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

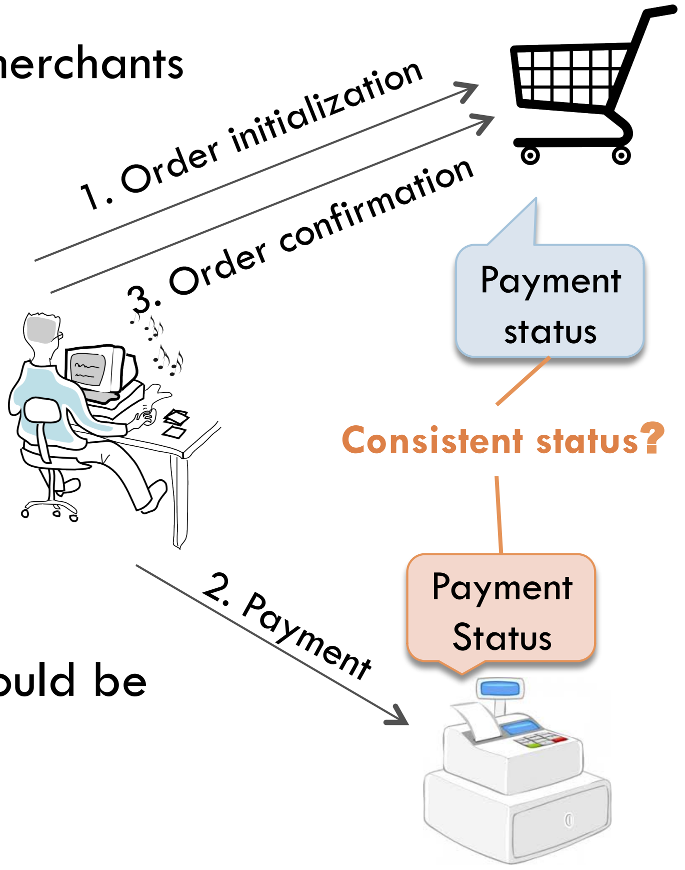
Background and motivation

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



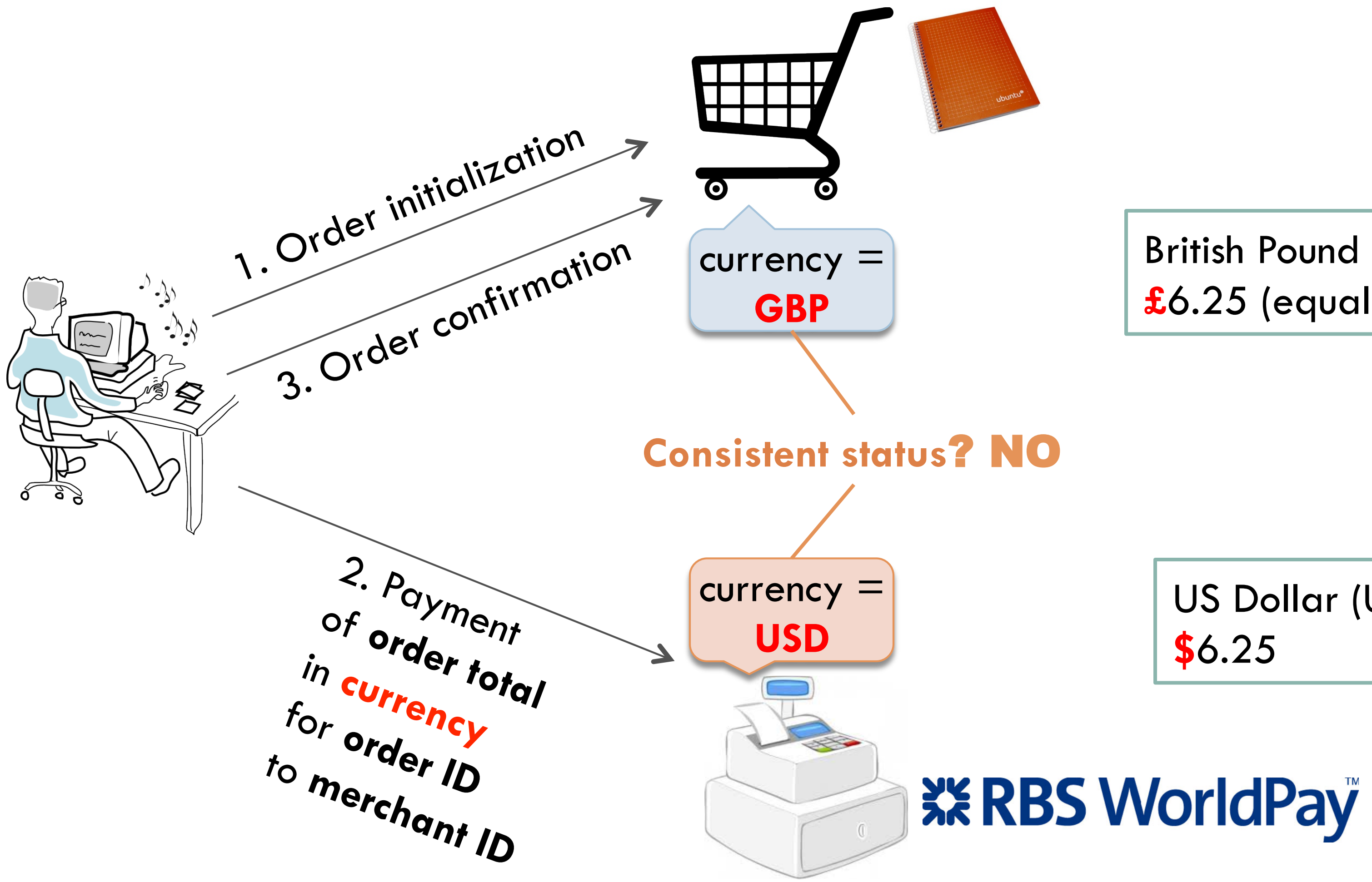
RBS WorldPay™

Authorize.Net®
a CyberSource solution

PayPal™

oscommerce

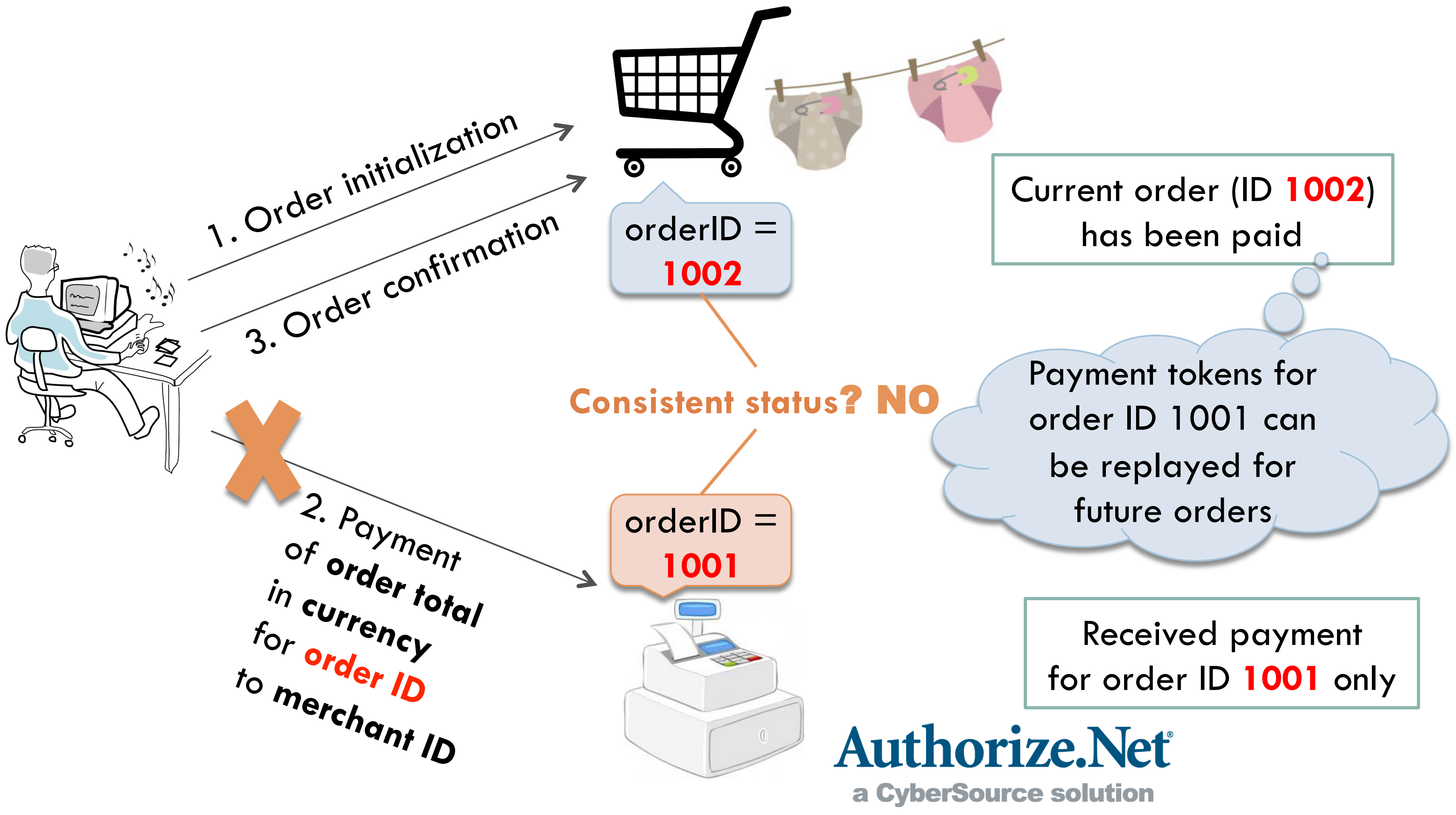
Attack on Currency



British Pound Sterling (GBP)
£6.25 (equals $10.43)

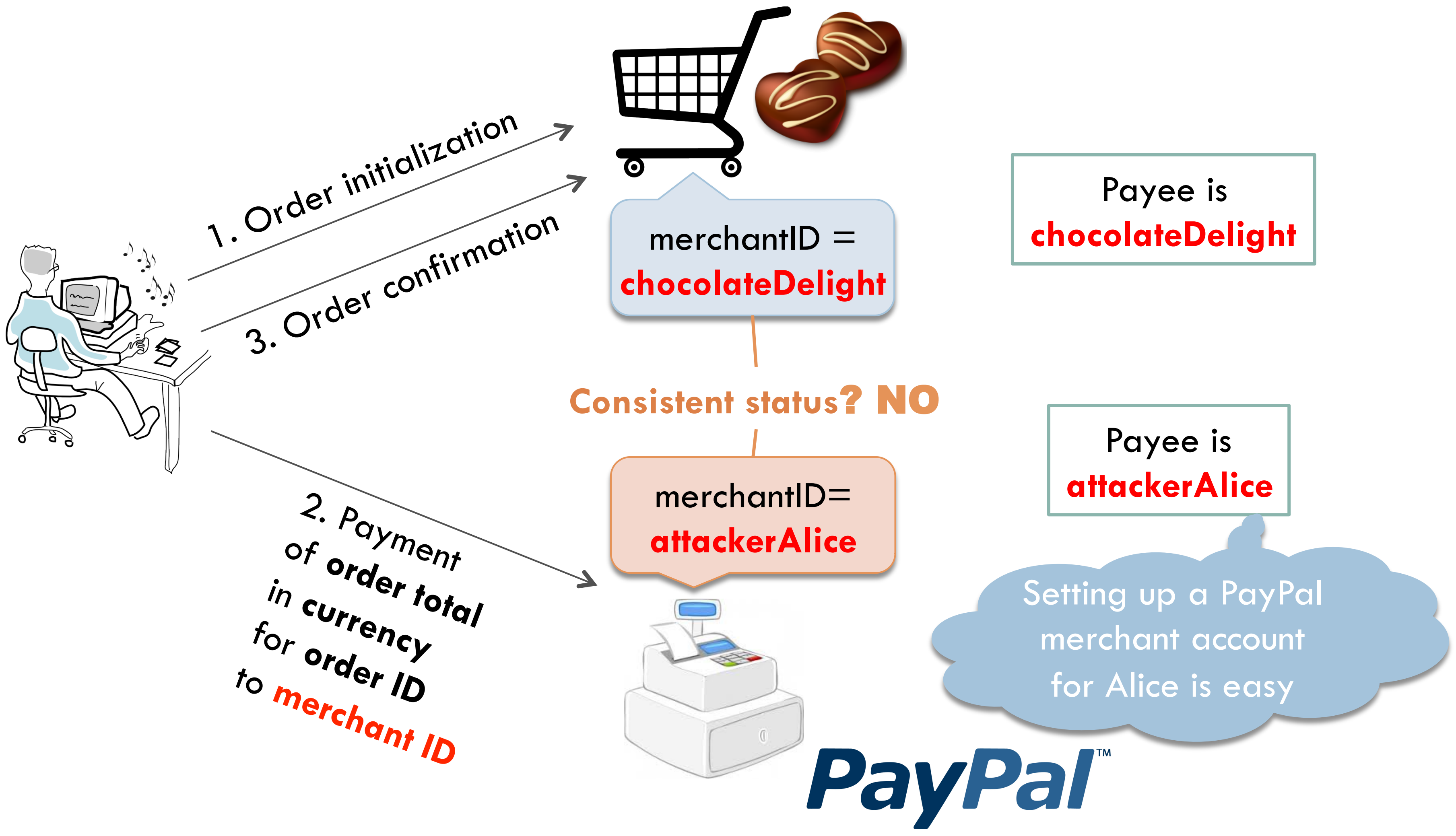
US Dollar (USD)
$6.25

Attack on Order ID



1. Order initialization
3. Order confirmation
2. Payment of **order total** in **currency** for **order ID** to **merchant ID**
orderID = 1002
orderID = 1001
Consistent status? NO
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
Authorize.Net
a CyberSource solution

Attack on Merchant ID



1. Order initialization
3. Order confirmation
merchantID = chocolateDelight
Payee is chocolateDelight
Consistent status? NO
2. Payment of order total in currency for order ID to merchant ID
merchantID= attackerAlice
Payee is attackerAlice
Setting up a PayPal merchant account for Alice is easy
PayPal™

Related Work

Problem

		Explicit Documentation	
		Yes	No
Source code	Yes	White-box	White-box
	No		

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

Problem

		Explicit Documentation	
		Yes	No
Source code	Yes	White-box Design verification	White-box
	No	Design verification	

- 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!

Problem

		Explicit Documentation	
		Yes	No
Source code	Yes	Black-box White-box Design verification	Black-box White-box
	No	Black-box Design verification	Black-box

- 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 [DoupèDIMVA10, WangS&P11, WangS&P12]
  - Cannot <u>automatically</u> detect logic flaws

➔ ***Testing for logic flaws is done manually***

Comparing testing methods

	Whitebox	Blackbox	Design Verification
Coverage	Fair	Poor	Good
Scalability	Fair	Good	Poor
Efficiency	Fair	Good	Poor
Requires source code	Yes	No	No
Requires app specification	No	No	Yes

Design Verification
BrowserID SSO Analysis

- 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

Blackbox Approach
InteGuard: Web Service Integration Security

- 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

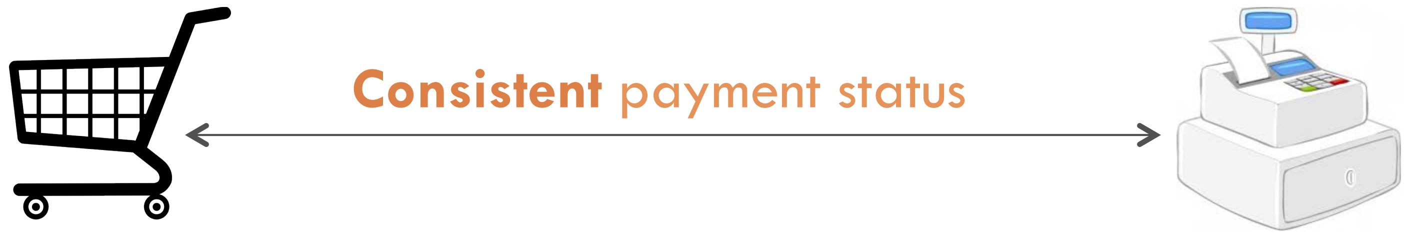
- Key challenge of automated detection

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)



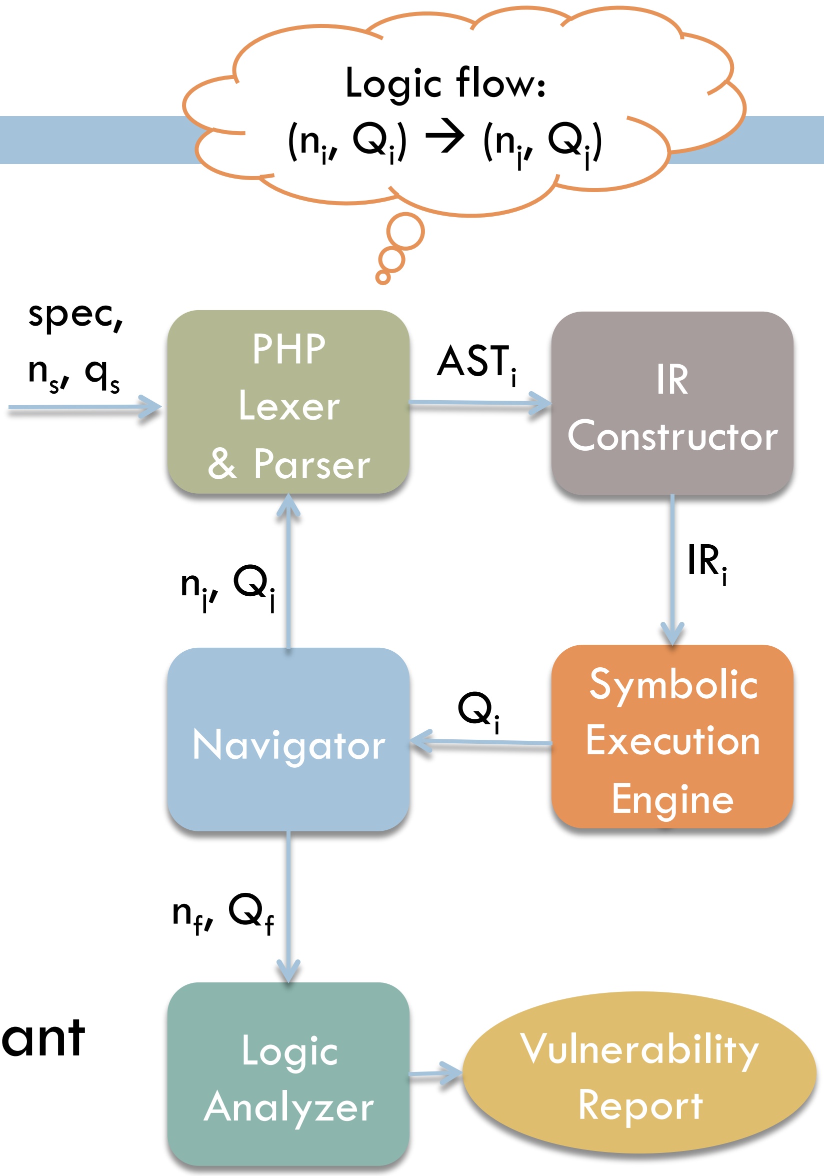
Whitebox Approach

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)



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

```
md5(SECRET . $_SESSION['order']→info['total']) == md5(SECRET . $_GET['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

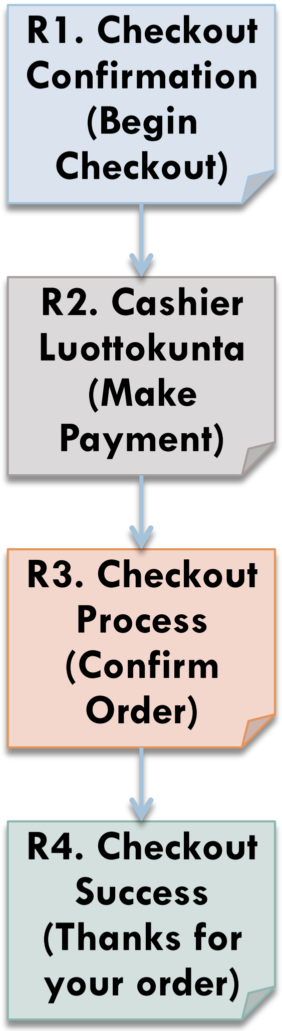
- Add an exposed signed token when used in a conditional check of a cashier-to-merchant request
  - Security by obscurity is insufficient

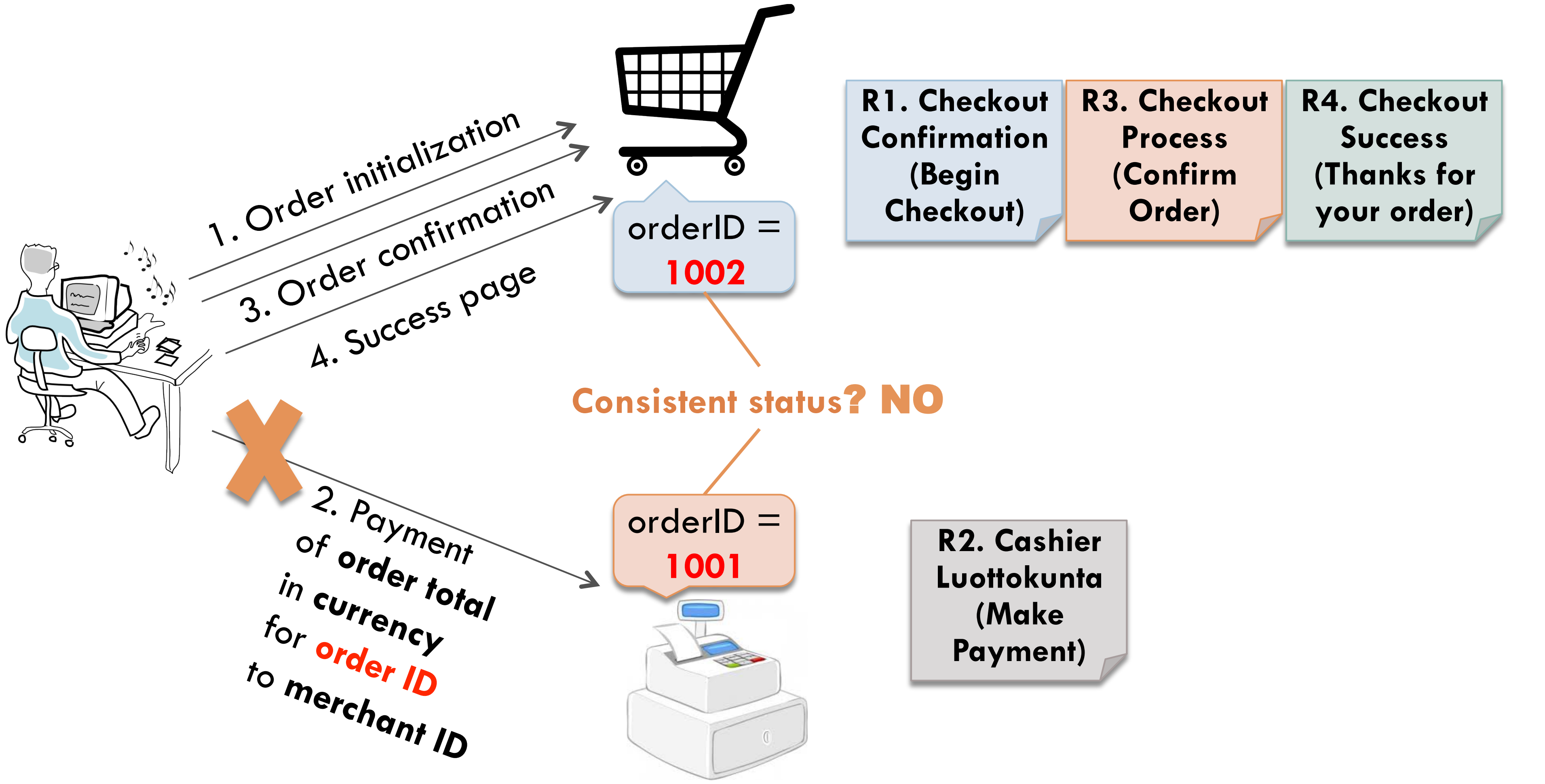
- Example
  - Hidden HTML form element: md5($secret . $orderId . $orderTotal)
  - $_GET['hash'] == md5($secret . $_GET['oId'] . $_GET['oTotal'])
  - This exposed signed token md5($secret . $orderId . $orderTotal) nullifies checks on order ID and order total

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)


R1. Checkout Confirmation (Begin Checkout)
R2. Cashier Luottokunta (Make Payment)
R3. Checkout Process (Confirm Order)
R4. Checkout Success (Thanks for your order)


1. Order initialization
3. Order confirmation
4. Success page
orderID = 1002
R1. Checkout Confirmation (Begin Checkout)
R3. Checkout Process (Confirm Order)
R4. Checkout Success (Thanks for your order)
Consistent status? NO
2. Payment of order total in currency for order ID to merchant ID
orderID = 1001
R2. Cashier Luottokunta (Make Payment)

Should be **SecretMD5 For1002**

R3 for order ID **1002**: http://merchant.com/checkoutProcess.php?orderID=**1001**&LKMAC=**SecretMD5For1001**

Whitebox Approach

Evaluation and Results

Evaluation

oscommerce

- 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

Payment Module	Safe	Payment Module	Safe
2Checkout	✗	PayPal Pro - Direct Payments	✔
Authorize.net CC AIM	✔	PayPal (Payflow) - Direct Payments	✔
Authorize.net CC SIM	✗	PayPal (Payflow) - Express Checkout	✔
ChronoPay	✗	PayPal Standard	✗
inpay	✔	PayPoint.net SECPay	✗
iPayment (Credit Card)	✗	PSiGate	✗
Luottokunta (v1.2)	✗	RBS WorldPay Hosted	✗
Luottokunta (v1.3)	✗	Sage Pay Direct	✔
Moneybookers	✓	Sage Pay Form	✗
NOCHEX	✗	Sage Pay Server	✔
PayPal Express	✔	Sofortüberweisung Direkt	✔*

Taint Annotations of 12 Vulnerable Payment Modules

Payment Module	Order Id	Order Total	Merchant Id	Currency	Signed Tokens
2Checkout	✗	✗	✗	✗	
Authorize.net SIM	✗			✗	
ChronoPay	✗	✗	✗	✗	✗
iPayment (Credit card)	✗				
Luottokunta (v1.2)	✗	✗	✗	✗	
Luottokunta (v1.3)	✗			✗	
NOCHEX	✗	✗	✗	✗	
PayPal Standard			✗		
PayPoint.net SECPay	✗	✗		✗	
PSiGate	✗	✗	✗	✗	
RBS WorldPay Hosted				✗	✗
Sage Pay Form		✗		✗	
Total	**9**	**7**	**6**	**10**	**2**

Performance Results of 12 Vulnerable Payment Modules

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

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?

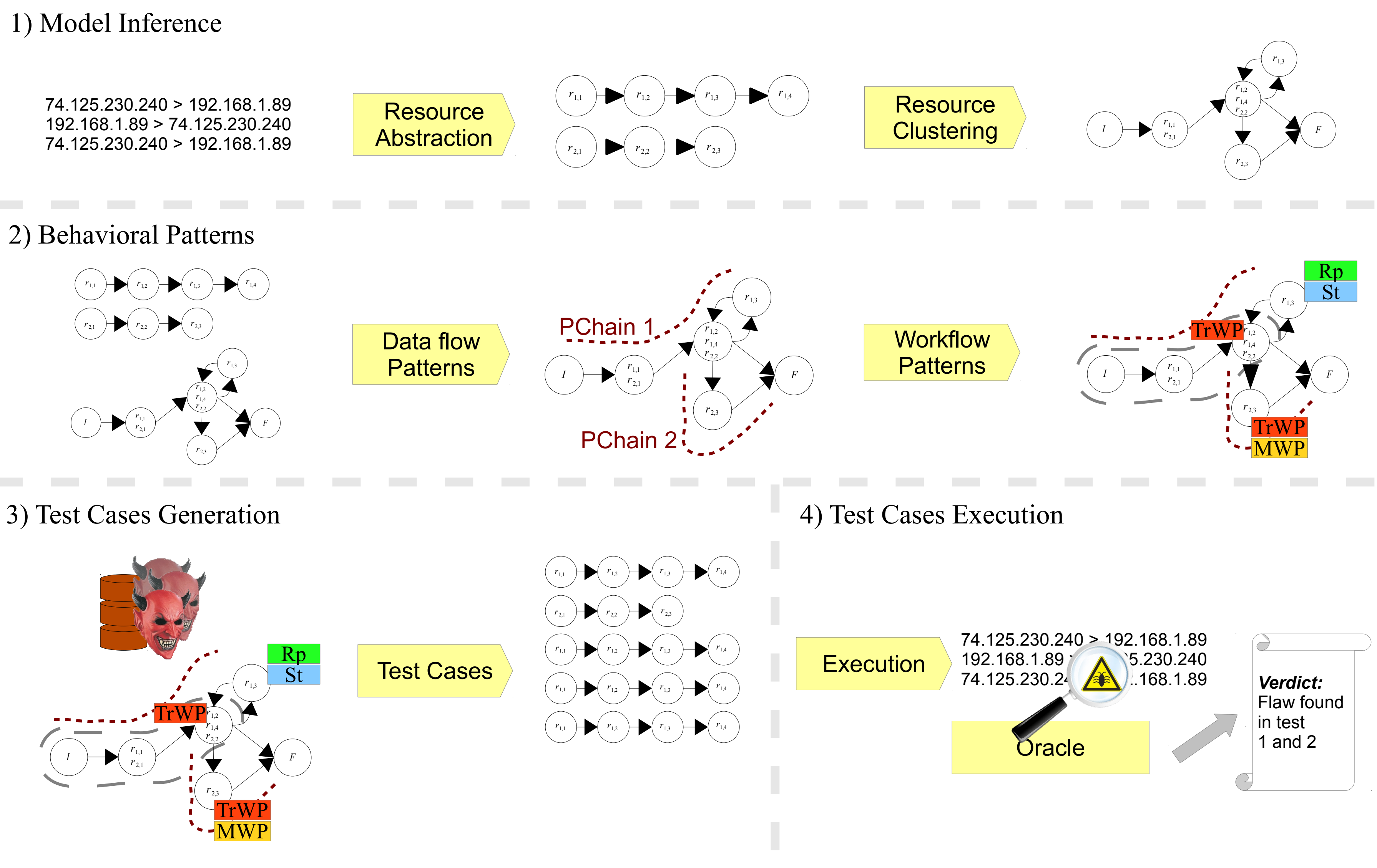
Blackbox Approach

Basic Problem

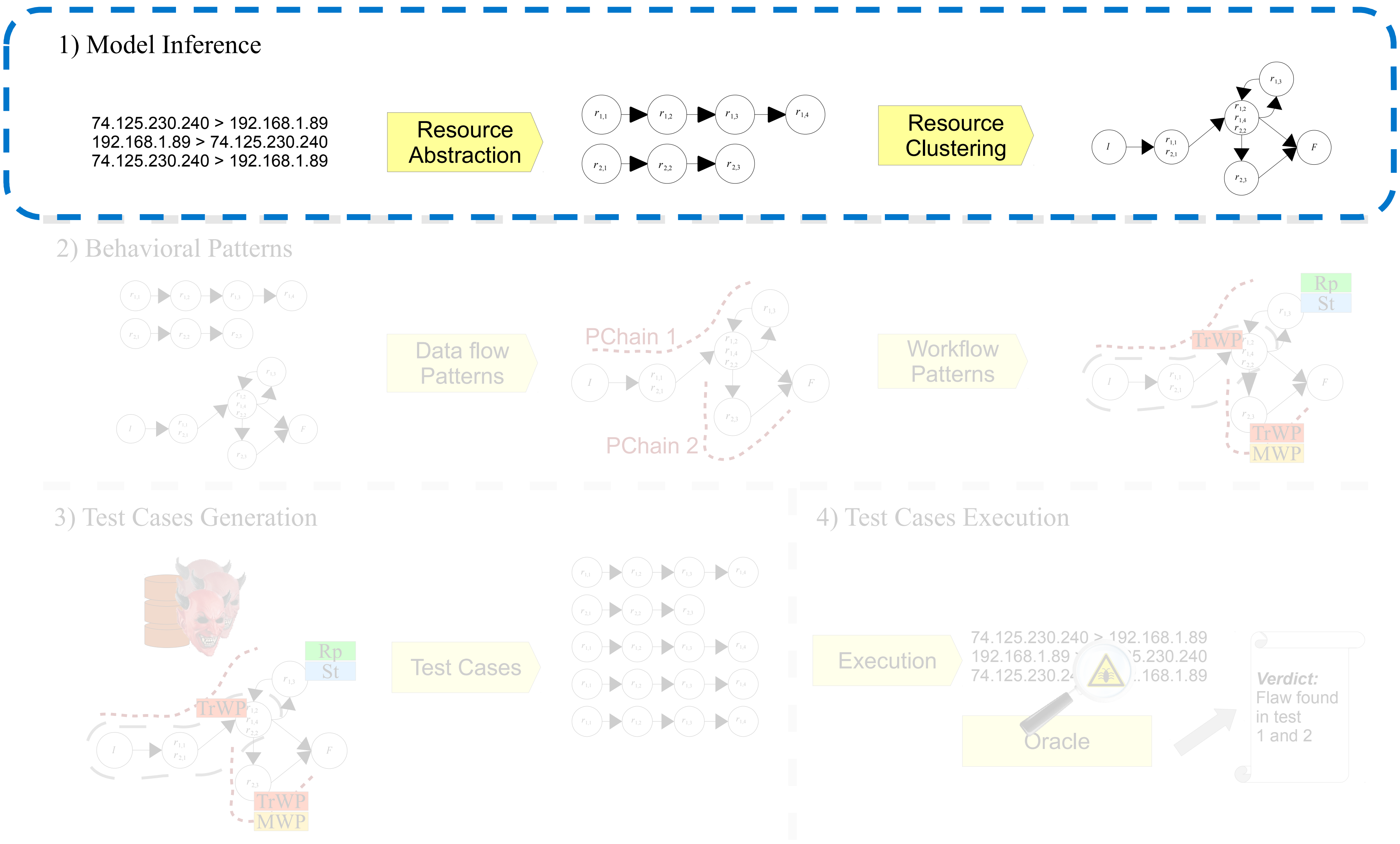
Blackbox Approach

Main Ideas

Overview

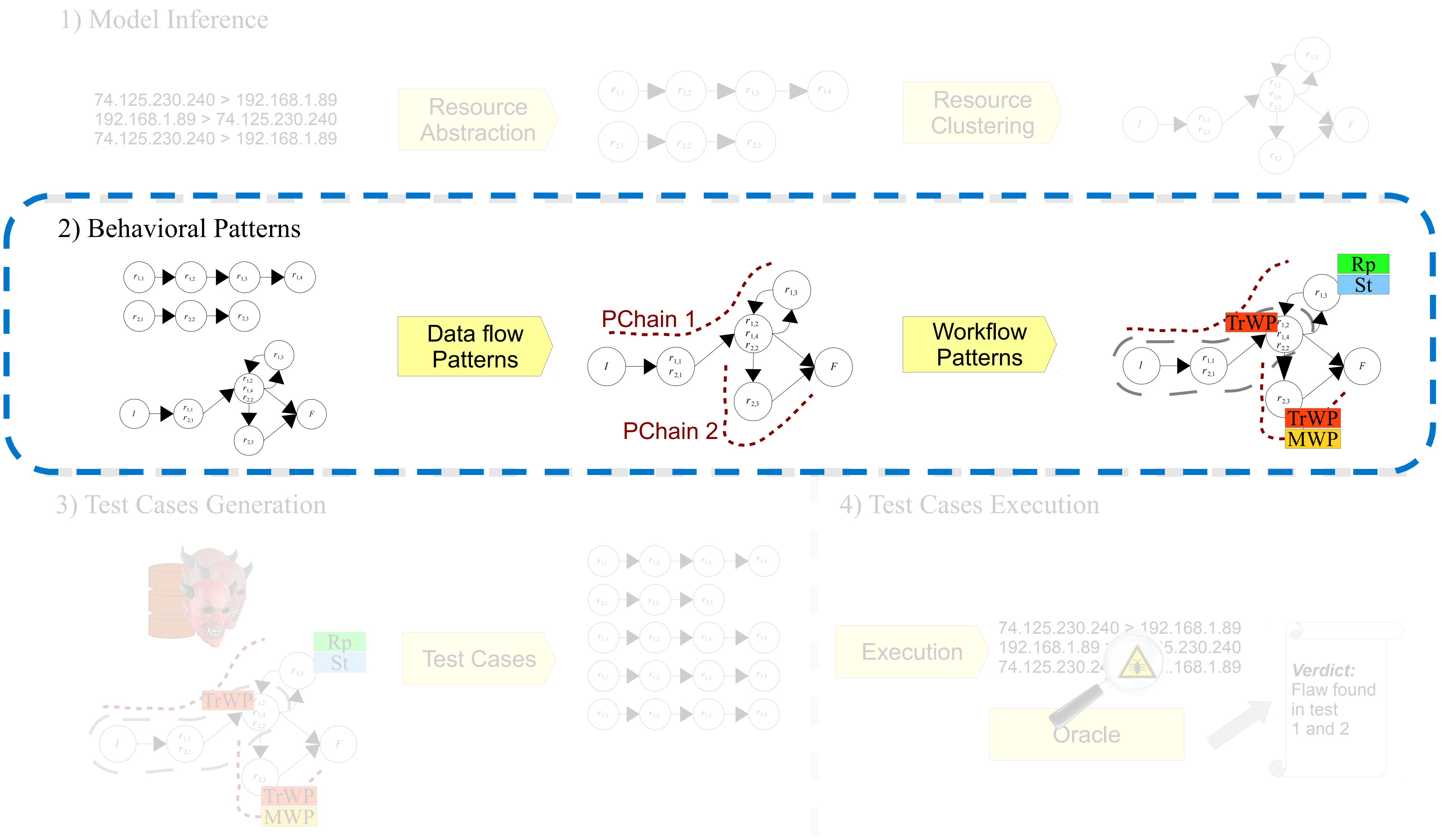

1) Model Inference
74.125.230.240 > 192.168.1.89
192.168.1.89 > 74.125.230.240
74.125.230.240 > 192.168.1.89
Resource Abstraction
Resource Clustering
2) Behavioral Patterns
Data flow Patterns
PChain 1
PChain 2
Workflow Patterns
Rp
St
TrWP
MWP
3) Test Cases Generation
Test Cases
4) Test Cases Execution
Execution
Oracle
Verdict: Flaw found in test 1 and 2

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
```

Behavioral Patterns Extraction



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:

$$\begin{tikzcd}
a \arrow[dr, bend left] \arrow[dd] & \\
 & b \arrow[ul, bend left] \\
c \arrow[d] & \\
d \arrow[d] & \\
e \arrow[r, bend left] & f \arrow[l, bend left]
\end{tikzcd}$$

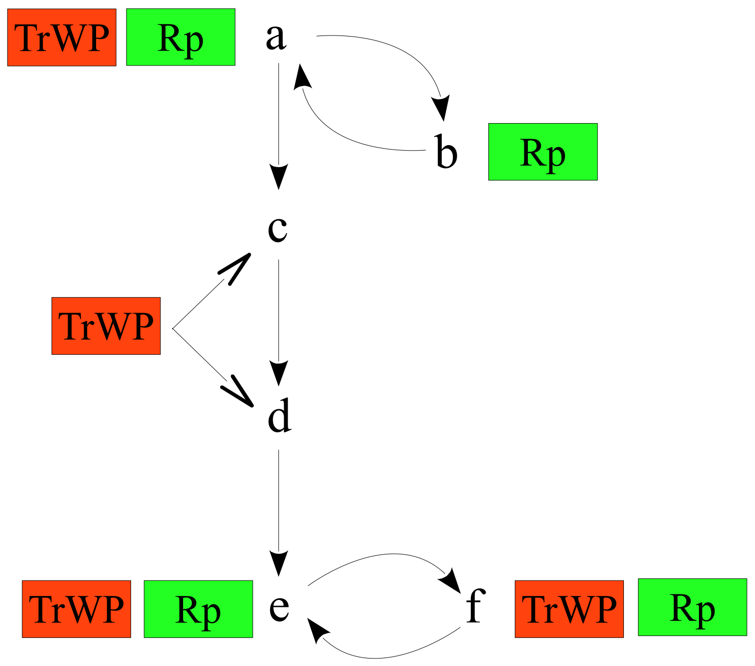
Workflow Patterns

Traces:

$$\pi_1 = \langle a, b, a, c, d, e, f, e \rangle$$

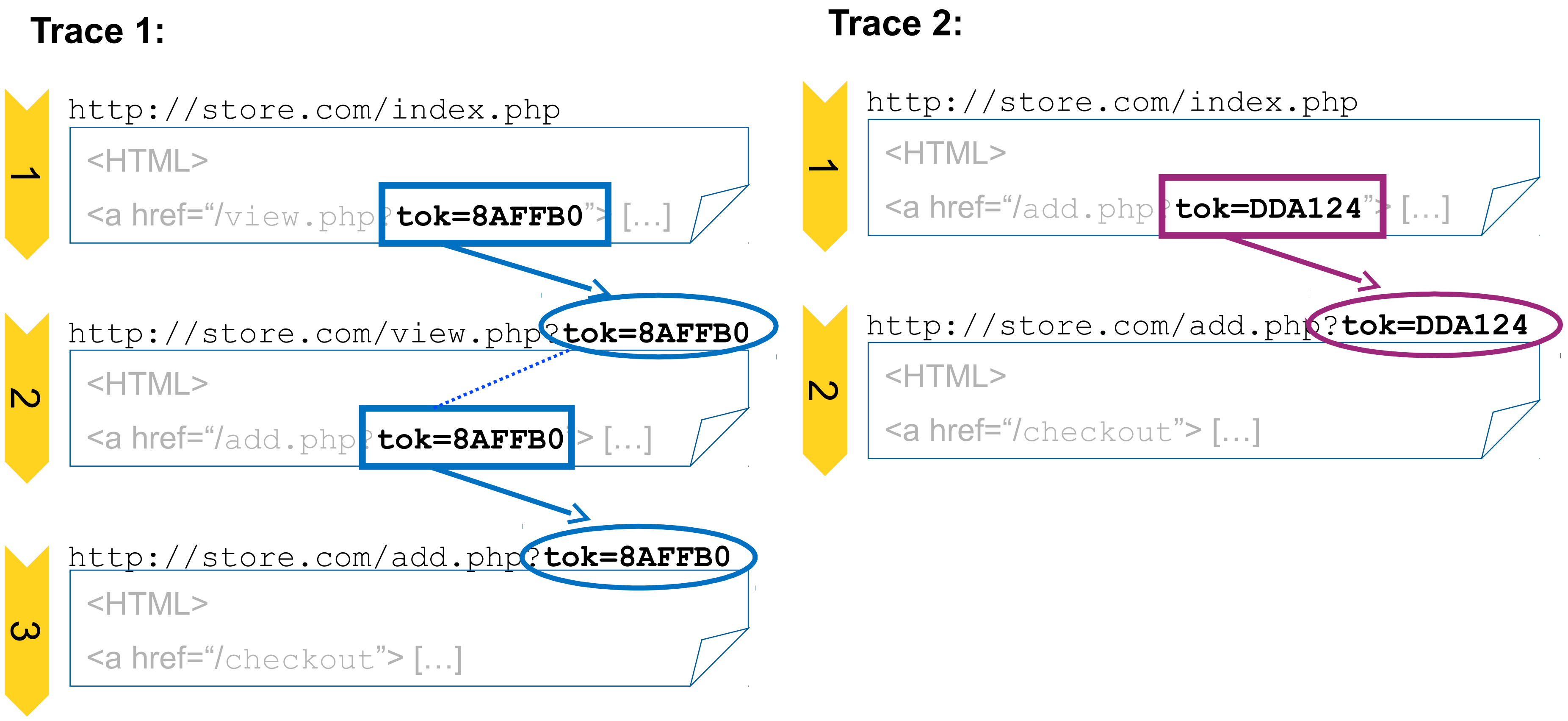
$$\pi_2 = \langle a, c, d, e, f, e \rangle$$

Model:

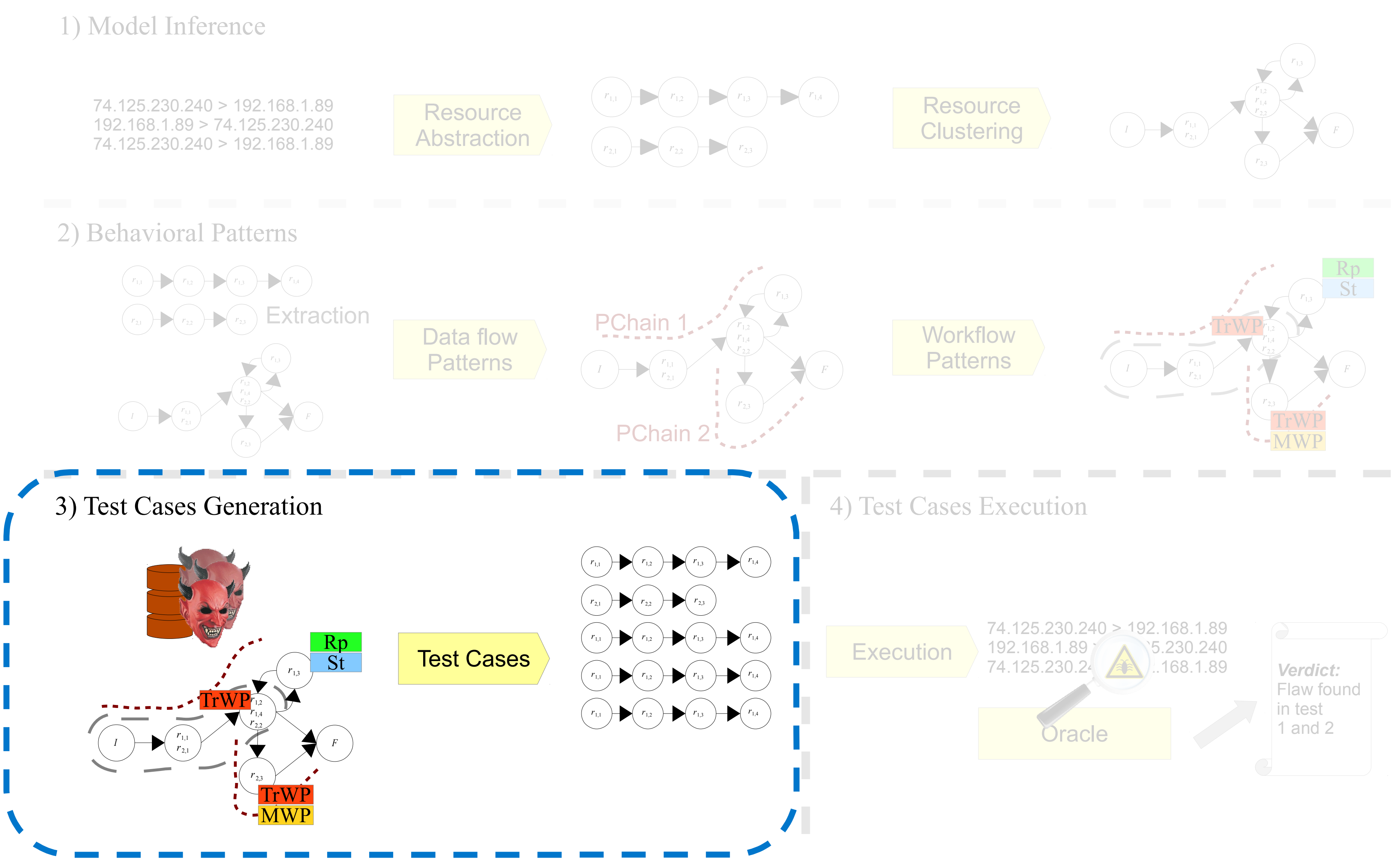


TrWP	: Trace Waypoints
Rp	: Repeatable Operations

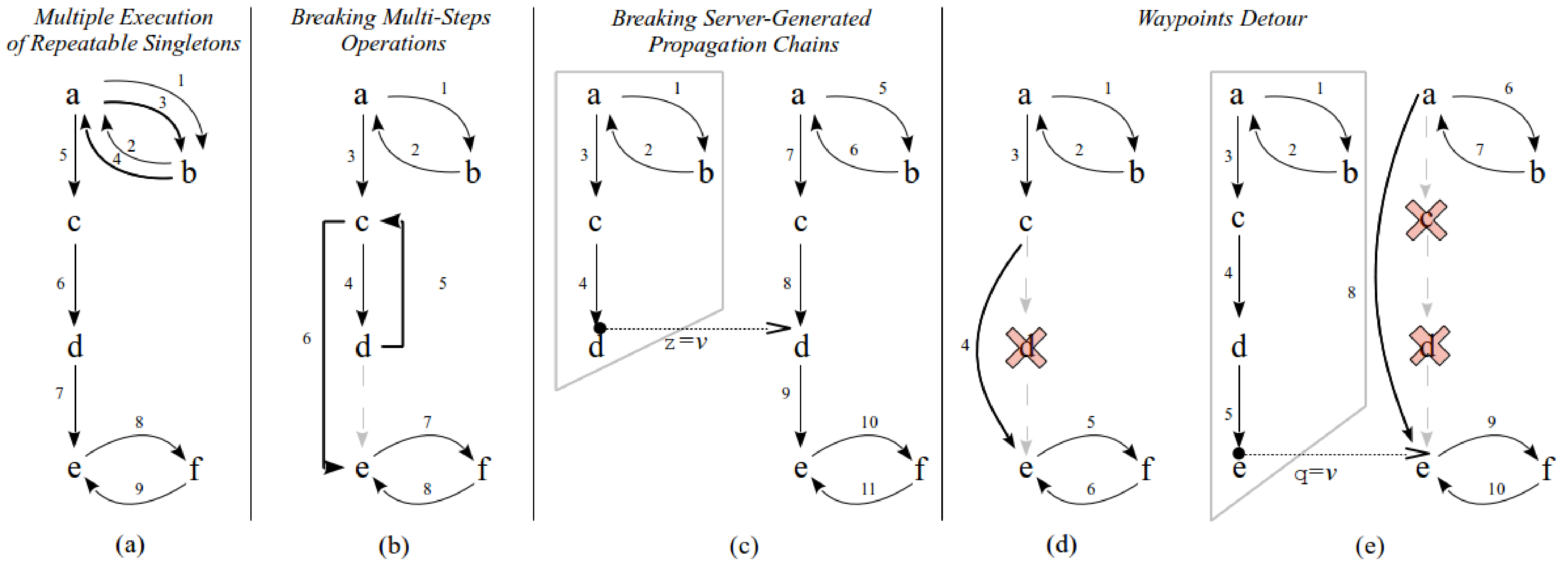
Data flow Patterns


Trace 1:
1
http://store.com/index.php
<HTML>
<a href="/view.php?tok=8AFFB0"> [...]
2
http://store.com/view.php?tok=8AFFB0
<HTML>
<a href="/add.php?tok=8AFFB0"> [...]
3
http://store.com/add.php?tok=8AFFB0
<HTML>
<a href="/checkout"> [...]
Trace 2:
1
http://store.com/index.php
<HTML>
<a href="/add.php?tok=DDA124"> [...]
2
http://store.com/add.php?tok=DDA124
<HTML>
<a href="/checkout"> [...]

Test Case Generation

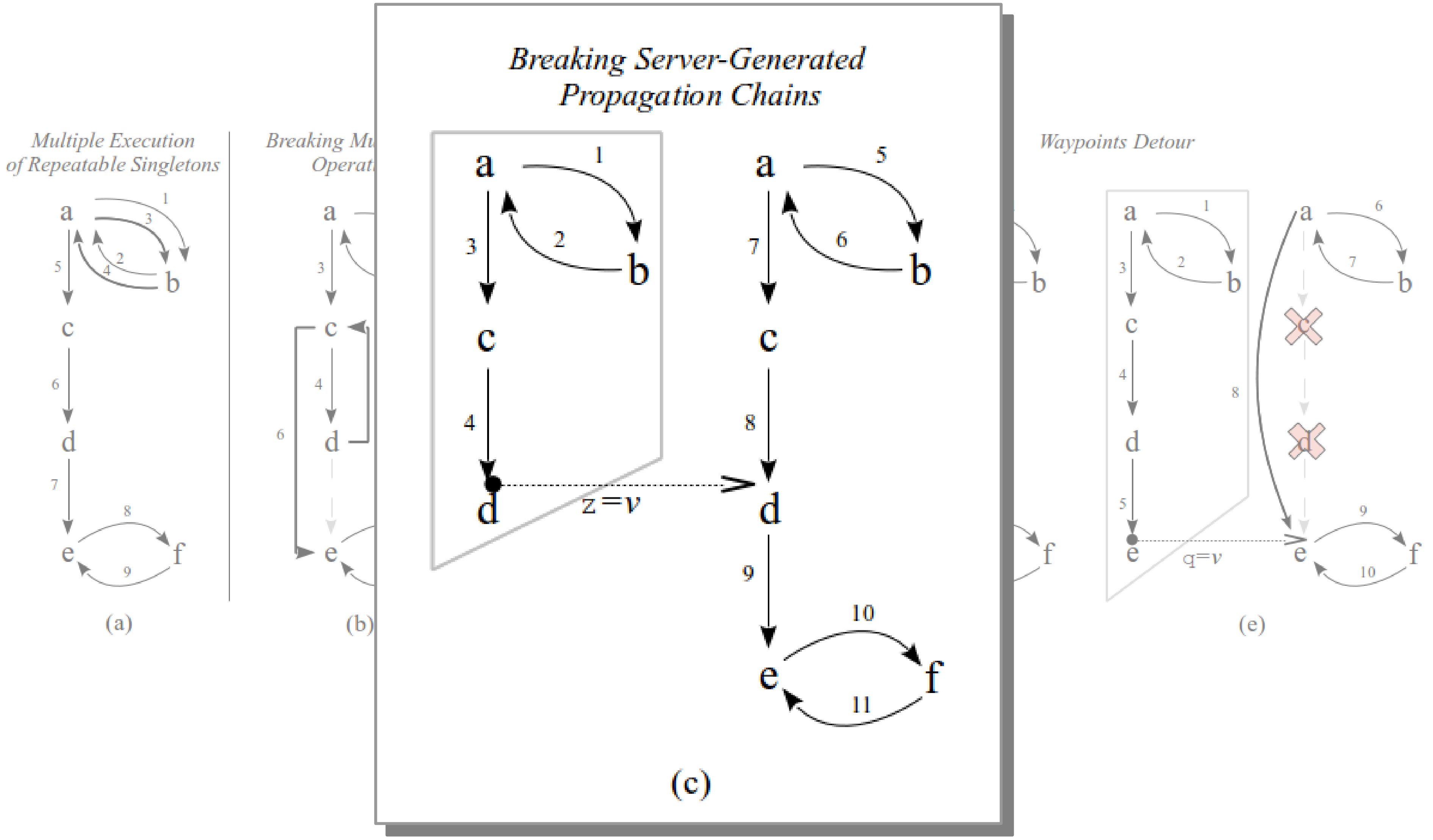


Attack Pattern-based Test Case Generation

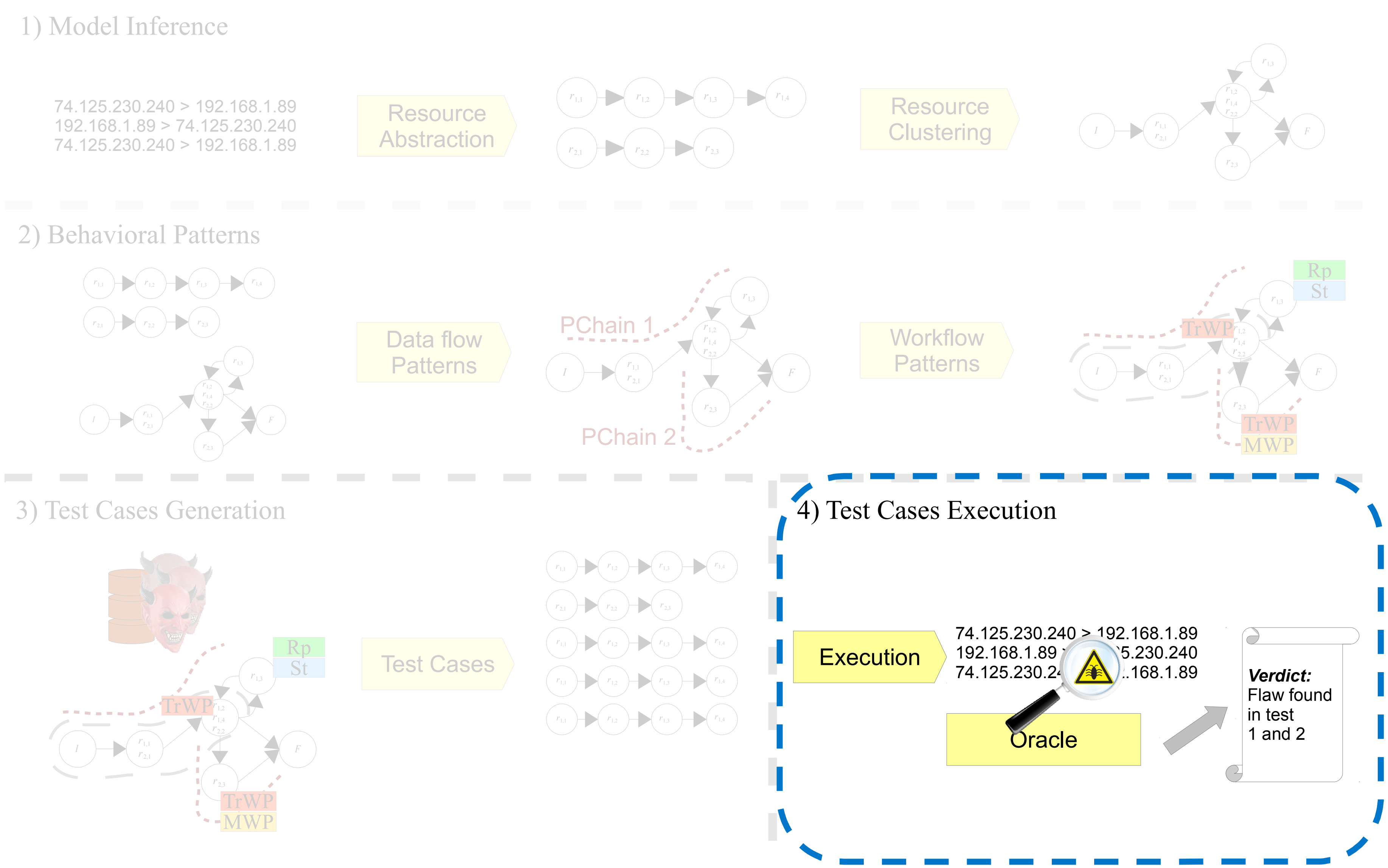


Multiple Execution of Repeatable Singletons
Breaking Multi-Steps Operations
Breaking Server-Generated Propagation Chains
Waypoints Detour
z=v
q=v
(a)
(b)
(c)
(d)
(e)

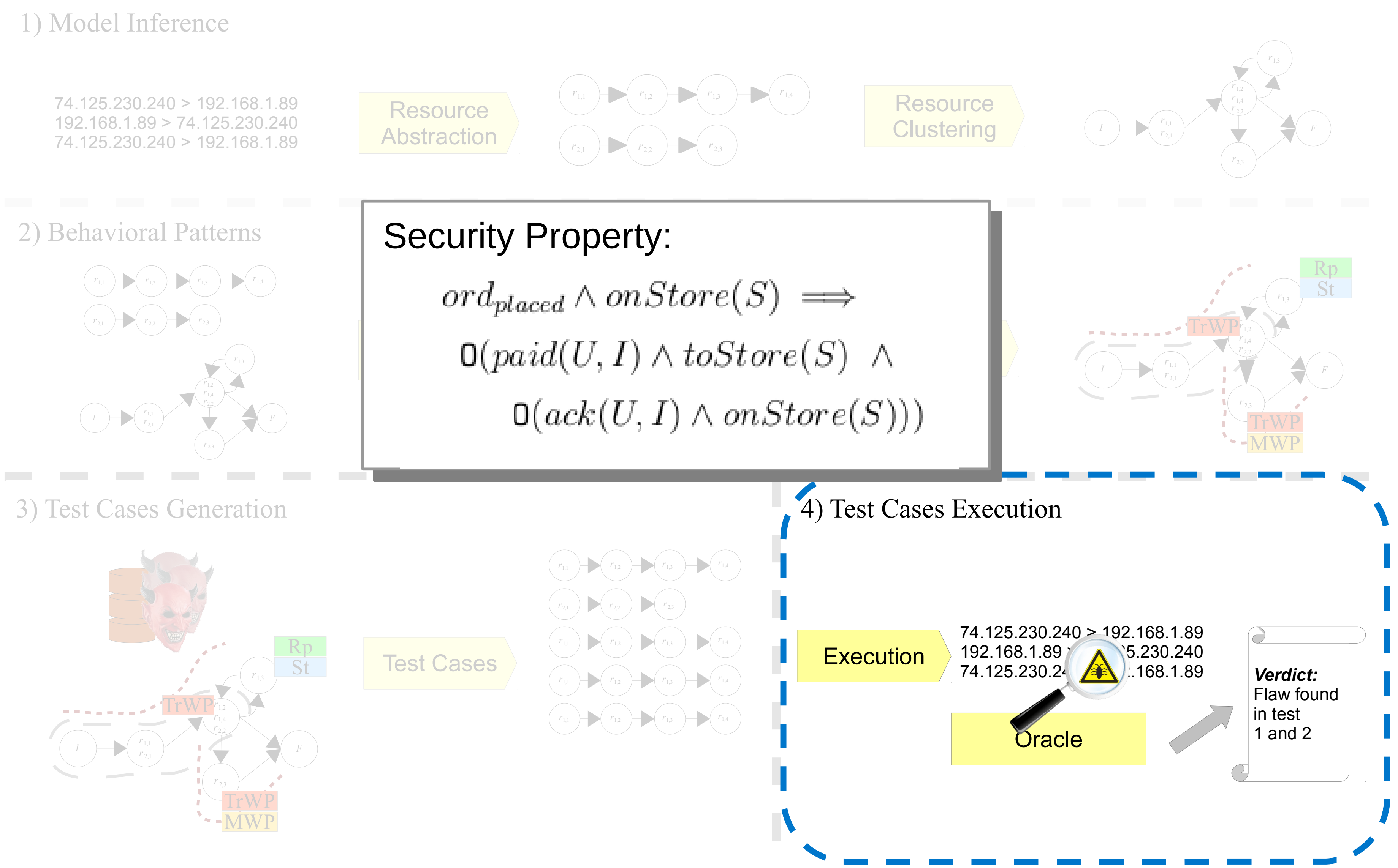
Attack Pattern-based Test Case Generation


Multiple Execution of Repeatable Singletons
Breaking Mu
Operati
Breaking Server-Generated Propagation Chains
Waypoints Detour
z=v
q=v
(a)
(b)
(c)
(e)

Test Case Execution and Oracle



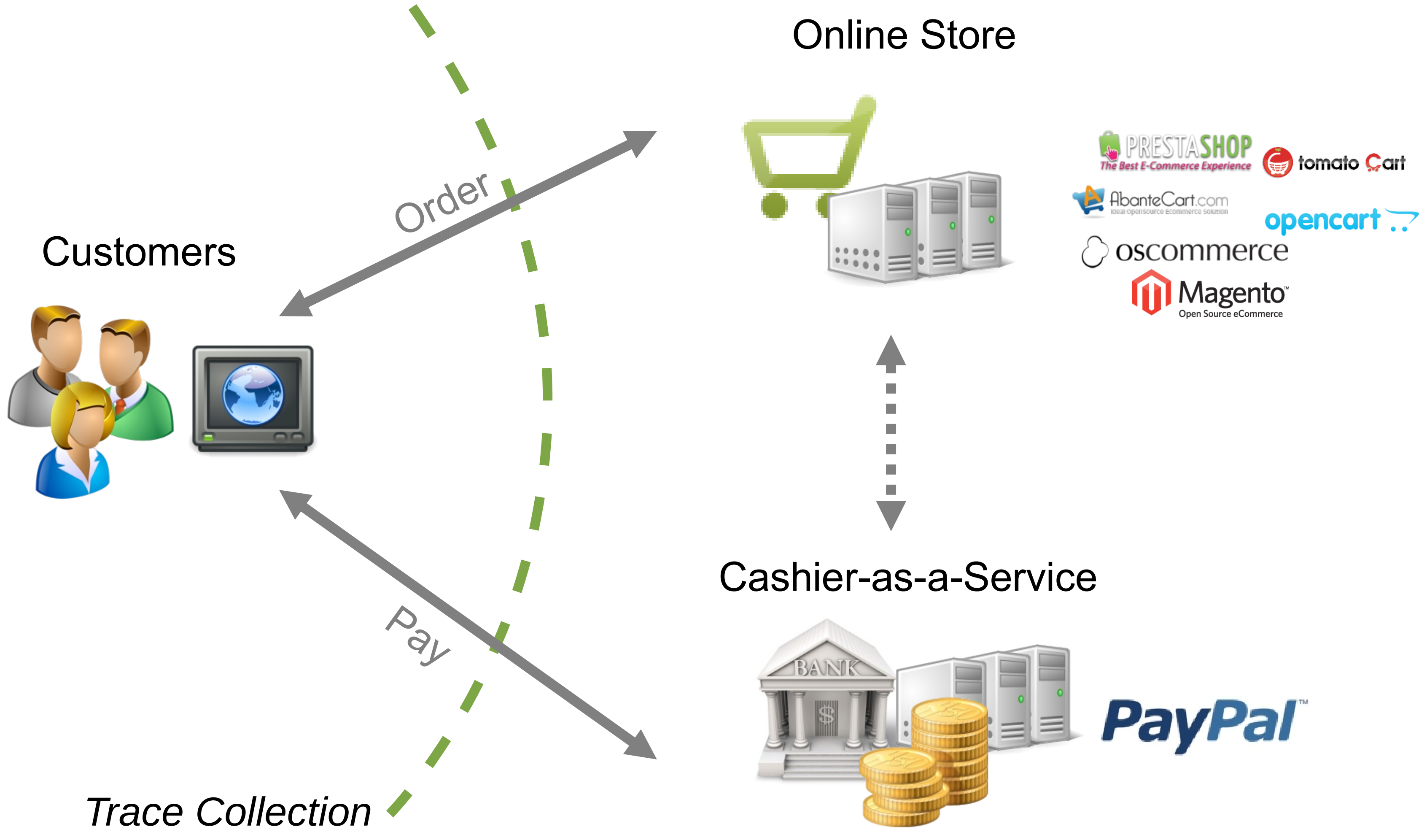
Test Case Execution and Oracle



Security Property:

$$ord_{placed} \land onStore(S) \implies$$
$$\mathrm{O}(paid(U,I) \land toStore(S) \;\; \land$$
$$\mathrm{O}(ack(U,I) \land onStore(S)))$$

Case Study: Shopping Cart Web Applications


Online Store
PRESTASHOP The Best E-Commerce Experience
tomato Cart
AbanteCart.com Ideal OpenSource Ecommerce Solution
opencart
oscommerce
Magento Open Source eCommerce
Customers
Order
Pay
Cashier-as-a-Service
BANK
PayPal
Trace Collection

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

- In total **3,145 test cases**

1,253 “misuse” detected

1,892 were executed

EURECOM

Experiments and Results

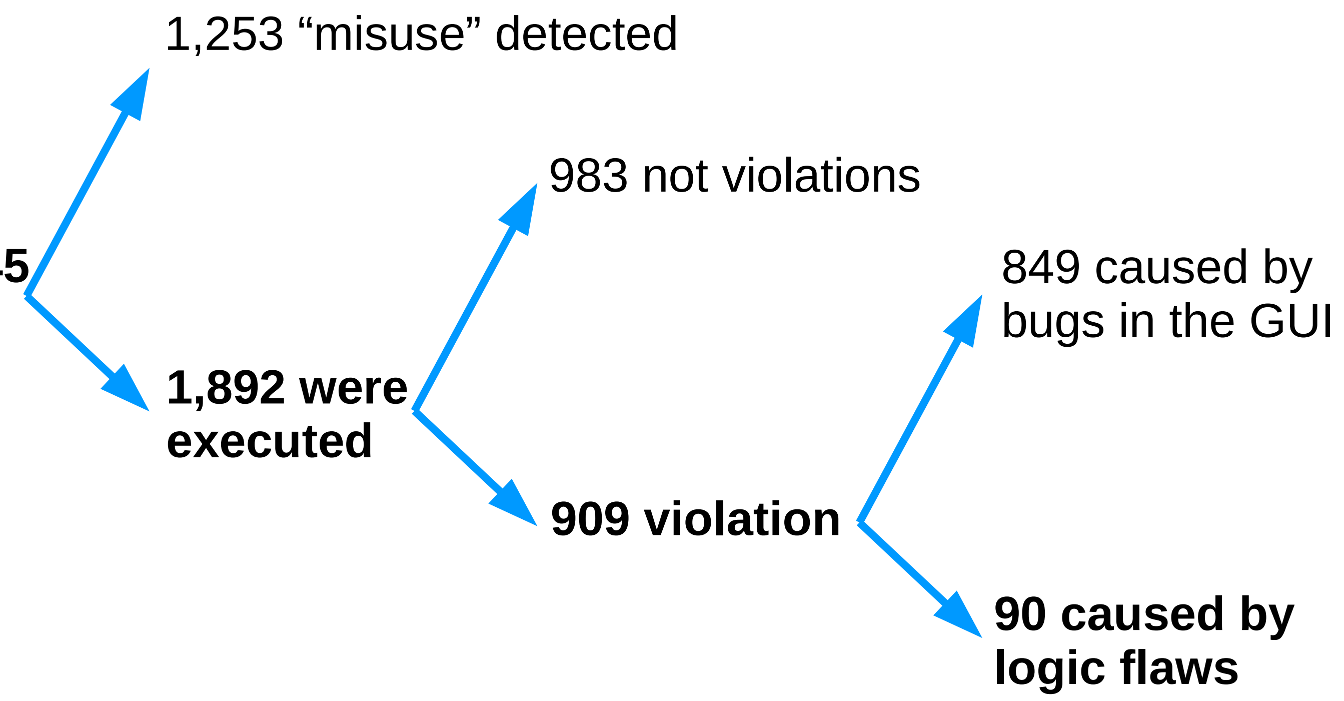
- Target: 7 popular eCommerce Web Applications
  - Deployed by >13M online stores

- Testbed: created 12 Paypal sandbox configurations

$$\begin{tikzcd}
 & \text{1,253 “misuse” detected} & \\
\text{In total }\textbf{3,145 test cases} \arrow[ur] \arrow[dr] & & \text{983 not violations} \\
 & \textbf{1,892 were executed} \arrow[ur] \arrow[dr] & \\
 & & \textbf{909 violation}
\end{tikzcd}$$

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**



Vulnerabilities

- 10 previously-unknown vulnerabilities
  - Allowing to shop for free or pay less

Application	Shop for free	Pay less	Session Fixation	
AbanteCart	x			Notified Devel.
Magento				
OpenCart		x x		Notified Devel.
osCommerce	x	x		CVE-2012-2991
PrestaShop				
TomatoCart	x	x x	x	CVE-2012-4934
CS-Cart	x			CVE-2013-0118

EURECOM

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?