Security Testing

Eileen Donlon
CMSC 737 Spring 2008
Testing for Security

- Functional tests
  - Testing that role based security functions correctly

- Vulnerability scanning and penetration tests
  - Testing whether there are any flaws in the application or configuration that leave the system vulnerable to attack
Role Based Security
Table 2a – MyReferences experimental results.

<table>
<thead>
<tr>
<th>Fault Types</th>
<th># Faults</th>
<th>Scanner 1 XSS</th>
<th>Scanner 1 SQL Inject.</th>
<th>Scanner 2 XSS</th>
<th>Scanner 2 SQL Inject.</th>
<th>Scanner 3 XSS</th>
<th>Scanner 3 SQL Inject.</th>
<th>sum of the distinct vulnerabilities found by scanners</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Fault Injected</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>2</td>
<td>14 (9%)</td>
</tr>
<tr>
<td>MIFS</td>
<td>23</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2 (9%)</td>
</tr>
<tr>
<td>MFC</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>MFC extended</td>
<td>71</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>16</td>
<td>6</td>
<td>36</td>
<td>59 (83%)</td>
</tr>
<tr>
<td>MLAC</td>
<td>48</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2 (4%)</td>
</tr>
<tr>
<td>MIA</td>
<td>55</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>5 (27%)</td>
</tr>
<tr>
<td>MLPC</td>
<td>97</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>MVAE</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>WLEC</td>
<td>76</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>7 (25%)</td>
</tr>
<tr>
<td>WVAV</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>MVI</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>MVAV</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>WAEP</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>WPFV</td>
<td>148</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>2</td>
<td>19 (14%)</td>
</tr>
<tr>
<td>Total (injected)</td>
<td>659</td>
<td>25</td>
<td>33</td>
<td>8</td>
<td>21</td>
<td>19</td>
<td>66</td>
<td>49 (18%)</td>
</tr>
</tbody>
</table>
Bypass Testing of Web Applications

Offutt, Wu, Du, and Huang
ISSRE, Nov 2004
Bypass Testing

Bypass client side input validation in order to create tests for web application robustness and security

- Allows automated test execution
- Provides access to hidden form fields
SQL Injection Attack

• Insertion of SQL statements into web applications in order to force a database to modify the database in an unintended way, or to return inappropriate data or to produce an error that reveals database access information.
  – Web forms
  – Web services

• Two factors required:
  – The SQL statement is run in the context of a user with sufficient privileges to execute the attack.
  – Dynamic SQL
Database Security

• Stored Procedures and views can be used to enhance security because permissions to access a view can be granted, denied, or revoked, regardless of the set of permissions to access the underlying table(s).
• Stored procedures and views can be used to conceal the underlying data objects.
• By using stored procedures and view, you can limit the data that is available to a user to a restricted set of the columns and rows instead of querying the entire table.
• This does not apply when you use dynamic SQL! Dynamic SQL involves checking permissions on all data objects used in the query.
SQL Injection Attack

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR - DID HE BREAK SOMETHING? IN A WAY -

DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
Types of Client Side Validation

• Semantic Validation

• Syntactic Validation
Semantic Input Validation

• Data type conversion
  – Convert strings to integers

• Data format validation
  – Phone numbers, currency, email addresses

• Inter-value constraints
  – Credit card number and expiration date
Syntactic Input Validation

• Built-in length restriction
• Built-in value restriction
  – Pick lists
• Built-in transfer mode
  – HTTP GET or POST
• Built-in data access
  – Hidden Form Fields
  – Cookies
Syntactic Input Validation

• Built-in field selection
  – Pre-defined fields, enabled/disabled
• Built-in control flow restriction
  – Action attributes in FORM tags, links
Server Input Validation

- Numeric limits
- Email addresses
  - Username and valid domain
- URLs
  - Valid form, exist
- Character Patterns
  - Regular expressions
- Character filters
<table>
<thead>
<tr>
<th>Illegal Character</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty String</td>
<td></td>
</tr>
<tr>
<td>Commas</td>
<td>,</td>
</tr>
<tr>
<td>Directory paths</td>
<td>.. ../</td>
</tr>
<tr>
<td>Strings starting with forward slash</td>
<td>/</td>
</tr>
<tr>
<td>Strings starting with a period</td>
<td>.</td>
</tr>
<tr>
<td>Ampersands</td>
<td>&amp;</td>
</tr>
<tr>
<td>Control character</td>
<td>NIL, newline</td>
</tr>
<tr>
<td>Characters with high bit set</td>
<td>decimal 254 and 255</td>
</tr>
<tr>
<td>XML tag characters</td>
<td>&lt;, &gt;</td>
</tr>
</tbody>
</table>

Table 1. Characters that sometimes cause problems for Web applications
Feasibility Study

Can bypass testing be used successfully to test real web applications?

• Cyber Chair, paper submission and reviewing open source web application used by ISSRE

• Black box approach

• Valid user id and access code to enter, saved web pages and modified for bypass testing
Feasibility Study Results

• Submission without authentication
  – Changed action from relative url to complete url
• Unsafe use of hidden field
  – Changed hidden user id field
• Disclosing information
  – Error messages on removing hidden user id field
• No validation for parameter constraint
  – Mismatch between actual and specified file types
• No data type or value validation
  – Negative values, non-integers, etc. as page count
How to do Bypass Testing?

• Static or dynamic web pages
• Possibly multiple forms per page
  – Amazon’s web page had 20 forms and 169 hyperlinks

• Bottom line:
  – Automated input validation needed
  – Facilitated by formal model for html inputs
Model of HTML Input

Input Unit IU = (S, D, T)

S = Server

D = set of ordered pairs (n, v), where n is the name and v is the set of values that can be assigned to n

T = Transfer mode (HTTP GET or POST)
Model of HTML Input

Types of IU

• Form
  • $S =$ Action attribute of Form tag
  • $D =$ Form fields
  • $T =$ Method attribute of Form tag

• Link
  – An anchor `<a href="prog?val=1">`
    • $S =$ Static html or server program
    • $D =$ Query string
    • $T =$ GET
Composing Input Units

Redundancy on dynamic pages is eliminated through 3 composition rules:

1. Identical IU composition:
   • Two IUs $iu_1 = (S_1, D_1, T_1)$, $iu_2 = (S_2, D_2, T_2)$, are identical IFF $S_1 = S_2$, $D_1 = D_2$, and $T_1 = T_2$.
   • Two identical IUs are merged to form one IU $iu = (S_1, D_1, T_1)$. 
Composing Input Units

2. Optional input element composition:

• Two IUs \( iu_1 = (S_1, D_1, T_1) \), \( iu_2 = (S_2, D_2, T_2) \), have optional elements if \( S_1 = S_2 \), \( T_1 = T_2 \), and one input has an element name that is not in the other.

• The two IUs are merged to form one IU \( iu = (S_1, D', T_1) \), where \( D' = \{D_1 \cup D_2\} \)
Composing Input Units

3. Optional input value composition:
   • Two IUs $iu_1 = (S_1, D_1, T_1), iu_2 = (S_2, D_2, T_2)$, have optional elements if $S_1 = S_2, T_1 = T_2$, and there exists $(n_1, v_1) \in D_1, (n_2, v_2) \in D_2$ such that $n_1 = n_2$, but $v_1 \neq v_2$
   • The two IUs are merged to form one IU $iu = (S_1, D', T_1)$, where $D' = \{D_1 - (n_1, v_1)\} \cup \{D_1 - (n_2, v_2)\} \cup \{(n_1, (v_1 \cup v_2))\}$
Bypass Testing

• Value Level
  – Addresses data type conversion, data value validation, and built-in value restriction
  – For each input, generate invalid values according to the 14 types of input validation (client + server)

• Examples
  – Modify select to return undefined values
  – Violate value length restriction
Bypass Testing

• Parameter Level
  – Addresses built-in parameter selection, built-in data access, and inter-value constraints
  – Execute test cases that violate restrictive relationships among parameters
  – Parameter relationships are hard to identify
    • Invalid pair
    • Required pair
Parameter Level Bypass Testing

Algorithm: Identify input patterns of web applications
Input: The start page of a web application, $S$
Output: Identifiable input patterns

Step 1: Create a stack $ST$ to retain all input units that need to be explored. Initialize $ST$ to $S$. Create a set $IUS$ to retain all input units that have been identified. Initialize $IUS$ to empty.
Step 2: While $ST$ is not empty, pop an input unit (defined in Section 3) from $ST$, generate data for the input unit and send it to the server. When a reply is returned, analyze the HTML content. For each input unit $iu$:

- if $iu$ is a link input unit, and $iu$ does not belong to a different server, do not push $iu$ onto the stack.
- if $iu \in IUS$ (it has already been found), do not push $iu$ onto the stack.
- if there exists an input unit $iu' \in IUS$ such that $iu$ and $iu'$ have optional input elements, update the possible value of $iu$. Do not push $iu$ onto the stack.
- Otherwise, a new input pattern has been identified; add $iu$ to $IUS$ as an optional input unit, and then push $iu$ onto $ST$. 
Parameter Level Bypass Testing

Results of applying the algorithm are:

• Collection of IUs where $D = \{P_1, P_2, ..., P_k\}$ and $P_i = \{(n_1,v_1)_i, (n_2,v_2)_i, ..., (n_a,v_a)_i\}$. Each $P_i$ is a valid input pattern for the IU.

• Generate invalid input patterns using values from the set of valid values

• Goal is testing relationships among parameters
Parameter Level Bypass Testing

Three types of invalid input patterns:

• Empty input pattern
  – Submits no data
  – Violates all required pairs

• Universal input pattern
  – Submits values for all known parameters
  – Violates all invalid pairs

• Differential input pattern
  – Appropriate values for all parameters in an input pattern + a value for one parameter not in the input pattern
Bypass Testing

Third level is Control Flow Bypass Testing

- Execute test cases that break the normal execution sequence
  - Backward and forward control flow alteration
    - Reverse the order of a transition between 2 UIs
  - Arbitrary control flow alteration
Evaluation

• Small Text Information System (STIS)
  – Mysql database
  – 17 Java server pages, 8 of which process parameterized requests
• 3 Response Types:
  – Invalid inputs recognized and handled
  – Invalid inputs not recognized, abnormal server behavior handled
  – Invalid inputs not recognized, abnormal server behavior exposed to users
Table 2. Failures found for each dynamic component

I: Value Level, No Parameter or Control
II: Parameter Level, No Control Level
III: Control Level, No Parameter Level
IV: Parameter Level and Control Level

T = number of tests, F = number of failures

<table>
<thead>
<tr>
<th>Component</th>
<th>I</th>
<th></th>
<th>II</th>
<th></th>
<th>III</th>
<th></th>
<th>IV</th>
<th></th>
<th>Total</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>login</td>
<td>15</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>n/a</td>
<td></td>
<td>n/a</td>
<td></td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>browse</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>record_edit</td>
<td>17</td>
<td>9</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>28</td>
<td>17</td>
</tr>
<tr>
<td>record_delete</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>record_insert</td>
<td>13</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>categories</td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>category_edit</td>
<td>13</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>register_save</td>
<td>25</td>
<td>11</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>38</td>
<td>19</td>
</tr>
<tr>
<td>Total (#tests &amp; #failures)</td>
<td>107</td>
<td>37</td>
<td>23</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>21</td>
<td>17</td>
<td>158</td>
<td>66</td>
</tr>
</tbody>
</table>
Results

• Only 55 of 158 tests could have been executed without using bypass testing
  – 9 failures (of 66 total) from these 55 tests
Contributions

• Introduces Bypass testing
• Detailed model for choosing inputs to server side components
• Model supports general input validation testing, and rules are defined for bypass and input validation
• Empirical results from open source conference management system and home grown web apps
Conclusions

• Bypass testing is a novel technique for web application test case generation
• Approach requires no back end source code, only what’s received by a browser
• Complexity of inputs on dynamically generated web forms was handled by the algorithm presented
• Future work: automated form analysis and generation of bypass tests