Finding Your Way in Testing Jungle

A Learning Approach to Web Security Testing.

Research Questions

- Why is it important to improve website security?
- What techniques are already in place to test security?
- What are the benefits of a learning based web testing approach?

Why is it important to improve web security?

- Studies show web apps highly vulnerable to security attacks
  - report by WASC lists 97,554 detected vulnerabilities
  - 49% of sites contain high-risk vulnerabilities
- Black-box security testing of web apps is a hard problem
  - only able to find 58.5% of high-risk vulnerabilities and 12.1% of medium risk
- Cross-site scripting (XSS)
  - one of top two web vulnerabilities

Learning Approach to Testing

- Fresh approach - cast into learning setting
- Testing algorithm has large database of test payloads
- If web app's defenses are broken, one of these payloads is able to demonstrate the vulnerability
- Question: how do we search through payloads to find a good candidate?

XSS Analyzer

- Learning algorithm for black-box detection of XSS vulnerabilities
- 500 million test payloads
- Infers from failed tests which other payloads are also likely to fail and prunes the search space

Example - PHP Sanitizer

```php
<?php function filter($str) {
  $str = stripslashes($str);
  $str = eregi_replace("<[[:space:]]*([^ >]*)[[:space:]]*>",'<\\1>', $str);
  $str = eregi_replace("<a[^ >]*href[[:space:]]*=[[:space:]]*\'?[[:space:]]*([^ \' >]*)[[:space:]]*\"?[^ >]*>",'<a href="\\1">', $str);
  $str = eregi_replace("<[[:space:]]* img[[:space:]]*([^ >]*)[[:space:]]*>", '', $str);
  $str = eregi_replace("<a[^ >]*href[[:space:]]*=[[:space:]]*\'?javascript[[:punct:]]*\"?[^ >]*>", '', $str);   $tmp = "";
  while (ereg("<(/?[[:alpha:]]*)[[:space:]]*([^ >]*)>",$str,$reg)) {
    $i = strpos($str,$reg[0]); $l = strlen($reg[0]); $tmp .= substr($str,0,$i); $str = substr($str,$i+$l); }
  $str = $tmp .  $str;
  $str = htmlentities(trim($str), ENT_QUOTES);
  if ($type != "preview" AND $save != 1) { $str = html_entity_decode($str, ENT_QUOTES); }
  return $str }
?>
```

Example - PHP Sanitizer

- *filter* function sanitizes its input string
  - deletes all spaces from HTML tags
  - deletes *img* tags
  - deletes javascript directives contained in *href* attribute values
  - deletes all tags with no attributes
- This permits only <a> tags with *href* values

Example - PHP Sanitizer

- <script>alert(‘XSS’)</script>
- Fails due to removal of tags with no attributes
- Problem with *filter* : user can replace space character with tab character (\t)
- <input autofocus onfocus=’alert(‘XSS’)>
  - can penetrate through *filter’s* defense

How XSS Analyzer works

1. Sends basic payload, <script>alert(1)</ script>
   - fails, XSS analyzer learns nothing
2. Sends another:

```
<input type=”text” onfocus=”alert(‘XSS’)”/>
```

   - fails, but teaches XSS Analyzer a constraint, *filter* rejects spaces

3. Sends one with tab character

```
<input autofocus     onfocus=”alert(‘XSS’)”>
```

   - this will demonstrate vulnerability

What techniques are already in place to test security?

- Black Box Security Testing
- Brute Force Testing
- Random Testing
- Expert Testing

Brute-Force Testing

- Accepts as input list of L payloads and iterates over it trying each payload
- Optimal from coverage standpoint, ensures 0 misses with respect to available payloads
- High cost of HTTP traffic restricts number of payloads that can be spent on a given input

Random Testing

- Parameterized by list of payloads L and sample size n
- n payloads sampled at random are tried with brute-force algorithm
- Advantage: prevents biases such as giving more weight to payloads in beginning of list
- Disadvantages:
  - interconnections between payloads ignored
  - random testing not reproducible in general

Expert Testing

- Rely on expert knowledge when making short list of payloads
- Works really well for “average case” defects
- Non-standard or uncommon defects, which are most dangerous, are outside of reach

The Learning Approach

- Using fingerprinting, we test for structural and bypass attacks
- Fingerprints are defined by a certain word being used in the attack
- Bypassing attacks
- Structural attacks

What does the XSS Analyzer actually do?

- Tests to find vulnerabilities in the webpage
- Uses probes to ensure certain attacks are prevented
- Sees where tests fail, and gathers tokens
- tl;dr its testing the sanitizer for allowing malicious tokens through

Important Concepts

Test Fails: The sanitizer blocks the payload

Payload: Set of tokens that represents a client side injection

Token: a word in a script, examples being ‘onmouseover’ or ‘=’

Sanitizer: removes words from payload

Structural XSS Analyzer

- When a test fails, the analyzer parses the payload to find out what happened

```
for each token
  send to website
  if token is not accepted
        add it to the constraint list
```

Bypass Strategy

- In addition to structural constraints, the analyzer checks bypass strategies
- A bypass strategy is if someone is using a similar word that could become a word used in a XSS attack
- These words are added to a mapping, so the analyzer knows what the word maps back to

Bypass Example

`<script>`

- Test fails, but now XSS Analyzer knows that script is a constraint

`<SCscriptRIPT>`

- Test passes, uncovers a flaw in the filter which allows the bypass SCscriptRIPT to be used

This all works together

Create a new set of constraints, structural and bypass

while there are more payloads

traverse tokens

replace bypass tokens with mapping

test if modified payload is blocked by sanitizer

if blocked :

run structural analysis

for each token sanitized, try bypass mapping

if accepted : try next payload

What are the benefits of a learning based web testing approach?

- Better
- Quicker
- More awesome

Testing Experiment

XSS Analyzer, along with 21 other testing alternatives were compared with one another

- 15,552 different server side defenses
- defenses contain wide rang of sanitization and validation strategies

Performance and Coverage

Coverage: measured by the total number of vulnerabilities detected

Performance: average the number of requests sent by the testing algorithm in total

Analysis of Results

Two issues addressed: (1) overall value of each algorithm and (2) viability of random algorithr

Algorithm	Vuln.s	Coverage	Requests (avg.)		
			Total	Success	Failure
brute force	10356	100%	2301	95	6481
analyzer	10245	99%	10	6	18
AppScan	4406	43%	40	40	40
R100	5659	55%	67	3	106
R200	6311	61%	123	6	209
R250	6591	64%	148	8	260
R300	6725	65%	174	9	311
R400	7021	68%	219	12	411
R500	7291	70%	263	15	511
R600	7338	71%	312	17	613
R700	7567	73%	349	20	710
R800	7620	74%	396	21	811
R900	7725	75%	437	24	910
R1000	7741	75%	481	25	1010
R1500	8046	78%	681	34	1500
R2000	8229	79%	860	42	1983
R2500	8294	80%	1062	51	2472
R3000	8378	81%	1235	56	2953
R3500	8569	83%	1385	66	3421
R4000	8611	83%	1554	72	3896
R4500	8660	84%	1724	77	4371
R5000	8713	84%	1890	88	4845

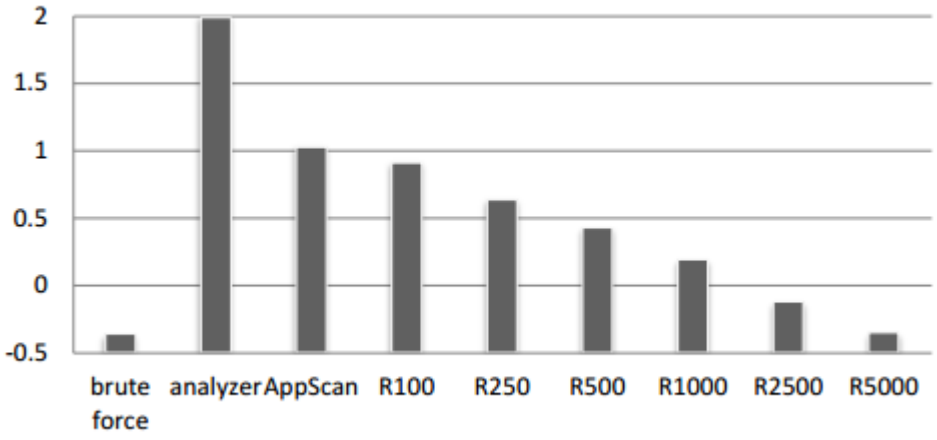
Overall Value

Measuring the effectiveness of each of the testing algorithms by computing a normalized ratio between the total number of detected vulnerabilities, v, and the total number of requests r

- log(10^3*(v/r))

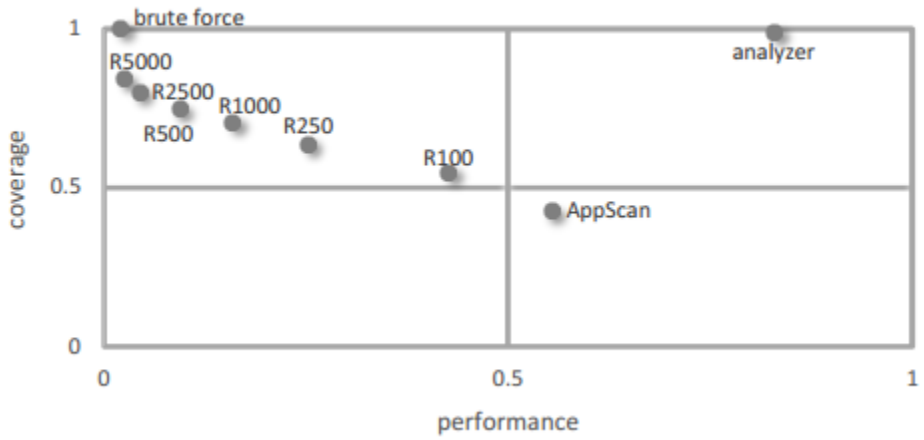
Overall Value

Normalized ratio between findings and sent requests



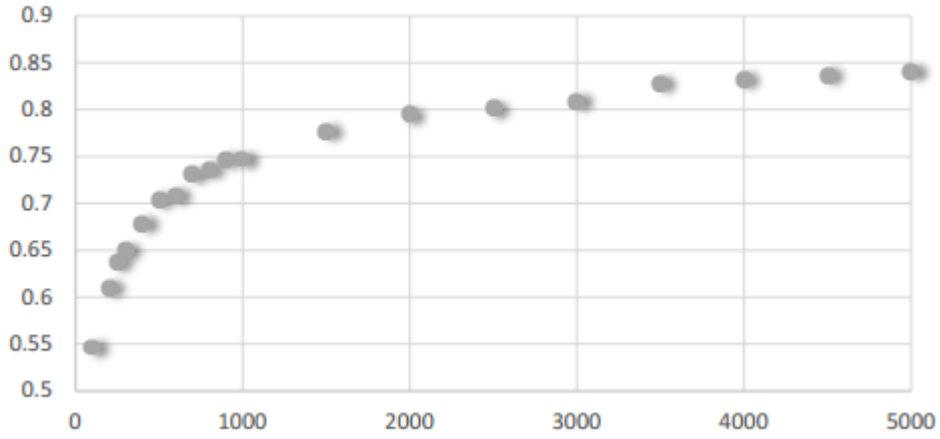
Overall Value

Coverage vs Performance



Random Testing

The rate at which coverage improves decays significantly



In conclusion..

- XSS Analyzer is being enhanced
  - Glass box
  - Address more issues
- Developed by IBM, learn more about it at their blogspot!

tinyurl.com/xssanalyzer

Questions

What are the benefits of a learning based web testing approach?

Why is R100 considered to be wasteful?

How does the XSS Analyzer complement the sanitizer?

Questions

How could the learning technique be applied to non-web based programs?

How may this approach be data heavy?

Why is black-box security testing generally a hard problem?