RESIN

The pain of DStar

- It is too heavy, as an application engineer, we don’t necessarily need to understand how the operating system works.
- It is not super intuitive to build a mental model for the labels and categories.
- As an application engineer, the biggest abstraction we are dealing with is actually programming languages.

Overview of RESIN

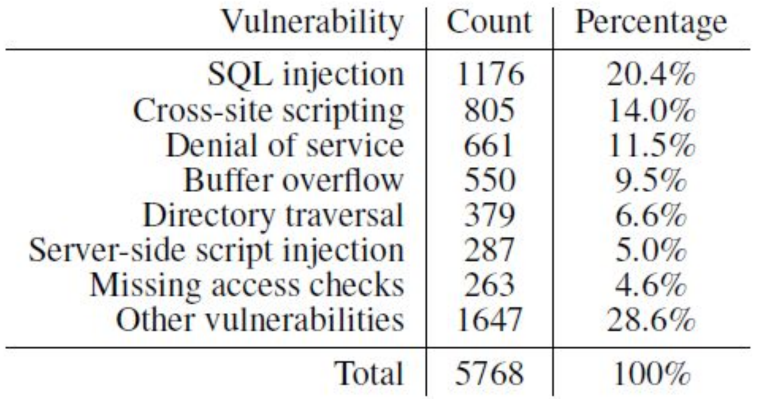
RESIN is a new language runtime that helps prevent security vulnerabilities, by allowing programmers to specify application-level data flow assertions.

1. **Policy object**: programmers use to specify assertion code and metadata
2. **Data tracking**: allows programmers to associate assertions with application data, and to keep track of assertions as the data flow through the application
3. **Filter object**: programmers use to define data flow boundaries at which assertions are checked

Vulnerabilities

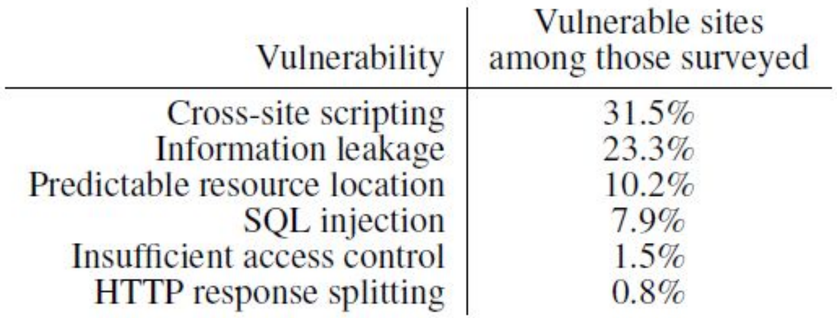
- **SQL Injection and XSS**: Any user input data must flow through a sanitization function before it flows into a SQL query.
- **Directory Traversal**: No data may flow into directory d unless the authenticated user has write permission for d.
- **Server Side Script Injection**: The interpreter may not interpret any user-supplied code.
- **Access Control**: Wiki page p may flow out of the system only to a user on p’s ACL.
- **Password Disclosure**: User u’s password may leave the system only via email to u’s email address, or to the program chair.

Prevalence of the Vulnerabilities



Vulnerability	Count	Percentage
SQL injection	1176	20.4%
Cross-site scripting	805	14.0%
Denial of service	661	11.5%
Buffer overflow	550	9.5%
Directory traversal	379	6.6%
Server-side script injection	287	5.0%
Missing access checks	263	4.6%
Other vulnerabilities	1647	28.6%
Total	5768	100%

Table 1: Top CVE security vulnerabilities of 2008 [41].



Vulnerability	Vulnerable sites among those surveyed
Cross-site scripting	31.5%
Information leakage	23.3%
Predictable resource location	10.2%
SQL injection	7.9%
Insufficient access control	1.5%
HTTP response splitting	0.8%

Table 2: Top Web site vulnerabilities of 2007 [48].

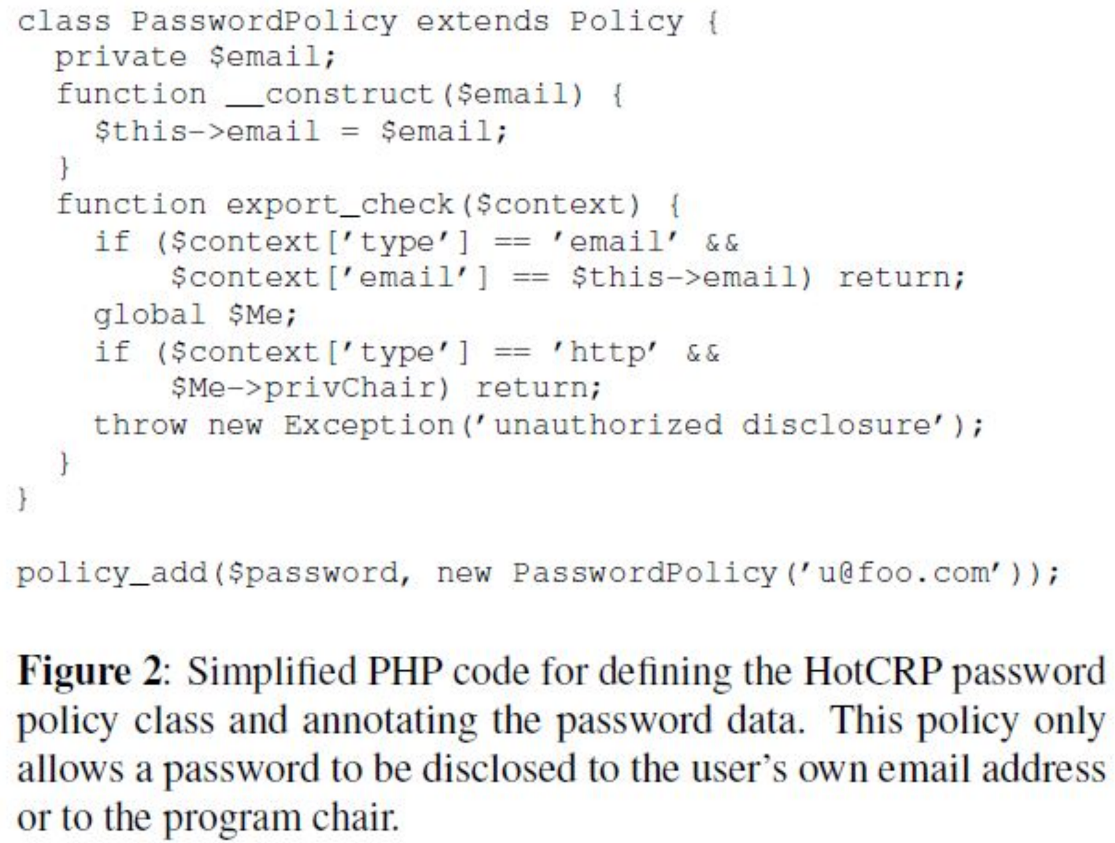
Treat Model and Design Challenges

Assumption:

- The code of application doesn’t have malicious intention.
- The language runtime is correct.

Filter Object	Programmer	Data boundaries
Policy Object	Programmer	Code and metadata for checking assertions
Data Tracking	RESIN runtime	RESIN runtime

A PHP example



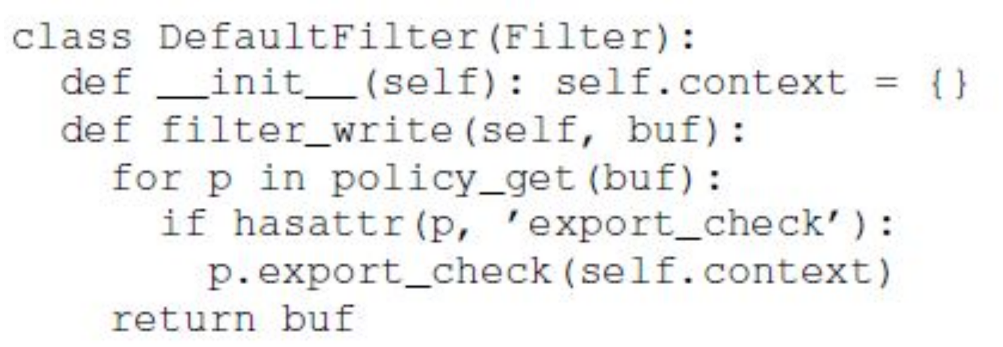
```
class PasswordPolicy extends Policy {
  private $email;
  function __construct($email) {
    $this->email = $email;
  }
  function export_check($context) {
    if ($context['type'] == 'email' &&
        $context['email'] == $this->email) return;
    global $Me;
    if ($context['type'] == 'http' &&
        $Me->privChair) return;
    throw new Exception('unauthorized disclosure');
  }
}

policy_add($password, new PasswordPolicy('u@foo.com'));
```

Figure 2: Simplified PHP code for defining the HotCRP password policy class and annotating the password data. This policy only allows a password to be disclosed to the user's own email address or to the program chair.

The Default Filter Object

By default, RESIN pre-defines filter objects on all I/O channels into and out of the runtime.



```
class DefaultFilter(Filter):
  def __init__(self): self.context = {}
  def filter_write(self, buf):
    for p in policy_get(buf):
      if hasattr(p, 'export_check'):
        p.export_check(self.context)
    return buf
```

Access Control Checks

1. annotate HTTP output channels with context that identifies the user on the other end of the channel;
2. define a PagePolicy object that contains an ACL;
3. implement an export_check method in PagePolicy that matches the output channel against the PagePolicy’s ACL;
4. attach a PagePolicy to the data in each wiki page.

Mitigating Server-Side Script Injection

1. define an empty CodeApproval policy object;
2. annotate application code and libraries with CodeApproval policy objects;
3. change the interpreter’s default input filter (see Section 3.2.2) to require a CodeApproval policy on all imported code.

Mitigating SQL Injection and XSS

1. defines two policy object classes: UntrustedData and SQLSanitized;
2. annotates untrusted input data with an UntrustedData policy;
3. changes the existing SQL sanitization function to attach a SQLSanitized object to the freshly sanitized data;
4. changes the SQL filter object to check the policy objects on each SQL query. If the query contains any characters that have the UntrustedData policy, but not the SQLSanitized policy, the filter will throw an exception and refuse to forward the query to the database.

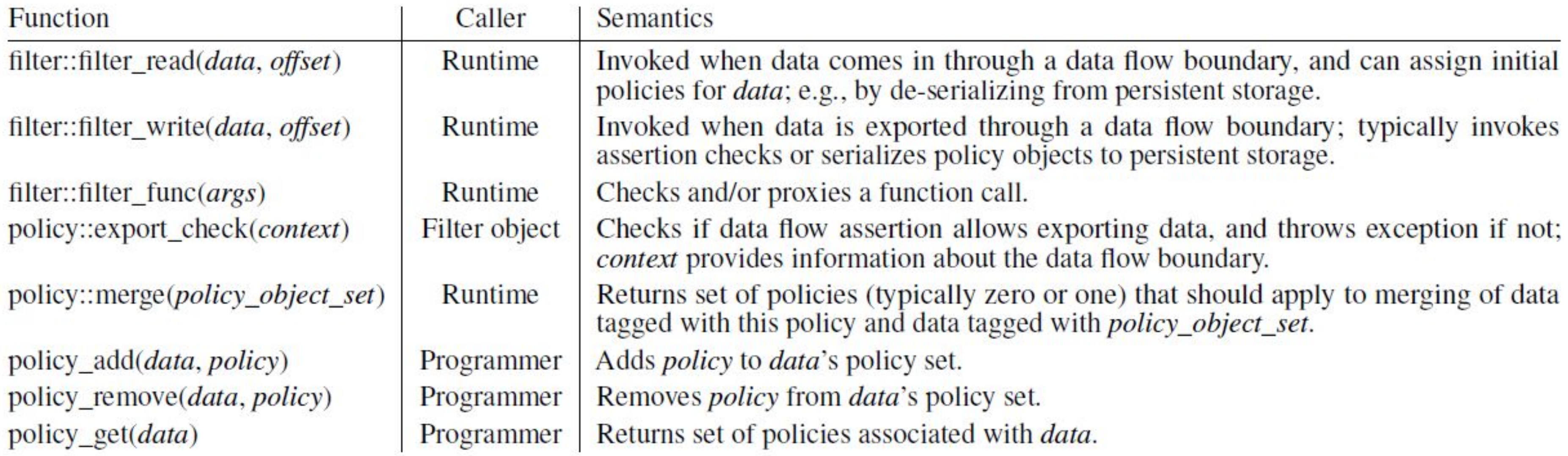
Mitigating Split HTTP Attack

A malicious user can inject a CRLF in his input and if the server blindlessly trusts the user and included the request in the response, the user can trick the browser.

Solution:

Attach a filter object at the I/O channel talking to the client to filter out any input that has a CRLF

The programming interface



Function	Caller	Semantics
filter::filter_read(*data*, *offset*)	Runtime	Invoked when data comes in through a data flow boundary, and can assign initial policies for *data*; e.g., by de-serializing from persistent storage.
filter::filter_write(*data*, *offset*)	Runtime	Invoked when data is exported through a data flow boundary; typically invokes assertion checks or serializes policy objects to persistent storage.
filter::filter_func(*args*)	Runtime	Checks and/or proxies a function call.
policy::export_check(*context*)	Filter object	Checks if data flow assertion allows exporting data, and throws exception if not; *context* provides information about the data flow boundary.
policy::merge(*policy_object_set*)	Runtime	Returns set of policies (typically zero or one) that should apply to merging of data tagged with this policy and data tagged with *policy_object_set*.
policy_add(*data*, *policy*)	Programmer	Adds *policy* to *data*'s policy set.
policy_remove(*data*, *policy*)	Programmer	Removes *policy* from *data*'s policy set.
policy_get(*data*)	Programmer	Returns set of policies associated with *data*.

Table 3: The RESIN API. A::B(args) denotes method B of an object of type A. Not shown is the API used by the programmer to specify and access filter objects for different data flow boundaries.

Merging Policies

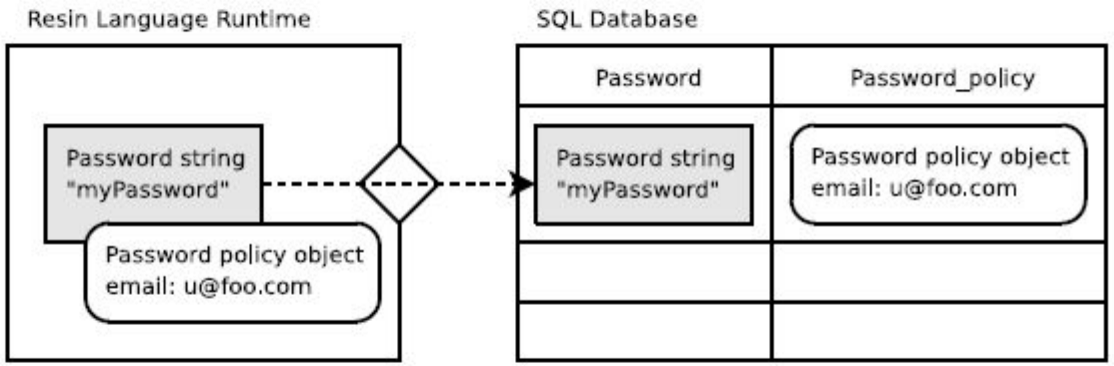
RESIN uses character-level tracking to avoid having to merge policies when individual data elements are propagated, such as through concatenation or taking a substring.

Programmer can choose to use two strategies

- UNION (default): UserData
- INTERSECTION: AuthenticData

Persistent Policies

Many applications store data persistently in file systems and databases.



Resin Language Runtime
Password string "myPassword"
Password policy object email: u@foo.com
SQL Database
Password
Password_policy
Password string "myPassword"
Password policy object email: u@foo.com

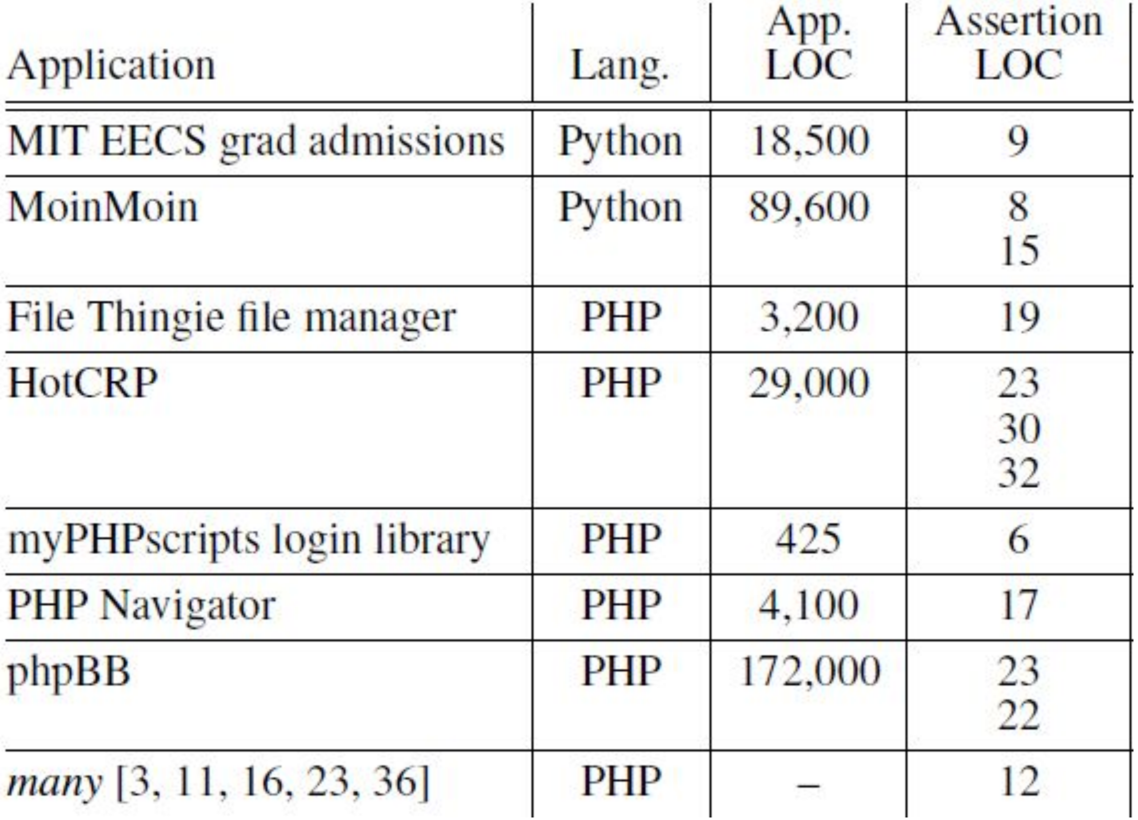
Figure 4: Saving persistent policies to a SQL database for HotCRP passwords. Uses symbols from Figure 1.

Output Buffering

To use output buffering, the application starts a new try block before running HTML generation code that might throw an exception. At the start of the try block, the application notifies the outgoing HTML filter object to start buffering output. If the try block throws an exception, the corresponding catch block notifies the HTML filter to discard the output buffer, and potentially send alternate output in its place.

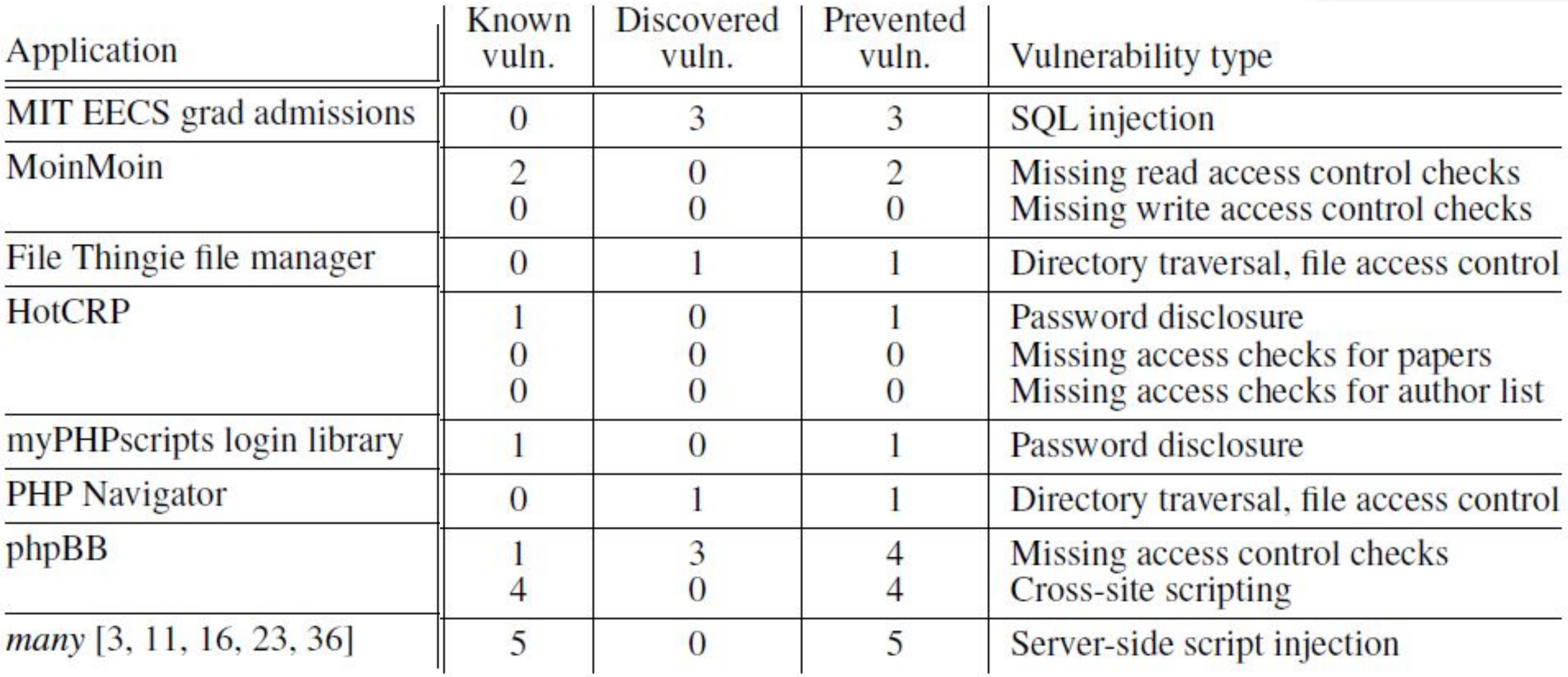
Security Evaluation

Programming Effort



Application	Lang.	App. LOC	Assertion LOC
MIT EECS grad admissions	Python	18,500	9
MoinMoin	Python	89,600	8
			15
File Thingie file manager	PHP	3,200	19
HotCRP	PHP	29,000	23
			30
			32
myPHPscripts login library	PHP	425	6
PHP Navigator	PHP	4,100	17
phpBB	PHP	172,000	23
			22
many [3, 11, 16, 23, 36]	PHP	–	12

Preventing Vulnerabilities



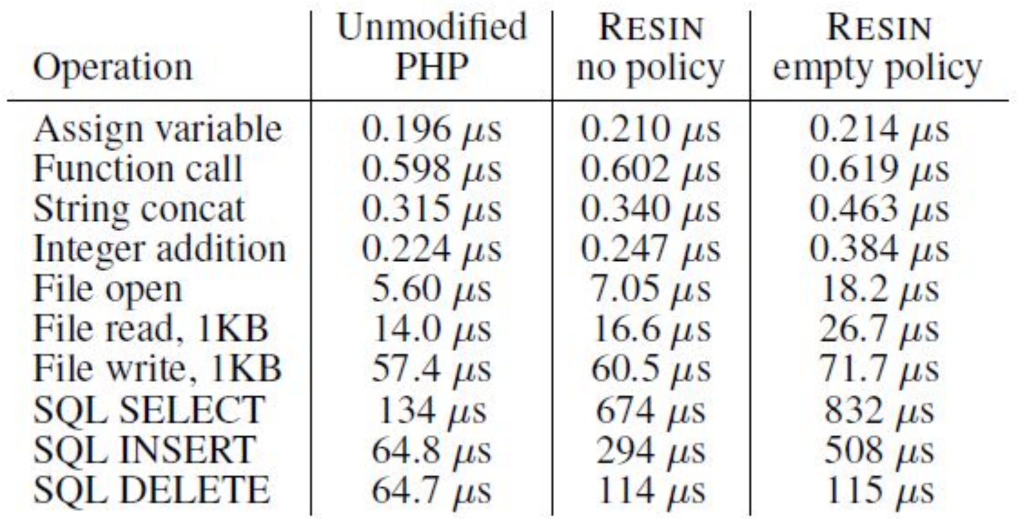
Application	Known vuln.	Discovered vuln.	Prevented vuln.	Vulnerability type
MIT EECS grad admissions	0	3	3	SQL injection
MoinMoin	2	0	2	Missing read access control checks
	0	0	0	Missing write access control checks
File Thingie file manager	0	1	1	Directory traversal, file access control
HotCRP	1	0	1	Password disclosure
	0	0	0	Missing access checks for papers
	0	0	0	Missing access checks for author list
myPHPscripts login library	1	0	1	Password disclosure
PHP Navigator	0	1	1	Directory traversal, file access control
phpBB	1	3	4	Missing access control checks
	4	0	4	Cross-site scripting
many [3, 11, 16, 23, 36]	5	0	5	Server-side script injection

Generality

- PhpBB requesting from whois
- PhpBB message quoting
- myPHPscripts login library password disclosure

Performance Evaluation

Performance impact



Operation	Unmodified PHP	RESIN no policy	RESIN empty policy
Assign variable	0.196 μs	0.210 μs	0.214 μs
Function call	0.598 μs	0.602 μs	0.619 μs
String concat	0.315 μs	0.340 μs	0.463 μs
Integer addition	0.224 μs	0.247 μs	0.384 μs
File open	5.60 μs	7.05 μs	18.2 μs
File read, 1KB	14.0 μs	16.6 μs	26.7 μs
File write, 1KB	57.4 μs	60.5 μs	71.7 μs
SQL SELECT	134 μs	674 μs	832 μs
SQL INSERT	64.8 μs	294 μs	508 μs
SQL DELETE	64.7 μs	114 μs	115 μs

Table 5: The average time taken to execute different operations in an unmodified PHP interpreter, a RESIN PHP interpreter without any policy, and a RESIN PHP interpreter with an empty policy.

Limitation and Future Work

- Verify the data flow assertion at the lower level.
- Not able to construct internal data flow boundaries within an application
- Cannot span multiple runtimes
- Easier ways to implement data tracking in a language runtime with OS or VMM support
- Reduce the performance overhead

Thank you