STRing **A**utomato**N** **GE**nerato**R**
(*STRANGER*)

Fang Yu (yuf@cs)
Marco Cova (marco@cs)

CS272 – Spring 2007

String Analysis

- Static analysis technique to determine the string values that a variable may hold at specific points in a program
- Answers questions such as:
  - which values may a string variable have?
  - is it possible for a string variable to hold a certain value?

Applications

- Security
  - Detect XSS, SQL injection vulnerabilities [Minamide05] [Gould04]
- Software maintenance
  - Check validity of dynamically generated SQL queries [Gould04]
  - Check validity of dynamically generated output [Kirkegaard04]
- Reverse Engineering
  - Set of possible values of variable at specific program points [Christodorescu05]

Example (XSS Detection)

```
$www = $_GET[“www”];

if (strpos($www, “http://”) == False) {
    $www = “http://” .  $www;
}

$clean_www = ereg_replace(“<script”,
    “&lt;script”, $www);

echo “www parameter is “ . $clean_www;
```

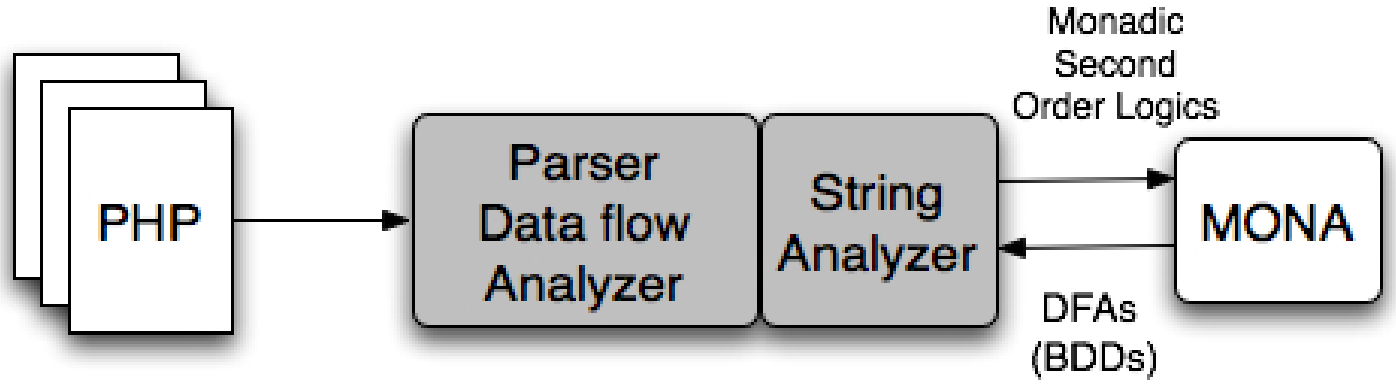
STRing AutomatoN GEneratoR

- STRANGER
  - An automaton-based tool to perform string analysis.
  - For each string variable, generate a deterministic finite automaton (DFA) to accept its possible values.
  - We currently target on PHP web applications.

Automata-based String Analysis

- Construct (e)
  - *e* is a regular expression over the ASCII alphabet
  - returns a dfa M such that L(M)={w| w $\in$ L(e)}.
- Concatenate (dfa M1, dfa M2)
  - returns a dfa M such that L(M)={w_1w_2| $w_1 \in$ L(M1) and $w_2 \in$ L(M2)}
- Replace (dfa M1, dfa M2, c)
  - returns a dfa M, such that L(M)={$w_1cw_2c\ldots w_kcw_{k+1}$| $\exists$ k, $w_1x_1w_2x_2\ldots w_kx_kw_{k+1} \in$ L(M1), $\forall$ i, $x_i \in$ L(M2), $w_i \notin$ L(M2)}
- Union (dfa M1, dfa M2)
  - returns a dfa M, such that L(M)=L(M1) $\cup$ L(M2)
- Intersect (dfa M1, dfa M2)
  - returns a dfa M, such that L(M)=L(M1) $\cap$ L(M2)
- EmpCheck(dfa M)
  - returns true iff L(M) is not empty

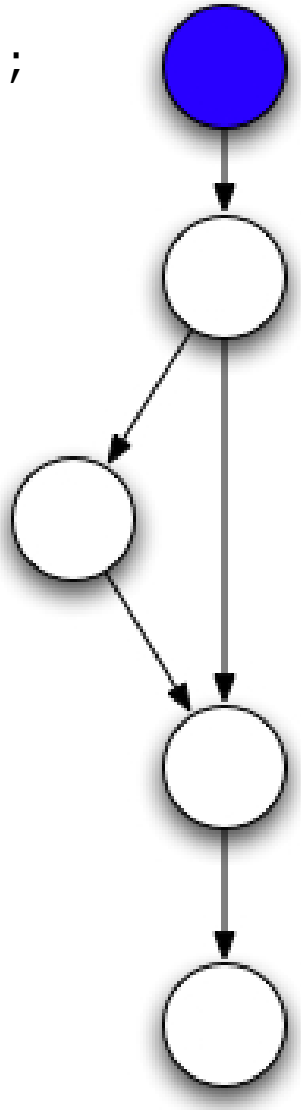
Framework



- The Front End
  - PHP Parser
  - String Analyzer
- The Back End
  - Constructor
    - Monadic Second Order Logics
  - Automata Manipulator
    - Deterministic Finite Automata
    - Binary Decision Diagrams

Example

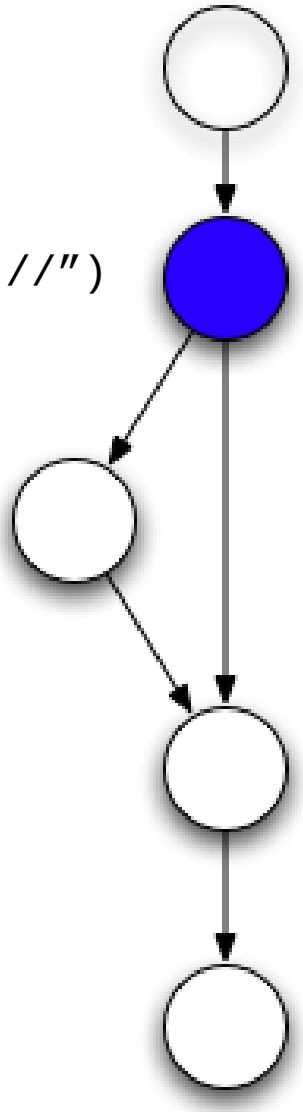
```
$www = $_GET[“www”];
```



$www = S*

Example

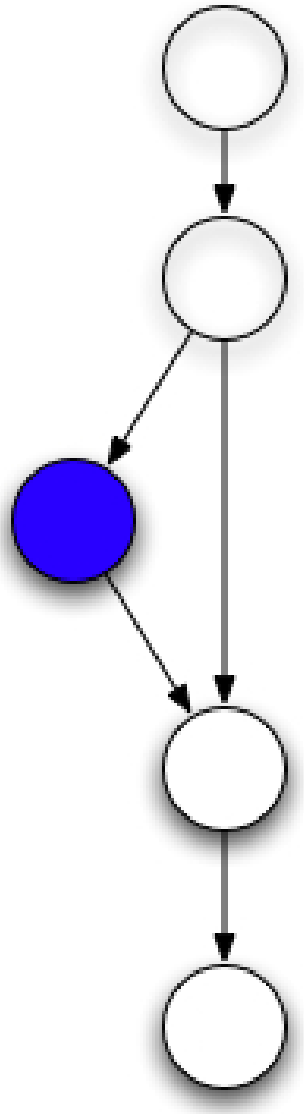
```
strpos($www, “http://”)
   == False
```



$www = S*

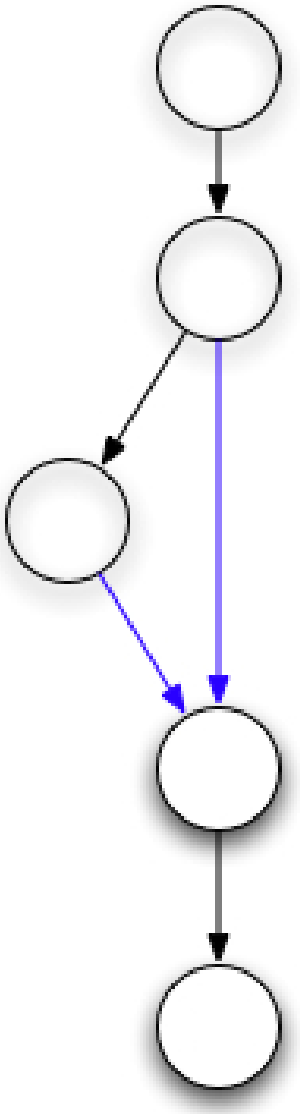
Example

```
$www = “http://” .
  $www;
```



$www = http://S*

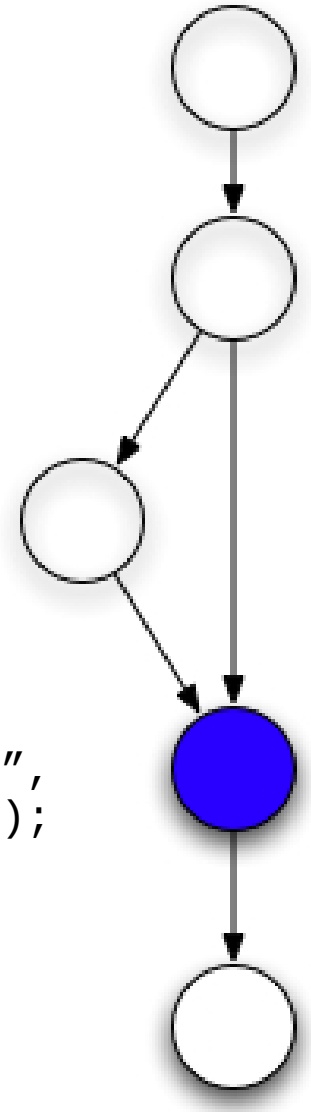
Example



$www = union (http://S*,S*) = S*

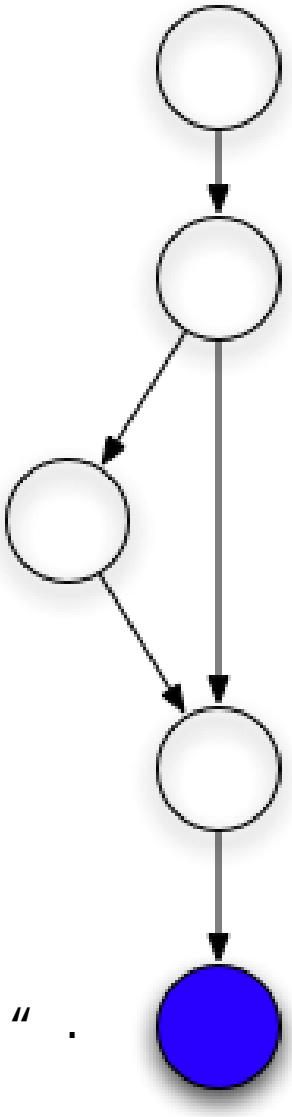
Example

```
$clean_www =
 reg_replace(“<script”,
   “&lt;script”, $www);
```



$clean_www = replace(“<script”, “<script”, S*) =
= S* - S*<scriptS*

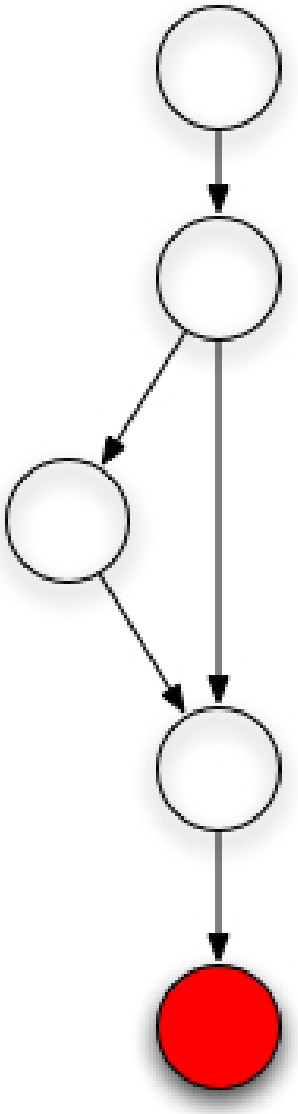
Example



```
echo
  “www parameter is “ .
  $clean_www;
```

EmpCheck (Intersect ($clean_www , $malicious))

Example



$malicious = “<img onfocus=…>”

Challenges

- **How to handle loops?**
- How to encode ASCII in MONA?
- How to do “replace”?

Loops

- Problem: in theory, lattice of data-flow elements has infinite height
  - Fixpoint computation through loops might not converge

```
for ($i = 0; $i < $_GET[“count”]; $i++)
    $www = $www . “a”;
```

- Solutions:
  - Set a cutoff and stop analyzing loops after a certain number N of iterations (unsound but no false positives). For example, if N = 2 and `$_GET[“count”] > 2,` then `$www = S*a ∪ S*aa`
  - Set a cutoff and widen to S* if fixpoint computation does not converge before cutoff (sound but it might generate lots of false positives if widening occurs). For example, if N = 2 and `$_GET[“count”] > 2,` then `$www = S*`
  - Set a cutoff and use a better widening (future work)
  - Bound the length of strings so that the lattice has finite height

Challenges

- How to handle loops?
- **How to encode ASCII in MONA?**
- How to do “replace”?

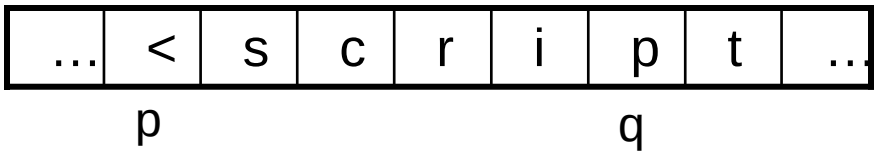
Encode ASCII in MONA

- MONA: Monadic Second Order Logics
  - Var0: Boolean
  - Var1: Int (p, q, r)
  - Var2: Int set (P, $)

- Declare a bounded integer set $ to denote the positions of a string
  - var2 $ where ~ex1 p where true: p notin $ & p+1 in $;

Encode ASCII in MONA

- Consider a string within a bounded length



- Recursively construct is_E(p,q), such that
  - is_E(p,q) holds iff the substring from position p to q (including p'th letter but not the q'th) belongs to the language defined by E
- L(E) (bounded in length)
  - is_E(0, max($)+1);

Encode ASCII in MONA

- Declare subsets for ASCII character
  - var2
    - bit0 where bit0 sub $,
    - bit1 where bit1 sub $,
    - bit2 where bit2 sub $,
    - bit3 where bit3 sub $,
    - bit4 where bit4 sub $,
    - bit5 where bit5 sub $,
    - bit6 where bit6 sub $,
    - bit7 where bit7 sub $;

Encode ASCII in MONA

- Character encoding
  - # ASCII 'b' is 98, which is 01100010 //the p'th position is b

    macro is_b(var1 p, var1 q) =

    q = p + 1 &

    p notin bit0 & p in bit1 & p notin bit2 & p notin bit3 &p notin bit4 & p in bit5 & p in bit6 & p notin bit7;
  - # ASCII 'all' //the p'th position is anything in S

    macro is_S(var1 p, var1 q) =

    q = p + 1;
- Consecutive position
  - macro consecutive_in_set(var1 p, var1 q, var2 P) =

    p < q & p in P & q in P &

    all1 r: p < r & r < q => r notin P;

Encode ASCII in MONA

- Concatenate (xy)
  - pred is_**xy**(var1 p, var1 q) =

    ex1 r where p<=r & r<=q: is_**x**(p, r) & is_**y**(r, q);
- Closure (x*)
  - pred is_**x**_star(var1 p, var1 q) =

    ex2 P: p in P & q in P &

    all1 r, r': consecutive_in_set(r, r', P) => is_**x**(r, r');

Any regular expression can be specified as a predicate

MONA

- The Monadic Second Order Logics can be used to generate Deterministic Finite Automaton
  - ./mona –w xxx.mona xxx.fsm
- The dfa is actually recorded as a BDD
  - ./mona –wx xxx.mona xxx.dfa
- Advantages
  - BDD has a canonical form and can be manipulated efficiently
  - Constant time to emptiness checking
  - Polynomial time to union, intersection, negation, reduction to canonical form

Challenges

- How to handle loops?
- How to encode ASCII in MONA?
- **How to do “replace”?**

Replacement

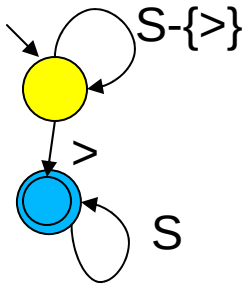
- Replace (dfa M1, dfa M2, c)
  - Output a dfa M, such that L(M)={$w_1cw_2c\ldots w_kcw_{k+1}$| $\exists$ k, $w_1x_1w_2x_2\ldots w_kx_kw_{k+1} \in$ L(M1), $\forall$ i, $x_i \in$ L(M2), $w_i \notin$ L(M2)}
- Algorithm
  - Generate a dfa M1', such that L(M1')={$w_1\#_1\underline{w}_2\#_2w_3\ldots$|$w_1w_2w_3\ldots \in$ L(M1)}
  - Generate a dfa M2', such that

    L(M2')={$w_1\#_1\underline{w}_2\#_2w_3\ldots$|$w_1,w_3,\ldots \notin$ L(M2), and $w_2,.. \in$ L(M2)}
  - Generate a dfa M', such that

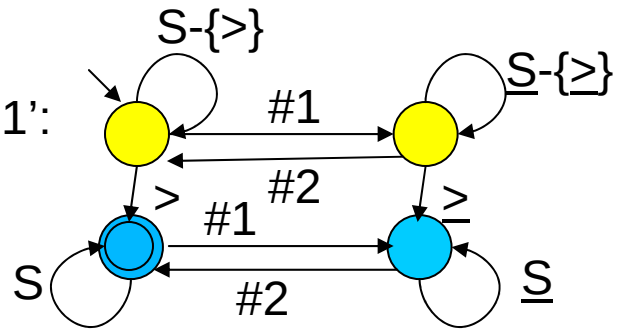
    L(M')={$w_1\#_1c\underline{w}_2\#_2w_3\ldots$| $w_1\#_1\underline{w}_2\#_2w_3\ldots \in$ L(M1') $\cap$ L(M2')}
  - M=*project*({$\#_1,\#_2,\_$}, M'), where _ ={$\underline{c}$ | c $\in$ S}, and *project* returns a dfa in which all specified characters are replaced as an empty string.

Internal DFAs

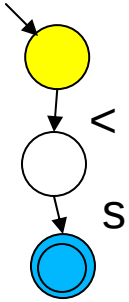
M1:



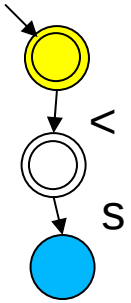
M1':



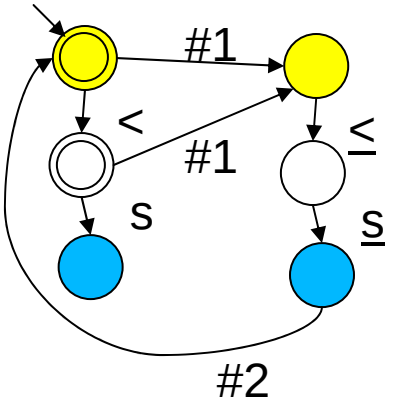
M2:



Complement M2:



M2':



Conclusion

- STRANGER
  - A BDD based STRing AutomatoN GEneratoR
  - A string analyzer for PHP web application

Questions?