CSCI 362 – Software Engineering

Team Project

Trotting Giraffes

Steven Aldinger
Michael Stenhouse
Marta Pancaldi
Seth Stoudenmier

COLLEGE of CHARLESTON

Summary

- Project specifications
- OWASP Java HTML Sanitizer – Overview
- Testing Framework – Test Cases
- Script
- Driver
- Fault Injections
- Results & Output
- Reflections

Project Specifications

- Development of an **automated testing framework** for the software project "OWASP Java HTML Sanitizer"
- 25 test cases
- Specification text file for each test case
- Run all tests through a single script
- Collect results and display testing report
- Build and compile on a Linux distributions (in our case, RHEL and Ubuntu)

OWASP Java HTML Sanitizer Overview

- Java-based sanitization of untrusted HTML: prevent malicious HTML code from being injected into a web application
- Organize poorly written code
- Protection against XSS (Cross-site Scripting)
- Extensive test suite

Sample code from one of the OWASP existing tests →

```
    Took 9 ms
    Took 10 ms
    Took 7 ms
    Took 12 ms
    Took 10 ms
    Took 6 ms
About to scan: http://deadspin.com/ size: 229390
    Took 44 ms
    Took 12 ms
    Took 13 ms
    Took 14 ms
    Took 15 ms
    Took 14 ms
    Took 16 ms
    Took 14 ms
    Took 12 ms
    Took 15 ms
    Took 14 ms
    Took 17 ms
    Took 17 ms
    Took 10 ms
    Took 13 ms
Total time ms: 1292
Average time per rep ms: 86
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
Running org.owasp.html.CssGrammarTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec
Running org.owasp.html.CssFuzzerTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.287 sec
Running org.owasp.html.UrlTextExampleTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
Running org.owasp.html.StylingPolicyTest
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec
Running org.owasp.html.ElementPolicyTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
Running org.owasp.html.SanitizersTest
Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.023 sec
Running org.owasp.html.EncodingTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec
```

```java
package org.owasp.html;

import org.junit.Test;

import junit.framework.TestCase;

@SuppressWarnings("javadoc")
public final class EncodingTest extends TestCase {

  @Test
  public static final void testDecodeHtml() {
    String html =
      "The quick brown fox&#xa;jumps over&#xd;&#10;the lazy dog&#x000a;";
    //          1         2         3         4         5         6
    // 123456789012345678901234567890123456789012345678901234567890123456789
    String golden =
      "The quick\u00a0brown fox\njumps over\r\nthe lazy dog\n";
    assertEquals(golden, Encoding.decodeHtml(html));

    // Don't allocate a new string when no entities.
    assertSame(golden, Encoding.decodeHtml(golden));

    // test interrupted escapes and escapes at end of file handled gracefully
    assertEquals(
        "\\\\u000a",
        Encoding.decodeHtml("\\\\u000a"));
    assertEquals(
        "\n",
        Encoding.decodeHtml("&#x000a;"));
    assertEquals(
        "\n",
        Encoding.decodeHtml("&#x00a;"));
    assertEquals(
        "\n",
        Encoding.decodeHtml("&#x0a;"));
    assertEquals(
        "\n",
        Encoding.decodeHtml("&#xa;"));
    assertEquals(
```

← Successful output of
$ mvn clean install
to test and build the HTML sanitizer

Our test plan: **Tested Items**

- Test if cssContent method can interpret '\\\\' correctly as \\
- Test if decodeHtml can properly interpret a Unicode carriage return
- Test if decodeHtml can properly interpret a String following a Unicode symbol
- Test if sanitize can block any tag with the Blocks Policy
- Test if sanitize will keep formatting tags with the FORMATTING Policy

Testing Framework – 1. **Test Cases**

- `/testCases` directory includes all test case *.txt files
- Sample Test case file:

```
1. 01 ##test number or ID
2. decodeHtml handles HTML entities to produce a string containing
only valid unicode scalar values ##requirement being tested
3. orgowasphtmlEncoding ##component being tested
4. decodeHtml ##method being tested
5. "&#x000a;" ##command-line arguments: test input
6. \\n ##expected outcomes (\\n in Java means \n in the bash script)
```

- runAllTest script reads each file and executes the testing driver, according to the information in the file

Examples of Test Cases

- Test #01: decodeHtml(String)
- Test #02: decodeHtml(String)
- Test #08: cssContent(String)
- Test #17: FORMATTING.sanitize(String)
- Test #21: BLOCKS.sanitize(String)

```
Argument: &#x000a
Oracle: \n
```

```
Argument: &#xa
Oracle: \n
```

```
Argument: t\\61ser
Oracle: taser
```

```
Argument: Hello,World!
Oracle: Hello,World!
```

```
Argument: Hello,World!
Oracle: Hello,World!
```

Testing Framework – 2. **Script**

- *./runAllTests.sh* script automatically runs all tests

```
TESTCASES="../testCases”                                    #TestCases Directory
TESTCASESEXEC="../testCasesExecutables” #Driver Directory
```

- The script compiles OWASP project files,
- then iterates through each text file within /TestCases directory,
- pulls information and runs the matching java driver

```
ls $TESTCASES | while read FILENAME
do
        echo "Running testcase executable based off of  $FILENAME
specifications:”
#Iterate through the individual text files and grab the information line-
by-line
        COUNTER=0
        cat "$TESTCASES"/"$FILENAME" | while read TESTCASESLINE
        do
                let COUNTER=(COUNTER + 1)
...
                if [ $COUNTER -eq 1 ] #1st line indicates test number
                then
                        TESTNUMBER=$CHOPPED
                fi
...
                if [ $COUNTER -eq 5 ] #5th line indicates the input value
to be inserted into the method
                then
                        TEST=$CHOPPED
                fi

                #If sixth line, this indicates the oracle, which is the
expected result returned from the tested method. At this step we also run
the driver
                if [ $COUNTER -eq 6 ]
                then
                        ORACLE=$CHOPPED
...
```

Testing Framework – 3. **Driver**

- `runAllTests` calls the driver that matches the information pulled from test case files – test number (e.g. `01`) , class (`htmlEncoding`) and method (`decodeHtml`):

```
public static void main(String[] args){
        try{
                String theOracle = args[1];
                String theTest = args[0];

                String result =
org.owasp.html.Encoding.decodeHtml(theTest);

#Replace "\\n" with "\n", since Java will interpret the HTML code
as a line break
                if(theOracle.contains("\\n")){
                                theOracle =
theOracle.replace("\\n","\n");
```

```
if((result).equals(theOracle)){
                try{

        System.out.println("org.owasp.html.Encoding");
                        System.out.println("decodeHtml(String)");
                        System.out.println(args[0]);
                        System.out.println(args[1]);
                        System.out.println("passed\n");
                }catch(Exception e){
                        e.printStackTrace();
                }
}
        else{
                try{
...
                        System.out.println("failed\n");

                }catch(Exception e){
                        e.printStackTrace();
                }
```

- If `result` equals `theOracle` the test has passed.
- The results are saved to `/reports` directory

Testing Framework – 4. **Faults Injection**

- Fault injection = modify the code we are testing to **make some test fail** and collect results → improve robustness

```
public static String cssContent(String token) {
  int n = token.length();
  int pos = 0;
  StringBuilder sb = null;
  if (n >= 2) {
    char ch0 = token.charAt(0);
    if (ch0 == '"' && ch0 == '\'') { // FAULT: Changed an || to an &&
    //if (ch0 == '"' || ch0 == '\'') {
      if (ch0 == token.charAt(n - 1)) {
        pos = 1;
        --n;
        sb = new StringBuilder(n);
```

'foo'	'foo'	foo	failed
\61zimuth	azimuth	azimuth	passed
t\61ser	taser	taser	passed
foo	foo	foo	passed
\"foo\"	\"foo\"	foo	failed
'	'	'	passed
\"	\"	\"	passed
\"\22\22\"	\"\"\"\"	\"\"	failed
\22\22	\"\"	\"\"	passed
'\\'	'\'	\	failed
'\a'	'a'	\n	failed

Testing Framework – 5. **Results**

- Finally the script creates a HTML file with all test results, which is automatically displayed

```
HTMLOUTPUTFILE="../reports/myResults.html”
cd ../scripts
(./toCompiledHtmlWithCss.sh $HTMLOUTPUTFILE ../reports
                                    && firefox -new-tab
./$HTMLOUTPUTFILE)
```

- Script *toCompiledHtmlWithCss* builds a table with result data from */reports* directory
- The table includes tested class and method, test input, oracle and actual result (passed/failed)

Output

```
martap@ubuntu:~/TrottingGiraffes/TestAutomation/scripts$ ./runAllTests.sh
Note: ./src/main/java/org/owasp/html/CssSchema.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Running testcase executable based off of testCase01.txt specifications:

01
decodeHtml handles HTML entities to produce a string containing only valid unicode scalar v
alues
orgowasphtmlEncoding
decodeHtml
"&#x000a;"
"\n"


Running testcase executable based off of testCase02.txt specifications:

02
decodeHtml handles HTML entities to produce a string containing only valid unicode scalar v
alues
orgowasphtmlEncoding
decodeHtml
"&#xa;"
"\n"


Running testcase executable based off of testCase03.txt specif

03
decodeHtml handles HTML entities to produce a string containing
alues
orgowasphtmlEncoding
decodeHtml
"&#x00ziggy"
"&#x00ziggy"


Running testcase executable based off of testCase04.txt specif
```

← Terminal output

HTML file with results →

Mozilla Firefox

file:///home...esults.html ×

Open a new tab (Ctrl+T)

file:///home/martap/TrottingGiraffes/TestAutomation/reports/myResu

Search

Test Number	Class	Requirement	Method	Input	Oracle	Result
1	org.owasp.html.Encoding	org.owasp.html.Encoding	decodeHtml(String)	" "	"n"	passed
2	org.owasp.html.Encoding	org.owasp.html.Encoding	decodeHtml(String)	" "	"n"	passed
3	org.owasp.html.Encoding	org.owasp.html.Encoding	decodeHtml(String)	"&#x00ziggy"	"&#x00ziggy"	passed
4	org.owasp.html.Encoding	org.owasp.html.Encoding	decodeHtml(String)	" "	"n"	passed
5	org.owasp.html.Encoding	org.owasp.html.Encoding	decodeHtml(String)	" "	"n"	passed
6	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	foo	'foo'	failed
7	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	61zimuth	azimuth	passed
8	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	t61ser	taser	passed
9	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	foo	foo	passed
10	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	foo	"foo"	failed
11	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	'	'	passed
12	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	"	"	passed
13	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	""	"2222"	failed
14	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	cssContent(String)	2222	""	passed

Reflections – Challenges we faced

- Initial choice of the program to test
- Getting familiar with Linux operating system
- Design of the testing framework
- Displaying the test results in a clear way
- Organizing a coherent documentation
- Meeting between people with different schedules

Reflections – What we learned

- Sharing knowledge among team members
- Understanding the structure of a test framework
- Becoming familiar with new scripting languages

Thanks for watching!