

University of Stuttgart  
Institute of  
Information Security

Thesis Defense

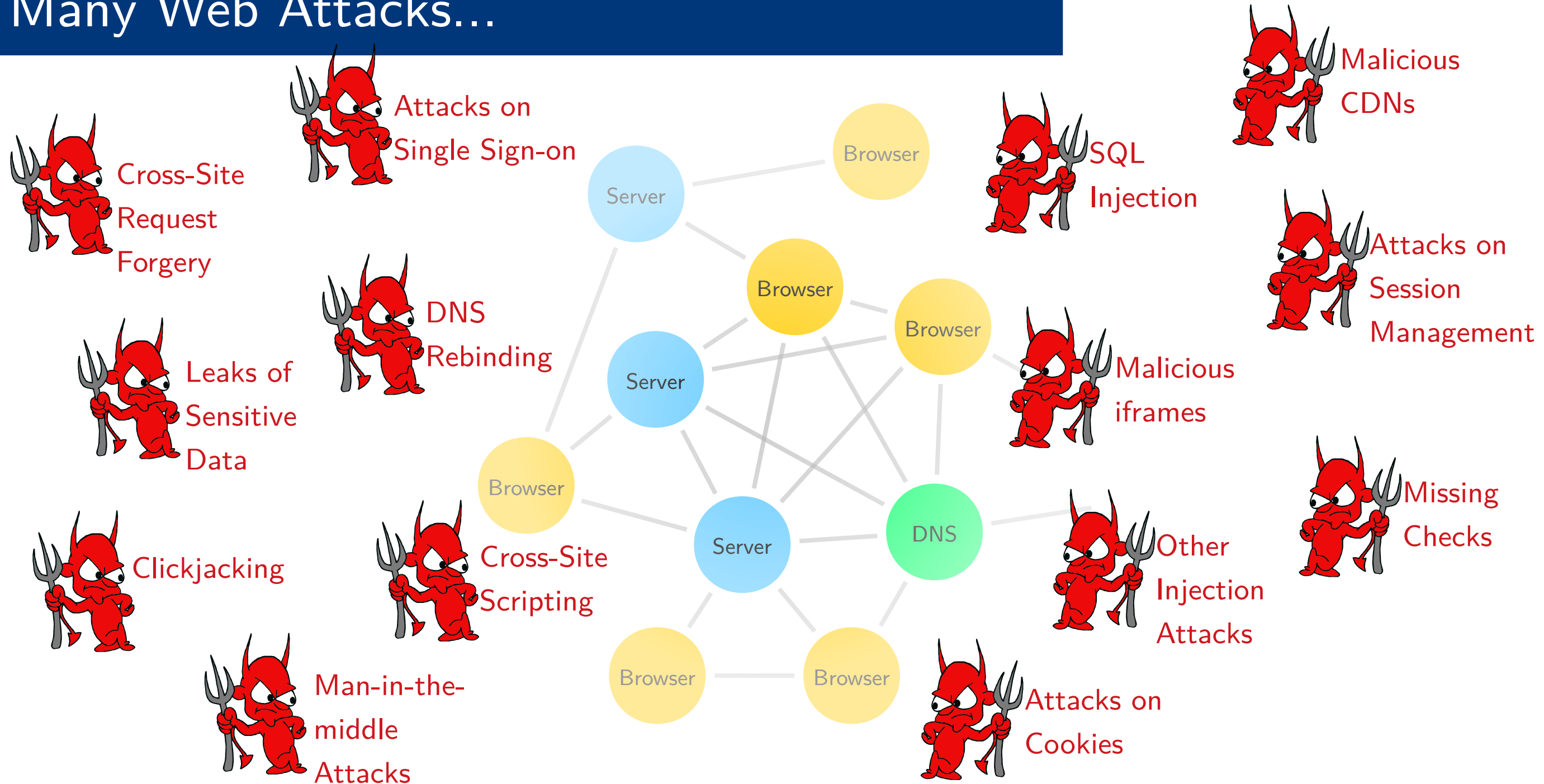
# An Expressive Formal Model of the Web Infrastructure

Dipl.-Inf. Daniel Fett

2018-10-19



# Many Web Attacks...



# ...but why?

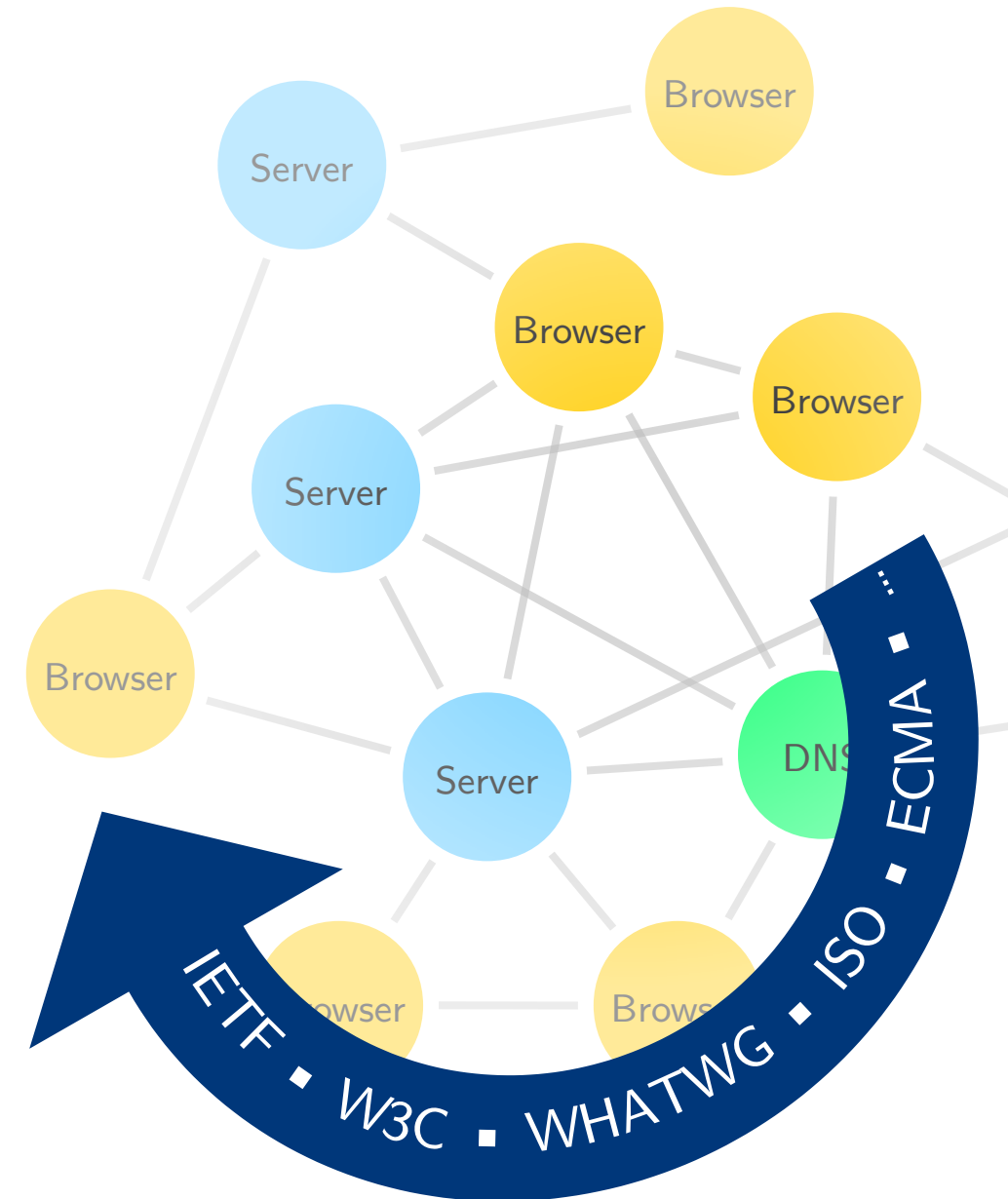
The web is complex ...

- ▶ 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**

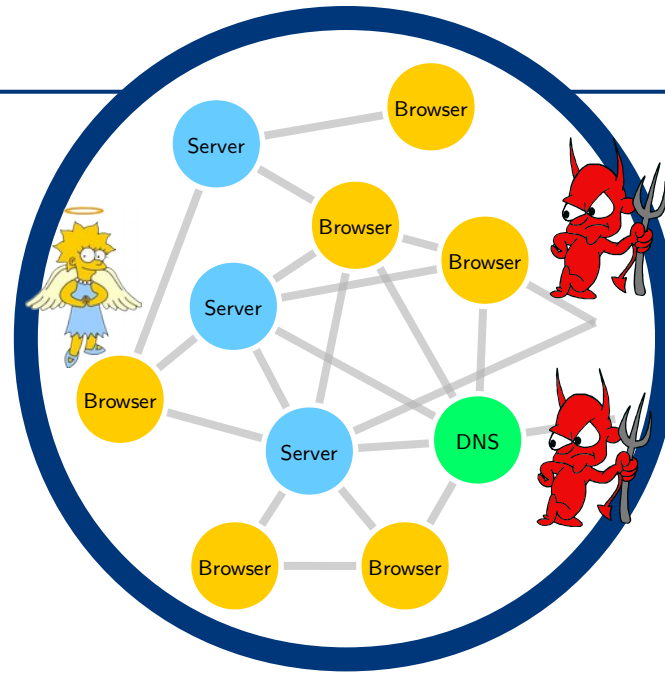
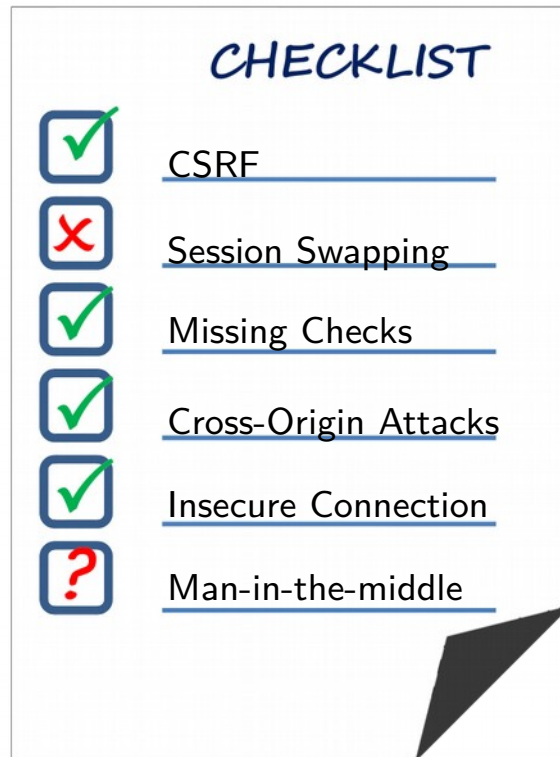
**Finding vulnerabilities  
is hard!**



# Current Methods

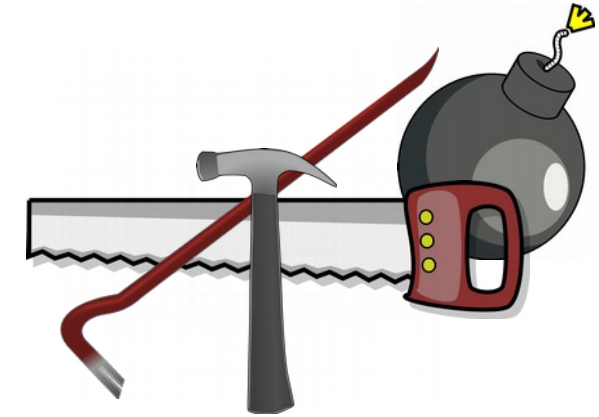
## Expert review

of standards and  
implementations



## Penetration testing

using tools or manual  
analysis

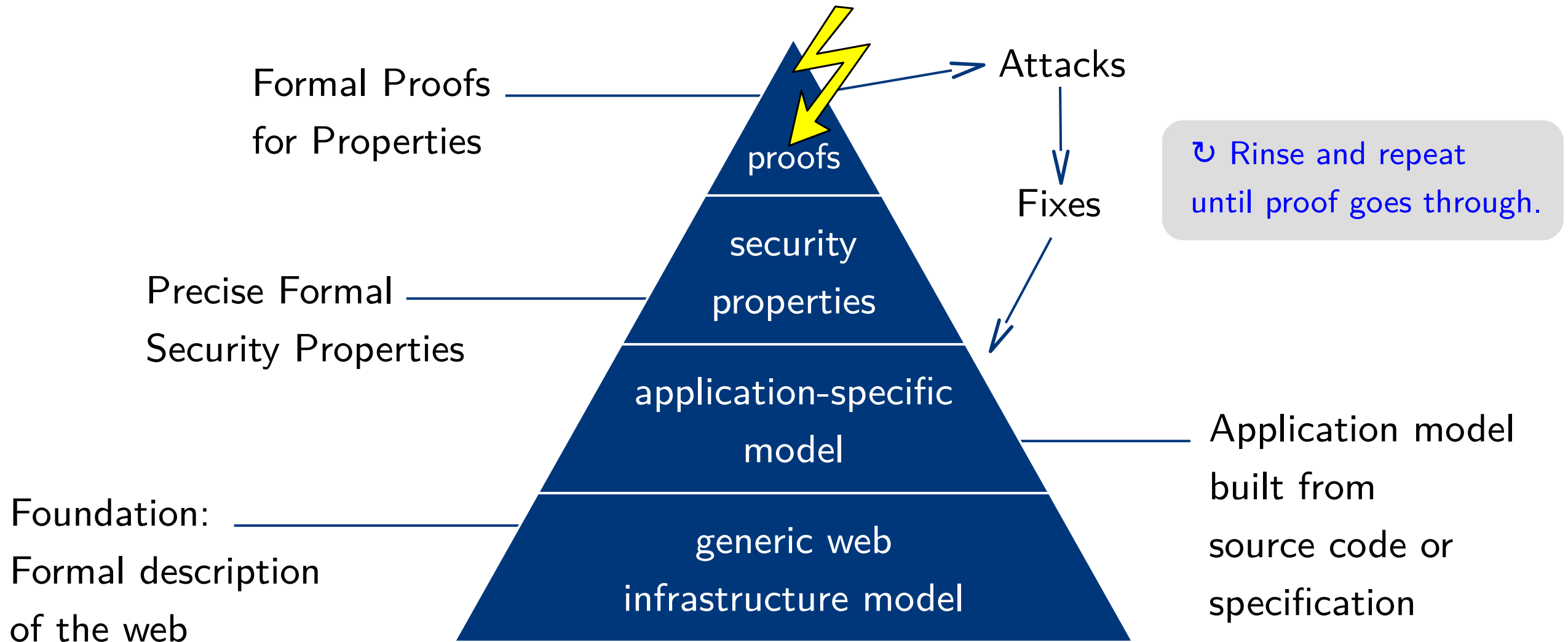


# Downsides

- ▶ 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?**

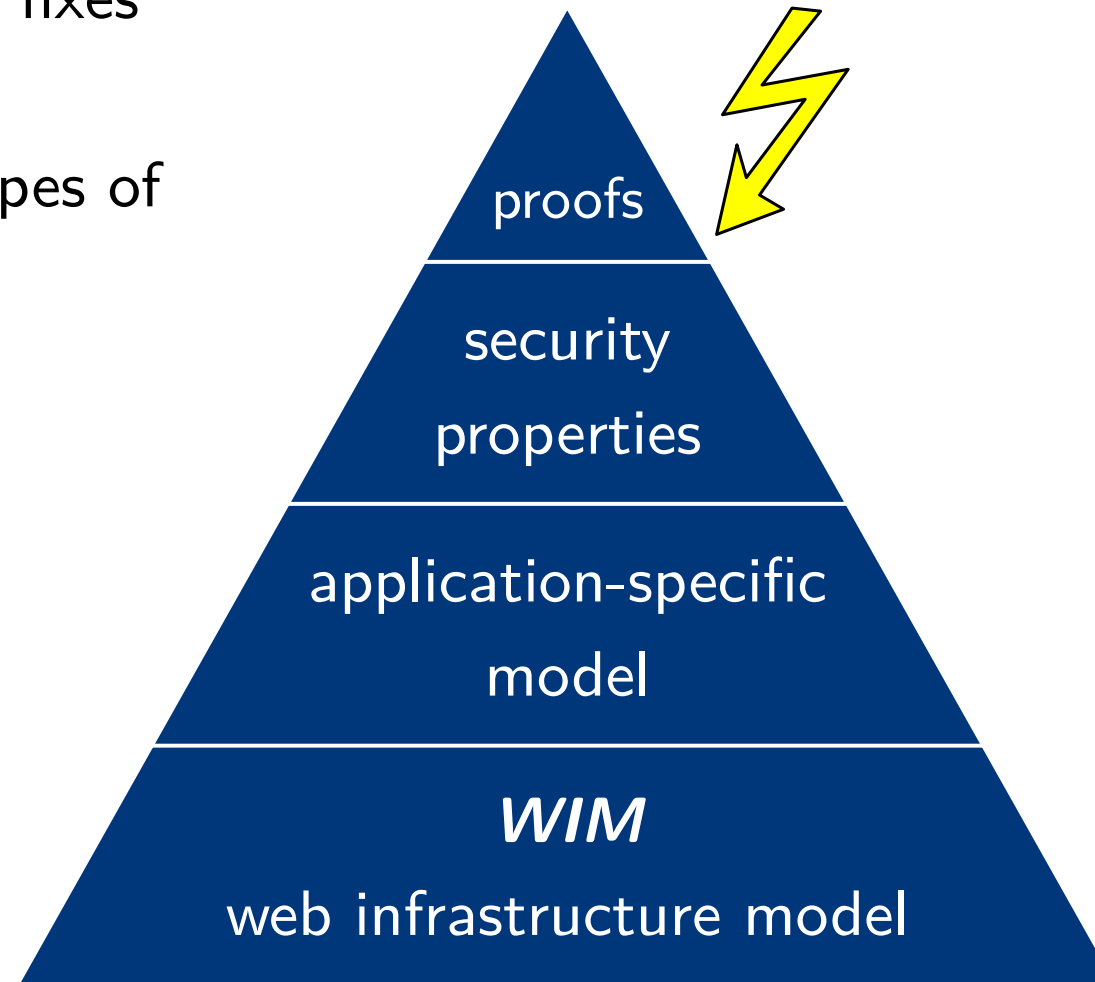
# Model-based Approach



# Advantages

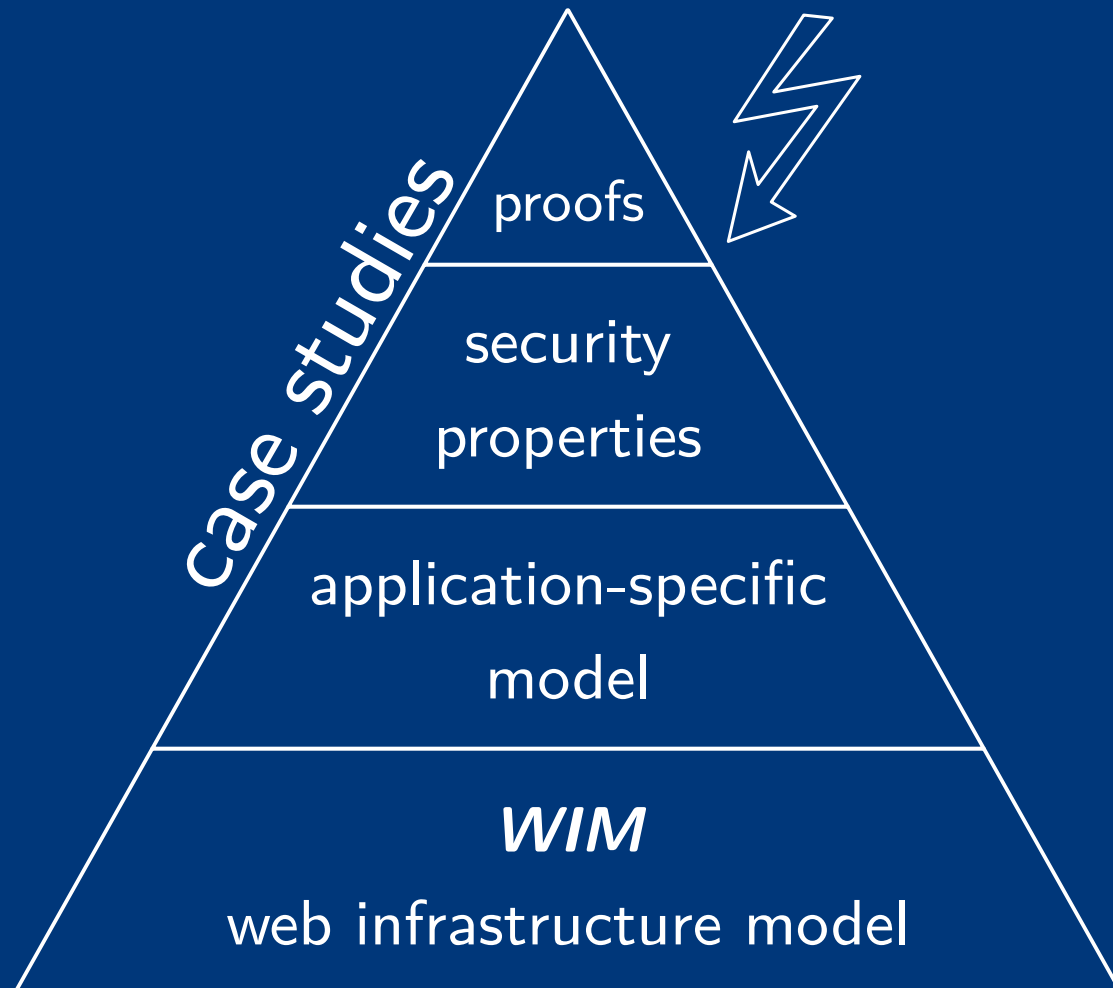
This approach can yield...

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



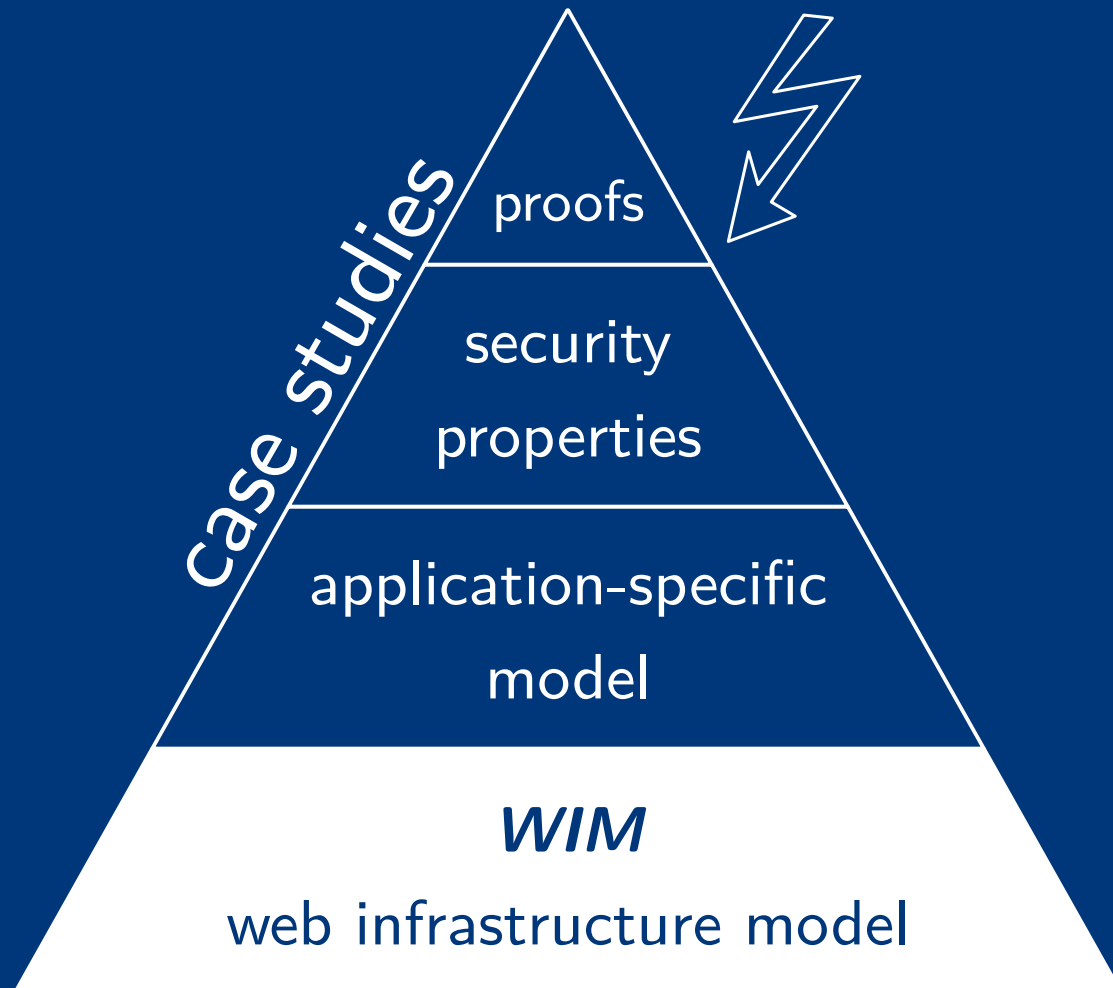


# An Expressive Formal Model of the Web Infrastructure





# 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

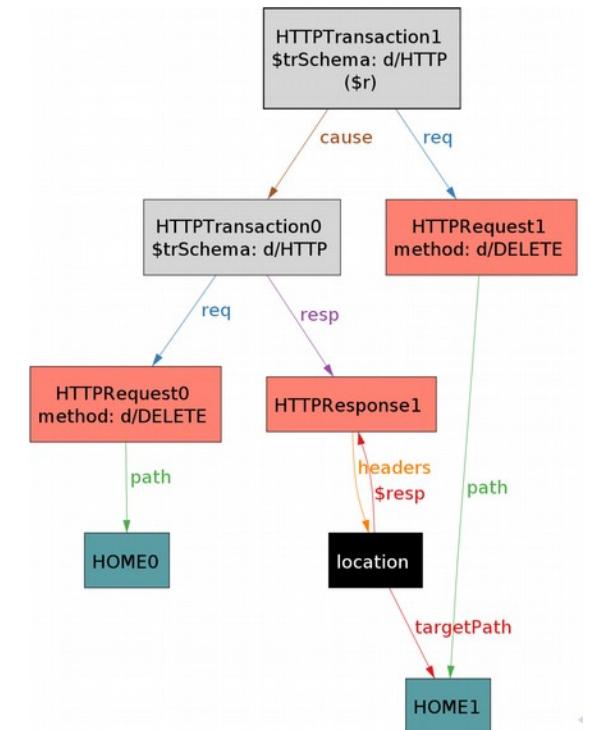
Focus on [automated, tool-based](#) analysis.

[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

## A. The Web Infrastructure Model

In this appendix, we present the Web Infrastructure Model as proposed in [PKS14] and extended in [PKS15a; PKS15b; PKS16; PKS17]. 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

The signature  $\Sigma$  for the terms and messages considered in this work is the union of the following pairwise disjoint sets of function symbols:

- constants  $C = \text{IPs} \cup \mathbb{S} \cup \{\top, \perp, \emptyset\}$  where the three sets are pairwise disjoint,  $\mathbb{S}$  is interpreted to be the set of ASCII strings (including the empty string  $\epsilon$ ), and  $\text{IPs}$  is interpreted to be a set of (IP) addresses,
- function symbols for public keys, (asymmetric) encryption/decryption, and signatures:  $\text{pub}(\cdot)$ ,  $\text{enc}_k(\cdot)$ ,  $\text{dec}_k(\cdot)$ ,  $\text{enc}_s(\cdot)$ ,  $\text{dec}_s(\cdot)$ ,  $\text{sig}(\cdot)$ ,  $\text{checksig}(\cdot, \cdot)$ , and  $\text{extractmsg}(\cdot)$ ,
- $n$ -ary sequences  $(\cdot)$ ,  $(\cdot, \cdot)$ ,  $(\cdot, \cdot, \cdot)$ , etc., and
- projection symbols  $\pi_i(\cdot)$  for all  $i \in \mathbb{N}$ .

For strings (elements in  $\mathbb{S}$ ), we use a specific font. For example, `HTTPReq` and `HTTPResp` are strings. We denote by  $\text{Doms} \subseteq \mathbb{S}$  the set of domains, e.g., `example.com`  $\in \text{Doms}$ . We denote by  $\text{Methods} \subseteq \mathbb{S}$  the set of methods used in HTTP requests, e.g., `GET`, `POST`  $\in \text{Methods}$ .

**Definition 6 (Nonces and Terms).** By  $X = \{x_1, x_2, \dots\}$  we denote a set of variables and by  $\mathcal{N}$  we denote an infinite set of constants (nonces) such that  $\Sigma$ ,  $X$ , and  $\mathcal{N}$  are pairwise disjoint. For  $N \subseteq \mathcal{N}$  we define the set  $\mathcal{T}_N(X)$  of terms over  $\Sigma \cup N \cup X$  inductively: (1) If  $t \in N \cup X$ , then  $t$  is a term. (2) If  $f \in \Sigma$  is an  $n$ -ary function symbol in  $\Sigma$  for some  $n \geq 0$  and  $t_1, \dots, t_n$  are terms, then  $f(t_1, \dots, t_n)$  is a term.

The equational theory associated with the signature  $\Sigma$  is given in Figure A.1. By  $\equiv$  we denote the congruence relation on  $\mathcal{T}_N(X)$  induced by this theory. For example, we have that  $\pi_1(\text{dec}_s(\text{enc}_s(a, b), \text{pub}(k)), k) \equiv a$ .

**Algorithm A.11** Web Browser Model: Main Algorithm.

```
Input:  $(n, f, m), s$ 
1: let  $s' := s$ 
2: if  $s$  is corrupted  $\& \neq \perp$  then
3:   let  $s'$ .pendingRequests :=  $(n, s$ .pendingRequests)  $\rightarrow$  Collect incoming messages
4:   let  $n := \mathbb{N}$ 
5:   let  $m'_1, \dots, m'_n \leftarrow d_i(s')$   $\rightarrow$  Create  $n$  new messages nondeterministically.
6:   let  $m'_1, \dots, m'_n \leftarrow \text{IPs}$ 
7:   stop  $\{(m'_1, n, m'_1), \dots, (m'_n, n, m'_n)\}, s'$ 
```

```
8: if  $m \equiv \text{TRIGGER}$  then
  let  $\text{switch} := \{\text{script}, \text{urlbar}, \text{reload}, \text{forward}, \text{back}\}$ 
  let  $\overline{w} \leftarrow \text{Subwindows}(s')$  such that  $s'.$ documents  $\neq ()$ 
   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some window.
  let  $\overline{w} \leftarrow ()$  such that  $s'.$ documents  $\neq ()$ 
   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some top-level window.
  if  $\text{switch} \equiv \text{script}$  then  $\rightarrow$  Run some script.
    let  $\overline{w} := \overline{w} \cup \{\text{activeDocument}\}$ 
    call RUNSCRIPT( $\overline{w}$ ,  $s'$ )
  else if  $\text{switch} \equiv \text{urlbar}$  then  $\rightarrow$  Create a new request.
    let  $n := \mathbb{N}$ 
    if  $n := \mathbb{N}$  then  $\rightarrow$  Create a new window.
      let  $\text{windowName} := n_1$ 
      let  $s' := (\text{windowName}, \emptyset, \perp)$ 
      let  $s'$ .windows :=  $s'$ .windows  $\cup \{s'\}$ 
      let  $\text{windowName} := s'$ .TheName
    else  $\rightarrow$  Use the existing top-level window.
      let  $\text{windowName} := s'$ .TheName
    let  $\text{protocol} := (P, \emptyset)$ 
    let  $\text{host} := \text{Doms}$ 
    let  $\text{path} := \mathbb{S}$ 
    let  $\text{fragment} := \mathbb{S}$ 
    let  $\text{parameters} := \mathbb{S} \times \mathbb{S}$ 
    let  $\text{url} := (\text{URL}, \text{protocol}, \text{host}, \text{path}, \text{parameters}, \text{fragment})$ 
    let  $\text{req} := (\text{HTTPReq}, n_2, \text{GET}, \text{host}, \text{path}, \text{parameters}, \emptyset, \emptyset)$ 
    call HTTP.SEND( $\text{REQ}$ ,  $\text{windowName}$ ,  $\text{req}$ ,  $\text{url}$ ,  $\perp, \perp, \perp, s'$ )
  else if  $\text{switch} \equiv \text{reload}$  then  $\rightarrow$  Reload some document.
    let  $\text{url} := s'.$ activeDocument.location
    let  $\text{req} := (\text{HTTPReq}, n_2, \text{GET}, \text{url}, \text{host}, \text{url}, \text{url}, \text{parameters}, \emptyset, \emptyset)$ 
    let  $\text{referrer} := s'.$ activeDocument.referrer
    let  $s' := \text{CANCELNAV}(s'.$ nonce,  $s'$ )
    call HTTP.SEND( $\text{REQ}$ ,  $s'.$ name,  $\text{req}$ ,  $\text{url}$ ,  $\perp, \text{referrer}$ ,  $\perp, s'$ )
  else if  $\text{switch} \equiv \text{forward}$  then
```

$\text{Subwindows}(s')$  such that  $s'.$ name  $\neq s'$ .p  
activeDocument.origin  $\neq s'.$ activeDocument.origin ( $\overline{w}$  is not a top-level window) there is an ancestor window  $\overline{w}'$  with an active document that has the same name as the active document in  $\overline{w}$ , then  $\overline{w}' \in \overline{w}$ , and  
 $\text{Subwindows}(s')$  such that  $s'.$ name  $\neq s'$ .p.name  $\wedge \overline{w} \in \overline{w}'$  ( $\overline{w}'$  is a top-level window) it has an opener—and  $\overline{w}$  is allowed to navigate the opener window of  $\overline{w}'$ ,  $\overline{w}' \in \overline{w}$ .

In the definition of the following functions, we use  $n$ ,  $f$ ,  $m$ , and  $s$  as read-only global input and other variables are local variables or arguments.

Places throughout the algorithms we use placeholders to generate “fresh” nonces. Figure A.3 shows a list of all placeholders used.

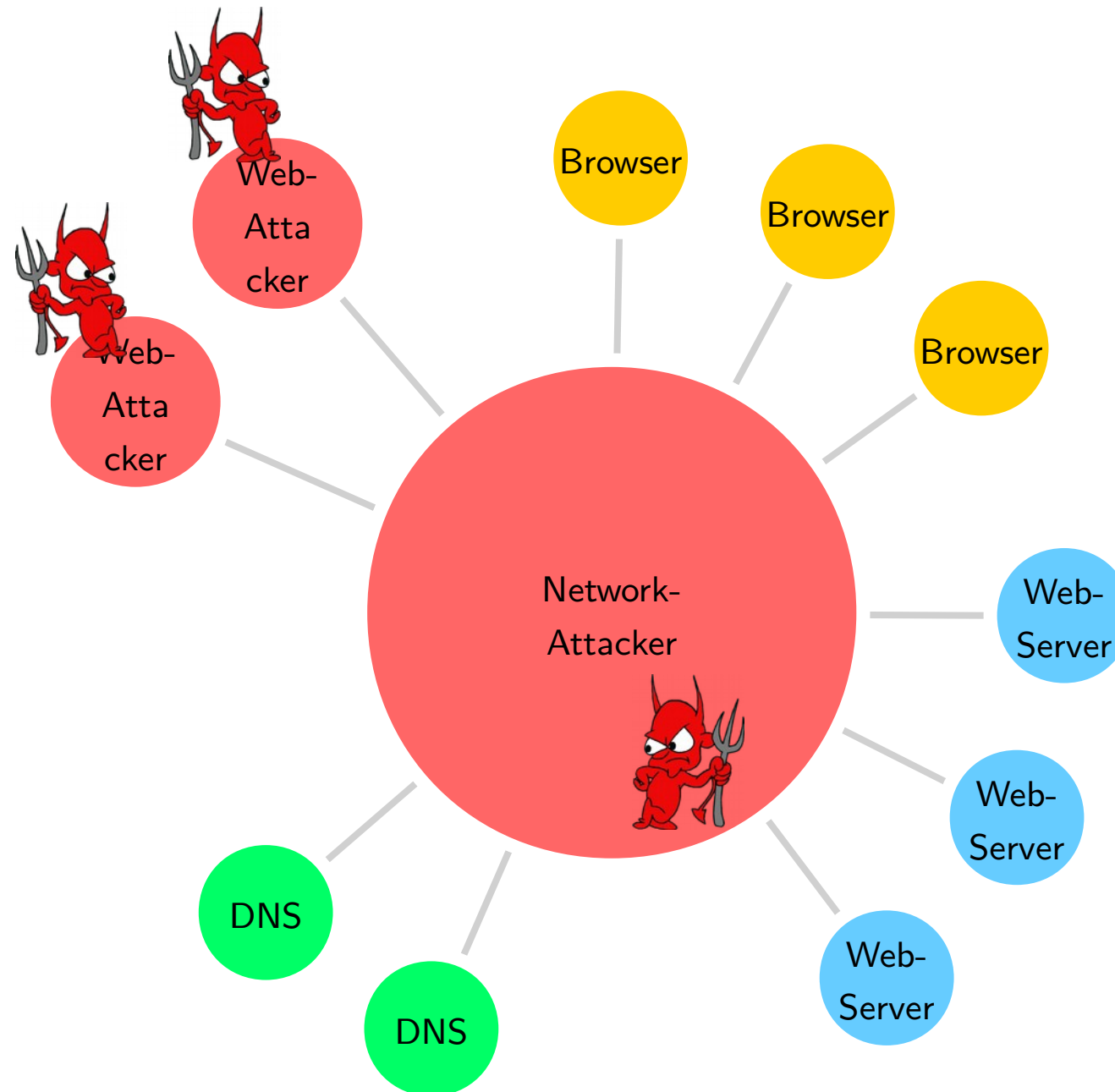
Placeholder	Usage
$\text{nonce}_{\text{window}}$	Algorithm A.11: window nonces
$\text{nonce}_{\text{httpreq}}$	Algorithm A.11: HTTP request nonce
$\text{nonce}_{\text{httpresp}}$	Algorithm A.11: lookup key for pending HTTP requests entry
$\text{nonce}_{\text{httpreq}}$	Algorithm A.8: HTTP request nonce (multiple lines)
$\text{nonce}_{\text{httpreq}}$	Algorithm A.8: subwindow nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.10: HTTP request nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.10: document nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.4: lookup key for pending DNS entry
$\text{nonce}_{\text{httpreq}}$	Algorithm A.1: window nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.7: window nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.7: HTTP request nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.8: WebRTC connection nonce
$\text{nonce}_{\text{httpreq}}$	Algorithm A.8: replacement for placeholders in script output

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

The function `GETNAVIGABLEWINDOW` (Algorithm A.1) is called by the browser to determine the window that is *actually* navigated when a script in the window  $s'.$ name provides a window reference for navigation (e.g., for opening a link). When it is given a window reference (nonce)  $\text{window}$ , this function returns a pointer to a selected window term in  $s'$ .

- If  $\text{window}$  is the string `BLANK`, a new window is created and a pointer to that window is returned.
- If  $\text{window}$  is a nonce (reference) and there is a window term with a reference of that value in the windows in  $s'$ , a pointer  $\overline{w}'$  to that window term is returned, as long as the window is navigable by the current window's document (as defined by `NavigableWindow` above).

# 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

## A. The Web Infrastructure Model

In this appendix, we present the Web Infrastructure Model as proposed in [PKS14] and extended in [PKS15a; PKS15b; PKS16; PKS17]. 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

The signature  $\Sigma$  for the terms and messages considered in this work is the union of the following pairwise disjoint sets of function symbols:

- constants  $C = \text{IPs} \cup \mathbb{S} \cup \{\top, \perp, \emptyset\}$  where the three sets are pairwise disjoint,  $\mathbb{S}$  is interpreted to be the set of ASCII strings (including the empty string  $\epsilon$ ), and  $\text{IPs}$  is interpreted to be a set of (IP) addresses,
- function symbols for public keys, (asymmetric) encryption/decryption, and signatures:  $\text{pub}(\cdot)$ ,  $\text{enc}_k(\cdot)$ ,  $\text{dec}_k(\cdot)$ ,  $\text{enc}_s(\cdot)$ ,  $\text{dec}_s(\cdot)$ ,  $\text{sig}(\cdot)$ ,  $\text{checksig}(\cdot, \cdot)$ , and  $\text{extractmsg}(\cdot)$ ,
- $n$ -ary sequences  $(\cdot)$ ,  $(\cdot)$ ,  $(\cdot, \cdot)$ ,  $(\cdot, \cdot, \cdot)$ , etc., and
- projection symbols  $\pi_i(\cdot)$  for all  $i \in \mathbb{N}$ .

For strings (elements in  $\mathbb{S}$ ), we use a specific font. For example, `HTTPReq` and `HTTPResp` are strings. We denote by  $\text{Doms} \subseteq \mathbb{S}$  the set of domains, e.g., `example.com`  $\in \text{Doms}$ . We denote by  $\text{Methods} \subseteq \mathbb{S}$  the set of methods used in HTTP requests, e.g., `GET`, `POST`  $\in \text{Methods}$ .

**Definition 6 (Nonces and Terms).** By  $X = \{x_1, x_2, \dots\}$  we denote a set of variables and by  $\mathcal{N}$  we denote an infinite set of constants (nonces) such that  $\Sigma$ ,  $X$ , and  $\mathcal{N}$  are pairwise disjoint. For  $N \subseteq \mathcal{N}$  we define the set  $\mathcal{T}_N(X)$  of terms over  $\Sigma \cup N \cup X$  inductively: (1) If  $t \in N \cup X$ , then  $t$  is a term. (2) If  $f \in \Sigma$  is an  $n$ -ary function symbol in  $\Sigma$  for some  $n \geq 0$  and  $t_1, \dots, t_n$  are terms, then  $f(t_1, \dots, t_n)$  is a term.

The equational theory associated with the signature  $\Sigma$  is given in Figure A.1. By  $\equiv$  we denote the congruence relation on  $\mathcal{T}_N(X)$  induced by this theory. For example, we have that  $\pi_1(\text{dec}_s(\text{enc}_s(a, b), \text{pub}(k)), k) \equiv a$ .

**Algorithm A.11** Web Browser Model: Main Algorithm.

```
Input:  $(n, f, m), s$ 
1: let  $s' := s$ 
2: if  $s$  is corrupted  $\neq \perp$  then
3:   let  $s'$  pending requests  $\leftarrow (n, s$  pending requests)  $\rightarrow$  Collect incoming messages
4:   let  $n := \mathbb{N}$ 
5:   let  $m'_1, \dots, m'_n \leftarrow d_i(s')$   $\rightarrow$  Create  $n$  new messages nondeterministically.
6:   let  $m'_1, \dots, m'_n \leftarrow \text{IPs}$ 
7:   stop  $\{(m'_1, n, m'_1), \dots, (m'_n, n, m'_n)\}, s'$ 
```

```
8: if  $m \equiv \text{TRIGGER}$  then
  let  $\text{switch} \leftarrow \{\text{script}, \text{urlbar}, \text{reload}, \text{forward}, \text{back}\}$ 
  let  $\overline{w} \leftarrow \text{Subwindow}(s')$  such that  $s' \cdot \overline{w}$  documents  $\neq \emptyset$ 
   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some window.
  let  $\overline{w} \leftarrow \mathbb{S}$  such that  $s' \cdot \overline{w}$  documents  $\neq \emptyset$ 
   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some top-level window.
  if  $\text{switch} \equiv \text{script}$  then  $\rightarrow$  Run some script.
    let  $\mathcal{D} \leftarrow \mathbb{S} \cup \text{active document}$ 
    call RUNSCRIPT( $\overline{w}, \mathcal{D}, s'$ )
  else if  $\text{switch} \equiv \text{urlbar}$  then  $\rightarrow$  Create some new request.
    let  $n := \mathbb{N}$ 
    if  $n := \top$  then  $\rightarrow$  Create a new window.
      let  $\text{windowname} := \pi_1$ 
      let  $s' := (\text{windowname}, \emptyset, \perp)$ 
      let  $s' \cdot \text{window} \leftarrow s' \cdot \text{window} \cup s' \cdot \overline{w}$ 
    else  $\rightarrow$  Use the existing top-level window.
      let  $\text{windowname} := s' \cdot \overline{w}$ 
    let  $\text{protocol} \leftarrow \{P, S\}$ 
    let  $\text{host} \leftarrow \text{Doms}$ 
    let  $\text{path} \leftarrow \mathbb{S}$ 
    let  $\text{fragment} \leftarrow \mathbb{S}$ 
    let  $\text{parameters} \leftarrow \mathbb{S} \times \mathbb{S}$ 
    let  $\text{url} \leftarrow \{\text{URL}, \text{protocol}, \text{host}, \text{path}, \text{parameters}, \text{fragment}\}$ 
    let  $\text{req} \leftarrow \{\text{HTTPReq}, \text{req}, \text{GET}, \text{host}, \text{path}, \text{parameters}, \emptyset, \emptyset\}$ 
    call HTTP.SEND( $\text{REQ}, \text{windowname}, \text{req}, \text{url}, \perp, \perp, \perp, s'$ )
  else if  $\text{switch} \equiv \text{reload}$  then  $\rightarrow$  Reload some document.
    let  $\text{url} \leftarrow s' \cdot \overline{w}$  active document location
    let  $\text{req} \leftarrow \{\text{HTTPReq}, \text{req}, \text{GET}, \text{url}, \text{host}, \text{url}, \text{path}, \text{url}, \text{parameters}, \emptyset, \emptyset\}$ 
    let  $\text{referrer} \leftarrow s' \cdot \overline{w}$  active document referrer
    let  $s' \leftarrow \text{CANCELNAV}(s' \cdot \overline{w}, \text{nonce}, s')$ 
    call HTTP.SEND( $\text{REQ}, s' \cdot \overline{w}, \text{nonce}, \text{req}, \text{url}, \perp, \text{referrer}, \perp, s'$ )
  else if  $\text{switch} \equiv \text{forward}$  then
```

$\text{Subwindow}(s')$  such that  $s' \cdot \overline{w} \cdot \text{document} \rightarrow s' \cdot \overline{w}$  active document origin  $\neq \emptyset$  is not a top-level window there is an ancestor window  $\overline{w}'$  of  $\overline{w}$  with an active document that has the same as the active document in  $\overline{w}$ , then  $\overline{w}' \in \overline{w}$ , and

$\text{Subwindow}(s')$  such that  $s' \cdot \overline{w} \cdot \text{opener} = s' \cdot \overline{w} \cdot \text{nonce} \wedge \overline{w} \in \overline{w}'$  ( $\overline{w}'$  is a top-level window) it has an opener—and  $\overline{w}$  is allowed to navigate the opener window of  $\overline{w}'$ ,  $\overline{w}'$   $\in \overline{w}$ .

In the definition of the following functions, we use  $n$ ,  $f$ ,  $m$ , and  $s$  as read-only global input and other variables are local variables or arguments.

Places throughout the algorithms we use placeholders to generate “fresh” nonces. Figure A.3 shows a list of all placeholders used.

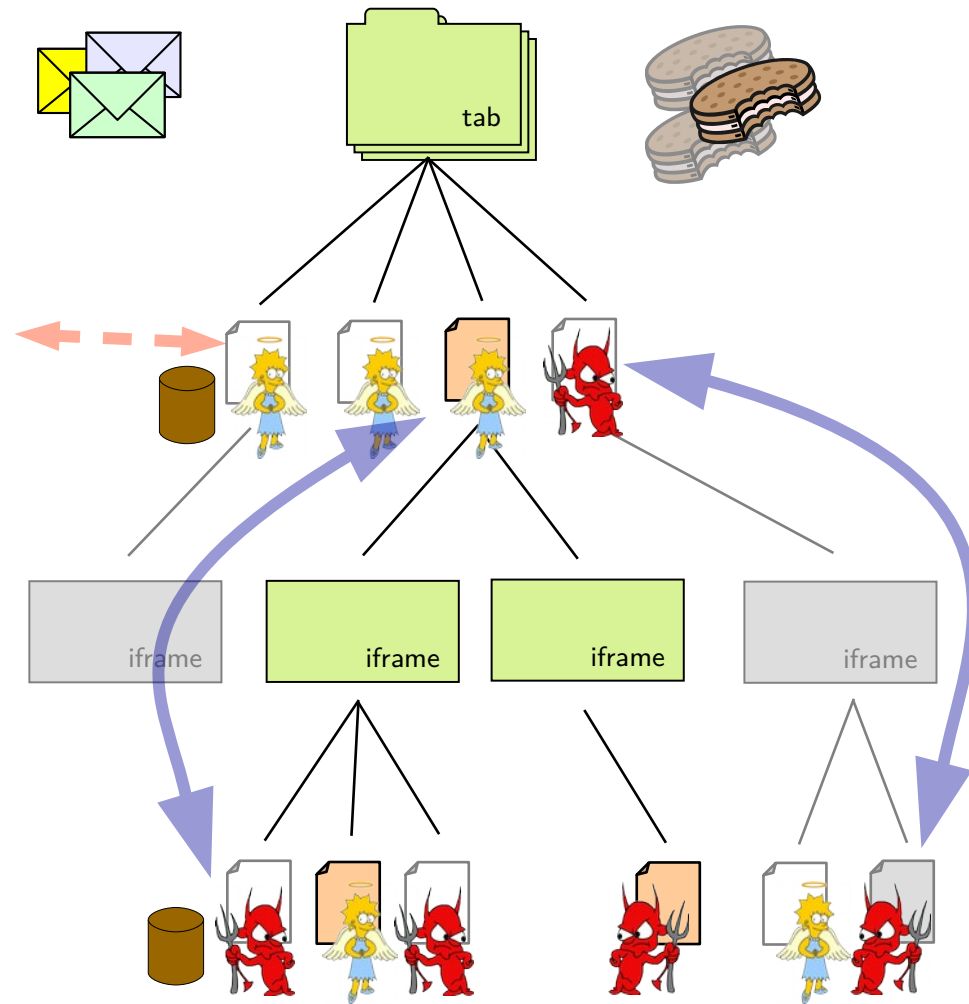
Placeholder	Usage
Algorithm A.11, window nonces	
Algorithm A.11, HTTP request nonce	
Algorithm A.11, lookup key for pending HTTP requests entry	
Algorithm A.8, HTTP request nonce (multiple lines)	
Algorithm A.8, subwindow nonce	
Algorithm A.10, HTTP request nonce	
Algorithm A.10, document nonce	
Algorithm A.4, lookup key for pending DNS entry	
Algorithm A.1, window nonce	
Algorithm A.7, window nonce	
Algorithm A.7, HTTP request nonce	
Algorithm A.8, WebRTC connection nonce	
Algorithm A.8, replacement for placeholders in script output	

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






The function `GETNAVIGABLEWINDOW` (Algorithm A.1) is called by the browser to determine the window that is *actually* navigated when a script in the window  $s' \cdot \overline{w}$  provides a window reference for navigation (e.g., for opening a link). When it is given a window reference (nonce)  $\text{window}$ , this function returns a pointer to a selected window term in  $s'$ :

- If  $\text{window}$  is the string `_BLANK`, a new window is created and a pointer to that window is returned.
- If  $\text{window}$  is a nonce (reference) and there is a window term with a reference of that value in the windows in  $s'$ , a pointer  $\overline{w}'$  to that window term is returned, as long as the window is navigable by the current window's document (as defined by `NavigableWindow` above).

# 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  $\in$  response.headers then
3:     for each  $c \in \langle \rangle$  response.headers[Set-Cookie],  $c \in$  Cookies do
4:       let s'.cookies[request.host]
          $\hookrightarrow :=$  AddCookie(s'.cookies[request.host], c)
5:   if Strict-Transport-Security  $\in$  response.headers  $\wedge$  requestUrl.protocol  $\equiv$  S then
6:     let s'.sts  $:= s'.sts + \langle \rangle$  request.host
7:   if Referer  $\in$  request.headers then
8:     let referrer  $:=$  request.headers[Referer]
9:   else
10:    let referrer  $:= \perp$ 
11:   if Location  $\in$  response.headers  $\wedge$  response.status  $\in \{303, 307\}$  then
12:     let url  $:=$  response.headers[Location]
13:     if url.fragment  $\equiv \perp$  then
14:       let url.fragment  $:=$  requestUrl.fragment
15:     let method'  $:=$  request.method
16:     let body'  $:=$  request.body
17:     if Origin  $\in$  request.headers then
18:       let origin  $:= \langle$  request.headers[Origin],  $\langle$  request.host, url.protocol  $\rangle \rangle$ 
19:     else
20:       let origin  $:= \perp$ 
21:     if response.status  $\equiv 303 \wedge$  request.method  $\notin \{\text{GET}, \text{HEAD}\}$  then
22:       let method'  $:=$  GET
23:       let body'  $:= \langle \rangle$ 
```



# 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 [PKS14] and extended in [PKS15a; PKS15b; PKS16; PKS17]. 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

The signature  $\Sigma$  for the terms and messages considered in this work is the union of the following pairwise disjoint sets of function symbols:

- constants  $C = \text{IPs} \cup \mathbb{S} \cup \{\top, \perp, \emptyset\}$  where the three sets are pairwise disjoint,  $\mathbb{S}$  is interpreted to be the set of ASCII strings (including the empty string  $\epsilon$ ), and  $\text{IPs}$  is interpreted to be a set of (IP) addresses,
- function symbols for public keys, (asymmetric) encryption/decryption, and signatures:  $\text{pub}(\cdot)$ ,  $\text{enc}_k(\cdot)$ ,  $\text{dec}_k(\cdot)$ ,  $\text{enc}_s(\cdot)$ ,  $\text{dec}_s(\cdot)$ ,  $\text{sig}(\cdot)$ ,  $\text{checksig}(\cdot, \cdot)$ , and  $\text{extractsig}(\cdot)$ ,
- $n$ -ary sequences  $(\cdot)$ ,  $(\cdot)$ ,  $(\cdot, \cdot)$ ,  $(\cdot, \cdot, \cdot)$ , etc., and
- projection symbols  $\pi_i(\cdot)$  for all  $i \in \mathbb{N}$ .

For strings (elements in  $\mathbb{S}$ ), we use a specific font. For example, `HTTPReq` and `HTTPResp` are strings. We denote by  $\text{Doms} \subseteq \mathbb{S}$  the set of domains, e.g., `example.com` in  $\text{Doms}$ . We denote by  $\text{Methods} \subseteq \mathbb{S}$  the set of methods used in HTTP requests, e.g., `GET`, `POST` in  $\text{Methods}$ .

**Definition 6 (Nonces and Terms).** By  $X = \{x_1, x_2, \dots\}$  we denote a set of variables and by  $\mathcal{N}$  we denote an infinite set of constants (nonces) such that  $\Sigma$ ,  $X$ , and  $\mathcal{N}$  are pairwise disjoint. For  $N \subseteq \mathcal{N}$  we define the set  $\mathcal{T}_N(X)$  of terms over  $\Sigma \cup N \cup X$  inductively: (1) If  $t \in N \cup X$ , then  $t$  is a term. (2) If  $f \in \Sigma$  is an  $n$ -ary function symbol in  $\Sigma$  for some  $n \geq 0$  and  $t_1, \dots, t_n$  are terms, then  $f(t_1, \dots, t_n)$  is a term.

The equational theory associated with the signature  $\Sigma$  is given in Figure A.1. By  $\equiv$  we denote the congruence relation on  $\mathcal{T}_N(X)$  induced by this theory. For example, we have that  $\pi_1(\text{dec}_s(\text{enc}_s(a, b), \text{pub}(k)), k) \equiv a$ .

#### Algorithm A.11 Web Browser Model: Main Algorithm.

```
Input:  $(n, f, m), s$ 
1: let  $s' := s$ 
2: if  $s$  is corrupted  $\& \neq \perp$  then
3:   let  $s'$  pending requests  $\leftarrow (n, s$  pending requests)  $\rightarrow$  Collect incoming messages
4:   let  $n := \mathbb{N}$ 
5:   let  $m'_1, \dots, m'_n \leftarrow d_i(s')$   $\rightarrow$  Create  $n$  new messages nondeterministically.
6:   let  $m'_1, \dots, m'_n \leftarrow \text{IPs}$ 
7:   stop  $\{(m'_1, n, m'_1), \dots, (m'_n, n, m'_n)\}, s'$ 
8: if  $m$  is TRIGGER then
9:   let  $\text{switch} \leftarrow \{\text{script}, \text{urlbar}, \text{reload}, \text{forward}, \text{back}\}$ 
10:  let  $\overline{w} \leftarrow \text{Subwindow}(s')$  such that  $s' \cdot \overline{w}$  documents  $\neq \emptyset$ 
11:   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some window.
12:  let  $\overline{w} \leftarrow \emptyset$  such that  $s' \cdot \overline{w}$  documents  $\neq \emptyset$ 
13:   $\rightarrow$  If possible; otherwise stop  $\rightarrow$  Pointer to some top-level window.
14:  if  $\text{switch}$  is script then  $\rightarrow$  Run some script.
15:    let  $\overline{w} \leftarrow \overline{w}$  active document.
16:    call RUNSCRIPT( $\overline{w}$ ,  $s'$ )
17:  else if  $\text{switch}$  is urlbar then  $\rightarrow$  Create some new request.
18:    let  $n := \mathbb{N}$ 
19:    if  $n := \mathbb{N}$  then  $\rightarrow$  Create a new window.
20:      let  $\text{windowname} := \pi_1$ 
21:      let  $s' := (\text{windowname}, \emptyset, \perp)$ 
22:      let  $s' \cdot \text{window} := s' \cdot \text{window} \cup s' \cdot \overline{w}$ 
23:      let  $\text{windowname} := s' \cdot \overline{w}$ 
24:    else  $\rightarrow$  Use existing top-level window.
25:      let  $\text{windowname} := s' \cdot \overline{w}$ 
26:      let  $\text{protocol} \leftarrow \{P, S\}$ 
27:      let  $\text{host} \leftarrow \text{Doms}$ 
28:      let  $\text{path} \leftarrow \mathbb{S}$ 
29:      let  $\text{fragment} \leftarrow \mathbb{S}$ 
30:      let  $\text{parameters} \leftarrow \mathbb{S} \times \mathbb{S}$ 
31:      let  $\text{url} \leftarrow \{\text{URL}, \text{protocol}, \text{host}, \text{path}, \text{parameters}, \text{fragment}\}$ 
32:      let  $\text{req} \leftarrow \{\text{HTTPReq}, \text{req}, \text{GET}, \text{host}, \text{path}, \text{parameters}, \emptyset, \emptyset\}$ 
33:      call HTTP.SEND( $\text{REQ}$ ,  $\text{windowname}$ ),  $\text{req}$ ,  $\text{url}$ ,  $\perp, \perp, \perp, s'$ 
34:    else if  $\text{switch}$  is reload then  $\rightarrow$  Reload some document.
35:      let  $\text{url} \leftarrow s' \cdot \overline{w}$  active document location
36:      let  $\text{req} \leftarrow \{\text{HTTPReq}, \text{req}, \text{GET}, \text{url}, \text{host}, \text{url}, \text{url}, \text{parameters}, \emptyset, \emptyset\}$ 
37:      let  $\text{referrer} \leftarrow s' \cdot \overline{w}$  active document.referrer
38:      call HTTP.SEND( $\text{REQ}$ ,  $s' \cdot \overline{w}$  nonce),  $\text{req}$ ,  $\text{url}$ ,  $\perp, \text{referrer}, \perp, s'$ 
39:    else if  $\text{switch}$  is forward then
40:      Subwindow( $s'$ ) such that  $s' \cdot \overline{w}$   $\text{document} \rightarrow s' \cdot \overline{w}$ 
41:      active document origin  $\neq s' \cdot \overline{w}$  active document origin ( $\overline{w}$  is not a top-level window)
42:      there is an ancestor window  $\overline{w}'$  of  $\overline{w}$  with an active document that has the same
43:      as the active document in  $\overline{w}$ , then  $\overline{w}' \in \overline{w}$ , and
44:      Subwindow( $s'$ ) such that  $s' \cdot \overline{w}' \text{ opener} = s' \cdot \overline{w}$  nonce  $\wedge \overline{w}' \in \overline{w}$  ( $\overline{w}'$  is a top-level window)
45:      it has an opener—and  $\overline{w}$  is allowed to navigate the opener window of  $\overline{w}'$ ,  $\overline{w}'$ 
46:       $\in \overline{w}$ .
```

Subwindow( $s'$ ) such that  $s' \cdot \overline{w}$   $\text{document} \rightarrow s' \cdot \overline{w}$  active document origin ( $\overline{w}$  is not a top-level window) there is an ancestor window  $\overline{w}'$  of  $\overline{w}$  with an active document that has the same as the active document in  $\overline{w}$ , then  $\overline{w}' \in \overline{w}$ , and

Subwindow( $s'$ ) such that  $s' \cdot \overline{w}$   $\text{opener} = s' \cdot \overline{w}$  nonce  $\wedge \overline{w}' \in \overline{w}$  ( $\overline{w}'$  is a top-level window) it has an opener—and  $\overline{w}$  is allowed to navigate the opener window of  $\overline{w}'$ ,  $\overline{w}'$   $\in \overline{w}$ .

In the definition of the following functions, we use  $n$ ,  $f$ ,  $m$ , and  $s$  as read-only global input and other variables are local variables or arguments.

Places throughout the algorithms we use placeholders to generate “fresh” nonces. Figure A.3 shows a list of all placeholders used.

Placeholder	Usage
Algorithm A.11, window nonces	
Algorithm A.11, HTTP request nonce	
Algorithm A.11, lookup key for pending HTTP requests entry	
Algorithm A.8, HTTP request nonce (multiple lines)	
Algorithm A.8, subwindow nonce	
Algorithm A.10, HTTP request nonce	
Algorithm A.10, document nonce	
Algorithm A.4, lookup key for pending DNS entry	
Algorithm A.1, window nonce	
Algorithm A.7, window nonce	
Algorithm A.7, HTTP request nonce	
Algorithm A.8, WebRTC connection nonce	
Algorithm A.8, replacement for placeholders in script output	

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

The function `GETNAVIGABLEWINDOW` (Algorithm A.1) is called by the browser to determine the window that is *actually* navigated when a script in the window  $s' \cdot \overline{w}$  provides a window reference for navigation (e.g., for opening a link). When it is given a window reference (nonce)  $\overline{w}$ , this function returns a pointer to a selected window term in  $s'$ :

- If  $\overline{w}$  is the string `_BLANK`, a new window is created and a pointer to that window is returned.
- If  $\overline{w}$  is a nonce (reference) and there is a window term with a reference of that value in the windows in  $s'$ , a pointer  $\overline{w}'$  to that window term is returned, as long as the window is navigable by the current window's document (as defined by `NavigableWindow` above).

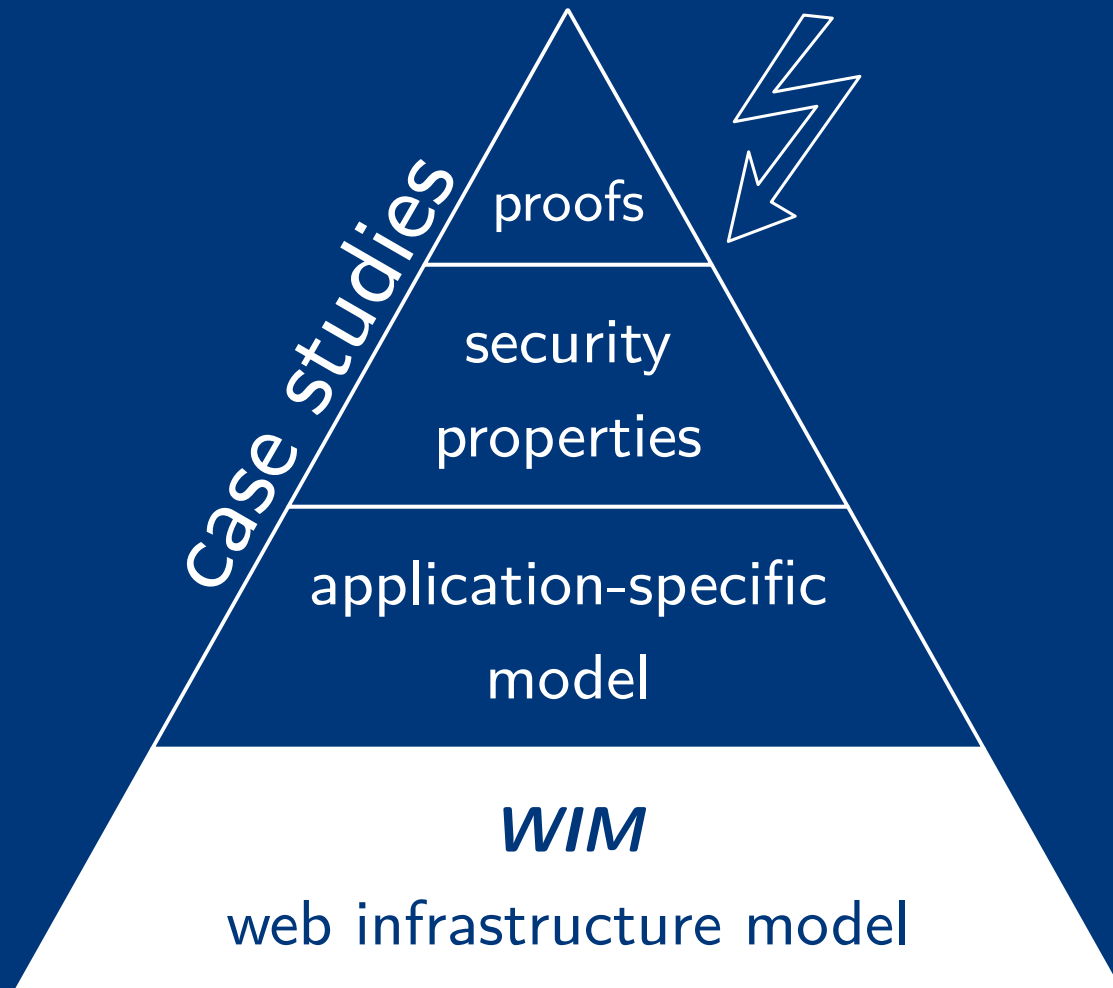
# Limitations

- ▶ 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

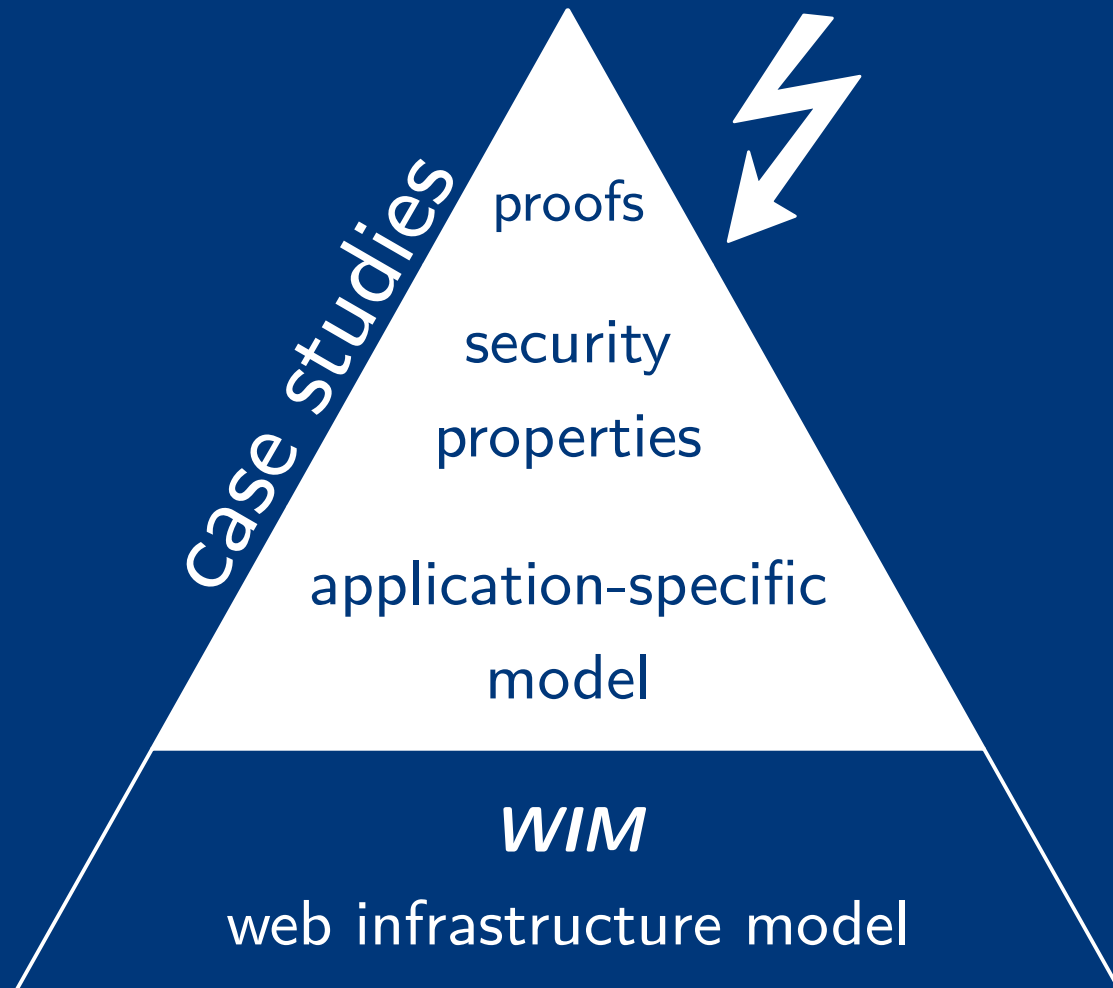
Model can in principle be extended to capture these aspects as well.

Trade-off: comprehensiveness vs. simplicity

# An Expressive Formal Model of the Web Infrastructure



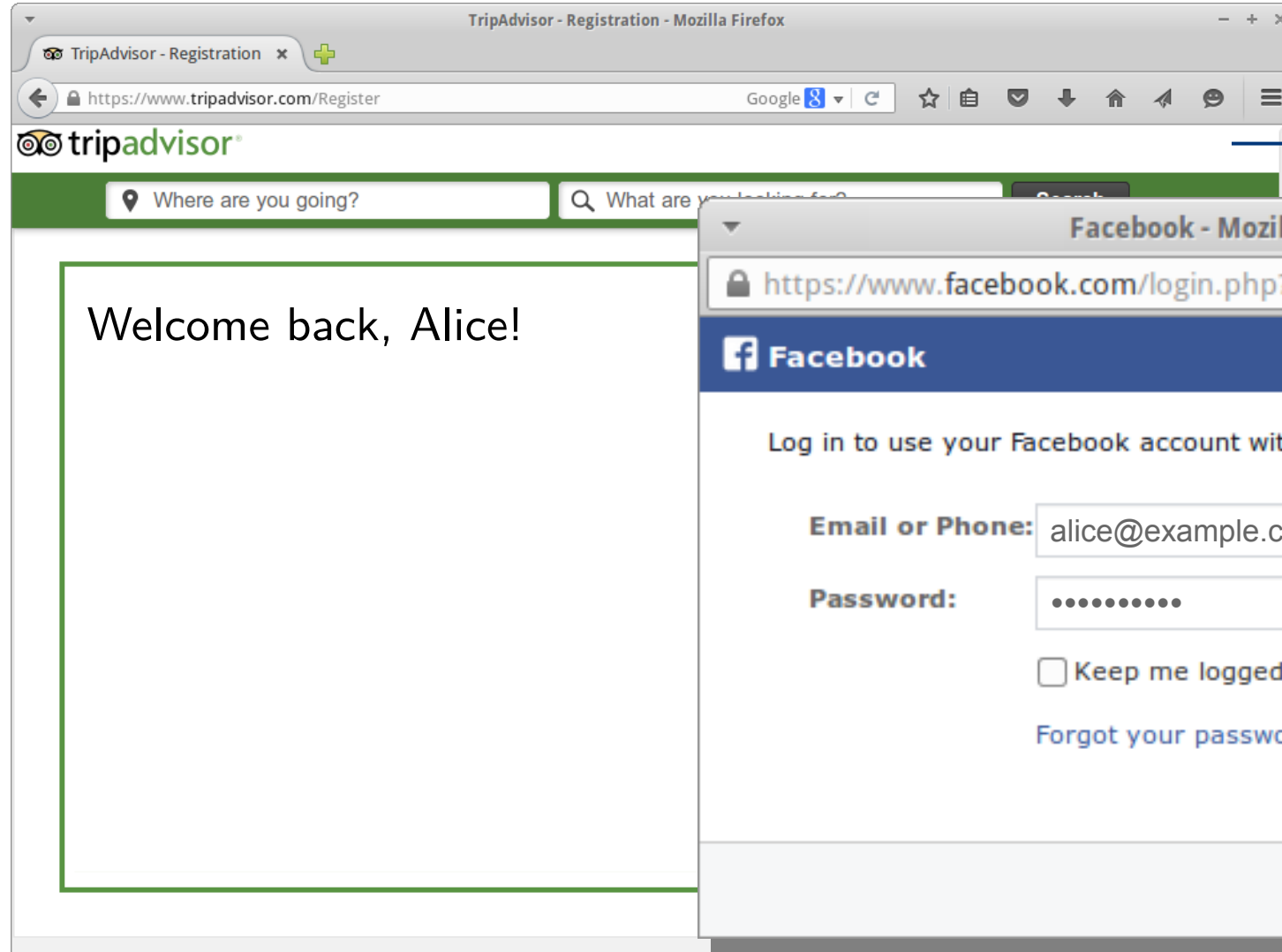
# An Expressive Formal Model of the Web Infrastructure



# WIM Case Studies

- ▶ Subject: web **single sign-on (SSO)** systems
- ▶ Interesting target for formal analysis:
  - Complex protocol flows
  - Multiple participants (typically  $\geq 3$ )
  - 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

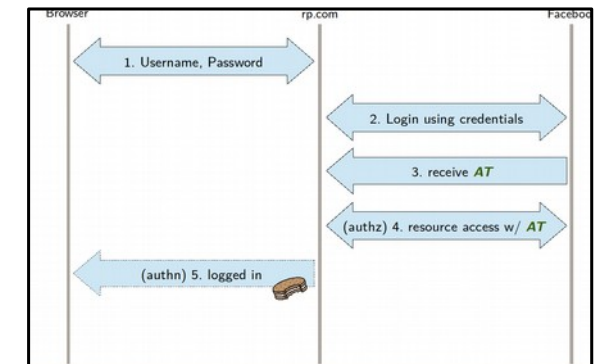
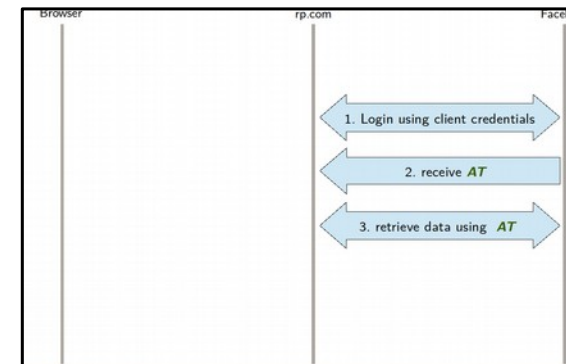
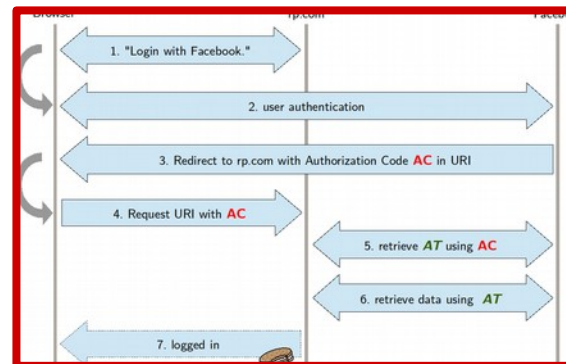
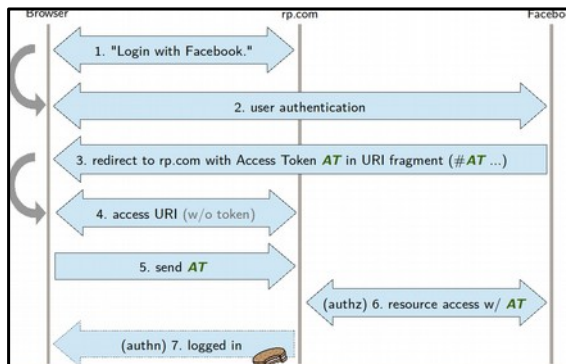


OpenID Connect

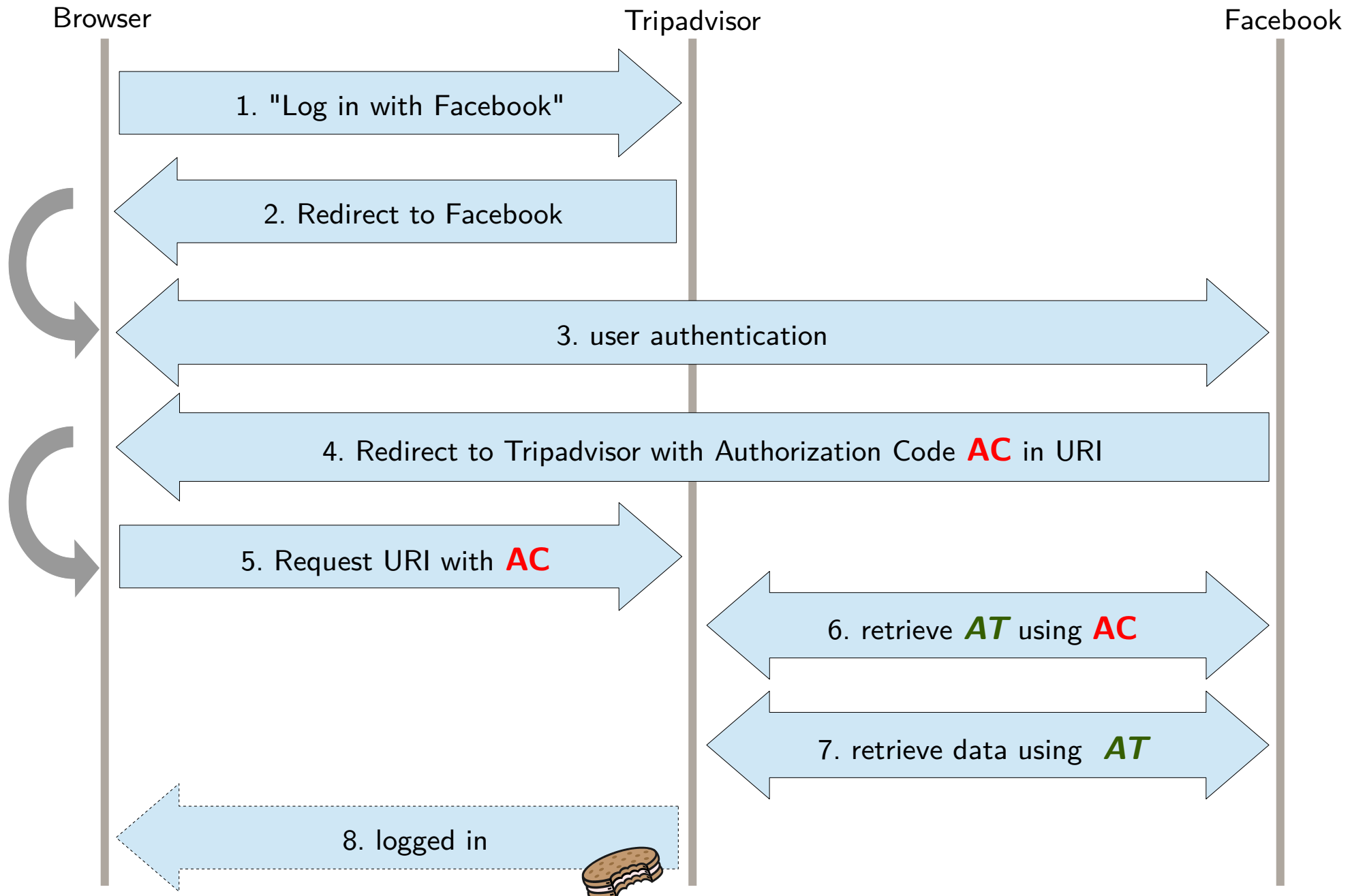


# OAuth 2.0


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

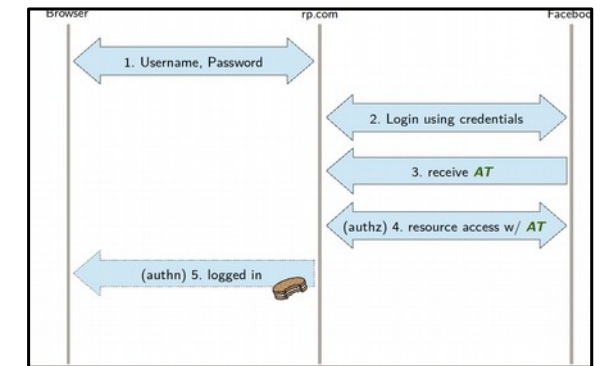
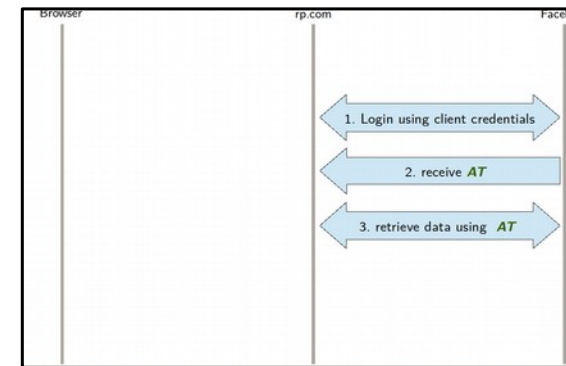
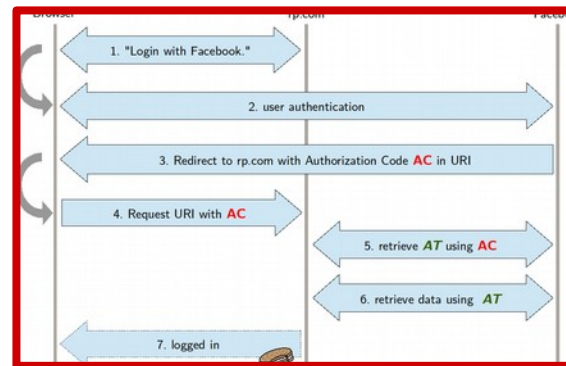
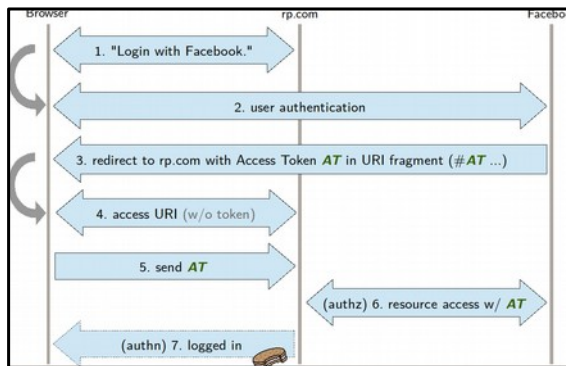
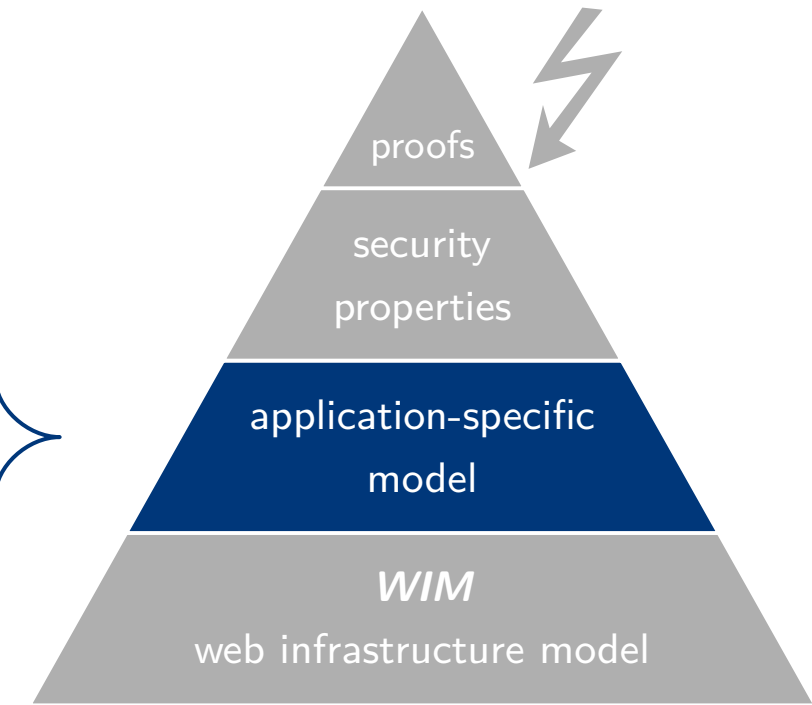


# OAuth 2.0



# OAuth 2.0

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



# OAuth 2.0: Security Properties

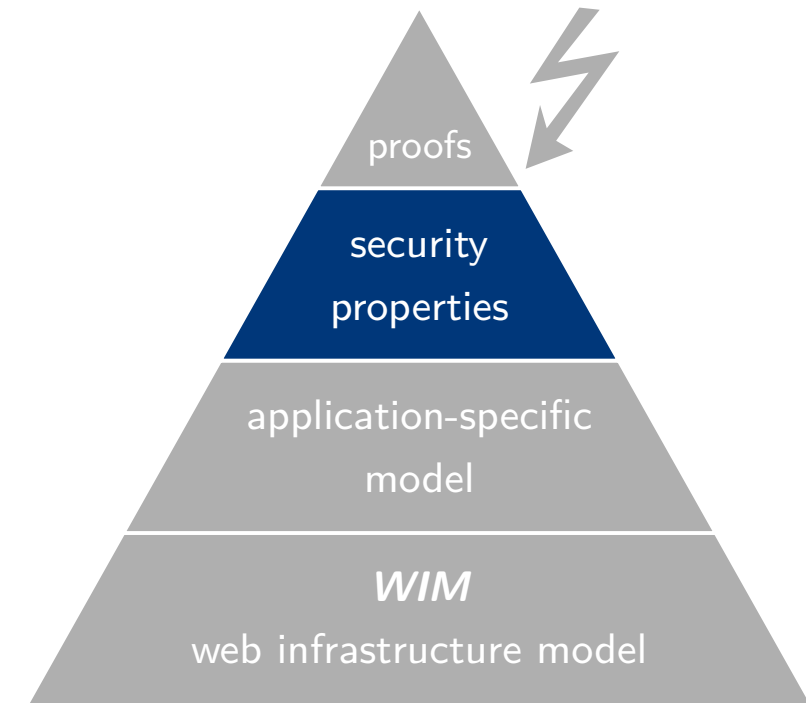
## ► Authentication

*Definition 56 (Authentication Property).* Let  $OAuthWS^n$  be an OAuth web system with a network attacker. We say that  $OAuthWS^n$  is secure w.r.t. authentication iff for every run  $\rho$  of  $OAuthWS^n$ , every state  $(S^j, E^j, N^j)$  in  $\rho$ , every  $r \in \text{Clients}$  that is honest in  $S^j$ , every  $i \in \text{OAP}$ , every  $g \in \text{dom}(i)$ , every  $u \in \mathbb{S}$ , every client service token of the form  $\langle n, \langle u, g \rangle \rangle$  recorded in  $S^j(r).\text{serviceTokens}$ , and  $n$  being derivable from the attackers knowledge in  $S^j$  (i.e.,  $n \in d_\emptyset(S^j(\text{attacker}))$ ), then the browser  $b$  owning  $u$  is fully corrupted in  $S^j$  (i.e., the value of *isCorrupted* is FULLCORRUPT), some  $r' \in \text{trustedClients}(\text{secretOfID}(\langle u, g \rangle))$  is corrupted in  $S^j$ , or  $i$  is corrupted in  $S^j$ .

## ► Authorization

*Definition 55 (Authorization Property).* Let  $OAuthWS^n$  be an OAuth web system with a network attacker. We say that  $OAuthWS^n$  is secure w.r.t. authorization iff for every run  $\rho$  of  $OAuthWS^n$ , every state  $(S^j, E^j, N^j)$  in  $\rho$ , every OAP  $i \in \text{OAP}$ , every  $r \in \text{Clients} \cup \{\perp\}$  with  $r$  being honest in  $S^j$  unless  $r = \perp$ , every  $u \in \text{ID} \cup \{\perp\}$ , for  $n = \text{resourceOf}(i, r, u)$ ,  $n$  is derivable from the attackers knowledge in  $S^j$  (i.e.,  $n \in d_\emptyset(S^j(\text{attacker}))$ ), it follows that

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



# OAuth 2.0: Security Properties

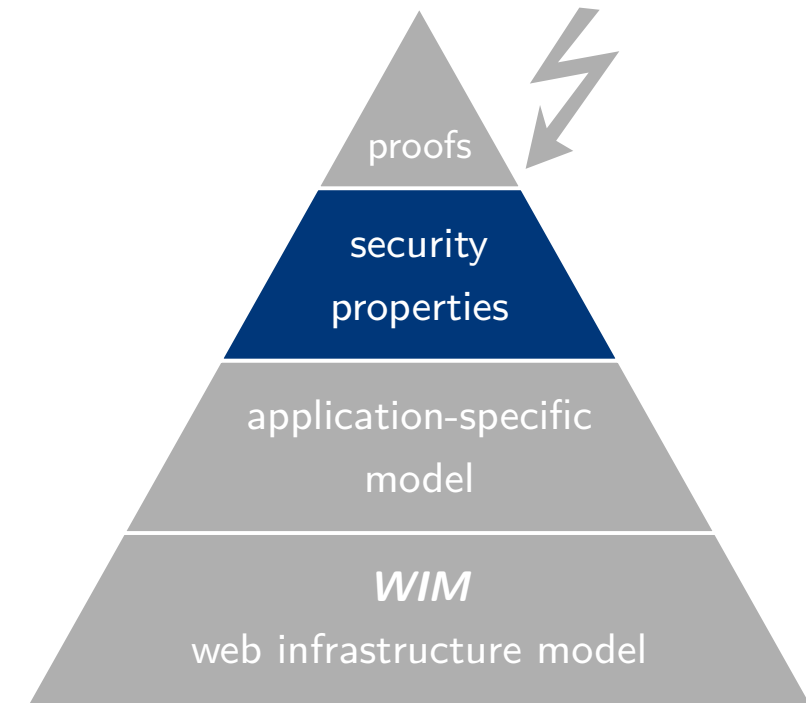
## ► Session Integrity for authentication

*Definition 64 (Session Integrity for Authentication).* Let  $OAuthWS^w$  be an OAuth web system with web attackers. We say that  $OAuthWS^w$  is secure w.r.t. session integrity for authentication iff for every run  $\rho$  of  $OAuthWS^w$ , every processing step  $Q_{login}$  in  $\rho$ , every browser  $b$  that is honest in  $Q_{login}$ , every  $r \in \text{Clients}$  that is honest in  $Q_{login}$ , every  $i \in \text{OAP}$ , 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  $m \in \text{dom}(i)$  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  $o \in \text{OASessions}(\rho, b, r, i)$ , and
- (b) if  $i$  is honest in  $Q_{login}$  then  $Q_{login}$  is in  $o$  and we have that

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

## ► Session Integrity for authorization (similar to above)



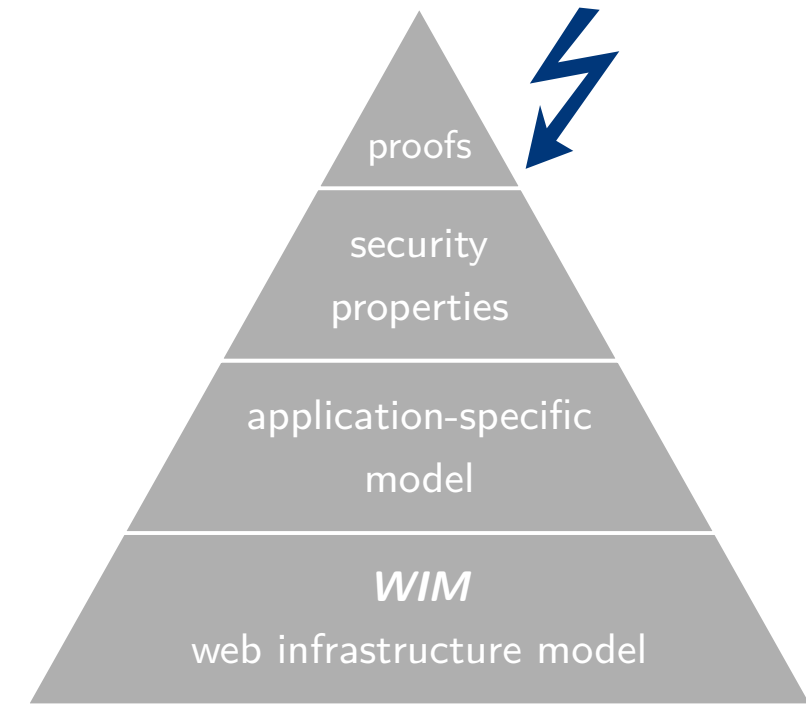


# OAuth 2.0: New Attacks

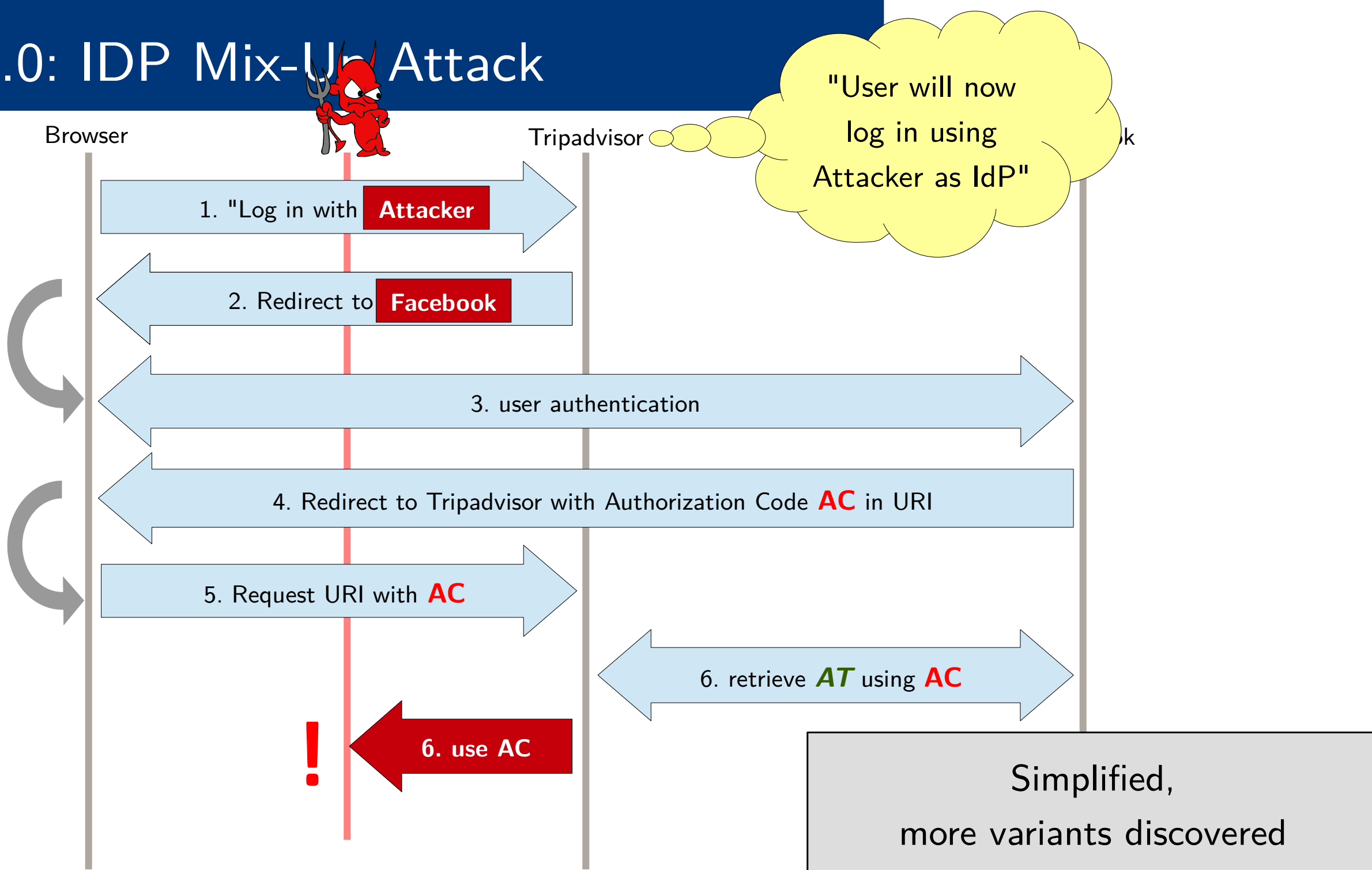
OAuth 2.0 had been analyzed many times before, but not in a comprehensive formal model.

## 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



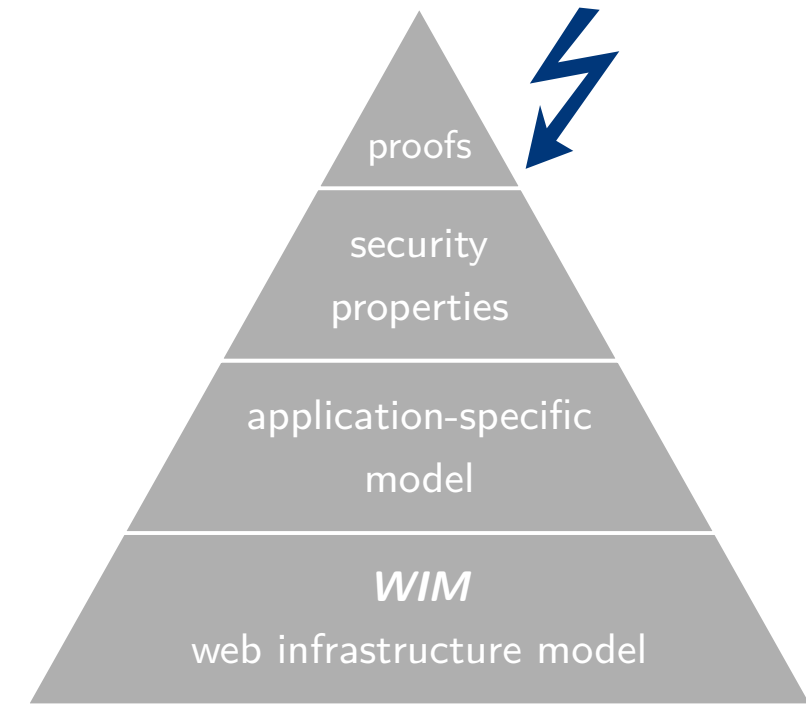


# OAuth 2.0: New Attacks

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

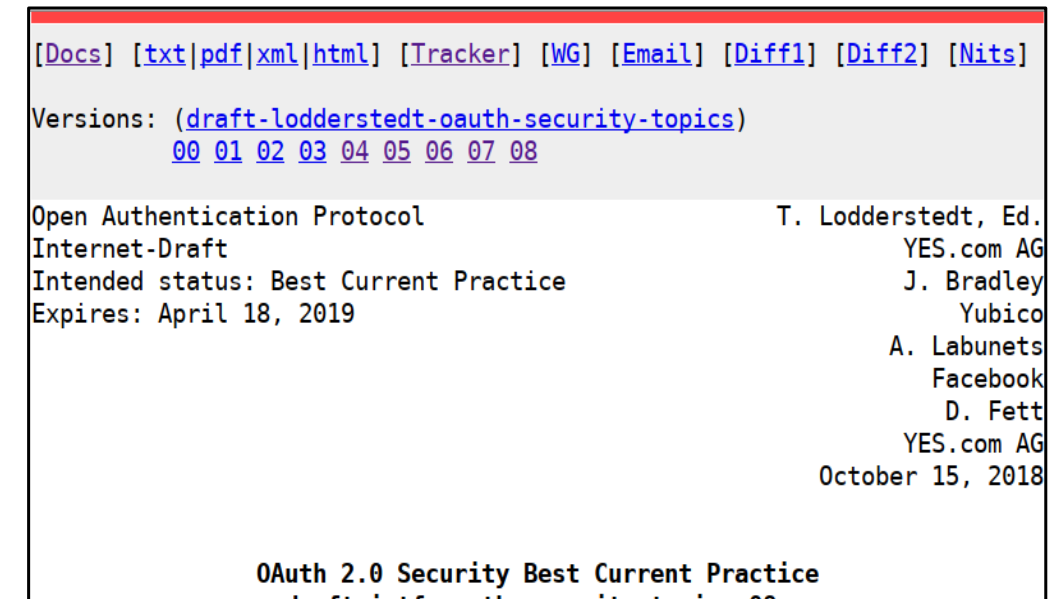


# OAuth 2.0: Impact

- ▶ 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]



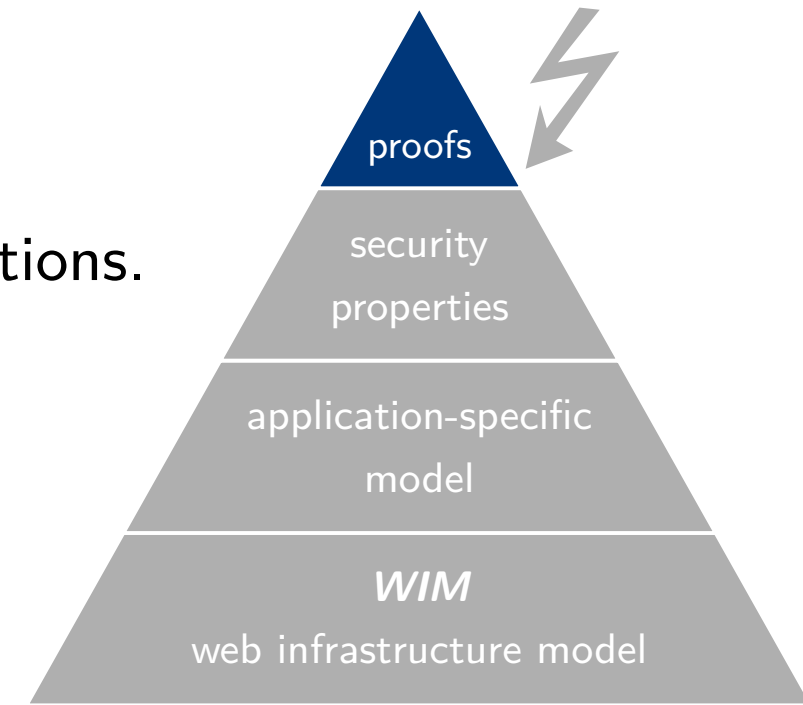
# OAuth 2.0: Proof of Security

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 3<sup>rd</sup> party scripts on certain web pages)
- ▶ [Fixes](#) against previously known and new attacks

*Theorem 1.* Let  $OAuthWS^n$  be an OAuth web system with a network attacker, then  $OAuthWS^n$  is secure w.r.t. authorization and secure w.r.t. authentication. Let  $OAuthWS^w$  be an OAuth web system with web attackers, then  $OAuthWS^w$  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

## 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  $OIDCWS^n$  be an OIDC web system with a network attacker. Then,  $OIDCWS^n$  is secure w.r.t. authentication and authorization. Let  $OIDCWS^w$  be an OIDC web system with web attackers. Then,  $OIDCWS^w$  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

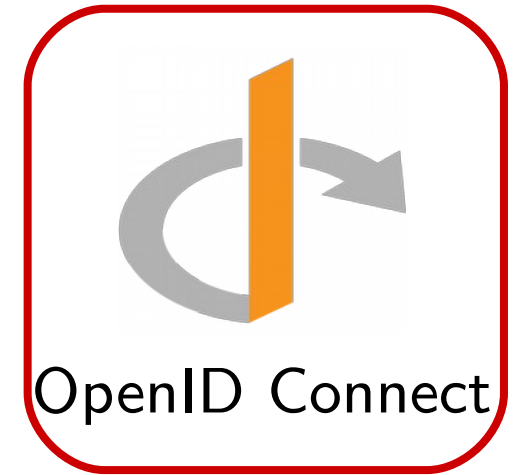
SPRESSO

- ▶ 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

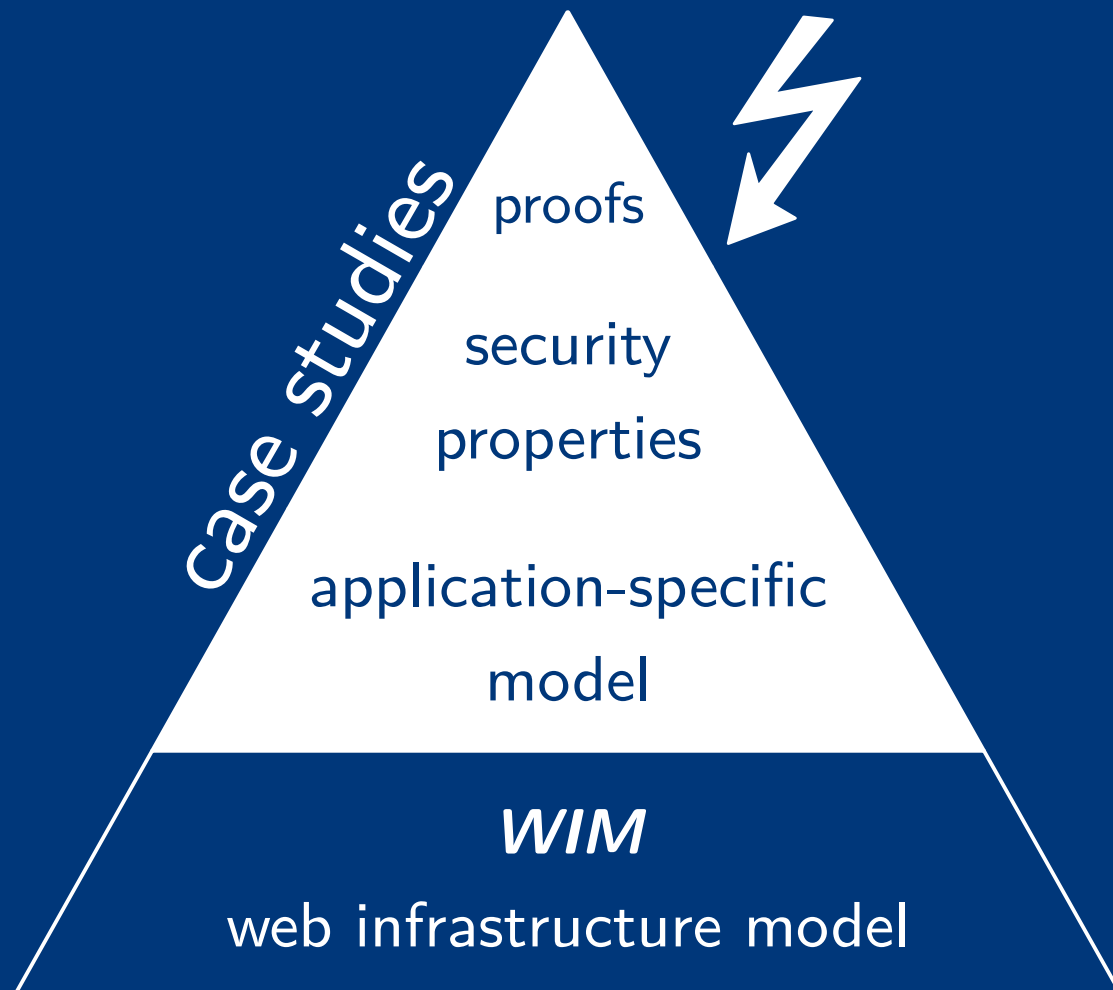


OpenID Connect

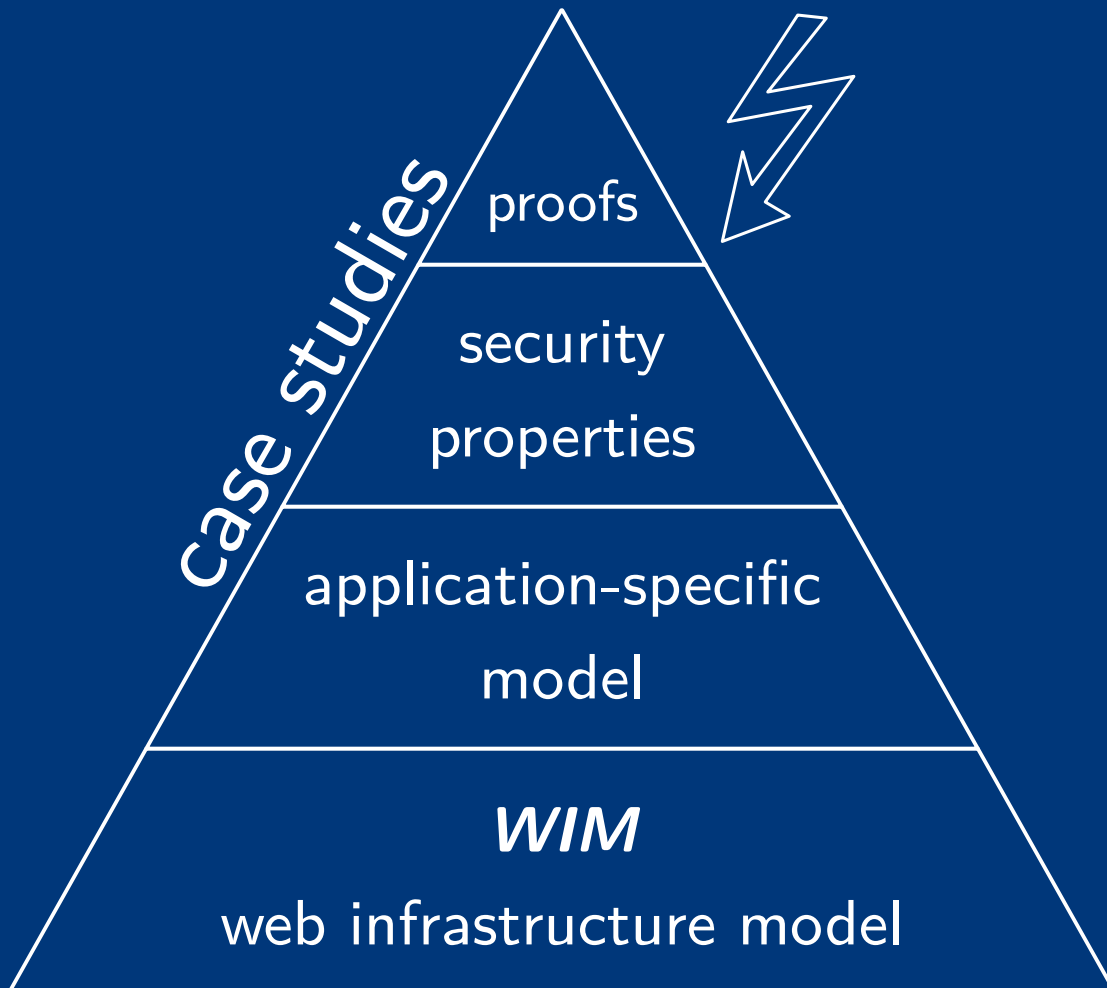
- ▶ Including extensions
- ▶ Developed best practices against known attacks
- ▶ Proof of security



# 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!**