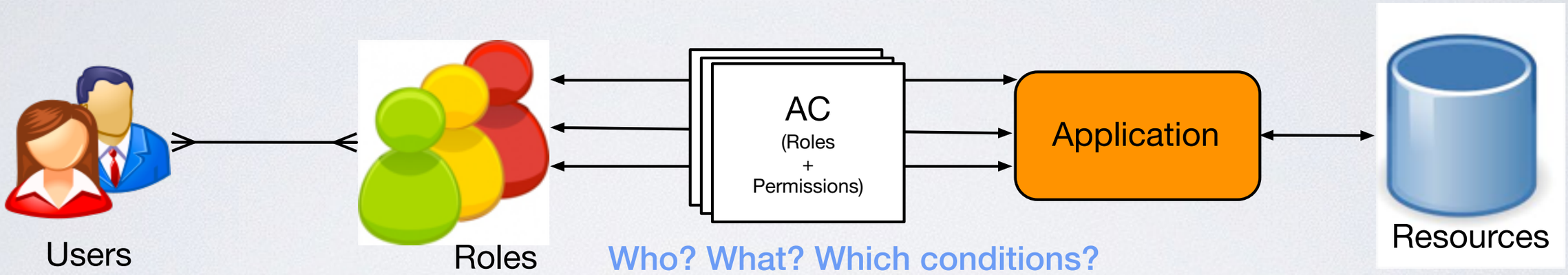
Reverse-engineering and Automated Testing of Access Control Policies

Thanh H. Le, Cu D. Nguyen, and Lionel Briand
Interdisciplinary Centre for Security, Reliability and Trust (SnT Centre)
Benjamin Hourte
HITEC Luxembourg

TAROT 2015

Access Control in Web Applications



AC policy specifications

AC policy implementations

AC Vulnerabilities and Exploitation Facts

OWASP Top 10 (2013)

A1 – Injection

A2 – Broken Authentication and Session Management

A3 – Cross-Site Scripting (XSS)

A4 – Insecure Direct Object References

A5 – Security Misconfiguration

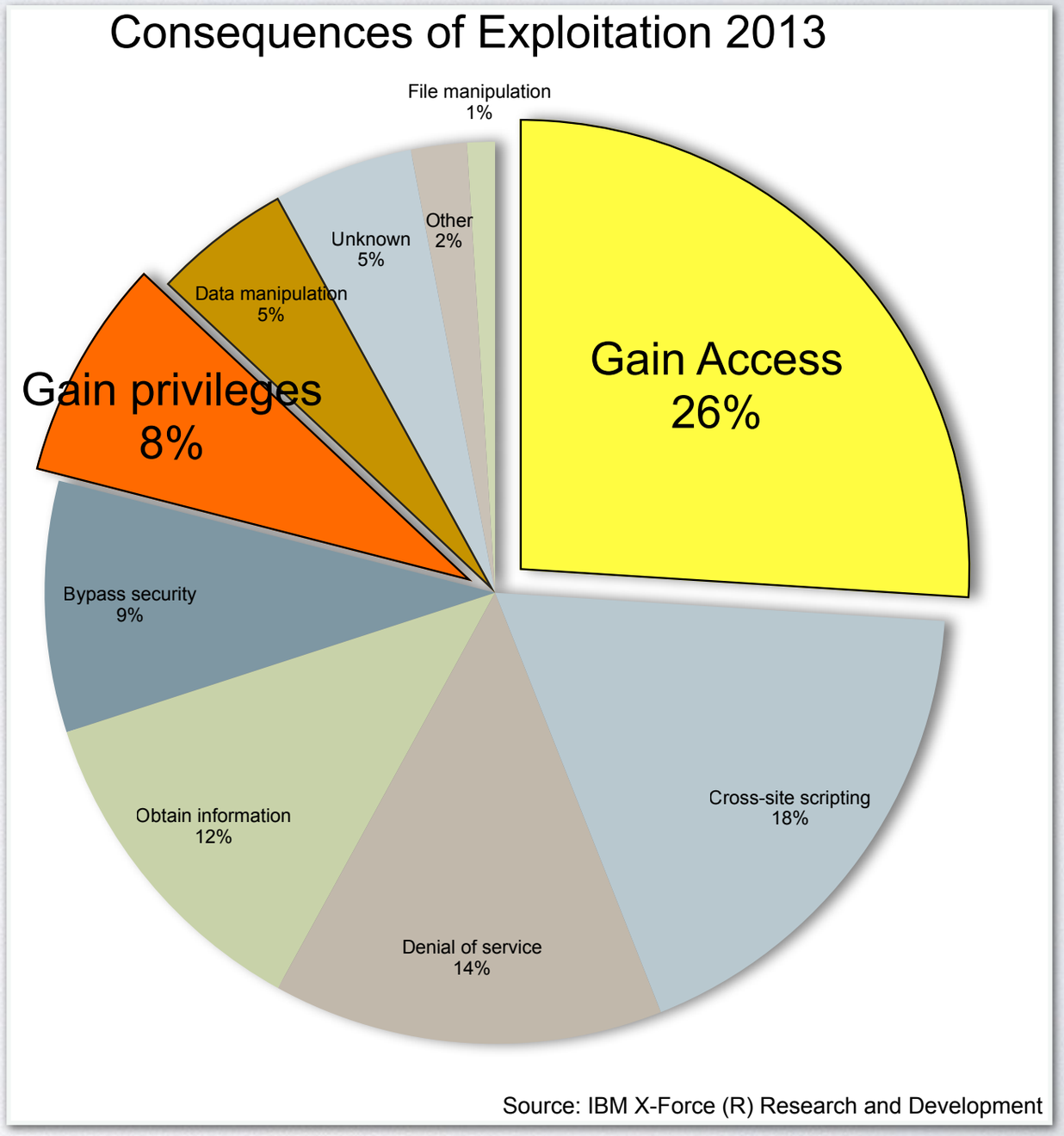
A6 – Sensitive Data Exposure

A7 – Missing Function Level Access Control

A8 – Cross-Site Request Forgery (CSRF)

A9 – Using Known Vulnerable Components

A10 – Unvalidated Redirects and Forwards



Research Problems and Goal

Problems:

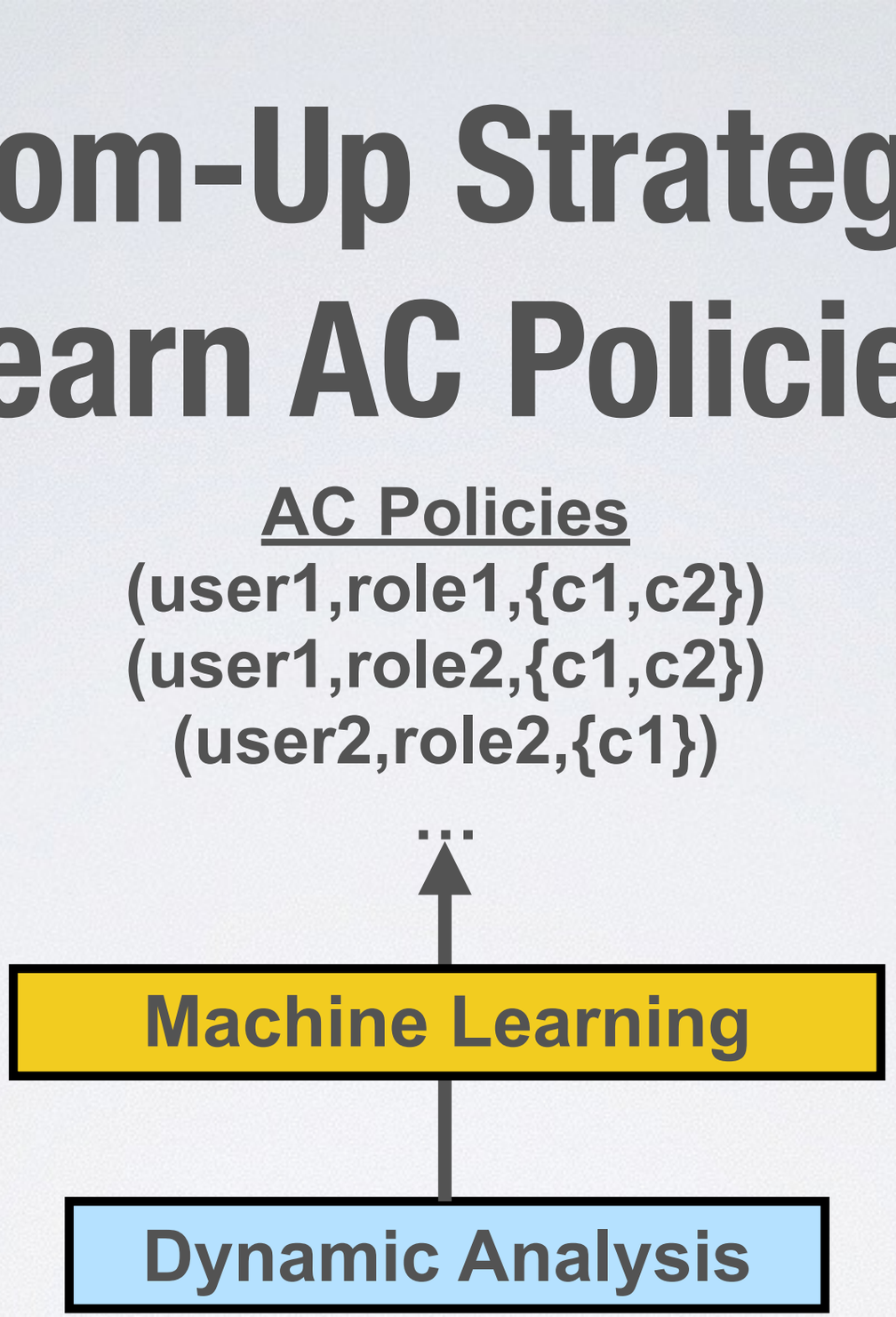
- **AC policies are not explicitly defined or modelled**
- **Lacks of a systematic, scalable, and automated approach and tools for testing AC**

Research Goal:

- **Bottom-up reverse engineering of AC policies using dynamic analysis and machine learning**

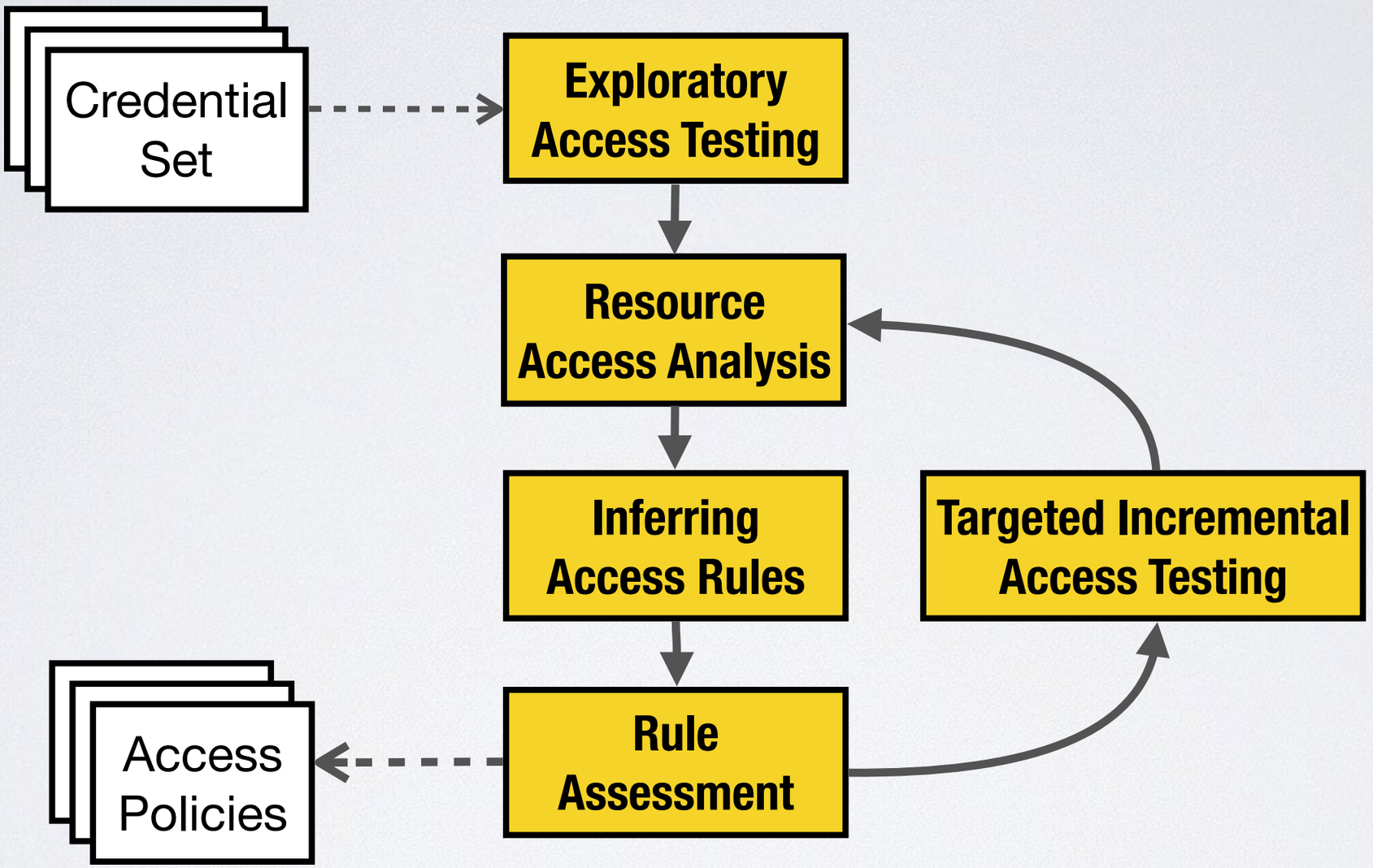
Bottom-Up Strategy To Learn AC Policies

AC Policies
(user1,role1,{c1,c2})
(user1,role2,{c1,c2})
(user2,role2,{c1})
…

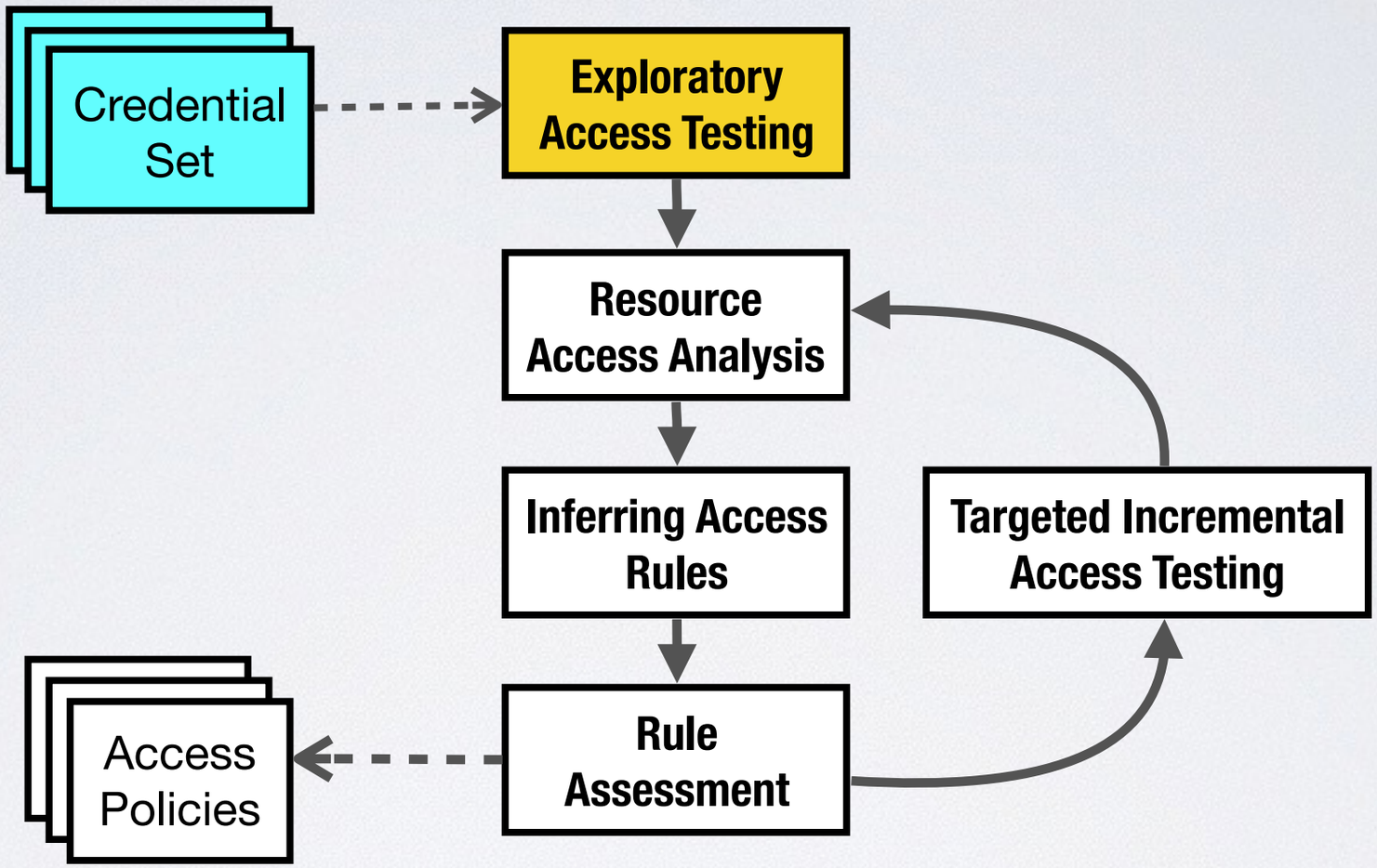


- **Automatically discover resources and access permissions**
- **Use machine-learning to learn AC policy specifications**

The Steps of Our Approach

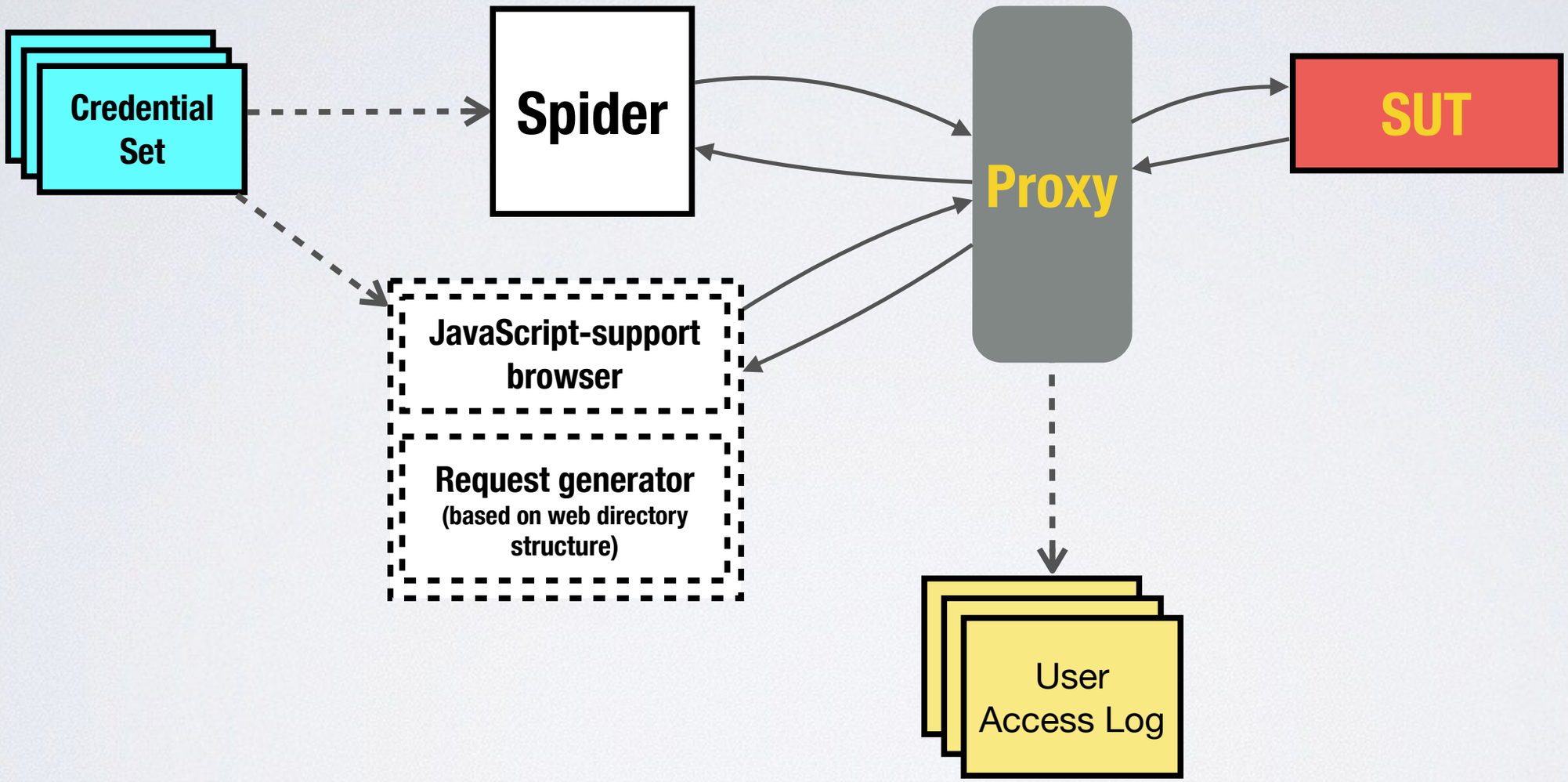


Step 1. Exploratory Access Testing



- Discovering SUT resources using semi-automated crawling
- Access testing: all users vs all resources

Step 1. Exploratory Access Testing

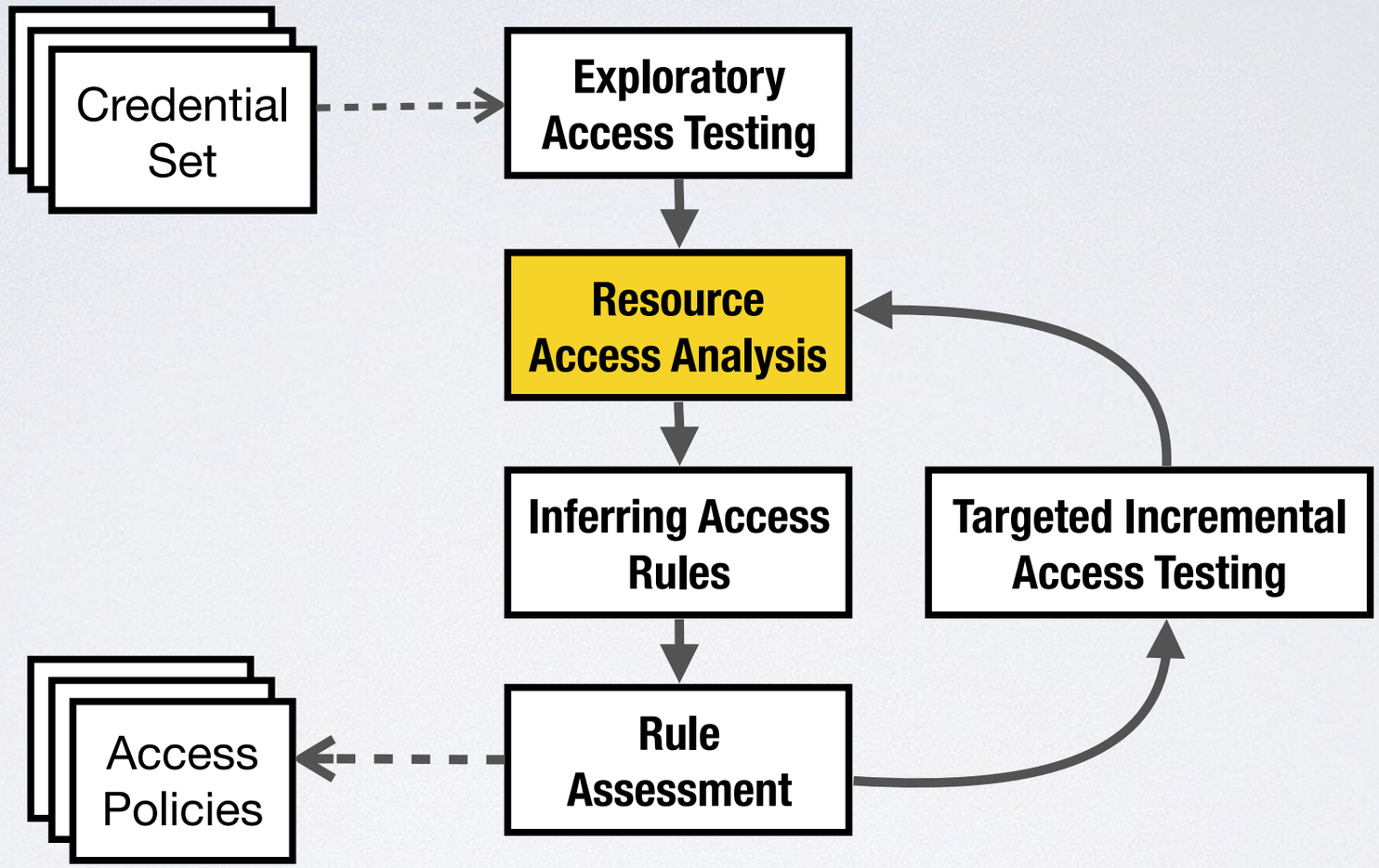


- **Provide more crawling entries to improve resources discovery**

Step 1 Output: User Access Logs

- HTTP requests (resource URLs, parameters, etc.)
- HTTP responses for requests from every specific user

Step 2. Resource Access Analysis



- Extract resources from access requests
- Determine access permissions based on response contents and HTTP codes

Step 2 (continued)

- Extract resources from HTTP requests
  - **base-URI-resource** : base URI (http://host/path/to/resouce)
  - **full-resource**: base URI + parameters (e.g.: query strings, session's cookies) (http://host/path/to/resouce?p1=v1&p2=v2)
- Determine access permissions from responses
  - based on HTTP codes and response content **patterns**

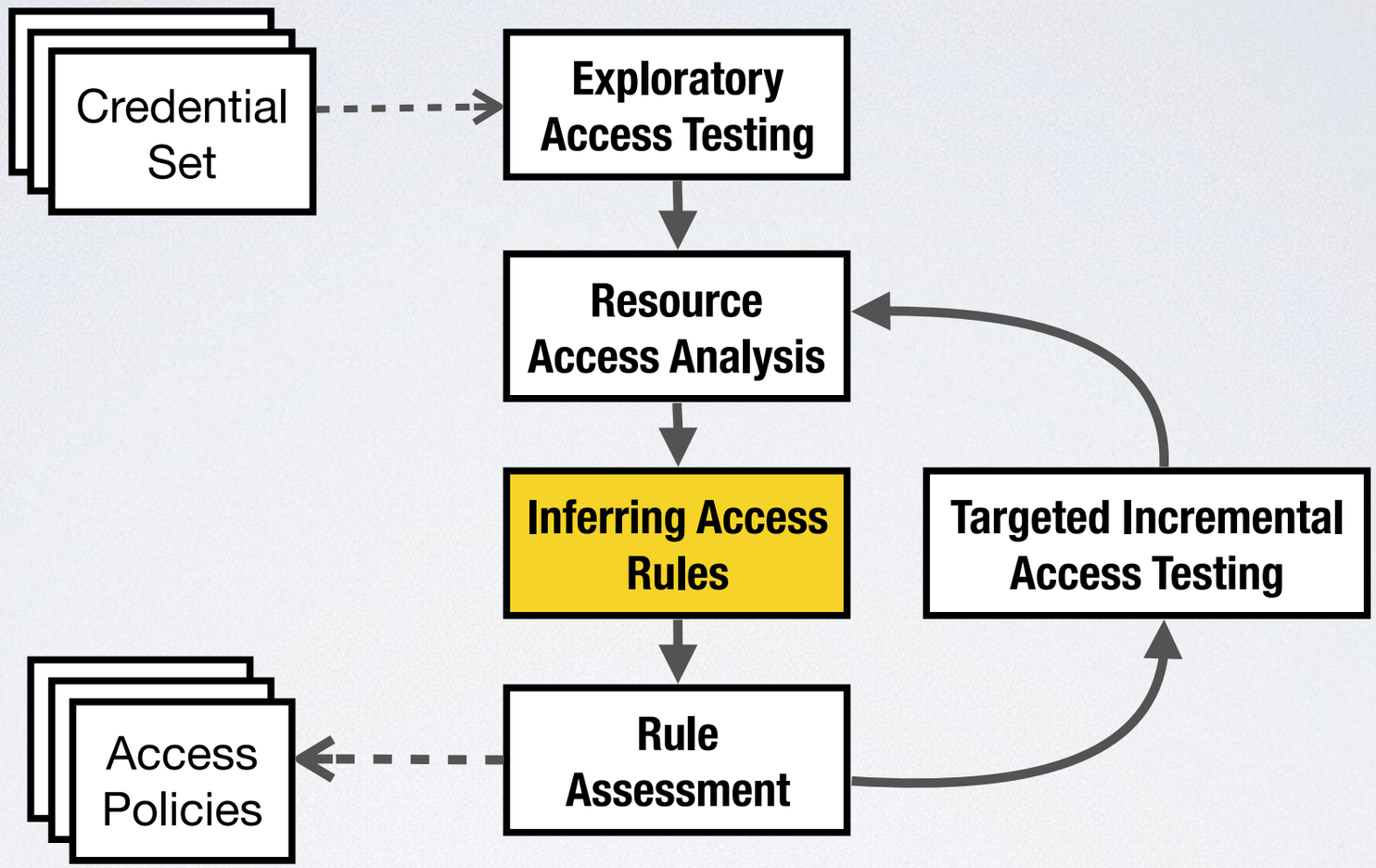
Condition	Permission
HTTP Code = 4xx or 5xx or 301, any content	denied
HTTP Code = 200 or 302 or 304, denial-matched content	denied
HTTP Code = 200 or 302 or 304, not-matched content	allowed

- Output: Access Data (user, role, resource, {res-attributes}, {context-attributes}, permission)

Example of the Access Data Obtained at Step 2

User	Role	Resource	Method	Request Attribute	Permission
admin	**adminRole**	/home/admin/createUser	GET		ALLOWED
admin	**adminRole**	/home/admin/createUser	POST	username=‘john007’&fullname	ALLOWED
admin	**adminRole**	/home/admin/createUser	POST	username=‘pte.lm’&fullname=	ALLOWED
admin	**adminRole**	/home/admin/createUser	POST	username=‘bob’&fullname=‘B	ALLOWED
junior	**juniorManagerRole**	/home/admin/createUser	GET		DENIED
senior	**seniorManagerRole**	/home/admin/createUser	GET		DENIED
senior	**seniorManagerRole**	/home/admin/createUser	GET		DENIED
junior	**juniorManagerRole**	/home/manager/manageDocument/ajax-create	GET		ALLOWED
junior	**juniorManagerRole**	/home/manager/manageDocument/ajax-create	POST	title=‘a title’&docType=‘note’&	ALLOWED
senior	**seniorManagerRole**	/home/manager/manageDocument/ajax-create	POST	title=‘new title’&docType=‘note	DENIED
junior	**juniorManagerRole**	/home/manager/manageDocument/ajax-delete	GET	docId=1	ALLOWED
senior	**seniorManagerRole**	/home/manager/manageDocument/ajax-delete	GET	docId=1	DENIED
junior	**juniorManagerRole**	/home/manager/manageDocument/ajax-update	POST	docId=1	ALLOWED
senior	**seniorManagerRole**	/home/manager/manageDocument/ajax-update	POST	docId=1	DENIED
user1	**userRole**	/home/manager/manageDocument/ajax-update	GET	docId=1	DENIED
user2	**userRole**	/home/user/viewDocument	GET	docId=1	DENIED
junior	**juniorManagerRole**	/home/user/viewDocument	GET	docId=1	DENIED
user1	**userRole**	/home/user/viewDocument	GET	docId=2	DENIED
user1	**userRole**	/home/user/viewDocument	GET	docId=1	ALLOWED

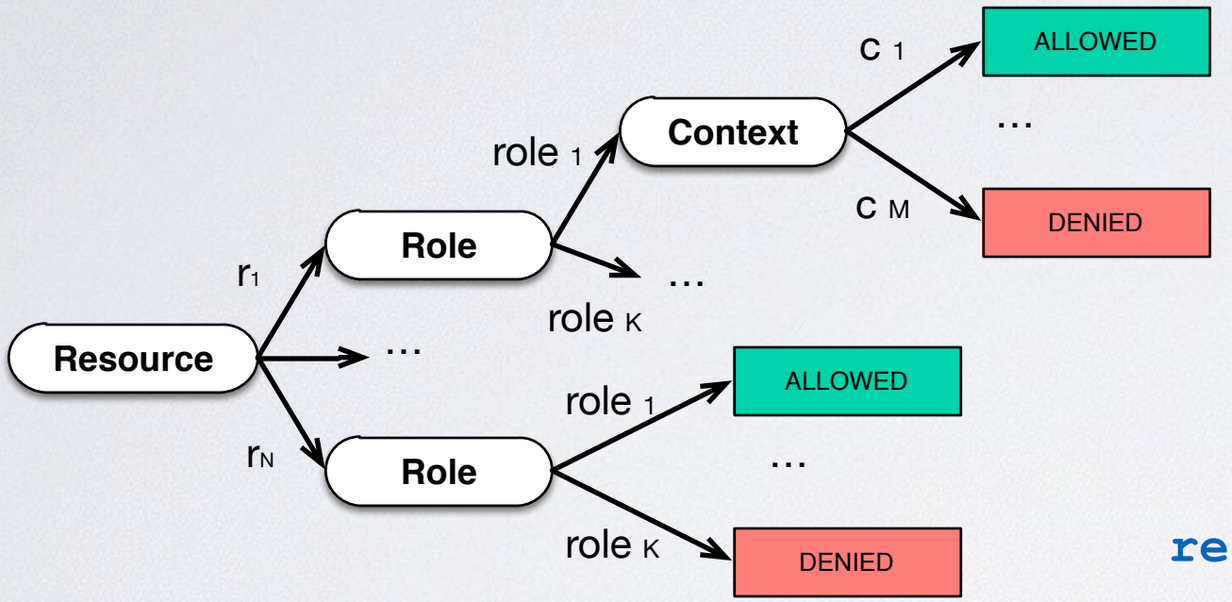
Step 3. Inferring Access Rules



Infer AC policies from access data using machine learning

Step 3 Output:

- **Resource access classification using Decision tree**



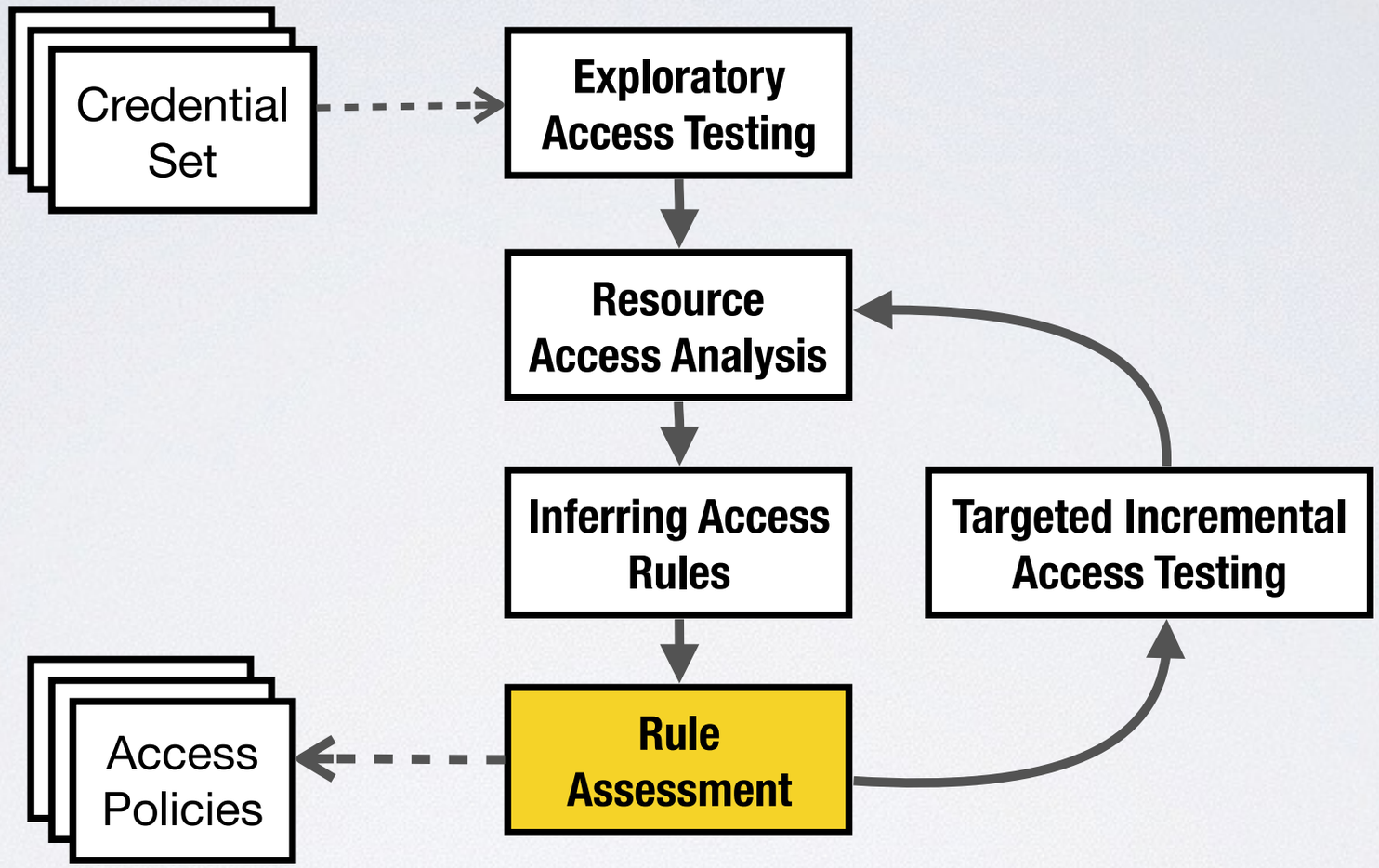
Example:

```
resource = "home/user/viewDocument"
    |   role = userRole :
        | method = "GET":  DENIED (3/1)
    |   role = juniorManagerRole:
        | method = "GET": DENIED (1/0)
```

- **Access rule:**

```
IF resource = "home/user/viewDocument"
    AND role = userRole
    AND method = "GET"
THEN DENIED
```

Step 4. Rule Assessment



Highlight inconsistent and “suspicious” policies

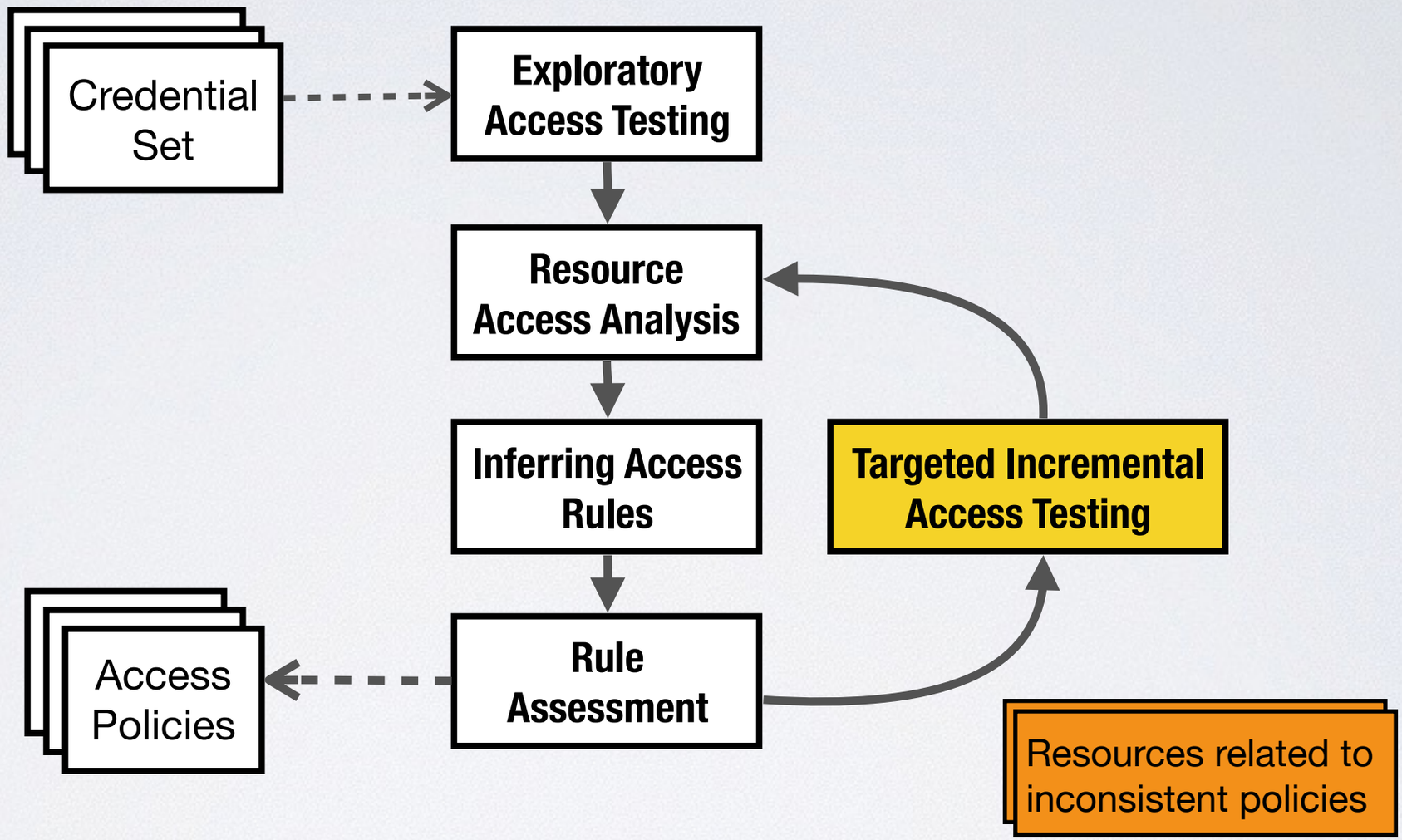
Step 4 (cont.)

- **Inconsistent policies (inferred with less than 100% confidence)**

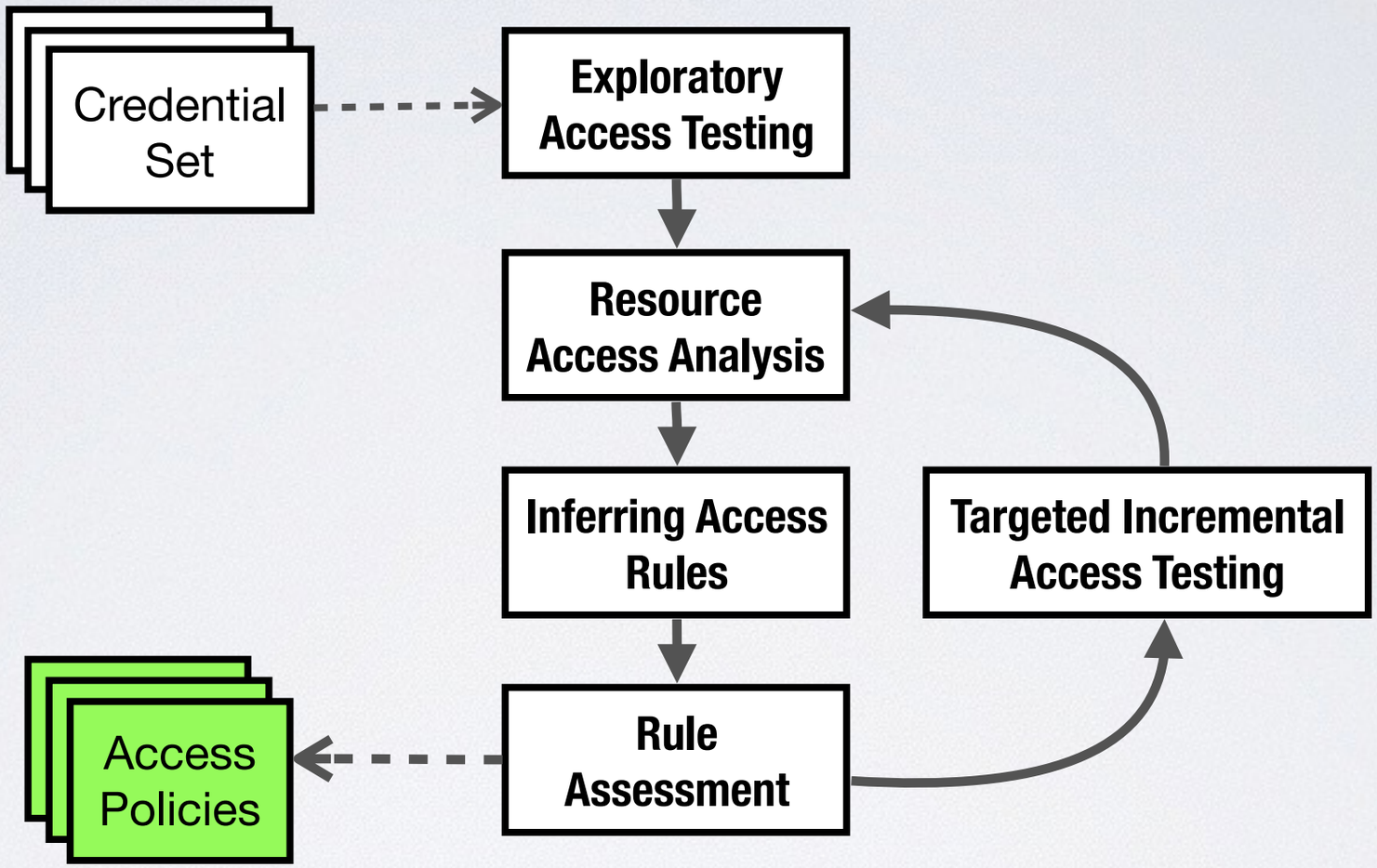
```
resource = “home/user/viewDocument”
    |  role = userRole :
        | method = “GET”:  DENIED (3/1)
```

  - **transient server errors**
    - **resource extraction**
    - **other factors like user’s profiles**
- **Suspicious policies**
  - **Related to sensitive resources (e.g. database, configuration, password)**
  - **Having permission as allowed to all users**

Step 5. Incremental Access Testing



Analyse and Infer AC Policies



- To be validated by experts

Evaluation

Research Questions

RQ1. Does the proposed approach effectively discover resources for inferring AC policies?

RQ2. What is the quality in terms of correctness and consistency of the inferred AC policies?

RQ3. How useful are the inferred AC rules in detecting AC issues?

Web Applications

	iTrust	ISP
Licence	Open Source	Commercial
AC specification	Hardcoded Role-based AC policies - *Gold standard*	Specified by ISP's super users in administration module

RQ1. Resource Discovery

	iTrust		ISP	
Discovery Method	Baseline	**All-entry**	without Javascript	**with Javascript**
# of Resources Found	130	**248 (100% of all resources)**	353	**680 (?)**

RQ2. Quality of Inferred AC Rules: iTrust Result

204 resources (out of 248):

- **~95% AC rules are correct with the gold standard**
- **~5% AC rules cannot be confirmed because they are not covered by the gold standard**
  - **38 resources are not protected by AC design and implementation**

RQ3. Detecting AC Issues

iTrust

- We detected three vulnerabilities in the unprotected resources
  - ../util/resetPassword.jsp allows a user to change passwords of other users
  - ../util/getUser.jsp is supposed to have access enforced from other pages but can be directly accessed without authorisation
  - ../errors/reboot.jsp allows any user to reboot the web server
- 44 resources (out of 248) return Java exception error pages which are accessible by all users
  - Disclose source codes and database information

ISP

- 15 CSV and JSON files that were not protected from direct accesses from all users
- Based on inconsistent rules, we found
  - Discrepancy between defined AC rules and inferred AC rules
  - AC enforcement is not correctly implemented
  - Confirmed by ISP developers

Challenges

Dealing with domain data, business flows & contexts

- Why: will help discovering more resources & attributes that affect AC policies
  - to learn more consistent policies
- How:
  - submit meaningful and diverse input data
  - consider business logic: data flows and request orders
  - consider access contexts & user profiles

Submit meaningful and diverse input data with combinatorial testing

- Specify input values classification using XInput
  - XML syntax, inspired by XML Schema (XSD), familiar to developers and easy to use
- Define data types & domains of input fields
  - Data types: integer, double, string, date, hex
  - Using value restrictions to define data domains:
    - minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits, fractionDigits, length, minLength, maxLength, enumeration
    - regular expression (regex)

Specifying XInput

- Extract from user interface
  - E.g.: <select>
- Extract from crawling logs automatically
- Ask domain experts

Example of XInput

```
<xinput id="mainBodySubView:subview:mainform:displayName:pssu-management-form-displayName" inputFieldId="POST_/pssuportal/
management/vehicles/vehicle/create/new/?mainBodySubView:subview:mainform:displayName:pssu-management-form-displayName"
source="interactive" type="POSTQSTR">

    <atomicParam id="mainBodySubView:subview:mainform:displayName:pssu-management-form-displayName">

        <dataClz base="string" name="vehicle display name">
            <minLength value=“5”/>
            <maxLength value=“40”/>
        </dataClz>
        <dataClz base=“string” name=“name with invalid character”>
            <enumeration value=“delete * where 1=1;”/>
            <enumeration value=“   “/>
        </dataClz>
    </atomicParam>
</xinput>
```

Summary

- We proposed a bottom-up reverse engineering of AC policies for Web applications
  - Effectively discover Web application resources and determine resource accesses
  - Inferred AC policies are highly correct with AC specification
- The inconsistency of inferred AC policies can help finding AC issues