InteGuard:
Toward Automatic Protection of
Third-Party Web Service Integrations

Luyi Xing (Indiana University)
Yangyi Chen (Indiana University)
XiaoFeng Wang (Indiana University)
Shuo Chen (Microsoft Research)
INTRODUCTION
Introduction

- Web applications integrate third-party Web services.
Introduction

- Web applications integrate third-party Web services.
Introduction-cont.

• Security challenge: coordinate **Website (Integrator)**, Service Provider and Web Client.

• Integrator error-prone, difficult to be secure ([Oakland’11, Oakland’12]).
Introduction-Cont.

- Protection
  - Integrator side more error-prone.
  - Traffic among integrator, provider and clients is generally mechanic.
BACKGROUND
Background

• Third-Party Web Service Integration
  ➢ RT
  (HTTP request/response pair, or HTTP Round Trip)

- Client
  (e.g. shopper)

- Provider
  (e.g. PayPal)

- Integrator
  (e.g. Bestbuy.com, Newegg.com)
### Logic Flaws

#### RT1.request

```
https://Jeff.com/placeOrder
```

#### RT2.request

```
```

#### RT3.request

```
https://alice.com/finishOrder?gross=9.99&orderID=123&payeeEmail=alice@email.com
```

- **AccountId**: bob
- **payeeEmail**: bob@email.com
Previous Research

• Web Logic Flaws.
  ➢ Swift, Ripley, Swaddler, MiMoSA, Waler, Rolecast, Execution After Redirect, SAFERPHP, WAPTEC, APP_LogGIC, Fix_Me_Up, NoTamper, Block

• Conventional two-party settings (websites, clients).
Adversary Model

- Logic flaws in Integration
- Service provider is trusted
- Client is not trusted
Contribution

• Integuard
  ➢ First step toward automatic and generic protection of Web service integrations.
  ➢ New challenges in multiple-party settings.
  ➢ Effective false positive control.
  ➢ Evaluate with real exploits and performance test.
  ➢ Practical protection.
DESIGN
Design – Architecture

InteGuard

- Traces
  - Trace Collector
    - Labeled traces
  - Security Policy Generator
    - FSMs

Integrator
- HTTP Messages incoming
- HTTP Messages generated

Proxy and ICAP
- HTTP Messages incoming
- HTTP Messages generated

Provider

Client
Design – Architecture

Client

Provider

Integuard

RT2

RT3

RT1

RT4
Design – Training Trace Collection

• Traces \(\rightarrow\) Security Invariants
• Challenge
  ➢ Random transactions for invariant extraction
  \(\rightarrow\) False Positive
Design – Training Trace Collection

• Four traces
• Integrator opens two accounts at service provider, e.g. open two PayPal Merchant accounts

Integrator-1
(Merchant Account - 1)

Transaction 1

Transaction 2

Integrator-2
(Merchant Account – 2)

Different users
Different transactions
Different quantities
Same integrator
Different merchant accounts

Integrator 1 and Integrator 2 are the same Web application configured with different merchant accounts
3. Design – Training Trace Collection

- Four traces

**Integrator-1** (Merchant Account - 1)
  - Transaction-1
  - Transaction-2

**Integrator-2** (Merchant Account - 2)
  - Transaction-1'
  - Transaction-2'

- Same user
- Same products
- Same transactions
- Same quantity
- Same address
- Different integrators
- ...

3. Design
Design – Training Trace Collection

- Four traces

**Integrator-1**
(Merchant Account - 1)
- Transaction-1
- Transaction-2

**Integrator-2**
(Merchant Account – 2)
- Transaction-1'
- Transaction-2'

Same Transactions
Different Integrators
Design – Training Trace Collection

- Four traces

**Integrator-1** (Merchant Account - 1)
  - Transaction-1
  - Transaction-2

**Integrator-2** (Merchant Account – 2)
  - Transaction-1'
  - Transaction-2'

**Differential Analysis**
Design – Invariant Analysis

• Integrator-specific invariant
• Local Invariant
  ➢ Transaction-specific invariant
• Other invariant
  ➢ Start of transaction
  ➢ End of transaction
  ➢ API sequence
3. Design

Design – Invariant Analysis

- Integrator-specific invariant
- Local Invariant
  - Transaction-specific invariant
- Other invariant
  - Start of transaction
  - End of transaction
  - API sequence
3. Design

Integrator-Specific Invariant:

- RT3: payeeEmail = jeff@email.com & ...
- RT3: payeeEmail = alice@email.com & ...

Specific to Integrator 1:

- jeff@email.com

Specific to Integrator 2:

- alice@email.com

RT3.payeeEmail
3. Design

Drop such Invariant

payMethod = creditCard
Design – Invariant Analysis

- Integrator-specific invariant
- **Local Invariant**
  - Transaction-specific invariant
- Other invariant
  - Start of transaction
  - End of transaction
  - API sequence
3. Design

Local Invariant:

amount == gross

RT1: amount = 9.99 & ...
RT3: gross = 9.99 & ...

RT1: amount = 13.99 & ...
RT3: gross = 13.99 & ...

Local Invariant:

amount == gross

RT1: amount = 9.99 & ...
RT3: gross = 9.99 & ...

RT1: amount = 13.99 & ...
RT3: gross = 13.99 & ...
3. Design

Drop Invariant
With
Length < 3

\[
\begin{align*}
\text{RT1: } & \text{returnFlag} = 1 \& \ldots \\
\text{RT3: } & \text{status} = 1 \& \ldots \\
\end{align*}
\]

\[
\begin{align*}
\text{RT1: } & \text{returnFlag} = 1 \& \ldots \\
\text{RT3: } & \text{status} = 1 \& \ldots \\
\end{align*}
\]

\[
\begin{align*}
\text{RT1: } & \text{returnFlag} = 1 \& \ldots \\
\text{RT3: } & \text{status} = 1 \& \ldots \\
\end{align*}
\]

\[
\begin{align*}
\text{RT1: } & \text{returnFlag} = 1 \& \ldots \\
\text{RT3: } & \text{status} = 1 \& \ldots \\
\end{align*}
\]

returnFlag = status
Design – Invariant Analysis

• Integrator-specific invariant
• Local Invariant
  ➢ Transaction-specific invariant
• Other invariant
  ➢ Start of transaction
  ➢ End of transaction
  ➢ API sequence
3. Design

\[
\begin{align*}
\text{transactionID} &= 0519 & \text{orderID} &= 0519 \\
\text{transactionID} &= 0520 & \text{orderID} &= 0520 \\
\text{transactionID} &= 0521 & \text{orderID} &= 0521 \\
\text{transactionID} &= 0522 & \text{orderID} &= 0522
\end{align*}
\]
Specific to each transaction

3. Design
3. Design

Local Invariant

Transaction specific invariant
3. Design

Same transactions
Different integrators
amount = gross
3. Design

Transaction-specific Invariant

Is RT4 different?
Transaction-specific Invariant

RT4 has no cookies
Transaction-specific Invariant

Which transaction does a RT4 belong to?
Transaction-specific Invariant

Transaction-specific Invariants help
Grouping RT4 into its belonging transaction
Design – Invariant Analysis

- Local Invariant
- Integrator-specific invariant
  - Transaction-specific invariant
- Other invariant
  - Start of transaction
  - End of transaction
  - API sequence
Design – Element Extraction

• Challenges
  ➢ RT2 not observable
  ➢ RT2 parameters in RT1’s response
  ➢ **Channels**: HTTP 3xx, meta refresh, HTML Form, JavaScript, JSON, XML

![Diagram showing client, provider, integrator, and RT1 and RT2 with red cross indicating Integuard solution]

3. Design
Design – Element Extraction

• Challenges
  ➢ RT2 not observable
  ➢ RT2 parameters in RT1’s response
  ➢ **Channels**: HTTP 3xx, meta refresh, HTML Form, JavaScript, JSON, XML

![Diagram of Redirection with Integuard and Providers/Integrators]
3. Design

From RT1, extract RT2's parameters in its response. Record the DOM locations for each parameter.

- RT2 parameters in RT1's response:

  Request of RT2:
  POST https://PayPal/pay?
  accountId=MULW&amount=9.99&orderId=0519 &...

  Trace parameters in RT1's response:
  HTML form in RT1's response:
  <form id="simplePay" name="SimplePay" method="POST"
  action="https://PayPal/pay">
  <input name="AccountId" value="MULW">
  <input name="amount" value="9.99">
  <input name="orderId" value="0519">... </form>

  Extract a DOM path for each parameter:
  
  Extracted DOM path:
  
  **AccountId**: form[id,name,action]->inputTag[AccountId]
  **Amount**: form[id,name,action]->inputTag[amount]
  **orderId**: form[id,name,action]->inputTag[orderId]
Design – Element Extraction

Request of RT2:
POST https://PayPal/pay?
accountID=MULW&amount=9.99&orderID=0519 &...

Trace parameters in RT1's response

HTML form in RT1's response:
<form id="simplepay" name="SimplePay" method="POST"
action="https://PayPal/pay">
<input name="AccountID" value="MULW">
<input name="amount" value="9.99">
<input name="orderID" value="0519">...
</form>

Extract a DOM path for each parameter

Extracted DOM path:
AccountID: form[id,name,action]->inputTag[AccountID]
Amount: form[id,name,action]->inputTag[amount]
orderID: form[id,name,action]->inputTag[orderID]

Don’t parse all content of RT1
Design – Element Extraction

**Request of RT2:**
POST https://PayPal/pay?
accountid=MULW&amount=9.99&orderID=0519 &...

Trace parameters in RT1’s response

**HTML form in RT1’s response:**
<form id="simplepay" name="SimplePay" method="POST"
action="https://PayPal/pay">
<input name="AccountId" value="MULW">
<input name="amount" value="9.99">
<input name="orderID" value="0519">...
</form>

Extract a DOM path for each parameter

**Extracted DOM path:**
AccountId: form[id,name,action]->inputTag[AccountId]
Amount: form[id,name,action]->inputTag[amount]
orderID: form[id,name,action]->inputTag[orderID]

**Just extract desired parameters from known locations**

3. Design
Design – Element Extraction

- HTTP 3xx
- Meta refresh
- **JavaScript**
  - Abstract Syntax Tree (AST)
- JSON
- XML
Design – Element Extraction

- RT2’s parameters in RT1’s response
  - HTTP 3xx
  - Meta refresh
  - JavaScript
  - JSON
  - XML
Design – Element Extraction

- **JavaScript**
  - Abstract Syntax Tree (AST)
  - Mark parameters’ locations
- **JSON, XML**
  - Tree structure, mark locations
Design – Security Policy Enforcement

- Security invariants.
- **Intercept** HTTP traffic on integrator.
- **Runtime** detection of invariant.

![Diagram showing the flow of communication between Client, Provider, Integrator, and Integuard with RT2, RT3, RT4, and RT1 connections.](image-url)
EVALUATION
Evaluation

• Integrations
  ➢ Web Shopping Cart applications with known vulnerabilities.
    o Intersipre starter edition 5.5.4
    o Nopcommerce v1.60
  ➢ 5 faulty SSO integrations.
    o involving sears.com, janrain.com, Google, Facebook, PayPal
### Effectiveness

<table>
<thead>
<tr>
<th>Application</th>
<th>Service Integrated</th>
<th>Invariant type</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nopcommerce</td>
<td>PayPal Std</td>
<td>Local</td>
<td>Yes</td>
</tr>
<tr>
<td>Nopcommerce</td>
<td>Amazon Simple Pay</td>
<td>Integrator-specific</td>
<td>Yes</td>
</tr>
<tr>
<td>Nopcommerce</td>
<td>Amazon Simple Pay</td>
<td>Integrator-specific</td>
<td>Yes</td>
</tr>
<tr>
<td>Interspire</td>
<td>PayPal Std</td>
<td>Transaction-specific</td>
<td>Yes</td>
</tr>
<tr>
<td>Interspire</td>
<td>PayPal Exp</td>
<td>Local</td>
<td>Yes</td>
</tr>
<tr>
<td>Interspire</td>
<td>Google Checkout</td>
<td>API Sequence</td>
<td>Yes</td>
</tr>
<tr>
<td>Smartsheet.com</td>
<td>Google ID</td>
<td>Local</td>
<td>Yes</td>
</tr>
<tr>
<td>Janrain</td>
<td>Google ID</td>
<td>Local</td>
<td>Yes</td>
</tr>
<tr>
<td>Sears.com</td>
<td>Facebook SSO</td>
<td>Integrator-specific</td>
<td>Yes</td>
</tr>
<tr>
<td>Shopgecko.com</td>
<td>PayPal Access</td>
<td>Local</td>
<td>Yes</td>
</tr>
<tr>
<td>Farmville</td>
<td>Facebook SSO</td>
<td>N/A</td>
<td>No</td>
</tr>
</tbody>
</table>

4. Evaluation
False Positives

- Each CaaS integration, 100 to 300 checkouts.
- Each SSO integration, 20 checkouts.
- Altogether 1,000 real transactions.
  - Random user behaviors, clicking back button, returning through old URLs, etc.
  - Randomly crawl URLs.

→ No false alarms
Performance

• 32 to 256 (default MaxClients of Apache Web server) concurrent transactions.
• Negligible overhead (3.32%).
• Memory:
  ➢ Almost constant 1,250 MB.
    (32 to 256 concurrency)
  ➢ 150MB difference.
    (256 concurrency, with and without security check)
CONCLUSION
Conclusion

- First to protect vulnerable integrations of third-party Web services.
- New challenges in multi-party settings.
- Generate invariants through a suit of new techniques.
- Effective false positive control and low performance expense.
THANK YOU!

LUYI XING
ICAP

- The **Internet Content Adaptation Protocol** (ICAP) is a lightweight HTTP-like protocol which is used to extend transparent **proxy servers**. ICAP is generally used to implement **virus scanning** and **content filters** (including **censorware**) in transparent HTTP proxy caches.