Computer Science 362

Final Project Part B

Scott Milton, Ashok Nayar, Andrew Shen

URLValidator.pdf (bug report)

Bug Reports

Bug #1: URLValidator returns false for any valid URL queries
Severity: **HIGH**. This bug means that any and all valid URL queries will be returned as false, thus preventing any data from queries to be used by the target application.
Priority: **HIGH**. This bug severely limits the data that can be passed to the target application and falsely returns the status of a valid query URL
Reported by: Ashok Nayar (nayara@oregonstate.edu)
Report Date: 12/7/2015
Steps to reproduce:
Define a UrlValidator variable with the following schemes:

```
String schemes[]  = {"https", "http"};
UrlValidator urlVal = new UrlValidator(schemes);
```

Pass any or all of the following valid URLs to the URLValidator:

```
url = https://www.amazon.com/test?action=true
url = https://www.amazon.com/test?data=somedata
url = https://www.amazon.com/test?data=somedata&username=nayara
url = https://www.amazon.com/test?
```

Call the isValid function and verify that the return value is false rather than true:

```
if (!urlVal.isValid(url))
{
        System.out.println("TEST Failed: "+url);
}
```

File and line where bug occurred: UrlValidator.java (line 446)
Additional details: This bug was initially discovered through manually testing of known valid URLs. The bug was confirmed through a more thorough testing of query URLs and using the Eclipse debugger. Through using the debugger, it was discovered that the bug exist in the isValidQuery() function of the URLValidator (additional discovery details are noted in this report)

Bug #2: URLs with any number of spaces (‘ ‘) immediately preceding ‘/’ are not being properly flagged as invalid URLs.

Reported by: Andrew Shen (shena@engr.oregonstate.edu)

Severity: **High**. This bug shows that URLValidator is not properly handling spaces. Most modern web browsers automatically handle whitespace and spaces are commonly input by users to initiate a search rather than navigating to a specific URL.

Priority: **High**. This bug poses no serious security threat but is a serious hindrance to the usability and proper functioning of a web browser.

Description:
This bug was originally found via manual testing using the testManualTest() function in UrlValidatorTest.java. In this bug, any URL with any number of spaces before a “/” are recognized as valid even though such URLs are not valid.

The cause of this bug stems from the isValidPath() function in UrlValidator.java, which fails to check for spaces before path names, indicated by a “/”. Thus, any number of spaces immediately preceding a “/” character will be ignored completely. This function starts on line 411 and ends at line 434. A logical analysis of the isValidPath() functions reveals that leading spaces are not even taken into consideration by the function because the function does not contain a check for leading spaces. Such a check would ideally be placed on line 416 after

```
if (path == null) {
    return false;
}
```

and before

```
if (!PATH_PATTERN.matcher(path).matches()) {
    return false;
}
```

To reproduce this bug, one can pass any of the following URLs into UrlValidator:

“http://www.google.com /drive”
“http://www.google.com /drive”
“http://www.google.com   /drive”
“https://www.linkedin.com/in   /user”

To pass the URL into UrlValidator:

```
  if(urlVal.isValid(url) == true){
    System.out.println("*Bug found: '" + url + "': isValid() returned true.");
  }
```

Manual Testing

Manual testing was performed in two steps. First, known valid URLs were passed to the URL validator and tested to see if “true” was returned from the isValid() function. Any URLs that did not return true were noted for future testing. The second step was to take a set of known invalid URLs and test them via the isValid() function. The return values were then checked for any return values of true and the URLs noted for future testing.

The test consisted of two String arrays, one containing valid URLs and the second containing invalid URLs, iterated over while passing each URL to the urlVal.isValid() function.

The test function prints a message indicating a bug has been found when the result of the isValid() function call does not match the expected return value, that is, where a confirmed Valid URL returns false or a confirmed Invalid URL returns true.

Input Partitions
Testing inputs were broken down into several categories, each relating to the type of URL that can exist. The following categories are:

URLs with only letters (http://www.google.com)
URLs with only numbers (http://www.123.com)
URLs with both letters and numbers (http://www.abc123.com)
URLs with invalid characters in them (http://www.%$!.com)
IP address (http://192.168.0.1)
URL Queries (http://www.google.com?action=true)
Port numbers (http://www.google.com:80)

These partitions limit the range of testing values, allowing us to focus upon exclusively valid and invalid URLs within each partition.

Program Based Testing
Using the partitions mentioned in the previous section, we were able to divide the testing into the following test functions:

testPorts()
testIPs()
testLetters()
testNumbers()

testLettersAndNumbers()
testParams()
testsOddities()

In the case of testParams, parameters and queries were tested using nested loops. The URLs were broken into different parts such as the schemes, domains, subdirectories, and various parameters and arguments. Loops were constructed in order to build different URLs and the loops were constructed in such a way that all URLs were valid URLs. As each URL was built, it was passed to the isValid() function and the return value was checked to make sure that it equalled true. If isValid() returns false at any point, then clearly there is something wrong with the function. In addition to the loops, outlier and invalid URLs (ie http://www.google.com?action=?) were also passed to the function to validate that the URLValidator was recognizing them as false. Through these tests, we were able to see a common pattern in the URLs that were being incorrectly validated. In this case, it appears that any valid query passed to the URLValidator is returned as false.

Debugging

Using the Eclipse debugger, we were able to step through the code in order to discover the cause of the bug. Through the debugger, we were able to determine that valid queries were being returned as false at line 446 of URLValidator.java:

```
return !QUERY_PATTERN.matcher(query).matches();
```

This line is performing a NOT operation on a valid query, return false instead of true.

As for Agan's rules, we followed nearly all of them mostly because many of the rules are inherent to testing in general, but also because it was the best way to approach the problem. Clearly we needed to understand the system first before we could do any meaningful testing and breaking the system simply involved running our standard suite of tests. Where Agan's rules really came into effect were when a potential bug was discovered. Because the URLValidator validates input using dozens of different rules, we need to change thing very careful and one at a time in order to correctly isolate the issue. This also meant that we needed to divide and conquer, which is represented in our individual partitions and functions. Additionally, because multiple inputs could possibly trigger the same bug, we need to keep a strong audit trail of inputs and their results.

Bug #3: URLValidator returns false for all port numbers 1000 through 65535

Severity: **Serious Bug**. This bug means that any and all valid URLs with port numbers from 1000 through 65635 will be returned as false, flagging them as invalid.

Priority: **Minor**. This bug poses no serious security threat, but represents a serious hindrance to the functionality of the program. Most URLs that contain a port number will likely use port 80, which is unaffected by this bug.

Date: 12/7/15
Reported by: Scott Milton (miltons@oregonstate.edu)

Is it reproducible: Yes, every time for port numbers in this range.

File and line where bug occurred: URLValidator.java (line 158)

Description: This bug was originally found via manual testing of known valid URLs. The initial round of manual tests included a URL with a port number of 65535, which was expected to return a value of true to indicate that it is valid. When a value of false was returned instead, we created additional manual tests to check the return values for other port numbers. This more thorough testing of URLs indicated that URLValidator is erroneously returning false values for valid ports 1000 through 65535. By using a test case named testPorts() along with the debugger in Eclipse, we were able to track the bug to line 158 of URLValidator.java, more specifically to the value of PORT_REGEX in that line, which is set to a regular expression that matches numbers with one to three digits. As a result, any port number with more than three digits (higher than 999) is returned as false, flagging it as invalid.

Steps to Reproduce:

1. Add any of the following lines to the testManualTest() function of URLValidatorTest.java:

```
System.out.println("Valid port, expected true, returned " + urlVal.isValid("http://www.facebook.com:1000"));

System.out.println("Valid port, expected true, returned " + urlVal.isValid("http://www.facebook.com:10000"));

System.out.println("Valid port, expected true, returned " + urlVal.isValid("http://www.facebook.com:65535"));
```

2. Run the program in Eclipse

3. Note the output in the console window

Attachments: URLValidatorTest.java

Other Information: As indicated in the section above on Debugging, we used many of Agans' principles to identify this bug. Significant efforts were made to understand the system (Rule #1) by reading materials on the Apache commons validator website and researching the bug tracking system employed by the Apache Software Foundation. By creating appropriate input partitions, we were able to follow Rule #4 (Divide and Conquer) and Rule #5 (Change One Thing at a Time) to isolate and track down the bug in question. In this way, we were able to deal with the port component of the URL by itself and we added higher-resolution input partitions with valid and invalid port inputs (i.e.-- negative port numbers, border cases like -1, 0, 65535, and 65536--as well as port numbers with one, two, three, four, and five digits. Tracking the bug to specific numeric ranges allowed us to to “Make it Fail” consistently (Rule #2). Moreover, we applied Rule #6, by “Keeping an Audit Trail” by writing down and noting what we had done and what we had tried as we were proceeding. The bug was left in place, as per the assignment instructions, to be fixed by the development team.