Center for IT-Security, Privacy and Accountability

Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs

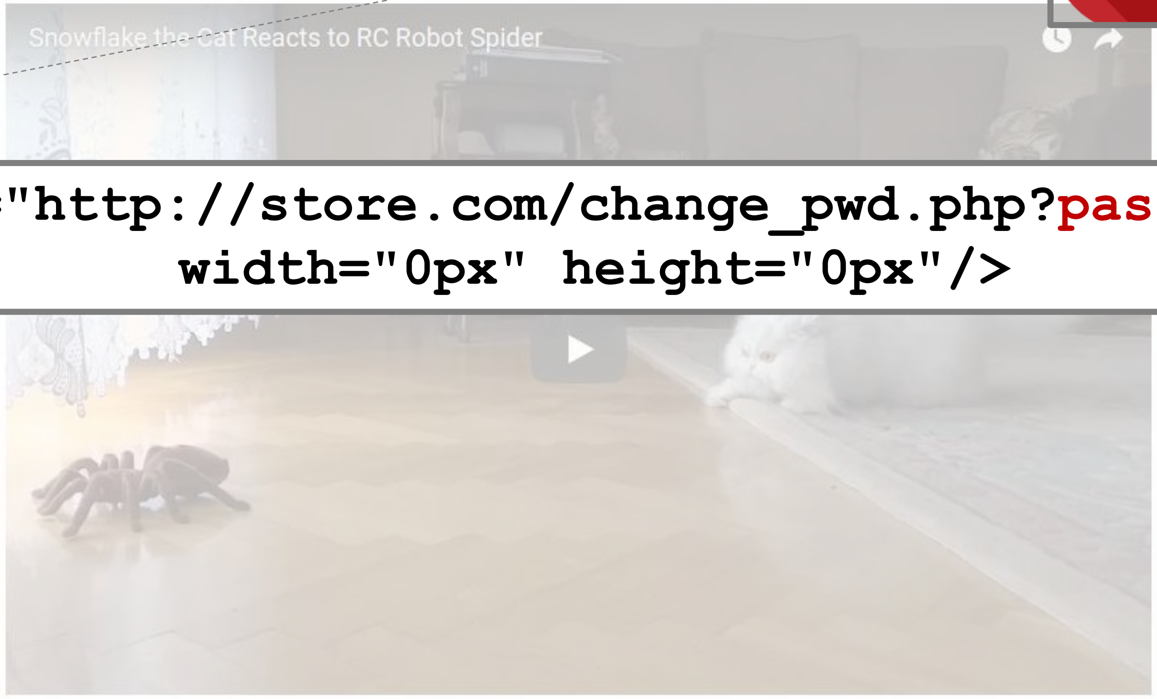
G. Pellegrino, M. Johns, S. Koch, M. Backes, C. Rossow

gpellegrino@cispa.saarland

ACM CCS 2017

Nov 2nd, Dallas, USA

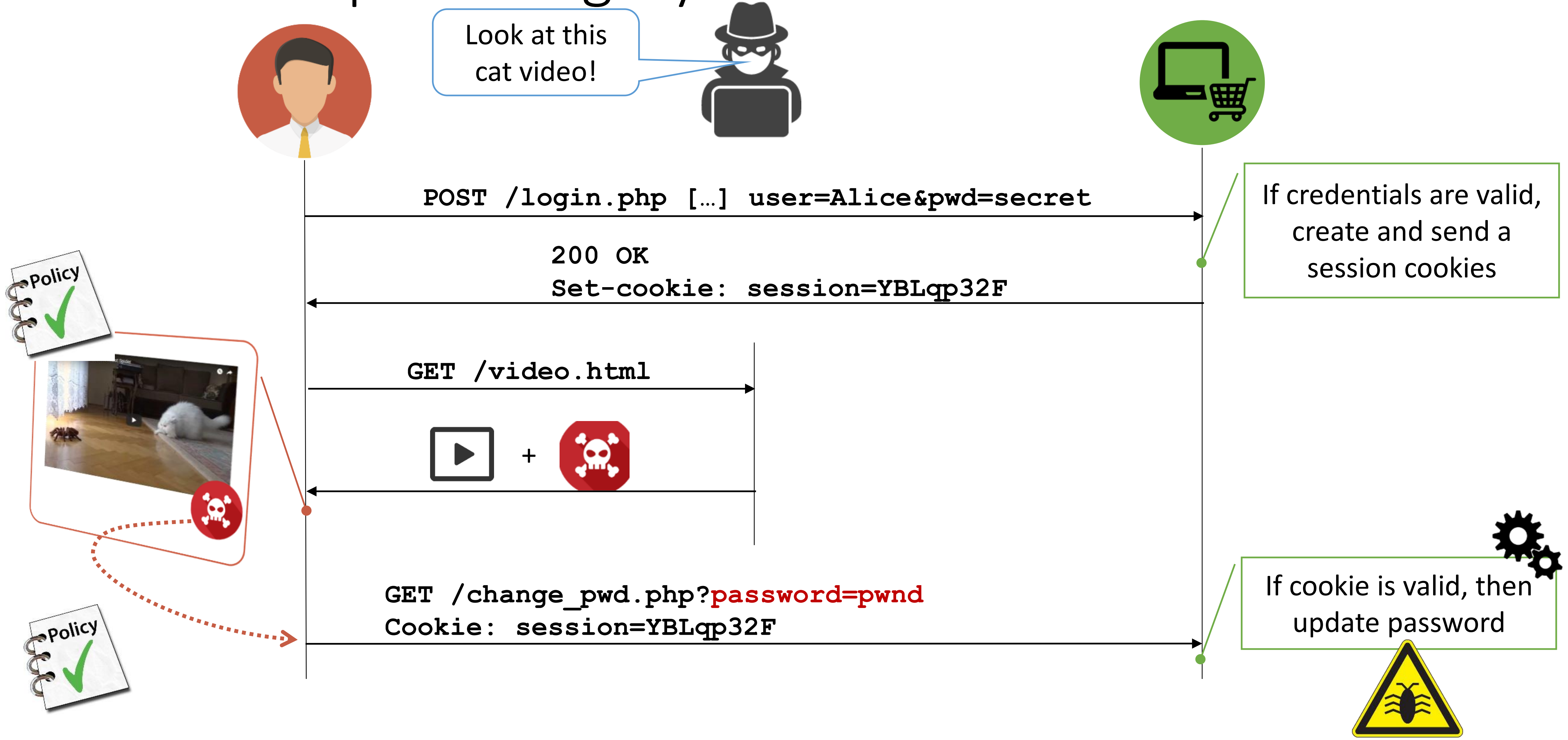
U WON'T BELIEVE WHAT DIS CAT IS DOIN' !!!1!



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

TWEET SHARE PIN SEND EMAIL

Cross-Site Request Forgery Attack


Look at this cat video!
POST /login.php [...] user=Alice&pwd=secret
200 OK
Set-cookie: session=YBLqp32F
If credentials are valid, create and send a session cookies
Policy
GET /video.html
+
GET /change_pwd.php?password=pwnd
Cookie: session=YBLqp32F
If cookie is valid, then update password
Policy

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

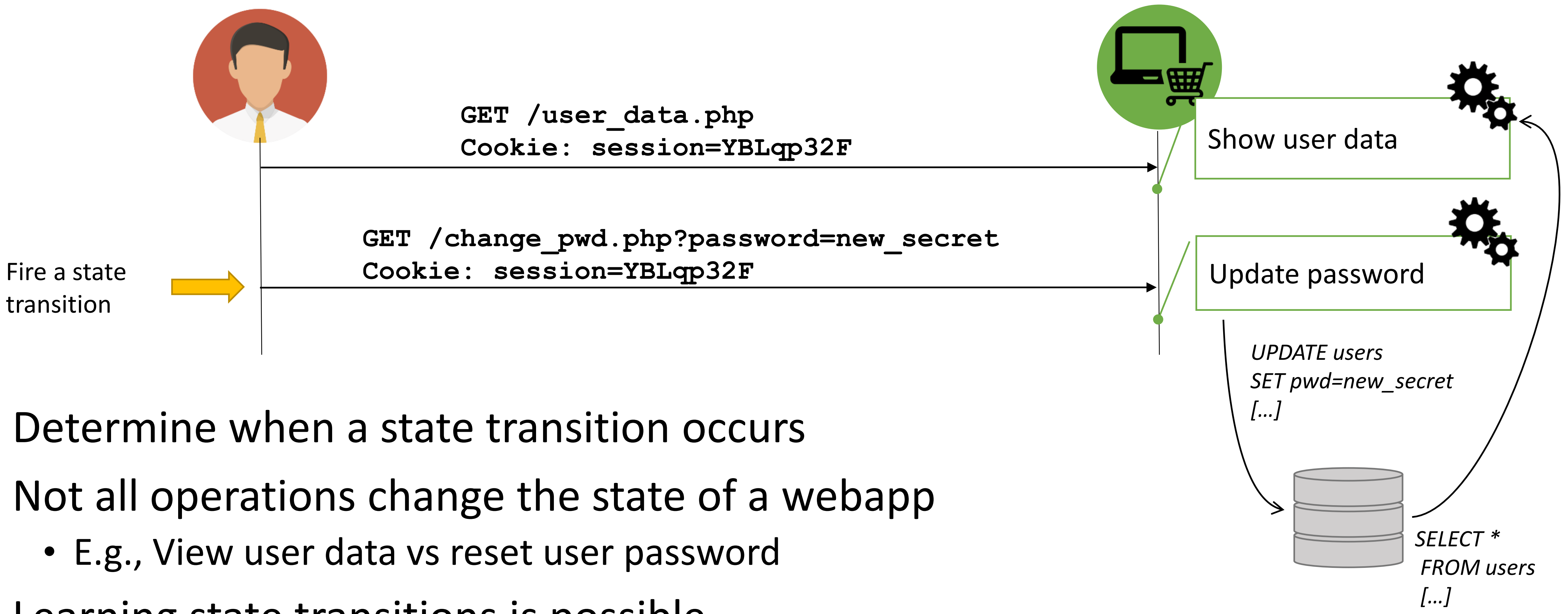
  →Detection of CSRF via manual inspection

Challenges

- Detection requires reasoning over relationships between application states, the roles and status of request parameters

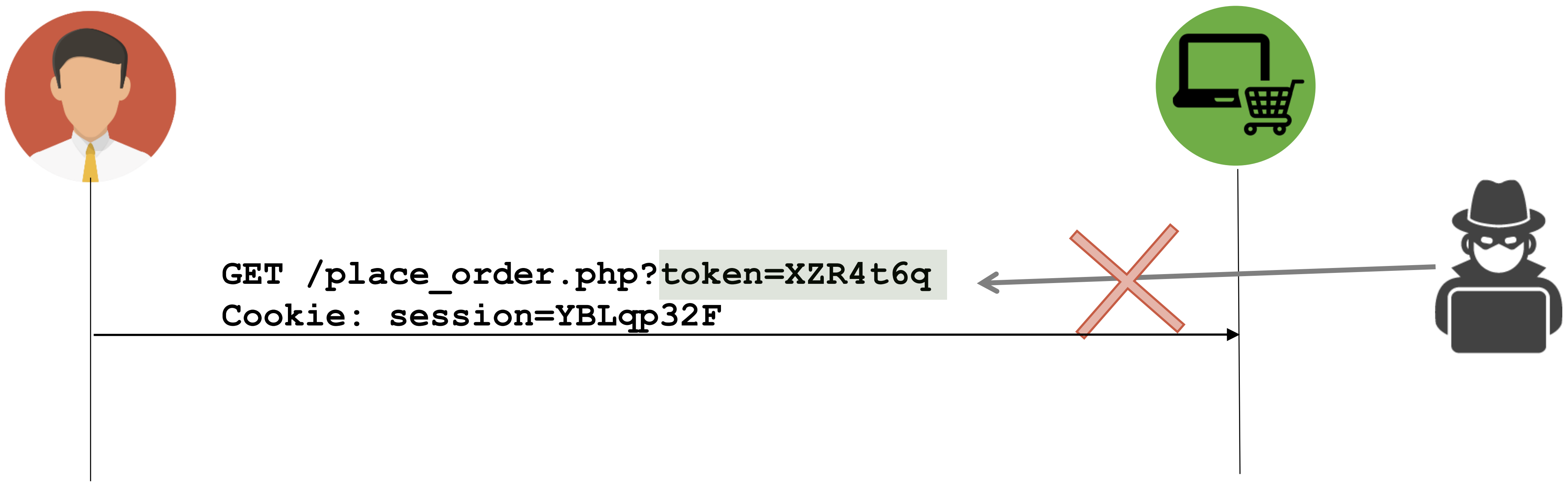
- Challenges:
  1) CSRF targets state transitions
  2) Attacker reliably create requests incl. parameters and values
  3) Not all state transitions are relevant

1) CSRF Targets State Transitions



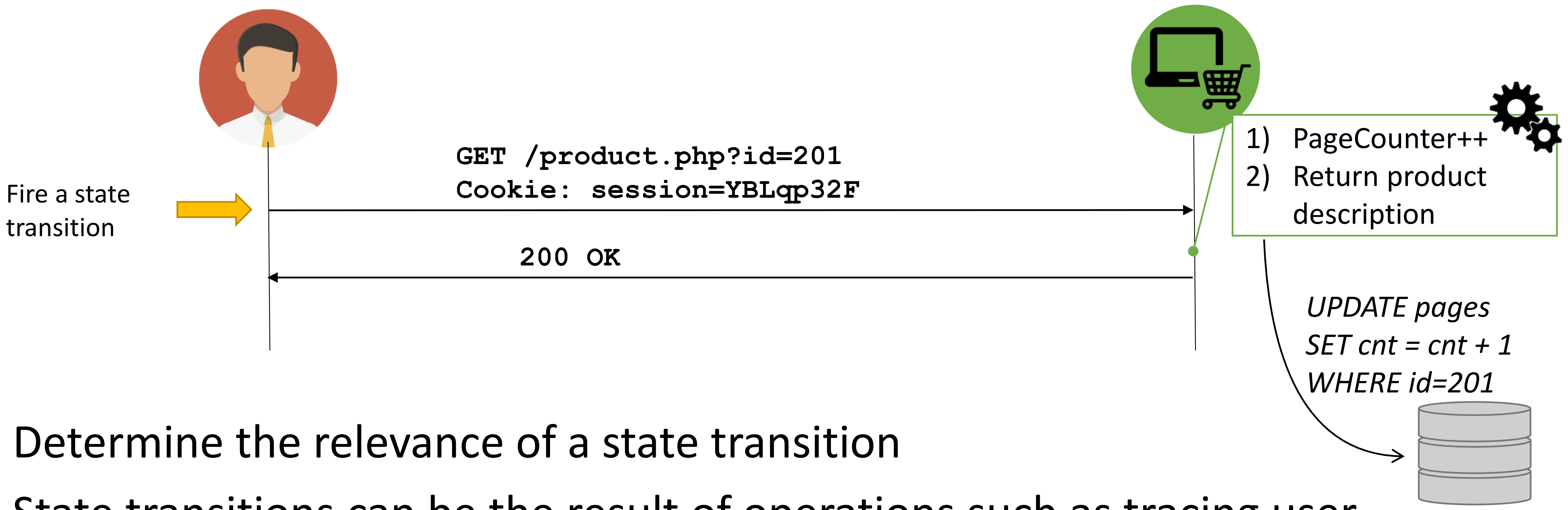
- Determine when a state transition occurs
- Not all operations change the state of a webapp
  - E.g., View user data vs reset user password
- Learning state transitions is possible
  - However, existing approach can be inaccurate or operation-specific

2) Attacker Reliably Creates Requests incl. Params



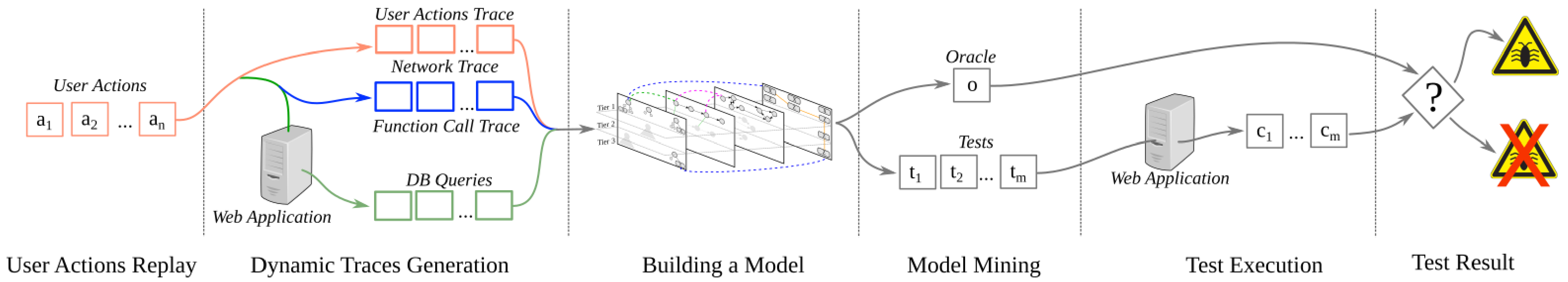
- Determine relationships between parameters and transitions
  - E.g., random security token may not be guessed by an attacker
- Existing techniques do not determine such a relationship
  - E.g., Web scanners match param names against list of predefined names (e.g., “token”)

3) Not all State Transitions are Relevant



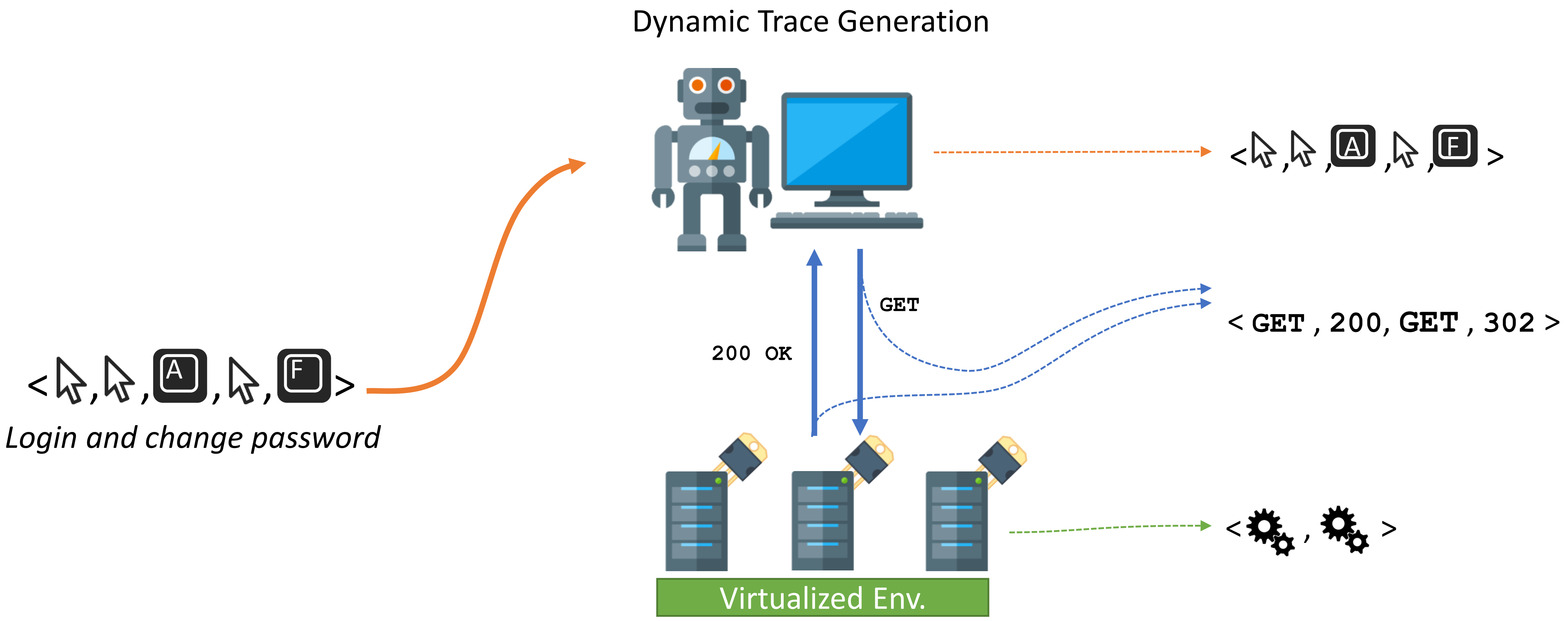
- Determine the relevance of a state transition
- State transitions can be the result of operations such as tracing user activities
  - They are state-changing operations but <u>not necessarily security-relevant</u>
- Easy for humans but hard for machines

Our Solution: Deemon

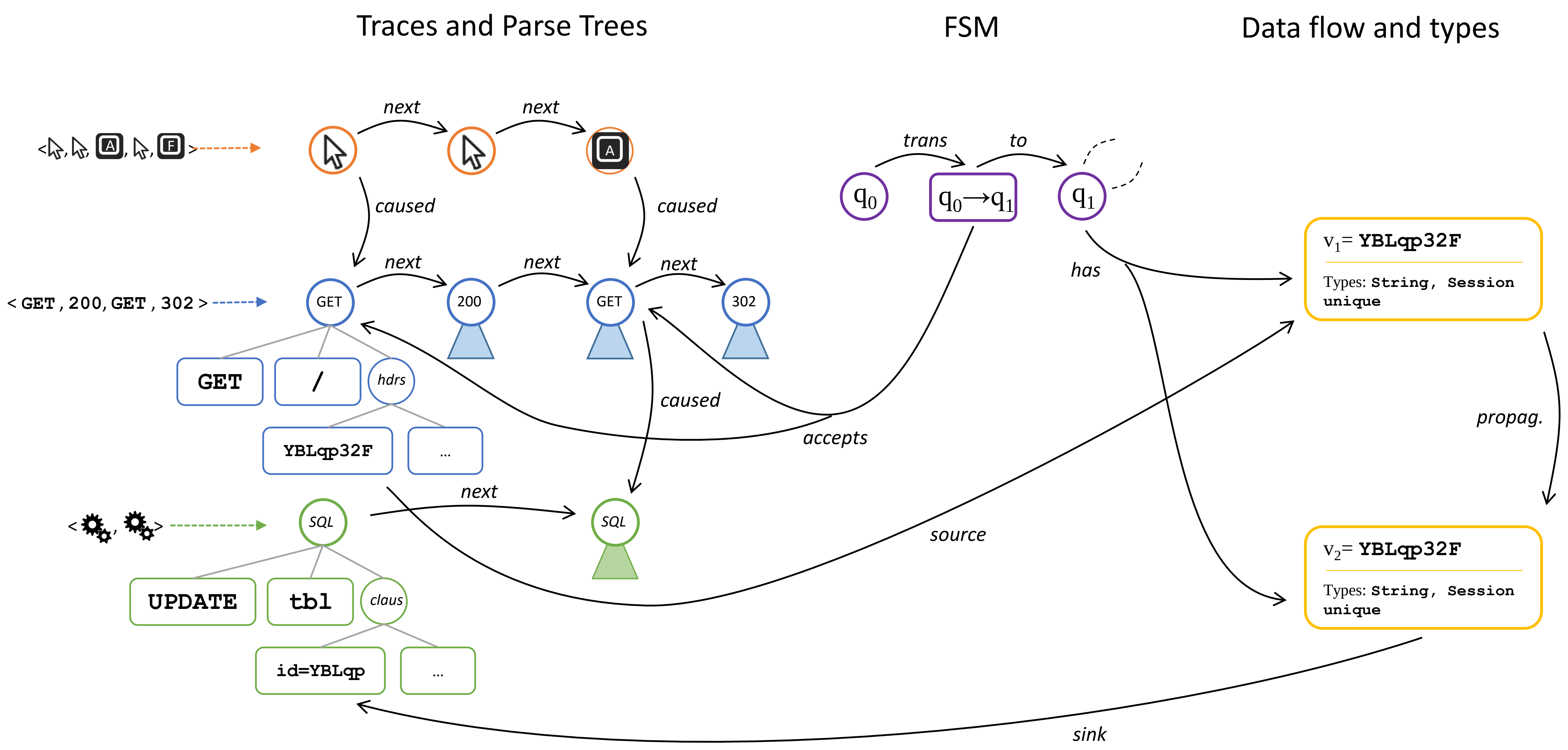


- Application-agnostic framework for developers and analysts
  1. Infer state transitions + data flow from program executions
  2. Property graphs for uniform and reusable model representation
  3. Graph traversals to select request candidates for testing
  4. Verify replay-ability of HTTP requests

Deemon: Trace Generation



Deemon: Model Construction



Deemon: Traversals

"Find all CSRF"

$\Downarrow$

"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**"

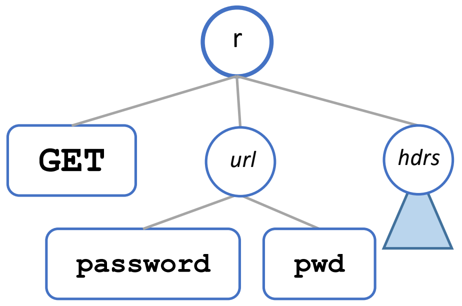
$\Downarrow$

"$\forall n$: request(n)

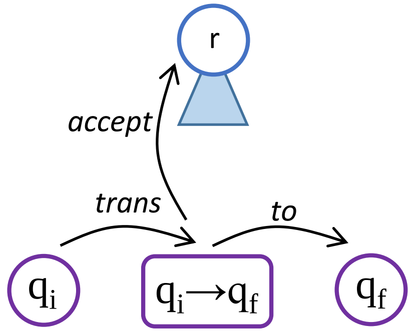
1) $\exists tr, q_i, q_f$: trans(tr, q_i, q_f)
   $\wedge$ accepts(tr, n)
2) $\forall$ v: variable(v)
   $\wedge$ has(q_f, v)
   $\wedge$ v.Types $\cap$ {"unguessable"} = $\emptyset$
3) relevant(r)"

$\Downarrow$

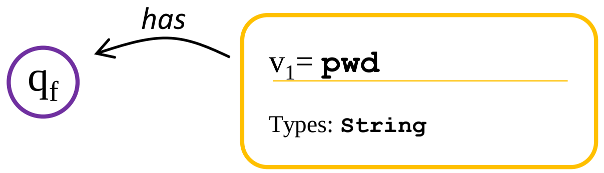
[Query processor]



request(r)

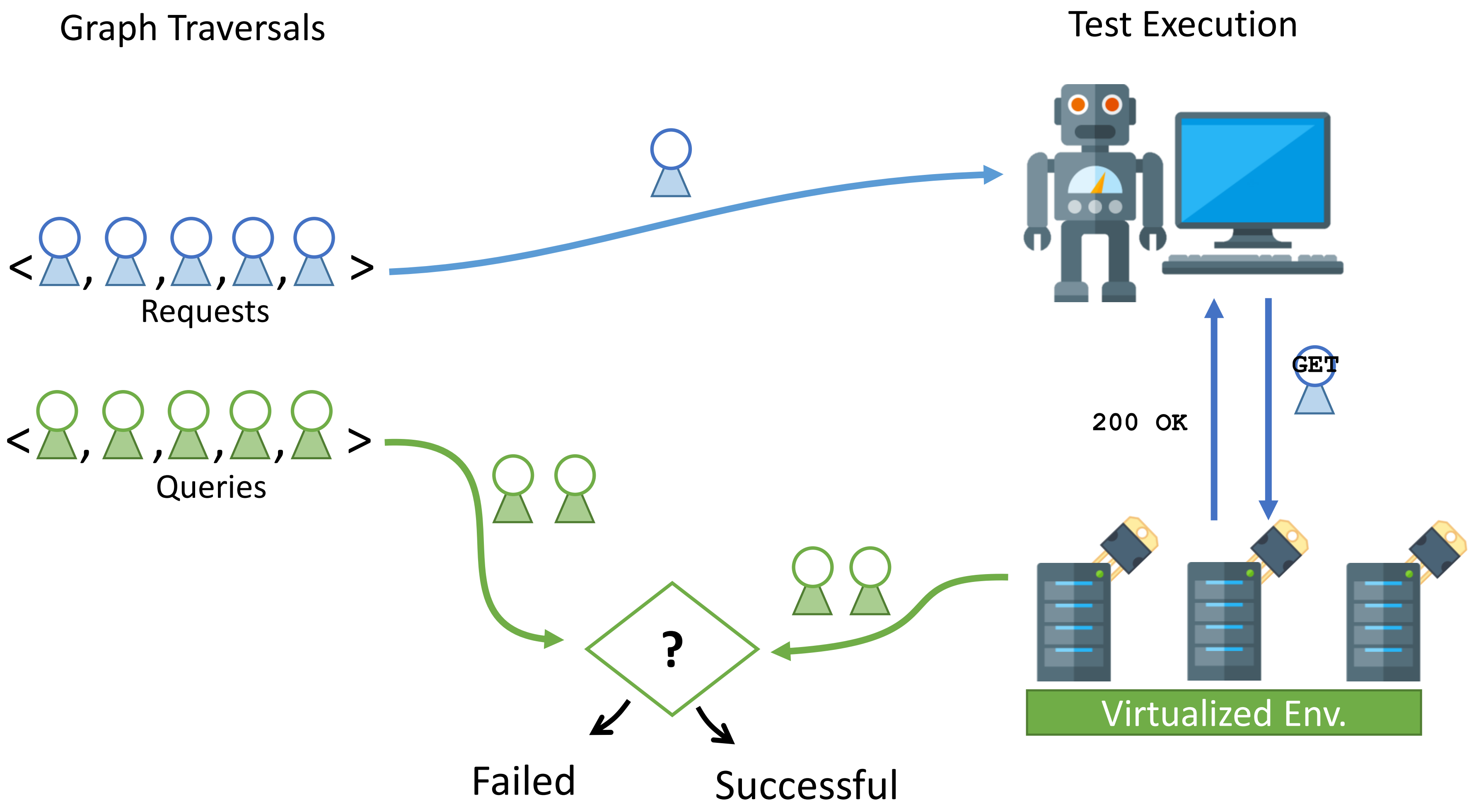


$\exists tr, q_i, q_f$: trans(tr, q_i, q_f) $\wedge$ accepts(tr, r)



$\forall$ v: variable(v) $\wedge$ has(q_f, v) $\wedge$ v.Types $\cap$ {"unguessable"} = $\emptyset$

Deemon: Testing



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)

- $$\begin{tikzcd}
 & \text{194 not st-ch} & \text{1,022 not relevant} & \text{53 protected (108 tokens)} \\
\text{1,380 requests} \arrow[ur] \arrow[r] & \text{1,186 st-ch} \arrow[ur] \arrow[r] & \text{164 relevant} \arrow[ur] \arrow[r] & \text{111 unprotected}
\end{tikzcd}$$

- $$\begin{tikzcd}
 & \text{190 failed} & \\
\text{219 tests} \arrow[ur] \arrow[r] & \text{29 succ.} \arrow[r] & \text{14 distinct CSRFs}
\end{tikzcd}$$

- 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
  - New modeling paradigm
- 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