Automatic Repair for Input Validation and Sanitization Bugs
Classification of Input Validation and Sanitization Functions

- **Pure Validator**
  - Input: Yes (valid) | No (invalid)

- **Validating Sanitizer**
  - Input: Yes (valid) | Output: No (invalid)

- **Pure Sanitizer**
  - Input: Yes (valid) | Output: No (invalid)
Overview

Sanitizer Function

String Analysis

Symbolic Forward Fix-Point Computation

Symbolic Backward Fix-Point Computation

Post-Image (Post-Condition)

Pre-Image (Pre-Condition)

Negative Pre-Image (Pre-Condition for reject)
Sanitizers

\[ \Sigma^* \cup \{ \bot \} \]

\[ \Sigma = \{ a, b \} \]

\[
\begin{align*}
\text{sanitizer}(x) \{ \\
\text{if (x != "aa" && x != "bb" && x != "ab")} \\
\text{reject;} \\
x = \text{replace}(/^ab$/, "ba", x); \\
\text{return } x; \\
\}\n\end{align*}
\]
Post-Image, Pre-Image and Negative Pre-Image

Pre-image

Negative Pre-image

Possible output (Post Image)

(Non) Preferred Output

Reject

sanitizer(x){
    if (x != "aa" && x != "bb" && x != "ab")
        reject;
    x = replace(/\^ab$/, "ba",x);
    return x;
}

5
Attack Patterns

• For example: for detecting XSS and SQLI

• An attack pattern is negation of a Max policy
  – Attack patterns specify bad strings

• Example:
  – /.*<script.*/ (for XSS)
  – Any string that contains the substring `<script` is bad
XSS Vulnerability Example

1: <?php
2: $www = $_GET["www"];
3: $l_otherinfo = "URL";
4: $www = preg_replace("[^A-Za-z0-9 .-@://]", "", $www);
5: echo "<td> $l_otherinfo : $www </td>";
6: ?>

- `[ ^A-Za-z0-9 .-@:/ ]`
- `. -@` means all characters from . to @
- This includes `<` and `>`
- **XSS Vulnerability**
Vulnerability Detection

\[ \sum^* \]

\[ \sum \]

\[ \bigcap \]

\[ /./*<./*/ \]

\[ = \]
Vulnerability Signature Generation and Vulnerability Repair

Vulnerability Signature
All inputs that can exploit the vulnerability (or an over approximation of that set)

Pre-Image (Bad Input)

Bad Output

Σ*

Σ* U ⊥
Generated Patch

```php
<?php
if(preg_match('/^[^<>]*<.*$/',$_GET['www']))
  $_GET['www'] = preg_replace('<',','',$_GET['www']);

$www = $_GET['www'];
$l_otherinfo = "URL";
$www = preg_replace("^[A-Za-z0-9 .-@://]","",$www);
 echo "<td>" . $l_otherinfo . ": ".$www. "</td>";
?>
```

<table>
<thead>
<tr>
<th>Input</th>
<th>Original Output</th>
<th>New Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foobar</td>
<td>URL: Foobar</td>
<td>URL: Foobar</td>
</tr>
<tr>
<td>Foo&lt;bar</td>
<td>URL: Foo&lt;bar</td>
<td>URL: Foobar</td>
</tr>
<tr>
<td>a&lt;b&lt;c&lt;d</td>
<td>URL: a&lt;b&lt;c&lt;d</td>
<td>URL: abcd</td>
</tr>
</tbody>
</table>

min-cut is {<}
Patches from Vulnerability Signatures

• Ideally, we want to modify the input (as little as possible) so that it does not match the vulnerability signature.

• Given a DFA, an *alphabet cut* is
  – a set of characters that after “removing” the edges that are associated with the characters in the set, the modified DFA does not accept any non-empty string.

• Finding a minimal alphabet cut of a DFA is an NP-hard problem (one can reduce the vertex cover problem to this problem).
  – We use a min-cut algorithm instead.
  – The set of characters that are associated with the edges of the min cut is an alphabet cut.
    • but not necessarily the minimum alphabet cut.
Experiments

• We evaluated our approach on five vulnerabilities from three open source web applications:
  (1) MyEasyMarket-4.1: A shopping cart program
  (2) BloggIT-1.0: A blog engine
  (3) proManager-0.72: A project management system

• We used the following XSS attack pattern:
  $\Sigma^*<\text{script}\Sigma^*$
Forward Analysis Results

- The dependency graphs of these benchmarks are simplified based on the sinks
  - Unrelated parts are removed using slicing

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#nodes</td>
<td>#edges</td>
</tr>
<tr>
<td></td>
<td>#sinks</td>
<td>#inputs</td>
</tr>
<tr>
<td>Time(s)</td>
<td>Mem (kb)</td>
<td>#states/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#bddds</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>2599</td>
<td>23/219</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>13633</td>
<td>48/495</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>1955</td>
<td>125/1200</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>4022</td>
<td>133/1222</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>3387</td>
<td>125/1200</td>
</tr>
</tbody>
</table>
Backward Analysis Results

- We use the backward analysis to generate the vulnerability signatures
  - Backward analysis starts from the vulnerable sinks identified during forward analysis

<table>
<thead>
<tr>
<th>Input</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#nodes</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>
Alphabet Cuts

- We generate cuts from the vulnerability signatures using a min-cut algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>#nodes</td>
<td>#edges</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

- **Problem**: When there are two user inputs the patch will block everything and delete everything
  - Overlooks the relations among input variables (e.g., the concatenation of two inputs contains < SCRIPT)
Relational Vulnerability Signature

• Perform forward analysis using multi-track automata to generate relational vulnerability signatures

• Each track represents one user input
  – An auxiliary track represents the values of the current node
  – We intersect the auxiliary track with the attack pattern upon termination
Relational Vulnerability Signature

• Consider a simple example having multiple user inputs

```php
1: $www = $_GET["www"];
2: $url = $_GET["url"];
3: echo $url. $www;
?>
```

• Let the attack pattern be $\Sigma^* < \Sigma^*$
Relational Vulnerability Signature

- A multi-track automaton: ($url, $www, aux)
- Identifies the fact that the concatenation of two inputs contains <
Relational Vulnerability Signature

- Project away the auxiliary variable
- Find the min-cut
- This min-cut identifies the alphabet cuts {<} for the first track ($url$) and {<} for the second track ($www$)

```
min-cut is {<},{<}
```
Patch for Multiple Inputs

- Patch: If the inputs match the signature, delete its alphabet cut

```php
<?php
if (preg_match('/[^ <>]*<.*/', $GET['url'].$GET['www'])) {
    $GET['url'] = preg_replace('<', '', $GET['url']);
    $GET['www'] = preg_replace('<', '', $GET['www']);
}

1: $www = $GET['www'];
2: $url = $GET['url'];
3: echo $url. $www;
?>
```
Differential Analysis and Repair

Target Sanitizer

Reference Sanitizer

String Analysis

? =

Generate Patch

Yes

No
Why Differential?

Web application (client side)

Request
http://site.com/unsubscribe.php?email=john.doe@mail.com

Web application (server side)

Web server

unsupcribe.php

Internet

DB

Java servlet
unsubscribe.jsp

public class FieldChecks {
... public boolean validateRequired(Object bean, Field field, ..) {
    String value = evaluateBean(bean, field);
    if ((value == null) || (value.trim().length() == 0)) {
        return false;
    }
else {
        return true;
    }
}
...}
A Javascript/Java Input Validation Function

function validateEmail(form) {
  var emailStr = form["email"].value;
  if(emailStr.length == 0) {
    return true;
  }
  var r1 = new RegExp("( )|(@\.*@)|(@\.)");
  var r2 = new RegExp("^\[\w\]+@([\w]+\.[\w]{2,4})$" );
  if(!r1.test(emailStr) && r2.test(emailStr)) {
    return true;
  }
  return false;
}

public boolean validateEmail(Object bean, Field f, ..) {
  String val = ValidatorUtils.getValueAsString(bean, f);
  Perl5Util u = new Perl5Util();
  if ((val == null || val.trim().length == 0)) {
    if ((!u.match("/( )|(@\.*@)|(@\.)/", val)) &&
      u.match("/^[\w]+@([\w]+\.[\w]{2,4})$/", val)){
      return true;
    } else {
      return false;
    }
  }
  return true;
}
1\textsuperscript{st} Step: Find Inconsistency

\begin{align*}
\Sigma^* & \rightarrow \Sigma^* \cup \bot \\
\emptyset & \rightarrow \emptyset \\
\emptyset & \rightarrow \emptyset
\end{align*}

Output difference: Strings returned by target but not by reference
Differential Analysis Evaluation

• Analyzed a number of Java EE web applications
  – Only looking for differences (inconsistencies)

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>JGOSSIP</td>
<td><a href="http://sourceforge.net/projects/jgossipforum/">http://sourceforge.net/projects/jgossipforum/</a></td>
</tr>
<tr>
<td>VEHICLE</td>
<td><a href="http://code.google.com/p/vehiclemanage/">http://code.google.com/p/vehiclemanage/</a></td>
</tr>
<tr>
<td>MEODIST</td>
<td><a href="http://code.google.com/p/meodist/">http://code.google.com/p/meodist/</a></td>
</tr>
<tr>
<td>MYALUMNI</td>
<td><a href="http://code.google.com/p/myalumni/">http://code.google.com/p/myalumni/</a></td>
</tr>
<tr>
<td>CONSUMER</td>
<td><a href="http://code.google.com/p/consumerbasedenforcement">http://code.google.com/p/consumerbasedenforcement</a></td>
</tr>
<tr>
<td>TUDU</td>
<td><a href="http://www.julien-dubois.com/tudu-lists">http://www.julien-dubois.com/tudu-lists</a></td>
</tr>
<tr>
<td>JCRBIB</td>
<td><a href="http://code.google.com/p/jcrbib/">http://code.google.com/p/jcrbib/</a></td>
</tr>
</tbody>
</table>
Analysis Phase Time Performance & Inconsistencies That We Found

<table>
<thead>
<tr>
<th>Subject</th>
<th>Time (s)</th>
<th>$A_{C-S}$</th>
<th>$A_{S-C}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>JGossip</td>
<td>3.2</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Vehicle</td>
<td>1.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MeoDist</td>
<td>1.7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MyAlumni</td>
<td>2.9</td>
<td>141</td>
<td>0</td>
</tr>
<tr>
<td>Consumer</td>
<td>1.0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Tudu</td>
<td>0.6</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>JcrBib</td>
<td>1.2</td>
<td>45</td>
<td>0</td>
</tr>
</tbody>
</table>
## Analysis Phase Memory Usage

<table>
<thead>
<tr>
<th>Subject</th>
<th>Client-Side DFA</th>
<th>Server-Side DFA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avr size (mb)</td>
<td>Min</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>B</td>
</tr>
<tr>
<td>JGOSSIP</td>
<td>6.0</td>
<td>4</td>
</tr>
<tr>
<td>VEHICLE</td>
<td>4.8</td>
<td>4</td>
</tr>
<tr>
<td>MEODIST</td>
<td>5.7</td>
<td>5</td>
</tr>
<tr>
<td>MYALUMNI</td>
<td>3.2</td>
<td>4</td>
</tr>
<tr>
<td>CONSUMER</td>
<td>5.3</td>
<td>4</td>
</tr>
<tr>
<td>TUDU</td>
<td>6.1</td>
<td>4</td>
</tr>
<tr>
<td>JCRBIB</td>
<td>5.4</td>
<td>4</td>
</tr>
</tbody>
</table>
2\textsuperscript{nd} Step: Differential Repair

\[ \Sigma^* \quad \Sigma^* \cup \bot \]

Target

Reference

\[ \Sigma^* \quad \Sigma^* \cup \bot \]

\[ \Sigma^* \quad \Sigma^* \cup \bot \]

\[ \Sigma^* \quad \Sigma^* \cup \bot \]

Repaired Function

\[ \Sigma^* \quad \Sigma^* \cup \bot \]
Composing Sanitizers?

• Can we run the two sanitizers one after the other?

• Does not work due to lack of **Idempotency**
  – Both sanitizers escape ’ with \\
  – Input ab’c
  – 1\textsuperscript{st} sanitizer $\rightarrow$ ab’c
  – 2\textsuperscript{nd} sanitizer $\rightarrow$ ab\’c
    • Security problem (double escaping)

• We need to find the difference
How to repair?
function target($x) {
    $x = preg_replace("'\", "\\", $x);
    return $x;
}

function reference($x) {
    $x = preg_replace("<", "", $x);
    if (strlen($x) < 4)
        return $x;
    else
        die("error");
}

Output difference:
Strings returned by target
but not by reference
Set of input strings that resulted in the difference

<table>
<thead>
<tr>
<th>Input</th>
<th>Target</th>
<th>Reference</th>
<th>Diff Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&lt;&quot;</td>
<td>&quot;&lt;&quot;</td>
<td>&quot;&quot;</td>
<td>Sanitization</td>
</tr>
<tr>
<td>&quot;'''&quot;</td>
<td>&quot;'''&quot;</td>
<td>&quot;'''&quot;</td>
<td>Sanitization + Length</td>
</tr>
<tr>
<td>&quot;abcd&quot;</td>
<td>&quot;abcd&quot;</td>
<td>&quot;abcd&quot;</td>
<td>Validation</td>
</tr>
</tbody>
</table>
- Minicut results in deleting everything
  - "foo" → ""

- Why?
  - You can not remove a validation difference using a sanitization patch

function target($x) {
    $x = str_replace('"', '\', $x);
    return $x;
}

function reference($x) {
    $x = str_replace('<', '', $x);
    if (strlen($x) < 4)
        return $x;
    else
        die("error");
}
(1) Validation Patch

```
function valid_patch($x){
    if (stranger_match1($x))
        die(“error”);
}
```

```
function target($x){
    $x = str_replace(“\”, “\”, $x);
    return $x;
}
```

```
function reference($x){
    $x = str_replace(“<”, “\”, $x);
    if (strlen($x) < 4)
        return $x;
    else
        die(“error”);
}
```
MinCut = \{‘,’,<\}\n
“fo’” \rightarrow “fo\’’
function valid_patch($x){
    if (stranger_match1($x))
        die(“error”);
}

function valid_patch($x){
    if (stranger_match1($x))
        die(“error”);
}

function target($x){
    $x = str_replace(“‘”, “\‘”, $x);
    return $x;
}

function reference($x){
    $x = str_replace(“<”, “\”, $x);
    if (strlen($x) < 4)
        return $x;
    else
        die(“error”);
}

Post-image_R = \{a, foo, baar\}
Len = \Sigma^1 \cup \Sigma^3 \cup \Sigma^4
Post-image_T = \{bb, car\}
Diff = \{bb\}
### (3) Sanitization Patch

**Function Definitions**

- **valid_patch($x)**: Checks if the input string is a valid patch. Returns the string if it is valid, otherwise prints an error message.
- **length_patch($x)**: Sanitizes the input string by replacing backslashes and single quotes. Returns the sanitized string.
- **target($x)**: Sanitizes the input string by replacing backslashes and single quotes. Returns the sanitized string.
- **reference($x)**: Sanitizes the input string by replacing backslashes and single quotes. If the string has a length less than 4, returns the string; otherwise, prints an error message.

**Diagram Description**

- The diagram illustrates the sanitization process for a DFA (Deterministic Finite Automaton) with restricted lengths.
- The input string is processed through the `target` function, which replaces backslashes and single quotes.
- The length of the resulting string is compared against a threshold.
- If the length exceeds the threshold, an error is returned.

**Key Points**

- **Unwanted length in target caused by escape**
- **Target Restricted Length**
- **Length of Reference DFA**
- **Sanitization Affected Length**
- **Reference Post-image**
function **valid_patch**($x){
    if (stranger_match1($x))
        die(“error”);
}

function **length_patch**($x){
    if (strlen($x) < 4)
        return $x;
    else
        die(“error”);
}

function **sanitize_patch**($x){
    $x = str_replace(“<”, “”, $x);
    return $x;
}

function **target**($x){
    $x = str_replace(“\”, “\\”, $x);
    return $x;
}

### (3) Sanitization Patch

function **reference**($x){
    $x = str_replace(“<”, “”, $x);
    if (strlen($x) < 4)
        return $x;
    else
        die(“error”);
}

MinCut = {<}
MinCut Heuristics

• We use two heuristics for mincut

• Trim:
  – Only if mincut contain space character
  – Test if reference Post-Image is does not have space at the beginning and end
  – Assume it is `trim()`

• Escape:
  – Test if reference Post-Image escapes the mincut characters
Differential Repair Evaluation

- We ran the differential patching algorithm on 5 PHP web applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHPNews v1.3.0</td>
<td>News publishing software</td>
</tr>
<tr>
<td>UseBB v1.0.16</td>
<td>forum software</td>
</tr>
<tr>
<td>Snipe Gallery v3.1.5</td>
<td>Image management system</td>
</tr>
<tr>
<td>MyBloggie v2.1.6</td>
<td>Weblog system</td>
</tr>
<tr>
<td>Schoolmate v1.5.4</td>
<td>School administration software</td>
</tr>
</tbody>
</table>
## Number of Patches Generated

<table>
<thead>
<tr>
<th>Mapping</th>
<th># Pairs</th>
<th># Valid.</th>
<th># Length</th>
<th># Sanit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Server</td>
<td>122</td>
<td>61</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Server-Client</td>
<td>122</td>
<td>53</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Server-Server</td>
<td>206</td>
<td>49</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Client-Client</td>
<td>19</td>
<td>34</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
Sanitization Patch Results

<table>
<thead>
<tr>
<th>Mapping</th>
<th>mincut Avr. size</th>
<th>mincut Max size</th>
<th>#trim</th>
<th>#escape</th>
<th>#delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-Client</td>
<td>4</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Server-Server</td>
<td>3</td>
<td>5</td>
<td>23</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Client-Client</td>
<td>7</td>
<td>15 (circled)</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
## Time and Memory Performance of Differential Repair Algorithm

<table>
<thead>
<tr>
<th>Repair phase</th>
<th>DFA size (#bddnodes)</th>
<th>peak DFA size (#bddnodes)</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>Valid.</td>
<td>997</td>
<td>32,650</td>
<td>484</td>
</tr>
<tr>
<td>Length</td>
<td>129,606</td>
<td>347,619</td>
<td>245,367</td>
</tr>
<tr>
<td>Sanit.</td>
<td>2,602</td>
<td>11,951</td>
<td>4,822</td>
</tr>
</tbody>
</table>