University of Stuttgart
Institute of
Information Security

Thesis Defense

An Expressive Formal Model of the Web Infrastructure

Dipl.-Inf. Daniel Fett

2018-10-19

Many Web Attacks...

Cross-Site Request Forgery

Attacks on Single Sign-on

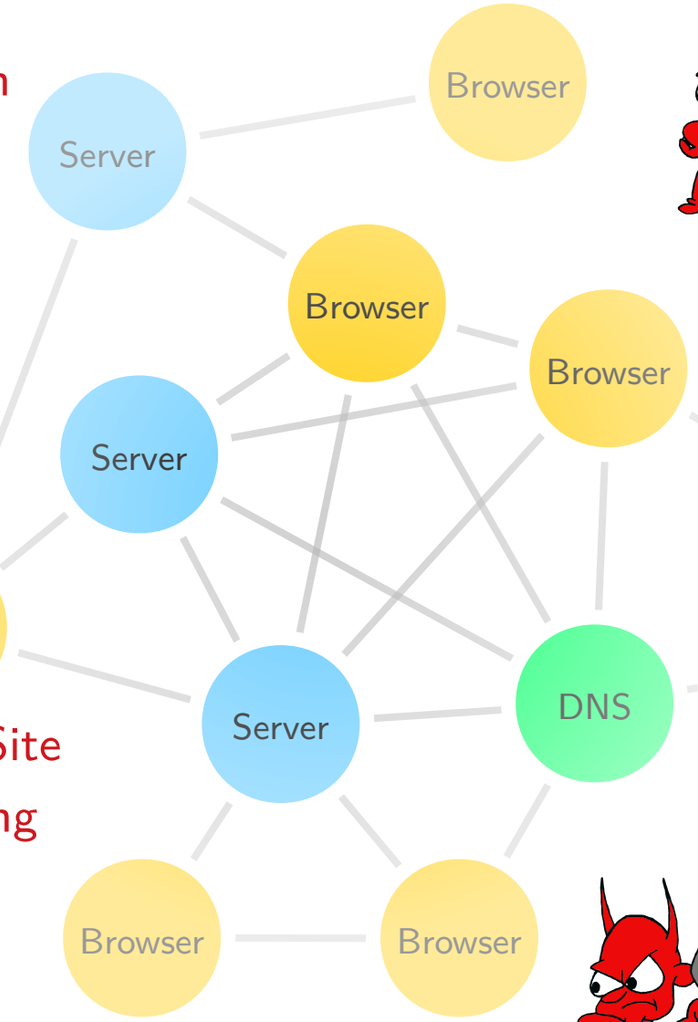
DNS Rebinding

Leaks of Sensitive Data

Clickjacking

Cross-Site Scripting

Man-in-the-middle Attacks



SQL Injection

Malicious CDNs

Attacks on Session Management

Malicious iframes

Missing Checks

Other Injection Attacks

Attacks on Cookies

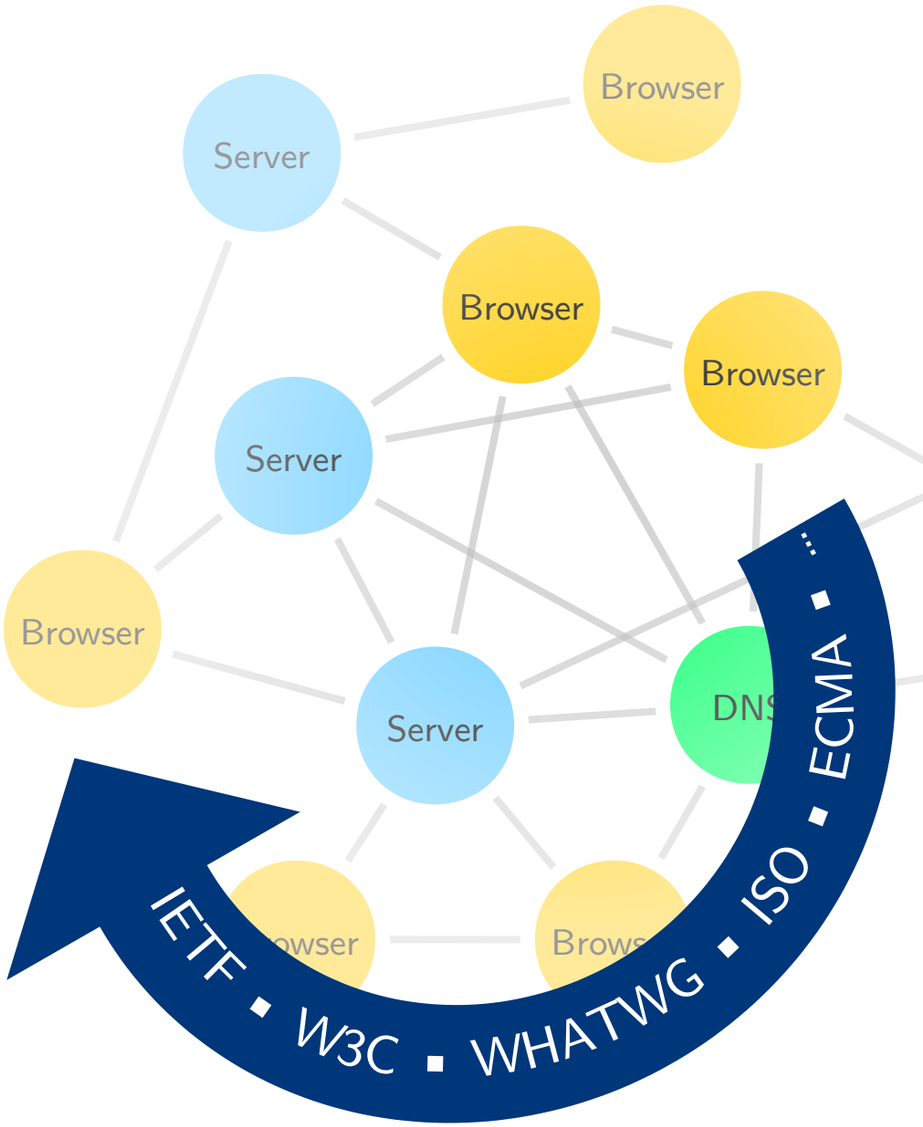
...but why?

The web is complex ...

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

... and web applications as well.

- More features, more interaction
- Many bugs and errors

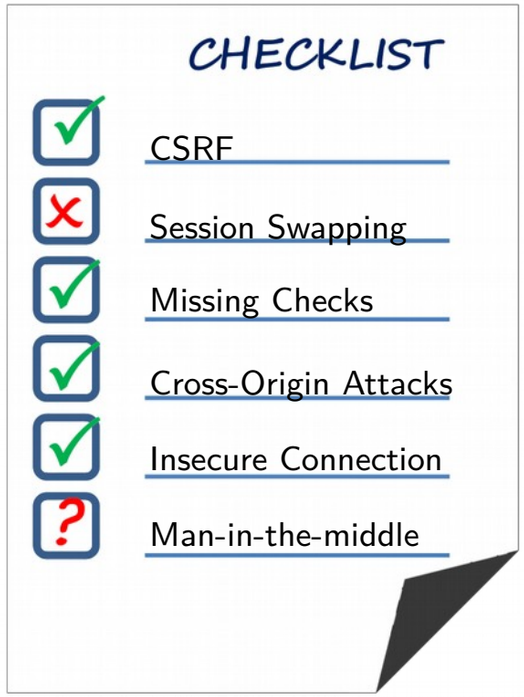


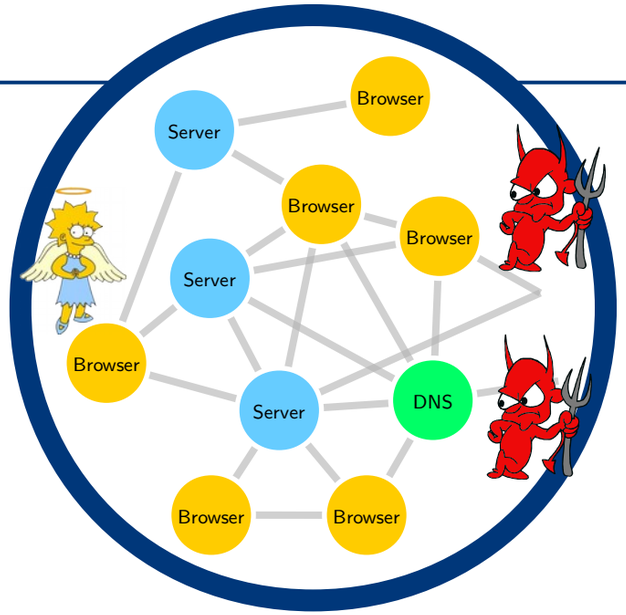
Finding vulnerabilities is hard!

Current Methods

Expert review

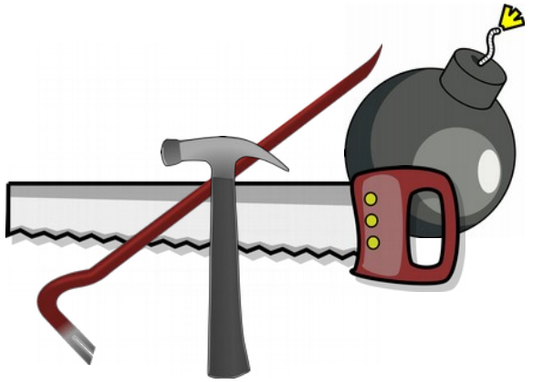
of standards and implementations





Penetration testing

using tools or manual analysis

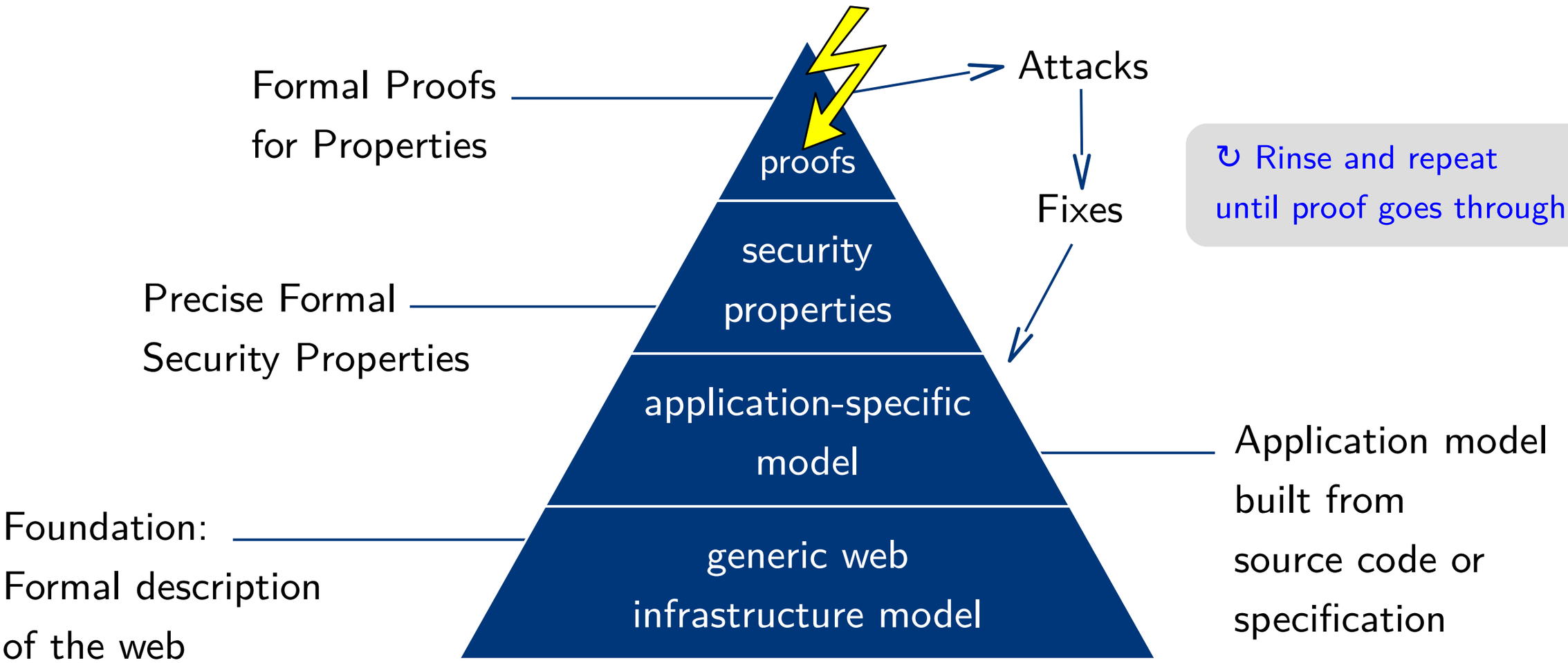


Downsides

- It is easy to miss attacks, even for experts
- Pentesting focuses on known attacks
- Finding new attack types depends on the creativity of the experts
- Both methods do not guarantee security, not even for a limited set of attacks

Can we develop a more systematic way of finding attacks?

Model-based Approach

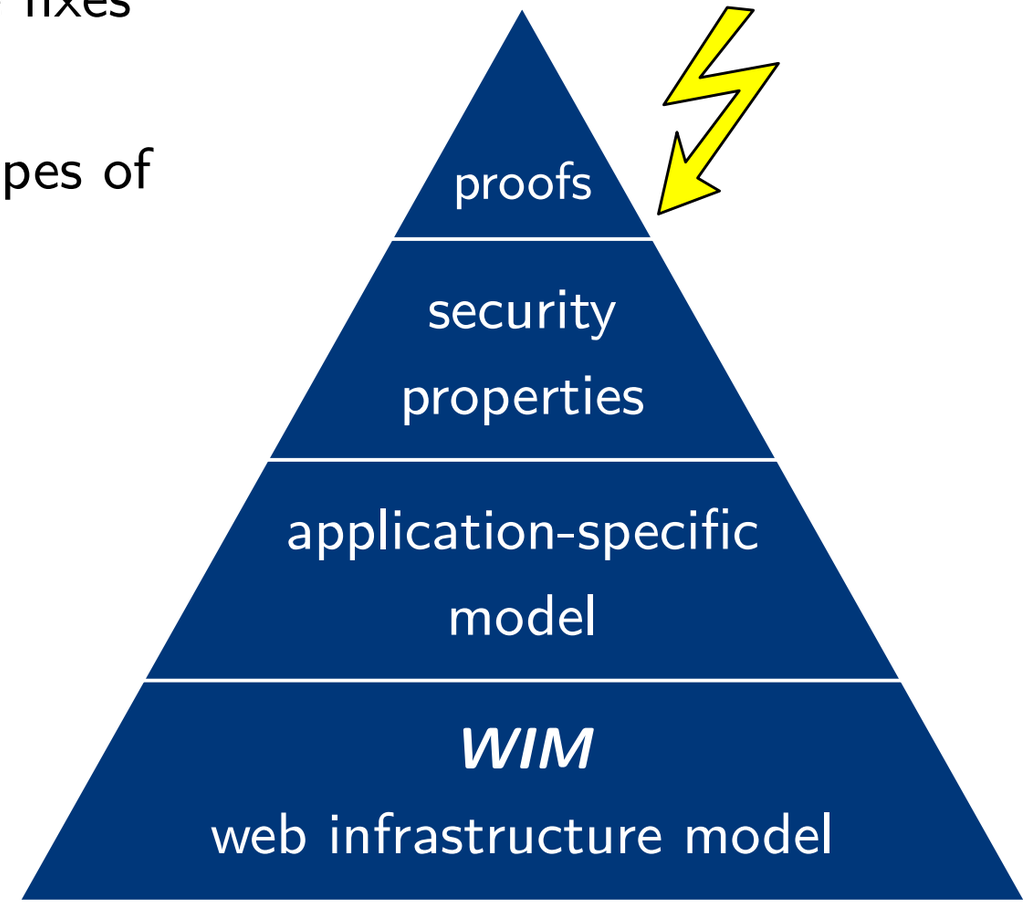


↻ Rinse and repeat
until proof goes through.

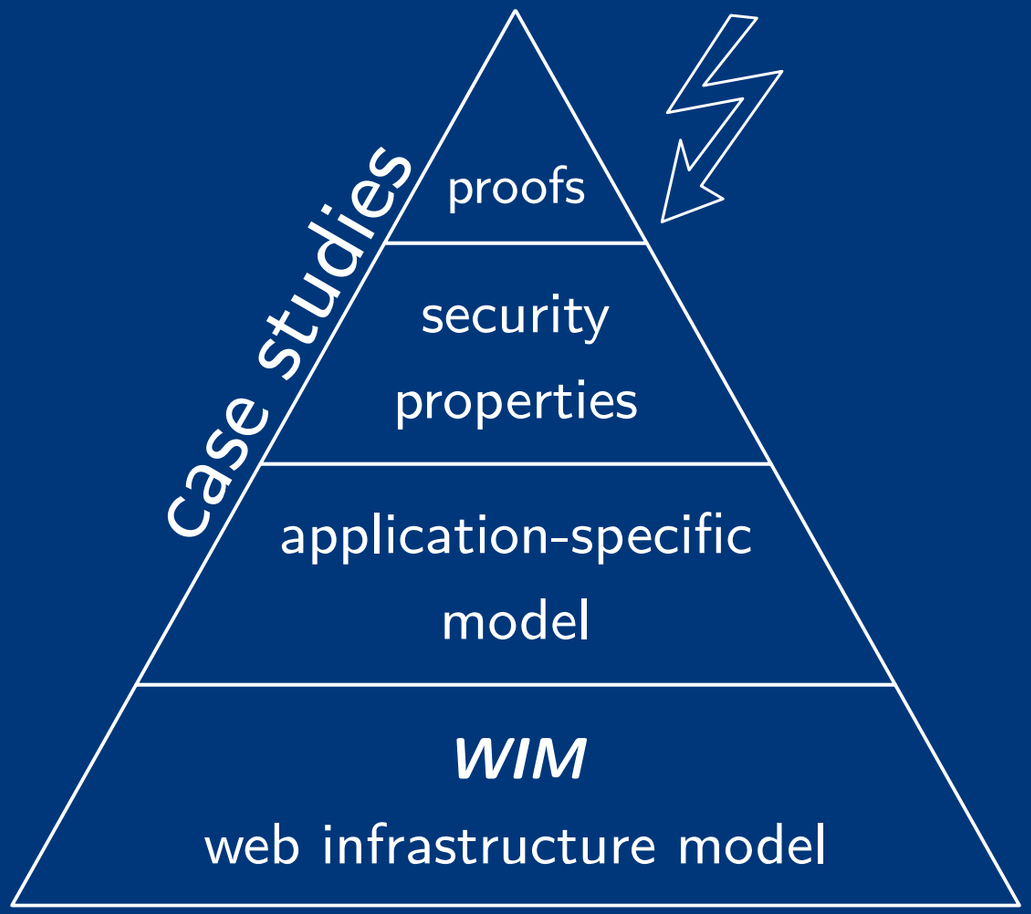
Advantages

This approach can yield...

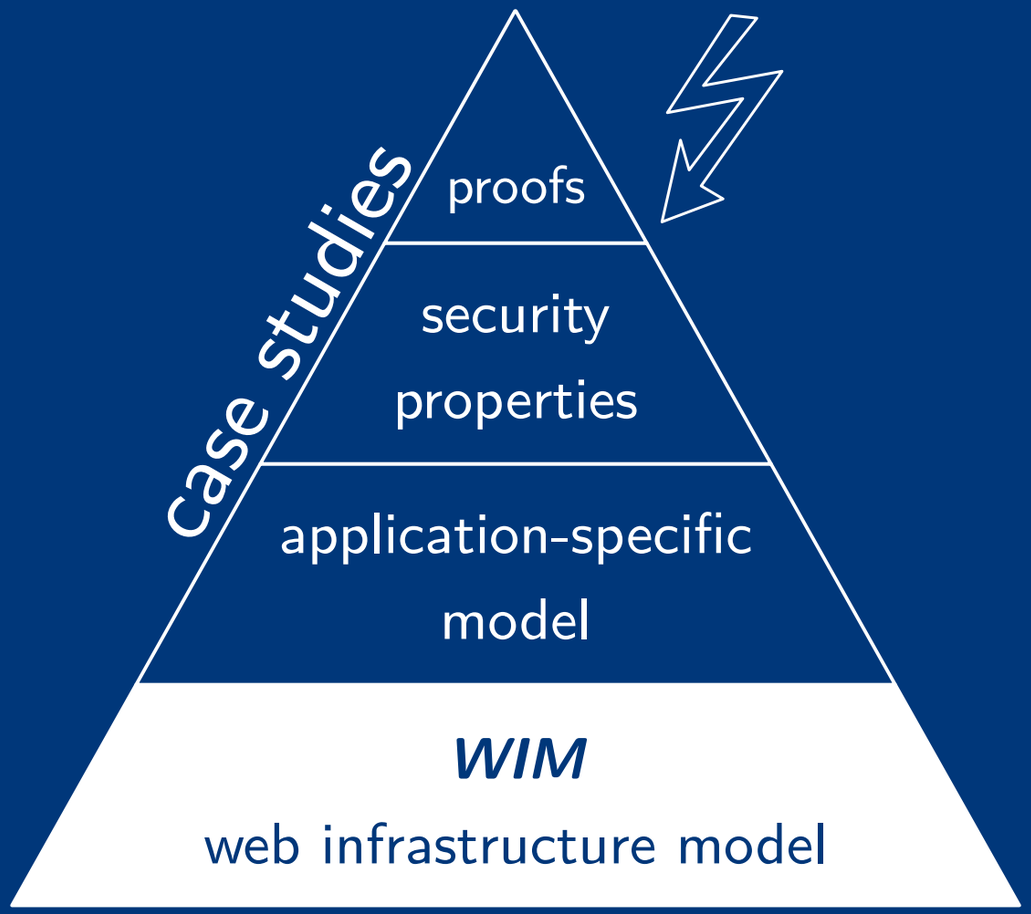
- new attacks and respective fixes
- strong security guarantees excluding even unknown types of attacks



An Expressive Formal Model of the Web Infrastructure



An Expressive Formal Model of the Web Infrastructure



A Short History of Web Models

- [Kerschbaum 2007]

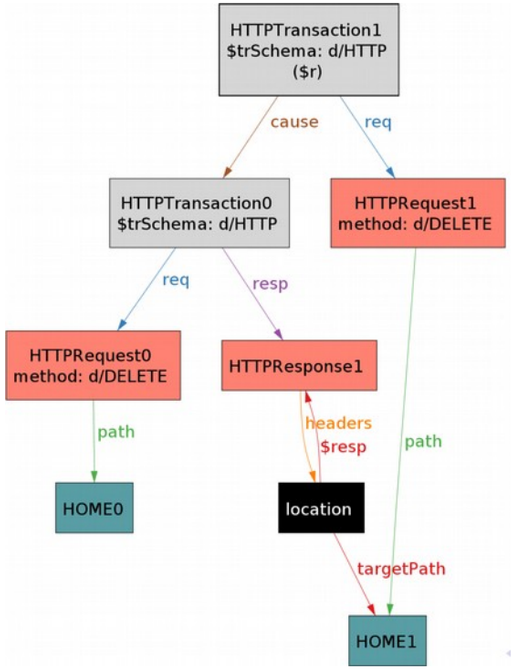
  Analysis of CSRF protection in Alloy model checker

- [Akhawe et al. 2010]

  First formal "web model", in Alloy, five case studies

- [Bansal et al. 2012, 2013, 2014]

  Formal web model with many web features, based on ProVerif tool, new attacks on encrypted cloud storage and OAuth 2.0



Focus on automated, tool-based analysis.

Drawbacks:

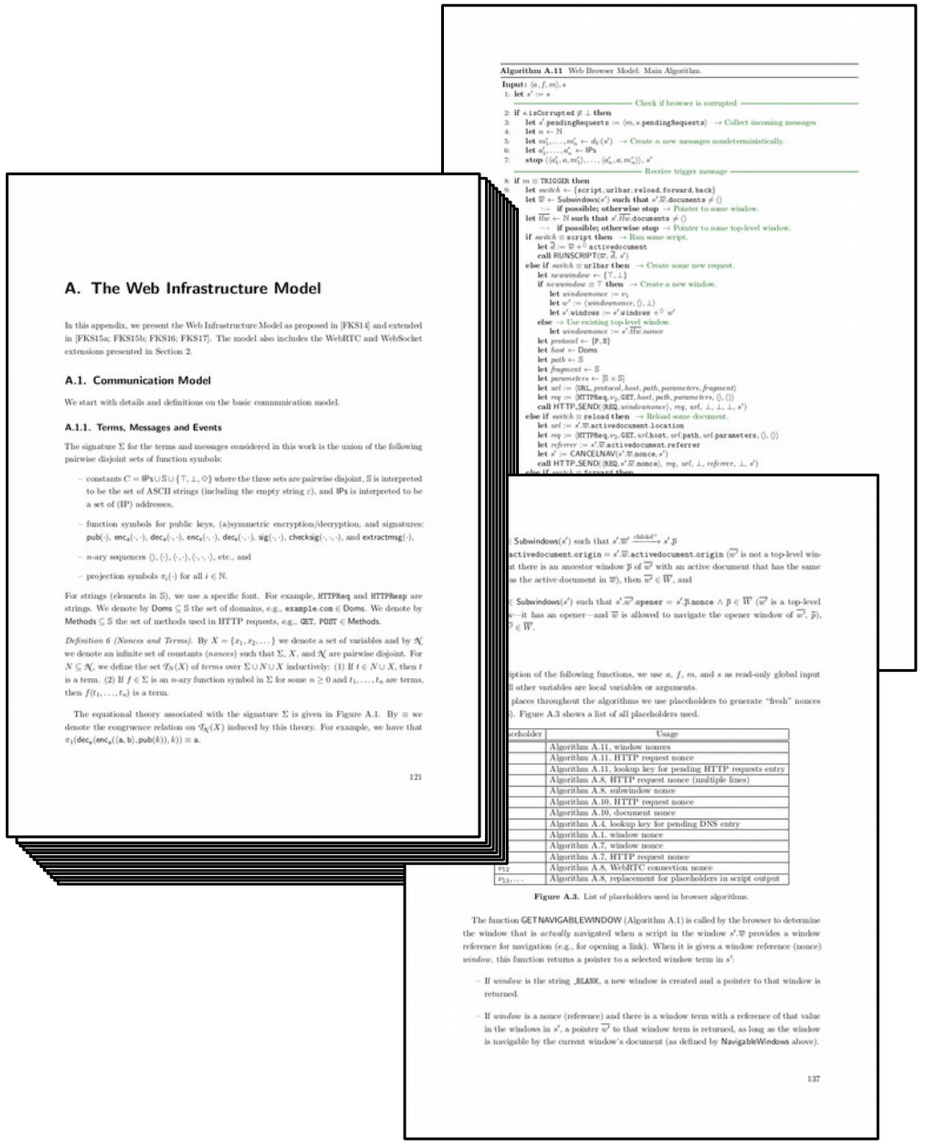
Limitations and constraints of tools (e.g., encoding of messages/terms and data structures)

My approach:

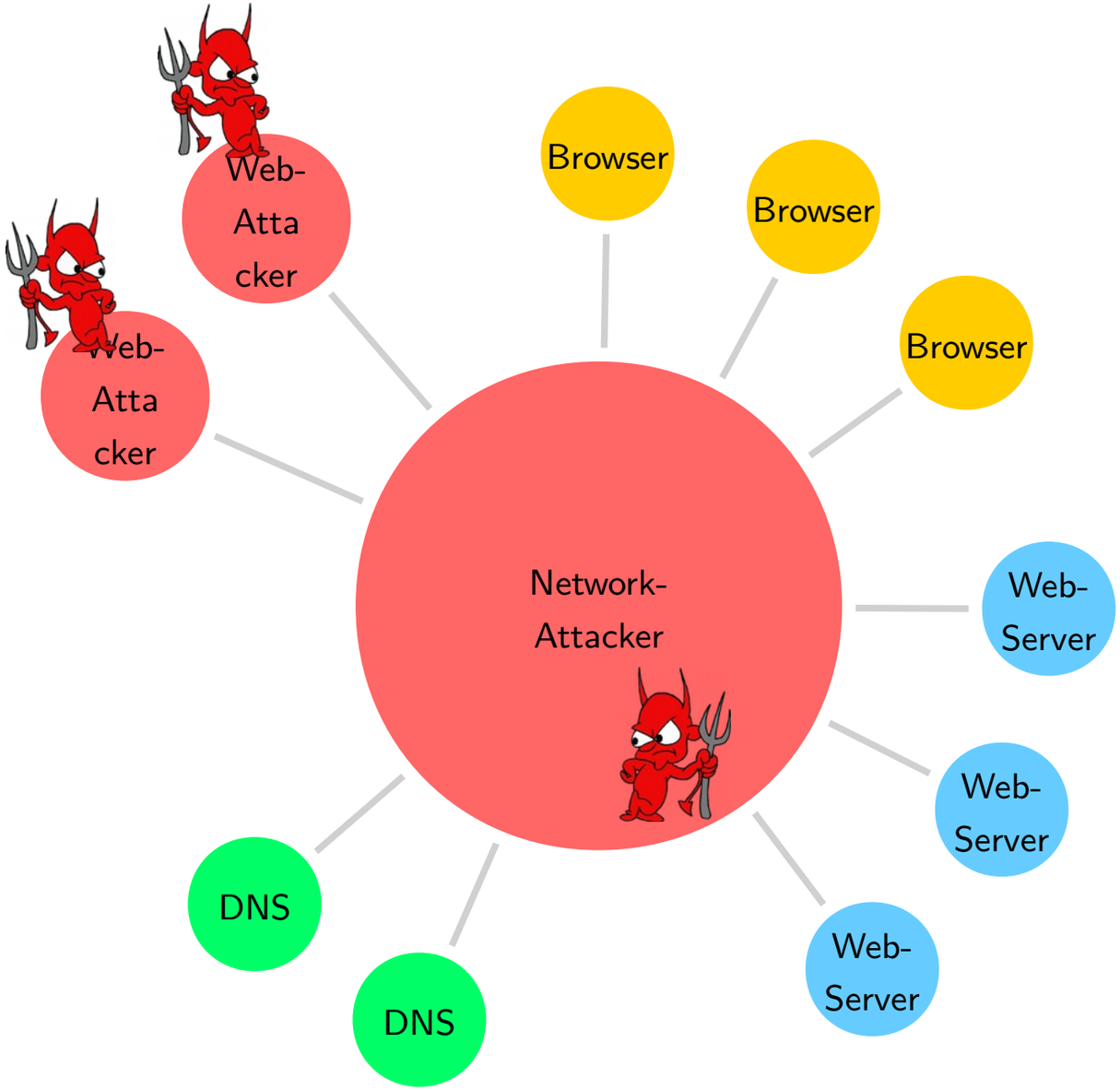
Manual (pen-and-paper) model

The Web Infrastructure Model *WIM*

- Detailed, comprehensive, and precise formal model
  - Network interactions
  - Attacker behaviour
  - DNS servers
  - Generic web server model
  - Web browsers
- Summarizes and condenses relevant standards
- Solid basis for security and privacy analyses
  - of web standards and applications
- Reference model
  - developers, researchers, teaching, and tool-based analysis



WIM Network Model and Attackers

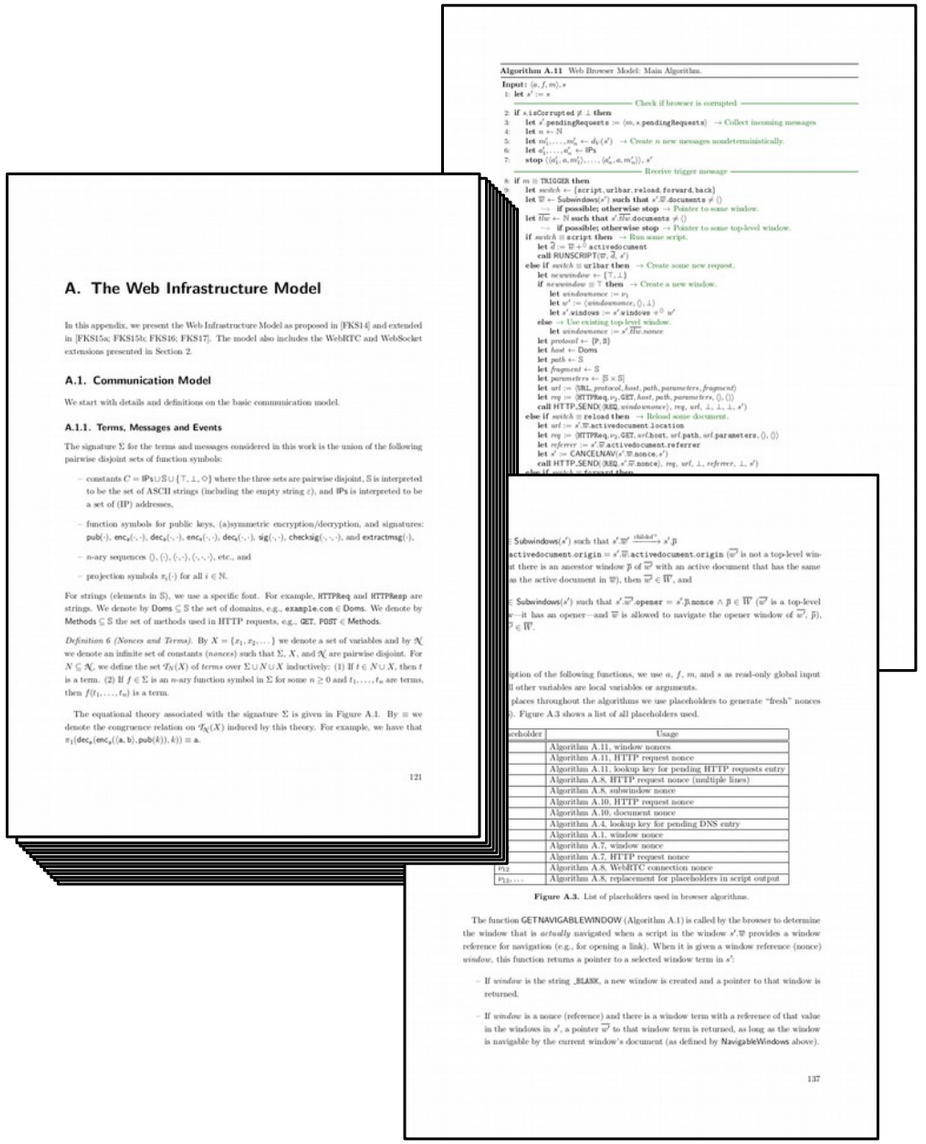


The Web Infrastructure Model *WIM*

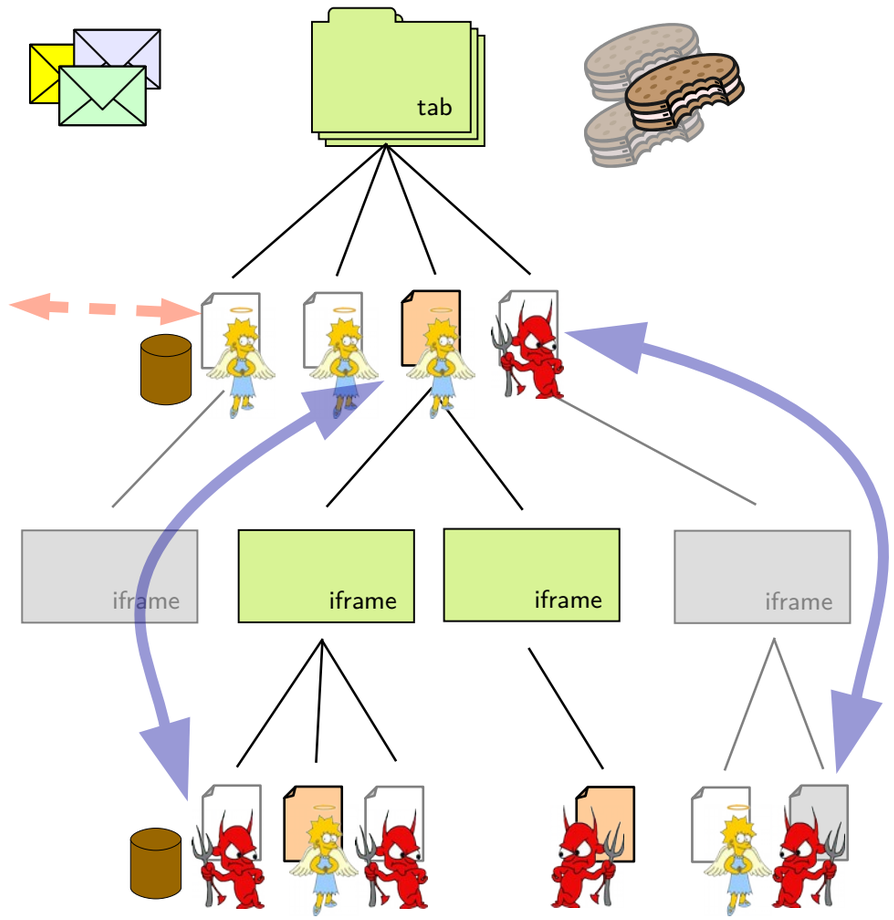
- Detailed, comprehensive, and precise formal model
  - Network interactions
  - Attacker behaviour
  - DNS servers
  - Generic web server model
  - Web browsers
- Summarizes and condenses relevant standards
- Solid basis for security and privacy analyses

  of web standards and applications
- Reference model

  developers, researchers, teaching, and tool-based analysis



WIM 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
- WebRTC (new!)
- ...






WIM Web Browser Model - Example

Algorithm 8 Web Browser Model: Process an HTTP response.

```
1:  function PROCESSRESPONSE(response, reference, request, requestUrl, key, f, s')
2:      if Set-Cookie ∈ response.headers then
3:          for each c ∈⟨⟩ response.headers[Set-Cookie], c ∈ Cookies do
4:              let s'.cookies[request.host]
                  ↪  := AddCookie(s'.cookies[request.host], c)
5:      if Strict-Transport-Security ∈ response.headers ∧ requestUrl.protocol ≡ S then
6:          let s'.sts := s'.sts +⟨⟩ request.host
7:      if Referer ∈ request.headers then
8:          let referrer := request.headers[Referer]
9:      else
10:         let referrer := ⊥
11:     if Location ∈ response.headers ∧ response.status ∈ {303, 307} then
12:         let url := response.headers[Location]
13:         if url.fragment ≡ ⊥ then
14:             let url.fragment := requestUrl.fragment
15:         let method' := request.method
16:         let body' := request.body
17:         if Origin ∈ request.headers then
18:             let origin := ⟨request.headers[Origin], ⟨request.host, url.protocol⟩⟩
19:         else
20:             let origin := ⊥
21:         if response.status ≡ 303 ∧ request.method ∉ {GET, HEAD} then
22:             let method' := GET
23:             let body' := ⟨⟩
```

The Web Infrastructure Model *WIM*

- Detailed, comprehensive, and precise formal model

  Network interactions

  Attacker behaviour

  DNS servers

  Generic web server model

  Web browsers

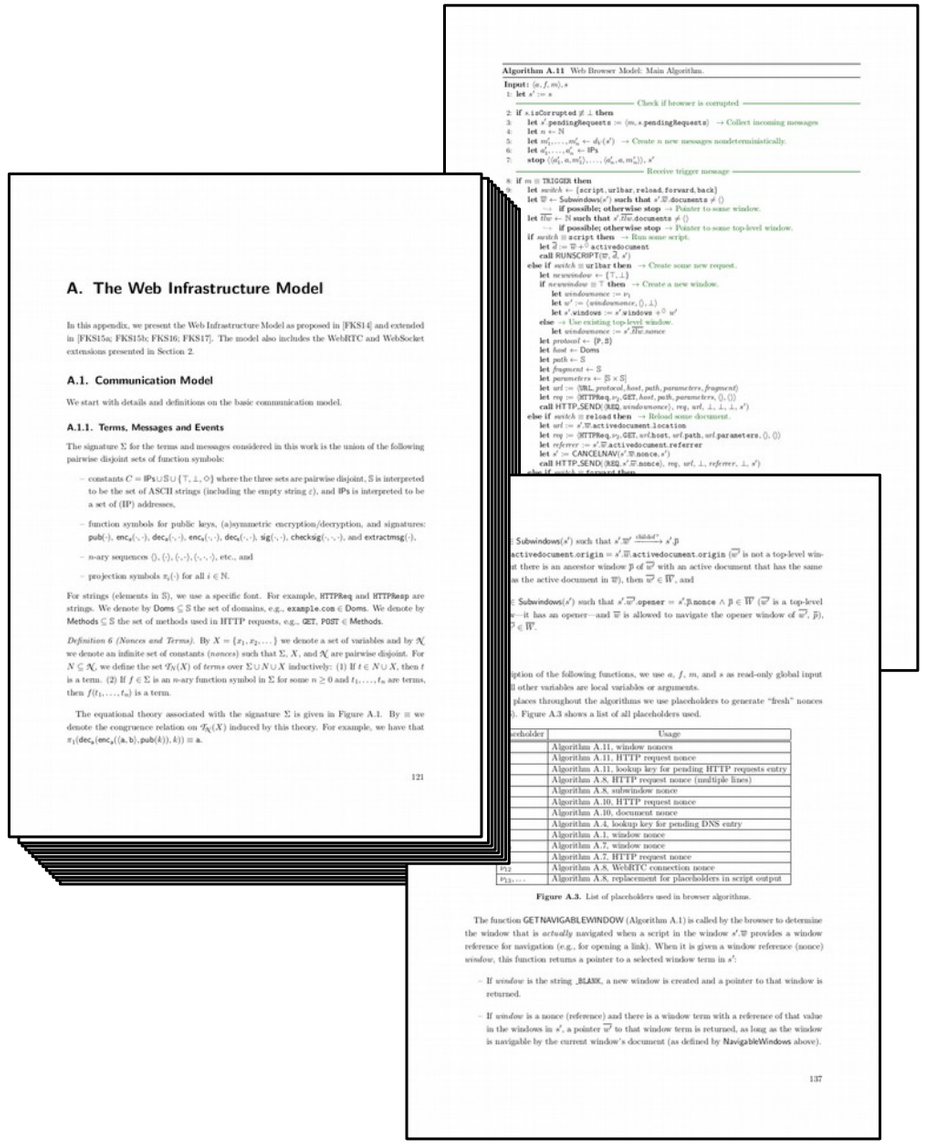
- Summarizes and condenses relevant standards

- Solid basis for security and privacy analyses

  of web standards and applications

- Reference model

  developers, researchers, teaching, and tool-based analysis



A. The Web Infrastructure Model

In this appendix, we present the Web Infrastructure Model as proposed in [FKS14] and extended in [FKS15a; FKS15b; FKS16; FKS17]. The model also includes the WebRTC and WebSocket extensions presented in Section 2.

A.1. Communication Model

We start with details and definitions on the basic communication model.

A.1.1. Terms, Messages and Events

Figure A.3. List of placeholders used in browser algorithms.

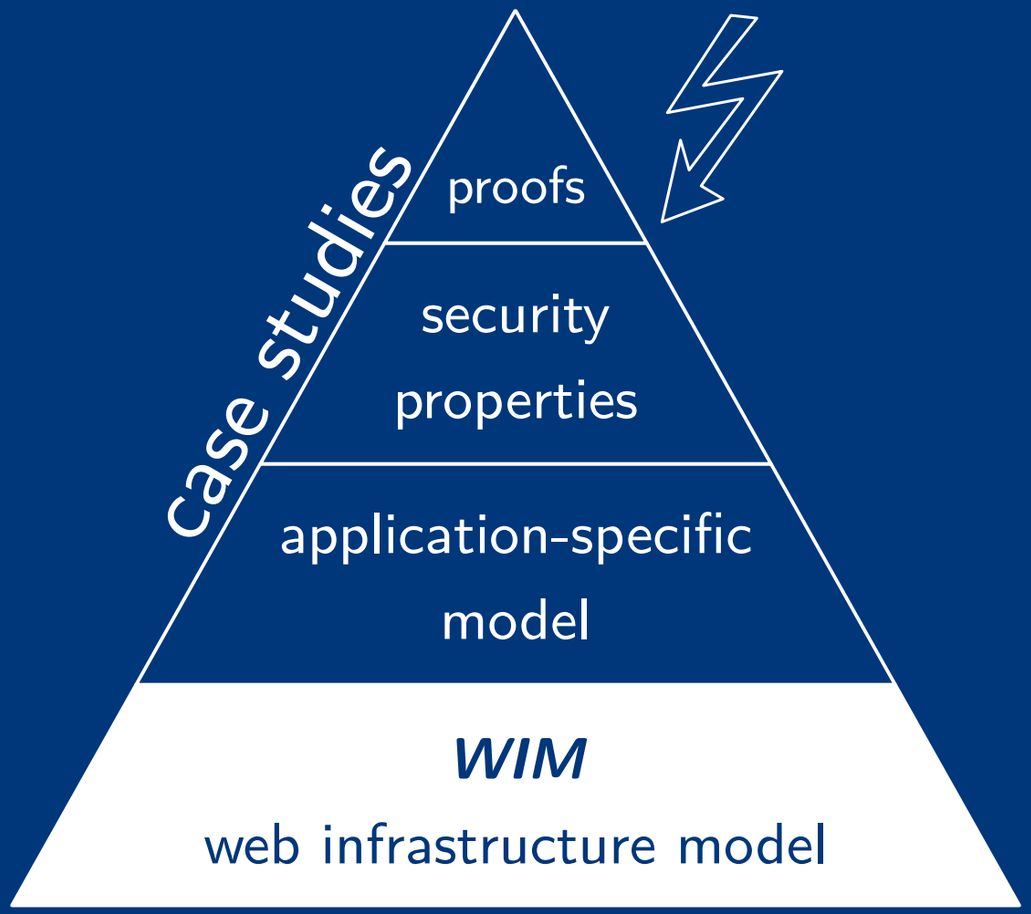
Limitations

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

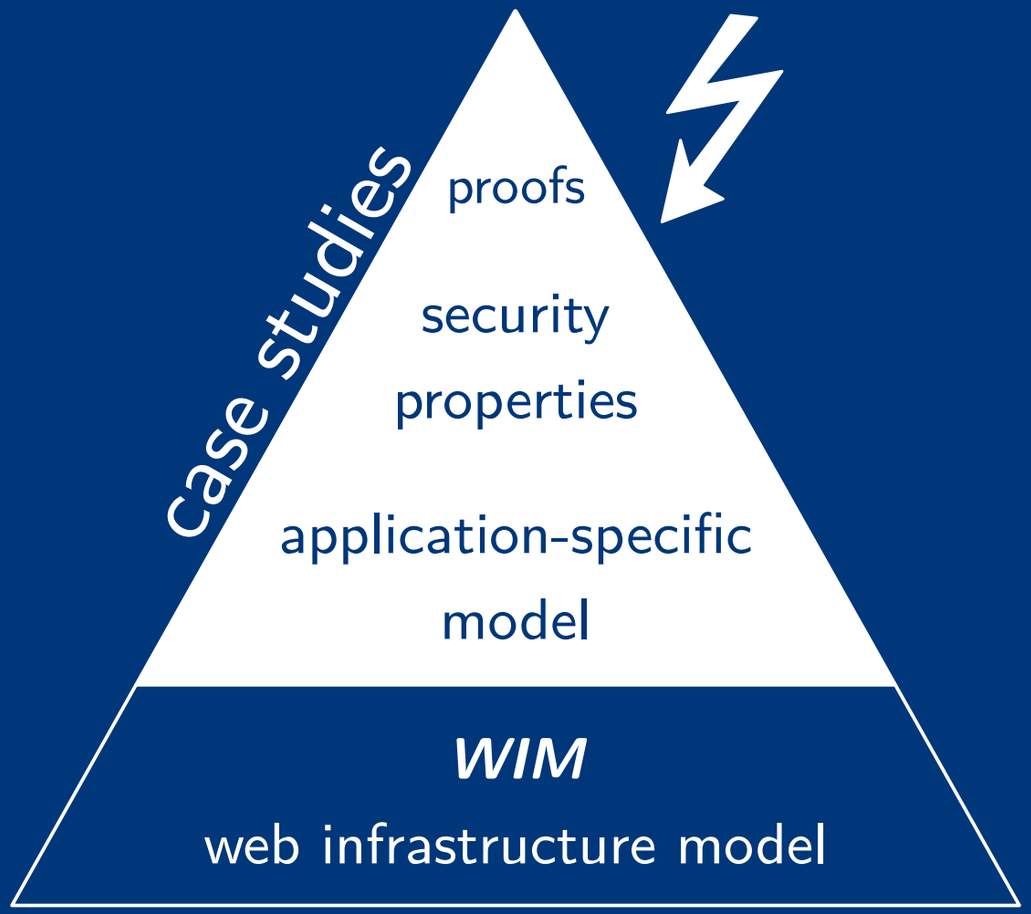
Model can in principle be extended to capture these aspects as well.

Trade-off: comprehensiveness vs. simplicity

An Expressive Formal Model of the Web Infrastructure



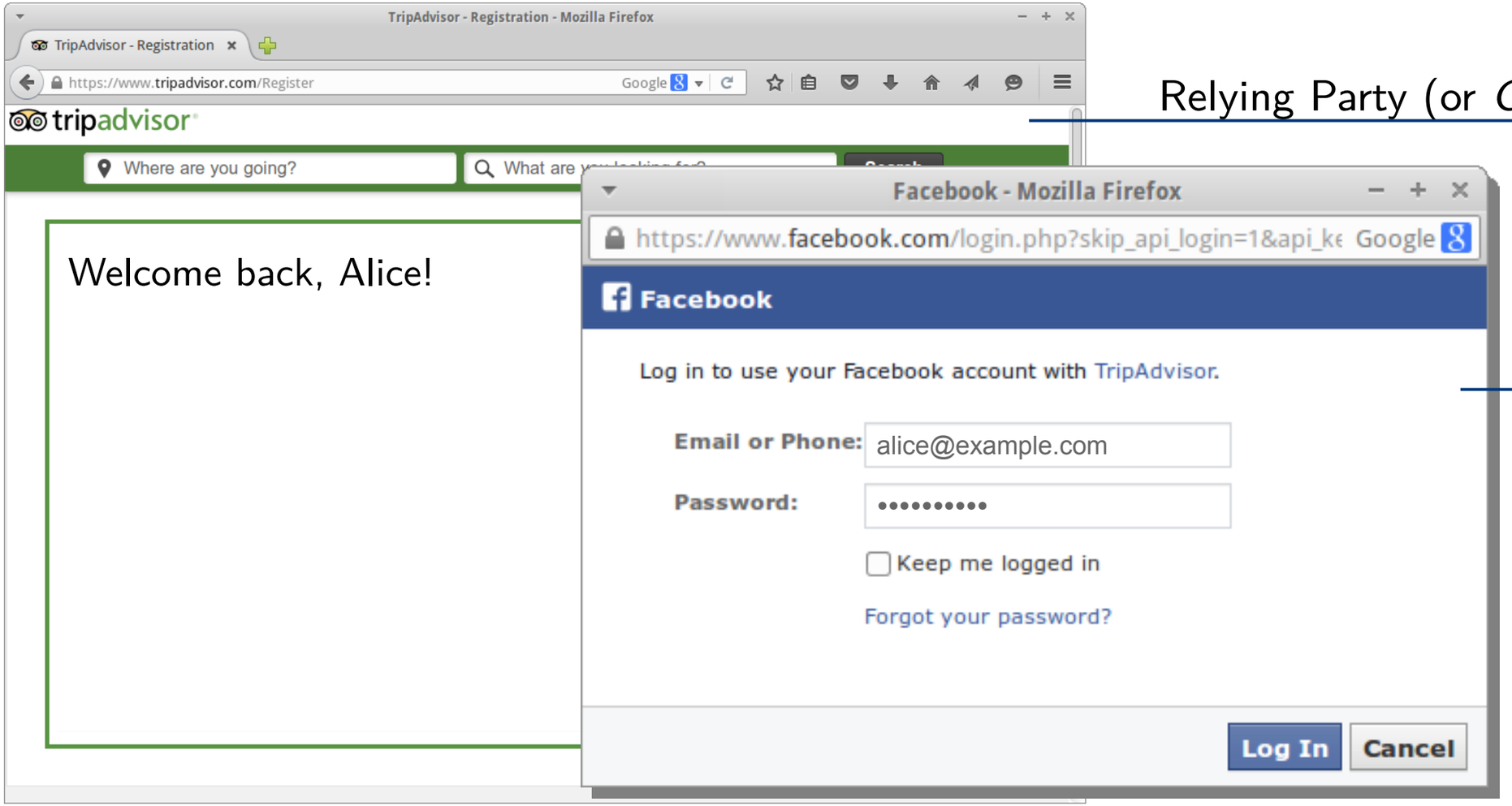
An Expressive Formal Model of the Web Infrastructure



WIM Case Studies

- Subject: web **single sign-on (SSO)** systems
- Interesting target for formal analysis:
  - Complex protocol flows
  - Multiple participants (typically ≥3)
  - High security requirements

WIM Case Studies



Relying Party (or *Client*)

Identity Provider

WIM Case Studies

Mozilla BrowserID

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

SPRESSO

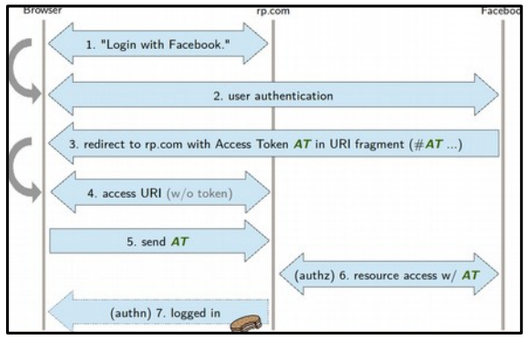
- Designed from scratch
- First formalized in *WIM*, then implemented
- First SSO with proven privacy and security

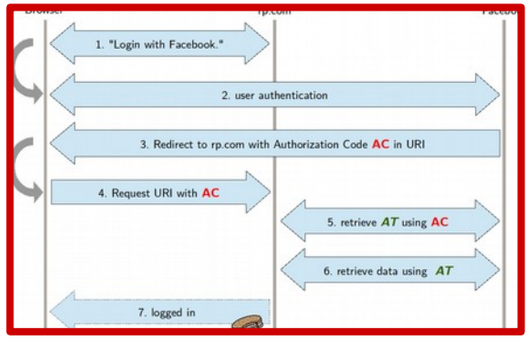
OAuth 2.0

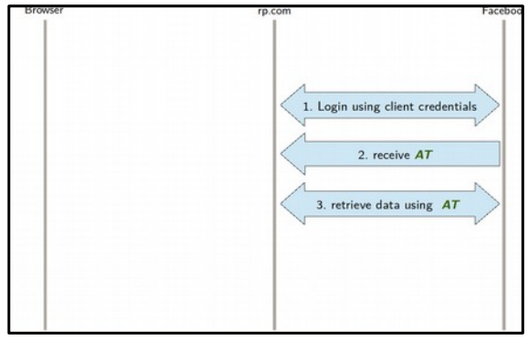
OpenID Connect

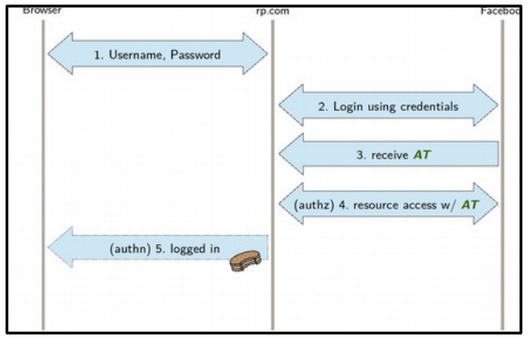
OAuth 2.0

- SSO framework used for authorization/authentication
- Specified by IETF (RFC6749), very widely used
  (e.g., Log in With Facebook)
- Many "variables":
  optional parameters, *public* and *confidential* clients, etc.
- Four different modes of interaction (*grants*)

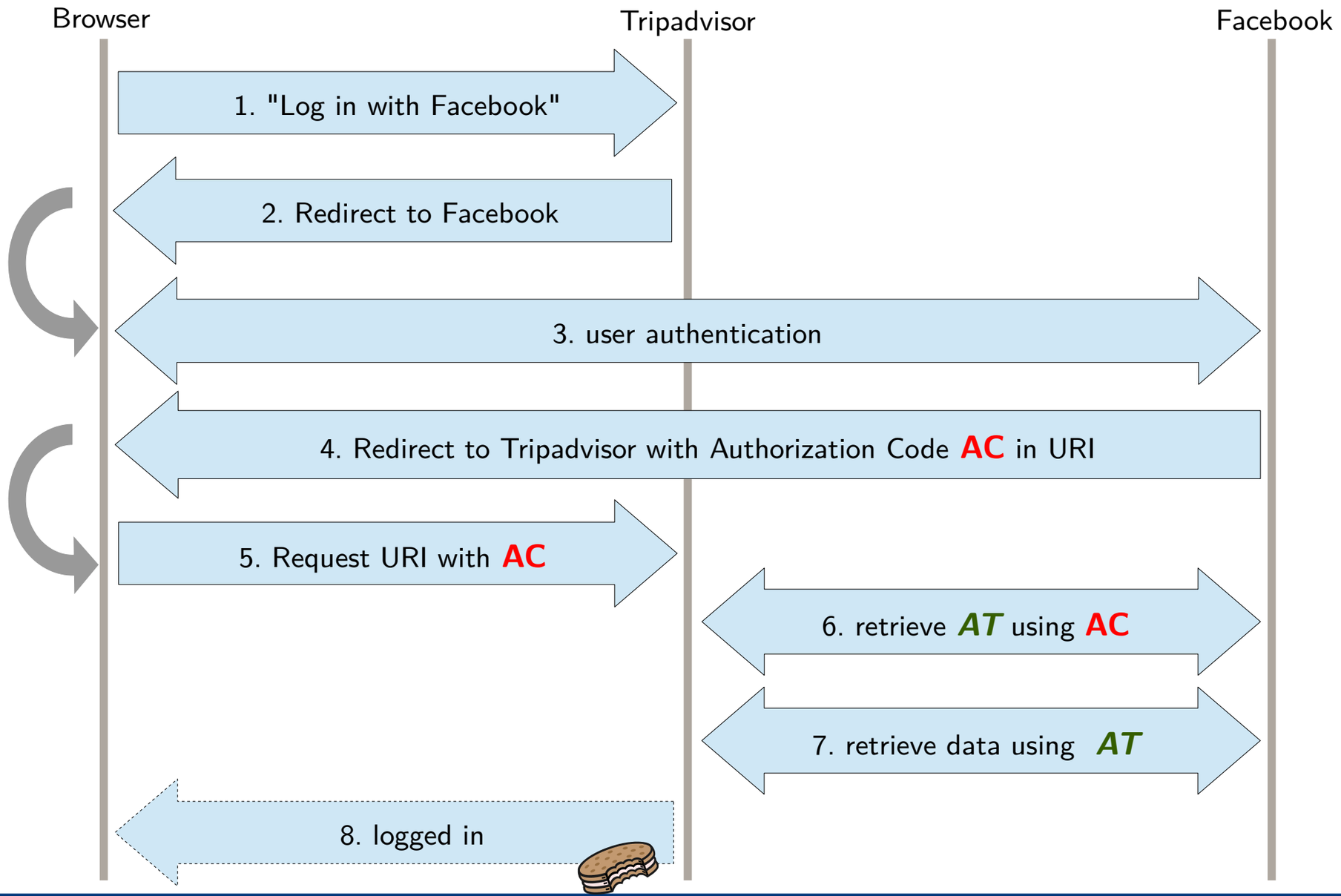






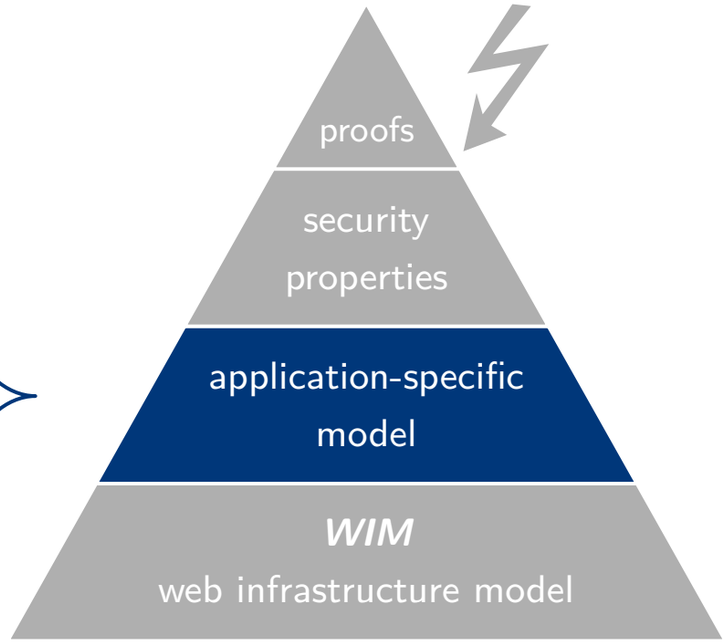


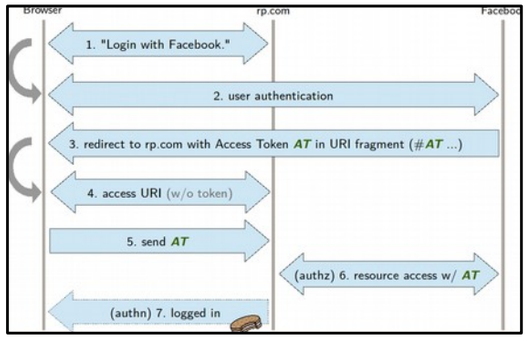
OAuth 2.0

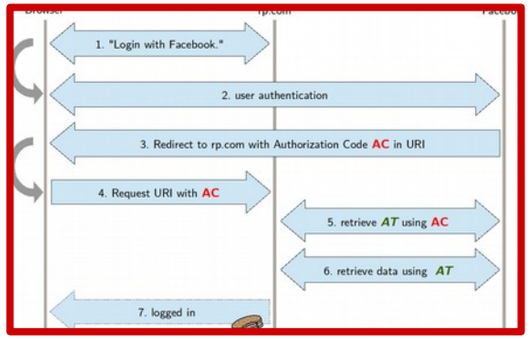

Browser
Tripadvisor
Facebook
1. "Log in with Facebook"
2. Redirect to Facebook
3. user authentication
4. Redirect to Tripadvisor with Authorization Code AC in URI
5. Request URI with AC
6. retrieve AT using AC
7. retrieve data using AT
8. logged in

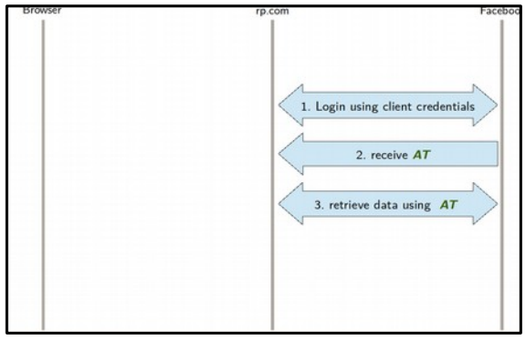
OAuth 2.0

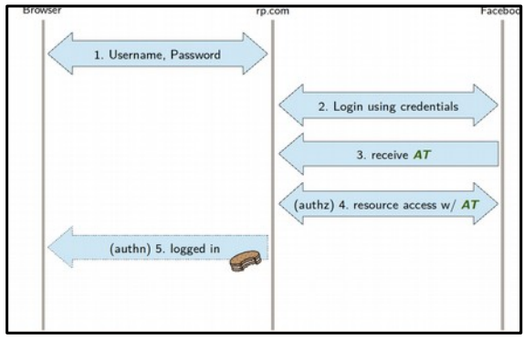
- SSO framework used for authorization/authentication
- Specified by IETF (RFC6749), very widely used
  (e.g., Log in With Facebook)
- Many "variables":
  optional parameters, *public* and *confidential* clients, etc.
- Four different modes of interaction (*grants*)



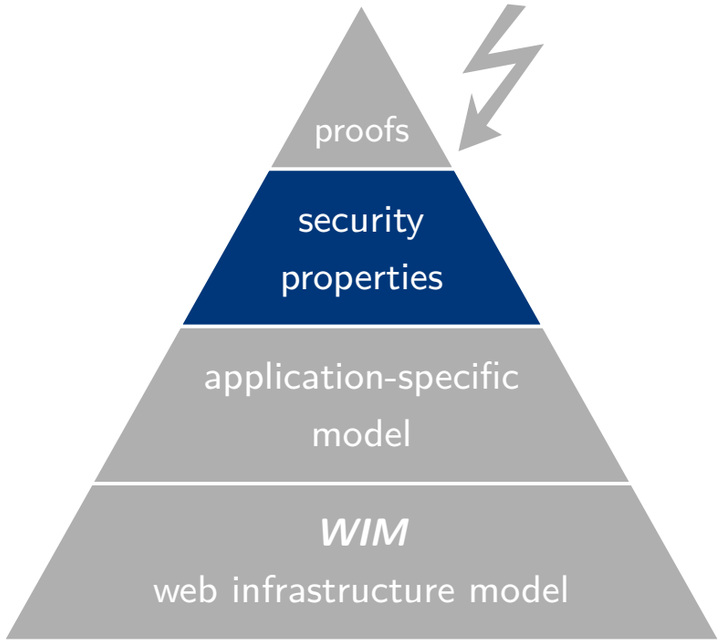








OAuth 2.0: Security Properties



- Authentication

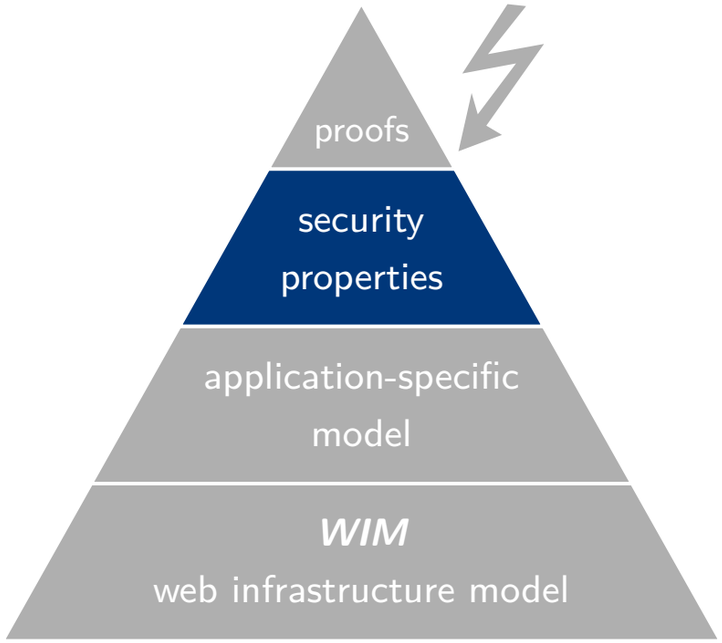
Definition 56 (Authentication Property). Let $\mathcal{OAuthWS}^n$ be an OAuth web system with a network attacker. We say that $\mathcal{OAuthWS}^n$ *is secure w.r.t. authentication* iff for every run ρ of $\mathcal{OAuthWS}^n$, every state (S^j, E^j, N^j) in ρ, every $r \in \mathsf{Clients}$ that is honest in S^j, every $i \in \mathsf{OAP}$, every $g \in \mathsf{dom}(i)$, every $u \in \mathbb{S}$, every client service token of the form $\langle n, \langle u, g \rangle\rangle$ recorded in $S^j(r).\texttt{serviceTokens}$, 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 `FULLCORRUPT`), some $r' \in \mathsf{trustedClients}(\mathsf{secretOfID}(\langle u, g \rangle))$ is corrupted in S^j, or i is corrupted in S^j.

- Authorization

Definition 55 (Authorization 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. authorization* iff for every run ρ of $\mathcal{OAuthWS}^n$, every state (S^j, E^j, N^j) in ρ, every OAP $i \in \mathsf{OAP}$, every $r \in \mathsf{Clients} \cup \{\bot\}$ with r being honest in S^j unless $r = \bot$, every $u \in \mathsf{ID} \cup \{\bot\}$, for $n = \mathsf{resourceOf}(i, r, u)$, n is derivable from the attackers knowledge in S^j (i.e., $n \in d_\emptyset(S^j(\mathsf{attacker}))$), it follows that

1. i is corrupted in S^j, or
2. $u \neq \bot$ and (i) the browser b owning u is fully corrupted in S^j or (ii) some $r' \in \mathsf{trustedClients}(\mathsf{secretOfID}(u))$ is corrupted in S^j.

OAuth 2.0: Security Properties



- Session Integrity for authentication

Definition 64 (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 $r \in \mathsf{Clients}$ that is honest in Q_{login}, every $i \in \mathsf{OAP}$, 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 B.4) and n is sent to the browser b, then

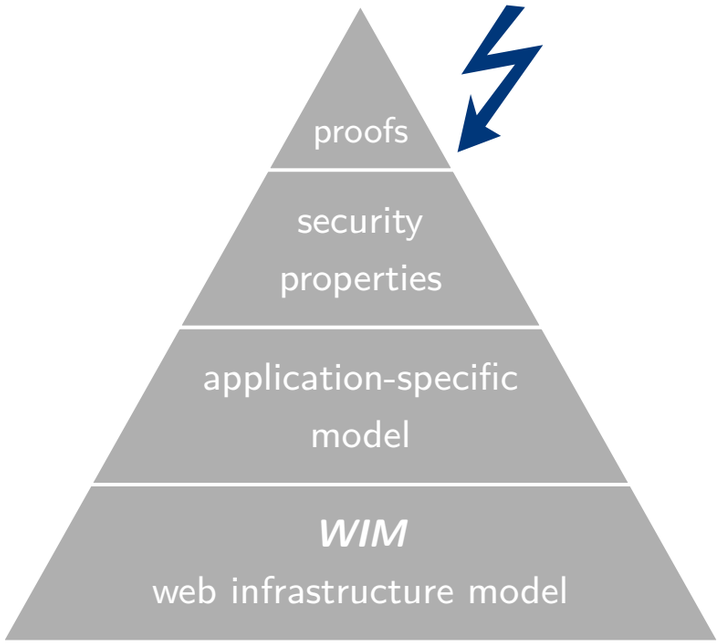
(a) there is an OAuth Session $o \in \mathsf{OASessions}(\rho, b, r, i)$, and

(b) if i is honest in Q_{login} then Q_{login} is in o and we have that

$$\left(\mathsf{selected}_{\text{ia}}(o, b, r, \langle u, g \rangle) \vee \mathsf{selected}_{\text{nia}}(o, b, r, \langle u, g \rangle)\right) \iff \left(\langle u, g \rangle \equiv \langle u', g' \rangle\right) .$$

- Session Integrity for authorization
  (similar to above)

OAuth 2.0: New Attacks

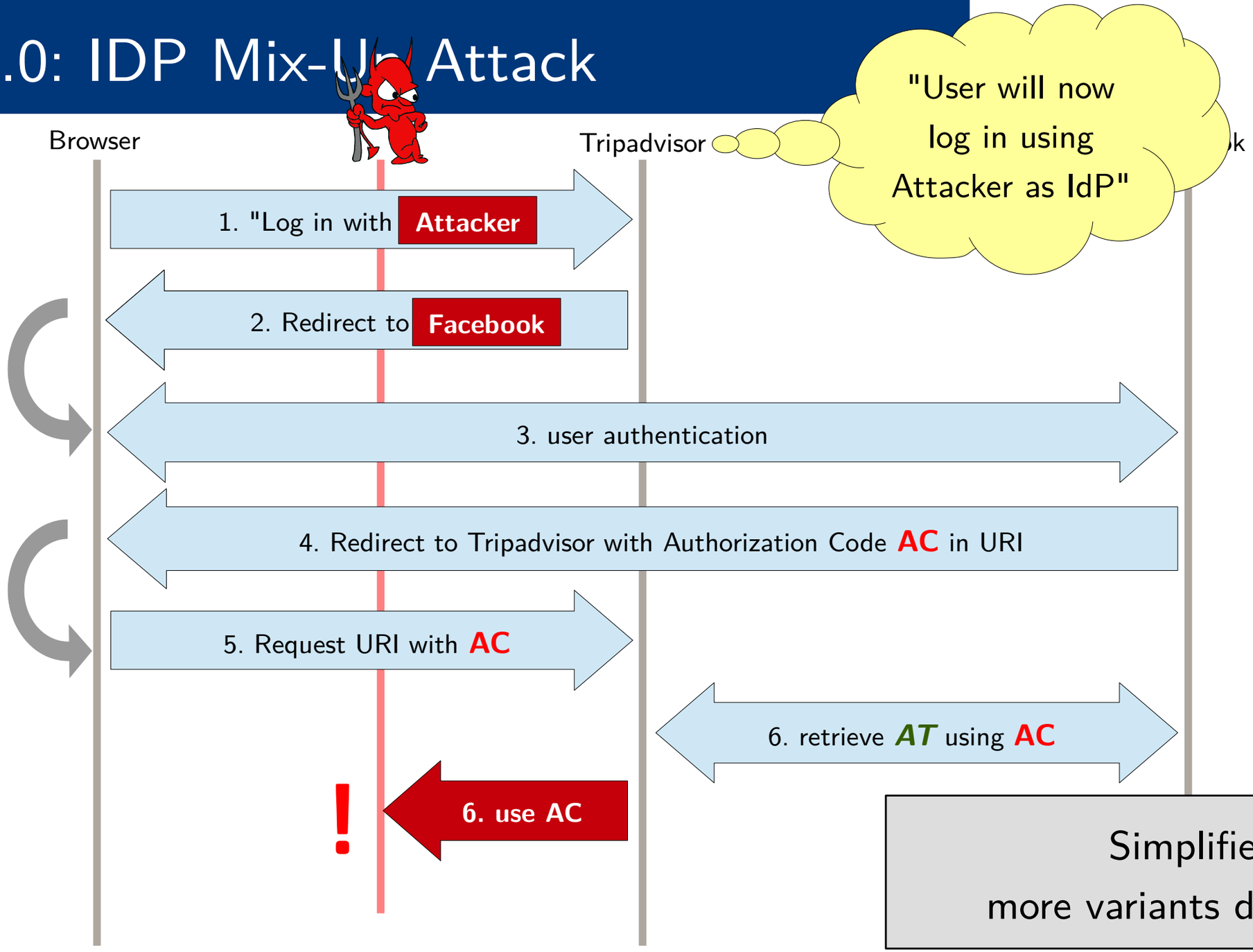


OAuth 2.0 had been analyzed many times before, but not in a comprehensive formal model.

New attacks:

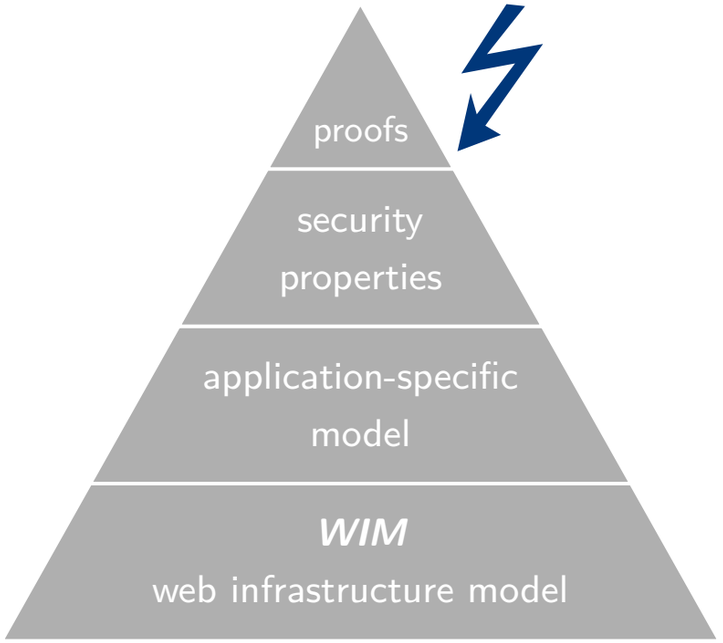
- 307 Redirect Attack
- Identity Provider Mix-Up Attack (new class of attacks)
- State Leak Attack
- Naïve Client Session Integrity Attack
- Across Identity Provider State Reuse Attack

OAuth 2.0: IDP Mix-Up Attack



Simplified,
more variants discovered

OAuth 2.0: New Attacks



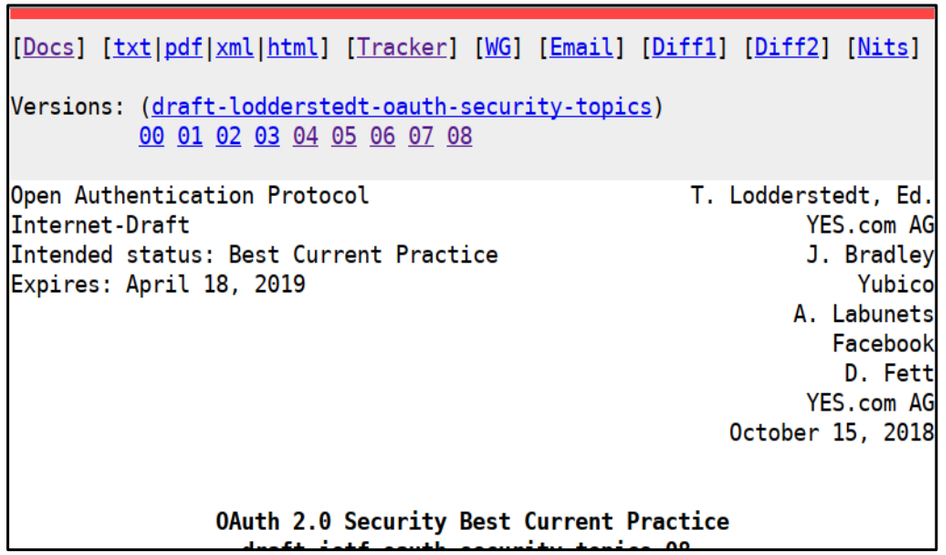
OAuth 2.0 had been analyzed many times before, but not in a comprehensive formal model.

New attacks:

- 307 Redirect Attack
- Identity Provider Mix-Up Attack (new class of attacks)
- State Leak Attack
- Naïve Client Session Integrity Attack
- Across Identity Provider State Reuse Attack

OAuth 2.0: Impact

- Disclosed OAuth attacks to the IETF Web Authorization Working Group in late 2015
- Emergency meeting with the working group four weeks later
- Initiated the OAuth Security Workshop (OSW) to foster the exchange between researchers, standardization groups, and industry
- Joined the working group to codify the fixes into a new RFC:
  OAuth 2.0 Security Best Current Practice
  [draft-ietf-oauth-security-topics]



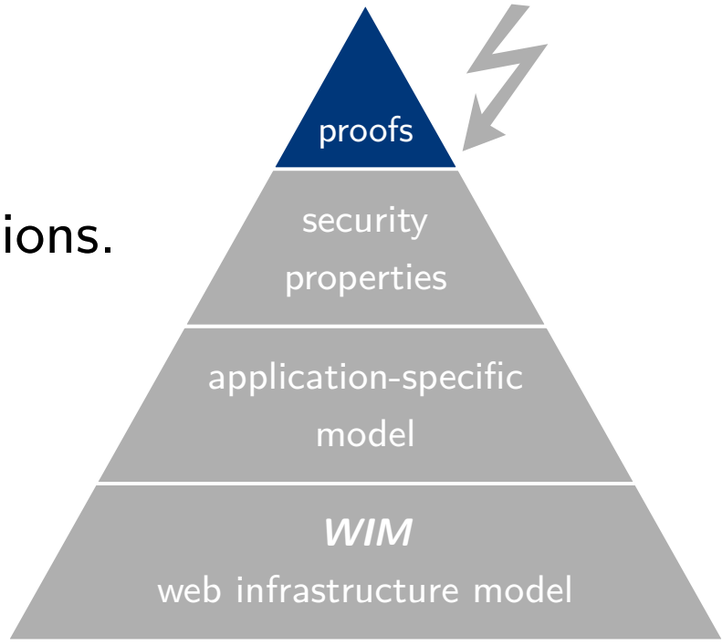
[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-lodderstedt-oauth-security-topics)
00 01 02 03 04 05 06 07 08

Open Authentication Protocol — T. Lodderstedt, Ed.
Internet-Draft — YES.com AG
Intended status: Best Current Practice — J. Bradley
Expires: April 18, 2019 — Yubico
A. Labunets
Facebook
D. Fett
YES.com AG
October 15, 2018

OAuth 2.0 Security Best Current Practice

OAuth 2.0: Proof of Security



Proof based on our model of OAuth 2.0 with all grant types and options.

Assumptions:

- Adherence to web best practices
  (e.g., regarding session handling)
- Adoption of our implementation guidelines
  (e.g., no 3rd party scripts on certain web pages)
- Fixes against previously known and new 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.

WIM Case Studies

Mozilla BrowserID

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

SPRESSO

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

OAuth 2.0

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

OpenID Connect

OpenID Connect

- OAuth 2.0 was built for authorization, not authentication
- OpenID Connect: "Identity Layer" for OAuth 2.0 to solve this
- Includes new extensions:
  - Automatic discovery of identity providers
  - Dynamic registration of clients at identity providers

  } Out of scope of plain OAuth 2.0

- New token type ("id token")
- Cryptographic mechanisms, e.g., signed id token

OpenID Connect

Results:

- All newly discovered OAuth attacks apply to OpenID Connect as well
- Implementation guidelines to avoid known attacks
- Proof of security (authentication, authorization, session integrity) including discovery and dynamic registration extensions

Theorem 2 (Security of OpenID Connect). Let $\mathcal{OIDCWS}^n$ be an OIDC web system with a network attacker. Then, $\mathcal{OIDCWS}^n$ is secure w.r.t. authentication and authorization. Let $\mathcal{OIDCWS}^w$ be an OIDC web system with web attackers. Then, $\mathcal{OIDCWS}^w$ is secure w.r.t. session integrity for authentication and authorization.

WIM Case Studies

Mozilla BrowserID

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

SPRESSO

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

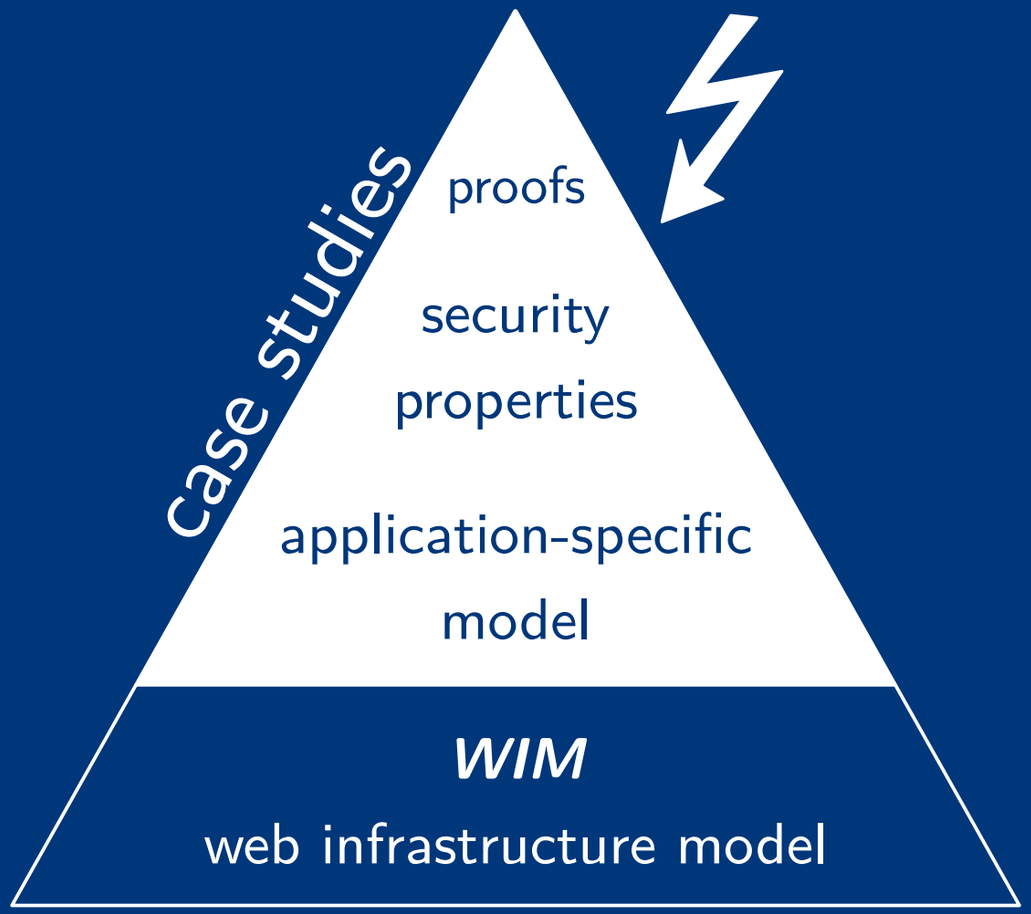
OAuth 2.0

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

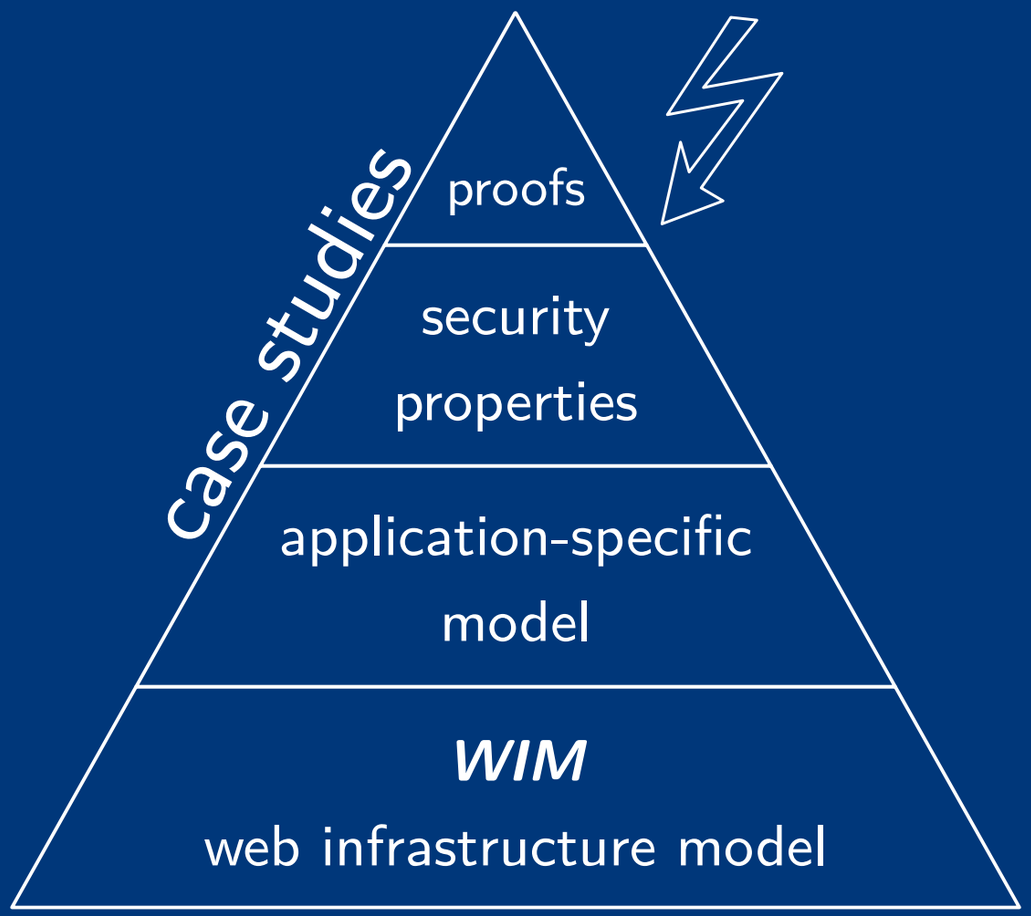
OpenID Connect

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

An Expressive Formal Model of the Web Infrastructure



An Expressive Formal Model of the Web Infrastructure



- Most detailed and comprehensive formal model of the web infrastructure
- Case studies (OAuth, OpenID Connect) with real-world impact
- Found new classes of attacks
- Formal proofs of web security with unprecedented level of detail

Thank you!