
BUILDING A TESTING FRAMEWORK WITH OWASP JAVA HTML SANITIZER

Steven Aldinger

Marta Pancaldi

Seth Stoudenmier

Michael Stenhouse

Trotting Giraffes

CSCI 362-03

Dr. Jim Bowring

Fall 2015

Table of Contents

[Introduction](#)

- Background
- Choosing a Candidate
- The Testing Framework
- Installation

[Chapter One: Git Clone](#)

- Building the Project

[Chapter Two: Test Plan](#)

[Chapter Three: Testing Framework](#)

- TestCases.txt
- Drivers.java
- runAllTests.sh

[Chapter Four: Test Cases](#)

- Testing Other Methods
- Displaying Our Results
- New Drivers

[Chapter Five: Fault Injection](#)

- Faults
- Results

[Chapter Six: Final Thoughts](#)

- Overall Experiences

Introduction

Background

This report is the culmination of our group work in Dr. Jim Bowring's Fall 2015 Software Engineering course. We were assigned to choose an HFOSS (Humanitarian Free Open Source Software) project, clone it into our own GitHub repository, and begin building an automated testing framework for the project. This assignment was divided into five different deliverables which are collected and revised as chapters in this report. The majority of us were new to Git and Linux at the beginning of this project.

All of the components we created were built and compiled using Linux distributions, specifically RHEL and Ubuntu.

Choosing a Candidate

Our first task was to choose an HFOSS project. The only constraint was that the code base must be designed to compile and run on Linux. We began looking at a wiki of potential [HFOSS projects](#). We needed a project that was well documented and easy to test.

DHIS 2

A modular web-based tool for managing and visualizing aggregate statistical data. The software is primarily used for integrated health information management activities. DHIS' home page cites specific use cases such as "health program monitoring and evaluation, facility registries, service availability mapping, logistics management, and mobile tracking of pregnant mothers in rural communities." Built with open source Java frameworks. Active development. ([Link to project repository.](#))

Mifos X

A Java-based management information system for financial services centered on microfinance and financial inclusion. The Mifos Initiative states that their platform and open-source service model will "increase access to technology for all microfinance institutions, ultimately enabling them to extend their reach to the world's poor." Active development. ([Link to project repository.](#))

Color Reader

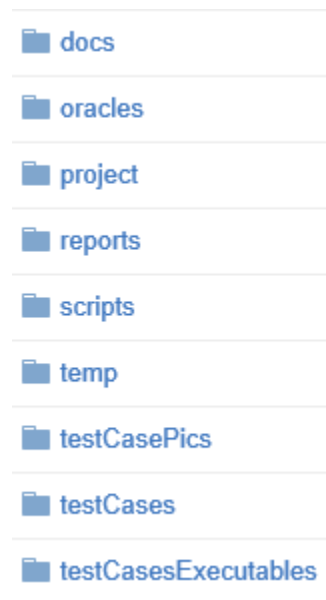
A Java application that "reads" color for those with color blindness. Within the interface, the user can click on a part of an image and the application will read the RGB values of nearby pixels. The RGB values are then converted into HSB (Hue Saturation Brightness) values. The name of the color is then output on the GUI via text and text-to-speech. A database is available with HSB values and their corresponding colors. Inactive development. ([Link to project repository.](#))

Between these three candidates, we chose Mifos X as it was frequently updated and our group is familiar with Java. However, we ran into trouble compiling the code. A web server was needed to run the service, so we changed our project to a standalone application. After researching and testing additional projects, we chose the OWASP Java HTML Sanitizer.

The OWASP Java HTML Sanitizer is described as "an efficient HtmlSanitizer configurable via a flexible HtmlPolicyBuilder". A method in the program named takes an HTML stream, and dispatches events to a policy object which will decide which elements and attributes to allow. In other words, the Sanitizer can take an input of HTML and output new HTML code without any undesirable tags that have been blacklisted by a Policy object. A user would be able to define their own policy for their own customizable white-list. The Sanitizer is open source and written in Java. ([Link to project repository.](#))

The Testing Framework

The directory structure for our framework is below.



Several test cases are in the /testCases folder. These .txt files are accessed by test case specification files, or test drivers. All test cases contain a [standard template](#) that anyone could follow to create their own test cases, given that a driver has been written for the method that is being tested.

A driver has been written for each method that has been tested. These [drivers](#) are written in Java and are found in the /testCasesExecutables/org/trotting folder. We have written drivers for four methods:

cssContent(String), decodeHtml(String), isHex(int), and sanitize(String, Policy).

These drivers are invoked by a [single script](#), runAllTests.sh. The script is written in Bash and found in the /scripts folder. Once the script has run all of our test cases through their respective drivers, the results are posted in the /reports folder as individual .txt files. A helper script, toCompiledHtmlWithCss.sh, echoes these results and formats them into an easy to read HTML table that will automatically open in the default browser.

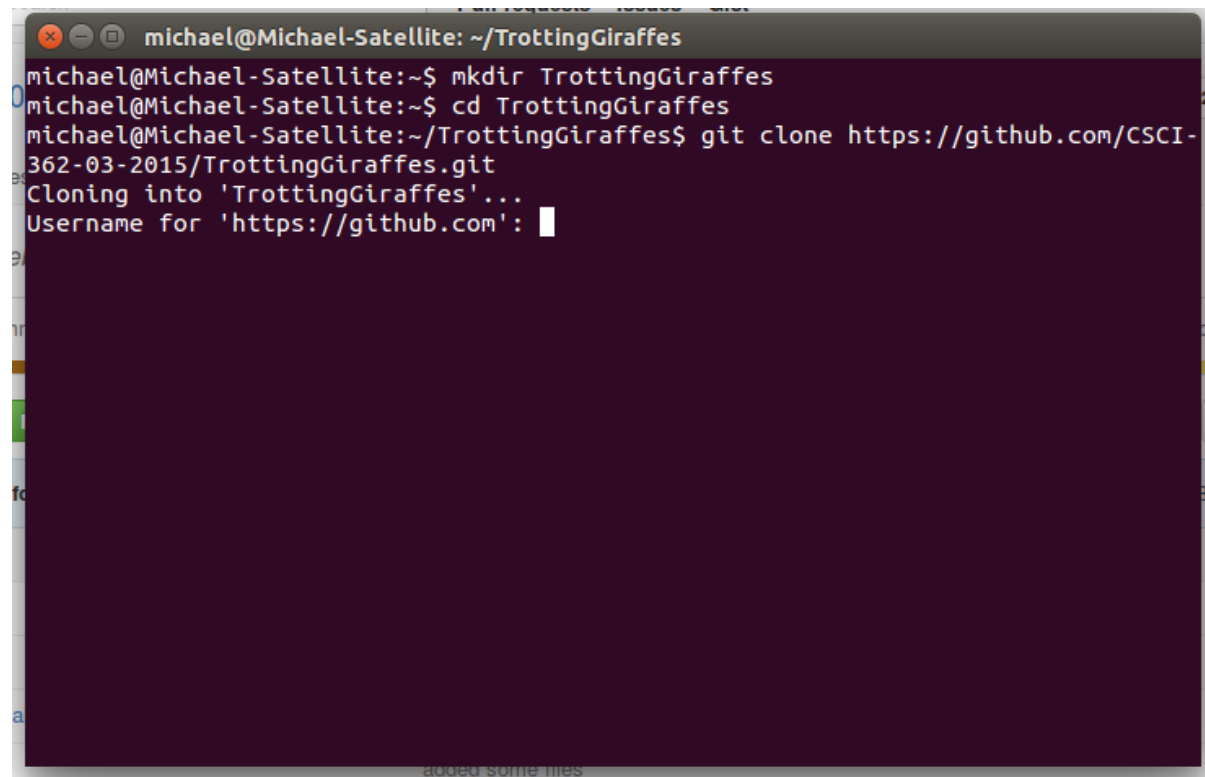
A README.txt is placed in the /docs folder. The original Sanitizer source code is placed within the /project folder. The HTML table is being saved to the /reports folder after the script has executed. Images displaying examples of our results are posted in /testCasePics.

Installation

Here we will detail how to install our testing framework on a fresh Ubuntu install. It is easy for anyone to add test cases to the test drivers that we have written. You could also add your own drivers to test additional methods found in the Sanitizer.

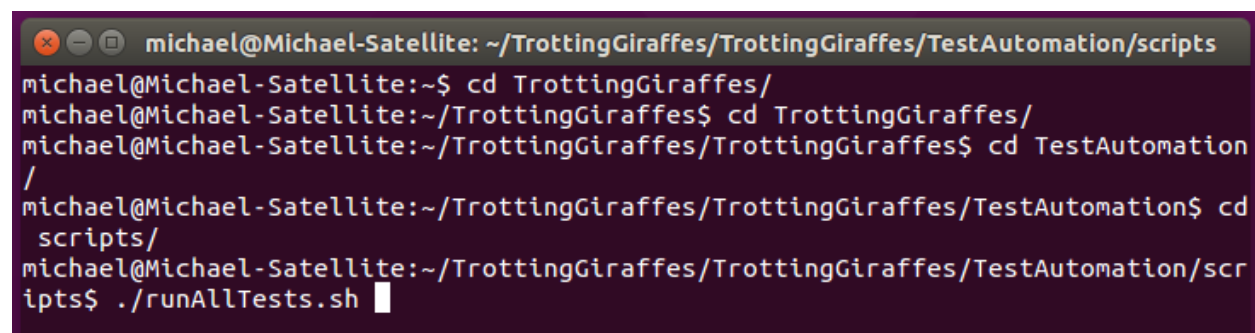
Before starting the framework installation, it is necessary to have Git and Java installed on your machine. Here are simple tutorials on how to install Git and Java via the command line: [\(Git installation.\)](#) [\(Java installation.\)](#)

Once Git and Java is set up, create a directory (for example, /TrottingGiraffes) and clone our repository onto your machine. At the screen below, you will enter in your GitHub username and password. (Our repository is private so we will assume that access has been granted to you before beginning this process.)

A terminal window titled 'michael@Michael-Satellite: ~/TrottingGiraffes' showing the following commands and output:

```
michael@Michael-Satellite:~$ mkdir TrottingGiraffes
michael@Michael-Satellite:~$ cd TrottingGiraffes
michael@Michael-Satellite:~/TrottingGiraffes$ git clone https://github.com/CSCI-362-03-2015/TrottingGiraffes.git
Cloning into 'TrottingGiraffes'...
Username for 'https://github.com':
```

Git will then download our project into the directory that you just created. Congratulations, you just cloned a Git project! You can now run the TrottingGiraffes test framework script by entering in the commands listed.

A terminal window titled 'michael@Michael-Satellite: ~/TrottingGiraffes/TrottingGiraffes/TestAutomation/scripts' showing the following commands and output:

```
michael@Michael-Satellite:~$ cd TrottingGiraffes/
michael@Michael-Satellite:~/TrottingGiraffes$ cd TrottingGiraffes/
michael@Michael-Satellite:~/TrottingGiraffes/TrottingGiraffes$ cd TestAutomation/
michael@Michael-Satellite:~/TrottingGiraffes/TrottingGiraffes/TestAutomation$ cd scripts/
michael@Michael-Satellite:~/TrottingGiraffes/TrottingGiraffes/TestAutomation/scripts$ ./runAllTests.sh
```

Chapter One: Git Clone

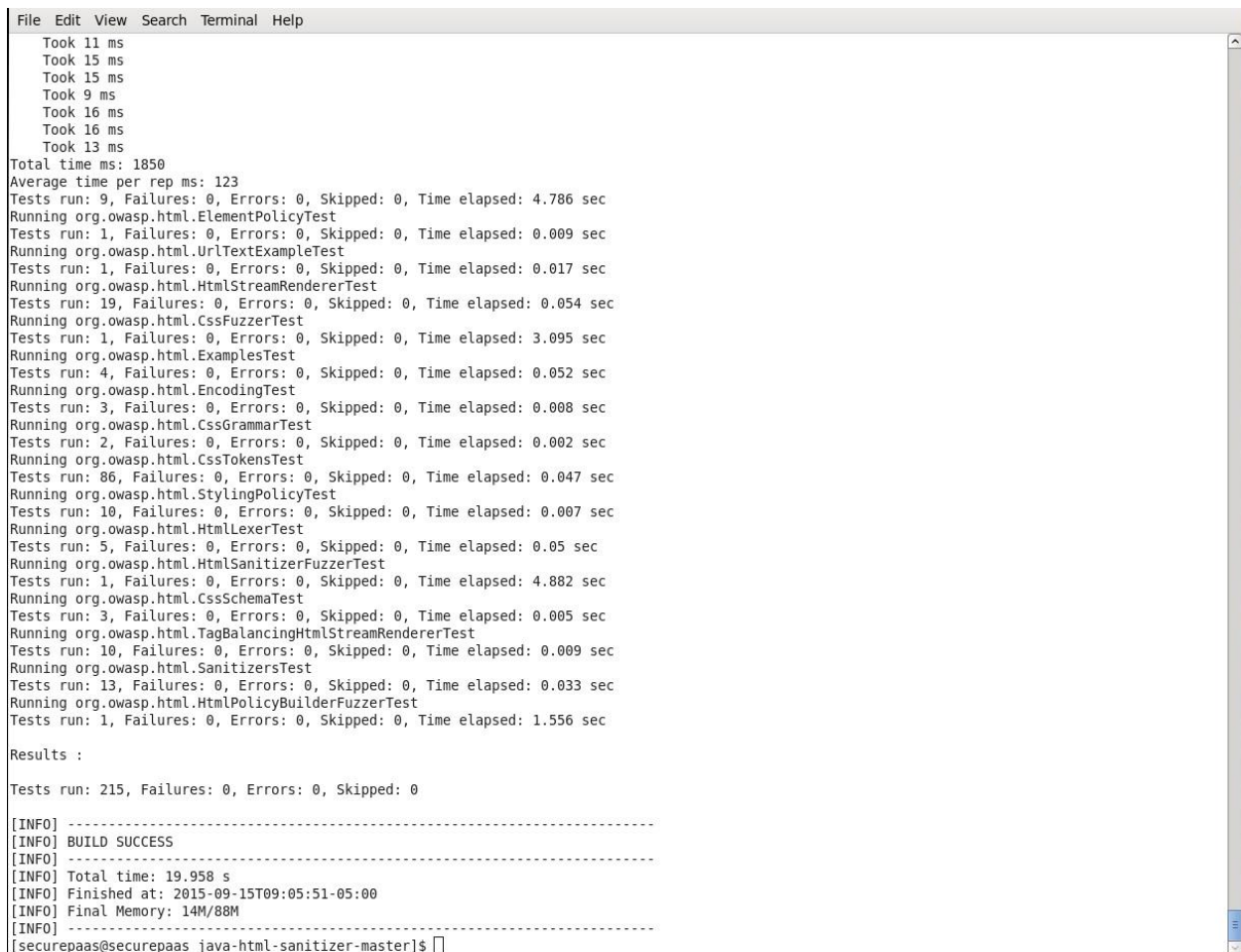
Task: Checkout or clone project from its repository and build it... Run existing tests and collect results. Evaluate experience and project so far.

Building the Project

Many of us are new to cloning and building projects via Git. It is interesting to see how much effort developers put into testing their software!

The Sanitizer included detailed instructions on how to get started. At the time we were building the Sanitizer with Maven. Originally, our script depended on Maven to test the project. We eventually removed the Maven dependencies from our scripts as they were unnecessary for the framework. With our current testing framework, we are using the standard Java compiler javac.

This image shows the output of a successful "mvn clean install" to test and build the HTML sanitizer.



```
File Edit View Search Terminal Help
Took 11 ms
Took 15 ms
Took 15 ms
Took 9 ms
Took 16 ms
Took 16 ms
Took 13 ms
Total time ms: 1850
Average time per rep ms: 123
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.786 sec
Running org.owasp.html.ElementPolicyTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec
Running org.owasp.html.UrlTextExampleTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.017 sec
Running org.owasp.html.HtmlStreamRendererTest
Tests run: 19, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.054 sec
Running org.owasp.html.CssFuzzerTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.095 sec
Running org.owasp.html.ExamplesTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.052 sec
Running org.owasp.html.EncodingTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.008 sec
Running org.owasp.html.CssGrammarTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec
Running org.owasp.html.CssTokensTest
Tests run: 86, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.047 sec
Running org.owasp.html.StylingPolicyTest
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec
Running org.owasp.html.HtmlLexerTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.05 sec
Running org.owasp.html.HtmlSanitizerFuzzerTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.882 sec
Running org.owasp.html.CssSchemaTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec
Running org.owasp.html.TagBalancingHtmlStreamRendererTest
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec
Running org.owasp.html.SanitizersTest
Tests run: 13, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.033 sec
Running org.owasp.html.HtmlPolicyBuilderFuzzerTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.556 sec
Results :
Tests run: 215, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 19.958 s
[INFO] Finished at: 2015-09-15T09:05:51-05:00
[INFO] Final Memory: 14M/88M
[INFO] -----
[securepaas@securepaas java-html-sanitizer-master]$
```

This image shows some example code from one of the tests. It is verifying that the html decoder is working properly.

```
File Edit View Search Terminal Help
// Copyright (c) 2012, Mike Samuel
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
// Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
// Neither the name of the OWASP nor the names of its contributors may
// be used to endorse or promote products derived from this software
// without specific prior written permission.
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
// COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
// BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
// LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
// CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
// LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
// ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.

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&nbsp;brown fox&#xa;jumps over&#xd;&#10;the lazy dog&#x000a;";
        //      1         2         3         4         5         6
        // 12345678901234567890123456789012345678901234567890123456789
        String golden =
            "The quick\u000a0brown fox\njumps over\n\nthe lazy dog\n";
        assertEquals(golden, Encoding.decodeHtml(html));

        // Don't allocate a new string when no entities.
        assertEquals(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(
            String.valueOf(Character.toChars(0x10000)),
            Encoding.decodeHtml("&#x10000;");
        assertEquals(
            "\n",
            Encoding.decodeHtml("&#xa");
        assertEquals(
            "&#x00ziggy",
            Encoding.decodeHtml("&#x00ziggy");
        assertEquals(
            "&#xa00z",
            Encoding.decodeHtml("&#xa00z;");
        assertEquals(
            "&#\n",
            Encoding.decodeHtml("&#&#x000a;");
        assertEquals(
            "&#\n",
            Encoding.decodeHtml("&#x&#x000a;");
        assertEquals(
            "\n\n",
            Encoding.decodeHtml("&#xa&#x000a;");
        assertEquals(
            "&#\n",
            Encoding.decodeHtml("&#&#xa;");
        -----
    }
}
```

Chapter Two: Test Plan

Task: Produce a detailed test plan for your project. As part of this plan, you will specify at least 5 of the eventual 25 test cases that you will develop for this software.

The Sanitizer is advertised as a program that can detect malicious HTML tags and remove them from input. We searched through the source code of the software and found a detailed testing framework that heavily influenced our test plan.

The first test plan:

1. Create a test around a new HTML document with several HTML tags.
2. Create a test around a new HTML document with a heavy amount of CSS styling.
3. Create a test around sanitizing the HTML source code of a popular website, such as Yahoo.
4. Create a test which reports all changes of HTML tags into a .txt file.
5. Create a test around creating a new policy for detecting certain HTML tags.

It turned out we misunderstood the assignment and created overly complex tests that would take far too much effort to implement. The idea of a testing framework is that anyone can add a simple test case to a folder and see whether it passes or fails.

After coming to a better understanding of our task, we decided to create a new test plan, detailed below.

A new test plan:

1. Test if `cssContent` method can interpret '\\\\' correctly as \\
2. Test if `decodeHtml` can properly interpret a Unicode carriage return
3. Test if `decodeHtml` can properly interpret a String following a Unicode symbol
4. Test if `sanitize` can block any tag with the BLOCKS Policy
5. Test if `sanitize` will keep formatting tags with the FORMATTING Policy

Chapter Three: Testing Framework

Task: Design and build an automated testing framework that you will use to implement your test plan.

Our framework directory was modeled after a [predetermined structure](#).

TestCases.txt

Each individual test case exists in its own separate .txt file. All test cases are found in the /testCases folder.

These .txt files contain a template for which we will follow when writing all future test cases.

1. ## test number or ID
2. ## requirement being tested
3. ## component being tested
4. ## method being tested
5. ## command-line arguments
6. ## expected outcomes

As stated in the previous chapter, the author of the Sanitizer included several test cases in the source code that we used for our own testing framework. We decided to test the decodeHtml method for our first five test cases. We tested if the method can handle HTML entities to produce a string containing only valid Unicode scalar values. One of our test cases is listed below.

1. 01##test number or ID
2. decodeHtml handles HTML entities to produce a string
3. orgowasp.html.Encoding##component being tested
4. decodeHtml##method being tested
5. "
"##command-line arguments
6. "\\n"##expected outcomes, only \n will actually be te

Drivers.java

Our test drivers are found in the /testCaseExecutables folder. Our first driver was built specifically to test the requirements of decodeHtml. Our driver will determine whether or not the expected outcome (theOracle) matches the actual outcome (result). In scenarios where we are testing HTML carriage returns \n, we wrote a conditional that will keep the test results from returning a blank return.

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

        String result = org.owasp.html.Encoding.decodeHtml(theTest);
        if(theOracle.contains("\\n")){
            //System.out.println("Oracle contains: \\n");
            theOracle = theOracle.replace("\\n","\n");
        }
    }
```

Below is an early version of our code that will run and print if our oracle matches the actual outcome.

```
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();
    }
}
```

If the oracle and actual outcome do not match, an else statement will print out that the test has failed. We would later update this driver to be more sophisticated, which we outline in the next chapter.

runAllTests.sh

We had to write a single script to invoke our testing framework and collect the results in a single file. The script accesses the directories of our test cases and test drivers.

```
#TestCases Directory which contains our test case text files
TESTCASES="../testCases"
#TestCasesExecutables directory which contains our java drivers for the methods to be tested
TESTCASESEXEC="../testCasesExecutables"
```

The script will then iterate through each text file within the /testCases directory. This will pull the information from the text file and run the matching java driver. We also have code that will delete the first three characters in each line and deletes comments marked at the end of a line with ##double hash.

```

ls $TESTCASES | while read FILENAME
do

    echo "Running testcase executable based off of $FILENAME specifications:"
    echo ""

    #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)

        #Get length of line string for formatting
        TESTCASESLINELENGTH=${#TESTCASESLINE}

        #Remove numbers at beginning of lines
        #For instance, if the line says 1. Hello, this will return Hello
        TESTCASESLINECHOPPED="${TESTCASESLINE:3:$TESTCASESLINELENGTH}"

        #Deletes any comments within the test case files, marked by ## (double hash)
        #Get position of double hash character (##)
        #Thanks to modified responses to:
        #http://stackoverflow.com/questions/20348097/bash-extract-string-before-a-colon

        CHARPOS=$(echo $TESTCASESLINECHOPPED | grep -b -o "##" | cut -d: -f1)

        #Get substring of the line with comment removed
        echo ${TESTCASESLINECHOPPED:0:$CHARPOS}
        CHOPPED=${TESTCASESLINECHOPPED:0:$CHARPOS}

        #If first line, this indicates the test number
        if [ $COUNTER -eq 1 ]
        then
            TESTNUMBER=$CHOPPED
        fi
    done
done

```

Pictured below is an early version of our script being run and the results being output.

```
File Edit View Search Terminal Help

5764
Testing the arbitrary requirement
Testing the arbitrary component
arbitraryMethod
"&#x000a;"
"\n"
Test passed.

Run testcase executable based off of testCase2.txt specifications.

2
Testing the arbitrary requirement
Testing the arbitrary component
arbitraryMethod
"&#xa;"
"\n"
Test passed.

Run testcase executable based off of testCase3.txt specifications.

3
Testing the arbitrary requirement
Testing the arbitrary component
arbitraryMethod
"&#x00ziggy"
"&#x00ziggy"
Test passed.

Run testcase executable based off of testCase4.txt specifications.

4
Testing the arbitrary requirement
Testing the arbitrary component
arbitraryMethod
"&#x0a;"
"\n"
Test passed.

Run testcase executable based off of testCase5.txt specifications.

2
Testing the arbitrary requirement
Testing the arbitrary component
arbitraryMethod
"&#x00a;"
"\n"
Test passed.
```

Chapter Four: Test Cases

Task: Complete the design and implementation of your testing framework... You will create 25 test cases that your framework will automatically use to test your H/FOSS project.

Testing Other Methods

In Chapter Three, we used the `decodeHTML` method to determine if the Sanitizer was capable of properly decoding Unicode character codes, such as decoding `
` into an HTML carriage return, `\n`. While the task for this deliverable didn't require us to test a new method for our additional 20 test cases, we wanted to include additional test drivers for a more useful test framework.

After scanning through the Java classes in the project source code, we decided that the `sanitize` method should be tested. We began to write a test driver that would determine if input with HTML tags, would be removed completely. Due to an overwhelming amount of dependencies and an hour of trial and error, we gave up and found a new method. The `cssContent` method handles escape sequences and strips any quotes from the input. A snippet of the method code is below.

```
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 == '\'') {
            if (ch0 == token.charAt(n - 1)) {
                pos = 1;
                --n;
                sb = new StringBuilder(n);
            }
        }
    }
}
```

After creating a second driver for the `cssContent` method, we went back to the `sanitize` method in an effort to better understand the method. We learned that the `sanitize` method takes in two arguments: a `String` and a `Policy`. The `Policy` is a global object that will receive events based on the tokens in HTML. Typically, this policy ends up routing the events to an `HtmlStreamRenderer` after filtering.

```
public static void sanitize(@Nullable String html, final Policy policy) {
    if (html == null) { html = ""; }

    TagBalancingHtmlStreamEventReceiver balancer
        = new TagBalancingHtmlStreamEventReceiver(policy);
```

Displaying Our Results

We added a helper script in our /scripts folder (toCompiledHtmlWithCss.sh) to take the individual .txt report files and compile them into a readable table. Below is an early representation of our work.

17	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	FORMATTING.sanitize(String)	Hello, World!	Hello, World!	passed
18	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	FORMATTING.sanitize(String)	>!	>!	passed
19	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	FORMATTING.sanitize(String)	Hello, World!	Hello, World!	passed
20	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)	Hello, World!	Hello, World!	passed
21	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)	Hello, World!	Hello, World!	passed
22	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)	Hello, World!	Hello, World!	passed
23	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)	Hello, World!	Hello, World!	passed
24	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)			passed
25	org.owasp.html.Sanitizers	org.owasp.html.Sanitizers	BLOCKS.sanitize(String)	hello	hello	passed

New Drivers

We went back and updated our old drivers.

```
public static void main(String[] args){
    try{
        String theOracle = args[2];
        String theTest = args[1];
        String staticElement = args[0];

        theOracle = theOracle.replaceAll("\\\"", "\\\"\\\"");
        theOracle = theOracle.replaceAll("\\\\", "\\\"\\\\\"");
        theOracle = theOracle.replaceAll("<", "&lt;");
        theOracle = theOracle.replaceAll(">", "&gt;");

        theTest = theTest.replaceAll("\\\"", "\\\"\\\"");
        theTest = theTest.replaceAll("\\\\", "\\\"\\\\\"");
        theTest = theTest.replaceAll("<", "&lt;");
        theTest = theTest.replaceAll(">", "&gt;");
```

Chapter Five: Fault Injection

Task: Design and inject faults into the code you are testing that will cause at least 5 tests to fail, but hopefully not all tests to fail. Exercise your framework and analyze the results.

Faults

For our initial batch of test cases, we tested three methods: `decodeHTML(String)`, `cssContent(String)`, and `sanitize(String, Policy)`. We went into `cssContent` to vandalize some of the method code. All faults are commented on in the code with the original code being commented out below the fault.

We noticed a conditional towards the beginning of the method. The conditional takes the very first character of a `String` and checks if it is a quotation mark or a backslash. In CSS, these characters will result in a character escape. ([CSS specs.](#)) As such, this code will decode any escape sequence and strip any quote from the input. With this first conditional forced to always return false (char `ch0` can never be both a quote and a backslash), any of our test cases containing these characters would fail.

```
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);
                }
            }
        }
    }
}
```

In addition, while searching for additional code to mess around with, we discovered an `isHex(int)` method directly below `cssContent(String)`. The method would interpret a character as an `int` to determine whether or not it could be a valid hex value.

```
public static boolean isHex(int codepoint) { // FAULT: 0x09Aa should be false now
    return ('1' <= codepoint && codepoint <= '8')
        //return ('0' <= codepoint && codepoint <= '9')
        || ('B' <= codepoint && codepoint <= 'Z')
        //return ('A' <= codepoint && codepoint <= 'F')
        || ('b' <= codepoint && codepoint <= 'e');
    //return ('a' <= codepoint && codepoint <= 'f')
}
```

After changing the method rules of what values could return “true”, we quickly wrote a new test driver and four additional test cases to see whether or not the method would return “false” with a valid hex value and vice versa.

Results

The results of our cssContent fault injection is presented in the table snippet below. The columns in this example are: Input, Output, Oracle, and Result. Simply screwing up a conditional with two characters can cause several tests to fail unexpectedly.

'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

The following table taken from our HTML output shows the results of our isHex fault injection. It is a screenshot of how our table looked before we were able to get the requirements table working properly.

26	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	isHex(int)	0x00AA	true	passed
27	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	isHex(int)	0x09Af	true	failed
28	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	isHex(int)	0x18BB	true	passed
29	org.owasp.html.CssGrammar	org.owasp.html.CssGrammar	isHex(int)	0xZZZZ	false	passed

We eventually removed the isHex test cases and driver as the method was originally private. We researched testing private methods and it seems like the public interface methods are the ones we should be testing.

Chapter Six: Final Thoughts

Overall Experiences

Steven: I came into this project with a lot of experience. By the end of the project, I learned how to open default browsers via the command line. Everyone in the group has been very willing to learn, and has shown individual initiative from the start. We accessed each team member's abilities and defined our roles before we began the project, which has led to a fairly smooth ride as we've progressed. If there's a single thing that I believe the team would unanimously agree on, it's that our different views, experiences, and talents produced an unexpected synergy. Each team member seems to trust the others to do their job, which reduces stress all around and lets us perform optimally as a team.

Marta: After this project, I have become familiar with how scripts work and how to write them. This is my first experience with Git. What I like most about working as a team is the possibility to learn new techniques and skills. Personally, I consider this opportunity a real goal to be achieved, just after the successful conclusion of the project, and I believe that at least this purpose has already been reached. In the future, I think we should have focused on providing adequate documentation for our work. This proved to be the greatest weaknesses of the project as we struggled to make our final report coherent and well understandable.

Seth: As we have moved along this semester, we have all done our best to follow along with our team leader Steven and learn as much as possible. We have tried to pitch in where we can so that it is not just him doing all of the work while we sit back and relax. He has been an excellent team leader. I have come to realize that I probably did not learn as much about Linux in this class as I should have, however, it has made me want to learn more about it over the winter break.

Michael: At the beginning of this course, I had zero experience with Linux and Git. Up to this point, I had only worked on simple programs written in various languages. This was also my first time working in a group for the entire semester. I always complained that the software I created consisted only of simple sort algorithms and now, through creating a testing framework, I feel more confident in my skills as a developer. Steven and the rest of the group have helped me understand better how our automated testing framework operates. I am excited to learn more about Bash scripts and will be using GitHub for a personal repository in the future.