A search based approach for security testing

Ceccato Mariano, Andrea Avancini
ceccato@fbk.eu
Background

Vulnerability Disclosures of Web applications

Web vulnerabilities by class

- Web applications are publicly exposed to a hostile environment
- Successful attacks may cause
  - Sensitive information disclosure
  - Revenue loss
- XSS is one of the most prominent security vulnerability
XSS example

<?php
    $a = $_GET["name"];
    echo "Your name is: ";
    echo $a;
?>

index.php

http://www.mysite.com?name=Mariano

Mozilla Firefox
file://home/m...2/MySite.html
Your name is: Mariano

Mozilla Firefox
file://home/m...2/MySite2.html
Your name is: click here

evil.php?data=23333456fdd333

HTTP request
XSS example

1: construct a malicious link
2: email the URL to user and convince user to click on it
3: request the page
4: page with malicious script
5: run
<?php
$user = $_GET["username"];
$pass = $_GET["password"]; 
$pass2 = $_GET["password2"]; 
if ( strpos($user, "<script"))
$user = htmlspecialchars($user);
if ($user in $users)
echo "username already taken";
else
if ( strlen($pass) < 5)
echo "password too short";
else
if ($pass == $pass2)
new_user($user, $pass);
echo "new account for";
echo $user; Sink
else
echo "passwords do not match";
?>

Validation
Tainted
Sink

London – March, 22nd-23rd 2012 – 18th CREST Open Workshop
<?php
$a = $_GET[ "first name" ];
$b = $_GET[ "surname" ];
if ( strpos ( $a, "<script" ) ) {
    $a = htmlspecialchars ( $a );
    if ( isset ($b) )
        $go_on_b = true;
    else
        $go_on_b = false;
    if ( $go_on_b )
        $b = htmlspecialchars ( $b );
echo $a;
if ( $go_on_b )
echo $b;
?>
Static analysis

- Static identification of candidate vulnerabilities based on control and data dependencies (flow analysis)
- Valuable help for manual review, it provides starting points for code inspection
- Missing evaluation of dynamic constructs (reflective calls, pointes, …)
- Conservative approach (false positives)
- Test cases are needed
Fitness function

- Static analysis identifies those statements to execute/skip
- We compute the target branches
- Fitness function = # of target branches executed by a candidate solution (approach level)

{ (username, Mariano), (password, xxx), (password2, yyy) }

http://mysite.com?username=Mariano&password=xxx&password2=yyy
Crossover

\{(\text{firstname, john}), (\text{surname, smith}), (\text{age, 23})\} \quad \rightarrow \quad \{(\text{firstname, mark}), (\text{address, broadway}), (\text{job, teacher})\} \\
\downarrow \\
\{(\text{firstname, john}), (\text{surname, smith}), (\text{job, teacher})\} \\
\{(\text{firstname, mark}), (\text{address, broadway}), (\text{age, 23})\}
Mutation

\{\text{firstname, john}, \text{surname, smith}\}

- Add pair
- Remove pair

\{\text{firstname, john}, \text{surname, smith}, \text{age, 23}\}

- Change parameter value

\{\text{firstname, john}, \text{surname, smith x3scr}, \text{age, 23}\}

\{\text{firstname, john}, \text{surname, xsmith}, \text{age, 23}\}
Genetic algorithm

- **PRO:**
  - Can be adopted when the analytical solution is not feasible
    - Too complex constraints
  - Effective on a big search space to reach a solution near to the optimum

- **CON:**
  - Solutions near to the optimum may not solve the search problem, they may not expose a possibly complex vulnerability
  - Problems on local optima
  - Difficult to generate input values that satisfy complex conditions on inputs
Need for a local search strategy

- A local optimum may pose a threat to the performance of GA
- Part of the search problem is already solved
- A different strategy may complement the GA search
- **Intuition:** apply the analytic solution to the local search problem
Symbolic path constraints

Concrete Input values

\{ (username, "mariano"), (password, "xxxxx"), (password2, "yyyyy") \}

\$user -> (InputU,"mariano")
\$pass -> (InputP,"xxxxx")
\$pass2 -> (InputP2, "yyyyy")

\text{not (strpos(InputU, "<script")})

\text{true}
\text{not ("mariano" in $users)}
\text{not (InputU in $users)}
\text{not (strlen(InputP)<5)}

InputP == InputP2
InputP => InputP2

1: \$user = \$_GET("username");
2: \$pass = \$_GET("password");
3: \$pass2 = \$_GET("password2");
4: if ( strpos($user, "<script") )
5: \$user=htmlspecialchars($user);
6: if ( $user in $users )
7: echo "username already taken";
8: if ( strlen($pass) < 5 )
9: 
10: if ( $pass == $pass2 )
11: new_user($user, $pass);
12: echo "new account for";
13: echo $user;
14: echo "passwords do not match";
Sat solver

\[
\neg \text{strpos(InputU, "<script")) AND True AND \neg \text{strlen(InputP) < 5) AND InputP == InputP2}
\]

\{
(username, "mariano"),
(password, "xxxxx"),
(password2, "xxxxx")
\}

http://mysite.com?username=mariano&password=xxxxx&password2=xxxxx

- **PRO:**
  - It solvers constraints that could be difficult for heuristics

- **CON:**
  - Limitations on the language accepted by the solver may require to use concrete values
    - Linear arithmetic
    - Simple conditions on strings
    - First order logics
  - The search problem is not completely defined
  - Path not known, but just constraints on some branches
  - Huge search space (constraints on strings)
  - Long execution time

London – March, 22nd-23rd 2012 – 18th CREST Open Workshop
Comparison

- **GA**: 70 individuals, 500 generations (elitist), $P_m=0.01$, $P_c=0.7$
- **Solver**: Yces sat solver (integer, float, bit-vectors) + apache module to track symbolic values and path constraints
- **Combination**: no improvements after 50 generations, switch to solver, and then back to GA
- **Sanity check**: 50,000 random tests (input names taken from the source code)
Empirical results

- Case study: Yapig 0.95-b
  - Php + MySql
  - 53 files, 9 kloc

- Static analysis reports 53 candidate vulnerabilities
  - Including infeasible paths
Considerations

- GA alone does not solve our problem
- Solver alone goes more near to the solution, but it takes a lot of computational time
- Combination: GA is a fundamental to solve the problem and speed up the generation security test cases
  - Search space is usually very large but GA heuristic helps in reducing it (global search)
  - With the reduced search space, resorting to a constraint solver does not create scalability issues (local search)
Parallel populations on evolution (with different parameters)
Parallel execution of alternative strategies
Parallel execution of resource-demanding parts of combined algorithms (e.g., the solver)
Conclusions

- Static analysis can be used to help manual review (candidate vulnerable points)
- We combined a genetic algorithm and a constraint solver to generate security test cases (input values)
- On a case study application, the combination
  - Produced better results (coverage)
  - Improved performances (computation time)
- Parallel architecture are a viable solution to improve further on combined algorithms