Resin: runtime-enforced information flow control

...
Plan

1. What is information flow control? What can it be used for?
2. How does Resin enforce information flow control?
3. Are there other approaches to information flow control?
4. Discussion!
What is [Information Flow] [Control]?
Web applications are about moving data around between users and components of the application!
Moving data $\Rightarrow$ Information Flow
Example Flow - Password Reset (1)
Example Flow - Password Reset (2)
Example Flow - Password Reset (3)
Example Flow - Password Reset (4)
Example Flow - Discussion Board (3)
Example Flow - Discussion Board (4)
Example Flow - Discussion Board (5)
Example Flow - Discussion Board (6)
Information Flow Control:
Must ensure that the “correct” information flows from/to the “correct” entities!
Information flow *entities* are broader than you think...

- Physically separated: Users / parties / servers

- Logically separated:
  - Trusted vs Untrusted code bases
  - Different threads or processes
  - Program/code components (classes, functions, etc)
Information flow *entities* are broader than you think...

![Diagram showing kernel space, user space, and kernel interface (system calls) with user processes 1 and 2.](image-url)
Information flow *entities* are broader than you think...

Diagram:
- **Android Operating System**
- **Kernel Interface (System Calls)**
- **Android Application**
  - Whatsapp
  - Contacts
More common than you think!

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

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

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site scripting</td>
<td>31.5%</td>
</tr>
<tr>
<td>Information leakage</td>
<td>23.3%</td>
</tr>
<tr>
<td>Predictable resource location</td>
<td>10.2%</td>
</tr>
<tr>
<td>SQL injection</td>
<td>7.9%</td>
</tr>
<tr>
<td>Insufficient access control</td>
<td>1.5%</td>
</tr>
<tr>
<td>HTTP response splitting</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

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

Percentage out of all reported vulnerabilities

Percentage of websites suffering from this vulnerability
More common than you think!

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

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOL injection</td>
<td>1176</td>
<td>20.4%</td>
</tr>
<tr>
<td>Cross-site scripting</td>
<td>805</td>
<td>14.0%</td>
</tr>
<tr>
<td>Denial of service</td>
<td>661</td>
<td>11.5%</td>
</tr>
<tr>
<td>Buffer overflow</td>
<td>550</td>
<td>9.5%</td>
</tr>
<tr>
<td>Directory traversal</td>
<td>379</td>
<td>6.6%</td>
</tr>
<tr>
<td>Server-side script injection</td>
<td>287</td>
<td>5.0%</td>
</tr>
<tr>
<td>Missing access checks</td>
<td>263</td>
<td>4.6%</td>
</tr>
<tr>
<td>Other vulnerabilities</td>
<td>1647</td>
<td>28.6%</td>
</tr>
<tr>
<td>Total</td>
<td>5768</td>
<td>100%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Percentage of websites suffering from this vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site scripting</td>
<td>31.5%</td>
</tr>
<tr>
<td>Information leakage</td>
<td>23.3%</td>
</tr>
<tr>
<td>Predictable resource location</td>
<td>10.2%</td>
</tr>
<tr>
<td>SOL injection</td>
<td>7.9%</td>
</tr>
<tr>
<td>Insufficient access control</td>
<td>1.5%</td>
</tr>
<tr>
<td>HTTP response splitting</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

Percentage out of all reported vulnerabilities

Percentage of websites suffering from this vulnerability
Bad Flow - Cross-Site Scripting (1)
Bad Flow - Cross-Site Scripting (2)

Regular comment about something...

```html
<script>
Malicious()
</script>
```
Bad Flow - Cross-Site Scripting (3)
Bad Flow - Cross-Site Scripting (4)
Bad Flow - Cross-Site Scripting (5)
Bad Flow - Cross-Site Scripting (6)
Bad Flow - Cross-Site Scripting (7)
Bad Flow - Cross-Site Scripting (7)
Good Flow

Sanitization Function
Regular comment about something...

```html
<script>
    Malicious()
</script>
```
Good Flow

Regular comment about something...
\[\text{\&lt;script\&rt; Malicious() \&lt;/script\&rt;}\]
Good Flow
Goal of Resin

- Help programmers avoid information flow mistakes
  - Omitting checks

- API for explicitly defining information flow assertions

- Automatic enforcement of these assertions
  - Runtime enforcement
How does Resin work?
def insert_post(request, response):
    post_content = request.post_content
    # Insert the new post into database
    Database.insert(post_content)
    # Signal success to user
    response.send_to_user("success")
def get_post(request, response):
    post_id = request.post_id
    # Look up post content from database
    post_content = Database.lookup(post_id)
    # Send post content to requesting user
    response.send_to_user(post_content)
def get_post(request, response):
    post_id = request.post_id
    # Look up post content from database
    post_content = Database.lookup(post_id)
    # Send post content to requesting user
    response.send_to_user(post_content)
def insert_post(request, response):
    post_content = request.post_content
    # Insert the new post into database
    Database.insert(post_content)
    # Signal success to user
    response.send_to_user("success")

Comes from a user! (Policy Object)
Retracing Resin’s Design (5)

def get_post(request, response):
    post_id = request.post_id
    # Look up post content from database
    post_content = Database.lookup(post_id)
    # Send post content to requesting user
    response.send_to_user(post_content)

Potentially came from a user!
(data tracking)

Something that was potentially came from a user is sent to a user!
(filter objects)
Resin Design

1. Programmers explicitly annotate data with policy objects
2. Programmers use filter objects to define boundaries
   a. Filter is like a channel
   b. Filter checks that the data going through this channel is annotated with appropriate policy.
3. Resin automatically tracks annotations as its associated data moves around.
Resin Design

1. Programmers explicitly annotate data with policy objects
2. Programmers use filter objects to define boundaries
   a. Filter is like a channel
   b. Filter checks that the data going through this channel is annotated with appropriate policy.
3. Resin automatically tracks annotations as its associated data moves around.

How would you implement something like this?
def insert_post(request, response):
    post_content_and_policy =
        (request.post_content, {"comes_from_user": True})
    # Insert the new post into database
    Database.insert(post_content_and_policy)
    # Signal success to user
    response.send_to_user("success")
def get_post(request, response):

    post_id = request.post_id

    # Look up post content from database
    post_content_and_policy = Database.lookup(post_id)

    # Send post content to requesting user
    user_filter(response.send_to_user, post_content_and_policy)
def user_filter(channel, data_and_policy):
    data, policy = data_and_policy
    if policy.comes_from_user:
        raise Error("Unsafe!")
    channel(data)
Design Evaluation (1)

Advantages:
Design Evaluation (2)

Advantages:

1. Simplicity!

Disadvantages:
Design Evaluation (3)

Advantages:

1. Simplicity!

Disadvantages:

1. Enforcement at runtime adds overhead (both space and time!)
   a. Resin has 33% runtime overhead

2. (for sample implementation, but not for Resin) Data tracking may be inaccurate
Taint Laundering

string_data, _ = string_data_and_policy

do_unsafe_things(string_data)

new_string = library_function(string_data_and_policy)

do_unsafe_things(new_string)
Other Issues

# what should be concat's policy!?

concat = string_with_policy1 + string_with_policy2

# what should be sum's policy?

sum = int_with_policy1 + int_with_policy2
Resin Design (1)

- Web languages are interpreted (python, php, nodejs, etc.)
  - Modify the runtime of the language so that the taints are stored within the runtime
  - “Make the taint part of the language”
  - Whenever the runtime interprets an operation, it can track the taint!
  - Similar to what happens in python if you add a string to an int!
Resin Design (2)

bab@bab-machine:~$ python
Python 2.7.18rc1 (default, Apr 7 2020, 12:05:55)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "Kinan" + 100
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>>
Detour (1) - Compiled vs Interpreted Languages

How Compiler Works

Source Code → Compiler → Machine Code → Output

© guru99.com

How Interpreter Works

Source Code → Interpreter → Output

Detour (2) - Language Runtime

Credit: https://medium.com/@olinations/the-javascript-runtime-environment-d58fa2e60dd0
Resin Design (3)

- Strings policy are set with respect to a range
  - Substrings can have different policies

- Operations on data with different policies combines the policies.
Resin Design (4)

- Resin does not protect against malicious developers:
  - They can intentionally mis-use policy objects and filters

- Resin does not protect against malicious or compromised code bases:
  - Resin does not protect against non-information flow based attacks (e.g. buffer overflow)
  - These attacks can be used to disable Resin’s runtime protections or corrupt taints
Resin Design (5)

- Resin does not track implicit flow:
  - Challenging to track and discover implicit flow
  - Unclear what the policy should be
  - Developers should transform implicit flow to explicit ones

```python
if condition(var_with_policy):
    # Information flows from var_with_policy to var_without_policy implicitly
    var_without_policy = some_value

# Information flow from index to value implicitly
var_without_policy = array[index_with_policy]
```
Can we enforce information control flow differently?
Related Work (1) - Static Information Flow Control

- “Prove” that all the information flow in the program satisfy our information flow assertions/requirements!
  - Compiler
  - Strong type systems: type contains static taint (jif)
  - Static program analysis, theorem proving (nickel)

- May be automatic, user-assisted/interactive, or manual
Related Work (1) - Static Information Flow Control

- “Prove” that all the information flow in the program satisfy our information flow assertions/requirements!
  - Compiler
  - Strong type systems: type contains static taint (jif)
  - Static program analysis, theorem proving (nickel)

- May be automatic, user-assisted/interactive, or manual

UserString s = request.post_content;

String<UserPolicy> s = request.post_content;
Related Work (2) - Dynamic Information Flow Control

- Without modifying the runtime: requires languages with a strong type system (e.g. Haskel)
  - Type contains a dynamic taint (lio)

- Operating System level (HiStar, Dstar)

- Control flow integrity (Microsoft Control Flow Guard)
Related Work (2) - Dynamic Information Flow Control

- Without modifying the runtime: requires languages with a strong type system (e.g. Haskel)
  - Type contains a dynamic taint (lio)

- Operating System level (HiStar, Dstar)

- Control flow integrity (Microsoft Control Flow Guard)

TaintedString s = new TaintedString(request.post_content, userPolicy);
Discussion

1. Would you use Resin in your application?
   a. What requirements would you use to determine if you would use Resin or a similar system?

2. How can Resin/Information Flow Control help us guarantee better privacy (e.g. GDPR compliance)?

3. Can you think of other examples of Information Flow assertions or applications?

4. How would you evaluate a system like Resin? Did the paper have adequate evaluation?

5. What do you think about persistent policies?