Improving security using data flow assertions

Alex Yip, Xi Wang, Nickolai Zeldovich, Frans Kaashoek
MIT CSAIL
Many security vulnerabilities caused by programming errors

<table>
<thead>
<tr>
<th>Attack vector</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL injection</td>
<td>20.4%</td>
</tr>
<tr>
<td>Cross-site scripting</td>
<td>14.0%</td>
</tr>
<tr>
<td>Buffer overflow</td>
<td>9.5%</td>
</tr>
<tr>
<td>Directory traversal</td>
<td>6.6%</td>
</tr>
<tr>
<td>Script eval injection</td>
<td>5.0%</td>
</tr>
<tr>
<td>Missing access checks</td>
<td>4.6%</td>
</tr>
<tr>
<td>( … long tail of others … )</td>
<td>39.8%</td>
</tr>
</tbody>
</table>

Top 6 classes of security vulnerabilities found in 2008 [CVE]
Many security vulnerabilities caused by programming errors

- **SQL injection**: attacker's input used in SQL query
- **XSS**: attacker's input used in HTML page
- **Directory traversal**: attacker-supplied path has “..”
- **Script injection**: attacker's input executed as code
- **Missing ACL**: sensitive data sent without check
Common programming error: missing checks
Common programming error: missing checks

Application
SQL injection attack

- Attacker's browser
- Stored query
- Application
- SQL database

• Goal: quote user input before using in SQL
SQL injection attack

Goal: quote user input before using in SQL
Missing access control check

- Goal: check ACL when sending file to user
Missing access control check

- Goal: check ACL when sending file to user
Cross-site scripting attack

- Goal: remove Javascript from user input before using in HTML
Cross-site scripting attack

- Goal: remove Javascript from user input before using in HTML
Challenge: knowing where to check

- Today: invoke check on all paths from source to sink
  - Easy to miss one (out of 572 in phpBB, a popular web app)
- Security check cannot be made based on data alone
  - At the source, don't know where data is going yet
  - At the sink, don't know where data came from
Approach: Associate checks with data

- Assume trusted runtime & non-malicious app code
- Programmers tag data with *assertions* at source
- Track assertions when data is copied or moved
- Assertions checked at the sinks
Example bug: HotCRP password disclosure

Email
nickolai@csail.mit.edu

Password

Sign me in
- I forgot my password, email it to me
- I'm a new user and want to create an account using this email address

Sign in
Example bug: HotCRP password disclosure
Example bug: HotCRP password disclosure

From: tom@cs.washington.edu
To: nickolai@csail.mit.edu

Dear Nickolai Zeldovich,

Here is your account information:

Email: nickolai@csail.mit.edu
Password: cluprerast
Example bug: HotCRP password disclosure

- Helpful feature: email preview mode
- Display emails instead of sending them
- Useful to fine-tune messages sent to everyone
From: tom@cs.washington.edu
To: nickolai@csail.mit.edu

Dear Nickolai Zeldovich,

Here is your account information:

Email: nickolai@csail.mit.edu
Password: cluprerast

Email
nickolai@csail.mit.edu

Password

- [x] Sign me in
- [ ] I forgot my password, email it to me
- [ ] I'm a new user and want to create an account using this email address

Sign in
From: tom@cs.washington.edu
To: tom@cs.washington.edu

Dear Tom Anderson,

Here is your account information:

   Email: tom@cs.washington.edu
   Password: phyts6phatr
Programmer has a security plan

- Programmers often have a data flow plan in mind
  - Sanitize HTML; only send password to user's email
  - Hard: plan must be enforced *everywhere*
Programmer has a security plan

- Programmers often have a data flow plan in mind
  - Sanitize HTML; only send password to user's email
  - Hard: plan must be enforced everywhere

- Challenge: many flow paths, easy to miss one
  - phpBB: 572 calls to check for cross-site scripting
Programmer has a security plan

- Programmers often have a data flow plan in mind
  - Sanitize HTML; only send password to user's email
  - Hard: plan must be enforced everywhere

- Challenge: many flow paths, easy to miss one
  - phpBB: 572 calls to check for cross-site scripting

- Challenge: 3rd-party developers don't know plan
  - phpBB: 879 plug-ins written by 505 programmers
Our approach: Allow programmers to make security plan explicit

- *Resin*: modified language runtime (Python, PHP)
  - Programmer specifies explicit *data flow assertions*
  - Runtime checks assertion on every source→sink path
  - Assertion prevents attacker from exploiting missing check

- Not a bug-finding tool; prevents exploits at runtime
Challenges and ideas

- Plan: “only send this password to nickolai@mit.edu”

- How would we check if a program obeys this plan?

- How would the programmer express this assertion?
Challenges and ideas

- Plan: “only send this password to nickolai@mit.edu”

- How would we check if a program obeys this plan?
  - Associate the assertions with data (e.g. password)
  - Track assertions along with data in language runtime
  - Check at programmer-defined boundaries
    - E.g. external I/O (file, network), when data leaves our control

- How would the programmer express this assertion?
Challenges and ideas

- Plan: “only send this password to nickolai@mit.edu”

- How would we check if a program obeys this plan?
  - Associate the assertions with data (e.g. password)
  - Track assertions along with data in language runtime
  - Check at programmer-defined boundaries
    - E.g. external I/O (file, network), when data leaves our control

- How would the programmer express this assertion?
  - Express using code – simple, general-purpose
  - Programmers can reuse code, data structures
Example: Preventing HotCRP's bug in Resin

Pipe to sendmail for nickolai@mit.edu

HTTP conn to browser

SQL database

World-readable log file

Resin Language Runtime

“myPassw0rd”
Programmer attaches a policy object to a string

Pipe to sendmail for nickolai@mit.edu

HTTP conn to browser

SQL database

World-readable log file

Resin Language Runtime

“myPassw0rd”

Policy: Only email to nickolai@mit.edu
Programmer attaches filter objects to security boundaries

Pipe to sendmail for nickolai@mit.edu

HTTP conn to browser

SQL database

World-readable log file

Resin Language Runtime

"myPassw0rd"

Policy:
Only email to nickolai@mit.edu
Runtime propagates policies for strings

Dear Nickolai Zeldovich,
Here is your account info
Email: nickolai@mit.edu
Password: myPassw0rd
Runtime propagates policies for strings

Policy:
Only email to nickolai@mit.edu

Dear Nickolai Zeldovich,
Here is your account info
Email: nickolai@mit.edu
Password: myPassw0rd

Policy:
Only email to nickolai@mit.edu

Pipe to sendmail for nickolai@mit.edu
HTTP conn to browser
SQL database
World-readable log file
Filters check assertions by invoking policy objects

Pipe to sendmail for nickolai@mit.edu

HTTP conn to browser

SQL database

World-readable log file

Resin Language Runtime

Policy: Only email to nickolai@mit.edu

“myPassw0rd”

Policy: Only email to nickolai@mit.edu

Dear Nickolai Zeldovich,
Here is your account info
Email: nickolai@mit.edu
Password: myPassw0rd

Policy: Only email to nickolai@mit.edu
Assertions avoid the need to understand all code

Pipe to sendmail for nickolai@mit.edu

HTTP conn to browser

SQL database

World-readable log file

Resin Language Runtime

Filter

Third-party email module

“myPassw0rd”

Policy:
Only email to nickolai@mit.edu

Dear Nickolai Zeldovich,
Here is your account info
Email: nickolai@mit.edu
Password: myPassw0rd

Policy:
Only email to nickolai@mit.edu
class PasswordPolicy extends Policy {

function __construct($username) {
    $this->user = $username;
}

function export_check($context) {
    if ($context['type'] == "mail" && $context['rcpt'] == $this->user)
        return;
    if ($Me->valid() && $Me->privChair)
        return;
    throw new Exception ("unauthorized disclosure");
}
}
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }
}

Stores owner's username (email address in HotCRP)
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }

    function export_check($context) {
        if ($context['type'] == "mail" && $context['rcpt'] == $this->user)
            return;
        if ($Me->valid() && $Me->privChair)
            return;
        throw new Exception("unauthorized disclosure");
    }
}
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }

    function export_check($context) {
        if ($context['type'] == "mail" &&
            $context['rcpt'] == $this->user)
            return;

        if ($Me->valid() && $Me->privChair)
            return;
        throw new Exception("unauthorized disclosure");
    }
}
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }

    function export_check($context) {
        if ($context['type'] == "mail" &&
            $context['rcpt'] == $this->user)
            return;
        if ($Me->valid() && $Me->privChair)
            return;
    }
}
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }

    function export_check($context) {
        if ($context['type'] == "mail" &&
            $context['rcpt'] == $this->user)
            return;
        if ($Me->valid() && $Me->privChair)
            return;
        throw new Exception ("unauthorized disclosure");
    }
}
class PasswordPolicy extends Policy {
    private $user;

    function __construct($username) {
        $this->user = $username;
    }

    function export_check($context) {
        if ($context['type'] == "mail" &&
            $context['rcpt'] == $this->user)
            return;
        if ($Me->valid() && $Me->privChair)
            return;
        throw new Exception ("unauthorized disclosure");
    }
}

policy_set($new_password, new PasswordPolicy($username));
Filters help track persistent data

Resin Language Runtime

/home/hotcrp/.htpasswd

File Filter

“myPassw0rd”

Policy:
Only email to nickolai@mit.edu
Filters help track persistent data

- File filter serializes/de-serializes policies to xattr

```
/home/hotcrp/.htpasswd
```

<table>
<thead>
<tr>
<th>Ext. attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-resin-policy</td>
<td>![xattr Icon]</td>
</tr>
</tbody>
</table>

```
"myPassw0rd"
```

**Policy:**
Only email to
nickolai@mit.edu

Resin Language Runtime
Filters help track persistent data

- Other apps (e.g. Apache) can check data policies to prevent attacker from obtaining sensitive data

```
/home/hotcrp/.htpasswd
```

```
myPassw0rd
```

```
Policy:
Only email to nickolai@mit.edu
```

```
x-resin-policy
```

<table>
<thead>
<tr>
<th>Ext. attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-resin-policy</td>
<td></td>
</tr>
</tbody>
</table>
Tracking multiple policies

- Set of policies for every primitive data element
  - Character in a string, integer, etc

- Policies propagated on explicit data flows
  - \( a = \text{concat}(b, c) \) propagates
  - \( a = \text{array}[b] \) does not propagate

- Runtime merges policies when data is combined
  - Common: merge strings: automatic (byte-level tracking)
  - Rare: merge integers: defined in policy object (e.g. union)
Two prototypes

- PHP: 5,944 lines of code added/changed
  - Complex due to poorly-engineered PHP code base

- Python: 681 lines of code added/changed
  - Python interpreter is better-engineered
  - No byte-level tracking or persistent policies in SQL DB
  - Mostly proof-of-concept: Resin isn't PHP-specific
Evaluation questions

• Resin's goal:
  programmers uphold security plan by writing explicit data flow assertions

• How hard is it to write an assertion?
• What attacks can assertions prevent?
• Do you need to know the attack to write asserts?
Experiment 1

- Took 5 applications with known security bugs
- Wrote assertions to prevent exploitation
## Experiment 1 results

<table>
<thead>
<tr>
<th>Application</th>
<th>Application LOC</th>
<th>Assert LOC</th>
<th>Vulnerability addressed (# found)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoinMoin Wiki</td>
<td>89,600</td>
<td>8</td>
<td>Missing access check (2)</td>
</tr>
<tr>
<td>HotCRP</td>
<td>29,000</td>
<td>23</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td>MyPhpScripts login</td>
<td>425</td>
<td>6</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td><em>many PHP apps</em></td>
<td>–</td>
<td>12</td>
<td>PHP script injection (5+)</td>
</tr>
<tr>
<td>phpBB</td>
<td>172,000</td>
<td>22</td>
<td>Cross-site scripting (4)</td>
</tr>
</tbody>
</table>
## Assertions are easy to write

<table>
<thead>
<tr>
<th>Application</th>
<th>Application LOC</th>
<th>Assert LOC</th>
<th>Vulnerability addressed (# found)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoinMoin Wiki</td>
<td>89,600</td>
<td>8</td>
<td>Missing access check (2)</td>
</tr>
<tr>
<td>HotCRP</td>
<td>29,000</td>
<td>23</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td>MyPhpScripts login</td>
<td>425</td>
<td>6</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td><em>many PHP apps</em></td>
<td>–</td>
<td>12</td>
<td>PHP script injection (5+)</td>
</tr>
<tr>
<td>phpBB</td>
<td>172,000</td>
<td>22</td>
<td>Cross-site scripting (4)</td>
</tr>
</tbody>
</table>
## Assertions prevent a range of bugs

<table>
<thead>
<tr>
<th>Application</th>
<th>Application LOC</th>
<th>Assert LOC</th>
<th>Vulnerability addressed (# found)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoinMoin Wiki</td>
<td>89,600</td>
<td>8</td>
<td>Missing access check (2)</td>
</tr>
<tr>
<td>HotCRP</td>
<td>29,000</td>
<td>23</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td>MyPhpScripts login</td>
<td>425</td>
<td>6</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td><em>many PHP apps</em></td>
<td>–</td>
<td>12</td>
<td>PHP script injection (5+)</td>
</tr>
<tr>
<td>phpBB</td>
<td>172,000</td>
<td>22</td>
<td>Cross-site scripting (4)</td>
</tr>
</tbody>
</table>
### Assertions are not specific to attack vectors

<table>
<thead>
<tr>
<th>Application</th>
<th>Application LOC</th>
<th>Assert LOC</th>
<th>Vulnerability addressed (# found)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoinMoin Wiki</td>
<td>89,600</td>
<td>8</td>
<td>Missing access check (2)</td>
</tr>
<tr>
<td>HotCRP</td>
<td>29,000</td>
<td>23</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td>MyPhpScripts login</td>
<td>425</td>
<td>6</td>
<td>Password disclosure (1)</td>
</tr>
<tr>
<td><em>many PHP apps</em></td>
<td>–</td>
<td>12</td>
<td>PHP script injection (5+)</td>
</tr>
<tr>
<td>phpBB</td>
<td>172,000</td>
<td>22</td>
<td>Cross-site scripting (4)</td>
</tr>
</tbody>
</table>

- HotCRP had a logic error (email preview mode)
- MyPhpScripts password file was web-accessible
- One assertion prevents many pwd disclosure flows
Experiment 2

- Experiment 1 focused on known bugs  
  - Resin used to avoid regressions

- More dangerous: attackers find, exploit new bugs

- Want to show *Resin* can prevent unknown bugs  
  - Wrote high-level asserts for 5 apps; not attack-specific  
  - Manually looked for unknown bugs to trigger assertion
Experiment 2 results: Assertions prevent unknown bugs

<table>
<thead>
<tr>
<th>Application</th>
<th>Application LOC</th>
<th>Assert LOC</th>
<th>Vulnerability addressed (# found)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HotCRP</td>
<td>29,000</td>
<td>30 32</td>
<td>Access check papers (0) Access check authors (0)</td>
</tr>
<tr>
<td>phpBB</td>
<td>172,000</td>
<td>23</td>
<td>Missing read access check (4)</td>
</tr>
<tr>
<td>FileThingie</td>
<td>3,200</td>
<td>19</td>
<td>Directory traversal (1)</td>
</tr>
<tr>
<td>PHP Navigator</td>
<td>4,100</td>
<td>17</td>
<td>Directory traversal (1)</td>
</tr>
<tr>
<td>EECS Grad Admission</td>
<td>18,500</td>
<td>9</td>
<td>SQL injection (3)</td>
</tr>
</tbody>
</table>

- Without assertions, attacker could have compromised at least 4 of the 5 apps
Performance evaluation

- Focus on application performance: HotCRP
  - 3 assertions: passwords, papers, authors
  - Workload: 30 min prior to SOSP '07 deadline

- Result: 30% CPU overhead
- Resin would increase CPU use from 14% to 19%
Future work

- Report errors earlier with static analysis
- Assertions across runtimes and machines
- Strong enforcement for untrusted code
Related work

- Perl taint & vuln-specific tools (XSS, SQL inj.)
- Information flow control (Jif, HiStar)
- Language security checks (AspectJ, Fable, PQL)
Summary

- Attackers exploit missing security checks

- Hard for programmers to check every flow

- *Resin* allows attaching security assertions to data
  - Checked for any possible data flow at runtime

- Data flow assertions prevent wide range of bugs