

# **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 regimes!  
Dependent on application-specific authentication and access control regimes!

3. Broken Authentication and Session Mgmt.

4. Insecure Direct Object References

5. Cross Site Request Forgery

6. Security Misconfiguration

7. Insecure Cryptographic Storage

....



# Authentication Snafus

## National Cyber-Alert System

Vulnerability

## National Cyber-Alert System

Original release

Vulnerability

## National Cyber-Alert System

Last revised

Original release

## Vulnerability Summary for CVE-2009-4929

Source: US

Last revised

Original release date: 07/12/2010

### Overview

Source: US

Last revised: 07/16/2010

WB News 2

### Overview

Source: US-CERT/NIST

authentication

modified W

setting this

siteadmin/ac

PageDirecto

### Overview

which allows

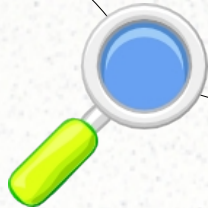
restrictions a

request.

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.

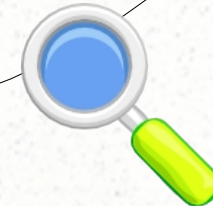
# Roads to Security

Attack vector #1



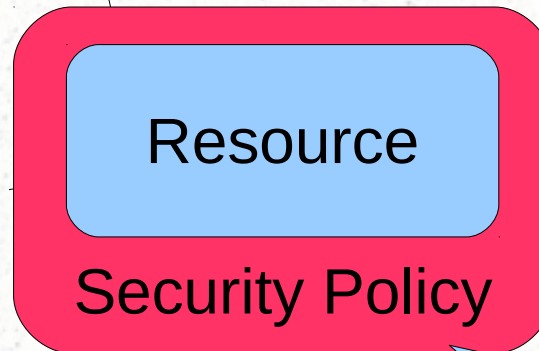
Audit

Attack vector #2

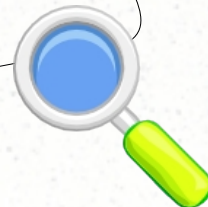


Audit

Surprise attack



Attack vector #3

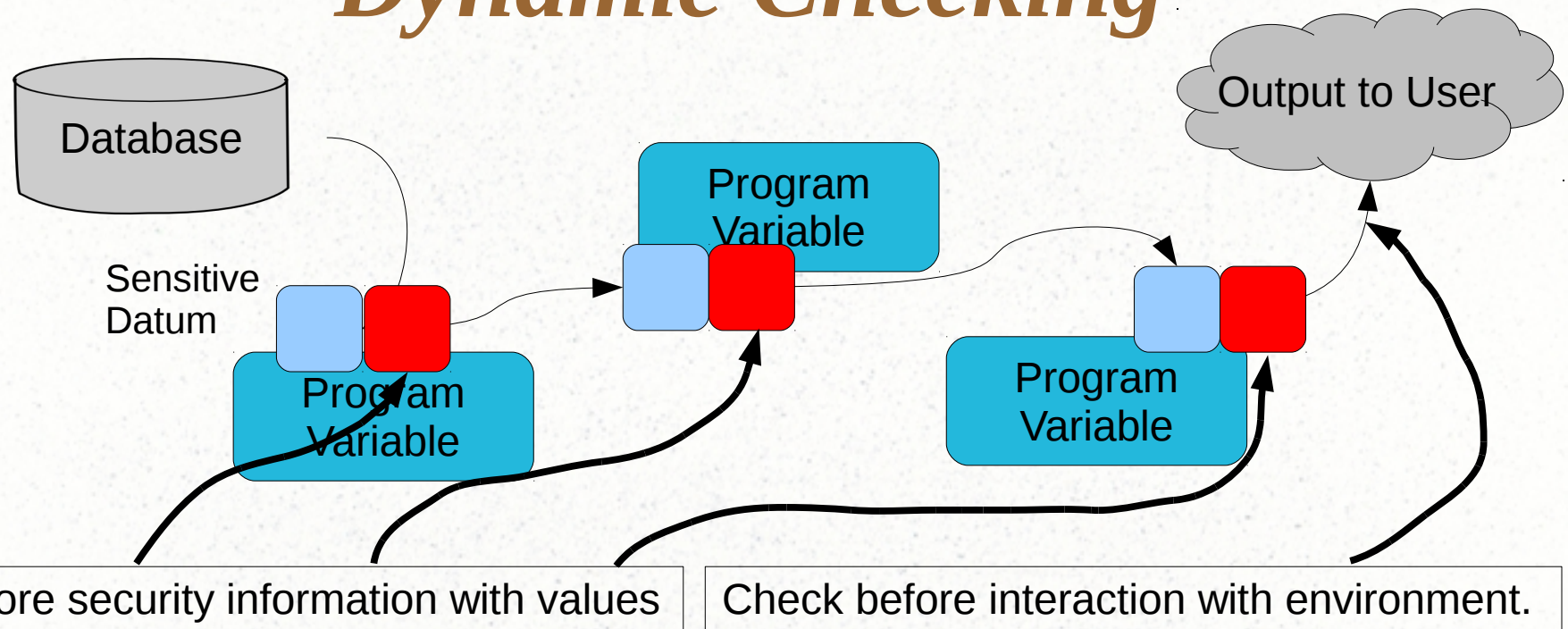


Audit

**Information flow:**  
who can learn what  
**Access control:**  
who can change what



# Dynamic Checking



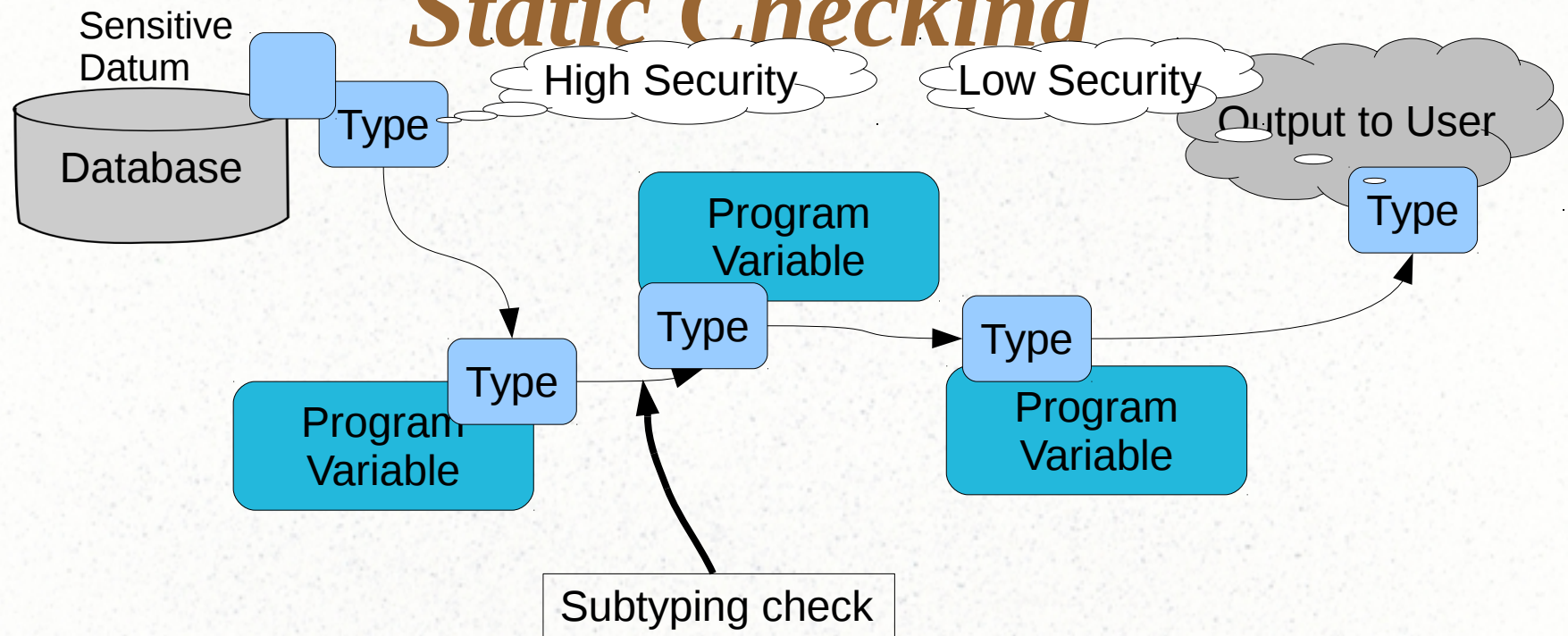
## Pros

- 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 Checkina



## 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*

## Like Dynamic Checking:

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

## The **UrFlow** analysis



for the  
**Ur/Web**  
programming  
language

## Like Security Typing:

- 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*

## Secrets

Id	Name	Secret

Client may learn anything this query could return.

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

# *Reasoning About Knowledge*

## Secrets

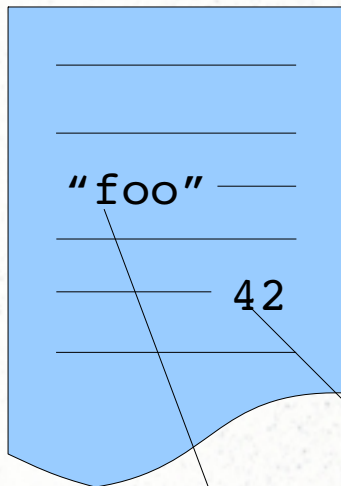
Id	Name	Secret	Code

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

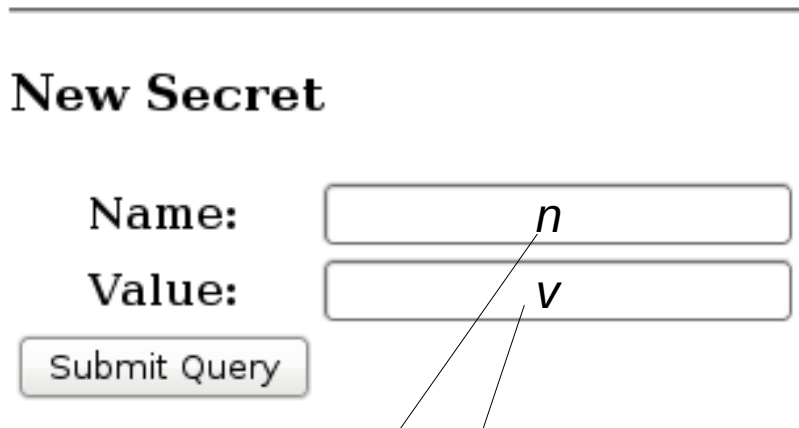


# What is “known”?

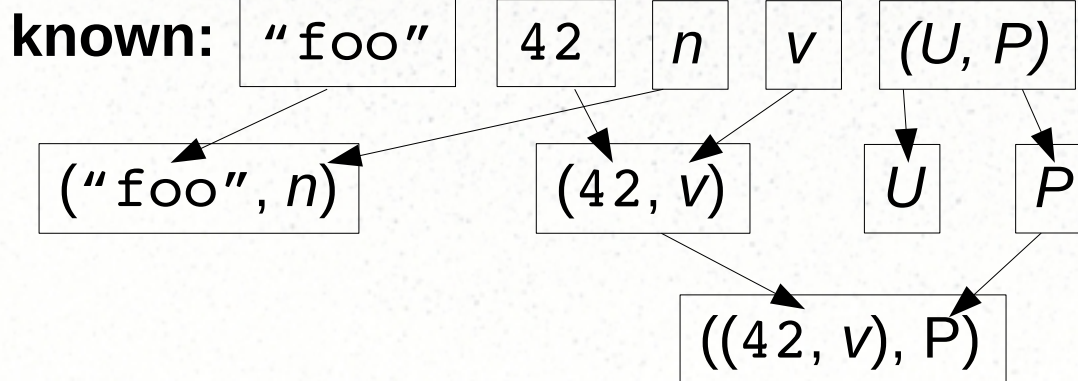
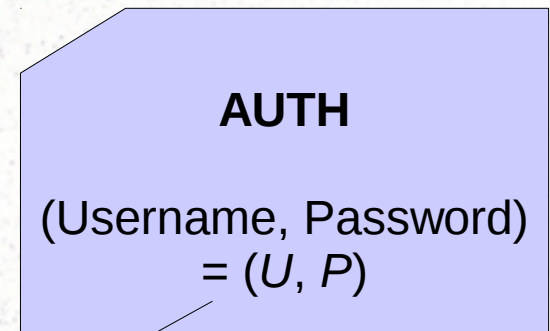
App source



Web page that generated this request



Cookies



# *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....
...   $\forall v. \text{mightSend}(v) \Rightarrow \exists s \in \text{Secrets}.
      s.\text{Owner} = U \wedge v = s.\text{Secret}$ 
```

# Understanding SQL Usage

**Prove:**

$\forall v. \text{mightSend}(v) \Rightarrow \text{allowed}(v)$

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

$\forall s \in \text{Secrets}. \forall u \in \text{Users}.$

$s.\text{Owner} = u.\text{Id} \wedge \text{known}(u.\text{Password})$

$\Rightarrow \text{allowed}(s.\text{Secret})$

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

$\wedge$

$\wedge$

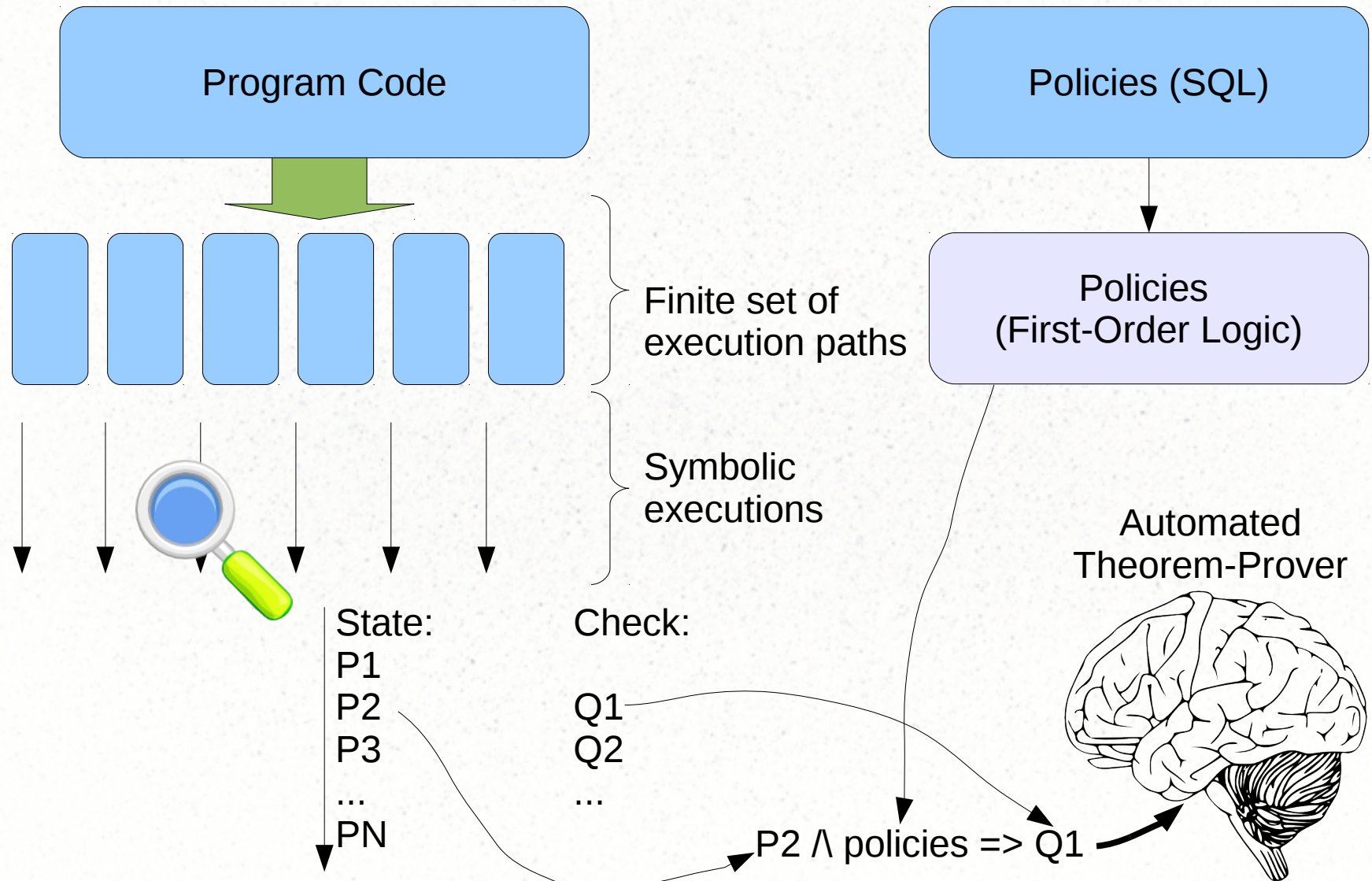
$\wedge$

$\forall v. \text{mightSend}(v) \Rightarrow \exists s \in \text{Secrets}.$

$s.\text{Owner} = U \wedge v = s.\text{Secret}$



# UrFlow Sketch



# *Fancier Policies*

## Messages

Forum	Body

## ACL

Forum	User	Level

## Users

Id	Password

```
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

## Secrets

Id	Name	Secret	Owner

## Users

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

<b>Application</b>	<b>Program (LoC)</b>	<b>Policies (LoC)</b>	<b>Check (sec)</b>
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



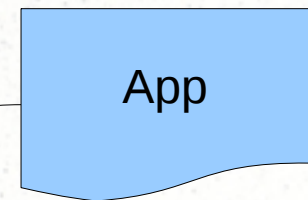
Programming  
Languages  
Researchers

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



Practitioners

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



UrFlow



## *Ur/Web Available At:*

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

Including online demos with syntax-highlighted source code