MULTI-REPRESENTATIONAL SECURITY ANALYSIS

Eunsuk Kang
MIT CSAIL

ExCAPE Webinar
April 30, 2015
Third-Party Authentication

OAuth protocol
Widely adapted, support from major vendors
Analyzed formally using verification tools
UPDATE — A serious vulnerability in the OAuth and OpenID protocols could lead to complications for those who use the services to log in to websites like Facebook, Google, LinkedIn, Yahoo, and Microsoft among many others.

Empirical study of OAuth providers [Sun & Beznosov, CCS12] Majority of them vulnerable (Google, Facebook,...)
Provably Secure, Practically Vulnerable?

Protocol analysis at the abstract level
Agents, messages, and rules at each protocol step

1.2. Protocol Flow

RFC OAuth 2.0, Internet Engineering Task Force
Provably Secure, Practically Vulnerable?

Protocol analysis at the abstract level
Agents, messages, and rules at each protocol step

Attacks exploiting details absent from model
Unanticipated behavior from browser features
e.g. HTTP redirects, hidden forms
Design flaws, not just bugs in code

Blame protocol designers?
Can’t predict all manners in which protocol will be used!
Abstractions in System Design

Most systems too complex
Decompose into overlapping aspects
Abstraction vs Security

System designer
One system view/abstraction at a time
Exclude details in other views
Abstraction vs Security

System designer
One system view/abstraction at a time
Exclude details in other views
Abstraction vs Security

Shopping cart interaction

Payment workflow

Deployment on HTTP

Mobile app API

Customer authentication

Attacker

Jump between abstractions & combine details
Property held over one view might be violated in another!
Poirot

A framework for incremental reasoning of security properties across multiple abstraction boundaries
Standard Verification Framework

Property

System Model

Verification Engine

Analysis Result

$M_S \models P$?
Our Framework: Poirot

- Property
- System Model
- Mapping
- Verification Engine
- Composition Operator
- Domain Model Library
- Analysis Result
Our Framework: Poirot

Start with & analyze an initial model $M_S$
Our Framework

1. Property
2. System Model
3. Mapping
4. Domain Model Library
5. Composition Operator
6. Verification Engine
7. Analysis Result

Generic, reusable models e.g., browser, protocols
Our Framework

- Property
- System Model
- Mapping
- Verification Engine
- Composition Operator
- Domain Model Library
- Analysis Result

Elaboration of \( M_S \) with knowledge in \( M_D \)

\[ M' = \text{Compose}(M_S, M_D, K) \]
Our Framework

P → Verification Engine → Analysis Result

M' → Composition Operator → Domain Model Library

Property
System Model
Mapping

M' ⊨ P?

Re-analyze P & repeat
Key Features

Increment, iterative analysis
Start with an initial model, elaborate, analyze & repeat
Find & fix simple attacks first, complex ones later
Explore impact of design alternatives

Reusing domain knowledge
Encoded by domain experts
Reusable across multiple systems
Details hidden from user
System Model

Property → Verification Engine → Analysis Result
System Model → Composition Operator
Mapping → Domain Model Library
Modeling Approach

**Inspirations**
Architectural description languages
Process algebras (CSP)

**Structure**
Components with input/output channels
End-to-end dataflow through channels

**Behavior**
Trace-based semantics (event traces)
Declarative constraints for specifying behavior
Structure

Model

Components (C), operations (O), datatypes (D)

D = \{CustomerID, ItemID, Cart, Credential, Order\}
Structure

Component
Input & output channels
Each channel typed to an operation
Connection between channels with the same type
Structure

Component states
Encoded as relations
Updated by operations

creds \subseteq \text{CustomerID} \times \text{Credential}
cart \subseteq \text{CustomerID} \times \text{ItemID}
orders \subseteq \text{CustomerID} \times \text{Order}
Behavior

Behavior as traces
A stream of events generated through each channel
Behavior of a component $\equiv$ set of all possible traces

$$\text{beh}: C \rightarrow \mathcal{P}(T)$$
Behavior

Interaction
Parallel composition, synchronized on shared operation

\[ \langle \text{addItem}_1, \text{addItem}_2, \text{showCart}_1 \rangle \in \text{beh}(\text{Customer} \parallel \text{MyStore}) \]
Behavior

Restricting behavior
Each channel associated with a *guard* condition
Specified as a declarative constraint

\[
\text{Guard on } Y_1: \\
\forall e \in E \mid (e.\text{cust}, e.\text{cred}) \in p.\text{creds}
\]
Dataflow Model

Inter-component data flow
Transmitted as arguments of an operation
\( v \in \text{hasAccess}(c, t) \) if \( C \) has access to value \( v \) after trace \( t \)

\[
\text{hasAccess}: (C \times T) \rightarrow \mathcal{P}(V)
\]

Assumptions
\( C \) has access to value \( v \) if

(1) it initially knows \( v \) or
(2) it has received an event with \( v \) as argument

Can send an event only if it has access to all arguments
Verification

- Property
- System Model
- Mapping

Verification Engine

Composition Operator

Domain Model Library

Analysis Result
Analysis

Encoding in Alloy
Modeling language based on first-order relational logic
Translation to a SAT formula; solve(M ∧ ¬P)
Bounded verification (trace length, data values)

Properties
Confidentiality & integrity
More broadly, safety properties

“Customer should be able to access only its own orders”

∀ t ∈ T, o: Order, s: MyStore, c: Customer | o ∈ hasAccess(c, t) ⇒ (c.id, o) ∈ s.orders(t)
Domain Model Library

- Property
- System Model
- Mapping
- Verification Engine
- Composition Operator
- Domain Model Library
- Analysis Result
Domain Model Library

Web-related domain models

<table>
<thead>
<tr>
<th>Category</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Browser, HTTP server, network endpoint</td>
</tr>
<tr>
<td>Feature</td>
<td>Encryption, scripting, same-origin policy, cross-site origin requests (CORS), PostMessage</td>
</tr>
<tr>
<td>Protocol</td>
<td>SSL, OAuth, OpenID</td>
</tr>
</tbody>
</table>

Benefits
Amortized cost; built once, reused multiple times
Expertise not required of user
Composing Models

Problem
Two partial models, describing different viewpoints
With distinct vocabularies
But potentially *overlapping* in reality

\[ M_1: \text{Store model} \]
\[ M_2: \text{HTTP model} \]
Representations

Representation relation \((R)\)
Informally, “represented as” \(\cong\) “deployed as”, “implemented as”, “encoded as”

Formally, every instance of \(S_1\) is also an instance of \(S_2\)

Introduced between a pair of components, operations, datatypes i.e., \(R = (R_C, R_O, R_D)\)
Relating Elements

Representation as a way of describing overlaps

\[ (\text{AddItem}', \text{HttpReq}) \in R \]

i.e. “AddItem is implemented as a kind of HTTP requests”
Mapping Elements

Describe how members of the two sets are mapped

AddItem(customer: CustomerID, item: ItemID, cred: Credential)

HttpReq (method: Method, url: URL, headers: set Header)

HttpReq

= = =

AddItem'

Mapping as a concretization relation
Corresponds to design decisions!
Complex Mapping

MyStore

Server

OAuth Client

Server

i.e. “MyStore is deployed as HTTP server and also participates in OAuth protocol”
Behavior Expansion

Effects of composition
Possible introduction of behavior into the system
Previously verified properties may no longer hold!

Types of expansion
Channel Addition
Dataflow Expansion
Alphabet Expansion
Channel Addition

Component obtains channels & generates new events
Customer interacts with other HTTP servers
Allows attacks where attacker sets up malicious pages
New types of data flow into component
Server may now receive credentials as headers
Component generates new event types
Any HTTP client can invoke AddItem operation
Can’t assume how AddItem will be invoked
Common mistake among web developers!
Composition Operator

Composition function

\[ M' = \text{compose}(M_1, M_2, K) \]

with mappings \( K = (K_C, K_O, K_D) \)

Two-step procedure

For each pair \((S_1, S_2)\) appearing in \( K \):

1. **Merge** Introduce representation relation between \( S_1 \) & \( S_2 \)
2. **Refine** Constrain members of \( S_1 \) & \( S_2 \) according to \( K \)
## Merge

1. Introduce representation relation between $S_1$ & $S_2$

Construct $S_1'$ that obtains characteristics (channels, arguments, fields) of $S_1$ & $S_2$

In resulting model $M'$:
- $(MyStore', Server) \in R_C$
- $(Customer', Browser) \in R_C$
- $(AddItem', HttpReq) \in R_O$
2. Constrain members of $S_1$ & $S_2$ according to $K$
Specified as a declarative constraint

$$\{a: \text{AddItem, } c: \text{HttpReq} \mid$$
$$c.\text{method} = \text{“POST”} \land$$
$$a.\text{customer} \in c.\text{url.query} \land$$
$$a.\text{item} \in c.\text{url.query} \land$$
$$a.\text{cred} \in c.\text{headers} \} \subseteq K_O$$
Partial Mappings

Mapping can be *partially* specified

\[
\{ a: \text{AddItem}, c: \text{HttpReq} \mid c.\text{method} = \text{"POST"} \} \subseteq K_0
\]

Analyzer generates mapping that leads to attack!
User can leave some design decisions unknown &
interactively explore alternative mappings
Case Studies

Two publicly deployed sites
IFTTT  End-user service composition
HandMe.In  Personal item tracking with QR code
Case Studies

Two publicly deployed sites
IFTTT & HandMe.In

Methodology
Start with an abstract workflow model
Elaborate with domain models & re-analyze
Replay attacks on the concrete system

Results
Previously unknown attacks on both sites
Confirmed with the developers
Analysis times < 15 seconds
Modeling effort: 1~2 weeks
Attack on IFTTT

Authentication on IFTTT
User creates a recipe with trigger & action
IFTTT must be given third-party access to both (OAuth)
Attack on IFTTT

Confidentiality Property
User’s private data is not accessible to attacker
Attack on IFTTT

Login CSRF
Attacker tricks user into signing in under its credentials
Relatively minor; may see history of user’s actions

But in combination with IFTTT
Security consequences far greater!
Can leak private data through IFTTT channels

Complex interaction
Between IFTTT, OAuth, browser models
Attack on IFTTT
Attack on IFTTT

1. User visits a page hosted by evil server
Attack on IFTTT

2. User gets logged onto Blogger under attacker’s credential
Attack on IFTTT

3. User creates a new recipe on IFTTT
Attack on IFTTT

4. Blogger authorizes IFTTT to perform action, but under the attacker’s account!
Attack on IFTTT

5. User posts a photo on Facebook
Attack on IFTTT

6. IFTTT is notified of the trigger
Attack on IFTTT

7. IFTTT creates a new blog on the attacker’s account, with the user’s photo!
Internet of Things Security

(Collaboration with Hamid Bagheri)
Looking Ahead

Synthesis:
Construct $K$ that satisfies $P$
Model Extraction:
Mine specs from domain sources (e.g. CVE)
Poirot
A framework for incremental analysis of security properties across multiple system abstractions

Any questions?