WIM: An Expressive Formal Model of the Web Infrastructure

Ralf Küsters

2019-06-27
Many Web Attacks...

- Cross-Site Request Forgery
- Leaks of Sensitive Data
- Clickjacking
- Man-in-the-middle Attacks
- Attacks on Single Sign-on
- DNS Rebinding
- Cross-Site Scripting
- Malicious iframes
- Malicious CDNs
- Missing Checks
- Other Injection Attacks
- SQL Injection
- Attacks on Session Management
- Attacks on Cookies
- Malicious injection attacks
- XSS
...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: Current Practice

Expert review of standards and implementations

Penetration testing using tools or manual analysis

CHECKLIST

- CSRF
- Session Swapping
- Missing Checks
- Cross-Origin Attacks
- Insecure Connection
- Man-in-the-middle
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 vulnerabilities, and even prove security (in a meaningful model of the web infrastructure as a whole)?**
Our Model-Based Approach

For instance:
Single Sign-On Standards and Applications (OAuth, OIDC, Financial-grade API, etc.)
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

Case studies

proofs

security properties

application-specific model

WIM
web infrastructure model
Prior Web Models

► [Kerschbaum 2007]
  Analysis of CSRF protection in the Alloy model checker

► [Akhawe, Barth, Lam, Mitchell, Song 2010]
  First formal "web model", in Alloy, five case studies

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

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

Our approach: goal was a very detailed, close-to-standards web model, (started with) pen-and-paper.
Further Related Work (Formal Analysis)

► [Kumar et al., 2011-2014]: Alloy-based with BAN logic
► [Bai et al., 2013]: AuthScan + ProVerif
► [Bohannon and Pierce, 2010]
  - "Featherweight Firefox"
  - Information Flow tracking in web browser core
  - No security policies by default
► [Sabelfeld et al. 2016]: Information-flow security for JavaScript and its APIs
► [Börger et al., 2012]
  - Abstract State Machines
  - Focus on web server, limited browser model
The Web Infrastructure Model \textit{WIM}

► Detailed, comprehensive, and precise formal model
  - Network interactions
  - Attacker behavior
  - 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
Dolev-Yao-Style Model:
- Messages are terms
- Attacker, Browsers, Servers, Scripts (honest or malicious) are Dolev-Yao processes
- **Not** just a standard Dolev-Yao model for protocol analysis, but rather covers web features, close to web standards.
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
Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts (honest and malicious)
- web storage & cookies
- web messaging & XHR
- message headers (Origin, STS, Location, Referer, ...)
- redirections
- security policies
- WebRTC
- dynamic corruption
- ...

Origin: https://example.com
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 $c \in \{\}$ response.headers[Set-Cookie], $c \in$ 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
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

Case studies

proofs

security properties

application-specific model

WIM
web infrastructure model
An Expressive Formal Model of the Web Infrastructure

Case studies
- proofs
- security properties
- application-specific model

WIM
web infrastructure model
Welcome back, Alice!

Web **single sign-on (SSO)** systems

Interesting target for formal analysis:
- Complex protocol flows
- Multiple participants (typically ≥3)
- High security requirements
WIM Case Studies

 Mozilla BrowserID

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

SPRESSO
https://spresso.me

OAuth 2.0

OpenID Connect
BrowserID: Privacy Attack

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

*Cannot be fixed without a major redesign of BrowserID!*

present iff user logged in at RP before.
WIM Case Studies

Mozilla BrowserID

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

SPRESSO
https://spresso.me

OAuth 2.0

OpenID Connect

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

https://spresso.me
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

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 $\text{OAuthWS}^n$ be an OAuth web system with a network attacker. We say that $\text{OAuthWS}^n$ is secure w.r.t. authentication iff for every run $\rho$ of $\text{OAuthWS}^n$, every state $(S^i, E^i, N^i)$ 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 $u \in S$, 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., $n \in d_0(S^j(\text{attacker}))$), then the browser $b$ owning $u$ is fully corrupted in $S^j$ (i.e., the value of $\text{isCorrupted}$ is FULLCORRUPT), some $r' \in \text{trustedClients}(\text{secretOfID}(\langle u, g \rangle))$ is corrupted in $S^j$, or $i$ is corrupted in $S^i$.

► Authorization

Definition 55 (Authorization Property). Let $\text{OAuthWS}^n$ be an OAuth web system with a network attacker. We say that $\text{OAuthWS}^n$ is secure w.r.t. authorization iff for every run $\rho$ of $\text{OAuthWS}^n$, every state $(S^j, E^j, N^j)$ in $\rho$, every OAP $i \in \text{OAP}$, every $r \in \text{Clients} \cup \{\bot\}$ with $r$ being honest in $S^j$ unless $r = \bot$, every $u \in \text{ID} \cup \{\bot\}$, for $n = \text{resourceOf}(i, r, u)$, $n$ is derivable from the attackers knowledge in $S^j$ (i.e., $n \in d_0(S^j(\text{attacker}))$), 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 $r' \in \text{trustedClients}(\text{secretOfID}(u))$ is corrupted in $S^j$. 
OAuth 2.0: Security Properties

► Session Integrity for authentication

Definition 64 (Session Integrity for Authentication). Let \( \text{OAuthWS}^w \) be an OAuth web system with web attackers. We say that \( \text{OAuthWS}^w \) is secure w.r.t. session integrity for authentication iff for every run \( \rho \) of \( \text{OAuthWS}^w \), every processing step \( Q_{\text{login}} \) in \( \rho \), every browser \( b \) that is honest in \( Q_{\text{login}} \), every \( r \in \text{Clients} \) that is honest in \( Q_{\text{login}} \), every \( i \in \text{OAP} \), every identity \( \langle u, g \rangle \), the following holds true: If in \( Q_{\text{login}} \) a service token of the form \( \langle n, \langle u', g' \rangle, m \rangle \) 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 \( Q_{\text{login}} \) then \( Q_{\text{login}} \) is in \( o \) and we have that

\[
\left( \text{selected}_{\text{in}}(o, b, r, \langle u, g \rangle) \lor \text{selected}_{\text{nia}}(o, b, r, \langle u, g \rangle) \right) \iff \left( \langle u, g \rangle \equiv \langle u', g' \rangle \right).\
\]

► 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.
Further Related Work (OAuth 2.0)

- [Bansal et al., 2012-2014]
- [Wang et al., 2013]
  - "Explicating SDKs"
  - Boogie/Corral
  - Extraction of SDK logic, definition of security properties, addition of assume statements, code verification.
- [Chari, Jutla, Roy, 2011]
  - UC model analysis of OAuth Authorization Code Grant
  - No web features
- Several empirical studies, focussed on typical implementation errors
Further Related Work (OpenID Connect)

- [Mladenov et al., 2016]
  - Specific variant of the IDP Mix-Up attack
  - No formal model

- [Li, Mitchell, 2016]
  - Implementation errors in deployments of Google Sign-In
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

1. "Log in with Attacker"

2. Redirect to Facebook

3. User authentication

4. Redirect to Tripadvisor with Authorization Code AC in URI

5. Request URI with AC

6. Use AC

"User will now log in using Attacker as IdP"

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: 307 Redirect Attack (I)

1. "Login with IdP."
2. user authentication
3. Redirect to rp.com with AT or AC
4. access URI
5. retrieve data using AT
OAuth: 307 Redirect Attack (II)

3. 307 Redirect to rp.com with AT or AC

4.a Request URI + username & password

HTTP Status Code 307:
Redirect repeats POST data in new request

User enters her login data
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 $\text{OAuthWS}^n$ be an OAuth web system with a network attacker, then $\text{OAuthWS}^n$ is secure w.r.t. authorization and secure w.r.t. authentication. Let $\text{OAuthWS}^w$ be an OAuth web system with web attackers, then $\text{OAuthWS}^w$ is secure w.r.t. session integrity for authorization and authentication.
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]
WIM Case Studies

Mozilla BrowserID

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

SPRESSO
https://spresso.me

► 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
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

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 $OIDDWS^n$ be an OIDC web system with a network attacker. Then, $OIDDWS^n$ is secure w.r.t. authentication and authorization. Let $OIDDWS^w$ be an OIDC web system with web attackers. Then, $OIDDWS^w$ 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
https://spresso.me

- 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

Most recent case study: Financial-grade API (FAPI)
Motivation FAPI

- Authorization and authentication in high-risk scenarios
- Laws and activities for opening financial services to third-party providers
  - OpenBanking UK: Financial-grade API already adopted by major banks in the UK
  - European Payment Services Directive 2 (PSD2): September 2019
  - Many other countries follow similar approaches

Authorize app: Access to banking account

Financial-grade API
OpenID Financial-grade API:

- Hardened version of **OAuth 2.0** for high-risk use-cases

- **New mechanisms:** OAuth 2.0 Token Binding, OAuth 2.0 Mutual TLS, Proof Key for Code Exchange, JWT Secured Authorization Response Mode
FAPI: Attacker Model

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 $AT$

7. Retrieve data using $AT$

Possible leakages according to specification
Overview FAPI

► OpenID Financial-grade API:
  - Hardened version of OAuth 2.0 for high-risk use-cases
  - **New mechanisms:** OAuth 2.0 Token Binding, OAuth 2.0 Mutual TLS, Proof Key for Code Exchange, JWT Secured Authorization Response Mode

► Our Work: formal security analysis of the Financial-grade API
  - Formal model of the Financial-grade API based on the Web Infrastructure Model
  - Precise definition of security properties
  - During formal analysis: **found several attacks** bypassing the new mechanisms
  - **Proof of security** for the fixed Financial-grade API

► Collaborating with OpenID Foundation to fix the standard
WIM Case Studies

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

SPRESSO
https://spresso.me
- 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

Most recent case study: Financial-grade API (FAPI)
An Expressive Formal Model of the Web Infrastructure

- proofs
- security properties
- application-specific model
- WIM
  web infrastructure model

Case studies
WIM: An Expressive Formal Model of the Web Infrastructure

Thank you!

- Most detailed and comprehensive formal model of the web infrastructure so far
- Case studies with real-world impact
- New classes of attacks
- Formal proofs of web security with very high level of detail
- Designed first privacy-preserving SSO system: SPRESSO
- Currently: mechanized model, in collaboration with Bhargavan et al.