ViewPoints: Differential String Analysis for Discovering Client- and Server-Side Input Validation Inconsistencies

Muath Alkhalaf\textsuperscript{1} \hspace{1cm} Shauvik Roy Choudhary\textsuperscript{2}
Mattia Fazzini\textsuperscript{2} \hspace{1cm} Tevfik Bultan\textsuperscript{1}
Alessandro Orso\textsuperscript{2} \hspace{1cm} Christopher Kruegel\textsuperscript{1}

\textsuperscript{1}UC Santa Barbara \hspace{2cm} \textsuperscript{2}Georgia Tech
Web Software is Becoming Increasingly Dominant

- Web applications are used extensively in many areas:
  - eBay
  - Amazon
  - YouTube
  - Apple
  - Facebook
  - Citibank
  - CNN
  - Twitter
  - Groupon
  - Google
  - HealthVault
  - Microsoft
  - Google Health
  - Google App Engine
  - Google Docs
  - Microsoft Office

- We will rely on web applications more in the future:

- Web software is also rapidly replacing desktop applications
Web applications are not trustworthy

**Web Application Vulnerabilities**

as a Percentage of All Disclosures in 2010

- **Web Applications:** 49 percent
- **Others:** 51 percent

**Web Application Vulnerabilities**

as a Percentage of All Disclosures in 2011

- **Web Applications:** 41 percent
- **Others:** 59 percent

IBM X-force report
Why Do We Need Input Validation?

- The user input comes in string form and must be **validated** before it can be used
  - Input validation uses **string manipulation** which is error prone

- We need to verify input validation to assure:
  - Correctness
  - Security
  - Consistency
Three tier architecture

- Web applications use the 3-tier architecture

- Most web applications check the inputs both on the client side and the server-side
  - This redundancy is necessary for security reasons (client-side checks can be circumvented by malicious users)
  - Not having client-side input validation results in unnecessary communication with the server, degrading the responsiveness and performance of the application
Client-Side is popular

- Size of Client Side code is growing rapidly
- Over 90% of web sites use javascript  
  
  *Source: According to an IBM study performed in 2010 - Salvatore Guarnieri*
Input validation functions

Input Validation Function

- True (valid)
- False (invalid)
function validateEmail(form) {
    var emailStr = form["email"].value;
    if(emailStr.length == 0) {
        return true;
    }
    var r1 = new RegExp("(^\s+|\(@.*@\)|(@\s+))"),
    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;
}
Under Constrained Validation Function

A function that accepts some bad input values

Good input

Bad input

True (valid)

False (Invalid)
Over Constrained Validation Function

A function that rejects some good input values

Good input

Bad input

True (valid)

False (Invalid)
How can we check the validation functions?

One approach that has been used in the past:
- Specify the input validation policy as a regular expression (attack patterns, max & min policies) and then use string analysis to check that validation functions conform to the given policy.

Someone has to manually write the input validation policies
- If the input validation policies are specific for each web application, then the developers have to write different policies for each application, which could be error prone.
The approach we present in this paper does not require developers to write specific policies.

Basic idea: Use the inherent redundancy in input validation to check the correctness of the input validation functions.
Motivating Scenario

Web application (client side)

Unsubscribe
Email: [input]
Submit

Internet

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

Web application (server side)

Confirmation Page
Congratulations!
Your account has been unsubscribed.

HTML page
Client Accepts – Server Rejects

Web application (client side)

Unsubscribe
Email: 
Submit

HTML page

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

Internet

Web application (server side)

Web server

ERROR
Reject

Java servlet unsubscribe.jsp

Web application (server side)

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;
        }
    }
    ...
}
Two problems may occur:

- Either the client side input validation function was under constrained and accepted **bad inputs**
- Or the server side input validation function was over constrained and rejected some **good input**
Client Rejects

Web application (client side)

Unsubscribe
Email: [input]
Submit

Reject

Internet

Web server

Web application (server side)
- A problem may occur:
  - the client side input validation function was over constrained and rejected some good input
- What happens when Input value is bad and the server accepts this value?
Client Rejects – Server Accepts

Web application (client side)

Request
http://site.com/unsubscribe.jsp?email= ...<script...>...

Web server

Unsubscribe
Email: 
Submit

Internet

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;
}
...}

Java servlet unsubscribe.jsp

Web application (server side)

Attack
The server side input validation function was under constrained and accepted bad inputs

Serious security problem
**Approach**

Web application

<table>
<thead>
<tr>
<th>JS</th>
<th>Client side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Server side</td>
</tr>
</tbody>
</table>

**Task 1:** Input validation mapping and extraction

Input validation operations

**Task 2:** Input validation modeling using DFAs

**Task 3:** Inconsistency identification and reporting

Counter example
Mapping & Extraction Phase

Task 1: Input validation mapping and extraction

Task 2: Input validation modeling using DFAs

Task 3: Inconsistency identification and reporting

Input validation DFAs

Counter example
For each input, we obtain
- Domain information
- Multiple parameterized validation functions with parameter values
- Path to access the web application form
Dynamic Extraction for Javascript

- Why extraction
  - Lots of event handling, error reporting and rendering code

- Why dynamic?
  - Javascript is very dynamic
  - Object oriented
  - Prototype inheritance
  - Closures
  - Dynamically typed
  - eval
- Number of valid inputs
  - Inputs are selected heuristically

- Instrument execution
  - **HtmlUnit**: browser simulator
  - **Rhino**: JS interpreter

- Convert all accesses on objects and arrays to accesses on memory locations
Static Extraction for Java

- **Transformations**
  - Library call and parameter inlining
  - Framework specific modeling and transformation
  - Constant propagation and Dead code elimination

- **Slicing (PDG based)**
  - Forward slicing on input parameter
  - Backward slicing for the true path

Input validation routines

Parsing and CFG Construction
(uses Soot)

Control flow graph

Transformations and Slicing

Static Slice
Input Validation Modeling

Task 1: Input validation mapping and extraction

Task 2: Input validation modeling using DFAs

Task 3: Inconsistency identification and reporting

Input validation DFAs

Web application

JS
Client side

Java
Server side
Compute two automata for each input field:
- Client-Side DFA $A_c$
  - $L(A_c)$ Over approximation of set of values accepted by client-side input validation function
- Server-Side DFA $A_s$
  - $L(A_s)$ Over approximation of set of values accepted by server-side input validation function

We use automata based static string analysis to compute $L(A_c)$ and $L(A_s)$
Static String Analysis

- Static string analysis determines all possible values that a string expression can take during any program execution

- We use automata based string analysis
  - Associate each string expression in the program with an automaton
  - The automaton accepts an over approximation of all possible values that the string expression can take during program execution

- We built our javascript string analysis on Closure compiler from Google and java string analysis on Soot

- Flow sensitive, intraprocedural and path sensitive
Symbolic Automata

Explicit DFA representation

Symbolic DFA representation
We use an automata based widening operation to over-approximate the fixpoint.
- Widening operation over-approximates the union operations and accelerates the convergence of the fixpoint computation.

Due to loops we need fixpoint computation.

Lattice with infinite height
Modeling String Operations

- **Modeling string operations**
  - **CONCATENATION**
    - $y = x + "b"$
  
  - **REPLACEMENT**
    - Language based replacement
    - `replace(x, "a", "d")`
  
  - **RESTRICTION**
    - If $(x = "a")$
      - ...
var emailStr = form["email"].value;

if (emailStr.length == 0)
    return true;
else
    if (!r1.test(emailStr) || r2.test(emailStr))
        return true;
    else
        return false;

$L(A_c) = (\Sigma^\ast\|(\ )|(\@.\@)|(\@\.|\@))|(\^[\w]+@([\w]+\.[\w]{2,4})\$)
String Analysis Example
Server-Side

String val = ValidatorUtils.getValueAsString(bean, f);

if (Pred \equiv var.length == intlit)
    return Σ^{intlit};

if (Pred \equiv regexp.match(var))
    if (checkregexp(regexp)=partialmatch)
        return CONCAT(CONCAT(Σ*, L(regexp)), Σ*);
    else
        return L(regexp);

if (Pred \equiv var.trim().length == 0)
    return true

if (u.match("/( )|(\.*@)|(@\:\//", val)) &&
    u.match("/^[\\w]+@([\\w]+\.\\w{2,4})$", val))
    return true

return false

L(A_s) = ([^ ]+((( )|(\.*@)|(\@\:)))|([^\w]+@([\w]+\\.\\w{2,4}$)))
Inconsistency Identification

Task 1: Input validation mapping and extraction

Task 2: Input validation modeling using DFAs

Task 3: Inconsistency identification and reporting

Counter example
Computing Difference Signature

- Compute two difference signatures:
  - \( L(A_{s-c}) = L(A_s) \setminus L(A_c) \)
  - \( L(A_{c-s}) = L(A_c) \setminus L(A_s) \)

- If \( L(A_{s-c}) \neq \emptyset \)

- If \( L(A_{c-s}) \neq \emptyset \)
Five possible relationships between $L(A_c)$ and $L(A_s)$:

1. $L(A_c) = L(A_s)$
2. $L(A_c) \cap L(A_s) \neq \emptyset$
3. $L(A_c) \cap L(A_s) = \emptyset$
Server-Client Difference Signature

We compute \( L(A_{s-c}) \)

\[ L(A_{s-c}) = \emptyset \]

\[ L(A_{s-c}) \neq \emptyset \]
We compute $L(A_{c-s})$

$L(A_{c-s}) = \emptyset$ ✅

$L(A_{c-s}) \neq \emptyset$ ❌
Computing Inconsistencies for the Two Example Functions

- Compute two difference signatures:
  - $L(A_{c-s}) = L(A_c) \setminus L(A_s) = \emptyset$
  - $L(A_{s-c}) = L(A_s) \setminus L(A_c)$

$L(A_s) = ([^\^]+|((\.)|(@.*@)|(\.|\|)))(^[w]+@([w]+\.[w]{2,4}$))

$L(A_c) = (\Sigma^*|(\.)|(@.*@)|(\.|\|)))(^[w]+@([w]+\.[w]{2,4}$))

$L(A_{s-c}) = [\ ]^+$

Counter Example = “ “
Evaluation

- Analyzed a number of Java EE web applications

<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>
## Extraction Phase Performance

<table>
<thead>
<tr>
<th>Subject</th>
<th>Frm</th>
<th>Inputs</th>
<th>VI_C</th>
<th>ET_C(s)</th>
<th>VI_S</th>
<th>ET_S(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JGossip</td>
<td>25</td>
<td>83</td>
<td>74</td>
<td>329.8</td>
<td>83</td>
<td>4.38</td>
</tr>
<tr>
<td>Vehicle</td>
<td>17</td>
<td>41</td>
<td>41</td>
<td>155.5</td>
<td>41</td>
<td>2.04</td>
</tr>
<tr>
<td>MeoDist</td>
<td>18</td>
<td>62</td>
<td>62</td>
<td>192.2</td>
<td>62</td>
<td>1.93</td>
</tr>
<tr>
<td>MyAlumni</td>
<td>46</td>
<td>141</td>
<td>0</td>
<td>0</td>
<td>141</td>
<td>4.28</td>
</tr>
<tr>
<td>Consumer</td>
<td>3</td>
<td>21</td>
<td>14</td>
<td>68.4</td>
<td>21</td>
<td>1.1</td>
</tr>
<tr>
<td>Tudu</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0.78</td>
</tr>
<tr>
<td>JcrBib</td>
<td>21</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>45</td>
<td>1.51</td>
</tr>
</tbody>
</table>
## Analysis Phase Memory Performance

<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>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>
## Analysis Phase Time Performance & Inconsistencies That We Found

<table>
<thead>
<tr>
<th>Subject</th>
<th>Time (s)</th>
<th>AC-S</th>
<th>AS-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>
Related Work

- **String Analysis**
  - String analysis based on context free grammars: [Christensen et al., SAS’03] [Minamide, WWW’05]
  - Application of string analysis to web applications: [Wassermann and Su, PLDI’07, ICSE’08] [Halfond and Orso, ASE’05, ICSE’06]
  - Automata based string analysis: [Xiang et al., COMPSAC’07] [Shannon et al., MUTATION’07]

- **Input Validation Verification**
  - FLAX [ P. Saxena et al., NDSS’10 ]
  - Kudzu [ P. Saxena et al., SSP’10 ]
  - NoTamper [ P. Bisht et al., CCS’10 ]
  - WAPTEC [ P. Bisht et al., CCS’11 ]
  - [ M. Alkhalaf et al., ICSE’12 ]
Questions
Web applications are not trustworthy

Extensive string manipulation:

- Web applications use extensive string manipulation
  - To construct html pages, to construct database queries in SQL, to construct system commands, etc.

- The user input comes in string form and must be validated before it can be used

- String manipulation is error prone