Web Logic Vulnerability

By Eric Jizba and Yan Chen

With slides from Fangqi Sun and Giancarlo Pellegrino
Outline

- Background and Motivation
- Related Work
- Whitebox approach
  - Detecting Logic Vulnerabilities in E-Commerce Applications
    - Fangqi Sun, Liang Xu, Zhendong Su
- Blackbox approach
  - Toward Black-box Detection of Logic Flaws in Web Applications
    - Giancarlo Pellegrino, Davide Balzarotti
Background and motivation
Logic Flaws

- Also known as design flaws/errors, business/application logic errors/flaws
- Lack a formal definition
  - CWE-ID 840: Business logic errors are “weaknesses [...] that commonly allow attackers to manipulate the business logic of an application”
- Mainly caused by insufficient validation of the application workflow and data flow
- Can exhibit patterns, e.g.
  - Improper authentication/authorization
Logic Vulnerabilities in E-Commerce Web Applications

- Third-party cashiers
  - Bridge the trustiness gap between customers and merchants
  - Complicate logic flows during checkout

- Logic vulnerabilities in e-commerce web applications
  - Abuse application-specific functionality
  - Allow attackers to purchase products or services with incorrect or no payment
  - Have multiple attack vectors
    - Assumptions of *user inputs* and *user actions* should be explicitly checked
  - Example
    - CVE-2009-2039 is reported for Luottokunta (v1.2) but the patched Luottokunta (v1.3) is still vulnerable
**Attack on Currency**

1. Order initialization
2. Payment of order total in currency for order ID to merchant ID
3. Order confirmation

Consistent status? **NO**

**Currency:**
- **GBP (British Pound Sterling):** £6.25 (equals $10.43)
- **USD (US Dollar):** $6.25

**RBS WorldPay**
Current order (ID 1002) has been paid

Payment tokens for order ID 1001 can be replayed for future orders

Received payment for order ID 1001 only

---

1. Order initialization
2. Payment of order total in currency for order ID to merchant ID
3. Order confirmation

Consistent status? NO

orderID = 1002

orderID = 1001
Attack on Merchant ID

1. Order initialization
2. Payment of order total in currency for order ID to merchant ID
3. Order confirmation

merchantID = chocolateDelight

Consistent status? NO

merchantID = attackerAlice

Payee is chocolateDelight

Payee is attackerAlice

Setting up a PayPal merchant account for Alice is easy
Related Work
### Problem

<table>
<thead>
<tr>
<th>Source code</th>
<th>Explicit Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes: White-box</td>
</tr>
<tr>
<td></td>
<td>No: White-box</td>
</tr>
<tr>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

- **White-box testing**  
  [BalzarottiCCS07, FelmetsgerUSENIX10, ...]  
  - Source code of WA may not be available → White-box not applicable!
### Problem

<table>
<thead>
<tr>
<th>Source code</th>
<th>Explicit Documentation</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>White-box</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td><strong>White-box</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design verification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **White-box testing** [BalzarottiCCS07, FelmetsgerUSENIX10, ...]
  - Source code of WA may not be available $\rightarrow$ White-box not applicable!

- **Design verification** [LoweCSF97, ArmandoCSF07, ...]
  - Specification of WA may not be available $\rightarrow$ DV not applicable!
Problem

- **White-box testing**  [BalzarottiCCS07, FelmetsgerUSENIX10, ...]
  - Source code of WA may not be available → White-box not applicable!

- **Design verification**  [LoweCSF97, ArmandoCSF07, ...]
  - Specification of WA may not be available → DV not applicable!

- **Black-box testing, e.g., web scanners**  [DoupêDIMVA10, WangS&P11, WangS&P12]
  - Cannot automatically detect logic flaws

➤ **Testing for logic flaws is done manually**
## Comparing testing methods

<table>
<thead>
<tr>
<th></th>
<th>Whitebox</th>
<th>Blackbox</th>
<th>Design Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>Fair</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Scalability</td>
<td>Fair</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Fair</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Requires source code</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Requires app specification</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Daniel Fett, Ralf Küsters, and Guido Schmitz created expressive model for web infrastructure
- Manual analysis: more comprehensive and accurate

Discovered logic vulnerability in BrowserID
- Allows attacker to sign-in to any service that supports BrowserID using the email address of any user without know their credentials
- Proposed fix adopted by Mozilla simply involves verifying the email address is correct
Third party APIs are more and more popular

SSO, Cashier Services, Maps, Search, etc.

Key Insight: most web APIs require small number of simple input parameters that are usually set by the user

InteGuard looks at web traffic between the app and third part service to analyze invariants (e.g. orderID, price)

Does not require source code

Mostly automatic

Cannot handle more complex invariant relations (such as the relation between signed content and its signature)
Blackbox Approach
Parameter Pollution Vulnerabilities

- Common attack vector used in Logic Vulnerabilities
  - E.g. using the same OrderID for two transactions

- Different from workflow attack vector (also used in Logic Vulnerabilities)
  - E.g. bypassing a required page in a payment application

- NoTamper detects insufficient server-side validations where the server fails to replicate validations on the client side

- PAPAS uses a blackbox scanning technique for vulnerable parameters
Whitebox Approach

Basic Problem
Key Challenge

- Logic vulnerabilities in e-commerce web applications are application-specific
  - Thorough code review of all possible logic flows is non-trivial
  - Various application-specific logic flows, cashier APIs and security checks make automated detection difficult

- Key challenge of automated detection

  The lack of a general and precise notion of correct payment logic
Key Insight

- A common invariant for automated detection

A checkout is secure when it guarantees the **integrity** and **authenticity** of critical payment status (order ID, order total, merchant ID and currency)
Whitebox Approach

Main Ideas
Our Approach

- A symbolic execution framework that explores critical control flows exhaustively

- Tracking taint annotations across checkout nodes
  - Payment status
  - Exposed signed token (signed with a cashier-merchant secret)

Logic flow: 
\((n_i, Q_i) \rightarrow (n_f, Q_f)\)
Taint Removal Rules

- Conditional checks of (in)equality
  - When an untrusted value is verified against a trusted one
  - Example of removing taint from order total
    \[
    \text{md5(SECRET . \$_SESSION[\text{"order"}] \rightarrow \text{info[\text{"total"}]}) == md5(\text{SECRET . \$_GET[\text{"oTotal"}]})}
    \]

- Writes to merchant databases
  - When an untrusted value is included in an INSERT/UPDATE query
  - Merchant employee can easily spot tampered values

- Secure communication channels (merchant-to-cashier cURL requests)
  - Remove taint from order ID, order total, merchant ID or currency
  when such components are present in request parameters
Taint Addition Rule

- Add an exposed signed token when used in a conditional check of a cashier-to-merchant request
  - Security by obscurity is insufficient

- Example
  - Hidden HTML form element: $md5($secret . $orderId . $orderTotal)
  - $_GET['hash'] == $md5($secret . $_GET['old'] . $_GET['oTotal'])
  - This exposed signed token $md5($secret . $orderId . $orderTotal) nullifies checks on order ID and order total
Vulnerability Detection Example

- R1. User → Merchant(checkoutConfirmation.php)
  - Symbolic HTML form contains two URLs: cashier URL and return URL(checkoutProcess.php).

- R2. User → Cashier(https://dmp2.luottokunta.fi)
  - Modeling cashier as trusted black box

- R3. User → Merchant(checkoutProcess.php), redirection
  - Representing all possible cashier responses with symbolic inputs

- R4. User → Merchant(checkoutSuccess.php), redirection
  - Analyzing logic states at this destination node (end of checkout) to detect logic vulnerabilities

Luottokunta (v1.3)
Whitebox Approach

Evaluation and Results
Evaluation

Subjects: 22 unique payment modules of osCommerce
- More than 14,000 registered websites, 928 payment modules, 13 years of history (osCommerce v2.3)
- 20 out of 46 default modules with distinct CFGs
- 2 Luottokunta payment modules (v1.2 & v1.3)

Metrics
- Effectiveness: Detected 12 logic vulnerabilities (11 new) with no false positives
- Performance
## Logic Vulnerability Analysis Results

<table>
<thead>
<tr>
<th>Payment Module</th>
<th>Safe</th>
<th>Payment Module</th>
<th>Safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>2Checkout</td>
<td>✗</td>
<td>PayPal Pro - Direct Payments</td>
<td>✓</td>
</tr>
<tr>
<td>Authorize.net CC AIM</td>
<td>✓</td>
<td>PayPal (Payflow) - Direct Payments</td>
<td>✓</td>
</tr>
<tr>
<td>Authorize.net CC SIM</td>
<td>✗</td>
<td>PayPal (Payflow) - Express Checkout</td>
<td>✓</td>
</tr>
<tr>
<td>ChronoPay</td>
<td>✗</td>
<td>PayPal Standard</td>
<td>✗</td>
</tr>
<tr>
<td>inpay</td>
<td>✓</td>
<td>PayPoint.net SECPay</td>
<td>✗</td>
</tr>
<tr>
<td>iPaiyment (Credit Card)</td>
<td>✗</td>
<td>PSiGate</td>
<td>✗</td>
</tr>
<tr>
<td>Luottokunta (v1.2)</td>
<td>✗</td>
<td>RBS WorldPay Hosted</td>
<td>✗</td>
</tr>
<tr>
<td>Luottokunta (v1.3)</td>
<td>✗</td>
<td>Sage Pay Direct</td>
<td>✓</td>
</tr>
<tr>
<td>Moneybookers</td>
<td>✓</td>
<td>Sage Pay Form</td>
<td>✗</td>
</tr>
<tr>
<td>NOCHEX</td>
<td>✗</td>
<td>Sage Pay Server</td>
<td>✓</td>
</tr>
<tr>
<td>PayPal Express</td>
<td>✓</td>
<td>Sofortüberweisung Direkt</td>
<td>✓*</td>
</tr>
</tbody>
</table>
## Taint Annotations of 12 Vulnerable Payment Modules

<table>
<thead>
<tr>
<th>Payment Module</th>
<th>Order Id</th>
<th>Order Total</th>
<th>Merchant Id</th>
<th>Currency</th>
<th>Signed Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>2Checkout</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Authorize.net SIM</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ChronoPay</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>iPayment (Credit card)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luottokunta (v1.2)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Luottokunta (v1.3)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>NOCHEX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PayPal Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PayPoint.net SECPay</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PSiGate</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RBS WorldPay Hosted</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Sage Pay Form</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>
## Performance Results of 12 Vulnerable Payment Modules

<table>
<thead>
<tr>
<th>Payment Module</th>
<th>Files</th>
<th>Nodes</th>
<th>Edges</th>
<th>Stmts</th>
<th>States</th>
<th>Flows</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2Checkout</td>
<td>105</td>
<td>5,194</td>
<td>6,176</td>
<td>8,385</td>
<td>40</td>
<td>4</td>
<td>16.04</td>
</tr>
<tr>
<td>Authorize.net SIM</td>
<td>105</td>
<td>5,221</td>
<td>6,221</td>
<td>8,435</td>
<td>46</td>
<td>4</td>
<td>16.89</td>
</tr>
<tr>
<td>ChronoPay</td>
<td>99</td>
<td>5,013</td>
<td>5,969</td>
<td>8,084</td>
<td>69</td>
<td>5</td>
<td>31.51</td>
</tr>
<tr>
<td>iPayment (Credit card)</td>
<td>99</td>
<td>4,999</td>
<td>5,932</td>
<td>7,918</td>
<td>38</td>
<td>5</td>
<td>21.86</td>
</tr>
<tr>
<td>Luottokunta (v1.2)</td>
<td>105</td>
<td>5,158</td>
<td>6,127</td>
<td>8,291</td>
<td>34</td>
<td>4</td>
<td>15.33</td>
</tr>
<tr>
<td>Luottokunta (v1.3)</td>
<td>105</td>
<td>5,164</td>
<td>6,135</td>
<td>8,308</td>
<td>35</td>
<td>4</td>
<td>15.33</td>
</tr>
<tr>
<td>NOCHEX</td>
<td>105</td>
<td>5,145</td>
<td>6,111</td>
<td>8,237</td>
<td>33</td>
<td>4</td>
<td>15.03</td>
</tr>
<tr>
<td>PayPal Standard</td>
<td>99</td>
<td>5,040</td>
<td>6,006</td>
<td>8,170</td>
<td>68</td>
<td>6</td>
<td>33.01</td>
</tr>
<tr>
<td>PayPoint.net SECPay</td>
<td>105</td>
<td>5,174</td>
<td>6,152</td>
<td>8,332</td>
<td>40</td>
<td>4</td>
<td>15.80</td>
</tr>
<tr>
<td>PSigate</td>
<td>106</td>
<td>5,231</td>
<td>6,228</td>
<td>8,436</td>
<td>44</td>
<td>4</td>
<td>16.82</td>
</tr>
<tr>
<td>RBS WorldPay Hosted</td>
<td>99</td>
<td>5,019</td>
<td>5,977</td>
<td>8,121</td>
<td>79</td>
<td>5</td>
<td>36.12</td>
</tr>
<tr>
<td>Sage Pay Form</td>
<td>106</td>
<td>5,315</td>
<td>6,329</td>
<td>8,762</td>
<td>55</td>
<td>4</td>
<td>19.96</td>
</tr>
<tr>
<td><strong>Average of 22</strong></td>
<td>102.73</td>
<td>5,173</td>
<td>6,162</td>
<td>8,376</td>
<td>67.27</td>
<td>5.05</td>
<td>31.43</td>
</tr>
</tbody>
</table>
Conclusion

- First static detection of logic vulnerabilities in e-commerce applications
  - Based on an application-independent invariant
  - A scalable symbolic execution framework for PHP applications, incorporating taint tracking of payment status

- Three responsible proof-of-concept experiments on live websites

- Evaluated our tool on 22 unique payment modules and detected 12 logic vulnerabilities (11 are new)
Open Issues

- Cannot identify all logic vulnerabilities
- Does not support JavaScript analysis
- Limited analysis of dynamic language features
Questions?
Blackbox Approach

Basic Problem
Blackbox Approach

Main Ideas
Overview

1) Model Inference

74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

Oracle

Verdict:
Flaw found in test 1 and 2

Execution

Test Cases
1) Model Inference

74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

Oracle

Flaw found in test 1 and 2

Verdict:
Behavioral Patterns Extraction

1) Model Inference

74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89

Verdict: Flaw found in test 1 and 2
Traces:
\[ \pi_1 = \langle a, b, a, c, d, e, f, e \rangle \]
\[ \pi_2 = \langle a, c, d, e, f, e \rangle \]
Workflow Patterns

Traces:

\[ \pi_1 = \langle a, b, a, c, d, e, f, e \rangle \]
\[ \pi_2 = \langle a, c, d, e, f, e \rangle \]

Model:

- TrWP: Trace Waypoints
- Rp: Repeatable Operations
Data flow Patterns

Trace 1:

   <HTML>
   <a href="/view.php?tok=8AFFB0">[...]

   <HTML>
   <a href="/add.php?tok=8AFFB0">[...]

   <HTML>
   <a href="/checkout">[...]

Trace 2:

   <HTML>
   <a href="/add.php?tok=DDA124">[...]

   <HTML>
   <a href="/checkout">[...]
Test Case Generation

1) Model Inference

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

- Test Cases Generation
  - Resource Abstraction
  - Resource Clustering
- Data flow Patterns
- Workflow Patterns
- PChain 1
- PChain 2
- MWP
- Rp
- TrWP
- St
- Execution
- Oracle

Verdict: Flaw found in test 1 and 2
Attack Pattern-based Test Case Generation

Multiple Execution of Repeatable Singletons

Breaking Multi-Steps Operations

Breaking Server-Generated Propagation Chains

Waypoints Detour
Attack Pattern-based Test Case Generation

![Diagram showing breaking server-generated propagation chains]
Test Case Execution and Oracle

1) Model Inference

74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

Verdict: Flaw found in test 1 and 2
Test Case Execution and Oracle

1) Model Inference

Security Property:

\[ \text{ord}_{\text{placed}} \land \text{onStore}(S) \implies (\Box (\text{paid}(U, I) \land \text{toStore}(S)) \land \Box (\text{ack}(U, I) \land \text{onStore}(S))) \]

2) Behavioral Patterns

3) Test Cases Generation

4) Test Cases Execution

Verdict: Flaw found in test 1 and 2
Case Study: Shopping Cart Web Applications

Customers → Order → Online Store

Cashier-as-a-Service

Pay → Trace Collection → Bank

Online Store

Cashier-as-a-Service

PayPal

PrestaShop

TomatoCart

AbanteCart.com

osCommerce

Magento

OpenCart
Blackbox Approach

Evaluation and Results
Experiments and Results

- Target: 7 popular eCommerce Web Applications
  - Deployed by >13M online stores

- Testbed: created 12 Paypal sandbox configurations

In total 3,145 test cases
Experiments and Results

- **Target:** 7 popular eCommerce Web Applications
  - Deployed by >13M online stores

- **Testbed:** created 12 Paypal sandbox configurations
  
  1,253 "misuse" detected

  In total **3,145** test cases
  
  1,892 were executed
Experiments and Results

- Target: 7 popular eCommerce Web Applications
  - Deployed by >13M online stores

- Testbed: created 12 Paypal sandbox configurations

In total 3,145 test cases
- 1,253 “misuse” detected
- 1,892 were executed
- 983 not violations
- 909 violation
Experiments and Results

- **Target:** 7 popular eCommerce Web Applications
  - Deployed by >13M online stores

- **Testbed:** created 12 Paypal sandbox configurations

- In total **3,145** test cases
  - **1,253** “misuse” detected
  - **1,892** were executed
  - **983** not violations
    - **849** caused by bugs in the GUI
    - **90** caused by logic flaws
  - **909** violation
    - **90** caused by logic flaws
10 previously-unknown vulnerabilities

- Allowing to shop for free or pay less

<table>
<thead>
<tr>
<th>Application</th>
<th>Shop for free</th>
<th>Pay less</th>
<th>Session Fixation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbanteCart</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magento</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCart</td>
<td></td>
<td>x x</td>
<td></td>
</tr>
<tr>
<td>osCommerce</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>PrestaShop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TomatoCart</td>
<td>x</td>
<td>x x</td>
<td></td>
</tr>
<tr>
<td>CS-Cart</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CVE-2012-2991
CVE-2012-4934
CVE-2013-0118
Conclusion

- Proposed a black-box technique to detect logic flaws in web applications

- Combined passive model inference and attacker pattern-based test case generation

- Developed a prototype
  - assessed against 7 popular eCommerce web applications

- Discovered 10 previously-unknown logic flaws
  - allow an attacker to shop for free or pay less
Open Issues

- Only tests attacks through data flow and workflow
  - E.g. does not test unauthorized access to resources
- Automation favors efficiency over coverage
Questions?