Improving security using data flow assertions

Alex Yip, Xi Wang, *Nickolai Zeldovich*, Frans Kaashoek
MIT CSAIL

Many security vulnerabilities caused by programming errors

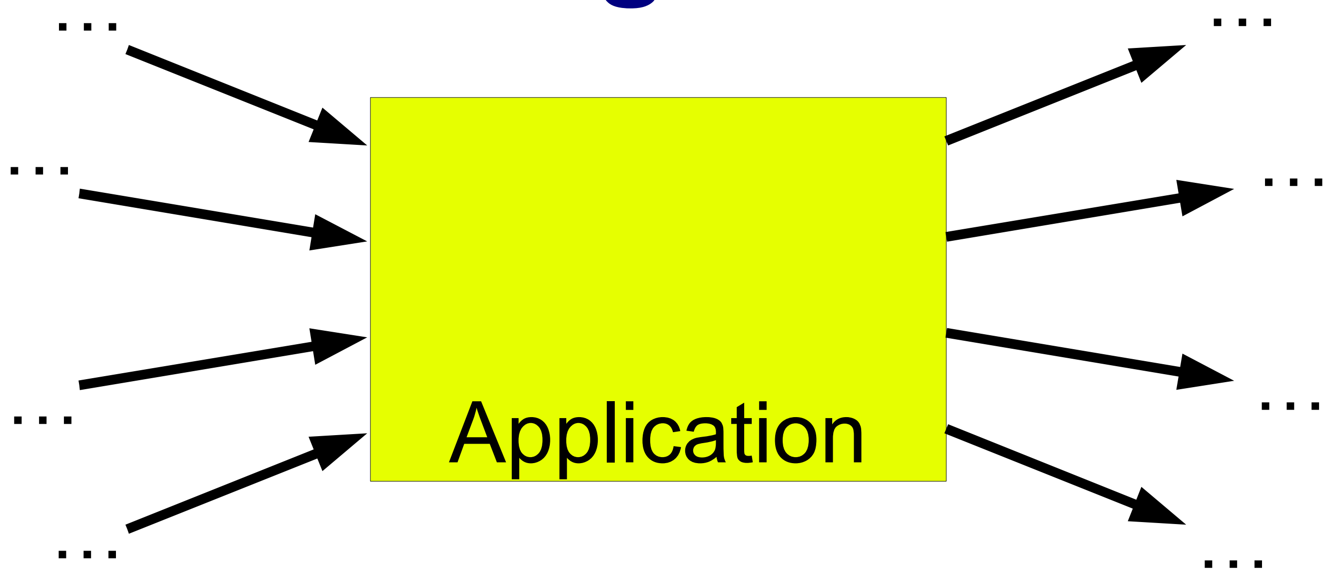
Attack vector	Percentage
SQL injection	20.4%
Cross-site scripting	14.0%
Buffer overflow	9.5%
Directory traversal	6.6%
Script eval injection	5.0%
Missing access checks	4.6%
(… long tail of others …)	39.8%

Top 6 classes of security vulnerabilities found in 2008 [CVE]

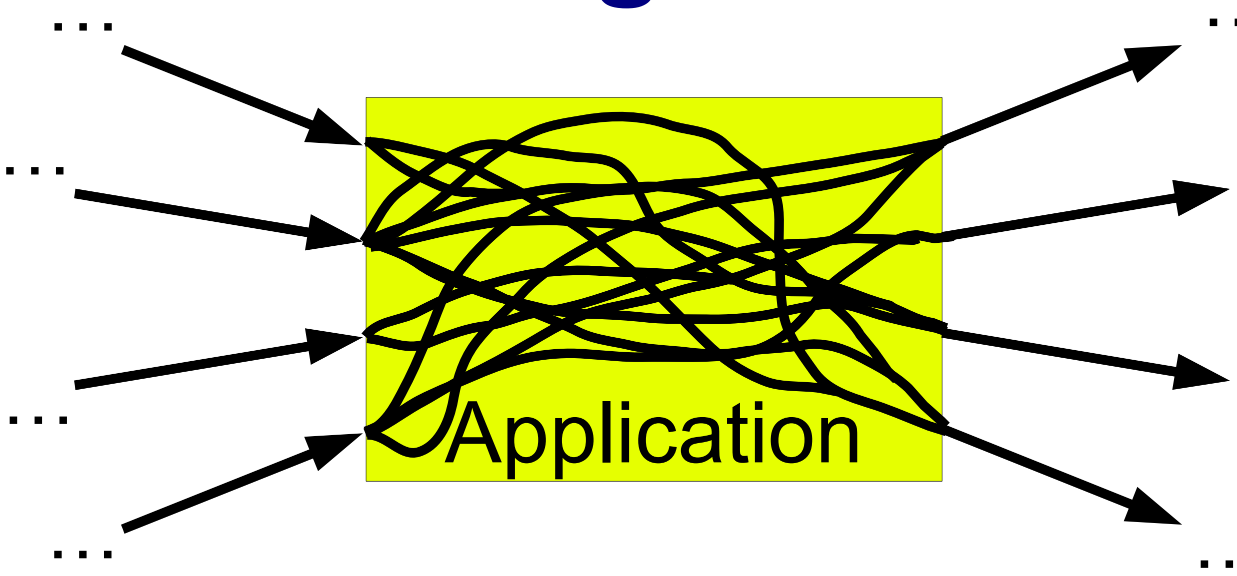
Many security vulnerabilities caused by programming errors

- ***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

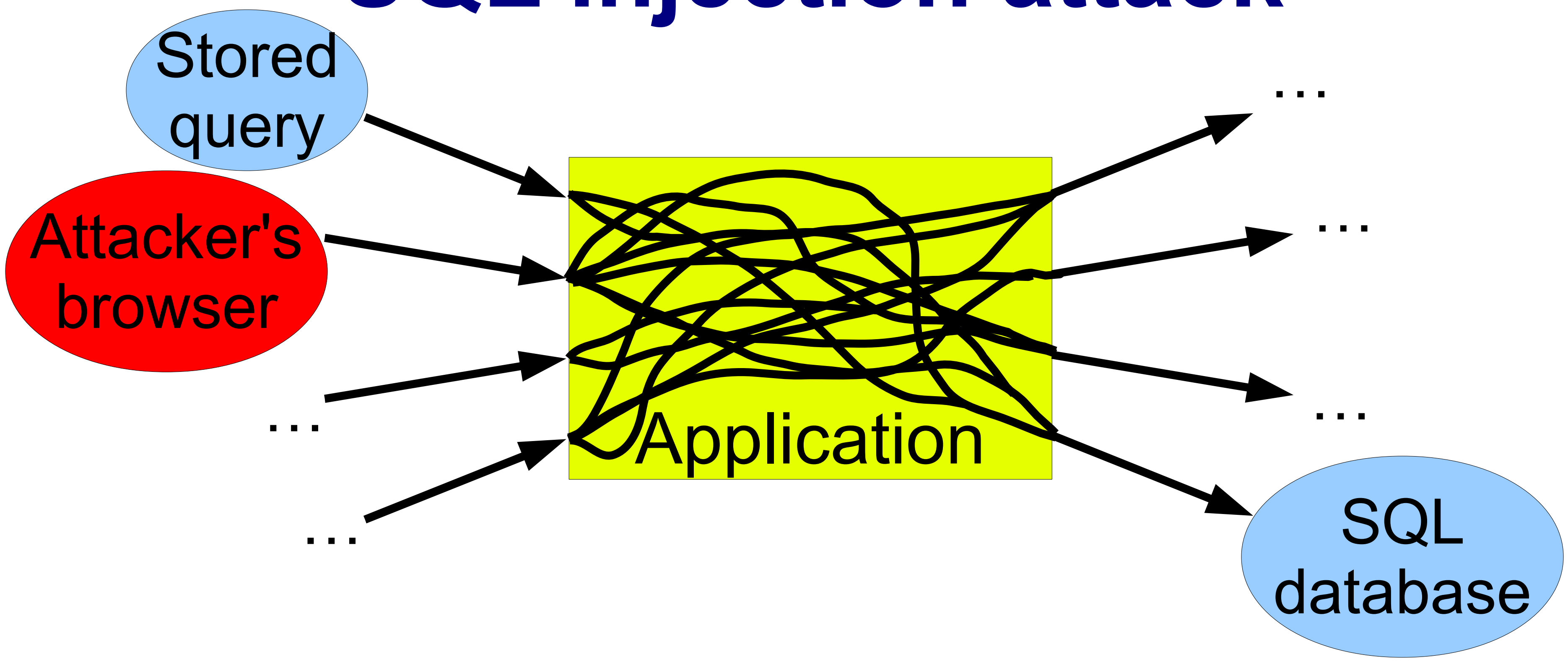
Common programming error: missing checks



Common programming error: missing checks

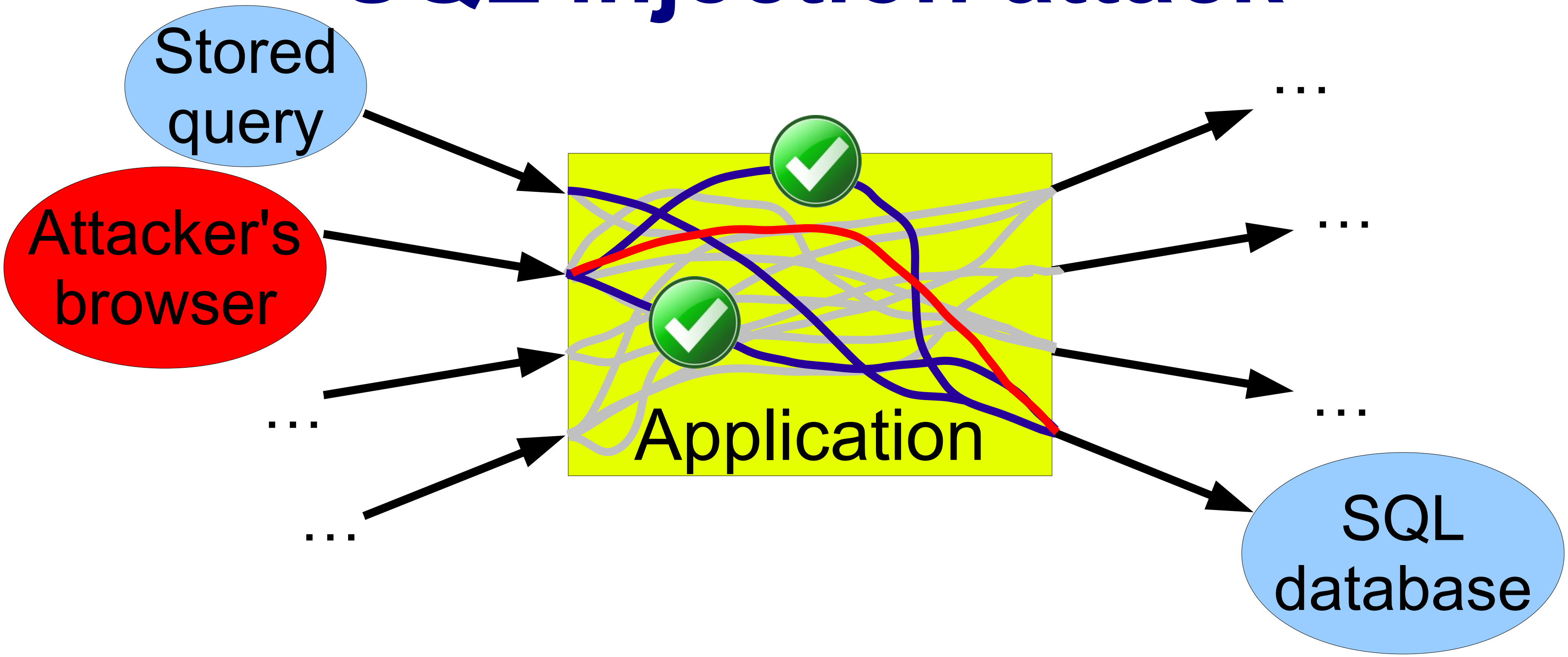


SQL injection attack



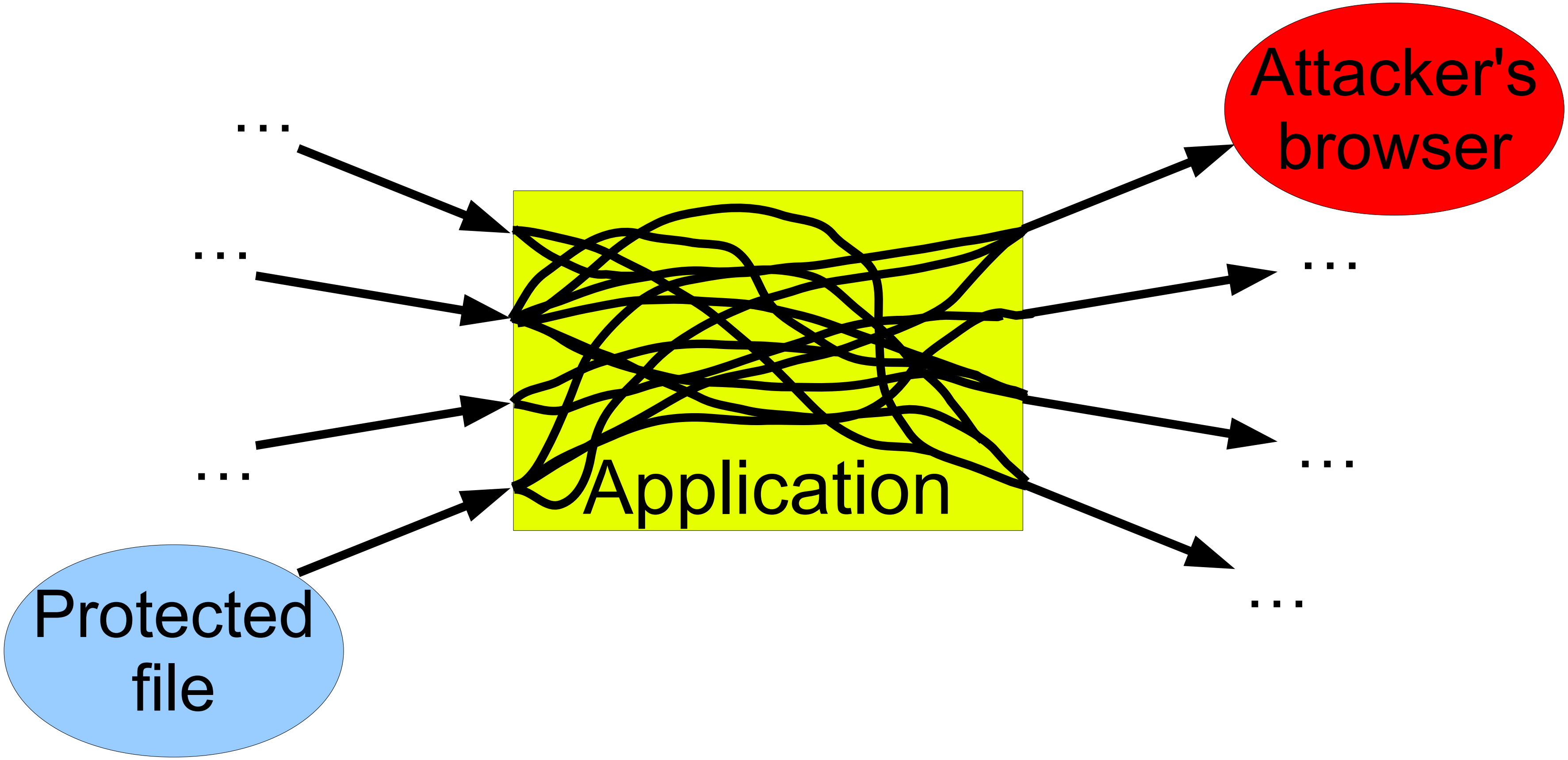
- Goal: quote user input before using in SQL

SQL injection attack



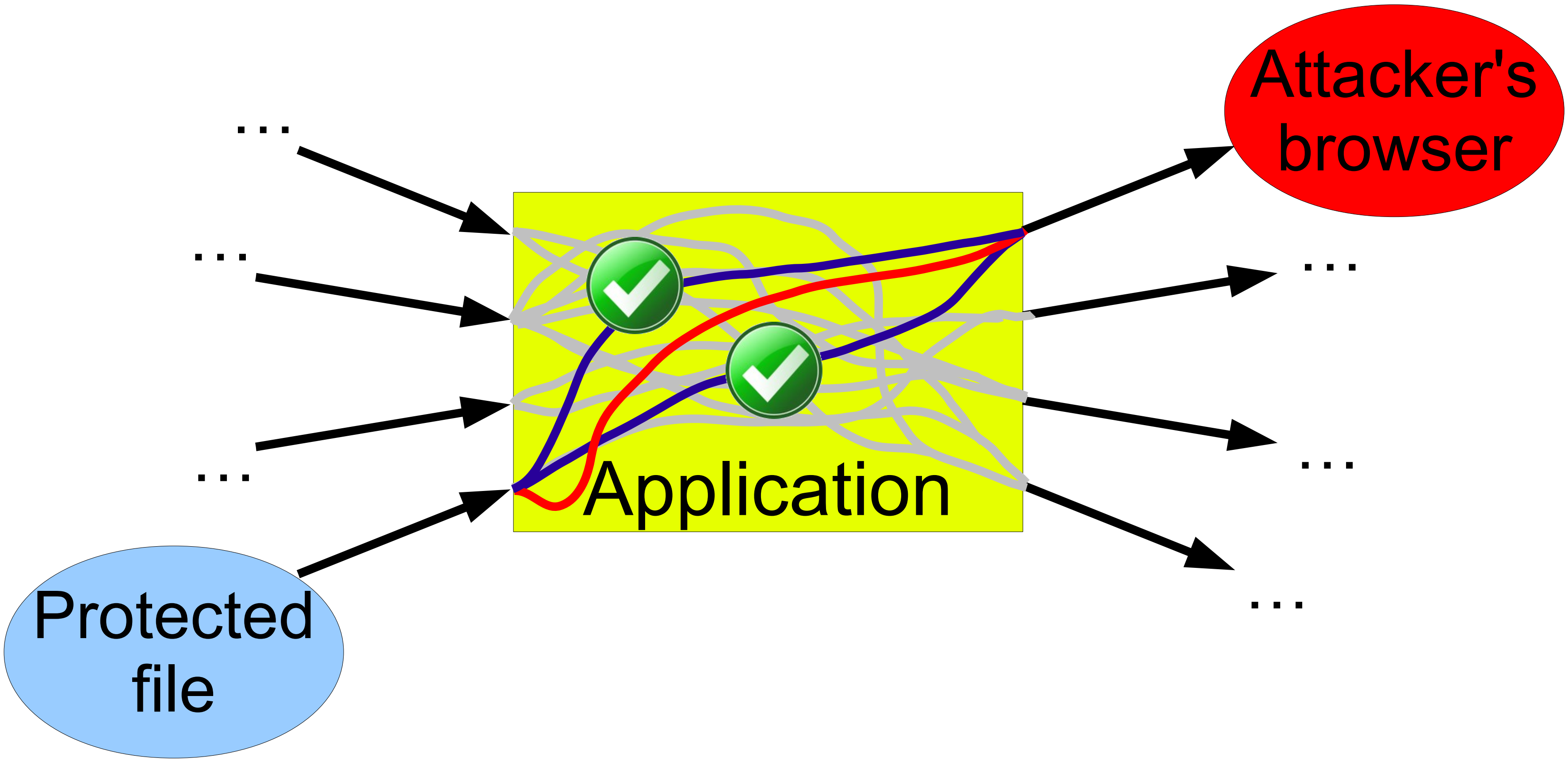
- Goal: quote user input before using in SQL

Missing access control check



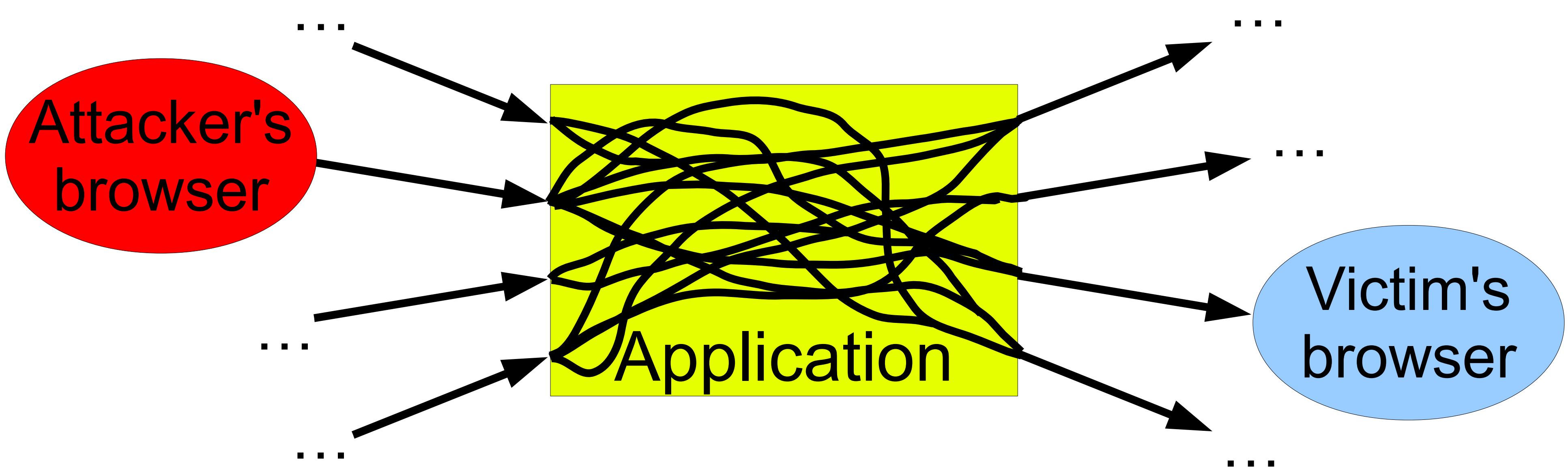
- Goal: check ACL when sending file to user

Missing access control check



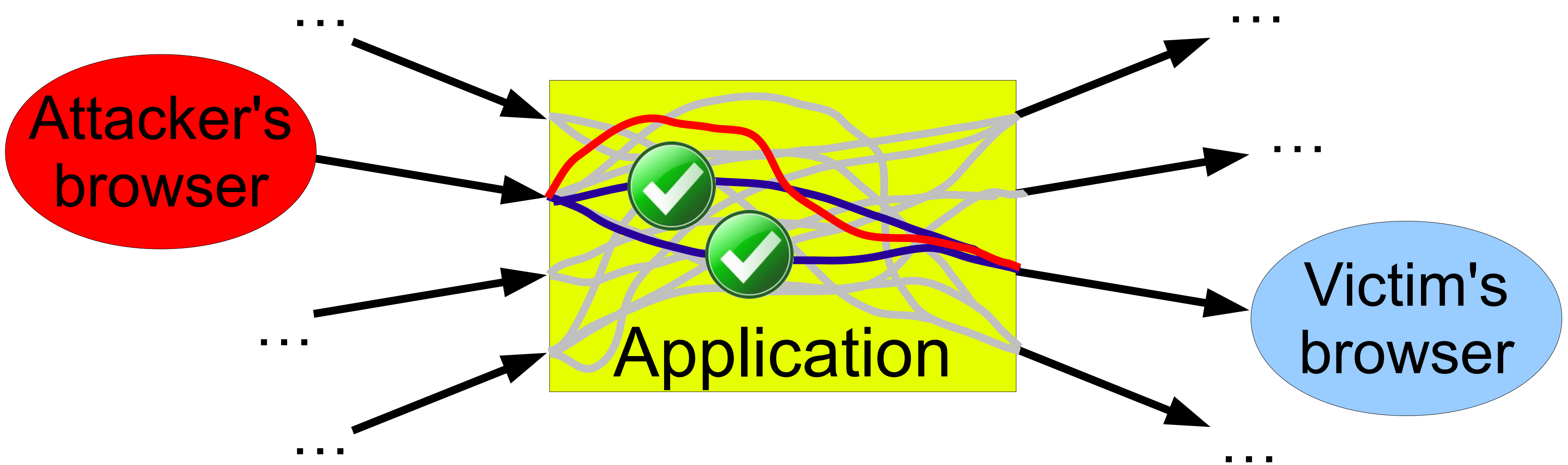
- Goal: check ACL when sending file to user

Cross-site scripting attack



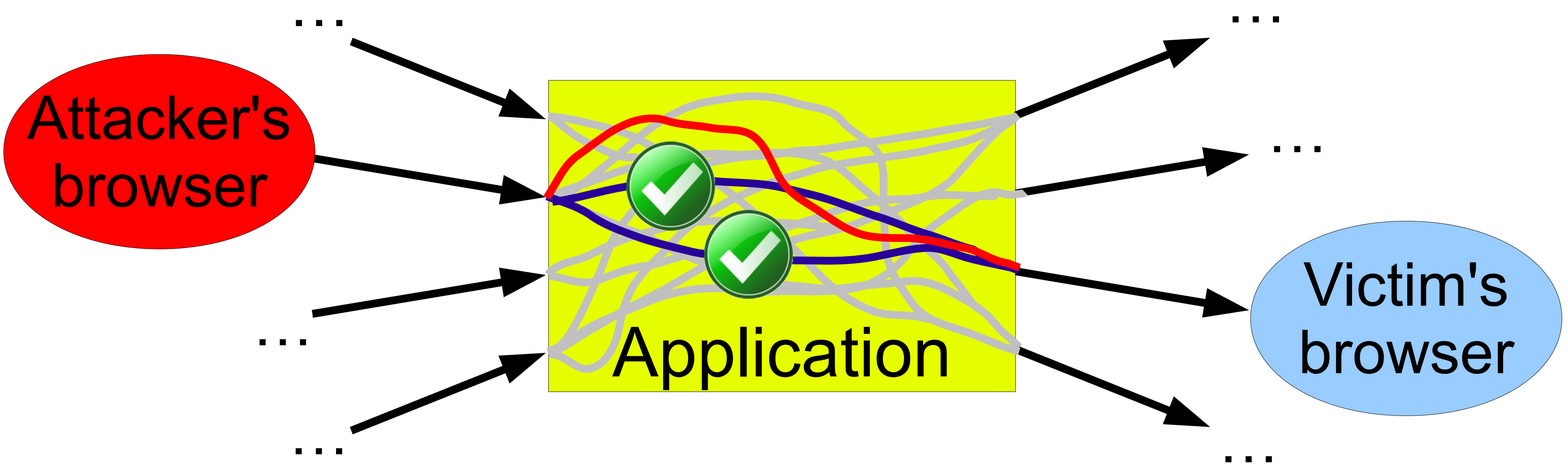
- Goal: remove Javascript from user input before using in HTML

Cross-site scripting attack



- Goal: remove Javascript from user input before using in HTML

Challenge: knowing where to check

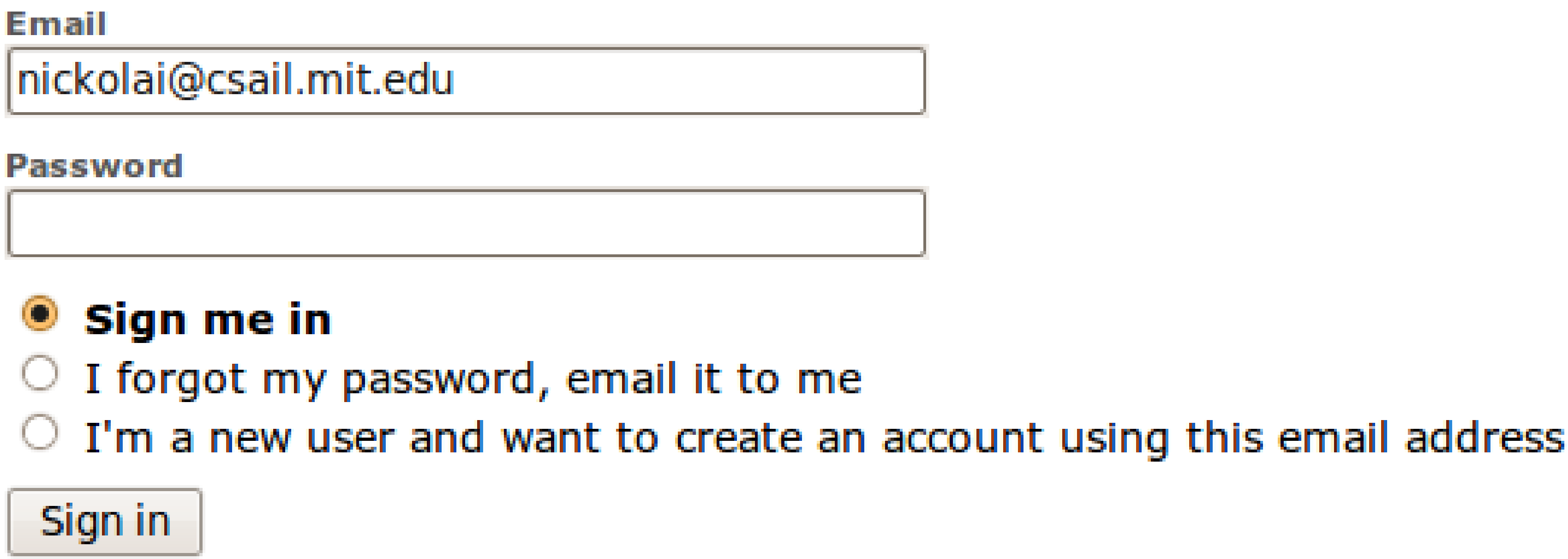


- Today: invoke check on all paths from source to sink
  - Easy to miss one (out of 572 in phpBB, a popular web app)
- Security check cannot be made based on data alone
  - At the source, don't know where data is going yet
  - At the sink, don't know where data came from

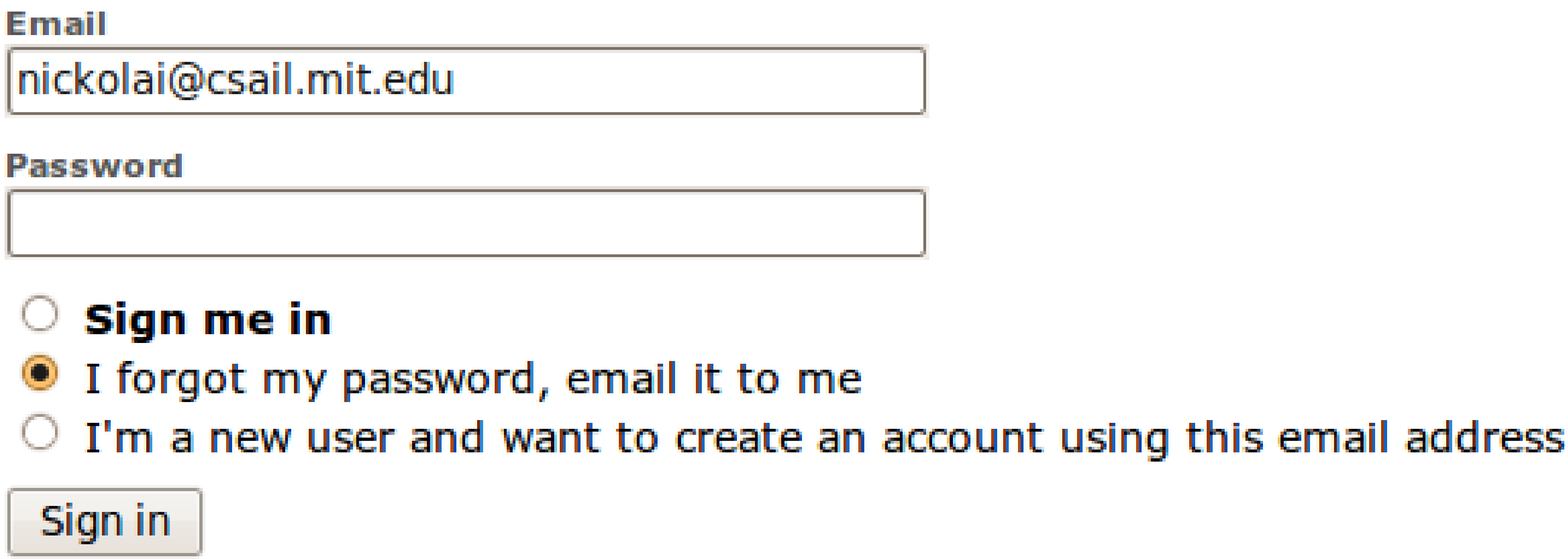
Approach: Associate checks with data

- 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

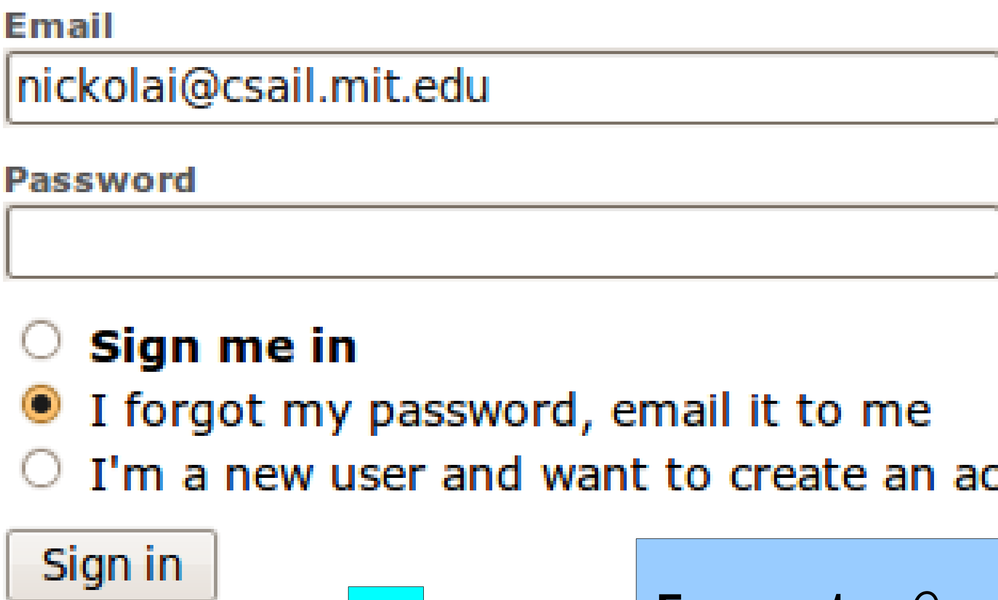
Example bug: HotCRP password disclosure



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

- ○ **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
```

Email

tom@cs.washington.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

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?

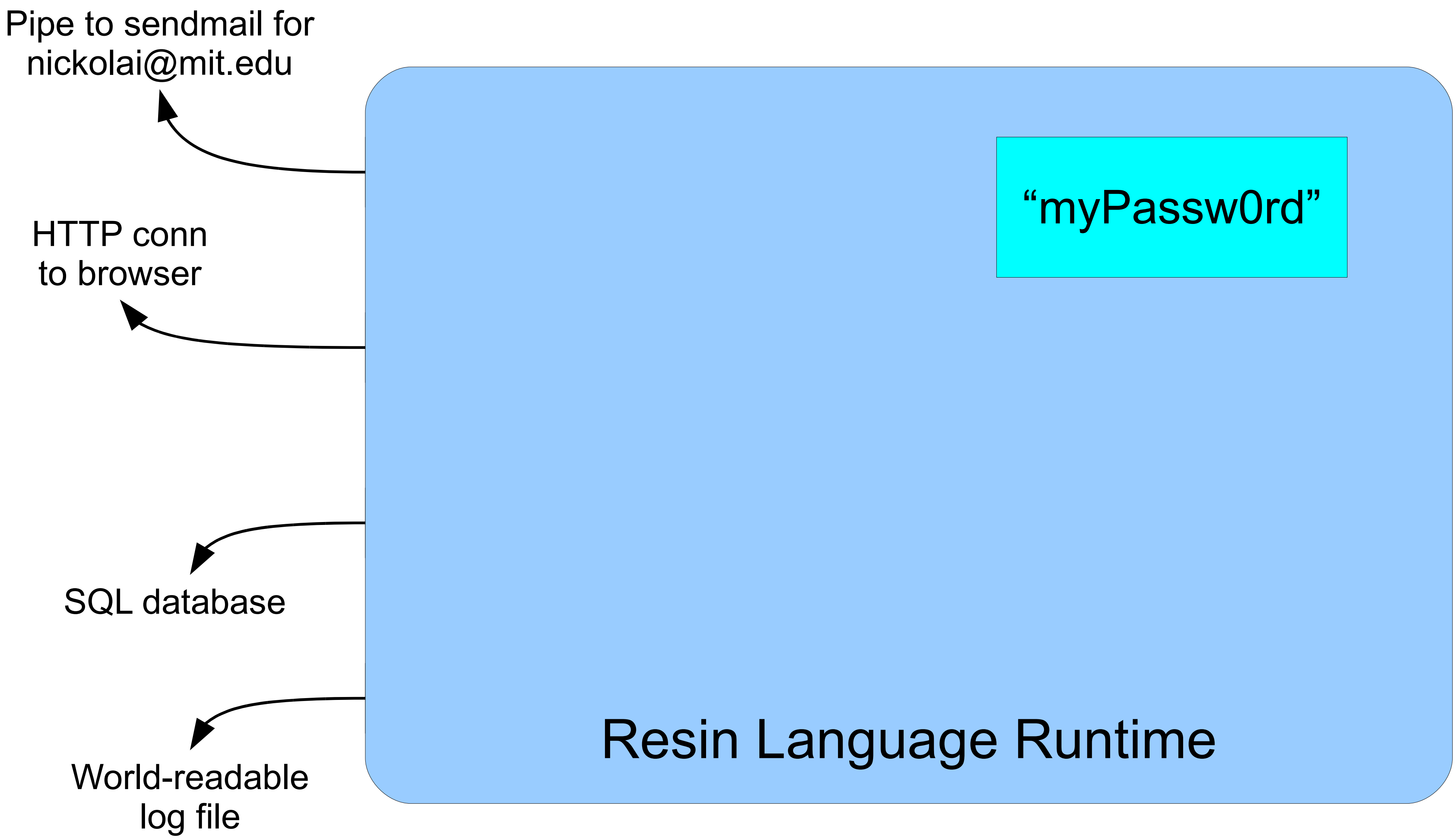
Challenges and ideas

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

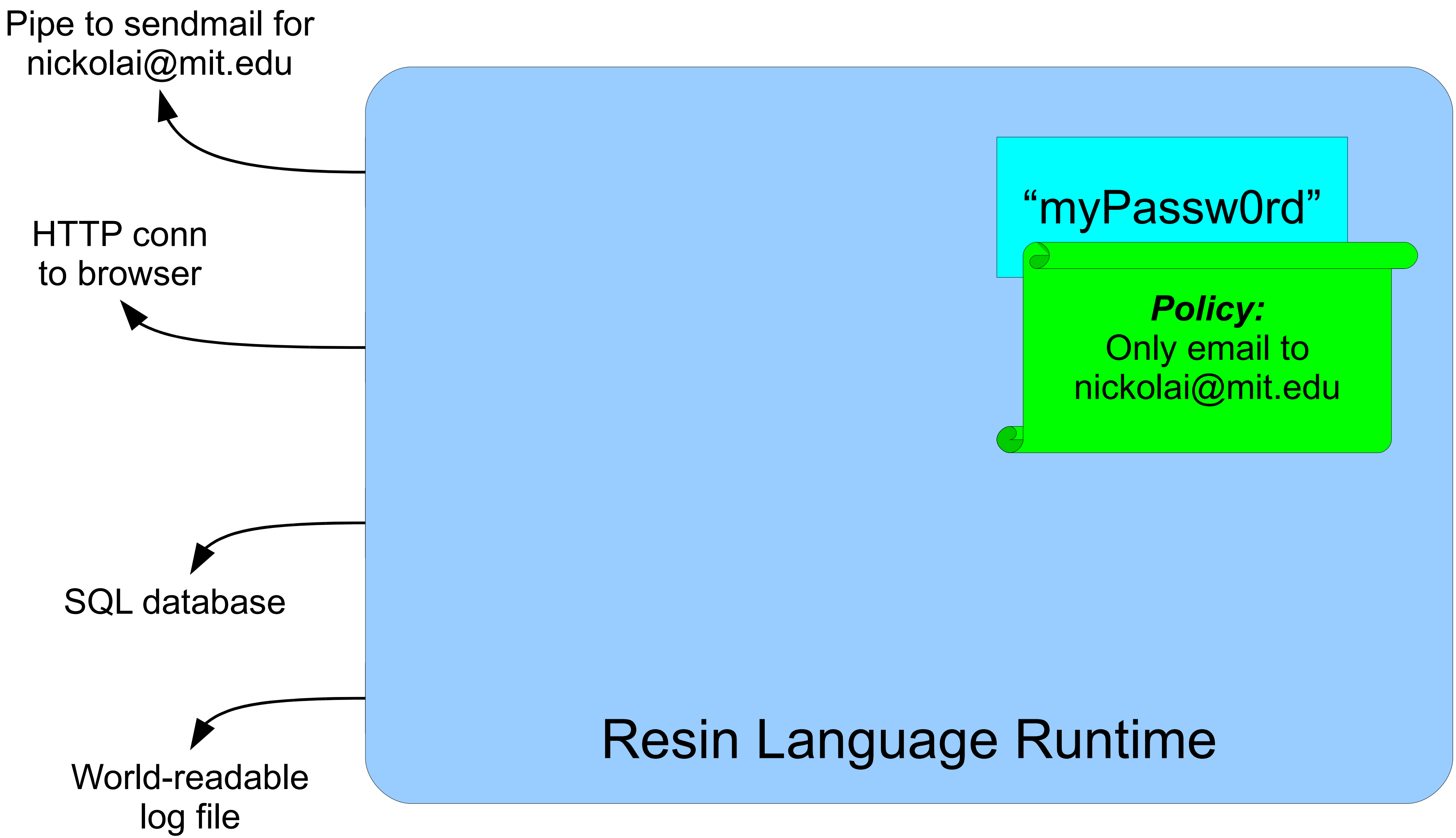
- How would we check if a program obeys this plan?
  - Associate the assertions with data (e.g. password)
  - Track assertions along with data in language runtime
  - Check at programmer-defined boundaries
    - *E.g.* external I/O (file, network), when data leaves our control

- How would the programmer express this assertion?
  - Express using code – simple, general-purpose
  - Programmers can reuse code, data structures

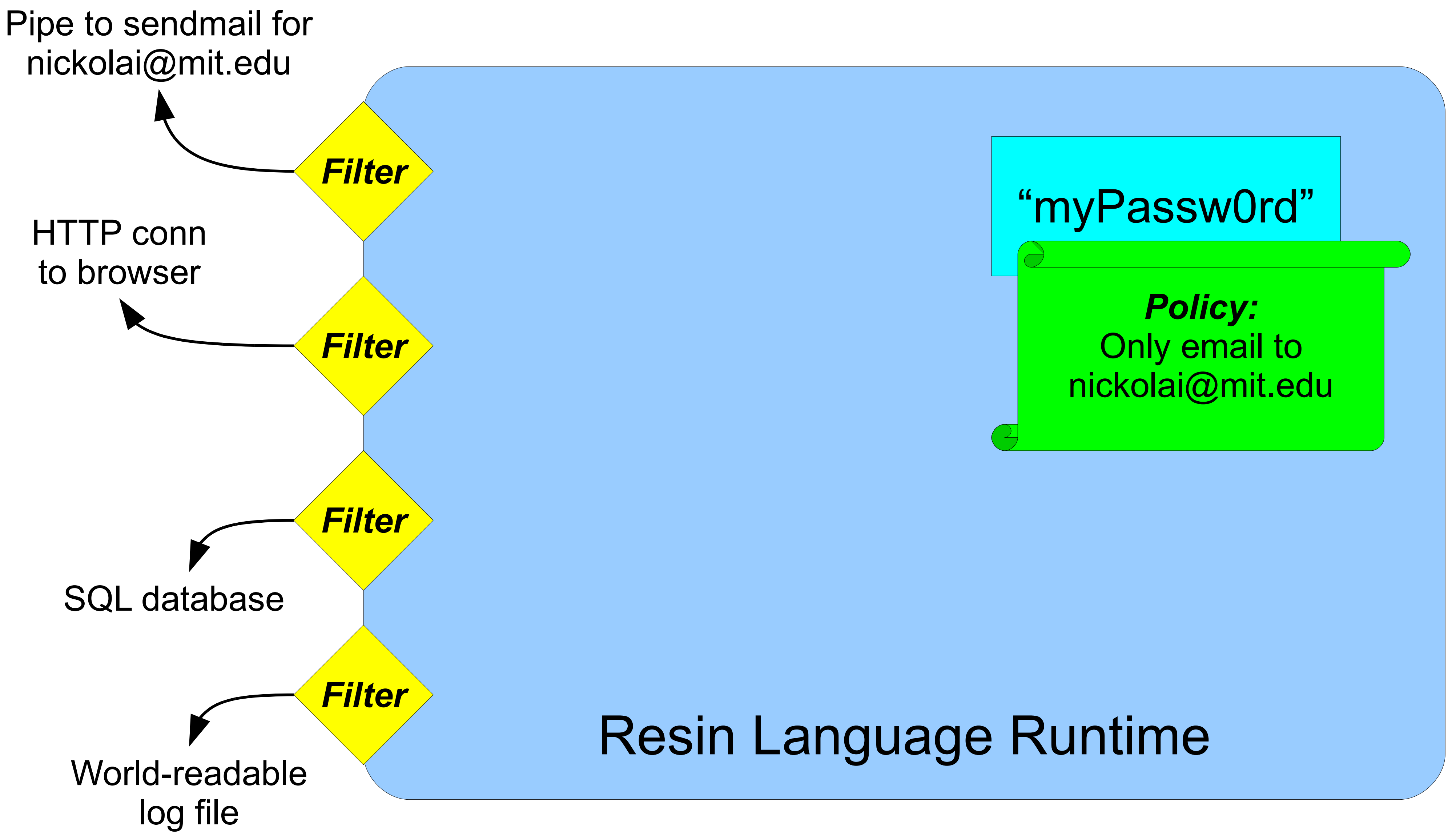
Example: Preventing HotCRP's bug in Resin



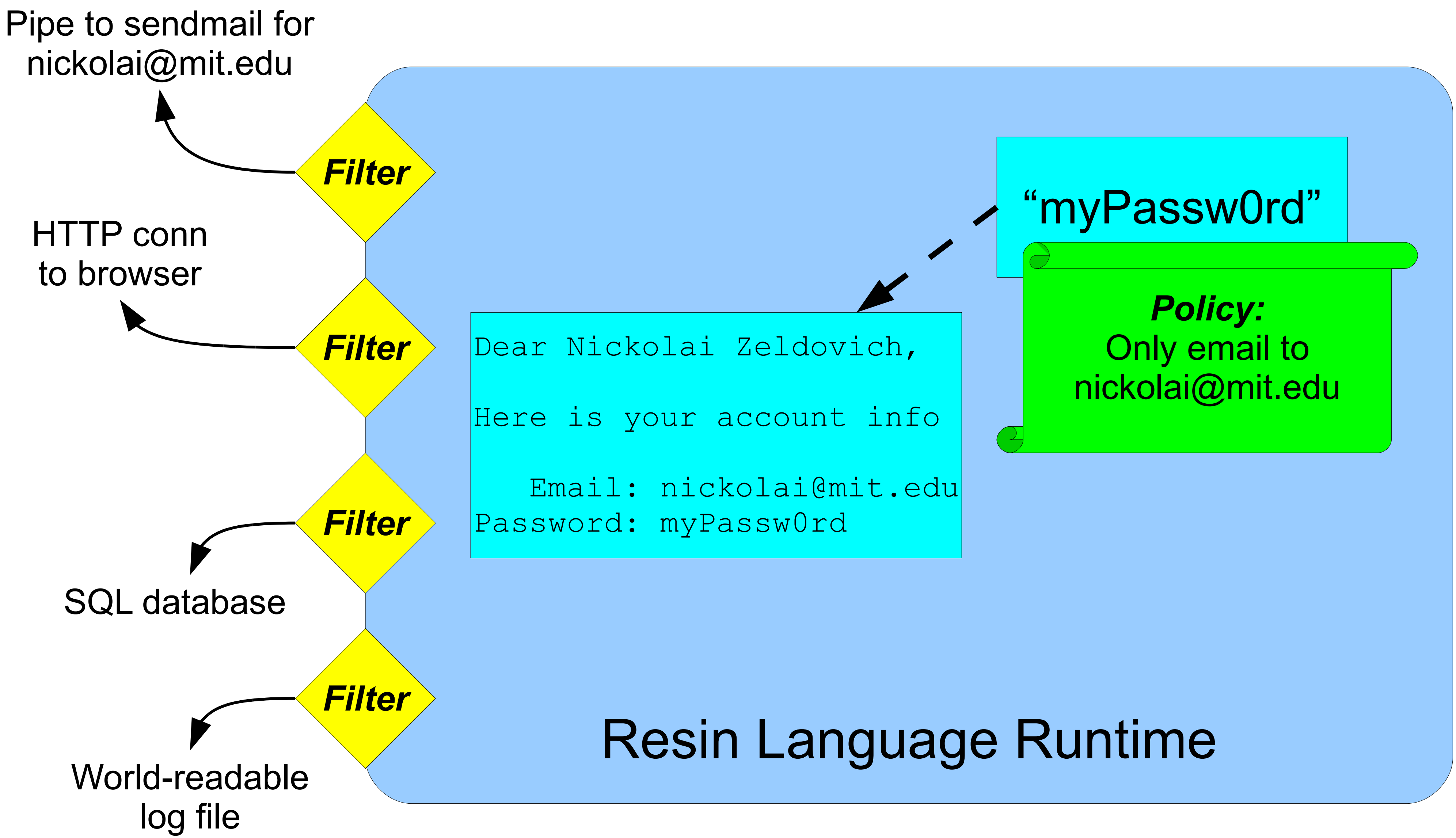
Programmer attaches a policy object to a string



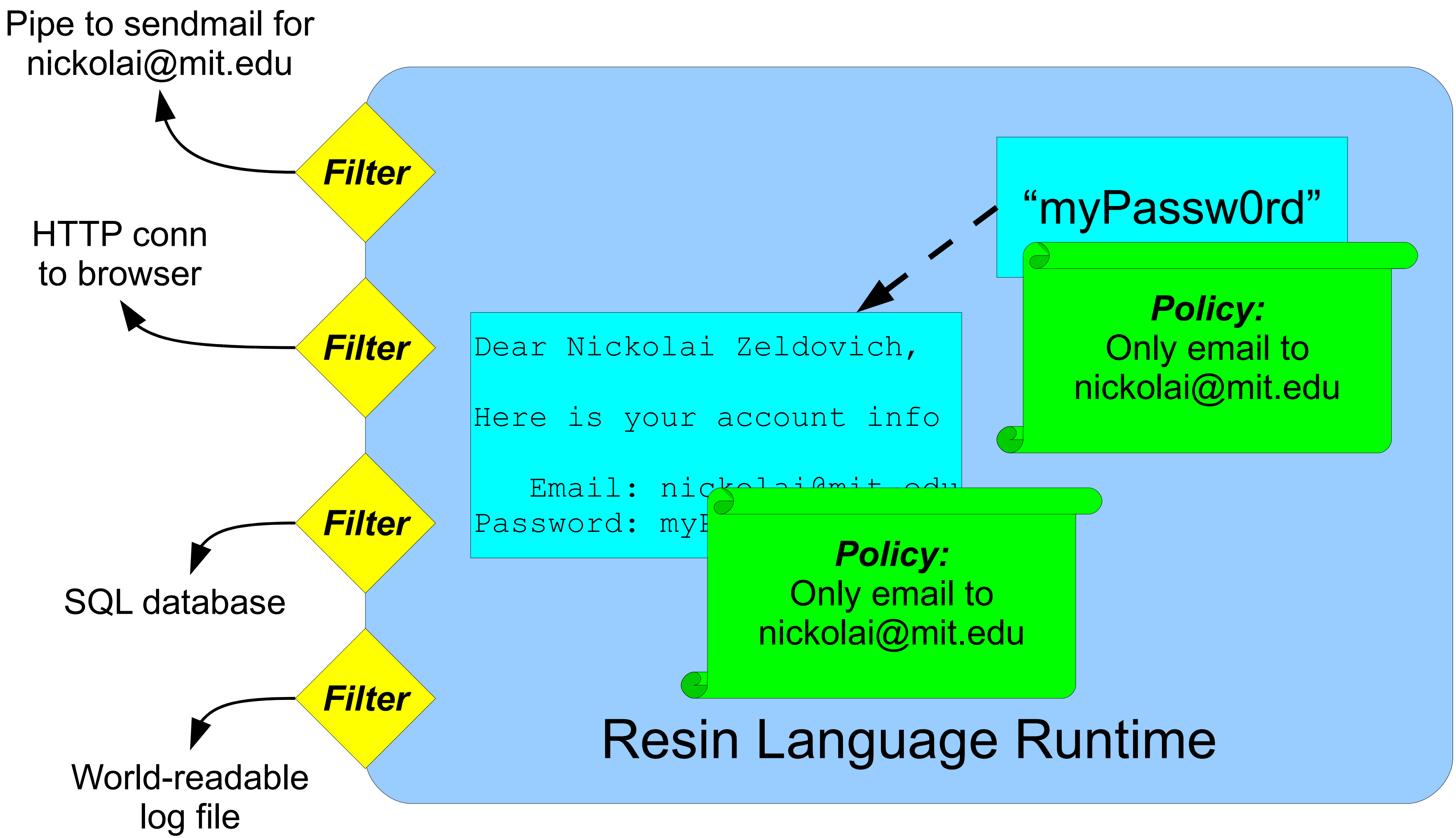
Programmer attaches filter objects to security boundaries



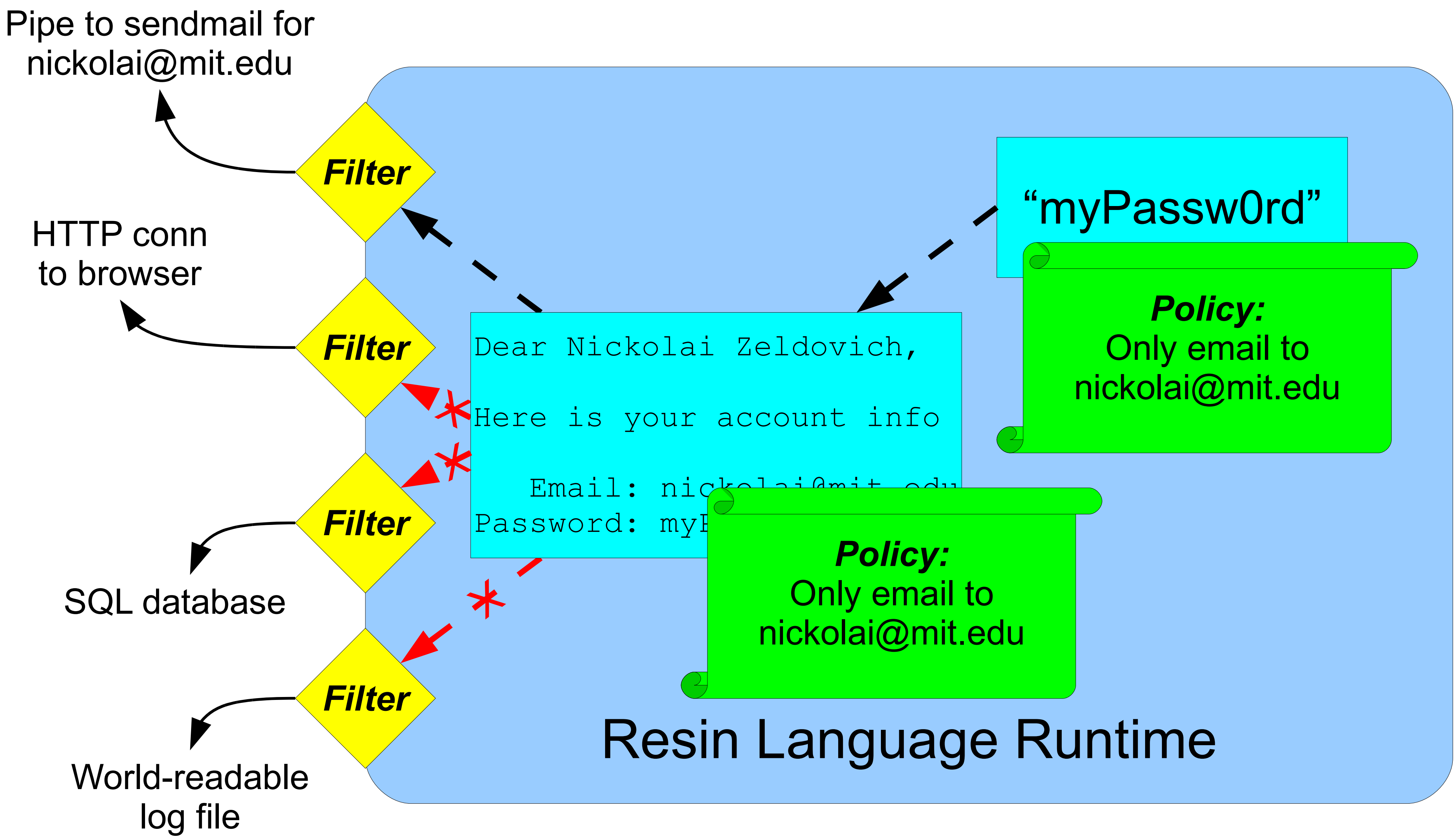
Runtime propagates policies for strings



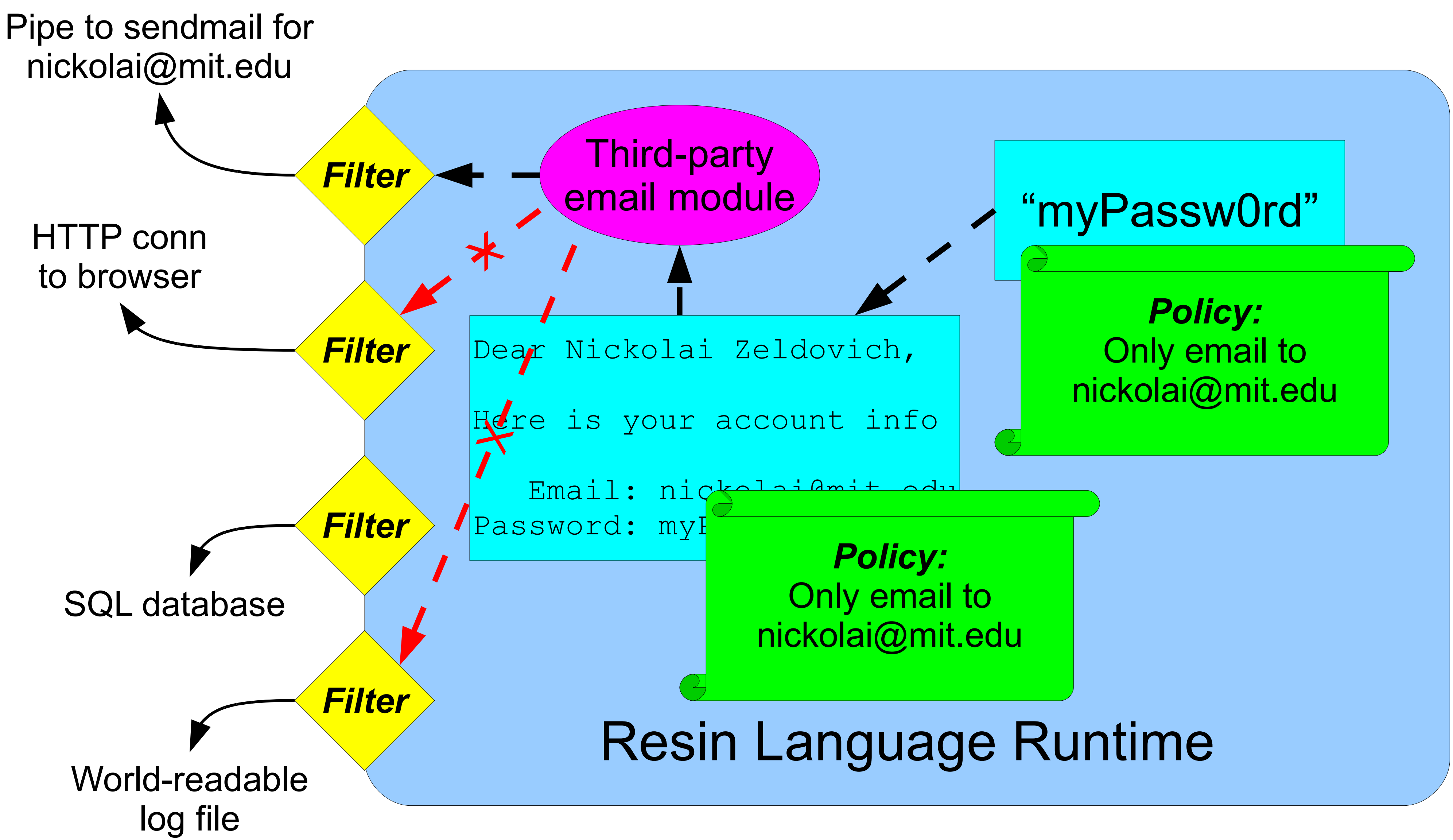
Runtime propagates policies for strings



Filters check assertions by invoking policy objects



Assertions avoid the need to understand all code



PHP code for HotCRP's policy

```
class PasswordPolicy extends Policy {

}
```

PHP code for HotCRP's policy

```
class PasswordPolicy extends Policy {
  private $user;

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




}
```

Stores owner's username (email address in HotCRP)

PHP code for HotCRP's policy

```
class PasswordPolicy extends Policy {
  private $user;

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

  function export_check($context) {




  }
}
```

Filter consults policy;
context provided by filter
at security boundary

PHP code for HotCRP's policy

```
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;



  }
}
```

Allows password to
be emailed to owner;
only cares about mail filter

PHP code for HotCRP's policy

```
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;

  }
}
```

Reuse code and data to allow PC chair override

PHP code for HotCRP's policy

```
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");
  }
}
```

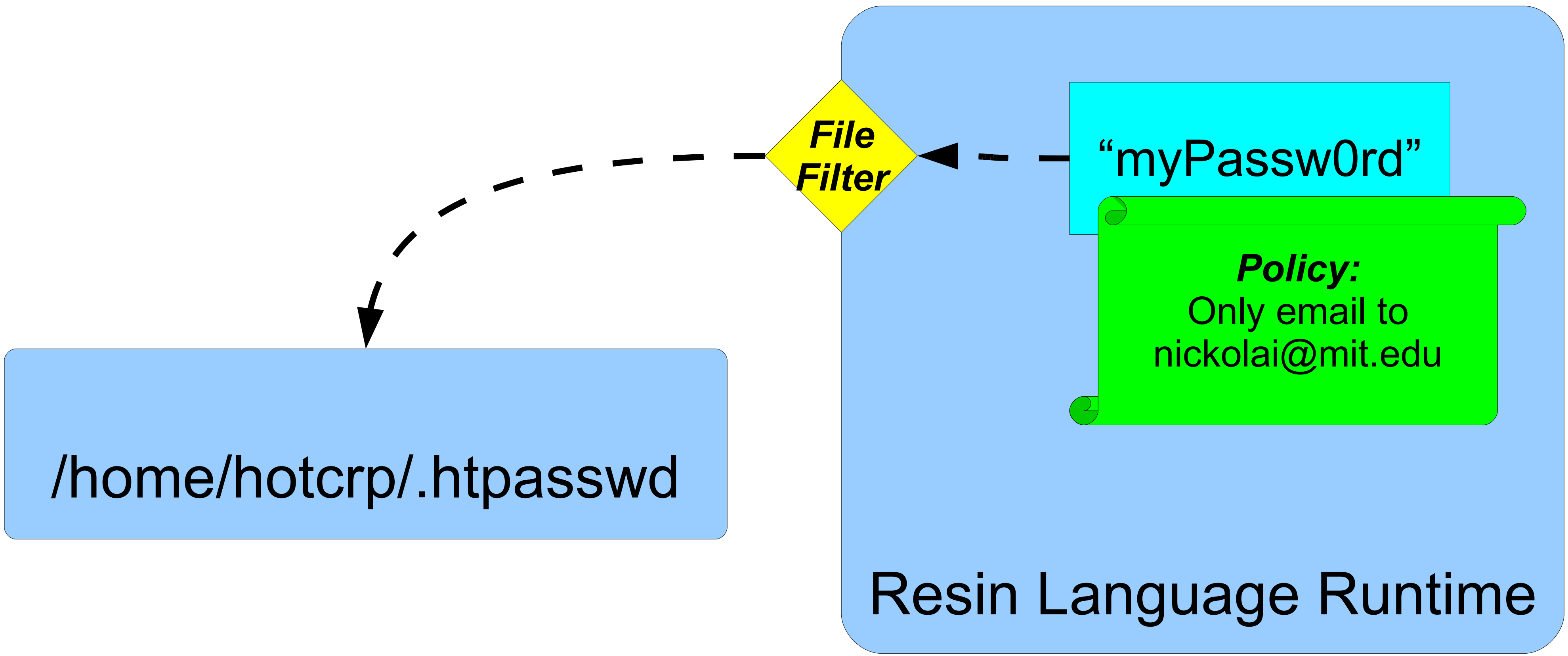
Otherwise, throw an exception to deny

PHP code for HotCRP's policy

```
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() &&
      return;
    throw new Excepti                              );
  }
}

policy_set($new_password, new PasswordPolicy($username));
```

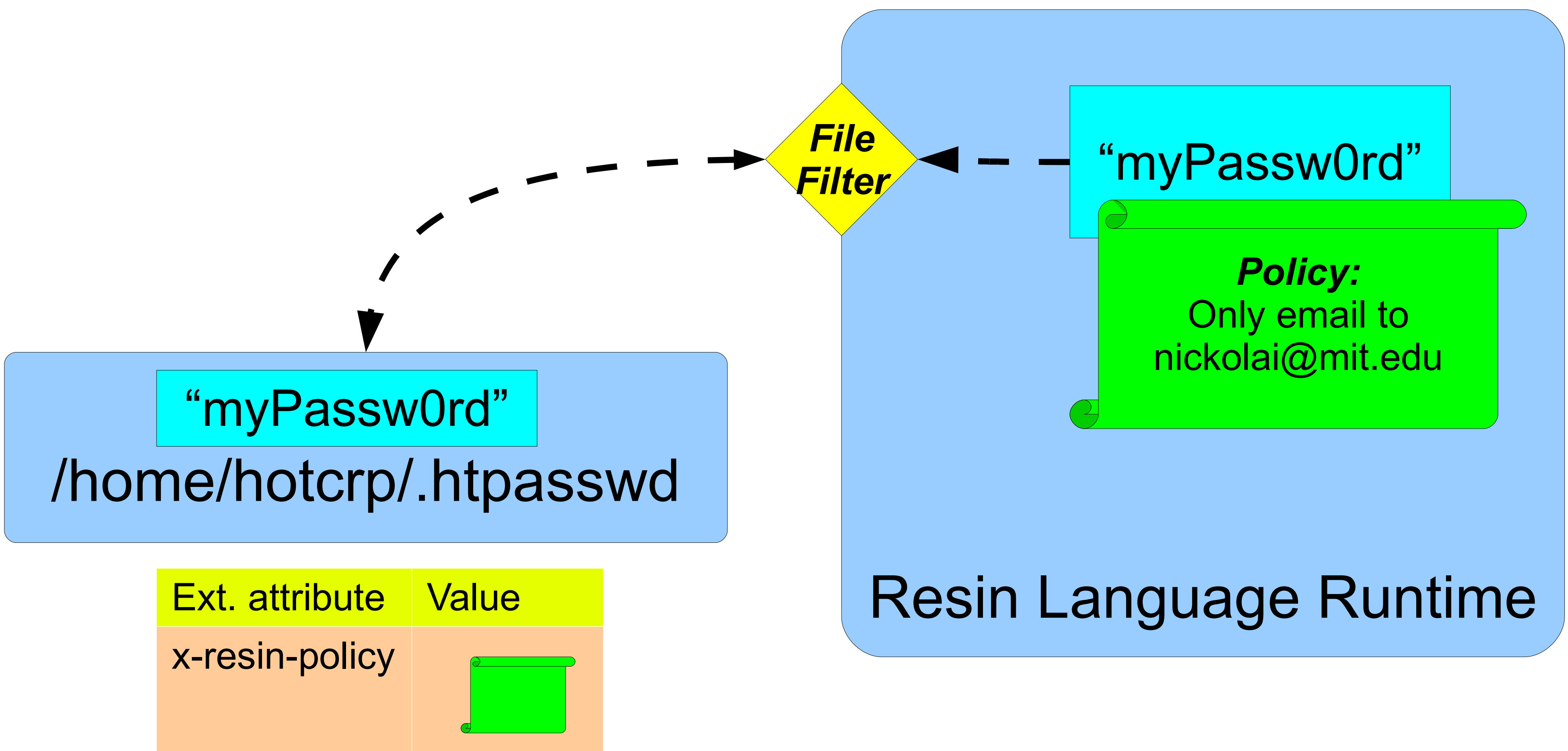
Specify policy once,
when data enters system

Filters help track persistent data



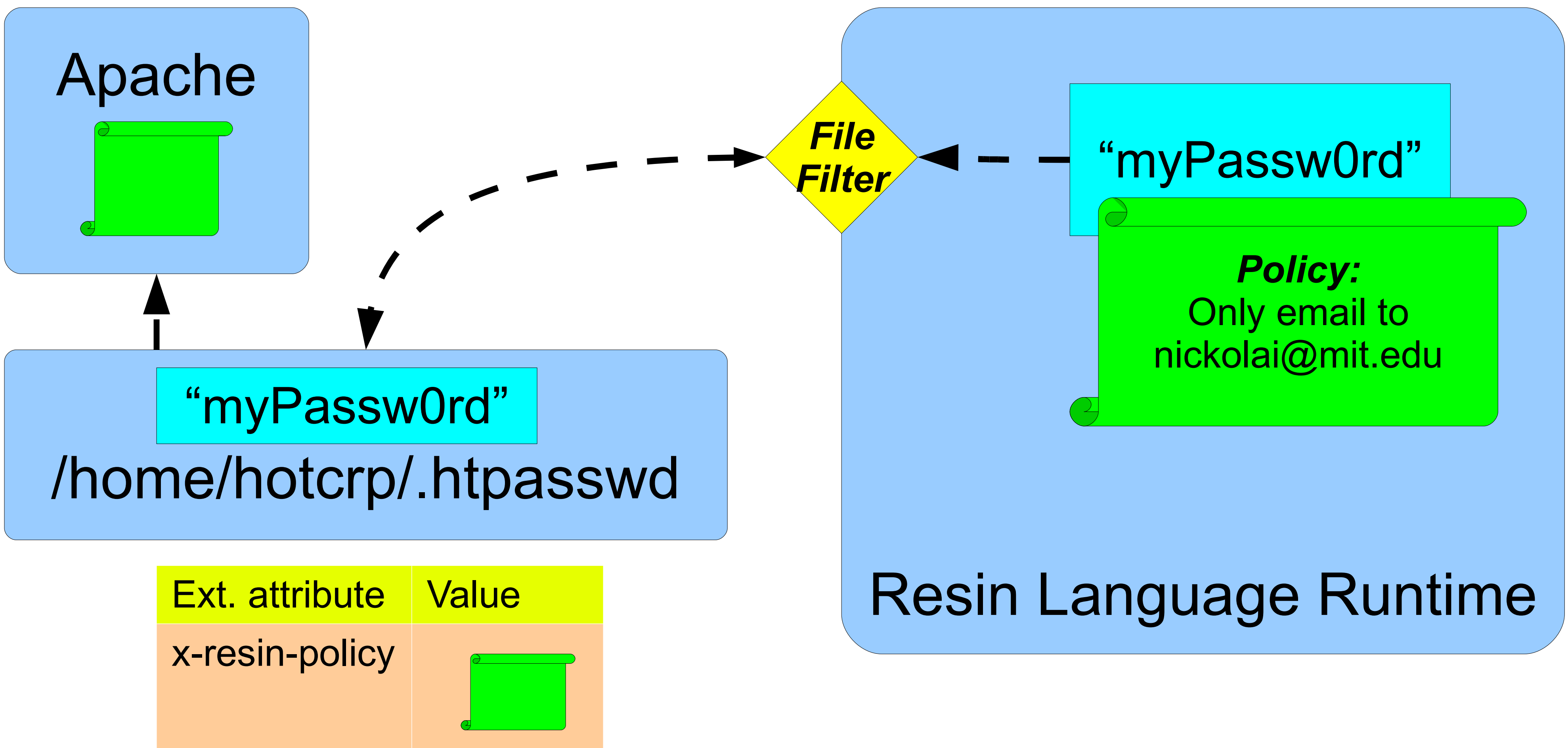
Filters help track persistent data

- File filter serializes/de-serializes policies to xattr



Filters help track persistent data

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



Tracking multiple policies

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

- Policies propagated on explicit data flows

  `a = concat(b, c)` propagates

  `a = 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

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

Assertions are easy to write

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

Assertions prevent a range of bugs

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

Assertions are not specific to attack vectors

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

- 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

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

- 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