Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications

Adam Chlipala
OSDI 2010

Beyond Code Injection

1. Injection
2. Cross Site Scripting

An application includes an unintended run-time program interpreter

3. Broken Authentication and Session Mgmt.
4. Insecure Direct Object References

Dependent on application-specific authentication and access control regimes!

5. Cross Site Request Forgery
6. Security Misconfiguration
7. Insecure Cryptographic Storage

....

Authentication Snafus

National Cyber-Alert System

Vulnerabil

Original re

Last revis

Source: U

Overvi

WB News 2
authenticati
modified WI
setting this

National Cyber-Alert System

Vulnerabilit

Original rel

Last revise

Source: US

Overvie

siteadmin/ac
PageDirecto
which allows
restrictions a
request.

National Cyber-Alert System

Vulnerability Summary for CVE-2009-4929

Original release date: 07/12/2010

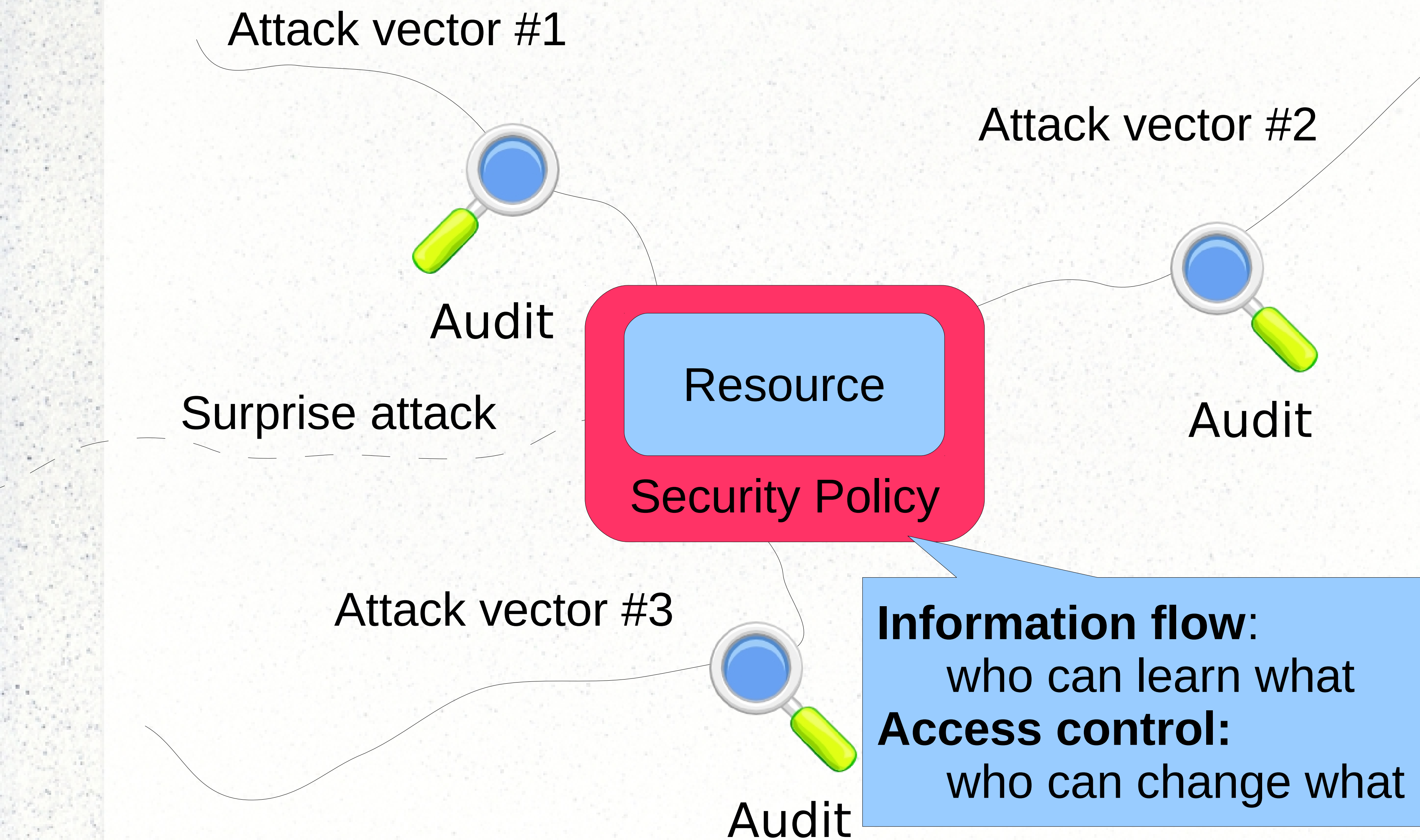
Last revised: 07/16/2010

Source: US-CERT/NIST

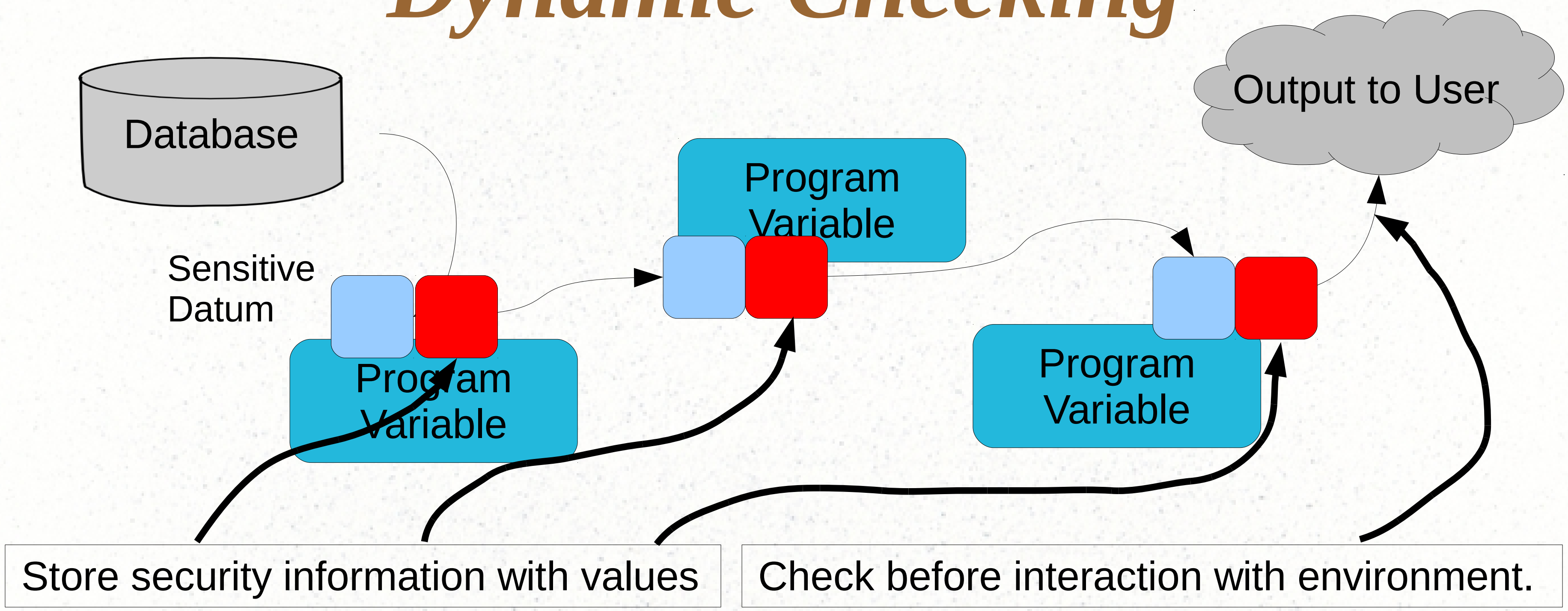
Overview

admin/manage_users.php in TotalCalendar 2.4 does not require administrative authentication, which allows remote attackers to change arbitrary passwords via the newPW1 and newPW2 parameters.

Roads to Security



Dynamic Checking



Pros

•Easy to add to existing programs

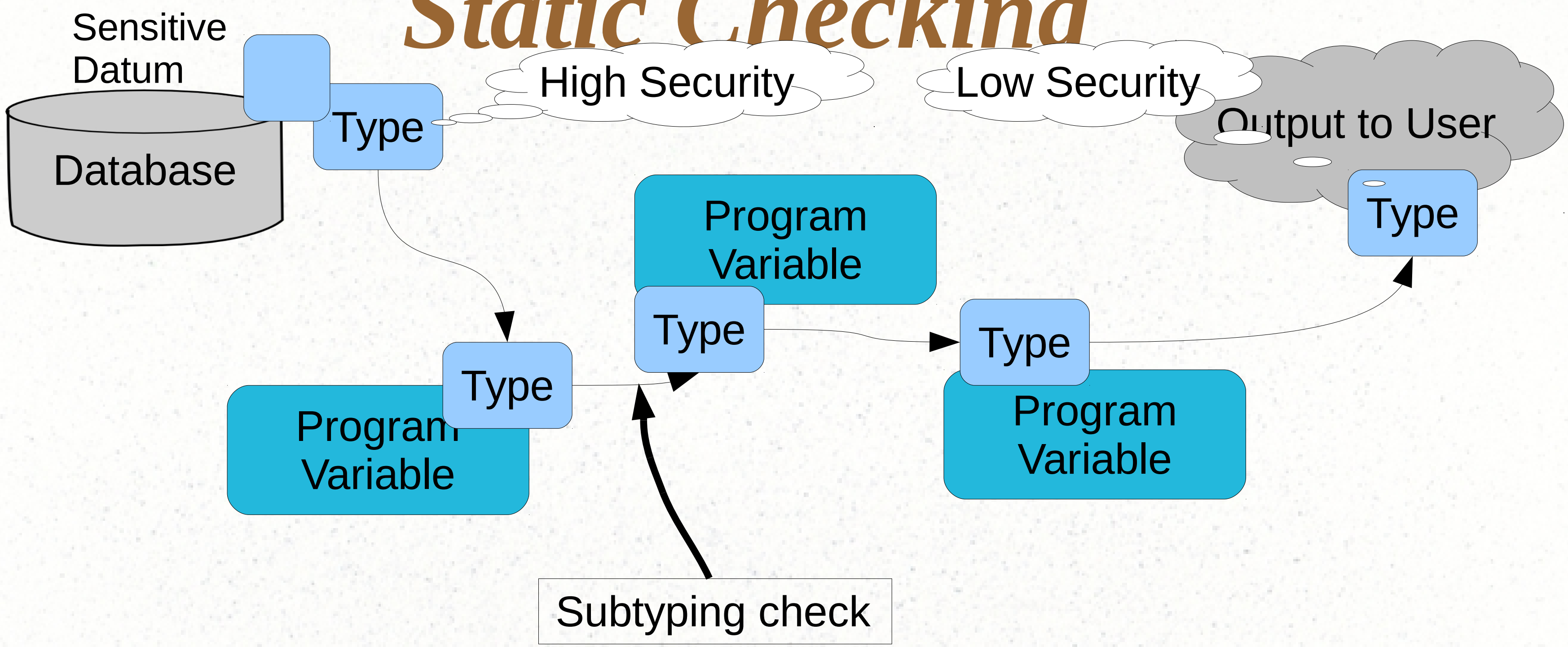
•Flexibility in coding security checks

Cons

•Bugs are only found for program paths that are tested.

•Performance overhead

Static Checking



Pros

- Checks all program paths at compile time
- No changes to run-time behavior required

Cons

- Usually requires extensive program annotation
- Limited policy expressiveness

The Best of Both Worlds

<u>Like Dynamic Checking:</u>

- No program annotations required
- Flexible and programmer-accessible policy language **(SQL)**

The
UrFlow
analysis

for the
Ur/Web
programming language

<u>Like Security Typing:</u>

- Checks all program paths statically
- No run-time overhead

A Word About Ur/Web

```
queryX (SELECT * FROM t)
  (fn row => <xml><tr>
     <td>{[row.T.A]}</td>
     <td>{[row.T.B]}</td>
     <td>{[row.T.C]}</td>
     <td>{[row.T.D]}</td>
     <td><form><submit
        action={delete row.T.A}
        value="Delete"/>
     </form></td>
   </tr></xml>);
```

Integrated parsing and type-checking of SQL and HTML

Simple Policies

<u>Secrets</u>

Id	Name	Secret

Client may learn anything this query could return.

```
policy sendClient
    (SELECT Id, Name
     FROM Secrets)
```

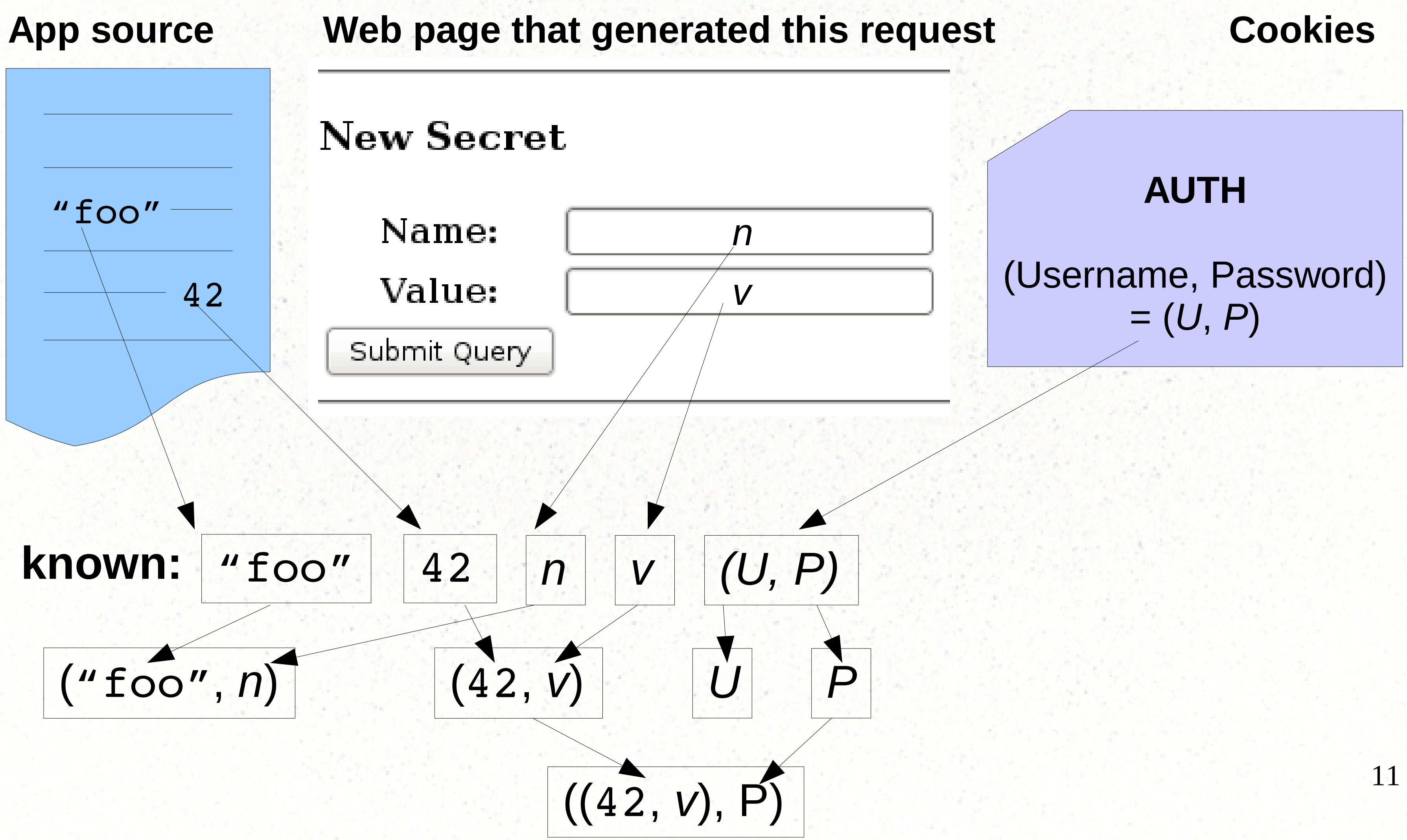
Reasoning About Knowledge

Secrets

Id	Name	Secret	Code

```
policy sendClient
     (SELECT *
      FROM Secrets
      WHERE known(Code))
```

What is “known”?



Multi-Table Policies

Secrets

Id	Name	Secret	Owner

Users

Id	Name	Password

```
policy sendClient
    (SELECT Secret
     FROM Secrets, Users
     WHERE Owner = Users.Id
       AND known(Password))
```

Understanding SQL Usage

Program Execution

```
...
...
(U, P) = readCookie(AUTH);
pass = SELECT Password
       FROM Users
       WHERE Id = U;
if (pass != P) abort();
...
...
...
rows = SELECT Secret
       FROM Secrets
       WHERE Owner = U;
// Send rows to client....
...
```

known (U, P)

$\exists$ u $\in$ Users. u.Id = U $\wedge$ u.Password = P

$\forall$ v. mightSend(v) $\Rightarrow$ $\exists$ s $\in$ Secrets.
s.Owner = U $\wedge$ v = s.Secret

Understanding SQL Usage

Prove:
$\forall$ v. mightSend(v)
$\Rightarrow$ allowed(v)

```
policy sendClient
    (SELECT Secret
     FROM Secrets, Users
     WHERE Owner = Users.Id
       AND known(Password))
```

$\forall$ s $\in$ Secrets. $\forall$ u $\in$ Users.
s.Owner = u.Id $\wedge$ known(u.Password)
$\Rightarrow$ allowed(s.Secret)

$\exists$ u $\in$ Users. u.Id = U $\wedge$ u.Password = P

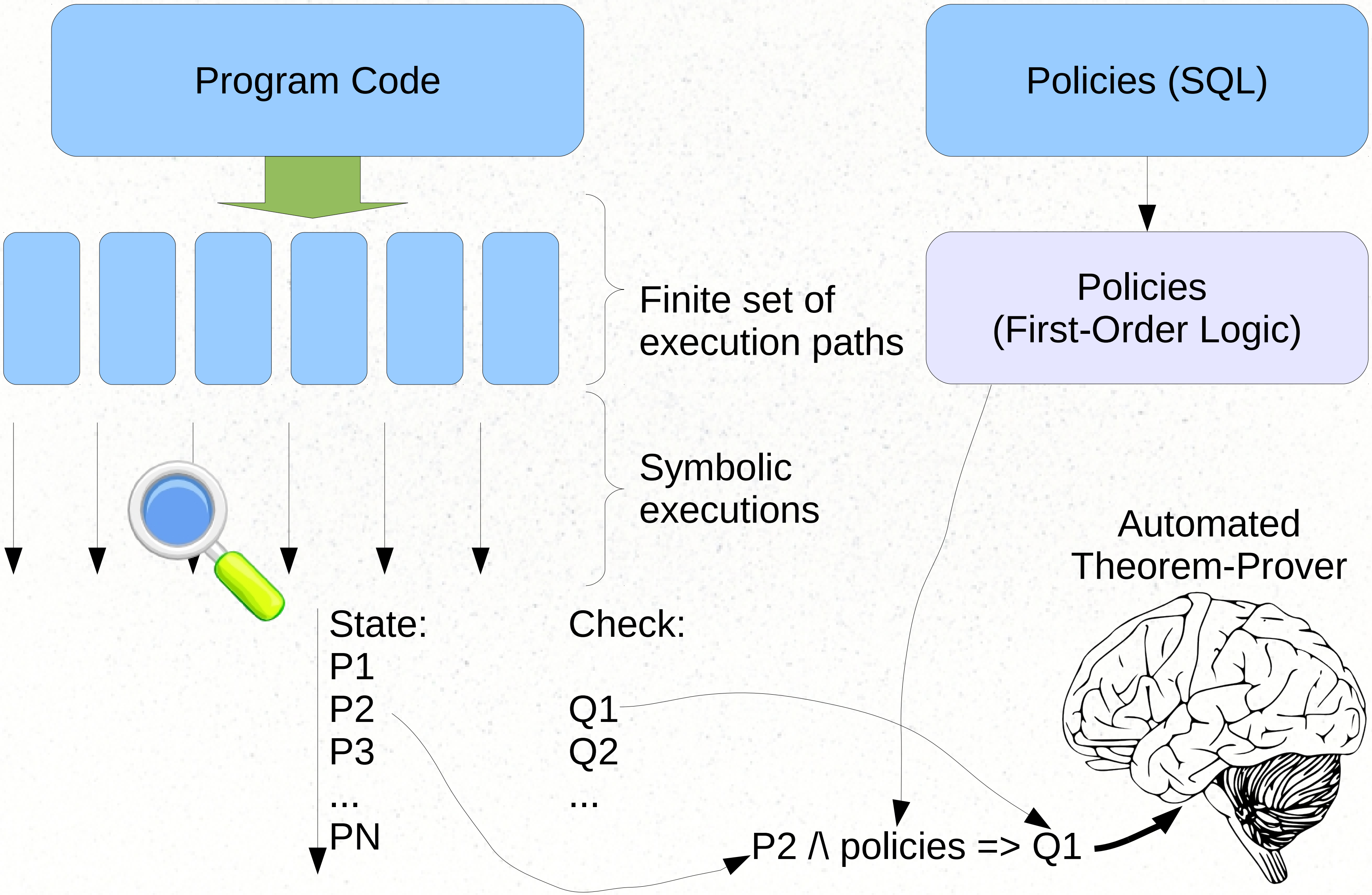
$\wedge$

$\wedge$

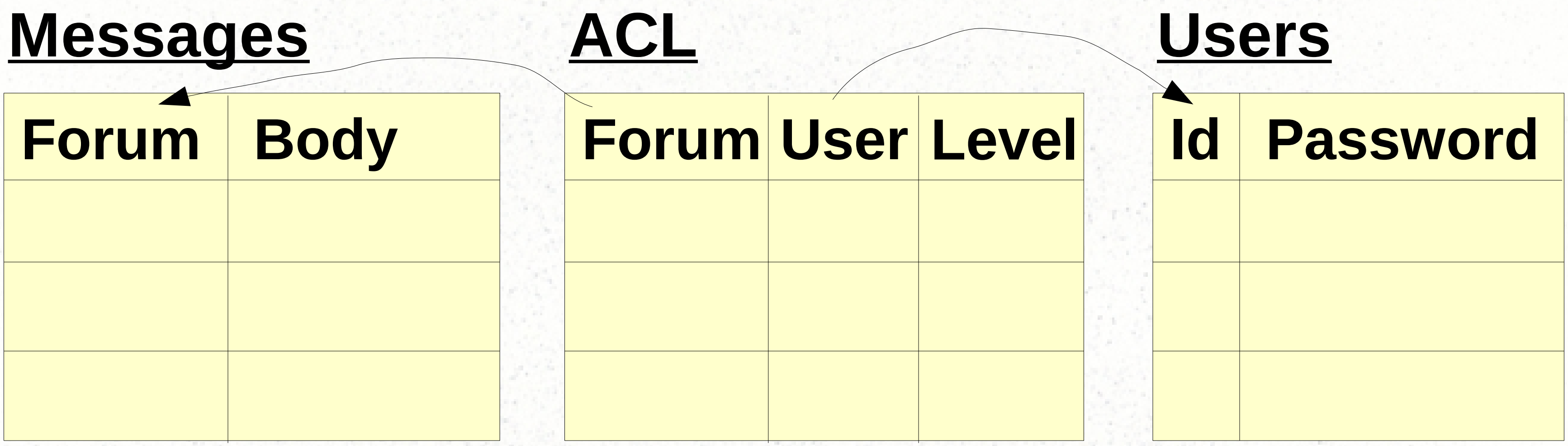
$\wedge$

$\forall$ v. mightSend(v) $\Rightarrow$ $\exists$ s $\in$ Secrets.
s.Owner = U $\wedge$ v = s.Secret

UrFlow Sketch



Fancier Policies




```
policy sendClient
     (SELECT Body
      FROM Messages, ACL, Users
      WHERE ACL.Forum = Messages.Forum
        AND ACL.User = User.Id
        AND known(Password)
        AND Level >= 42)
```

Write Policies

<u>Secrets</u>

Id	Name	Secret	Owner

<u>Users</u>

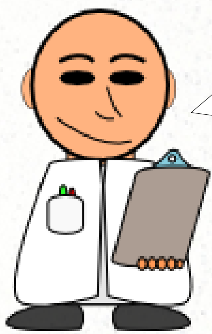
Id	Name	Password

```
policy mayInsert
    (SELECT *
     FROM Secrets AS New, Users
     WHERE New.Owner = Users.Id
       AND known(Password)
       AND known(New.Secret))
```

Case Studies

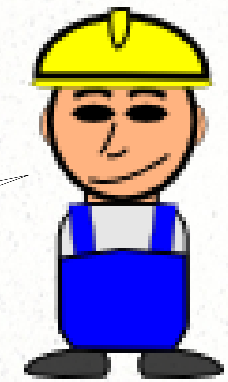
Application	Program (LoC)	Policies (LoC)	Check (sec)
Secret	138	24	0.02
Poll	196	50	0.035
User DB	84	8	-
Calendar	255	46	0.28
Forum	412	134	17.68
Gradebook	342	61	1.49

Progress



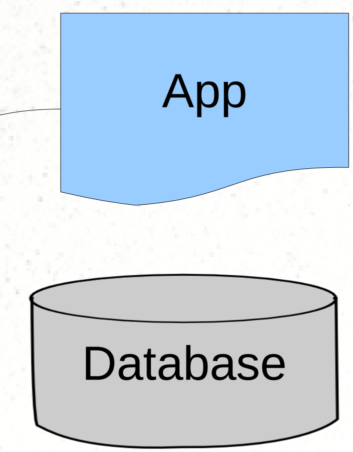
Imperative programs are too hard to analyze!
Just use declarative languages, and your life will be so much easier.

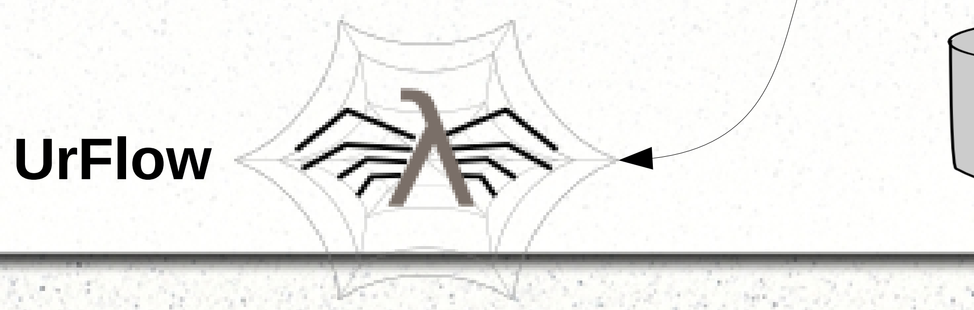
Programming Languages Researchers



Maybe later. I'm going to get back to coding my web application, which does almost nothing besides SQL queries.

Practitioners





Ur/Web Available At:

`http://www.impredicative.com/ur/`

Including online demos with syntax-highlighted source code