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

Muath Alkhalaf[1] Shauvik Roy Choudhary[2]
Mattia Fazzini[2] Tevfik Bultan[1]
Alessandro Orso[2] Christopher Kruegel[1]

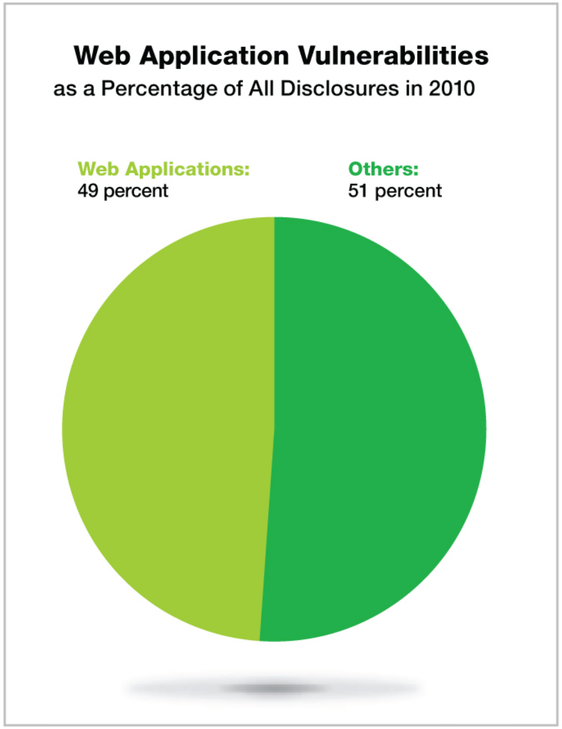
[1]UC Santa Barbara [2]Georgia Tech

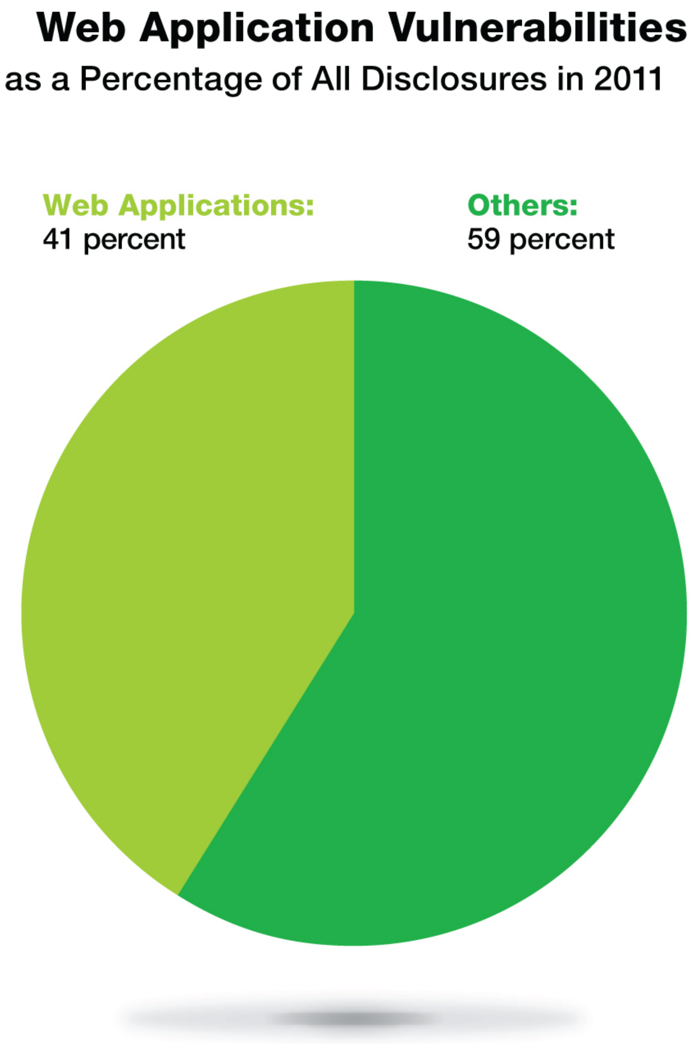
Web Software is Becoming Increasingly Dominant

- Web applications are used extensively in many areas:
- We will rely on web applications more in the future:

- Web software is also rapidly replacing desktop applications

Web applications are not trustworthy





IBM X-force report

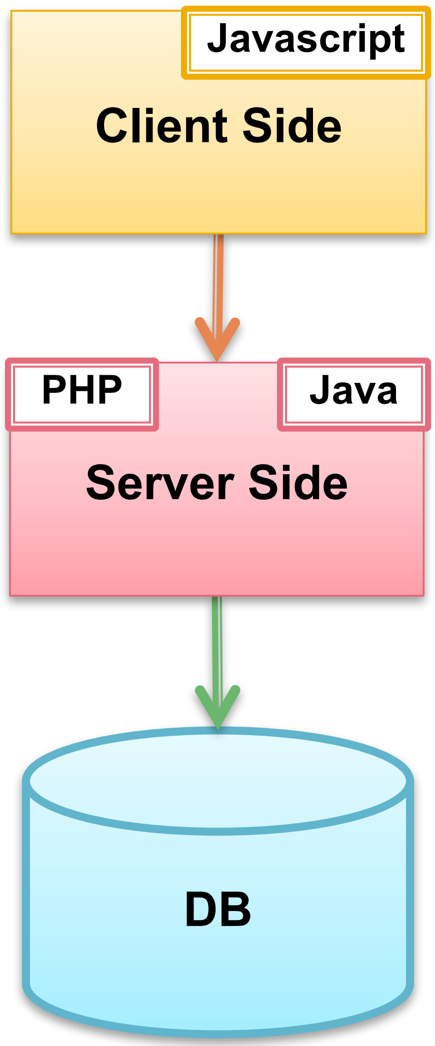
Why Do We Need Input Validation?

- 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

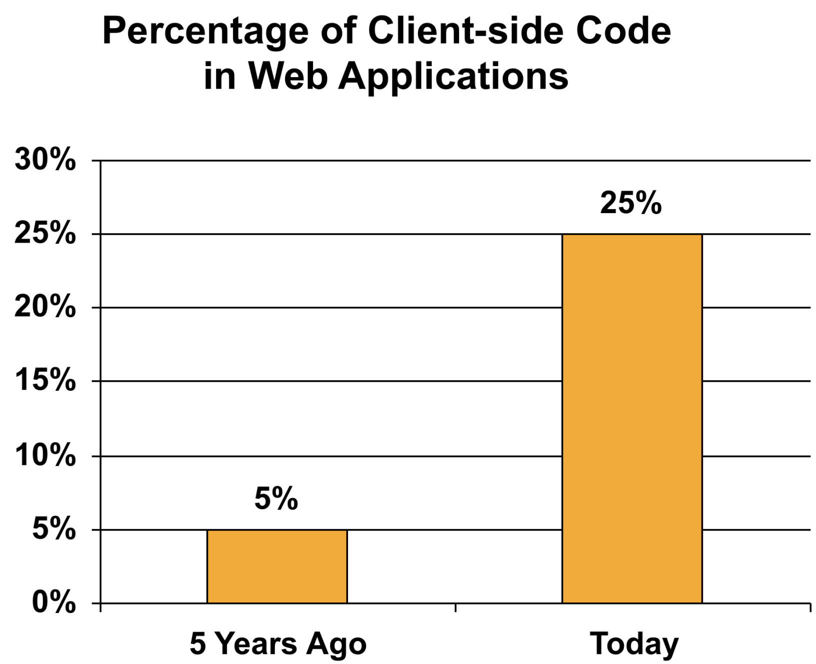
Three tier architecture

- 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



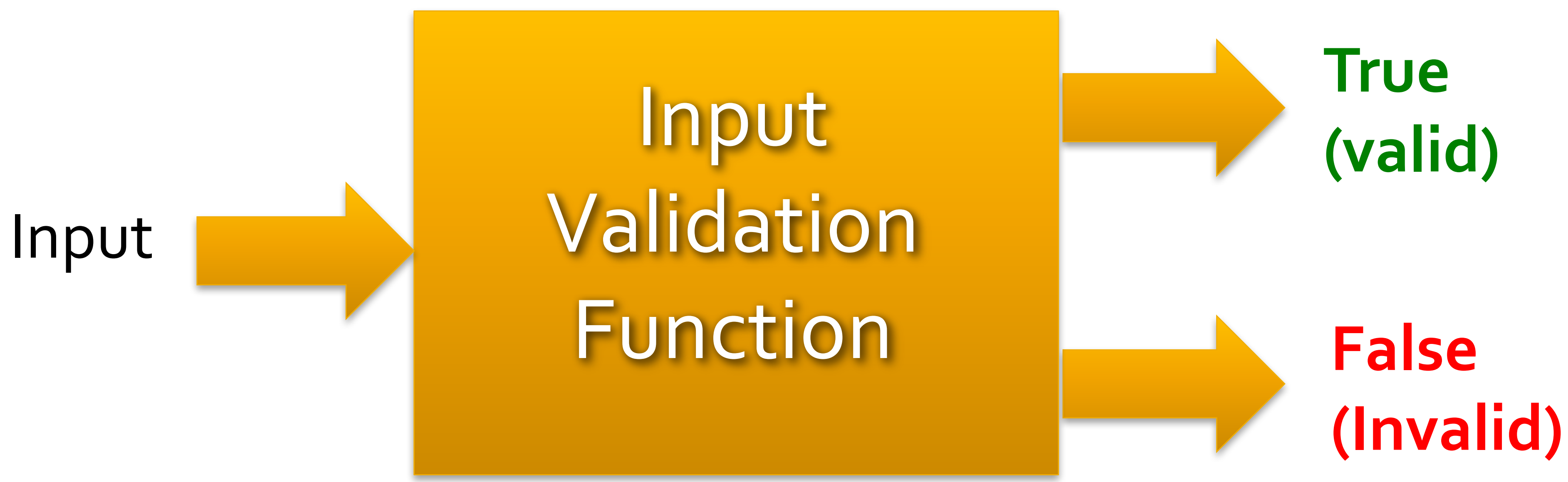
Client-Side is popular

- Size of Client Side code is growing rapidly
- Over 90% of web sites use javascript *source: W3Techs*



Source: According to an IBM study performed in 2010 - Salvatore Guarnieri

Input validation functions



A Javascript Input Validation Function

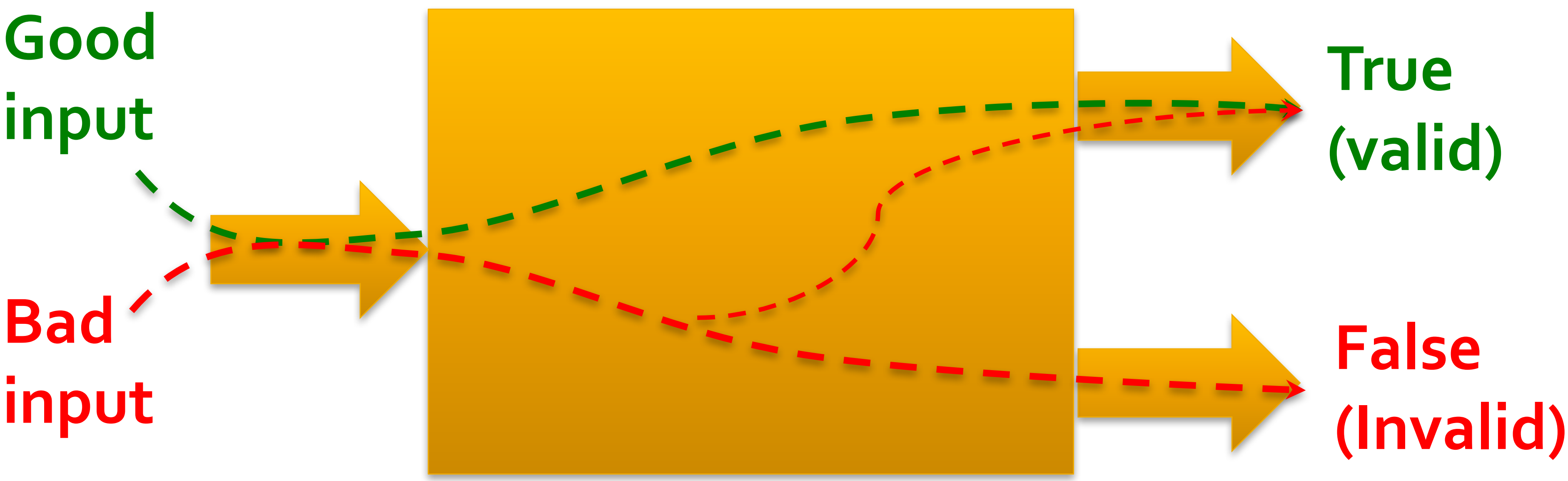
```
function validateEmail(form) {
   var emailStr = form["email"].value;
   if(emailStr.length == 0) {
      return true;
   }
   var r1 = new RegExp("( )|(@.*@)|(@\\.)");
   var r2 = new RegExp("^[\\w]+@([\\w]+\\.
        [\\w]{2,4})$");

   if(!r1.test(emailStr) &&
        r2.test(emailStr)) {
           return true;
   }
   return false;
}
```

A Java Input Validation Function

```
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;
}
```

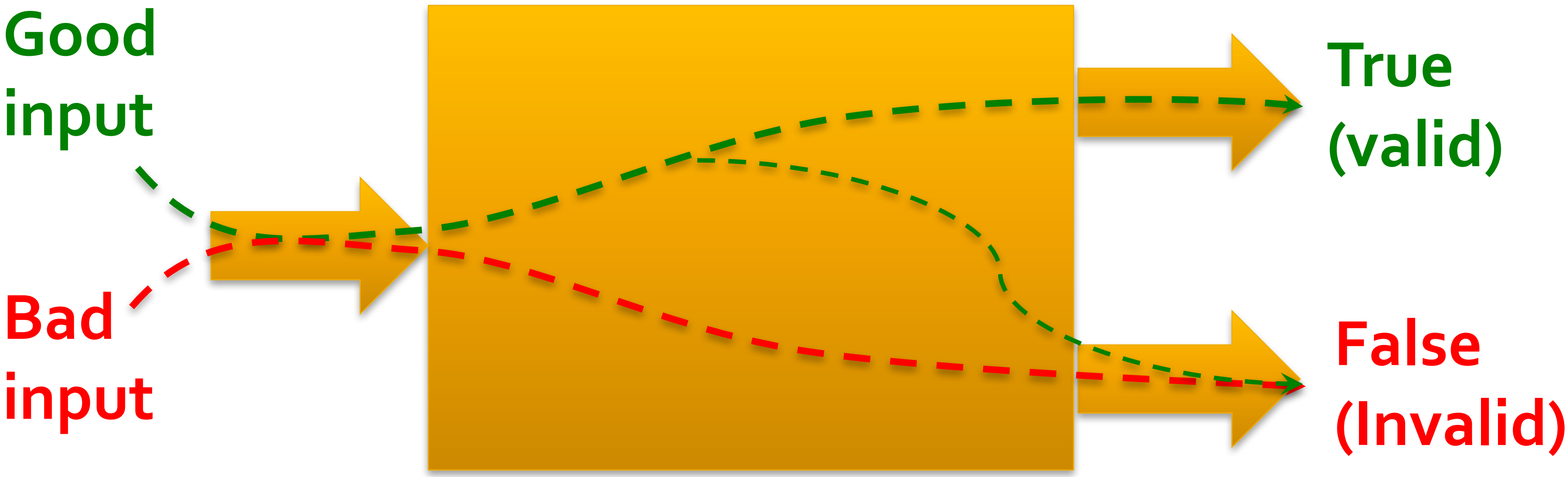
Under Constrained Validation Function

A function that accepts some bad input values



Over Constrained Validation Function

A function that rejects some good input values



How to Check Validation Function Correctness?

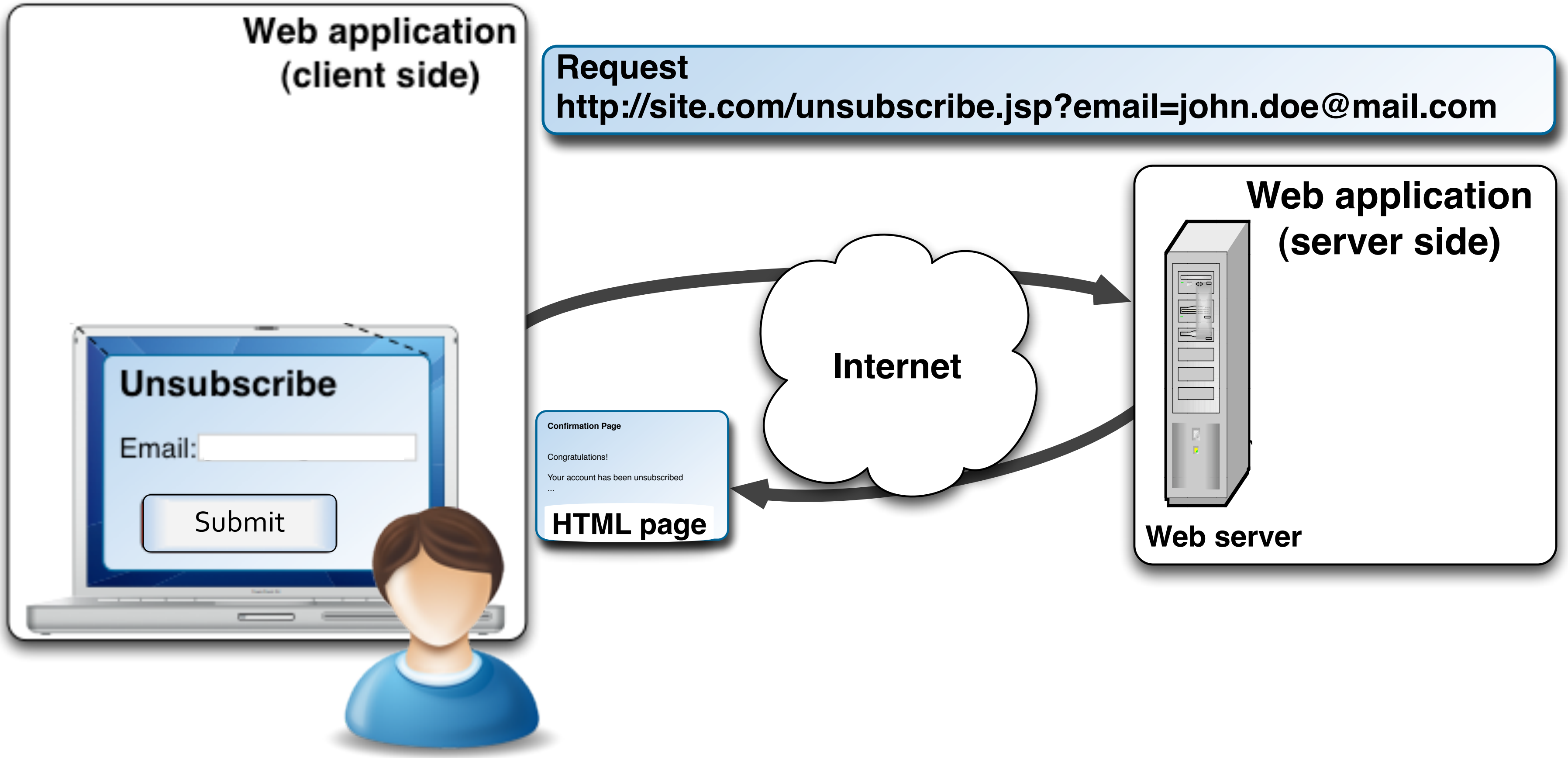
- How can we check the validation functions?
- One approach that has been used in the past:
  - Specify the input validation policy as a regular expression (attack patterns, max & min policies) and then use string analysis to check that validation functions conform to the given policy.
- Someone has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

Differential Analysis

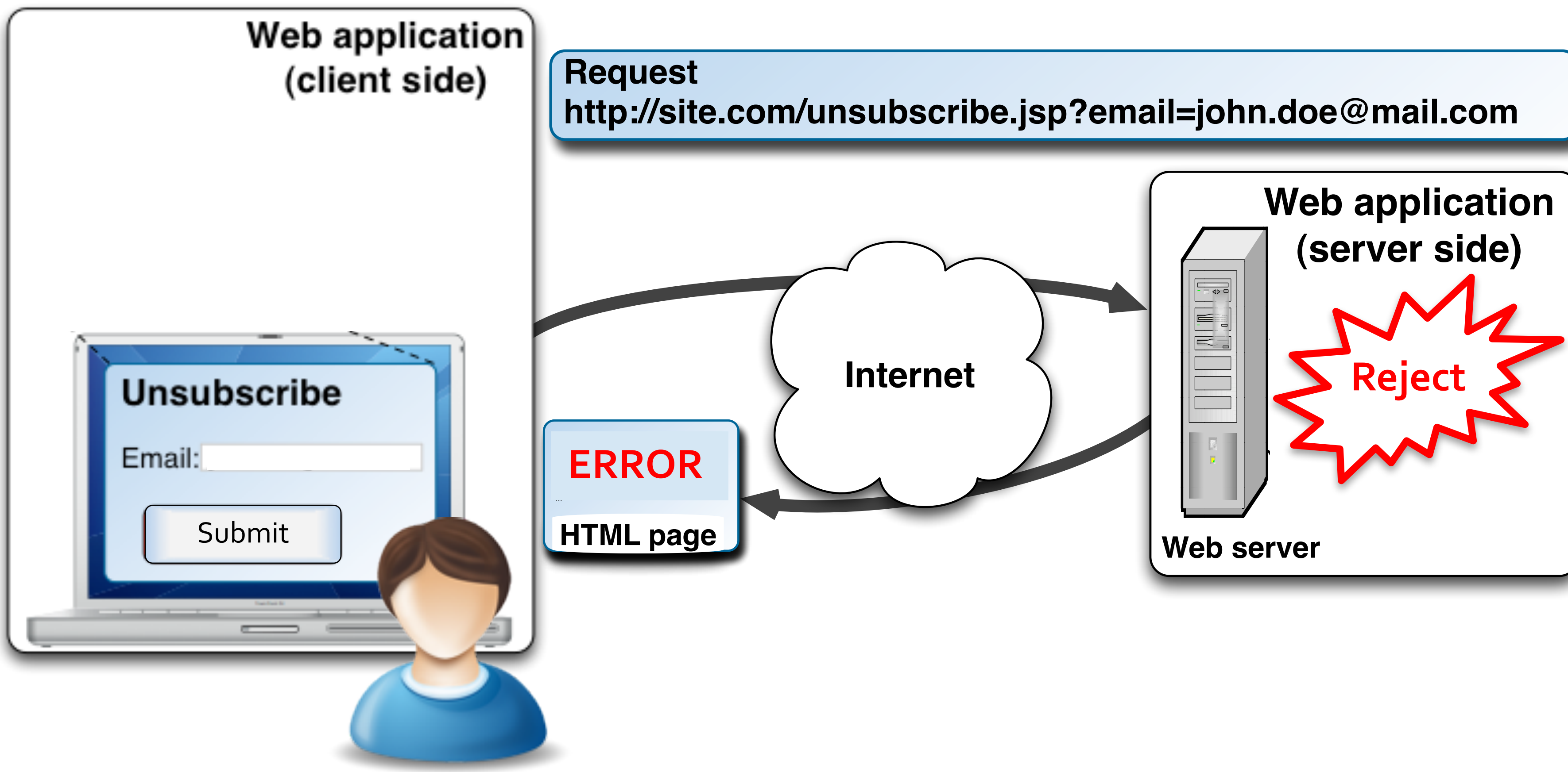
- The approach we present in this paper **does not require developers to write specific policies**

- **Basic idea**: Use the *inherent redundancy in input validation* to check the correctness of the input validation functions

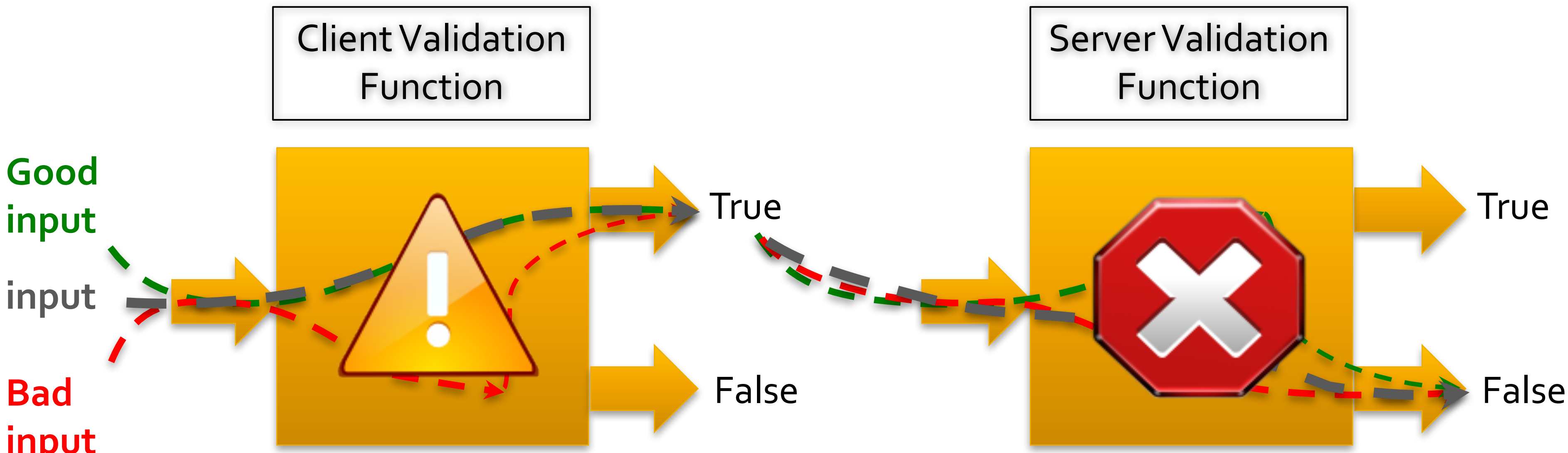
Motivating Scenario



Client Accepts – Server Rejects

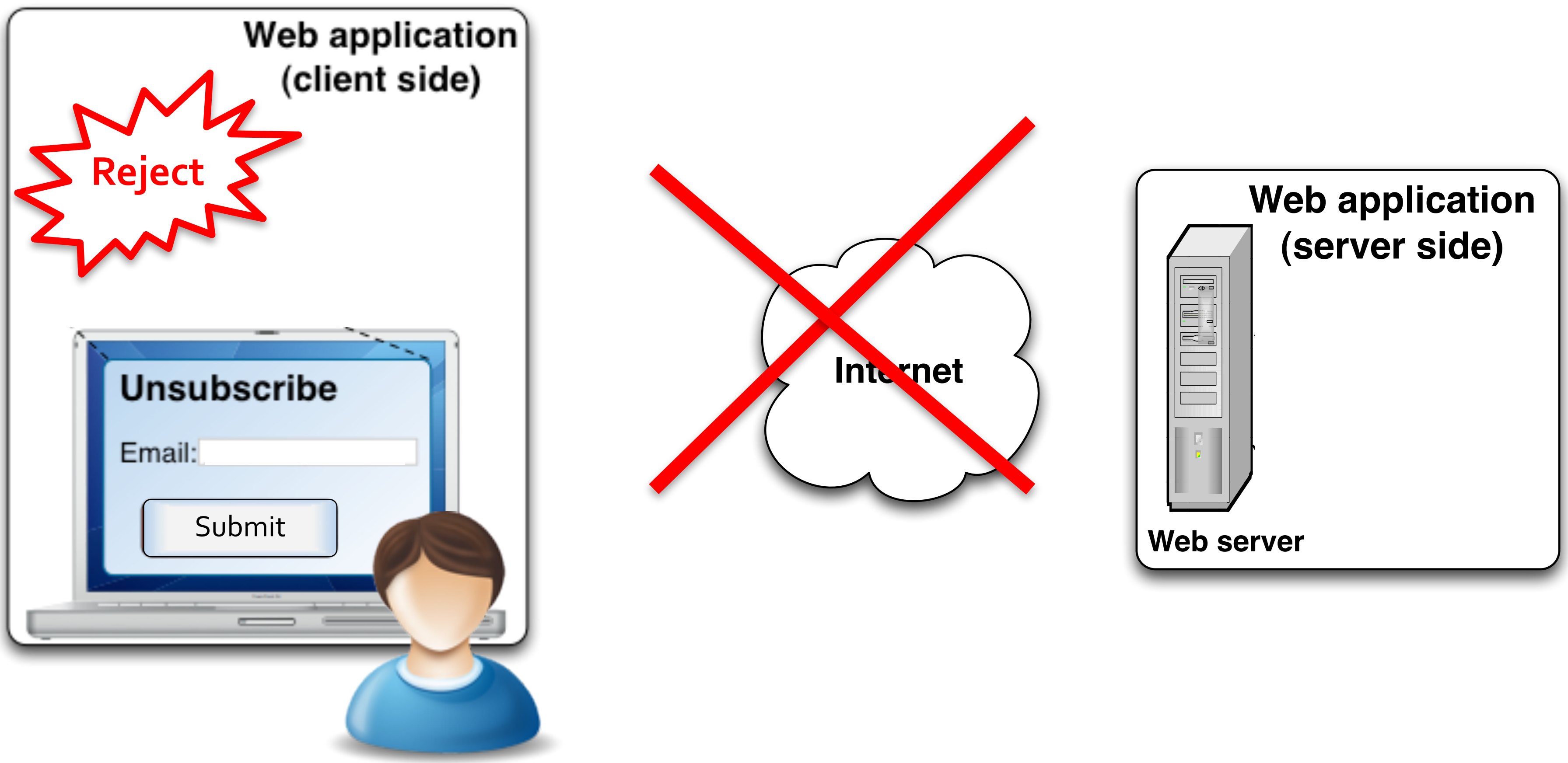


Client Accepts – Server Rejects

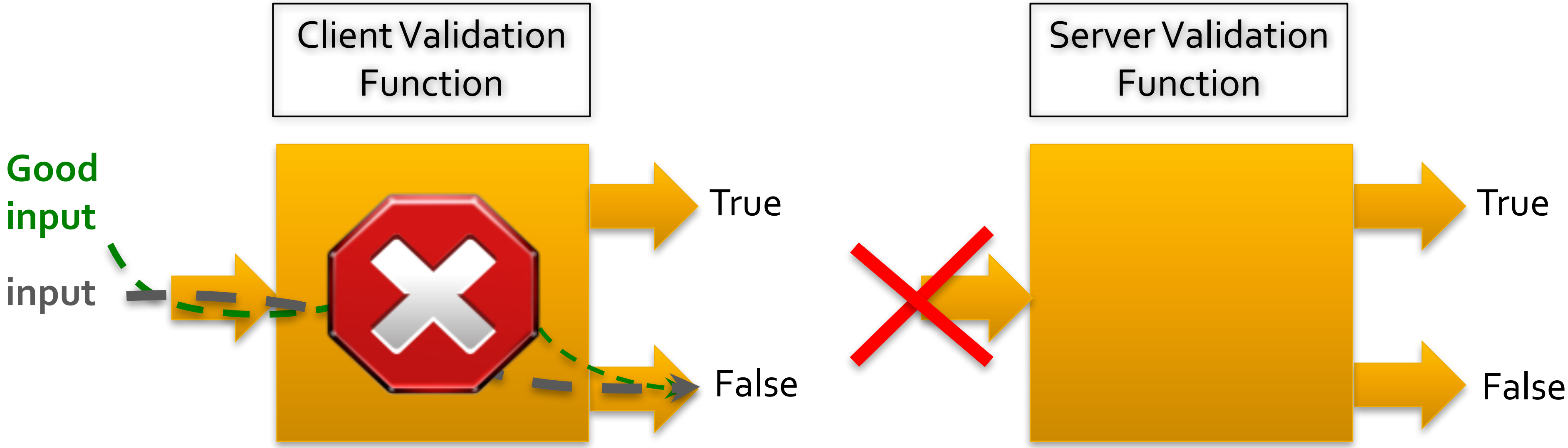


- Two problems may occur:
  - Either the client side input validation function was under constrained and accepted bad inputs
  - Or the server side input validation function was over constrained and rejected some good input

Client Rejects

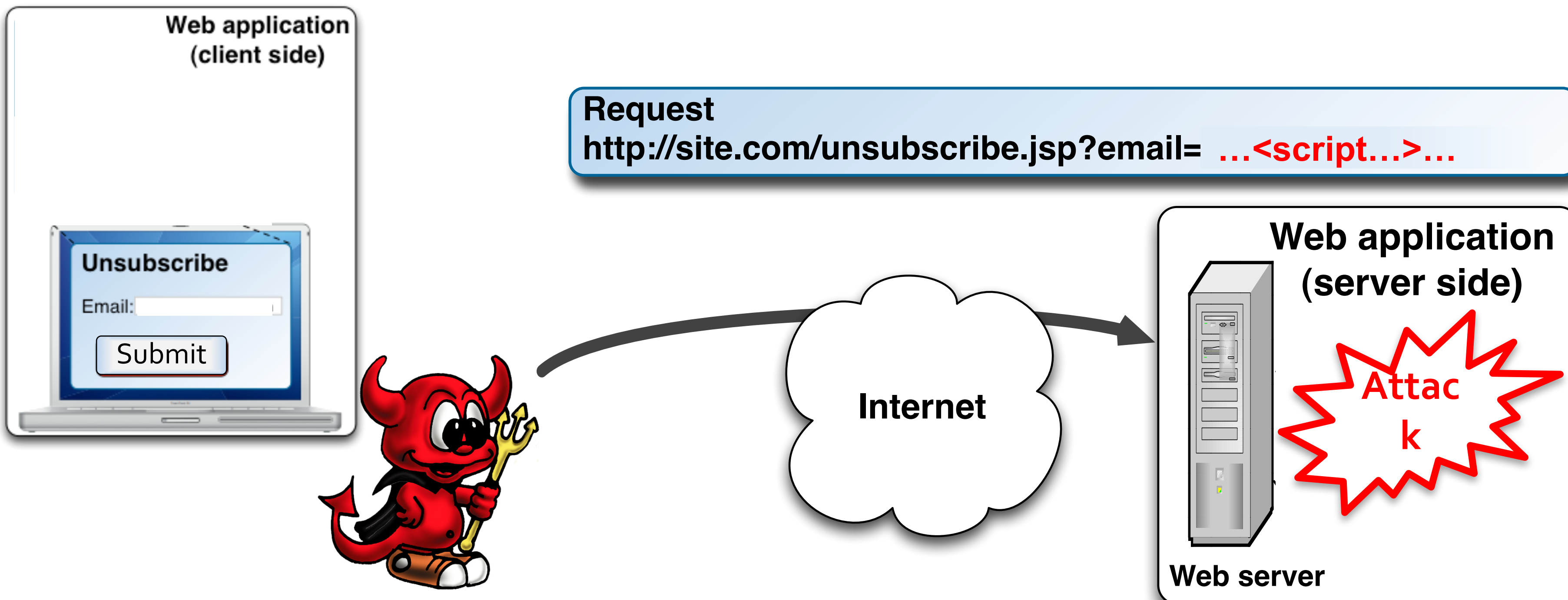


Client Rejects

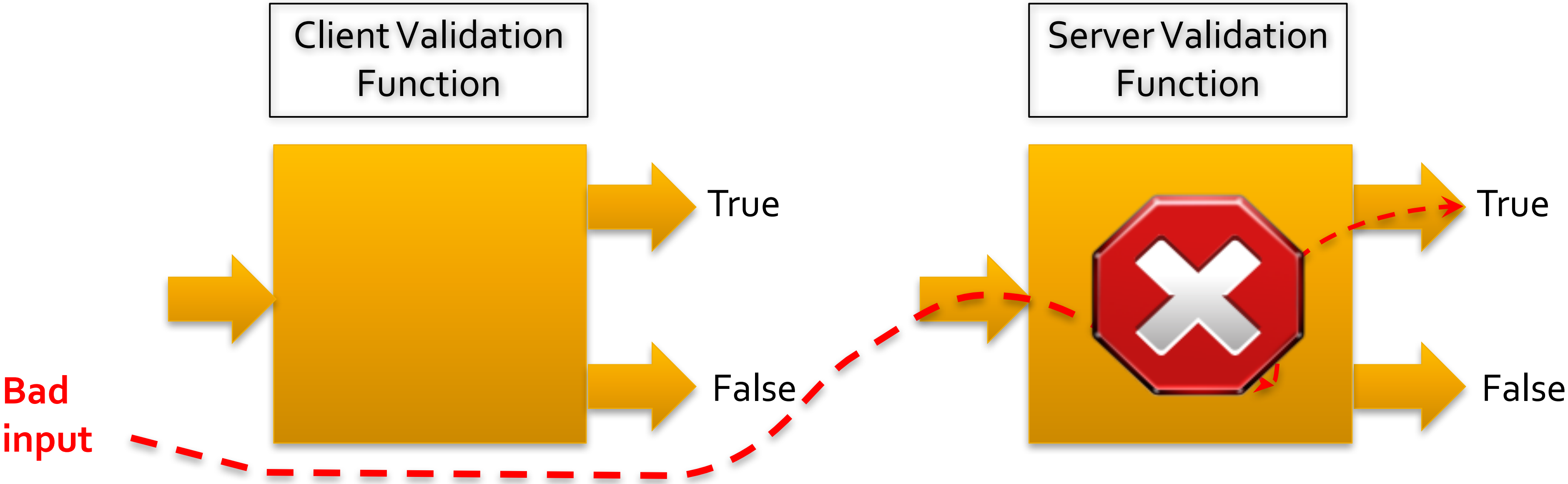


- A problem may occur:
  - the client side input validation function was over constrained and rejected some good input
- What happens when Input value is bad and the server accepts this value?

Client Rejects – Server Accepts

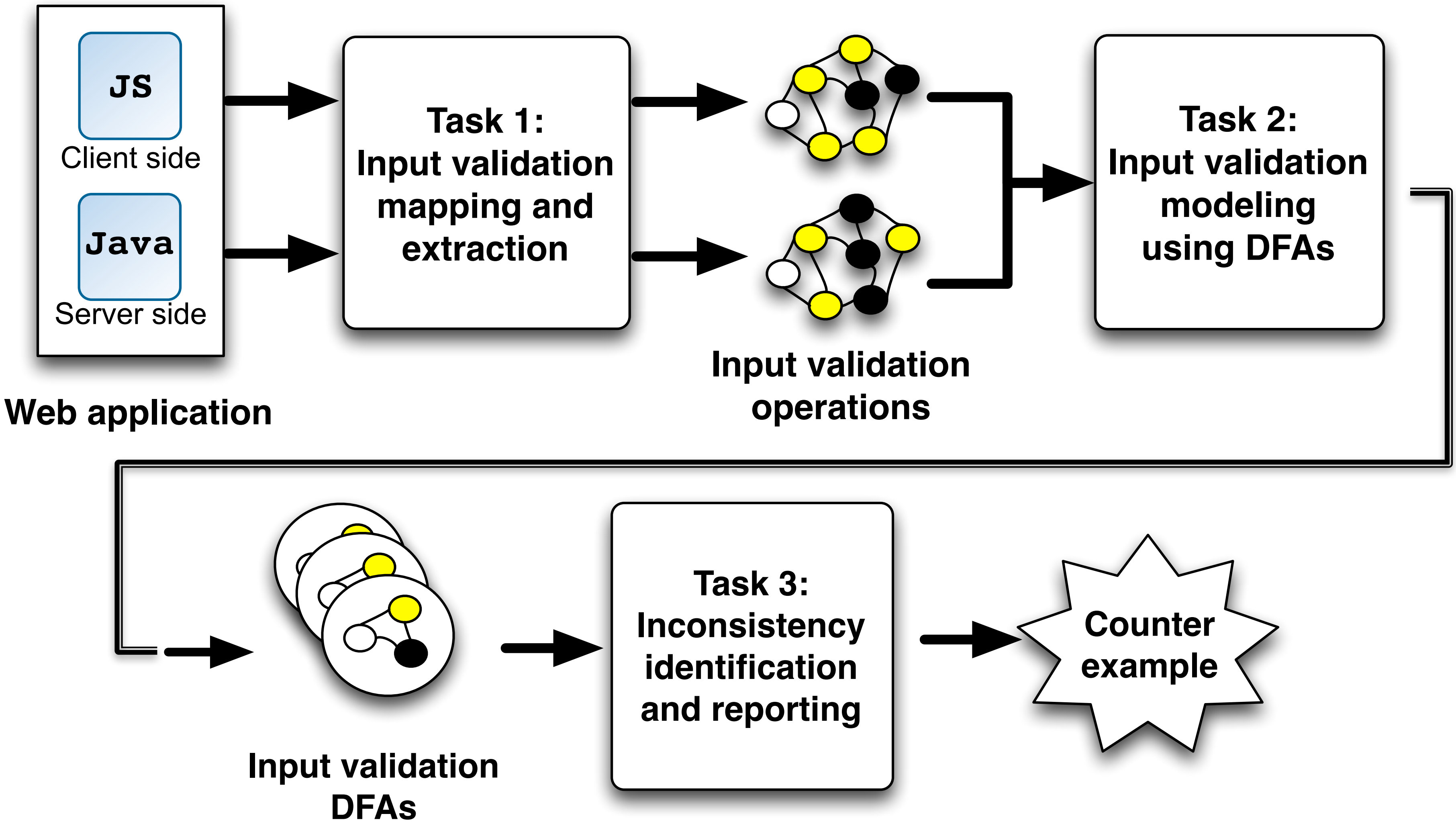


Client Rejects – Server Accepts

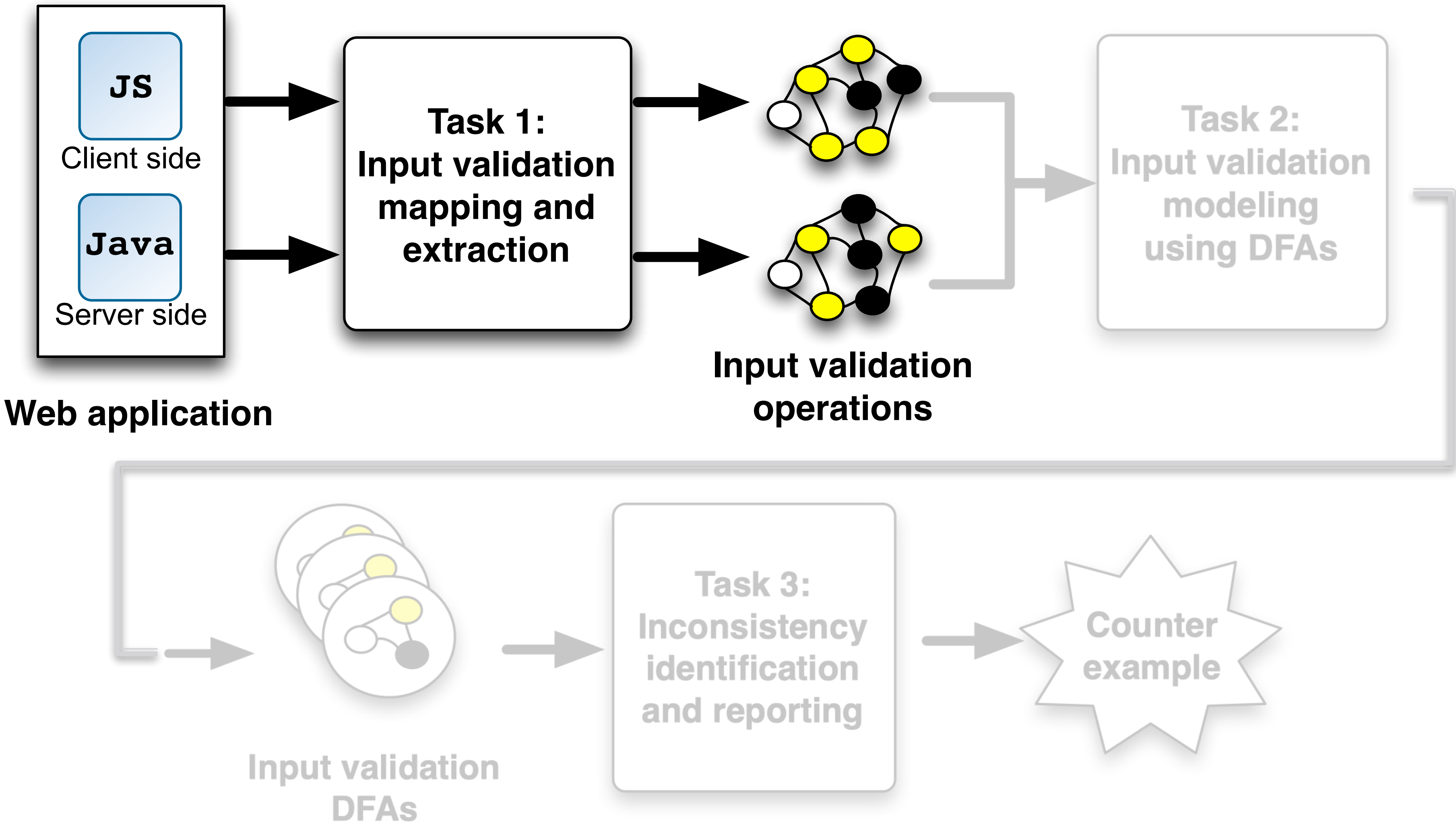


- The server side input validation function was under constrained and accepted bad inputs
- Serious security problem

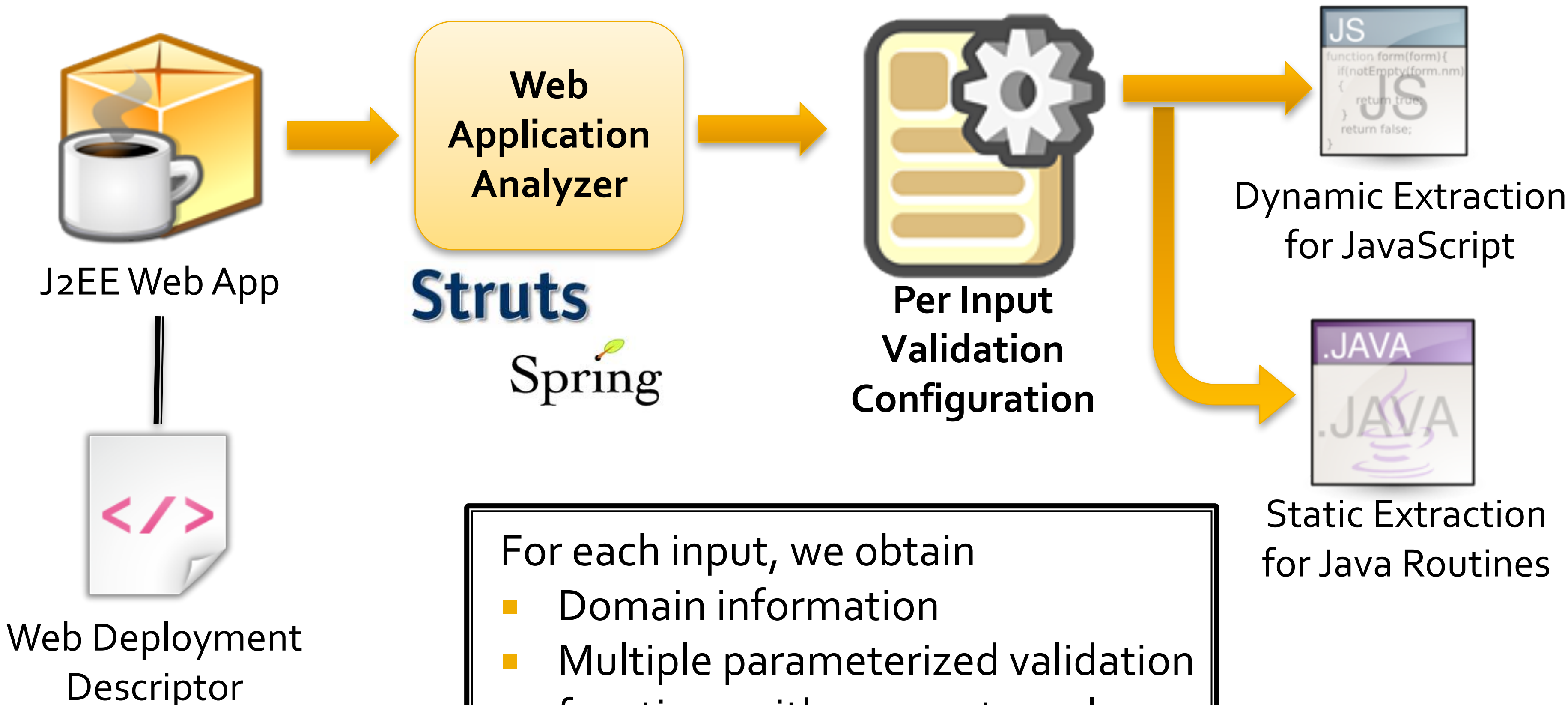
Approach


JS
Client side
Java
Server side
Web application
Task 1:
Input validation
mapping and
extraction
Input validation
operations
Task 2:
Input validation
modeling
using DFAs
Input validation
DFAs
Task 3:
Inconsistency
identification
and reporting
Counter
example

Mapping & Extraction Phase



Input Validation Mapping



For each input, we obtain

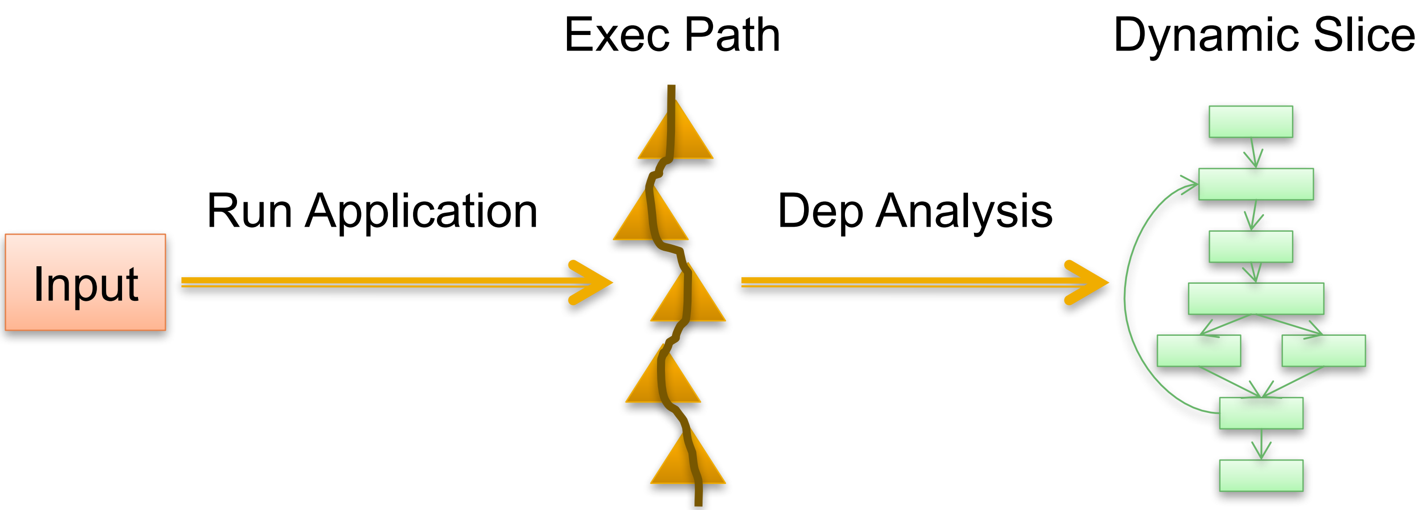
- Domain information
- Multiple parameterized validation functions with parameter values
- Path to access the web application form

Dynamic Extraction for Javascript

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

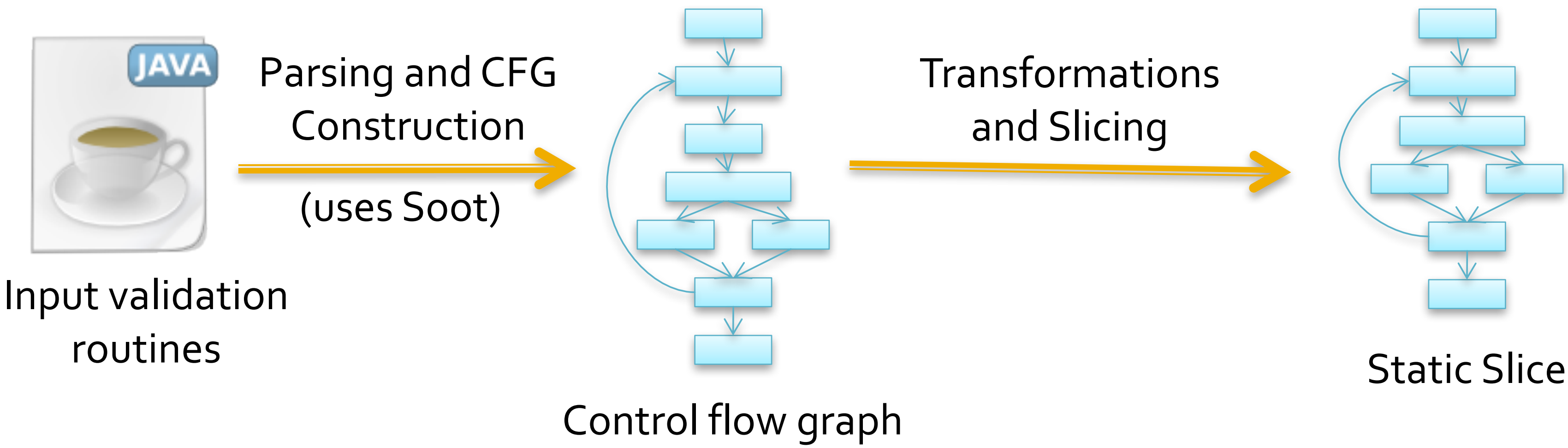
- Why dynamic?
  - Javascript is very dynamic
  - Object oriented
  - Prototype inheritance
  - Closures
  - Dynamically typed
  - eval

Dynamic Extraction for Javascript



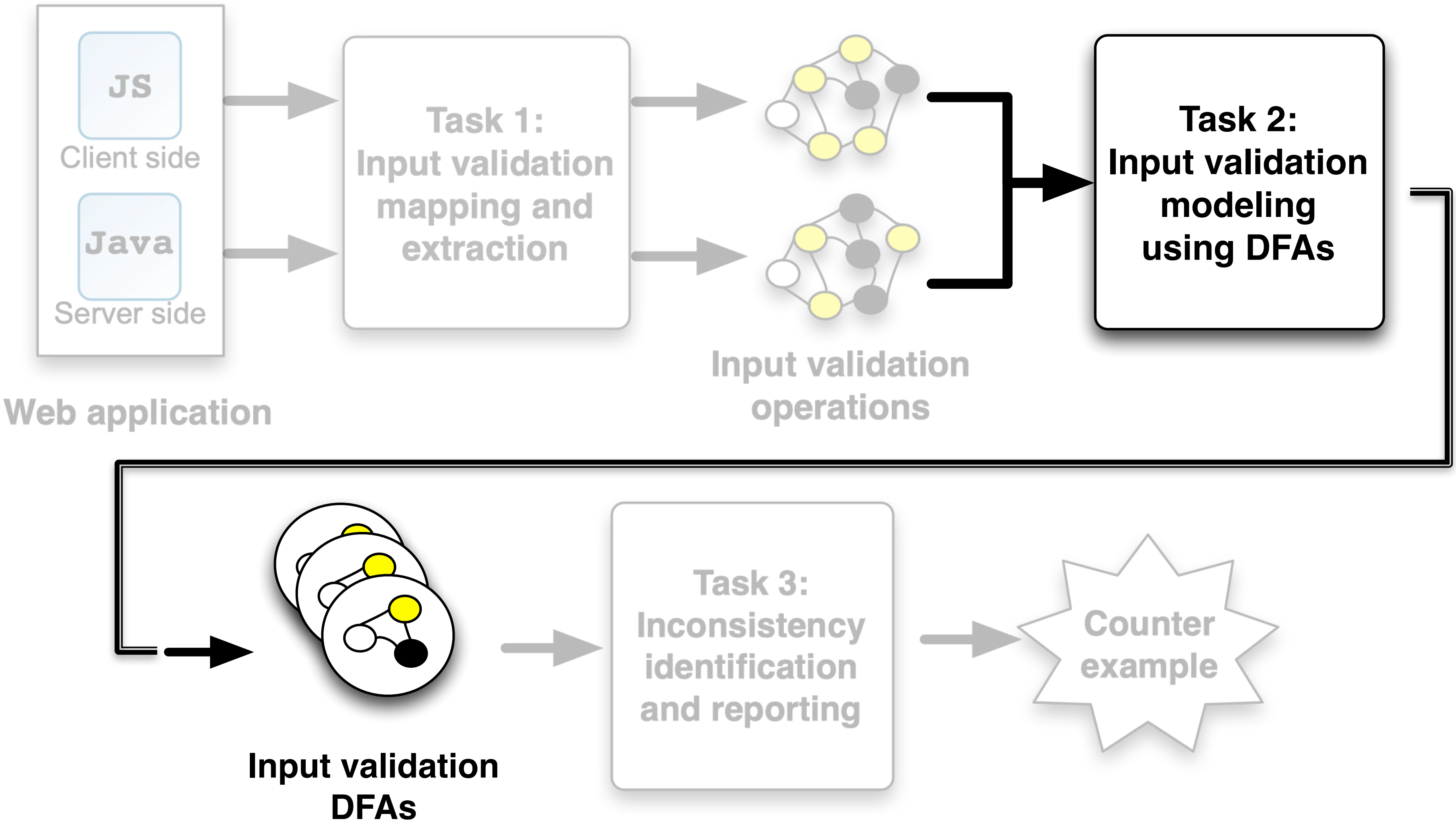
- Number of valid inputs
  - Inputs are selected heuristically
- Instrument execution
  - **HtmlUnit**: browser simulator
  - **Rhino**: JS interpreter
- Convert all accesses on objects and arrays to accesses on memory locations

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 Modeling



Compute Client & Server DFAs

- 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

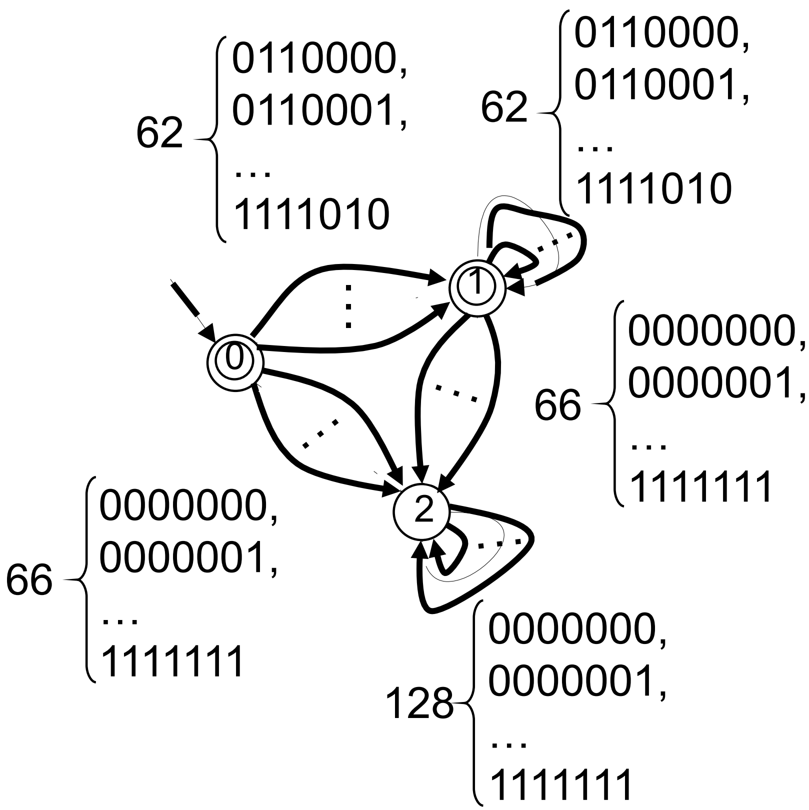
- We use automata based string analysis
  - Associate each string expression in the program with an automaton
  - The automaton accepts an over approximation of all possible values that the string expression can take during program execution

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

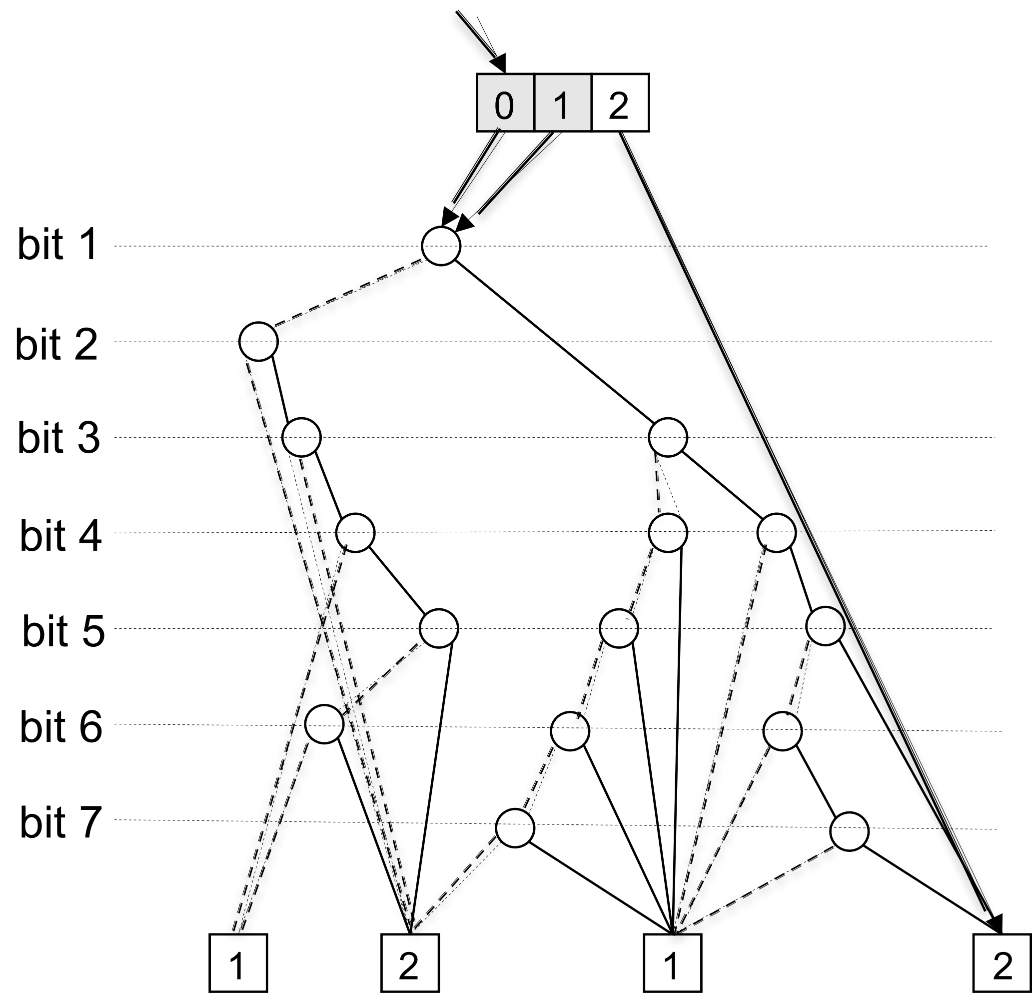
- Flow sensitive, intraprocedural and path sensitive

Symbolic Automata

Explicit DFA representation

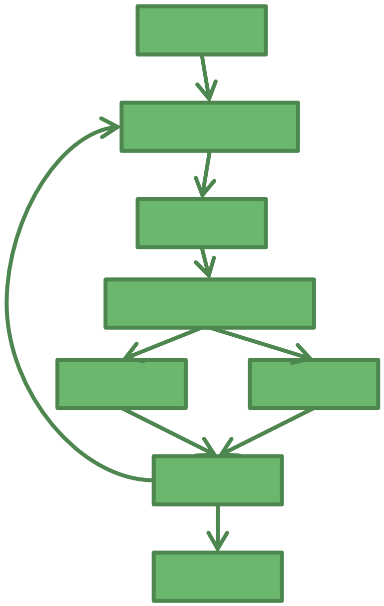


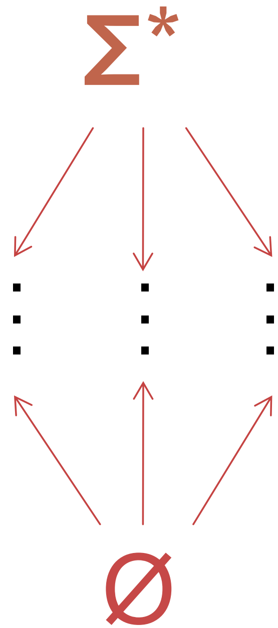
Symbolic DFA representation



Fixpoint & Widening

Due to loops we need fixpoint computation



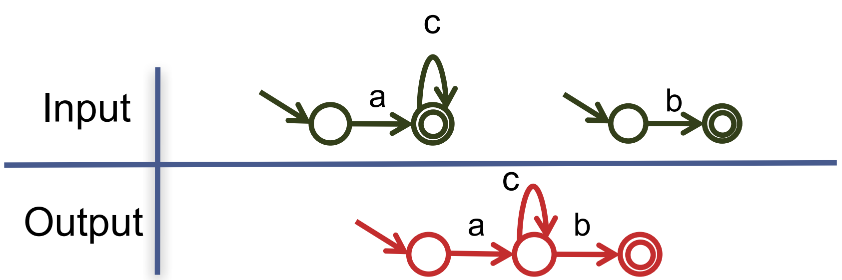


Lattice with infinite height

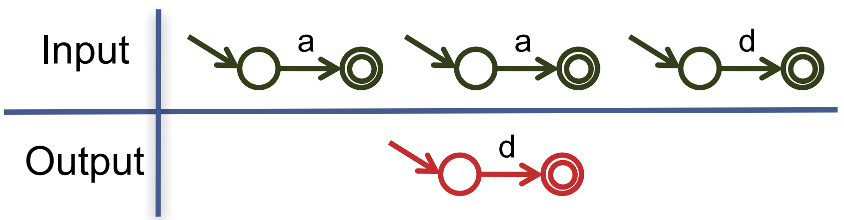
- We use an automata based widening operation to over-approximate the fixpoint
  - Widening operation over-approximates the union operations and accelerates the convergence of the fixpoint computation

Modeling String Operations

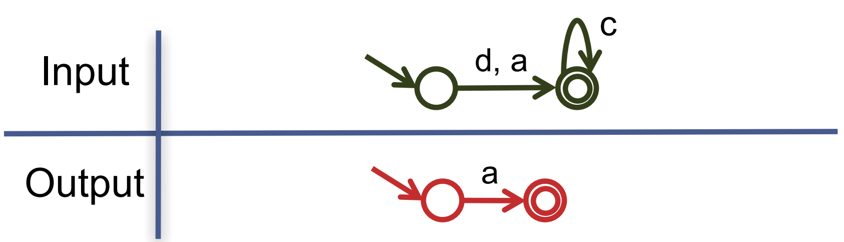
- Modeling string operations
  - CONCATENATION
    - `y = x + "b"`



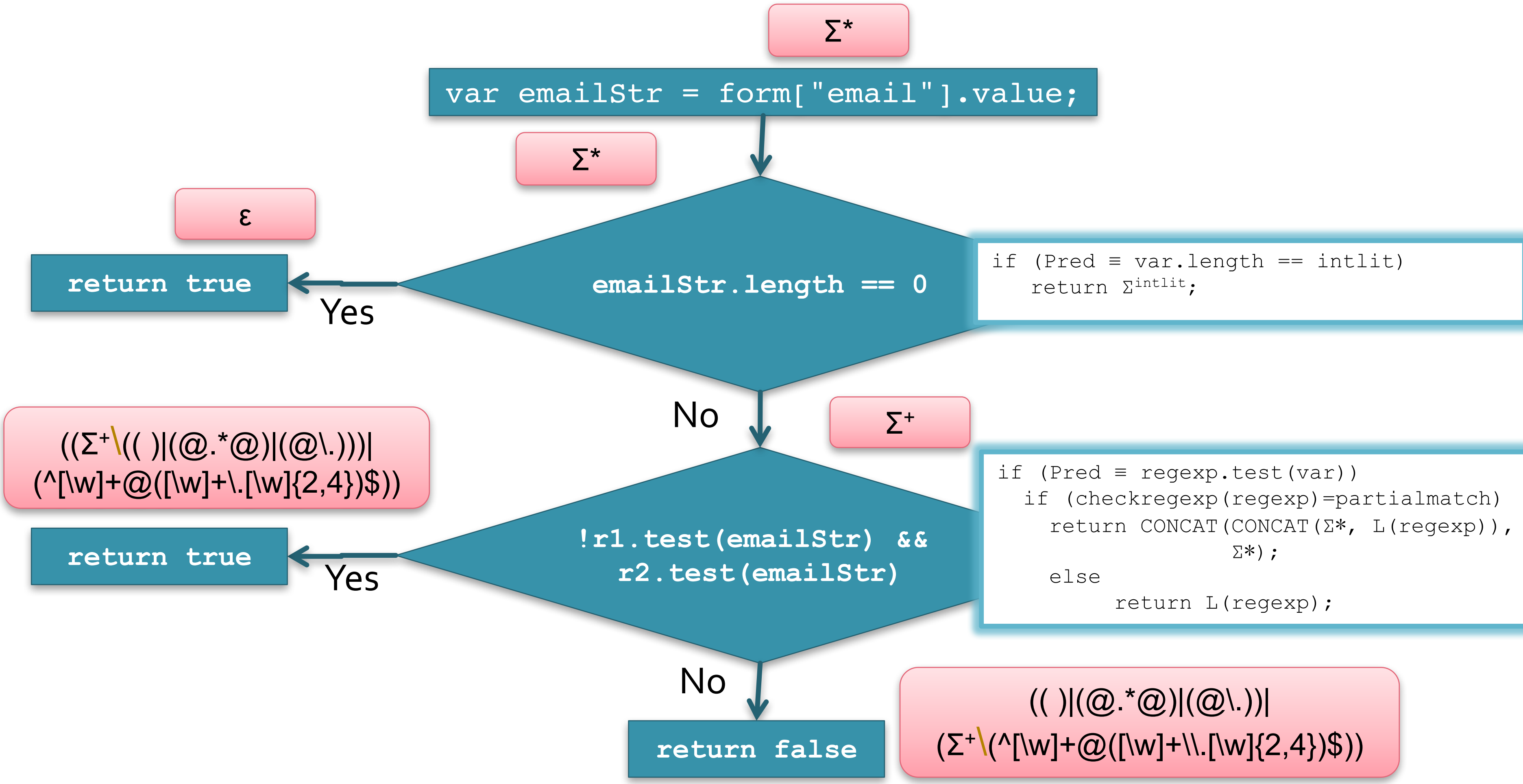
  - REPLACEMENT
    - Language based replacement
    - `replace(x, "a", "d")`



  - RESTRICTION
    - `If (x = "a"){ … }`

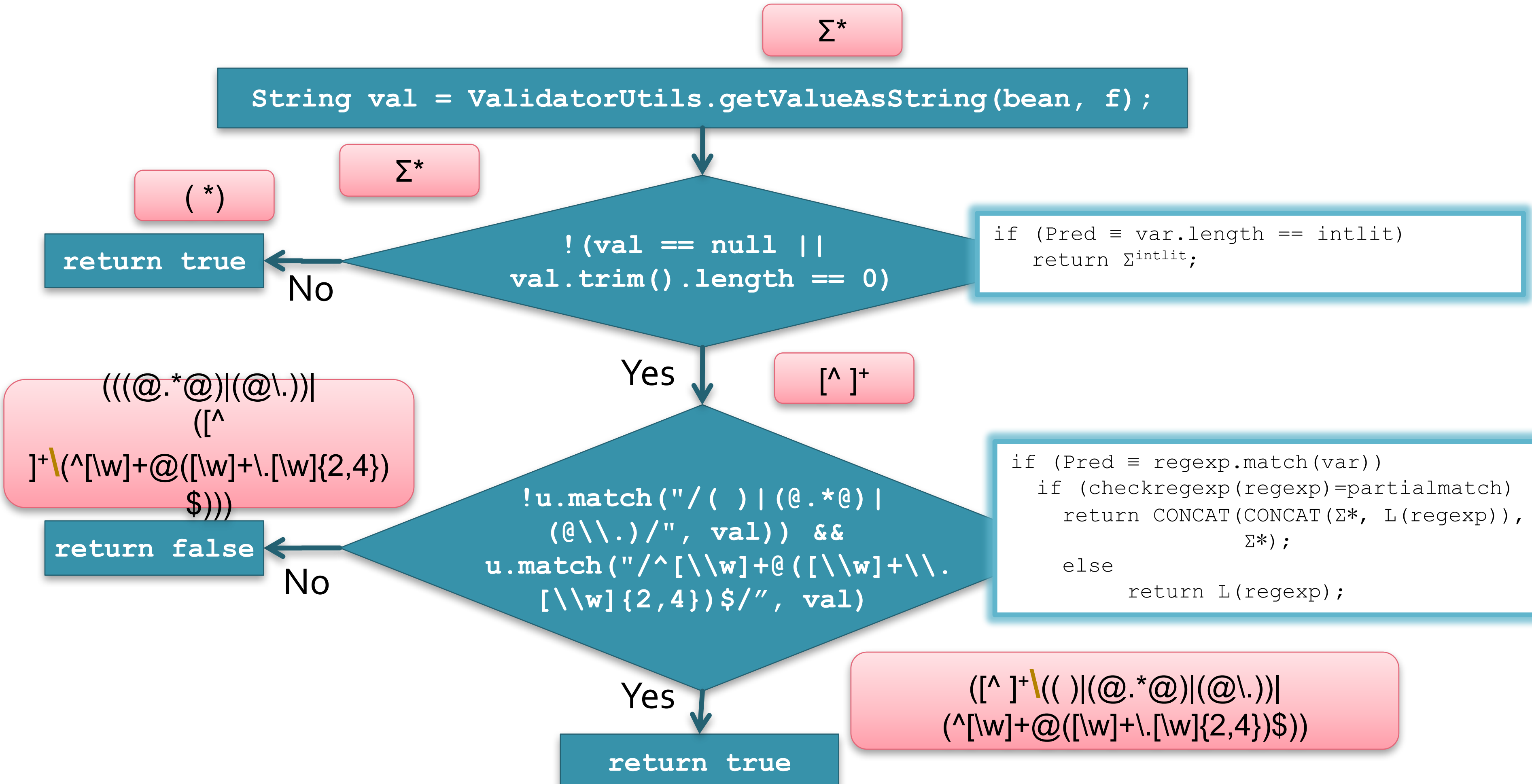


String Analysis Example Client-Side



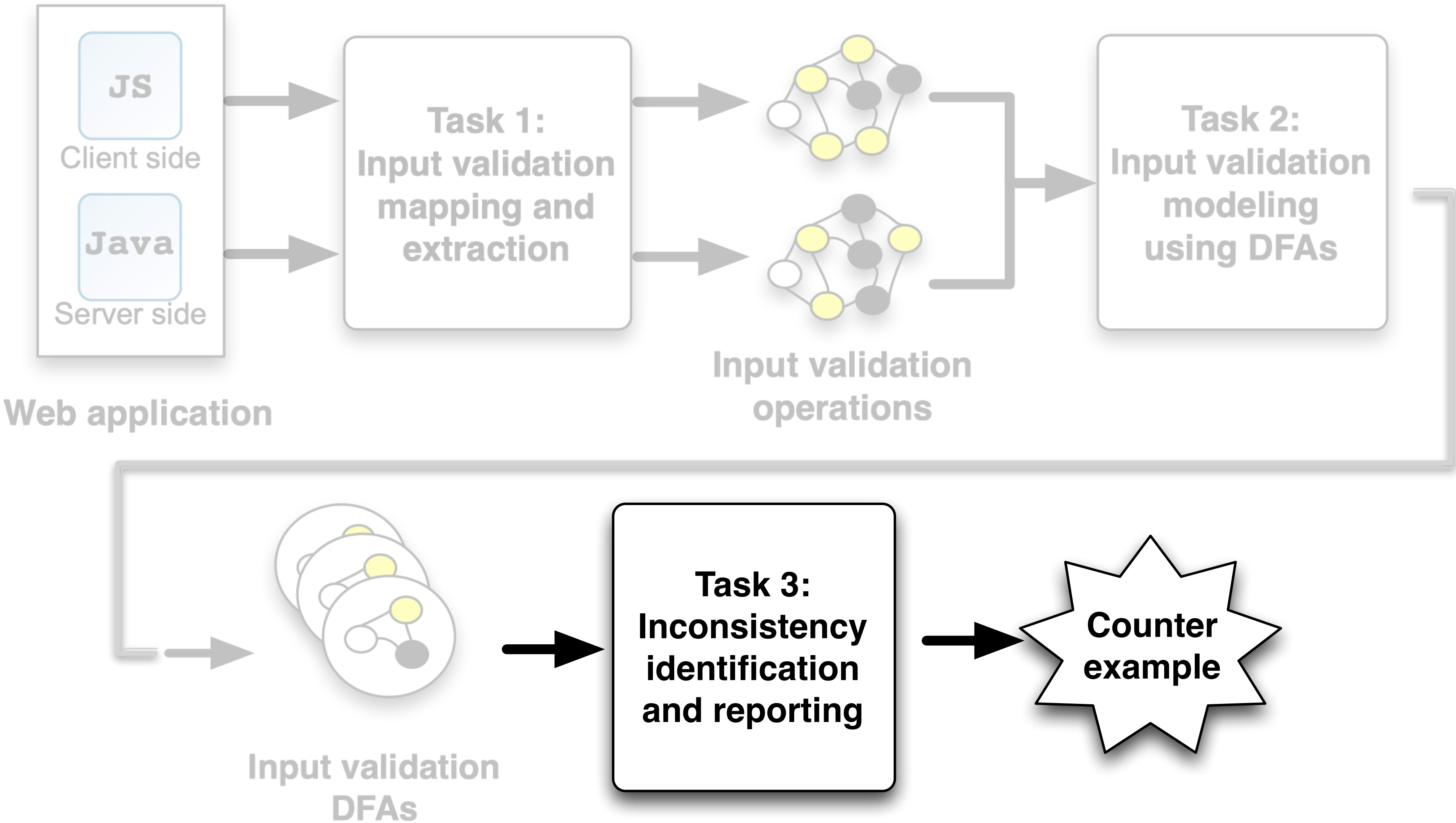
L(A_c) = (Σ*\(( )|(@.*@)|(@\.)))|(^[\w]+@([\w]+\.[\w]{2,4})$)

String Analysis Example Server-Side



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

Inconsistency Identification



Computing Difference Signature

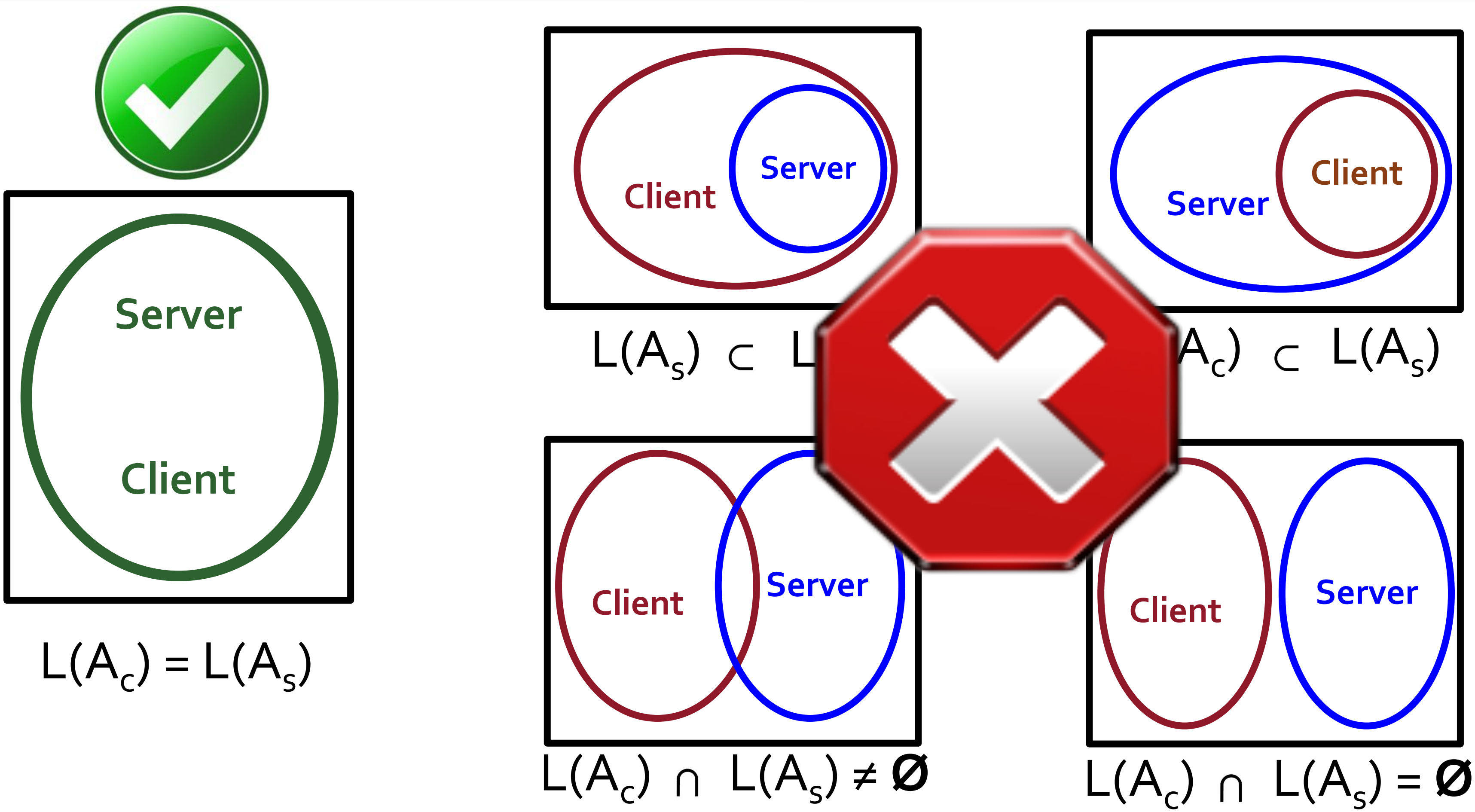
- Compute two difference signatures:
  - $L(A_{s\text{-}c}) = L(A_s) \setminus L(A_c)$
  - $L(A_{c\text{-}s}) = L(A_c) \setminus L(A_s)$

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

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

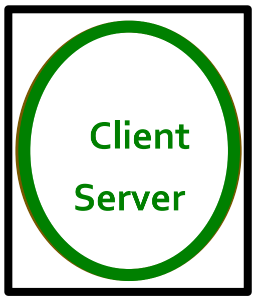
Client – Server Relation

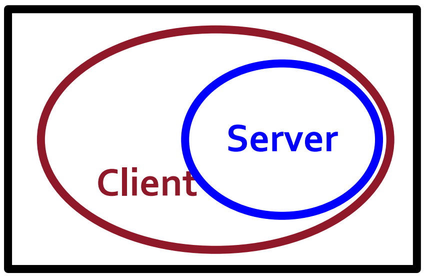
Five possible relationships between $L(A_c)$ and $L(A_s)$



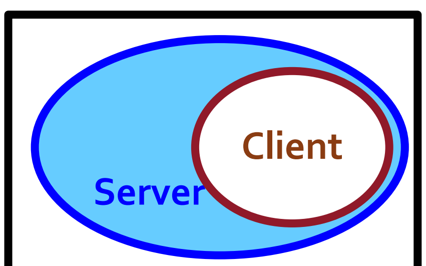
Server-Client Difference Signature

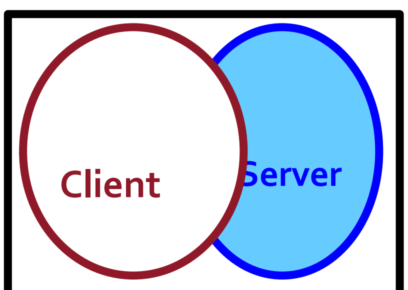
We compute $L(A_{s\text{-}c})$

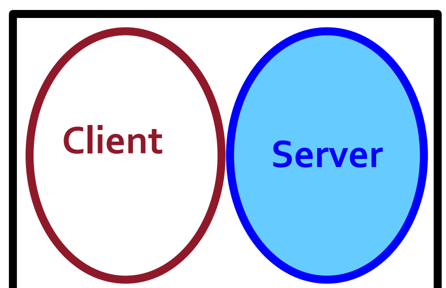




$L(A_{s\text{-}c}) = \emptyset$



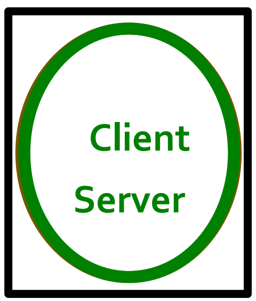


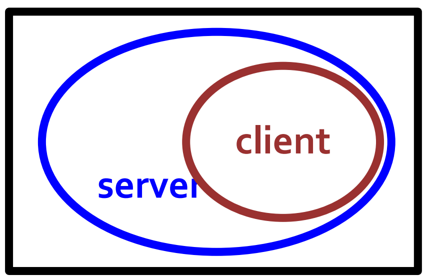


$L(A_{s\text{-}c}) \neq \emptyset$

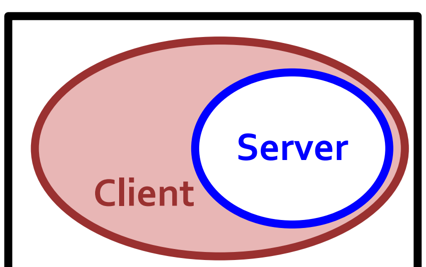
Client-Server Difference Signature

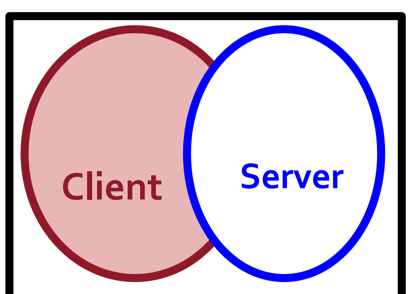
We compute $L(A_{c\text{-}s})$

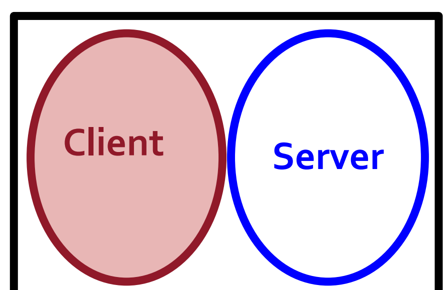




$L(A_{c\text{-}s}) = \emptyset$







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

Computing Inconsistencies for the Two Example Functions

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

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

\

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

=

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

Counter Example = " "

Evaluation

- Analyzed a number of Java EE web applications

Name	URL
JGOSSIP	http://sourceforge.net/projects/jgossipforum/
VEHICLE	http://code.google.com/p/vehiclemanage/
MEODIST	http://code.google.com/p/meodist/
MYALUMNI	http://code.google.com/p/myalumni/
CONSUMER	http://code.google.com/p/consumerbasedenforcement
TUDU	http://www.julien-dubois.com/tudu-lists
JCRBIB	http://code.google.com/p/jcrbib/

Extraction Phase Performance

Subject	Frm	Inputs	VI_C	ET_C(s)	VI_S	ET_S(s)
JGossip	25	83	74	329.8	83	4.38
Vehicle	17	41	41	155.5	41	2.04
MeoDist	18	62	62	192.2	62	1.93
MyAlumni	46	141	0	0	141	4.28
Consumer	3	21	14	68.4	21	1.1
Tudu	3	11	0	0	11	0.78
JcrBib	21	45	0	0	45	1.51

Analysis Phase Memory Performance

Subject	Client-Side DFA							Server-Side DFA						
	Avr size (mb)	Min		Max		Avr		Avr size (mb)	Min		Max		Avr	
		S	B	S	B	S	B		S	B	S	B	S	B
JGOSSIP	6.0	4	10	35	706	6	39	6.1	4	24	35	706	6	41
VEHICLE	4.8	4	24	7	41	5	26	4.8	4	24	7	41	5	26
MEODIST	5.7	5	25	5	25	5	25	5.7	5	25	5	25	5	25
MYALUMNI	3.2	4	10	4	10	4	10	3.2	3	24	5	25	5	25
CONSUMER	5.3	4	10	17	132	5	25	5.3	4	24	17	132	7	41
TUDU	6.1	4	10	4	10	4	10	6.1	3	24	23	264	8	68
JCRBIB	5.4	4	10	4	10	4	10	5.4	5	25	5	25	5	25

Analysis Phase Time Performance & Inconsistencies That We Found

Subject	Time (s)	AC-S	AS-C
JGossip	3.2	9	2
Vehicle	1.5	0	0
MeoDist	1.7	0	0
MyAlumni	2.9	141	0
Consumer	1.0	7	0
Tudu	0.6	11	0
JcrBib	1.2	45	0

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**