JPass Dataflow Coverage
Verificação e Validação de Software 2020–2021

Manuel Tomás 51054
Tiago Varela 51017

2021/04/19

Contents

Function Selection **1**

stripNonValidXMLCharacters(String in) **2**
Purpose . . . 2
Dataflow Analysis . . . 2
Flowchart . . . 3
Paths . . . 3
Designed Tests . . . 5
Unit tests . . . 6

encrypt(byte[] data, int length) **7**
Purpose . . . 7
Dataflow Analysis . . . 7
Flowchart . . . 7
Paths . . . 7
Designed Tests . . . 9
Unit tests . . . 10

getSha256Hash(char[], int) **11**
Purpose . . . 11
Dataflow Analysis . . . 11
Flowchart . . . 11
Paths . . . 11
Designed Tests . . . 13
Unit tests . . . 14

Function Selection

For selecting the functions we analysed various methods in the code and selected them based on whether they contained variable flow, that is, whether they contained ifs and

loops. We will select even methods we have covered before because the objective is to perform the analysis and contemplate paths that weren't covered through any of our other techniques. Then, we will obtain a number of paths to test through the analysis, and we'll develop new tests for the paths that aren't yet covered by any existing test. As such, the selected functions were as follows:

- jpass.util.StringUtils.stripNonValidXMLCharacters(in)
- jpass.util.CryptUtils.getSha256Hash(text, iteration)
- jpass.crypt.Cbc.encrypt(data, length)

stripNonValidXMLCharacters(String in)

Purpose

This method ensures that the output String has only valid XML unicode characters. The values being used for this implementation can be found in the function documentation. [1]

Dataflow Analysis

Flowchart

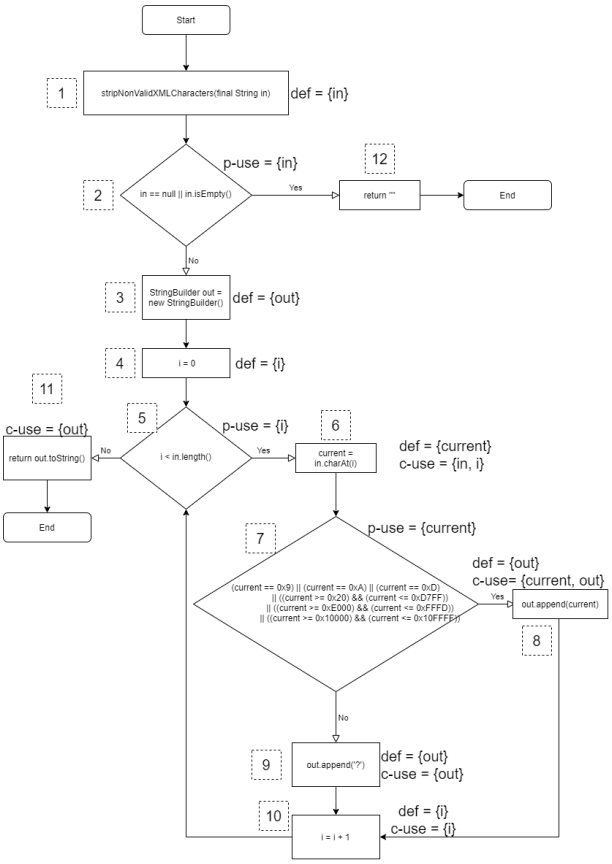

Figure 1: stripNonValidXMLCharacters() Flow Chart

Paths

In order to arrive at the necessary paths, we will be applying the various coverage criteria for Dataflow Testing. The paths should cover each def-case (all-defs), and from each def-case should hit each c-use in a def-clear path (all-c-uses), and from each def-case should hit each p-use in a def-clear path (all-p-uses), thereby contemplating all uses (all-uses).

One notable case is that we've opted to perform a loop when a variable is defined and used in the same line, inside the loop. So if line 4 were a def-case and a use-case we would create both the path <4,4> and the path that performs a full loop from 4 back to 4.

The path tables for each variable are as follows:

out			
Pair Id	Def	Use	Path
1	3	11	<3,4,5,11>
2	3	8	<3,4,5,6,7,8>
3	3	9	<3,4,5,6,7,9>
4	8	8	<8,8>
5	8	8	<8,10,5,6,7,8>
6	8	9	<8,10,5,6,7,9>
7	8	11	<8,10,5,11>
8	9	9	<9,9>
9	9	9	<9,10,5,6,7,9>
10	9	8	<9,10,5,6,7,8>
11	9	11	<9,10,5,11>

in			
Pair Id	Def	Use	Path
1	1	(2,T)	<1,2,12>
2	1	(2,F)	<1,2,3>
3	1	6	<1,2,3,4,5,6>

i			
Pair Id	Def	Use	Path
1	4	(5,T)	<4,5,6>
2	4	(5,F)	<4,5,11>
3	4	6	<4,5,6>
4	4	10	<4,5,6,7,8,10>
5	4	10	<4,5,6,7,9,10>
6	10	6	<10,5,6>
7	10	10	<10,10>
8	10	10	<10,5,6,7,8,10>

current			
Pair Id	Def	Use	Path
1	6	7	<6,7>
2	6	8	<6,7,8>

Designed Tests

Test 1			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,12>	in-1	shouldStripEmptyTest()	This function makes use of the if at the start.

Test 2			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,11>	in-2, out-1, i-2	Impossible flow	The previous if ensures the string has a length above 0, which means that the loop will loop once.

Test 3			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,8,10,5,11>	in-2, in-3, out-2, out-4, out-7, i-1, i-3, i-4, i-7, current-1, current-2	shouldStripChar0x9MinusOneTest()	The loop is exercised once, passing through 8, that is, with a valid character ('a', in this case).

Test 4			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,9,10,5,11>	in-2, in-3, out-3, out-8, out-11, i-1, i-3, i-5, i-7, current-1	shouldStripChar0x9MinusOneTest()	The loop is exercised once, passing through 9, that is, with an invalid character (0x8, in this case).

Test 5			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,8,10,5,6,7,8,10,5,11>	in-2, in-3, out-2, out-4, out-5, out-7, i-1, i-3, i-4, i-6, i-7, i-8, current-1, current-2	shouldStripStringTest()	The string contains two valid characters in a row, and therefore loops through 8 twice.

Test 6			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,8,10,5,6,7,9,10,5,11>	in-2, in-3, out-2, out-4, out-6, out-8, out-11, i-1, i-3, i-4, i-6, i-7, current-1, current-2	shouldStripChar0x9MinusOneTest()	The string contains a valid character followed by an invalid one, so it goes through the loop passing through 8 and afterwards again by passing through 9.

Test 7			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,9,10,5,6,7,8,10,5,11>	in-2, in-3, out-3, out-4, out-7, out-8, out-10, i-1, i-3, i-5, i-6, i-7, i-8, current-1, current-2	shouldStripChar0x9MinusOneTest()	The string contains an invalid character followed by avalid one, so it goes through the loop passing through 9 and afterwards again by passing through 8.

Test 8 - shouldStripTwoInvalidCharInARowTest()			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,9,10,5,6,7,9,10,5,11>	in-2, in-3, out-3, out-8, out-9, out-11, i-1, i-3, i-5, i-6, i-7, current-1	None	A new function will be developed with two invalid characters in a row, so as to loop through 9 twice.

Unit tests

We implemented only the tests which required paths not already covered by existing tests. In this case, that would be the following test:

shouldStripTwoInvalidCharInARowTest()	
What is being done?	Attempts to strip a string with two invalid chars in a row, which should exercise a def-use case.
What is being tested?	Tests if the returned value has each invalid char replaced with a '?'.
Result	Pass

encrypt(byte[] data, int length)

Purpose

This method encrypts the data in the "data" parameter up to index "length".

Dataflow Analysis

Flowchart

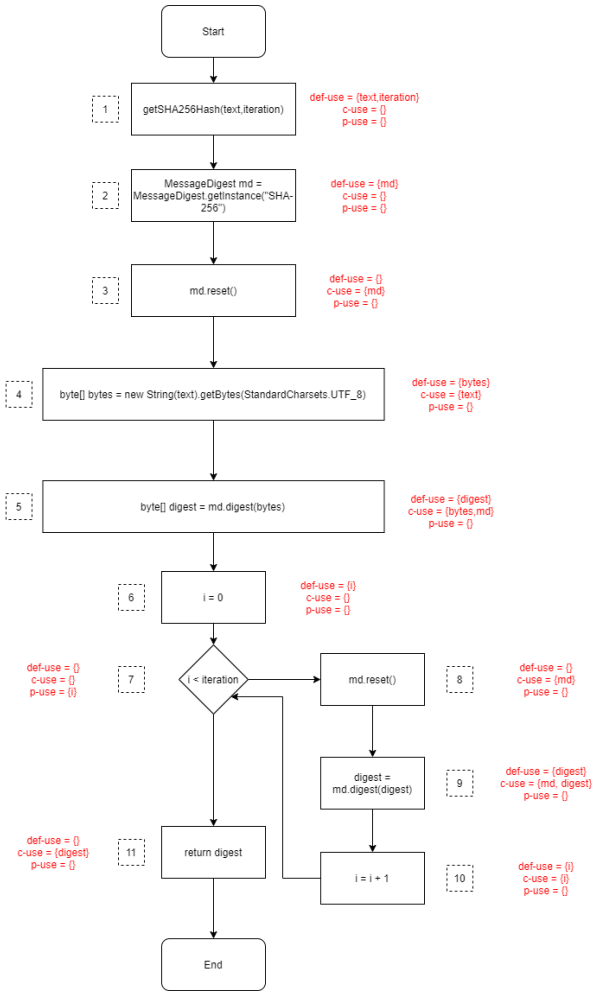

Figure 2: getSha256Hash() Flow Chart

Paths

In order to arrive at the necessary paths, we will be applying the various coverage criteria for Dataflow Testing. The paths should cover each def-case (all-defs), and from each def-case should hit each c-use in a def-clear path (all-c-uses), and from each def-case should hit each p-use in a def-clear path (all-p-uses), thereby contemplating all uses (all-uses).
One notable case is that we've opted to perform a loop when a variable is defined and used in the same line, inside the loop. So if line 4 were a def-case and a use-case we would create both the path <4,4> and the path that performs a full loop from 4 back to 4.

The path tables for each variable are as follows:

data			
Pair Id	Def	Use	Path
1	1	(2,T)	<1,2,3>
2	1	(2,F)	<1,2,4>
3	1	6	<1,2,4,5,6>

length			
Pair Id	Def	Use	Path
1	1	(2,T)	<1,2,3>
2	1	(2,F)	<1,2,4>
3	1	(5,T)	<1,2,4,5,6>
4	1	(5,F)	<1,2,4,5>

i			
Pair Id	Def	Use	Path
1	4	(5,T)	<4,5,6>
2	4	(5,F)	<4,5>
3	4	6	<4,5,6>
4	4	11	<4,5,6,7,8,9,10,11>
5	11	(5,T)	<11,5,6>
6	11	(5,F)	<11,5>
7	11	6	<11,5,6>
8	11	11	<11,5,6,7,8,9,10,11>

_overflow			
Pair Id	Def	Use	Path
1	6	8	<6,7,8>

_overflowUsed			
Pair Id	Def	Use	Path
1	0	6	<0,1,2,3,4,5,6>
2	6	7	<6,7>
3	10	6	<10,11,5,6>

_outBuffer			
Pair Id	Def	Use	Path
1	8	9	<8,9>

_output			
Pair Id	Def	Use	Path
1	9	[1]	<9>

Designed Tests

Test 1			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3>	data-1, length-1	shouldNotEncryptSizeNull()	This function makes use of the if at the start.

Test 2			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3>	data-1, length-1	shouldNotEncryptNegativeSize()	This function makes use of the if at the start.

Test 3			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,8,10,5,11>	data-2, data-3, length-2, length-3, i-1, i-3, i-4, i-5, i-6, i-7, i-8, _overflow-1, _overflowUsed-1, _overflowUsed-2, _overflowUsed-3, _outBuffer-1, _output-1	None	The loop is exercised 17 times, making the "if" statement inside the loop true at the 16th iteration.

Test 4			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,4,5>	data-2, length-2, length-4, i-2	Impossible flow	The previous if ensures that "length" is bigger than 0. As "i" get value 0, it is certain that the condition will be true in the first loop iteration.

Unit tests

We implemented only the tests which required paths not already covered by existing tests. In this case, that would be the following test:

shouldEncryptBiggerThan16()	
What is being done?	Calls the encrypt method with a byte[] bigger than 16 (which is the value of "BLOCK_SIZE"), which makes the if statement inside the loop test true at the 16th, exercising the flows that pass through it.
What is being tested?	Tests if the returned value is the same of the one given when decrypted.
Result	Pass

getSha256Hash(char[], int)

Purpose

This method calculates a SHA256 hash using the given text. A number of iterations for the process can be defined.

Dataflow Analysis

Flowchart

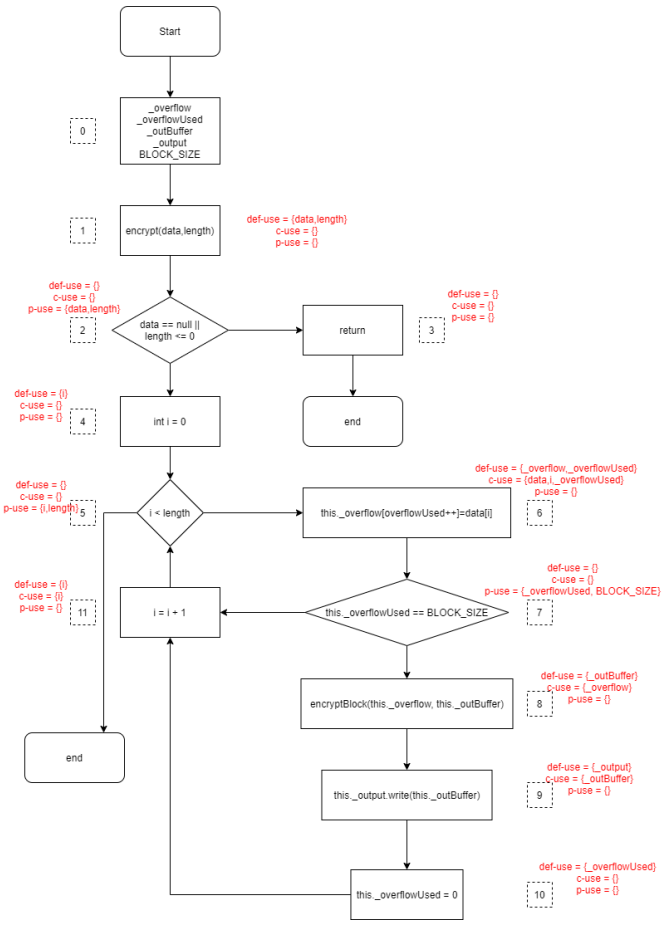

Figure 3: encrypt() Flow Chart

Paths

In order to arrive at the necessary paths, we will be applying the various coverage criteria for Dataflow Testing. The paths should cover each def-case (all-defs), and from each def-case should hit each c-use in a def-clear path (all-c-uses), and from each def-case should hit each p-use in a def-clear path (all-p-uses), thereby contemplating all uses (all-uses).
One notable case is that we've opted to perform a loop when a variable is defined and used in the same line, inside the loop. So if line 4 were a def-case and a use-case we would create both the path <4,4> and the path that performs a full loop from 4 back to 4.
The path tables for each variable are as follows:

text			
Pair Id	Def	Use	Path
1	1	4	<1,2,3,4>

iteration			
Pair Id	Def	Use	Path
1	1	(7,T)	<1,2,3,4,5,6,7,8>
2	1	(7,F)	<1,2,3,4,5,6,7,11>

md			
Pair Id	Def	Use	Path
1	2	3	<2,3>
2	2	5	<2,3,4,5>
3	2	8	<2,3,4,5,6,7,8>
4	2	9	<2,3,4,5,6,7,8,9>
5	2	11	<2,3,4,5,6,7,11>
6	9	9	<9,9>
7	9	9	<9,10,7,8,9>
8	9	11	<9,10,7,11>

bytes			
Pair Id	Def	Use	Path
1	4	5	<4,5>

i			
Pair Id	Def	Use	Path
1	6	(7,T)	<6,7,8>
2	6	(7,F)	<6,7,11>
3	6	10	<6,7,8,9,10>
4	10	(7,T)	<10,7,8>
5	10	(7,F)	<10,7,11>
6	10	10	<10,10>
7	10	10	<10,7,8,9,10>

Designed Tests

Test 1			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,11>	text-1, iteration-2, md-1, md-2, md-5, bytes-1, i-2	shouldGetSha256HashEmptyTest()	Using the SHA-256 hash, the "iteration" variable is passed as 0, so the loop is not exercised.

Test 2			
Path Covered	Pairs Covered	Equivalent to	Description
<1,2,3,4,5,6,7,8,9,10,7,11>	text-1, iteration-1, md-1, md-2, md-3, md-4, md-6, md-7, md-8, bytes-1, i-1, i-3, i-4, i-5, i-6, i-7	shouldGetPKCS5Sha256HashNotEmptyTest()	Using the PKCS5SHA-256 hash, the "iteration" variable is passed as 1000, so the loop is exercised.

Unit tests

We implemented only the tests which required paths not already covered by existing tests. In this case, all paths were already covered by the existing tests.

References

[1] W3C. Extensible markup language (xml) 1.0. [Online]. Available: http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char