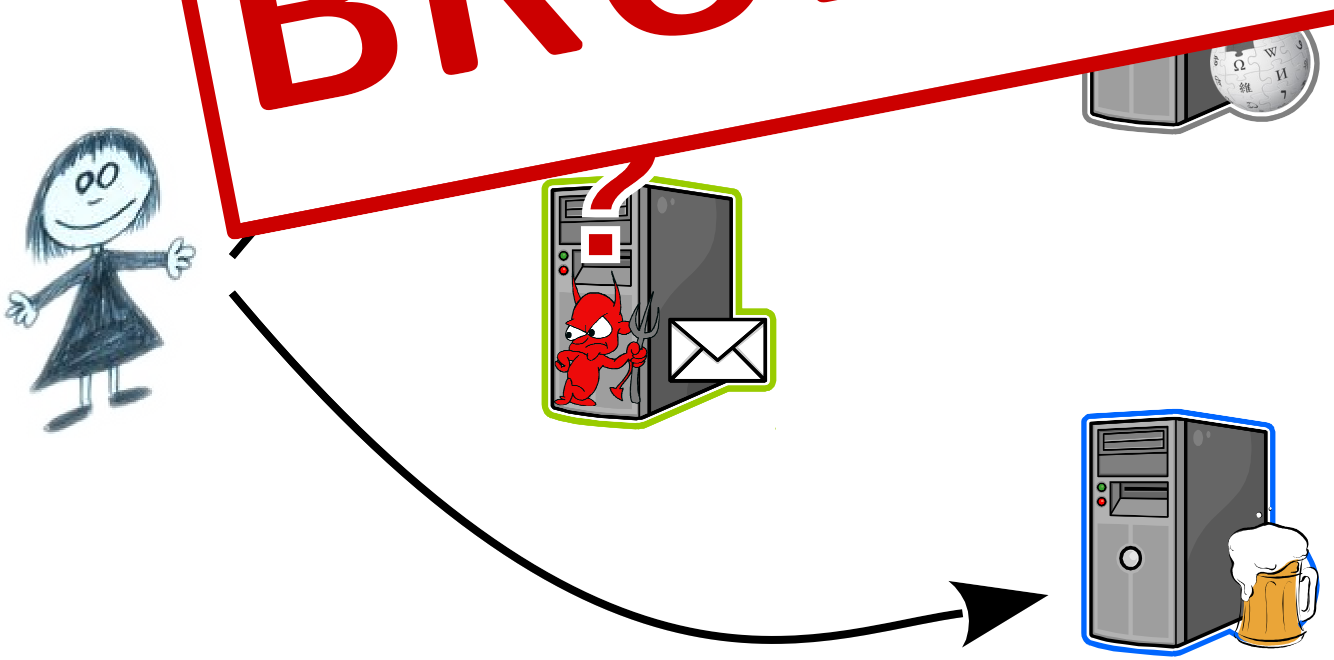
- Formal analysis of Mozilla's BrowserID
  Main design goal: privacy
  - Found severe attacks: Identity Injection Attack, PostMessage-Based Attack
  - Proposed fixes for authentication and proved security
  - Privacy broken beyond repair

BrowserID: Privacy

Unique claim:

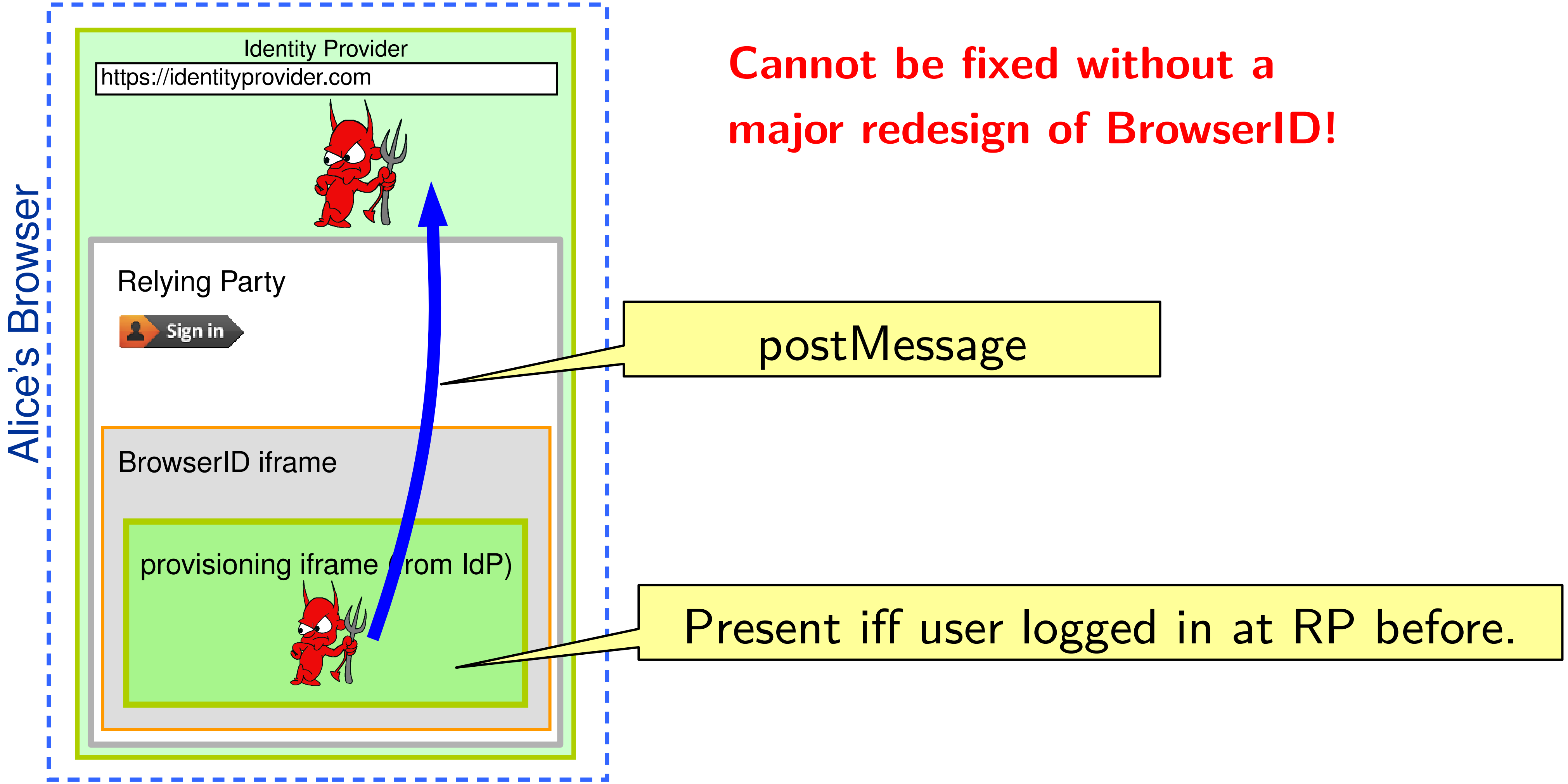
“… the BrowserID protocol **never leaks** trac… …**dentity Provider**.” - Mozilla Persona FAQ

BROKEN



BrowserID: Privacy Attack

Information is leaked by the **window structure** in the user's browser:



Cannot be fixed without a major redesign of BrowserID!

Previous Work

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016, CSF 2017]

- Formal analysis of Mozilla's BrowserID
  Main design goal: privacy
  - Found severe attacks: Identity Injection Attack, PostMessage-Based Attack
  - Proposed fixes for authentication and proved security
  - Privacy broken beyond repair
- Designed our own new SSO system: SPRESSO (https://spresso.me)
  Provably provides strong authentication and privacy properties.

  First SSO system that provides privacy!

Previous Work

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016, CSF 2017]

- Analysis of OAuth 2.0
  - Found attacks: 307 Redirect Attack, IdP Mix-Up Attack, State Leak Attack, Naive RP Session Integrity Attack
  - Proposed fixes and proved security
  - Working in the IETF to codify fixes into a new RFC
- OpenID Connect 1.0 with Discovery and Dynamic Registration Extensions
  - Developed formal model of the standard
  - Proposed security guidelines mitigating known attacks
  - Proved security for (fixed) standard

Formal Analysis of OAuth 2.0

- **Introduction to OAuth 2.0**
- Attacks on OAuth 2.0

Single-Sign On Systems

OAuth 1.0
(2006)

OAuth 2.0
(2011)

Log in with Facebook

- Incompatible to OAuth $<$ 2.0
- No cryptography involved (except for TLS)
- Widely used
- Only defined for authorization
- Nonetheless used for authorization **and authentication**
- Many flaws (mostly fixed)

Here:
Attacks and security proofs for OAuth 2.0

Attacks and mitigations also applicable to OpenID Connect

OpenID
(2007)

OpenID Connect
(2014)

- Incompatible to OpenID
- Authentication layer on top of OAuth 2.0
- IdP discovery and dynamic RP registration

Others: SAML, Shibboleth, WebAuth, CAS, ...

OAuth 2.0

OAuth 2.0: RFC 6749 (and others)

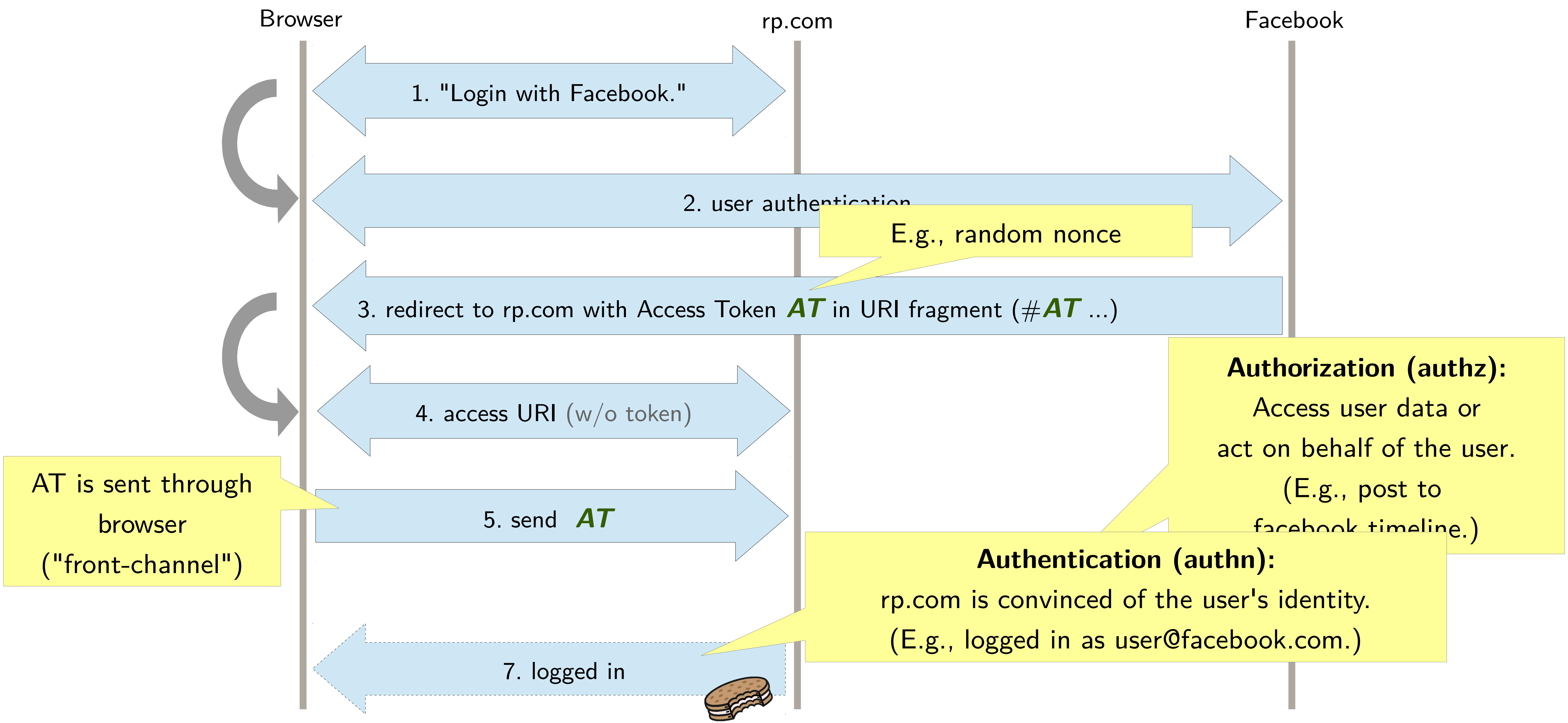
Four modes of Interaction:

- Implicit Mode
- Authorization Code Mode
- Resource Owner Password Credentials Mode
- Client Credentials Mode

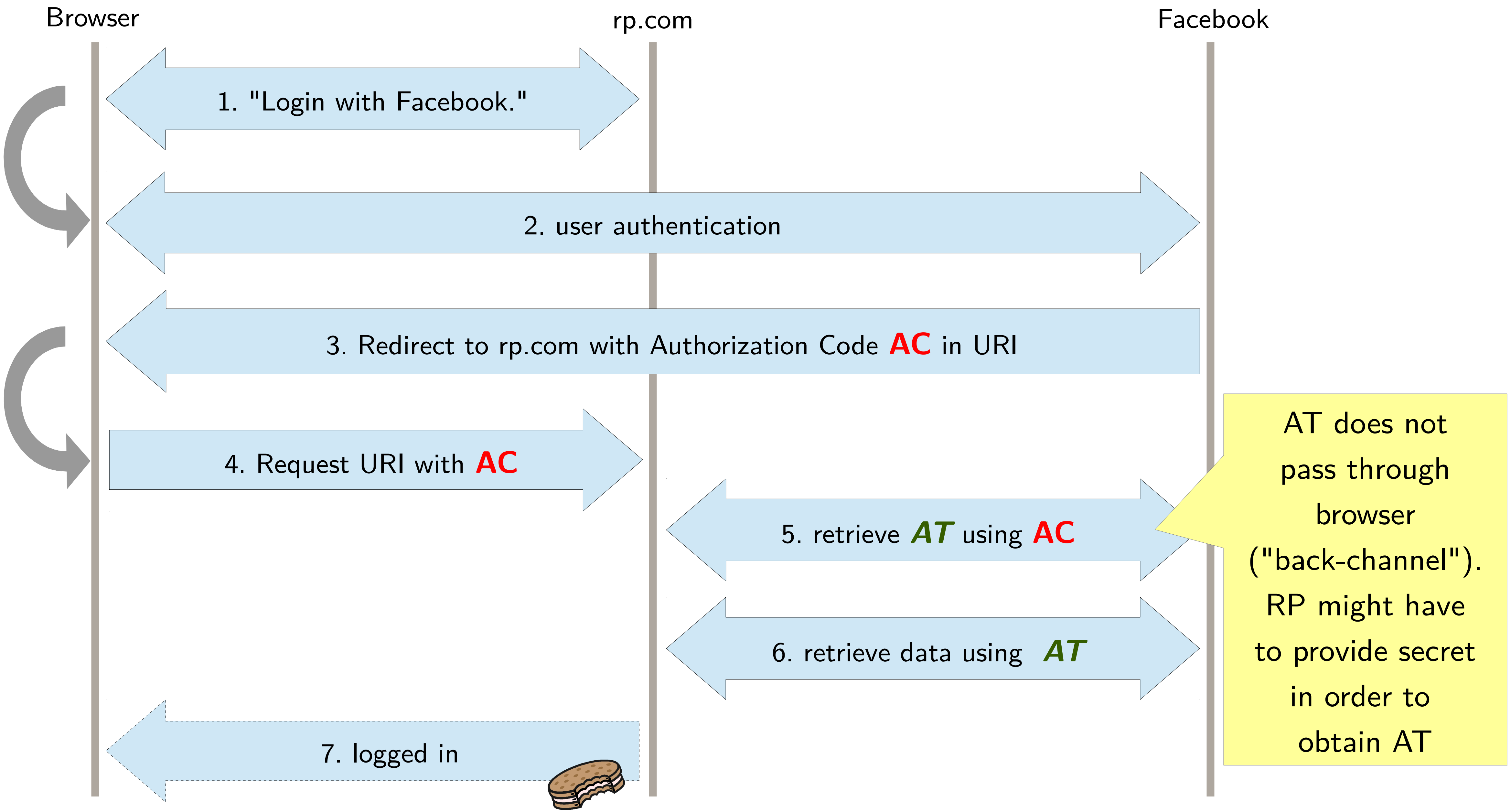
most common

... and many other options.

Implicit Mode



Authorization Code Mode



Desired Properties of OAuth 2.0

- Authentication
- Authorization
- Session Integrity
- ~~Privacy~~

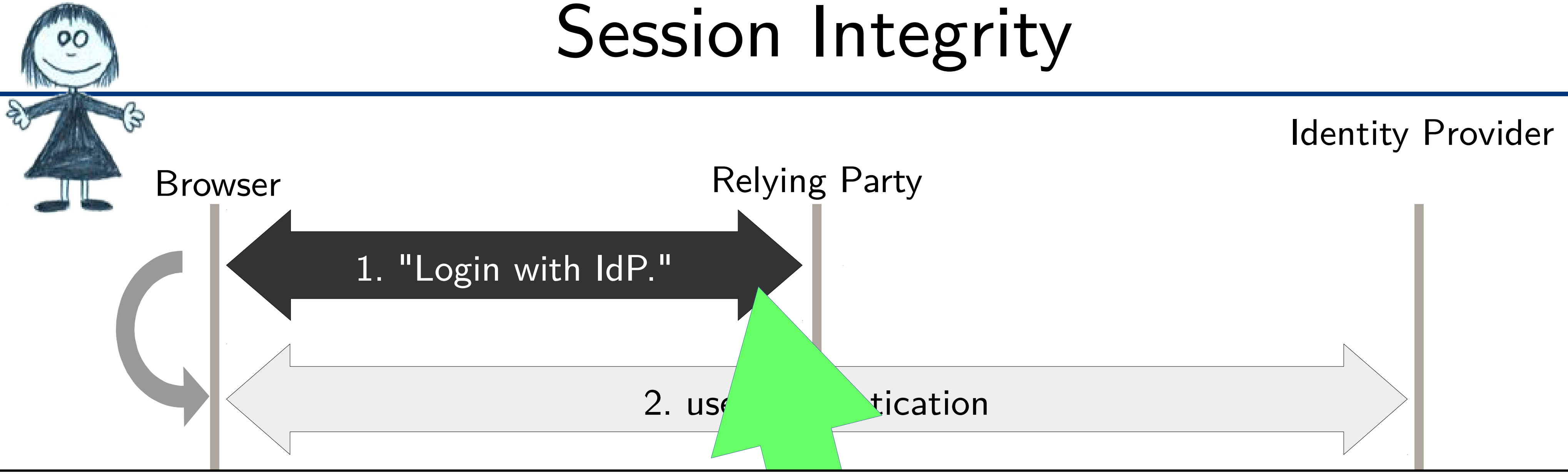
Authentication Property

An attacker (having full control over the network) should not be able to use a service of a relying party as an honest user.

Definition 58 (Authentication Property). Let $\mathcal{OAuthWS}^n$ be an OAuth web system with a network attacker. We say that $\mathcal{OAuthWS}^n$ *is secure w.r.t. authentication* iff for every run ρ of $\mathcal{OAuthWS}^n$, every state (S^j, E^j, N^j) in ρ, every $r \in \mathsf{RP}$ that is honest in S^j, every $i \in \mathsf{IDP}$, every $g \in \mathsf{dom}(i)$, every $u \in \mathbb{S}$, every RP service token of the form $\langle n, \langle u, g \rangle\rangle$ recorded in $S^j(r).\mathtt{serviceTokens}$, and n being derivable from the attackers knowledge in S^j (i.e., $n \in d_\emptyset(S^j(\mathsf{attacker}))$), then the browser b owning u is fully corrupted in S^j (i.e., the value of $isCorrupted$ is $\mathtt{FULLCORRUPT}$), some $r' \in \mathsf{trustedRPs}(\mathsf{secretOfID}(\langle u, g \rangle))$ is corrupted in S^j, or i is corrupted in S^j.

Analogously for authorization.

Session Integrity



Definition 66 (Session Integrity for Authentication). Let $\mathcal{OAuthWS}^w$ be an OAuth web system with web attackers. We say that $\mathcal{OAuthWS}^w$ *is secure w.r.t. session integrity for authentication* iff for every run ρ of $\mathcal{OAuthWS}^w$, every processing step Q_{login} in ρ, every browser b that is honest in Q_{login}, every $r \in \mathsf{RP}$ that is honest in Q_{login}, every $i \in \mathsf{IDP}$, every identity $\langle u, g\rangle$, the following holds true: If in Q_{login} a service token of the form $\langle n, \langle\langle u', g'\rangle, m\rangle\rangle$ for a domain $m \in \mathsf{dom}(i)$ and some n, u', g' is created in r (in Line 38 of Algorithm 18) and n is sent to the browser b, then

The user is logged in (authn) or the user's data is accessed (authz) only if the user expressed her wish to log in before.

Formal Analysis of OAuth 2.0

- Introduction to OAuth 2.0
- **Attacks on OAuth 2.0**

(Selected) Attacks on OAuth 2.0

- **307 Redirect Attack**
- IdP Mix-Up Attack

breaks authentication and authorization properties

- State Leak Attack

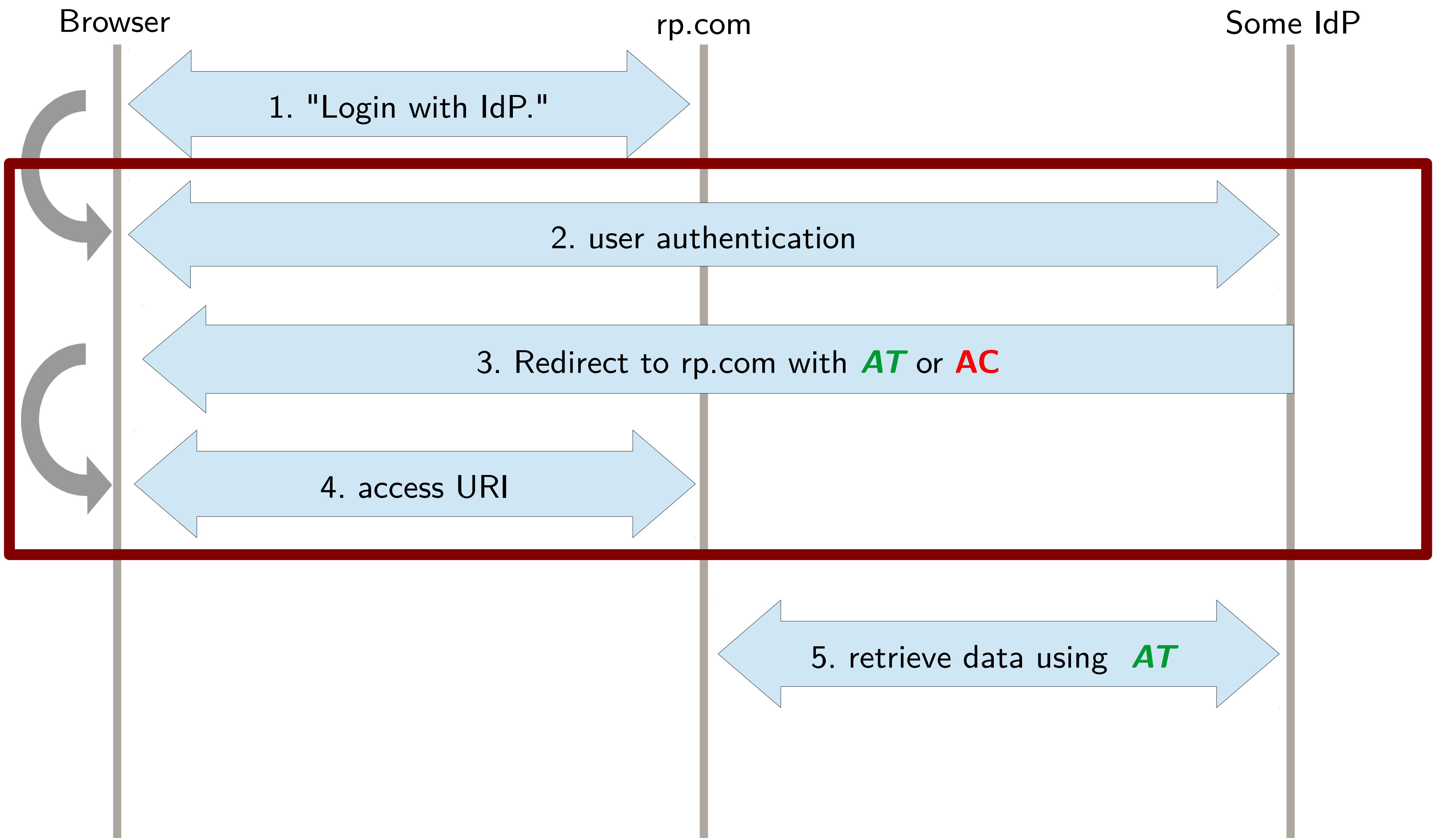
breaks session integrity property

Daniel Fett, Ralf Küsters, and Guido Schmitz.
A Comprehensive Formal Security Analysis of OAuth 2.0.
ACM CCS 2016.
https://sec.informatik.uni-stuttgart.de/publications

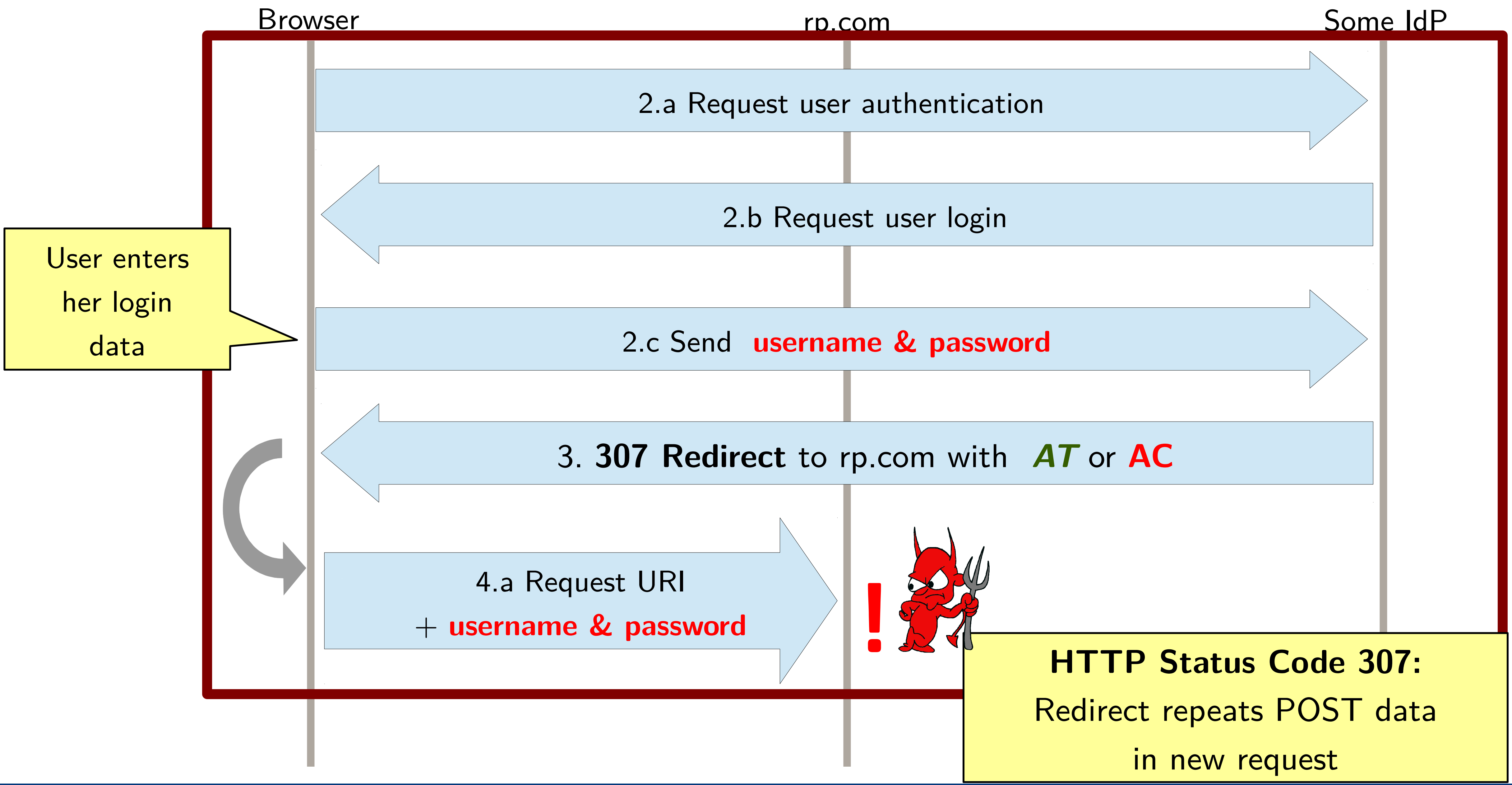
307 Redirect Attack

In the 307 Redirect Attack, the IdP accidentally instructs the browser to forward the user credentials to the RP.

307 Redirect Attack



307 Redirect Attack



307 Redirect Attack

The attacker receives the username and password of the user.

OAuth standard says:

1.7. HTTP Redirections

```
This specification makes extensive use of HTTP redirections, in which
the client or the authorization server directs the resource owner's
user-agent to another destination.  While the examples in this
specification show the use of the HTTP 302 status code, any other
method available via the user-agent to accomplish this redirection is
allowed and is considered to be an implementation detail.
```

Mitigation:

Use status code 303 or any other method that does not forward POST data.

(Selected) Attacks on OAuth 2.0

- 307 Redirect Attack
- **IdP Mix-Up Attack**

breaks authentication and authorization properties

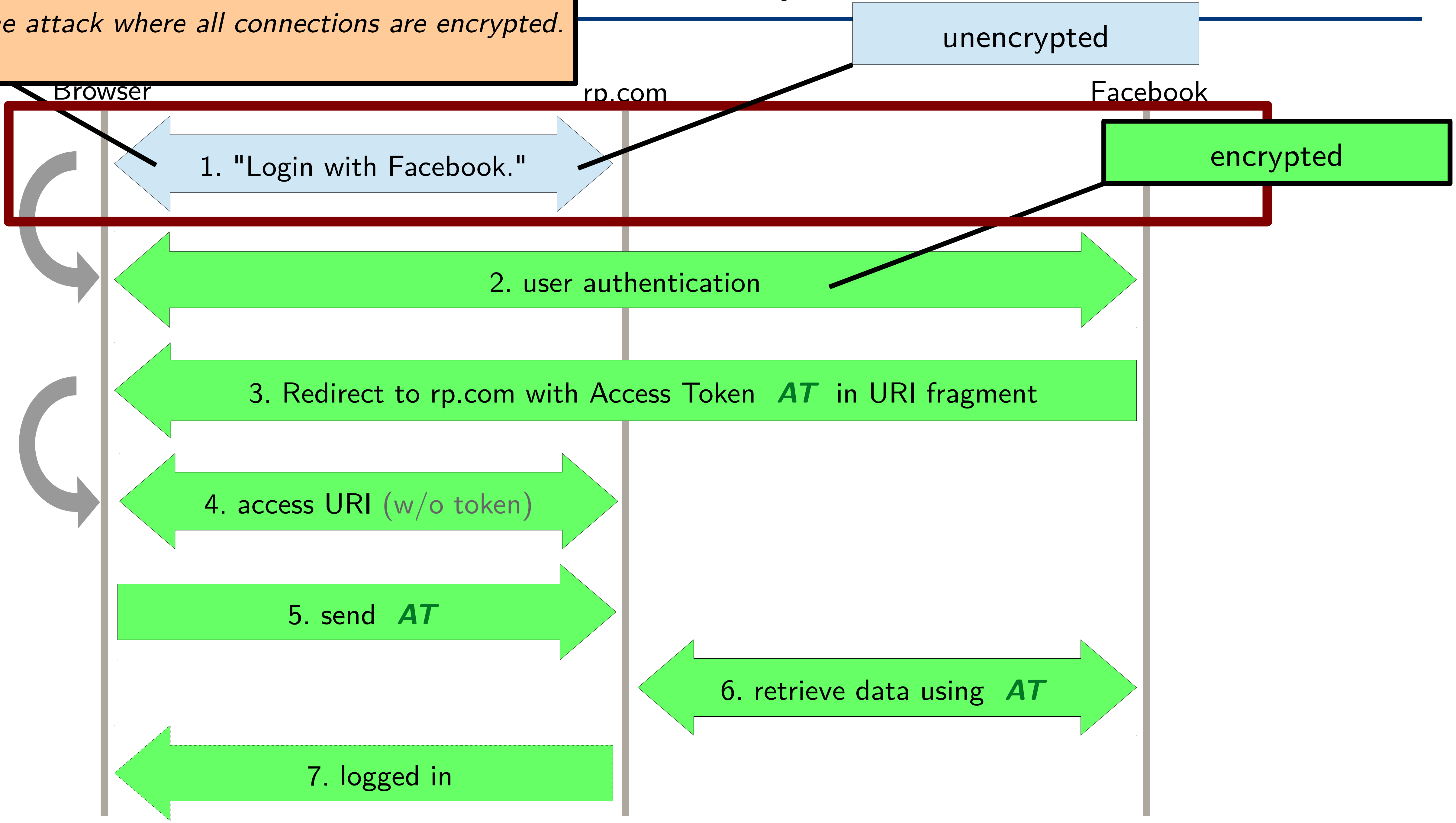
- State Leak Attack

breaks session integrity property

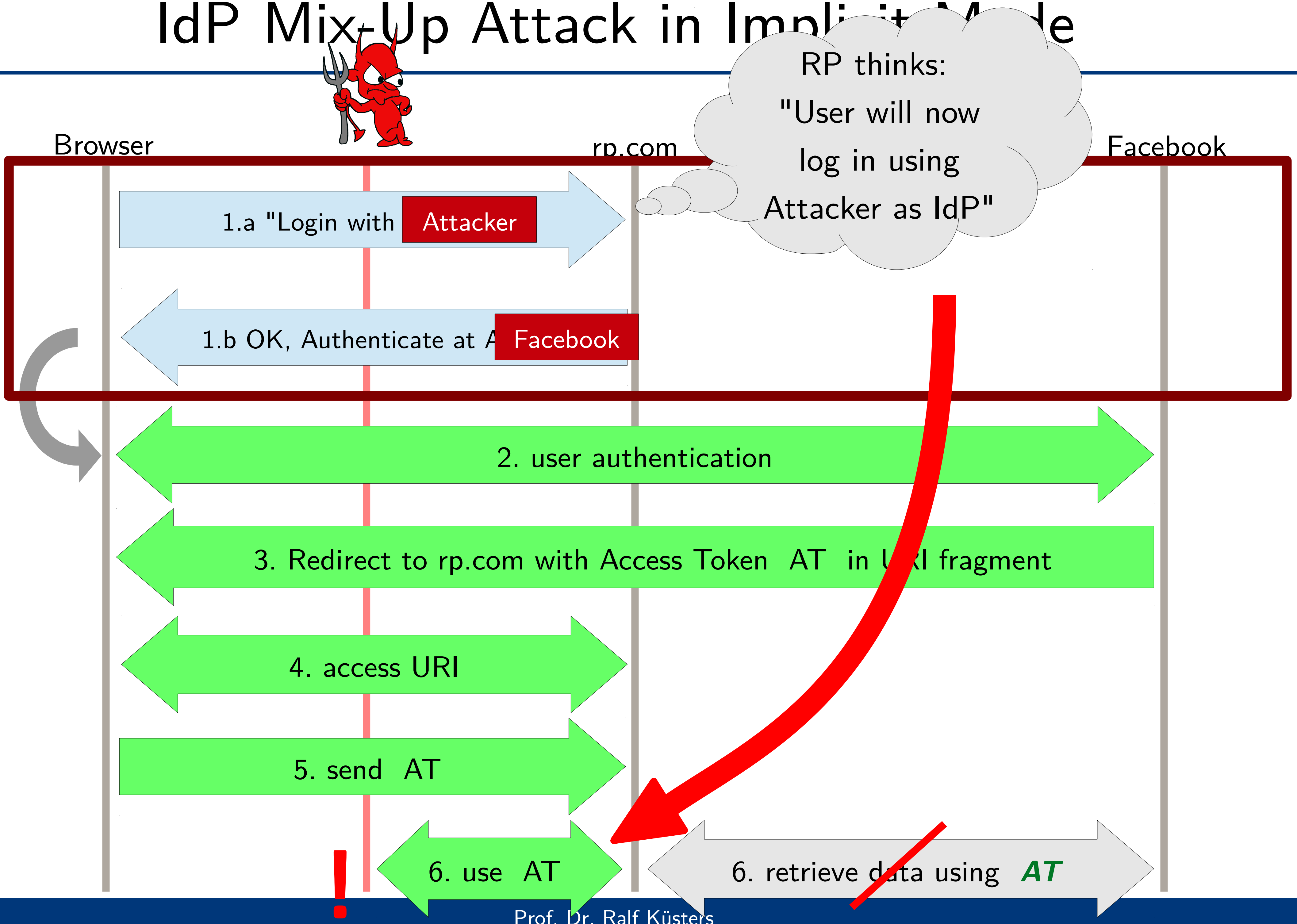
Daniel Fett, Ralf Küsters, and Guido Schmitz.
A Comprehensive Formal Security Analysis of OAuth 2.0.
ACM CCS 2016.
https://sec.informatik.uni-stuttgart.de/publications

...ck in Implicit Mode

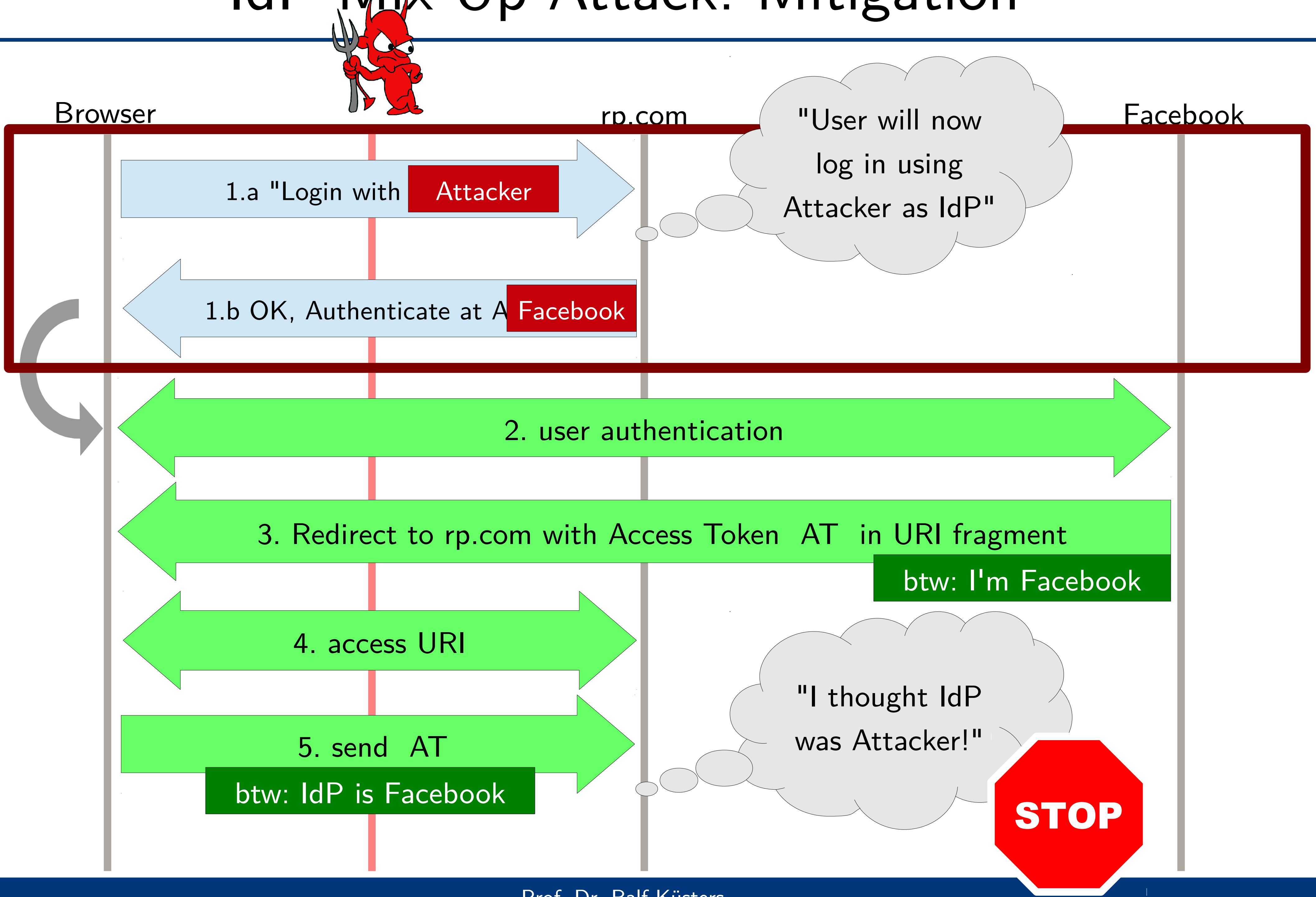
See our CCS 2016 paper
for a version of the attack where all connections are encrypted.



IdP Mix-Up Attack in Impli… …de



IdP Mix-Up Attack: Mitigation



(Selected) Attacks on OAuth 2.0

- 307 Redirect Attack
- IdP Mix-Up Attack

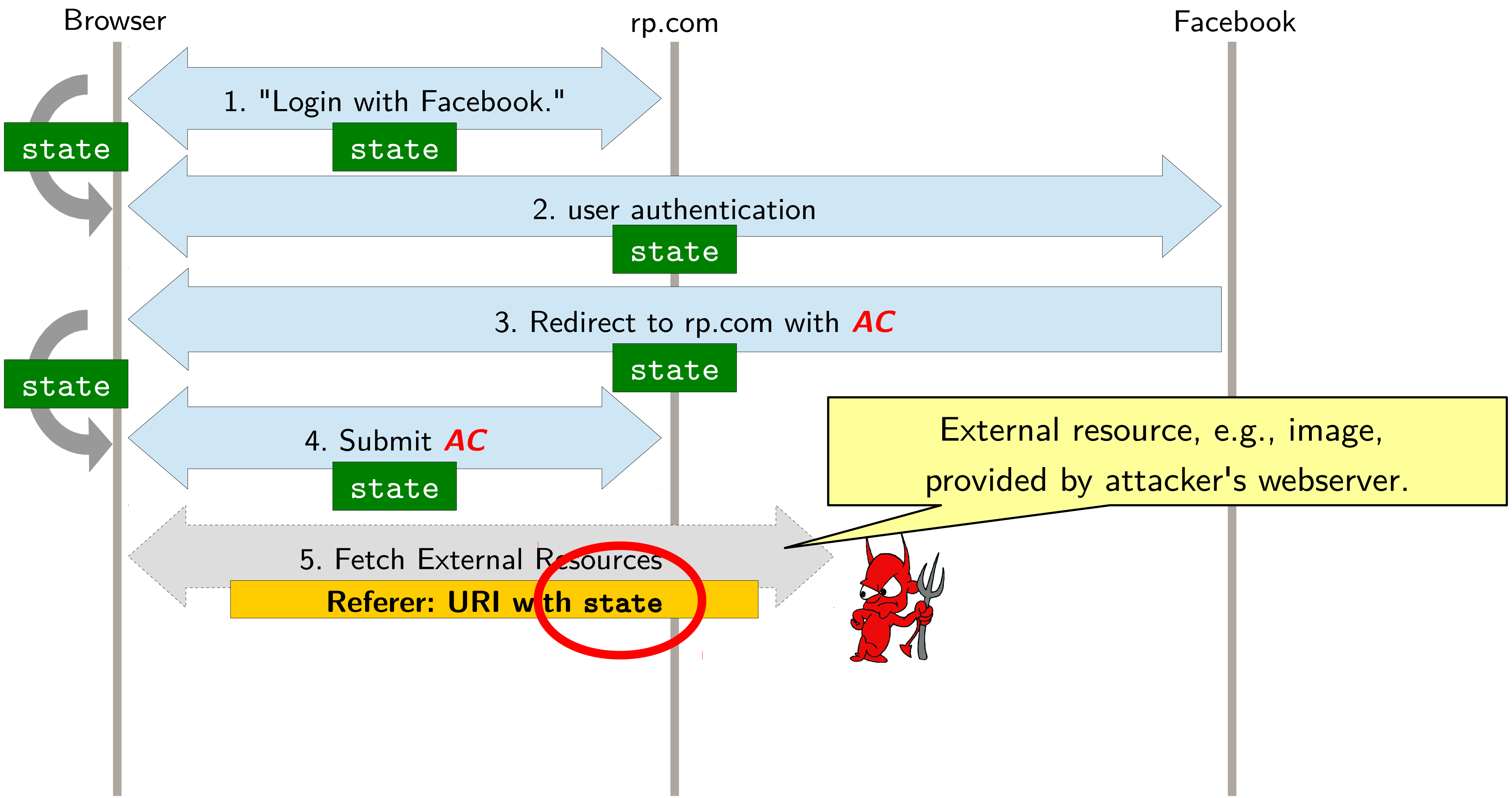
breaks authentication and authorization properties

- **State Leak Attack**

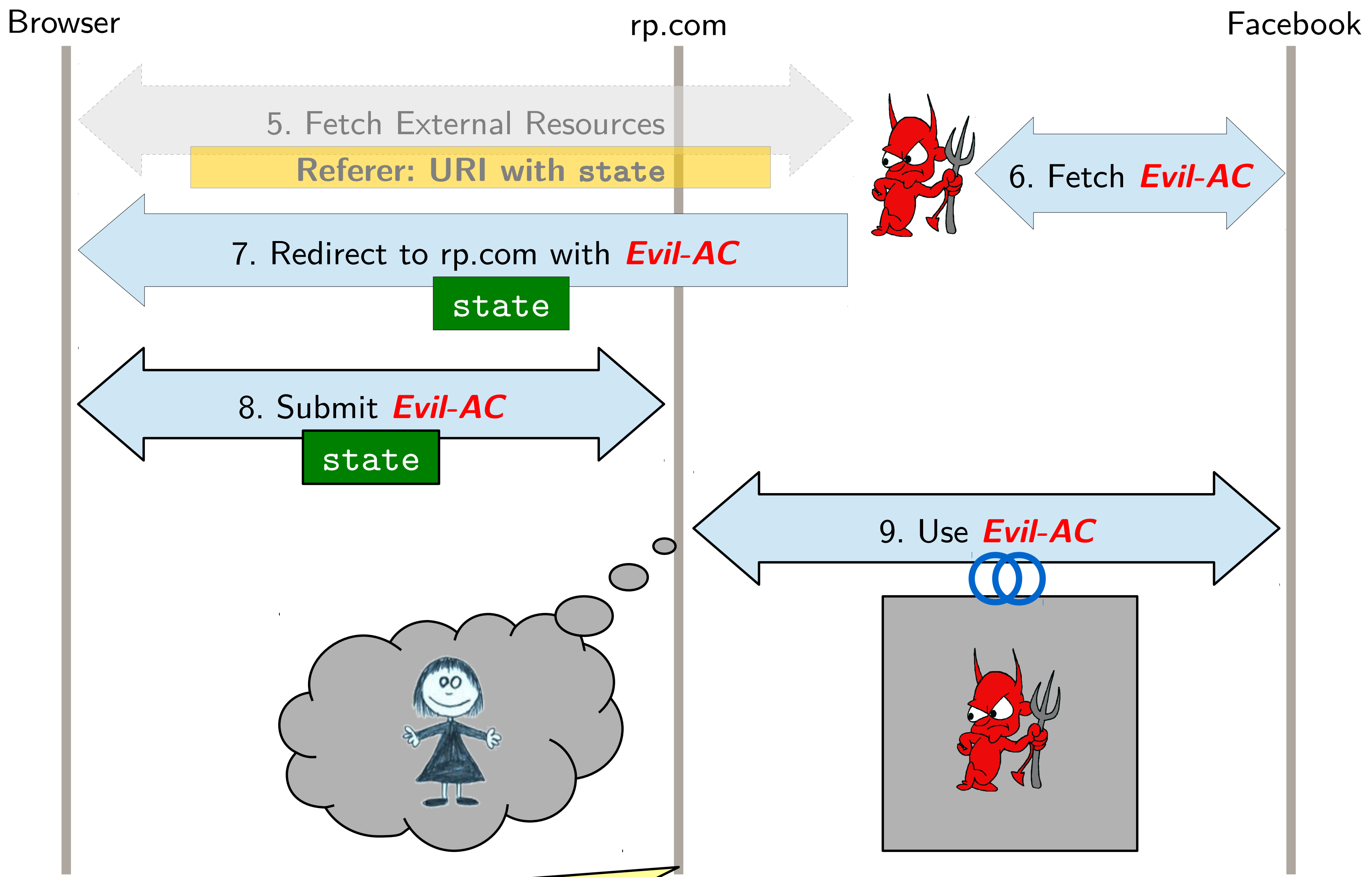
breaks session integrity property

Daniel Fett, Ralf Küsters, and Guido Schmitz.
A Comprehensive Formal Security Analysis of OAuth 2.0.
ACM CCS 2016.
https://sec.informatik.uni-stuttgart.de/publications

State Leak Attack



State Leak Attack



RP expects to use a resource of Alice
but instead uses a resource of the attacker.

(Selected) Attacks on OAuth 2.0

- 307 Redirect Attack
- IdP Mix-Up Attack

breaks authentication and authorization properties

- **State Leak Attack**

breaks session integrity property

Daniel Fett, Ralf Küsters, and Guido Schmitz.
A Comprehensive Formal Security Analysis of OAuth 2.0.
ACM CCS 2016.
https://sec.informatik.uni-stuttgart.de/publications

Proving the Security of OAuth 2.0

Requirement: Fixes for all discovered and previously known attacks

Theorem 1. Let $\mathcal{OAuthWS}^n$ be an OAuth web system with a network attacker, then $\mathcal{OAuthWS}^n$ is secure w.r.t. authorization and secure w.r.t. authentication. Let $\mathcal{OAuthWS}^w$ be an OAuth web system with web attackers, then $\mathcal{OAuthWS}^w$ is secure w.r.t. session integrity for authorization and authentication.

Impact

- Disclosed OAuth attacks to the IETF Web Authorization Working Group in late 2015
- Emergency meeting with the working group four weeks later
- Public disclosure early 2016
- Initiated the OAuth Security Workshop (OSW) to foster the exchange between researchers, standardization groups, and industry
- OSW held annually; next edition: 20-22 March 2019, Stuttgart
- Joined the working group to codify our fixes into a new OAuth Security RFC/BCP (Best Current Practice)

Note: OAuth 2.0 has been analyzed many times before, but not based on rigorous formal model. Formal proofs (proof attempts) revealed new attacks and model enabled security proofs of fixed systems.

Conclusion

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016, CSF 2017]

- We have developed a formal model of the web infrastructure to analyze the security of web standads and applications.
- Found several attacks on SSO systems
  (Mozilla BrowserID/Persona, OAuth 2.0, OpenID Connect)
- Proved security of fixed systems.
- Proposed SSO with unique privacy feature: SPRESSO
- Working in the IETF to fix OAuth standard

IETF

Thank you!