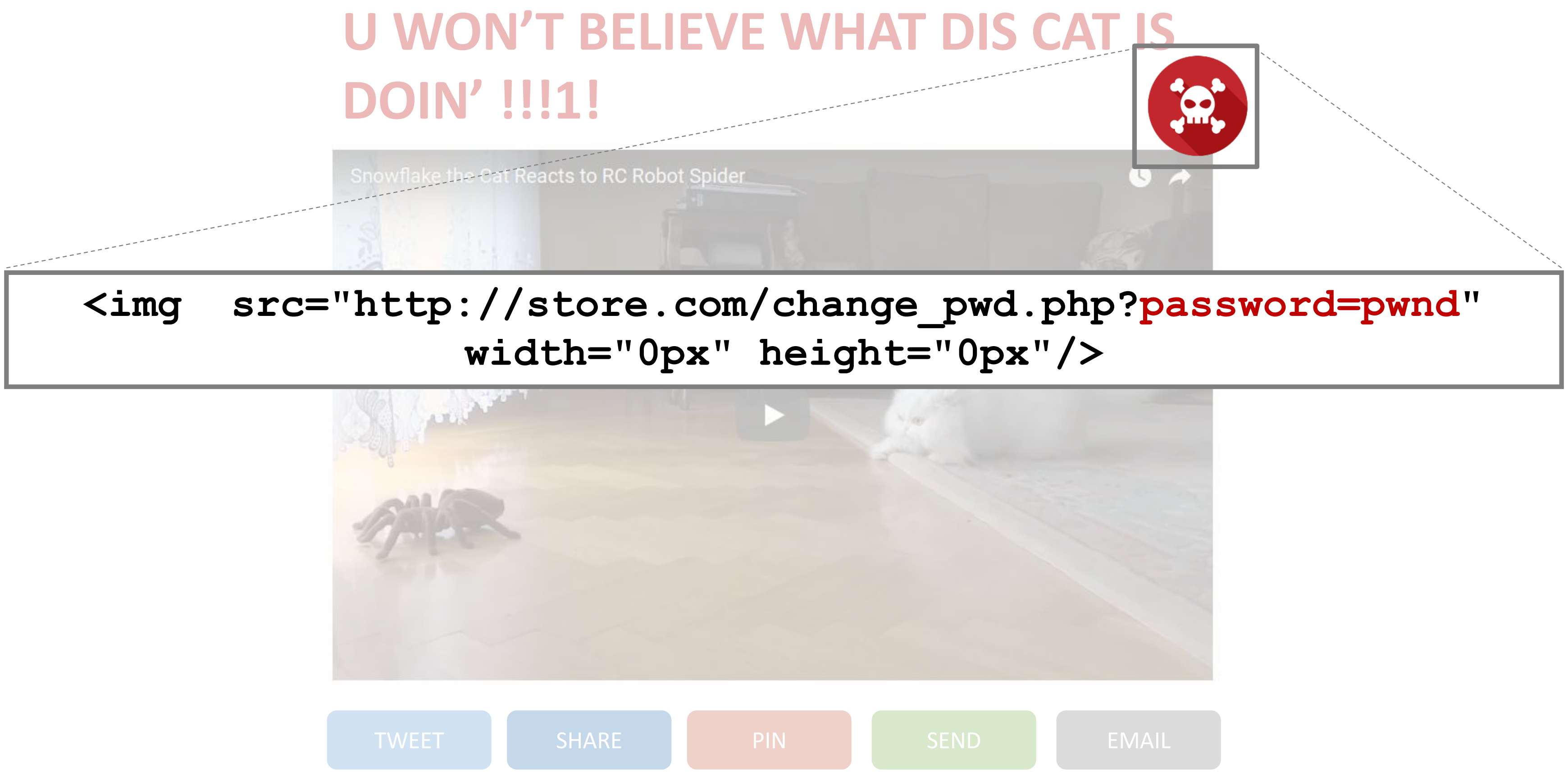
CISPA Center for IT-Security, Privacy and Accountability

Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs

G. Pellegrino et al.
giancarlo.pellegrino@cispa.saarland

(presented by Martin Johns, SAP Security Research)

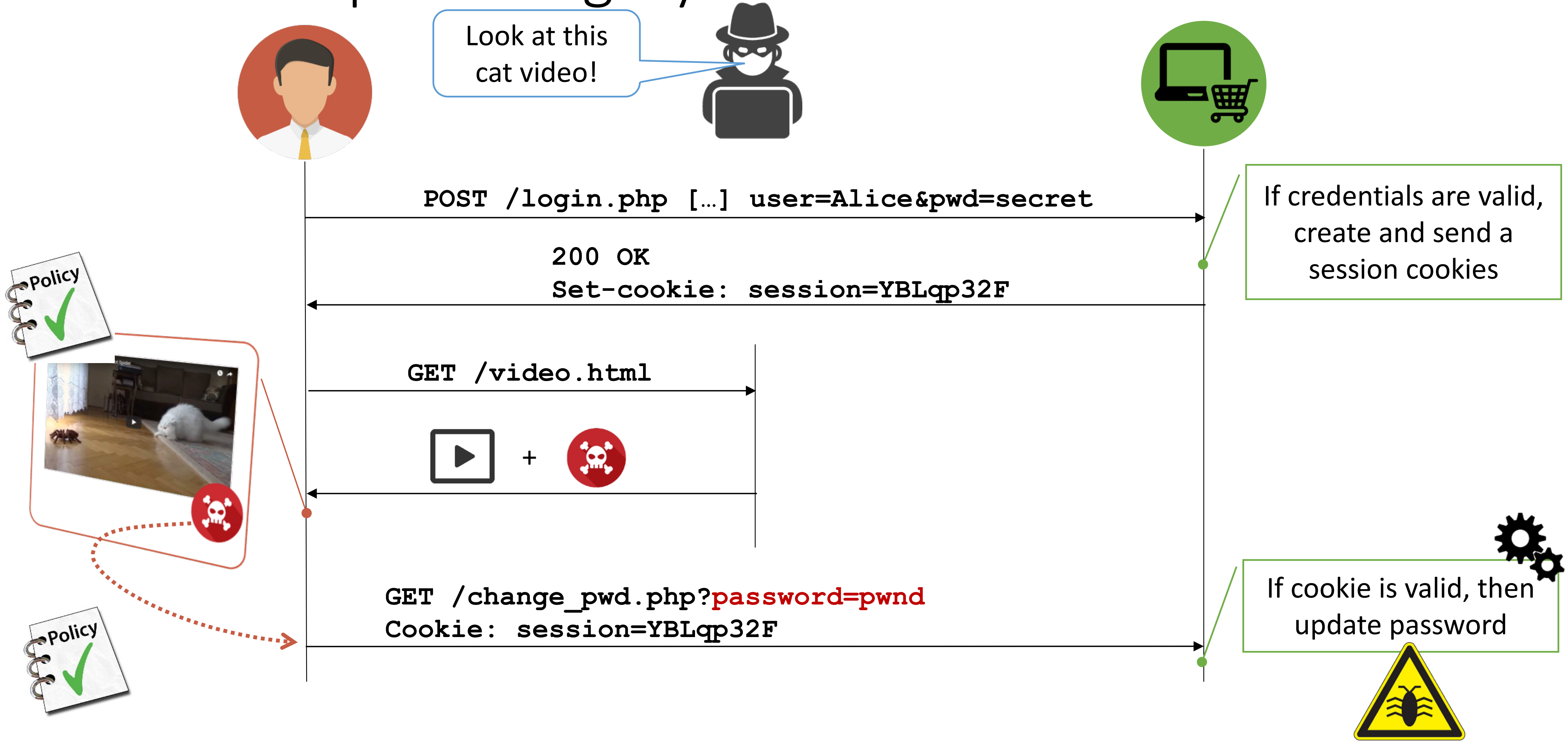


U WON’T BELIEVE WHAT DIS CAT IS DOIN’ !!!1!

Snowflake the Cat Reacts to RC Robot Spider

```
<img  src="http://store.com/change_pwd.php?password=pwnd"
         width="0px" height="0px"/>
```

Cross-Site Request Forgery Attack



Look at this cat video!

```
POST /login.php […] user=Alice&pwd=secret
```

If credentials are valid, create and send a session cookies

```
200 OK
Set-cookie: session=YBLqp32F
```

```
GET /video.html
```

```
GET /change_pwd.php?password=pwnd
Cookie: session=YBLqp32F
```

If cookie is valid, then update password

The Forgotten Sleeping Giant

- Popular vulnerability
  - Among top 10 security risks w/ XSS and SQLi [Top10_OWASP_2007-2013]
  - Discovered in popular websites, e.g., Gmail, Netflix, and ING

- Most of previous efforts spent on countermeasures:
  - Origin header, synchronizer tokens, and browser plugins

- A little has been done to provide techniques for the detection
  - Existing (semi-)automated techniques focus on input validation and logic flaws

→Detection of CSRF via manual inspection

Building a tool to find CSRF

- CSRF is not overly hard to find for pen testers or security experts during dedicated security testing
- But
  - Bug pattern is unintuitive for developers
  - Security testing is often used in automated processes, such as Q-Gates or regression testing
- Hence,
  - Can we build a tool to find CSRF automatically?

So, why is it hard to detect CSRF automatically?

- Challenges (Operational):
  1) Application interaction
  2) Side-effect free testing
- Challenges (Detection):
  1) CSRF targets state transitions
  2) Attacker reliably create requests incl. parameters and values
  3) Not all state transitions are relevant

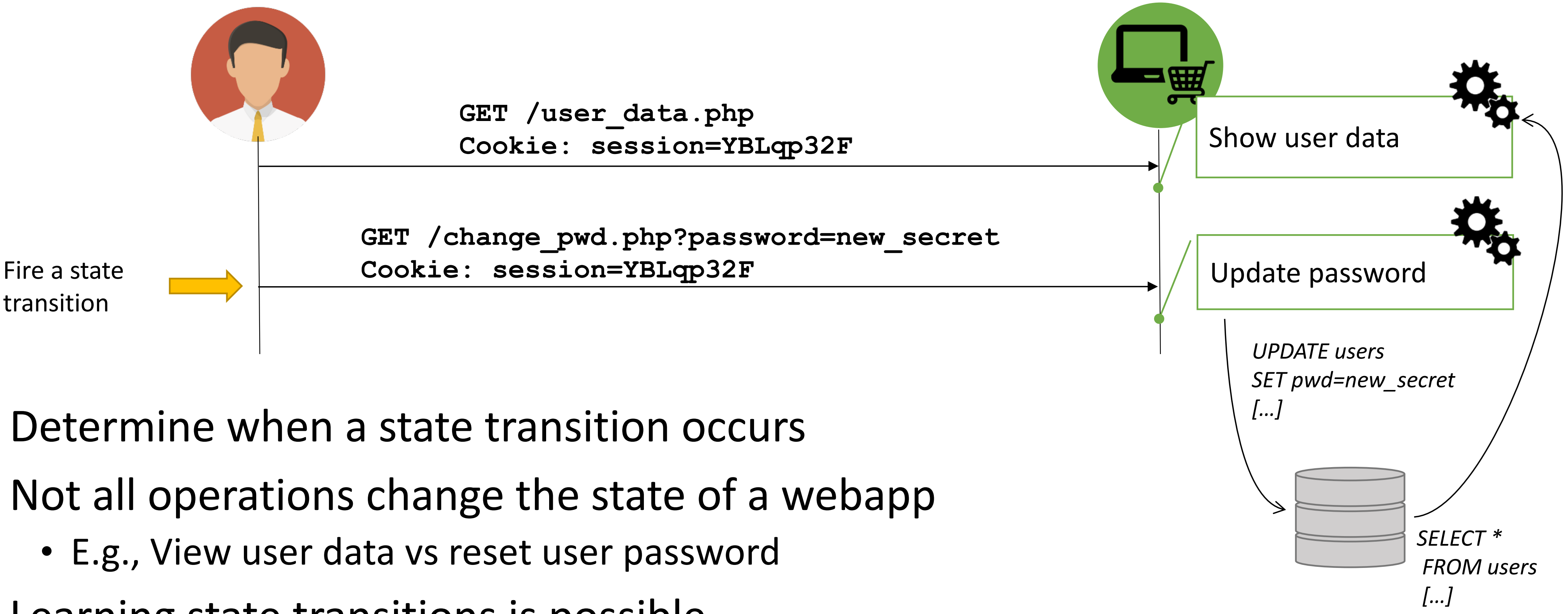
Challenge O1: Application interaction

- CSRF is rarely found on application entry pages
- Instead, in general it requires interaction with deeper functionality of the application
- Thus, “blind” black-box testing is unlikely to access all CSRF-relevant interfaces

Challenge O2: Side-effect free testing

- Remember: CSRF is all about causing lasting side-effects on the server-side
- But:
  - Testing for such side effects potentially causes… *side effects*
- Think:
  - Deletion of a shopping basket
  - Terminating an authenticated session
  - ...
- How can we ensure that our testing does interfere with our testing?

Challenge D1: CSRF Targets State Transitions



```
GET /user_data.php
Cookie: session=YBLqp32F
```

Fire a state transition

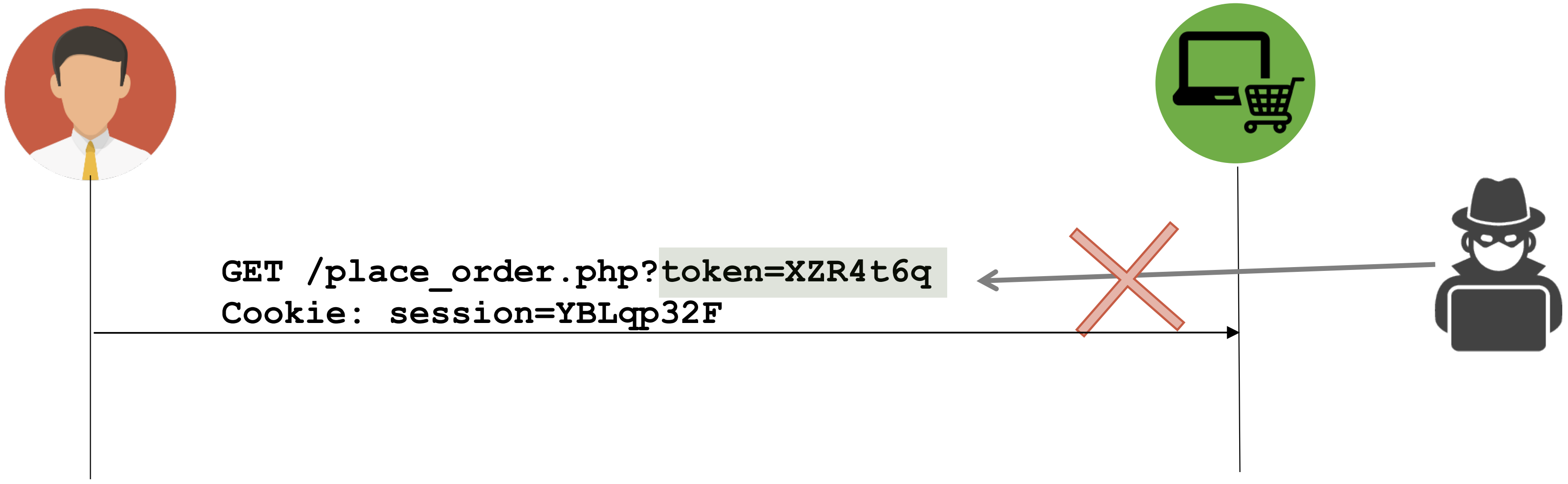
```
GET /change_pwd.php?password=new_secret
Cookie: session=YBLqp32F
```

```
UPDATE users
SET pwd=new_secret
[…]
```

```
SELECT *
 FROM users
[…]
```

- Determine when a state transition occurs
- Not all operations change the state of a webapp
  - E.g., View user data vs reset user password
- Learning state transitions is possible
  - However, existing approach can be inaccurate or operation-specific

Challenge D2: Attacker Reliably Creates Requests

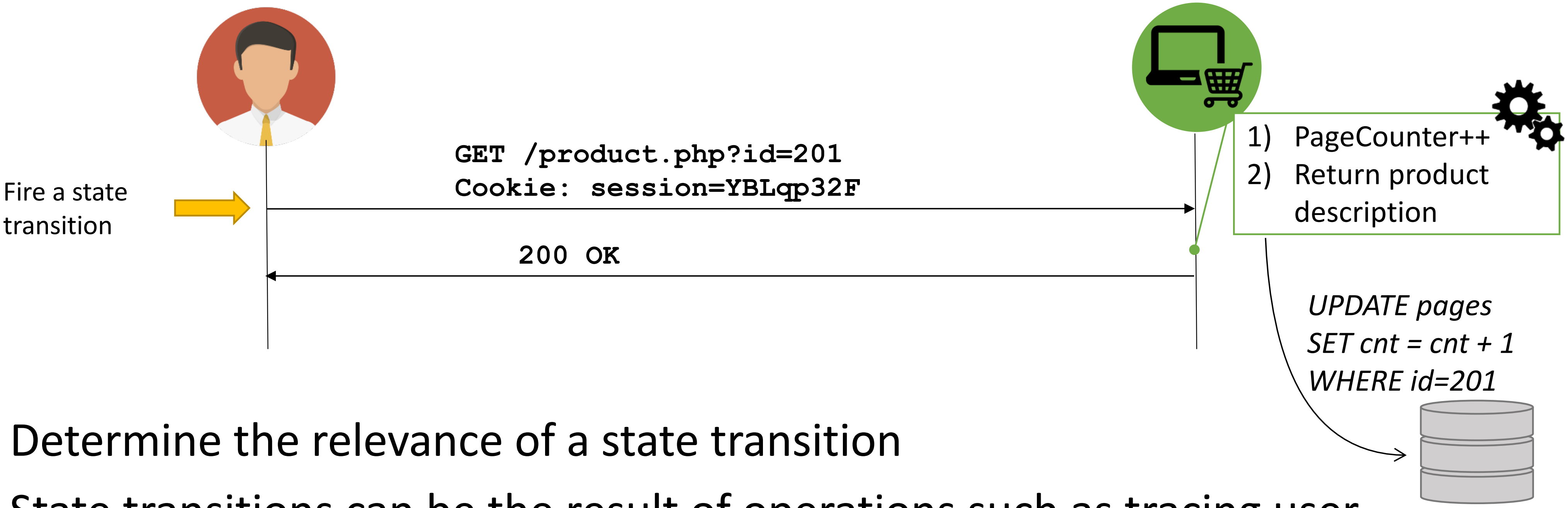


```
GET /place_order.php?token=XZR4t6q
Cookie: session=YBLqp32F
```

- Determine relationships between parameters and transitions
  - E.g., random security token may not be guessed by an attacker

- Existing techniques do not determine such a relationship
  - E.g., Web scanners match param names against list of predefined names (e.g., “token”)

Challenge D3: Not all State Transitions are Relevant



Fire a state transition

```
GET /product.php?id=201
Cookie: session=YBLqp32F
```

```
200 OK
```

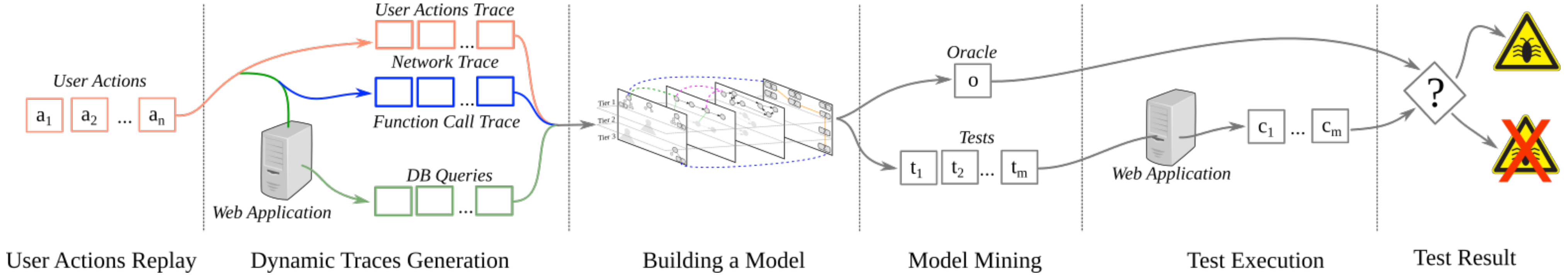
1) PageCounter++
2) Return product description

UPDATE pages
SET cnt = cnt + 1
WHERE id=201

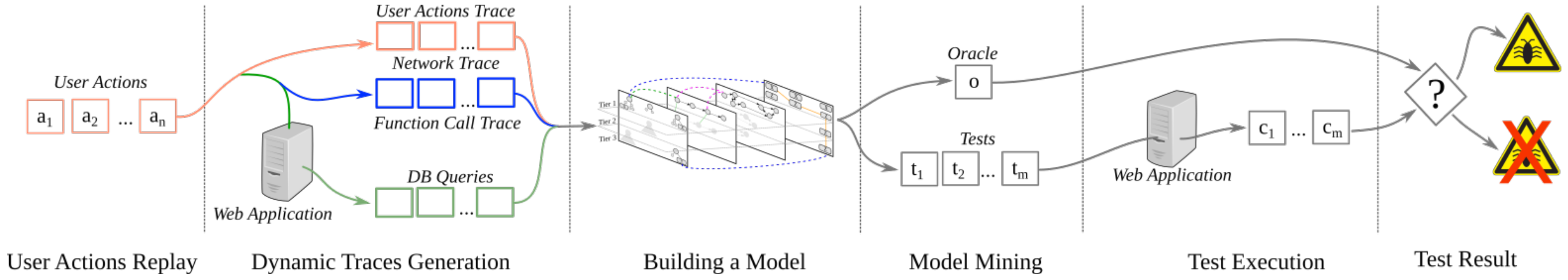
- Determine the relevance of a state transition
- State transitions can be the result of operations such as tracing user activities
  - They are state-changing operations but <u>not necessarily security-relevant</u>
- Easy for humans but hard for machines

Our approach: Deemon

- Approach: Guided grey-box testing
- Input: User generated interaction traces
  - E.g., Selenium scripts for regression/UI testing
- Infrastructure
  - HTTP observation
  - Instrumented server-side that monitors all state changes

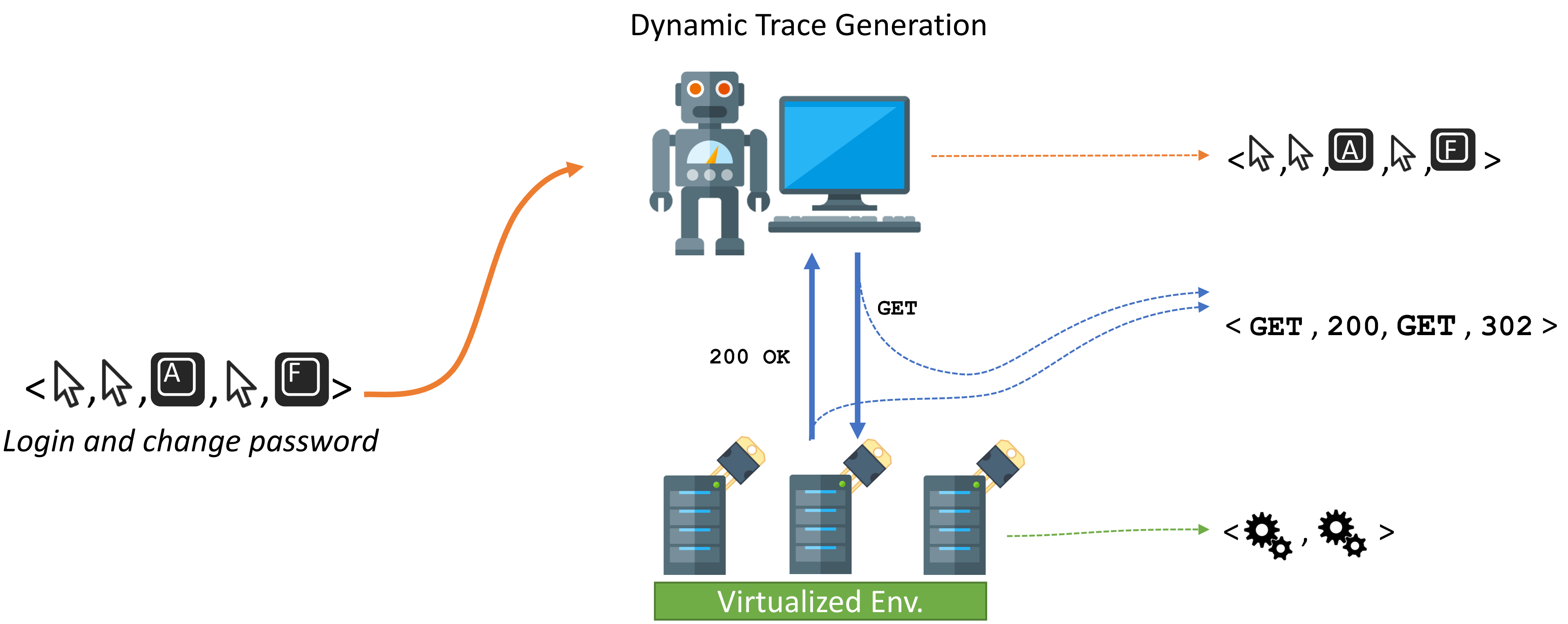


Our Solution: Deemon



- Application-agnostic framework for developers and analysts
    1. Infer state transitions + data flow from program executions
    2. Property graphs for uniform and reusable model representation
    3. Graph traversals to select request candidates for testing
    4. Verify replay-ability of HTTP requests

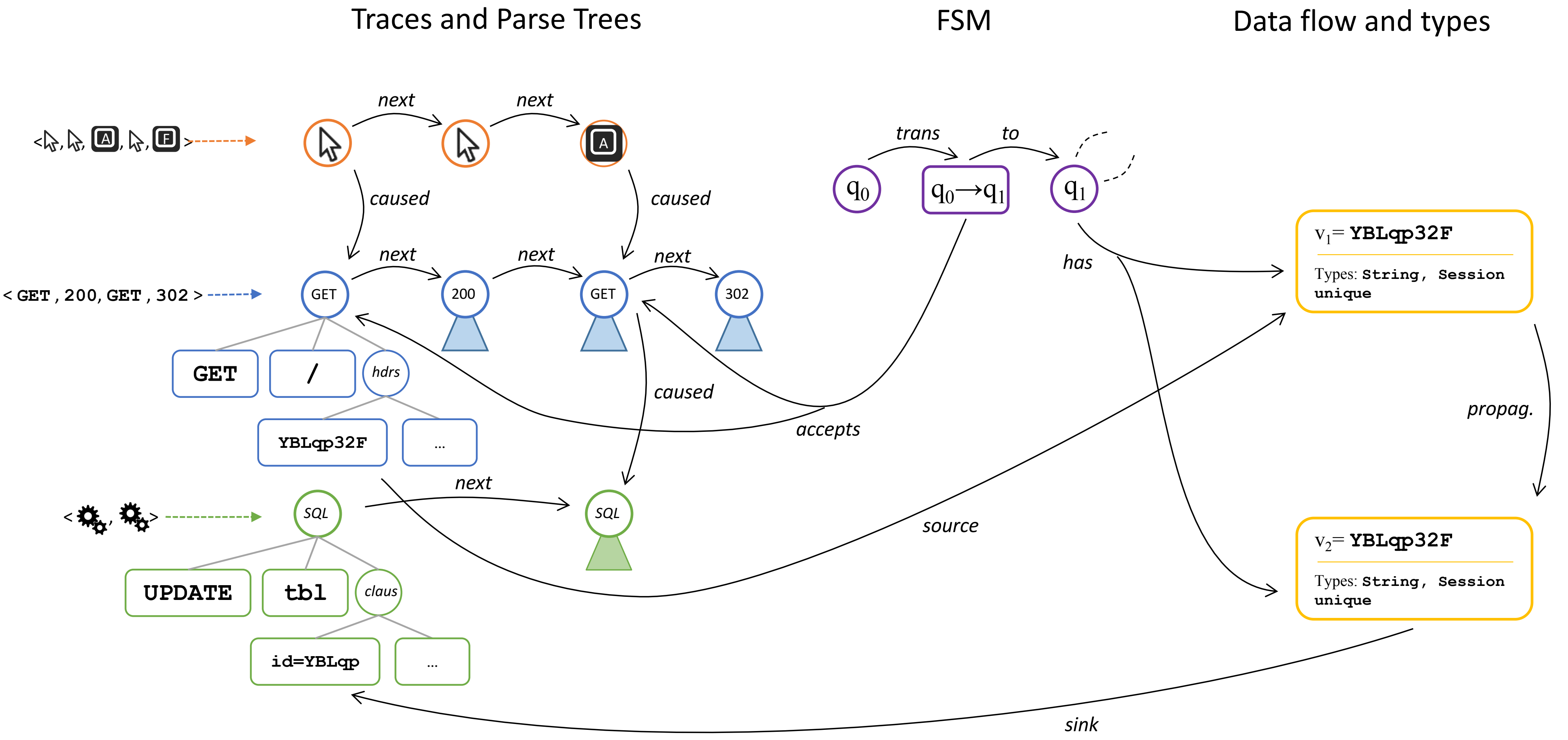
Deemon: Architecture



Reliable, repeatable workflow testing

- The architecture allows side-effect free testing
  - Set server VM into vanilla state
  - Run UI workflow and record all traffic & server-side effects
  - …repeat
- Clear mapping between: UI interaction / HTTP requests / server-side effects
  - This allow the identification of single requests between traces
- Running the same UI workflow multiple times and comparing HTTP request parameters
  - With the same user -> session specific parameters
  - With different users -> user specific parameters

Deemon: Model Construction



Deemon: Traversals

“Find all CSRF”

⇓

“Find all **requests** r such that:

1) r is **state-changing**

2) r can be **created** by an attacker

3) the state change is **relevant**”

⇓

“$\forall n$: request(n)

1) $\exists tr, q_i, q_f$: trans(tr, q_i, q_f)

$\land$ accepts(tr, n)

2) $\forall v$: variable(v)

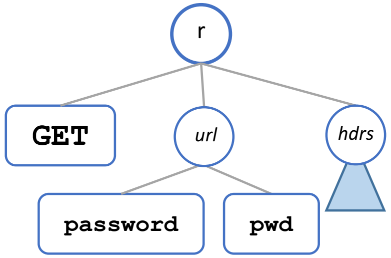
$\land$ has(q_f, v)

$\land$ v.Types $\cap$ {“unguessable”} $= \emptyset$

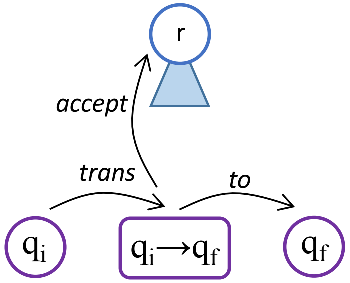
3) relevant(r)”

⇓

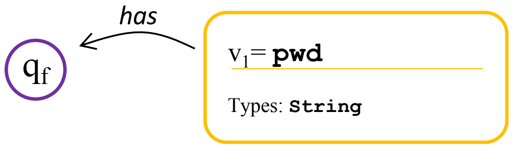
[Query processor]



request(r)

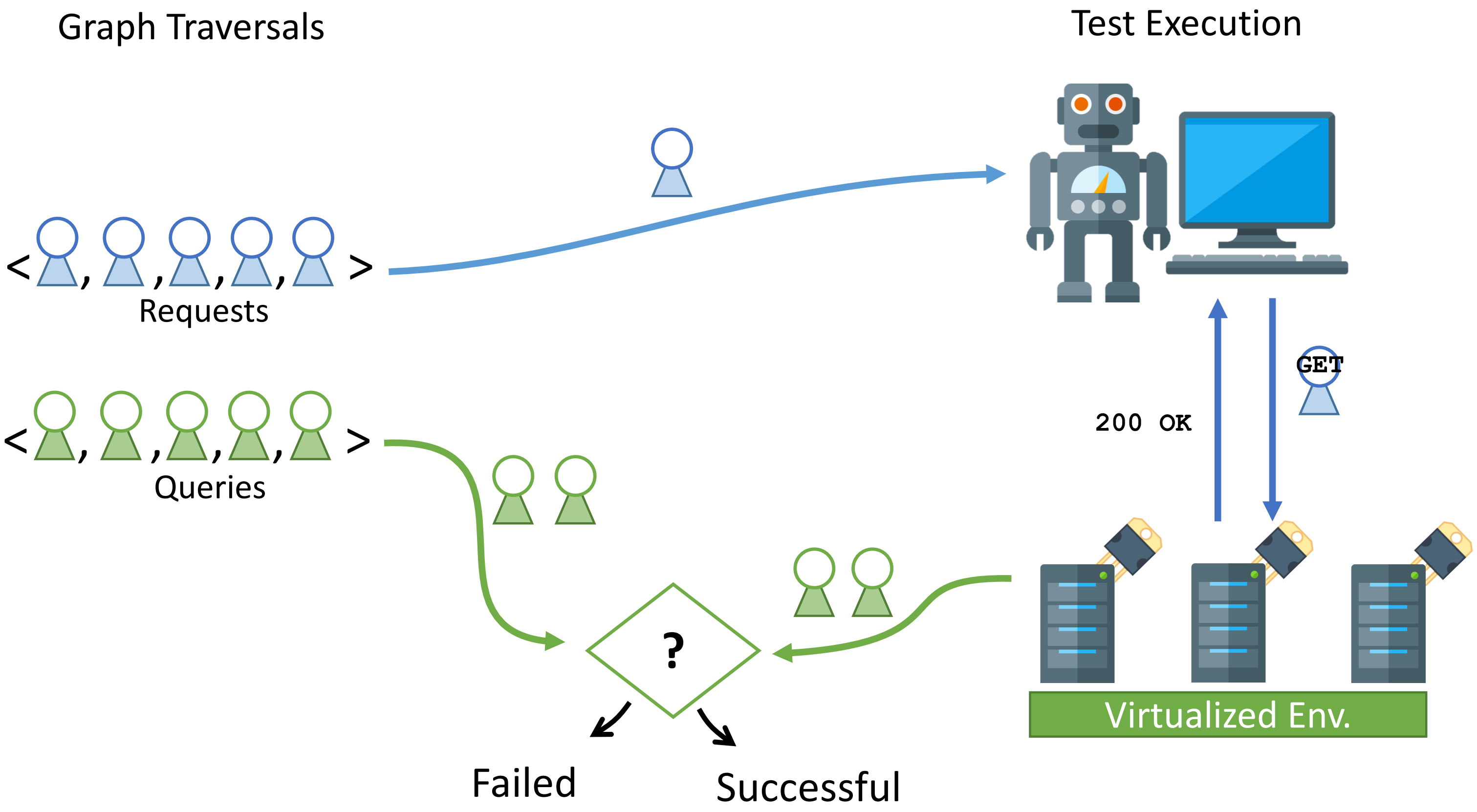


$\exists tr, q_i, q_f$: trans(tr, q_i, q_f) $\land$ accepts(tr, r)



$\forall v$: variable(v) $\land$ has(q_f, v) $\land$ v.Types $\cap$ {“unguessable”} $= \emptyset$

Deemon: Testing

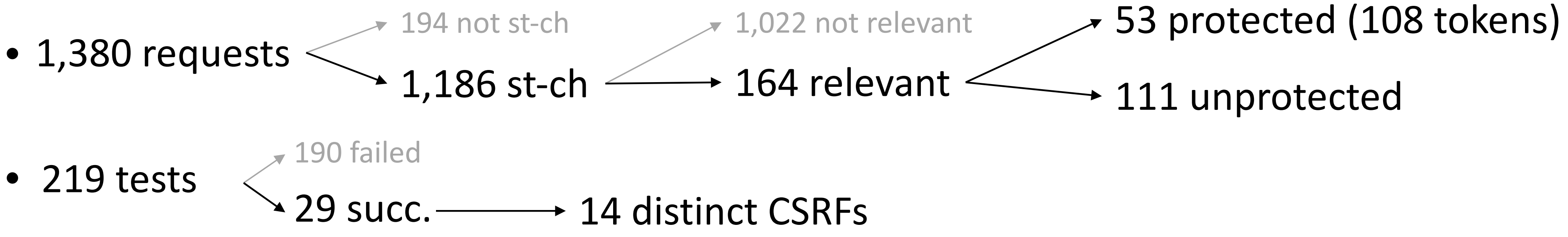


Revisiting the Challenges

- O1) Application interaction
  - Guided testing via recorded workflows
- O2) Side-effect free testing
  - Removal of side effects via VM snapshots
- D1) CSRF targets state transitions
  - Monitoring of server-side effects
- D2) Attacker reliably create requests incl. parameters and values
  - Automated analysis of parameter roles and information flows
- D3) Not all state transitions are relevant
  - Removal of non-authentication and generic state transitions

Evaluation

- Inputs:
  - 10 Web apps from the Bitnami catalog (avg 600k LoC)
  - 93 workflows (e.g., change password, username, add/delete user/admin, enable/disable plugin)



- 1,380 requests → 194 not st-ch; 1,186 st-ch
  - 1,186 st-ch → 1,022 not relevant; 164 relevant
  - 164 relevant → 53 protected (108 tokens); 111 unprotected
- 219 tests → 190 failed; 29 succ.
  - 29 succ. → 14 distinct CSRFs

- Attacks:
  - User account takeover in AbanteCart and OpenCart
  - Database corruption in Mautic
  - Web app takeover in Simple Invoices

Results Analysis: Awareness

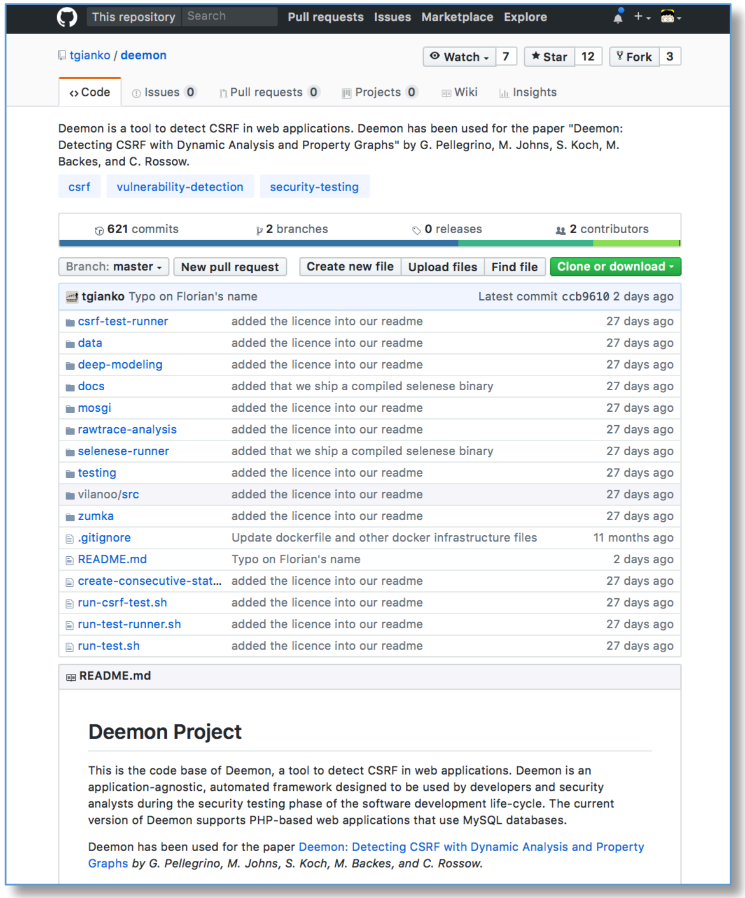
1. **Complete Awareness**: all state-changing operations are protected
   - E.g., Horde, Oxid, and Prestashop

2. **Unawareness**: none of the relevant state-changing operations are protected
   - I.e., Simple Invoices

3. **Partial Awareness**
   - *Role-based*: only admin is protected
     - I.e., OpenCart and AbanteCart
   - *Operation-based*: adding data items is protected, deleting is not
     - I.e., Mautic

Takeaways

- Presented Deemon: Dynamic analysis + property graphs
- Deemon detected 14 CSRFs that can be exploited to takeover accounts, websites, and compromise database integrity
- Discovered alarming behaviors: security-sensitive operations are protected in a selective manner
- Read all the gory details or play with Deemon:
  - G. Pellegrino et al.: **Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs** in 24th ACM Conference on Computer and Communications Security, 2017 (CCS 2017)
  - https://github.com/tgianko/deemon



Deemon is a tool to detect CSRF in web applications. Deemon has been used for the paper "Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs" by G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow.

tgianko Typo on Florian's name		Latest commit ccb9610 2 days ago
csrf-test-runner	added the licence into our readme	27 days ago
data	added the licence into our readme	27 days ago
deep-modeling	added the licence into our readme	27 days ago
docs	added that we ship a compiled selenese binary	27 days ago
mosgi	added the licence into our readme	27 days ago
rawtrace-analysis	added the licence into our readme	27 days ago
selenese-runner	added that we ship a compiled selenese binary	27 days ago
testing	added the licence into our readme	27 days ago
vilanoo/src	added the licence into our readme	27 days ago
zumka	added the licence into our readme	27 days ago
.gitignore	Update dockerfile and other docker infrastructure files	11 months ago
README.md	Typo on Florian's name	2 days ago
create-consecutive-stat…	added the licence into our readme	27 days ago
run-csrf-test.sh	added the licence into our readme	27 days ago
run-test-runner.sh	added the licence into our readme	27 days ago
run-test.sh	added the licence into our readme	27 days ago

README.md

Deemon Project

This is the code base of Deemon, a tool to detect CSRF in web applications. Deemon is an application-agnostic, automated framework designed to be used by developers and security analysts during the security testing phase of the software development life-cycle. The current version of Deemon supports PHP-based web applications that use MySQL databases.

Deemon has been used for the paper Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs *by G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow.*