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

Adam Chlipala
OSDI 2010
Beyond Code Injection

1. Injection
2. Cross Site Scripting
3. Broken Authentication and Session Mgmt.
4. Insecure Direct Object References
5. Cross Site Request Forgery
6. Security Misconfiguration
7. Insecure Cryptographic Storage

An application includes an unintended run-time program interpreter.
Authentication Snafus

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

The siteadmin/admin.php and PageDirector scripts in TotalCalendar 2.4 allow the admin/manager_users.php script to set passwords for site administrator accounts without requiring administrative authentication. This vulnerability allows remote attackers to change arbitrary passwords via the newPW1 and newPW2 parameters.
Surprise attack

Audit

Security Policy

Information flow:
who can learn what

Access control:
who can change what
Dynamic Checking

Store security information with values
Check before interaction with environment.

**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

Like Dynamic Checking:
- No program annotations required
- Flexible and programmer-accessible policy language (SQL)

Like Security Typing:
- Checks all program paths statically
- No run-time overhead

The UrFlow analysis for the Ur/Web programming language
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>);
Simple Policies

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Client may learn anything this query could return.

policy sendClient
  (SELECT Id, Name
   FROM Secrets)
Reasoning About Knowledge

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Secret</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

policy sendClient
(SELECT *
  FROM Secrets
  WHERE known(Code))
What is “known”?

known: “foo” 42 n v (U, P) ((42, v), P)

AUTH
(Username, Password) = (U, P)

New Secret
Name: n
Value: v

Cookies

App source

Web page that generated this request
**Multi-Table Policies**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Secret</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

policy sendClient

(SELECT Secret
  FROM Secrets, Users
  WHERE Owner = Users.Id
  AND known(Password))
Understanding SQL Usage

Program Execution

\[ (U, P) = \text{readCookie}(	ext{AUTH}); \]

\[
\text{pass} = \text{SELECT Password FROM Users WHERE Id = U};
\]

if (pass != P) abort();

\[
\exists u \in \text{Users. } u.\text{Id} = U \land u.\text{Password} = P
\]

\[
\text{rows} = \text{SELECT Secret FROM Secrets WHERE Owner = U};
\]

// Send rows to client....

\[
\forall v. \text{mightSend}(v) \Rightarrow \exists s \in \text{Secrets. } s.\text{Owner} = U \land v = s.\text{Secret}
\]
**Understanding SQL Usage**

**Prove:**
\[ \forall v. \text{mightSend}(v) \implies \text{allowed}(v) \]

\[ \forall s \in \text{Secrets}. \forall u \in \text{Users}. \]
\[ s.\text{Owner} = u.\text{Id} \land \text{known}(u.\text{Password}) \]
\[ \implies \text{allowed}(s.\text{Secret}) \]
\[ \exists u \in \text{Users}. u.\text{Id} = U \land u.\text{Password} = P \]

\[ \land \]
\[ \land \]
\[ \land \]

\[ \forall v. \text{mightSend}(v) \implies \exists s \in \text{Secrets}. \]
\[ s.\text{Owner} = U \land v = s.\text{Secret} \]
UrFlow Sketch

Program Code

Policies (SQL)

Finite set of execution paths

Symbolic executions

State:
P1
P2
P3
...
PN

Check:
Q1
Q2
...

Automated Theorem-Prover

P2 \land \text{policies} \implies Q1
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

<table>
<thead>
<tr>
<th>Secrets</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
<td>Secret</td>
<td>Owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Users</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
<td>Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

policy mayInsert

(SELECT *
FROM Secrets AS New, Users
WHERE New.Owner = Users.Id
AND known(Password)
AND known(New.Secret))
## Case Studies

<table>
<thead>
<tr>
<th>Application</th>
<th>Program (LoC)</th>
<th>Policies (LoC)</th>
<th>Check (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret</td>
<td>138</td>
<td>24</td>
<td>0.02</td>
</tr>
<tr>
<td>Poll</td>
<td>196</td>
<td>50</td>
<td>0.035</td>
</tr>
<tr>
<td>User DB</td>
<td>84</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>Calendar</td>
<td>255</td>
<td>46</td>
<td>0.28</td>
</tr>
<tr>
<td>Forum</td>
<td>412</td>
<td>134</td>
<td>17.68</td>
</tr>
<tr>
<td>Gradebook</td>
<td>342</td>
<td>61</td>
<td>1.49</td>
</tr>
</tbody>
</table>
Imperative programs are too hard to analyze! Just use declarative languages, and your life will be so much easier.

Maybe later. I’m going to get back to coding my web application, which does almost nothing besides SQL queries.
Ur/Web Available At:

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

Including online demos with syntax-highlighted source code