Testing Framework for Tor

Final Report

theMonkeyCatfishCoalition

December 1st, 2015

TABLE OF CONTENTS

Introduction

For this project, we were tasked with building a working testing framework for an open-sourced project of our choosing. After reviewing a list of approved projects we narrowed down our selection to three projects and ultimately selected Tor. As will be explained in this report, there were initial difficulties that came with having Tor as our selection, but we had so many reasons to stick with the project that even though we initially thought of switching to one of the other programs, we ultimately decided that we would stick it out with testing Tor and made few compromises and decisions that made the project completable in the amount of time we had.

This document is mostly made up of reports on the milestones throughout the project starting with the selection and initial tests we did with Tor. The main idea is to show the progression of the project from start to finish and to show the different alterations we had to make to plans, changes made to test cases, changes to the framework's structure, and our reflections at the milestones. This document also includes all the information one would need to know to download and run our framework, add on to the framework, and understand the structure of the framework.

Chapter 1: Project Selection and Initial Tests

At this point in the project, we selected the open source project that we'd want to build a testing framework for. We selected Tor from a list of three considerations for the project, the three considerations were the following:

- Tor Browser– An Anonymous Browser
- Martus – A Bulletin for Human Rights Violations and Abuse
- VuFind- A Library Resource Portal

Ultimately we chose Tor for reasons that will be later disclosed in this section.

What Is Tor?

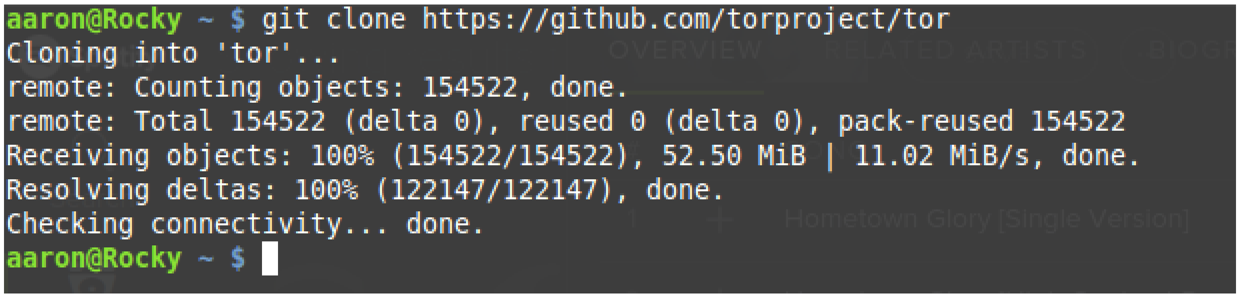
Tor is basically an anonymous network. This means that it obfuscates its users' information for a more secure online experience. Its primary purpose is to provide a way for individuals and organizations to privately share information over public networks. According to the project's website, it's also considered to be a tool for circumventing censorship, for correspondence between journalists and whistleblowers, socially sensitive communication like chat rooms for abuse victims, amongst other things. The browser we used in this project is configured to work with Tor.

Why We Choose Tor

A large part of why we chose Tor had to do with the goals of the project itself. Our group found Tor's purpose to be a noble one and it heavily impacted our decision to work with it. Another reason we picked Tor over the other considerations was that it is a very active and well documented project and thus we felt any issues we came across would be really simple to find solutions for either from the documentation or community behind the project.

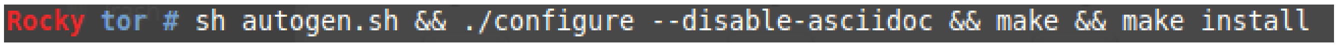
Getting and Compiling Tor

In order to even get close to starting this project, we had to clone Tor's repository from Github ,located here: https://github.com/torproject/tor.



```
aaron@Rocky ~ $ git clone https://github.com/torproject/tor
Cloning into 'tor'...
remote: Counting objects: 154522, done.
remote: Total 154522 (delta 0), reused 0 (delta 0), pack-reused 154522
Receiving objects: 100% (154522/154522), 52.50 MiB | 11.02 MiB/s, done.
Resolving deltas: 100% (122147/122147), done.
Checking connectivity... done.
aaron@Rocky ~ $
```

To compile Tor, we had to be running as root in the tor directory. We ran the following command to compile Tor (per the README in the repository):

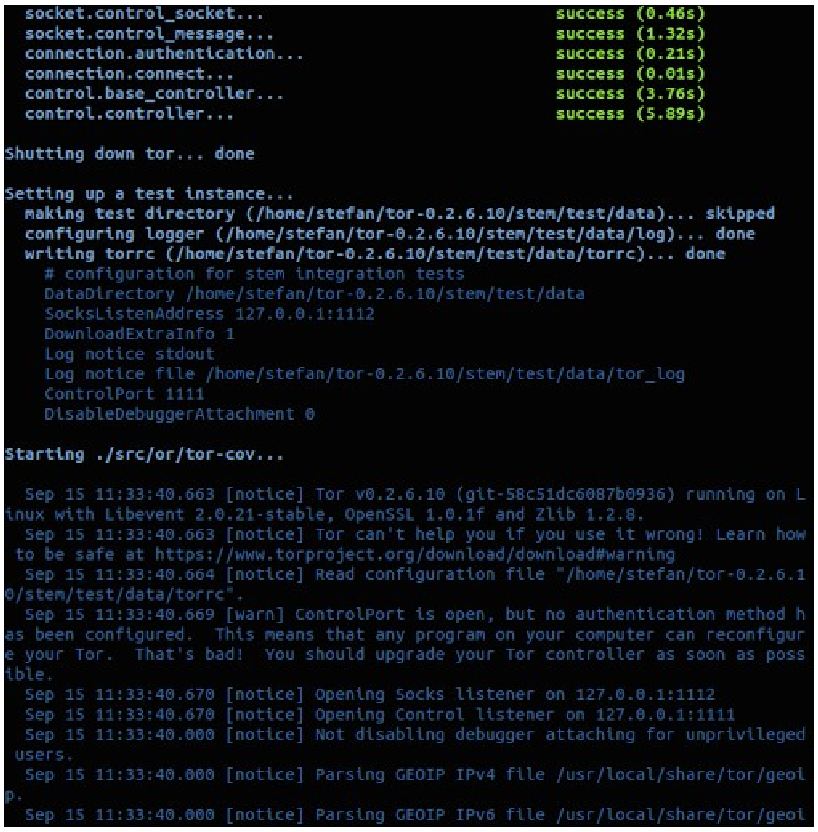


```
Rocky tor # sh autogen.sh && ./configure --disable-asciidoc && make && make install
```

After this, Tor was successfully compiled.

Initial Tests

We found some small unit tests packaged in the Tor repository and ran a few of them. These were the results:



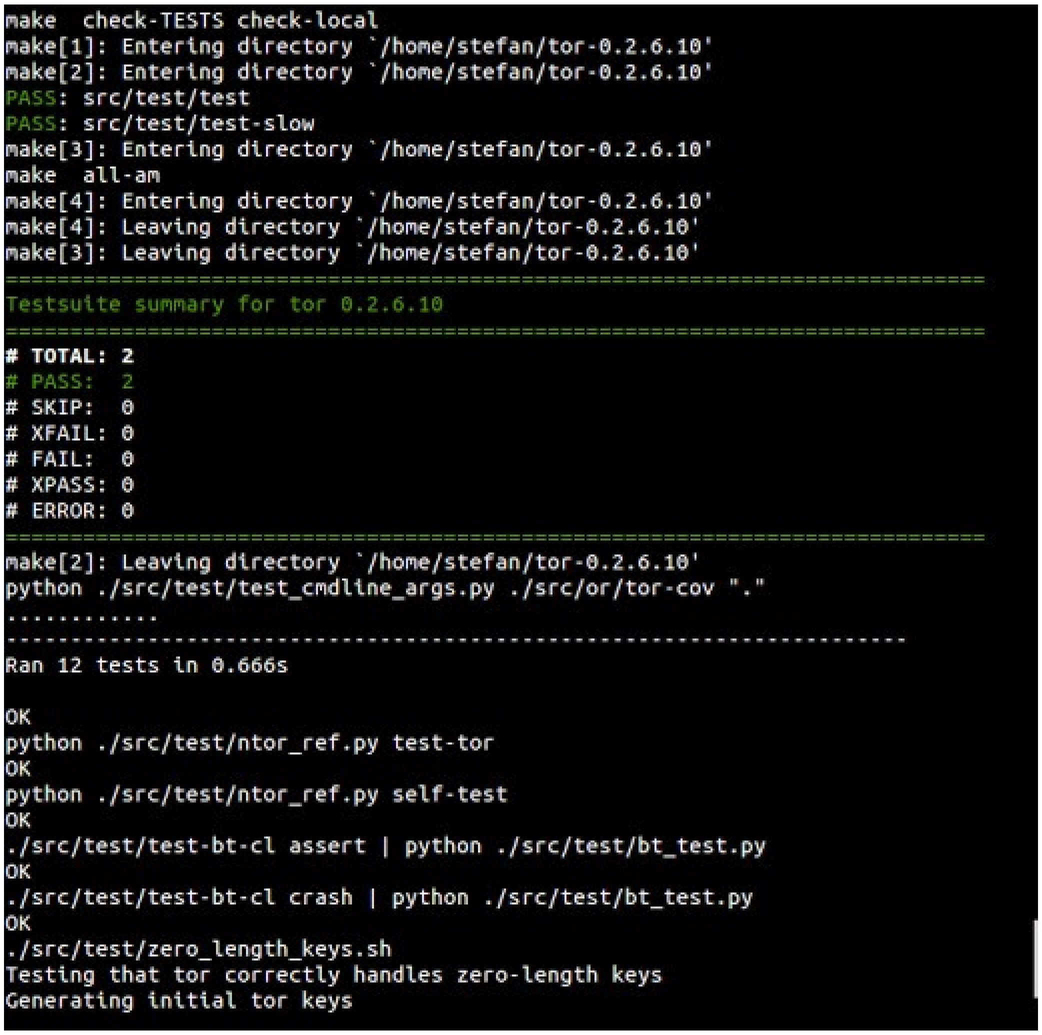
```
  socket.control_socket...                                   success (0.46s)
  socket.control_message...                                  success (1.32s)
  connection.authentication...                               success (0.21s)
  connection.connect...                                      success (0.01s)
  control.base_controller...                                 success (3.76s)
  control.controller...                                      success (5.89s)

Shutting down tor... done

Setting up a test instance...
  making test directory (/home/stefan/tor-0.2.6.10/stem/test/data)... skipped
  configuring logger (/home/stefan/tor-0.2.6.10/stem/test/data/log)... done
  writing torrc (/home/stefan/tor-0.2.6.10/stem/test/data/torrc)... done
    # configuration for stem integration tests
    DataDirectory /home/stefan/tor-0.2.6.10/stem/test/data
    SocksListenAddress 127.0.0.1:1112
    DownloadExtraInfo 1
    Log notice stdout
    Log notice file /home/stefan/tor-0.2.6.10/stem/test/data/tor_log
    ControlPort 1111
    DisableDebuggerAttachment 0

Starting ./src/or/tor-cov...

  Sep 15 11:33:40.663 [notice] Tor v0.2.6.10 (git-58c51dc6087b0936) running on L
inux with Libevent 2.0.21-stable, OpenSSL 1.0.1f and Zlib 1.2.8.
  Sep 15 11:33:40.663 [notice] Tor can't help you if you use it wrong! Learn how
 to be safe at https://www.torproject.org/download/download#warning
  Sep 15 11:33:40.664 [notice] Read configuration file "/home/stefan/tor-0.2.6.1
0/stem/test/data/torrc".
  Sep 15 11:33:40.669 [warn] ControlPort is open, but no authentication method h
as been configured.  This means that any program on your computer can reconfigur
e your Tor.  That's bad!  You should upgrade your Tor controller as soon as poss
ible.
  Sep 15 11:33:40.670 [notice] Opening Socks listener on 127.0.0.1:1112
  Sep 15 11:33:40.670 [notice] Opening Control listener on 127.0.0.1:1111
  Sep 15 11:33:40.000 [notice] Not disabling debugger attaching for unprivileged
 users.
  Sep 15 11:33:40.000 [notice] Parsing GEOIP IPv4 file /usr/local/share/tor/geoi
p.
  Sep 15 11:33:40.000 [notice] Parsing GEOIP IPv6 file /usr/local/share/tor/geoi
```



```
make  check-TESTS check-local
make[1]: Entering directory `/home/stefan/tor-0.2.6.10'
make[2]: Entering directory `/home/stefan/tor-0.2.6.10'
PASS: src/test/test
PASS: src/test/test-slow
make[3]: Entering directory `/home/stefan/tor-0.2.6.10'
make  all-am
make[4]: Entering directory `/home/stefan/tor-0.2.6.10'
make[4]: Leaving directory `/home/stefan/tor-0.2.6.10'
make[3]: Leaving directory `/home/stefan/tor-0.2.6.10'
============================================================================
Testsuite summary for tor 0.2.6.10
============================================================================
# TOTAL: 2
# PASS:  2
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[2]: Leaving directory `/home/stefan/tor-0.2.6.10'
python ./src/test/test_cmdline_args.py ./src/or/tor-cov "."
............
----------------------------------------------------------------------
Ran 12 tests in 0.666s

OK
python ./src/test/ntor_ref.py test-tor
OK
python ./src/test/ntor_ref.py self-test
OK
./src/test/test-bt-cl assert | python ./src/test/bt_test.py
OK
./src/test/test-bt-cl crash | python ./src/test/bt_test.py
OK
./src/test/zero_length_keys.sh
Testing that tor correctly handles zero-length keys
Generating initial tor keys
```

From these tests, we were able to see that the units tested worked properly within Tor's own framework and that we should expect relatively stable code from the project. It also allowed us, to an extent, to make sure that we acquired everything needed and that nothing was missing (at least within the dependencies of the units tested).

Chapter 2: Test Plan

At this point in the project, we built our testing plan and outlined it. Before we reached this milestone, we considered switching to a different project due to reasons outlined in the final section of this chapter.

The Testing Process

Our team plans to perform a series of tests on several methods found in the utils.c sub-system of the Tor project. We specifically plan on testing the following methods (as of now, we may add more later):
• round_to_the_next_multiple_of
• tor_strndup_
To do this, we will need to develop a small driver for the util.c methods so that our framework can function and also build a testing framework of at least twenty-five tests based off the test cases listed in this document. Once the framework is completed, we will run the tests and gather data on whether the methods passed or failed each.
As a final part of our testing process, we will inject faults into some of our tests to force a failure and analyze the results of the failures.

Tested Items

• util.c
◦ method: round_to_the_next_multiple_of
◦ tor_strndup

Test Cases

Case #	Method Tested	Inputs	Oracle
1	round_to_the_next_multiple_of	13, 4	16
2	round_to_the_next_multiple_of	0, 42	0
3	tor_strndup	abcdefg, 5	abcde
4	tor_strndup	qwerty, 0	(blank output)
5	tor_strndup	(empty), 5	(blank output)

Testing Schedule

Task	Completed By
Methods Driver Built	October 18th
Framework Built for First Five Test Cases	October 22nd
Full Framework Completed for All Twenty-Five Cases	November 12th
Report on Faults Injected into Test Cases	November 24th
Final Report on Tests	December 1st

Recording Procedures
We will keep a spreadsheet (which will be found on our wiki) to keep track of the results of each test. The spreadsheet rows will include the test case number, the method tested, the inputs tested, the expected output, the received output, and whether the test passed or failed. When tests fail, we will document the reason we believe to have failed (particularly in cases such as when we inject faults).

Constraints
Due to the amount of time we have to complete the testing framework, we do not expect to go over twenty-five test cases overall.

Further Comments
As of this chapter, the team all feels as though we have taken on quite the undertaking in picking Tor as the project we're testing. Tor's primary difficulty comes from the sheer size of the project, but also the heavy amount of code related to encryption in its repository. We chose to test utils.c files because of their simplicity in comparison to the rest of the project and hope that we are correct that they won't be as difficult to test as the more complex methods for Tor. We are mostly optimistic that we'll be able to complete all 25 test cases if we stick to the helper methods.

Chapter 3: Initial Testing Framework

At this point in the project we had a fledgling, but functional testing framework and 5 test cases made for it. All of the information in this chapter did not change in the final product.

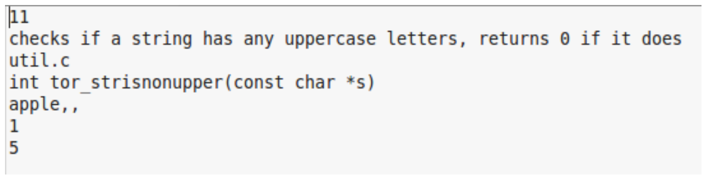
The Initial Framework

In order to have a working framework, we first had to build a driver that could call the methods we were testing. Currently we only are working with the utils.c methods and our drivers simply contain copied methods (we will change this later to call FROM utils.c) and calls. We have a driver for each method being tested,

Our framework currently reads in information from the testcase#.txt files to determine to test case identity. Based off of that it runs the designated driver for the test case and then compares the output with the expected result.

Structure of the Test Case Text Files

Line 1 - The number of the test case (1st, 2nd, etc.)
Line 2 - The purpose of the tested method
Line 3 - The location of the method to be tested
Line 4 - The name of the method to be tested
Line 5 - The input parameters
Line 6 - The expected output
Line 7 - A number to indicate the driver being used



```
11
checks if a string has any uppercase letters, returns 0 if it does
util.c
int tor_strisnonupper(const char *s)
apple,,
1
5
```

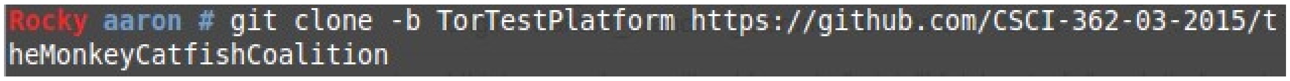
Example of Test Case

Directory Architecture

```
/TestAutomation
        /project
                /src
                 / …
        /scripts
                 runAllTests.py
        /testCases
                testCase1.txt
                testCase2.txt
                …
        /testCasesExecutables
                testCase1
                ...
        /temp
                 testCase1results
                 ...
        /reports
                report.html
```

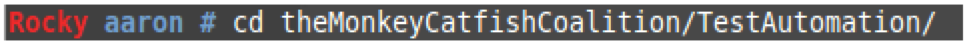
How To Use the Test Framework

1. Clone the 'TorTestPlatform' branch of the framework repository:



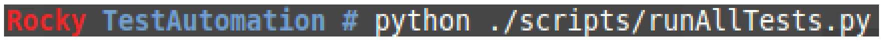
```
Rocky aaron # git clone -b TorTestPlatform https://github.com/CSCI-362-03-2015/t
heMonkeyCatfishCoalition
```

2. Change to the 'theMonkeyCatfishCoalition/TestAutomation' directory:



```
Rocky aaron # cd theMonkeyCatfishCoalition/TestAutomation/
```

3. Run the following command:



```
Rocky TestAutomation # python ./scripts/runAllTests.py
```

4. The script should automatically open the report in your browser. If there's an issue with your browser or if you need to access the report later, go into the 'reports' directory and open 'report.html'

Chapter 4: Completed Framework

At this point in the project we had a completed functioning framework and 25 test cases.

Framework Architecture

Our framework is a little different from the project specifications given that Tor is written in C. Our lead architect, Callum, constructed a separate main and look up table for Tor to be used as our primary driver. Each case is used as a specific case within the tor_main.c driver with arguments coming in normally from our test case .txt files and outputs going to a temp .txt file. Each result from the temp file is also compared to our specified oracles for each case. Aside from this obstacle, our testing framework still follows the proper specifications with a *runAllTests* script written in python to initiate action.

Test Cases

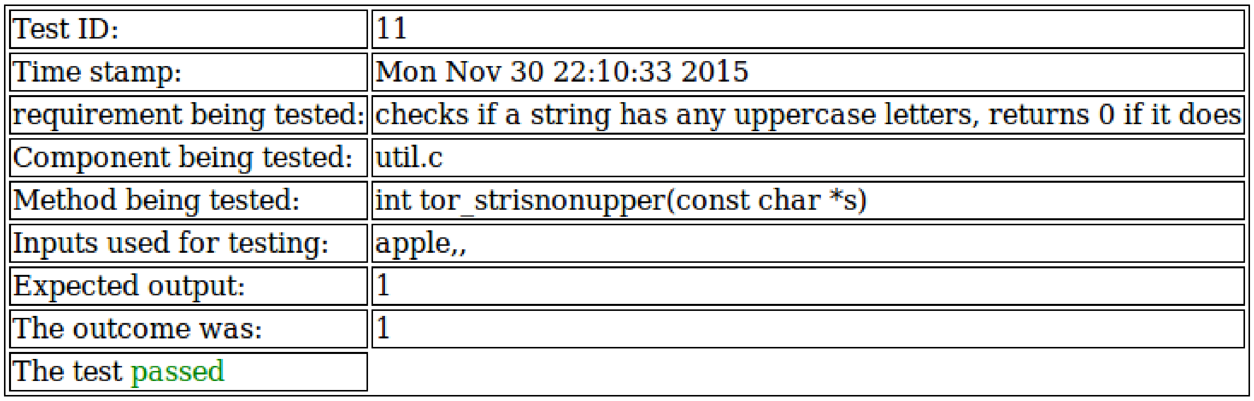
The following table includes all 25 test cases tested within our framework.

Test #	Component	Method	Input	Oracle
1	util.c	round_to_next_multiple_of	13,4	16
2	util.c	round_to_next_multiple_of	0,42	0
3	util.c	round_to_power_of_2	8223372036854776000	-9223372036854775808
4	util.c	round_to_power_of_2	4244967300	4294967296
5	util.c	round_to_power_of_2	-255	-256
6	util.c	round_to_power_of_2	0	1
7	util.c	digit_to_num	-1	error
8	util.c	digit_to_num	11	error
9	util.c	digit_to_num	1	1
10	util.c	digit_to_num	a	error
11	util.c	tor_strisnonupper	apple	1
12	util.c	tor_strisnonupper	Apple	0
13	util.c	tor_llround	5.6	6
14	util.c	tor_llround	5.3	5
15	util.c	tor_llround	-0.1	0
16	util.c	tor_llround	18446744073709551616	9223372036854775807
17	util.c	tor_llround	18446744073709551616	18446744073709551616
18	util.c	digit_to_num	3	3
19	util.c	digit_to_num	e	error
20	util.c	digit_to_num	-1	error

21	util.c	digit_to_num	(null)	error
22	util.c	digit_to_num	90	error
23	address.c	addr_mask_get_bits	0	0
24	address.c	addr_mask_get_bits	0x04A099B23	-1
25	address,c	addr_mask_get_bits	0xFFFFFFFFu	32

Test Report

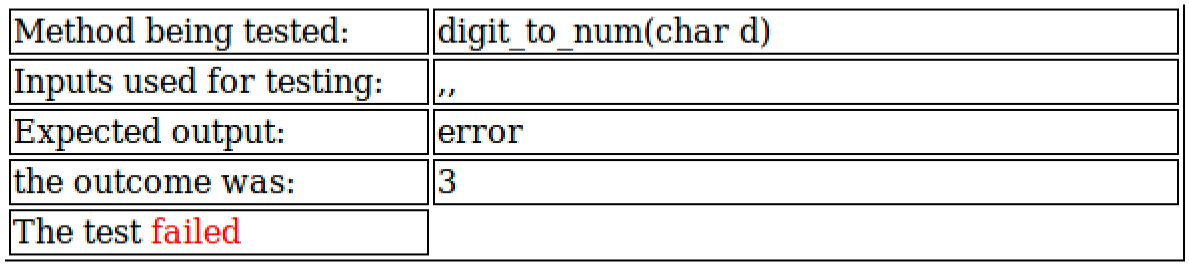
We setup the testing framework to generate a report in HTML and to open it on completion of all tests, here's and example of an item on the test report:



Test ID:	11
Time stamp:	Mon Nov 30 22:10:33 2015
requirement being tested:	checks if a string has any uppercase letters, returns 0 if it does
Component being tested:	util.c
Method being tested:	int tor_strisnonupper(const char *s)
Inputs used for testing:	apple,,
Expected output:	1
The outcome was:	1
The test passed	

As it stands, the test report does not sort the results in order of the test case number, and we're not quite sure why.

Reflection on Failed Tests



Method being tested:	digit_to_num(char d)
Inputs used for testing:	,,
Expected output:	error
the outcome was:	3
The test failed	

There were a few tests that failed, and pretty much all of them were tests of digit_to_num, which is supposed to convert a char into a int if the char is in the range of 0-9. However, there were cases such as the case below where the outcome was not an error as we expected, but a numeral. More perplexing are cases such as one of our tests result in a number despite having a null input as can be seen in the above figure. We are absolutely uncertain as to why this occurs, but did notice that nearly all the digit_to_num tests failed and it could either imply an actual issue with the code or a misunderstanding on our part of what the method is supposed to do and thus incorrect oracles in our test cases. However, results such as the one on the following page would imply that the documentation for the method is misleading at best if it is a correct output.

Chapter 5: Fault Injections

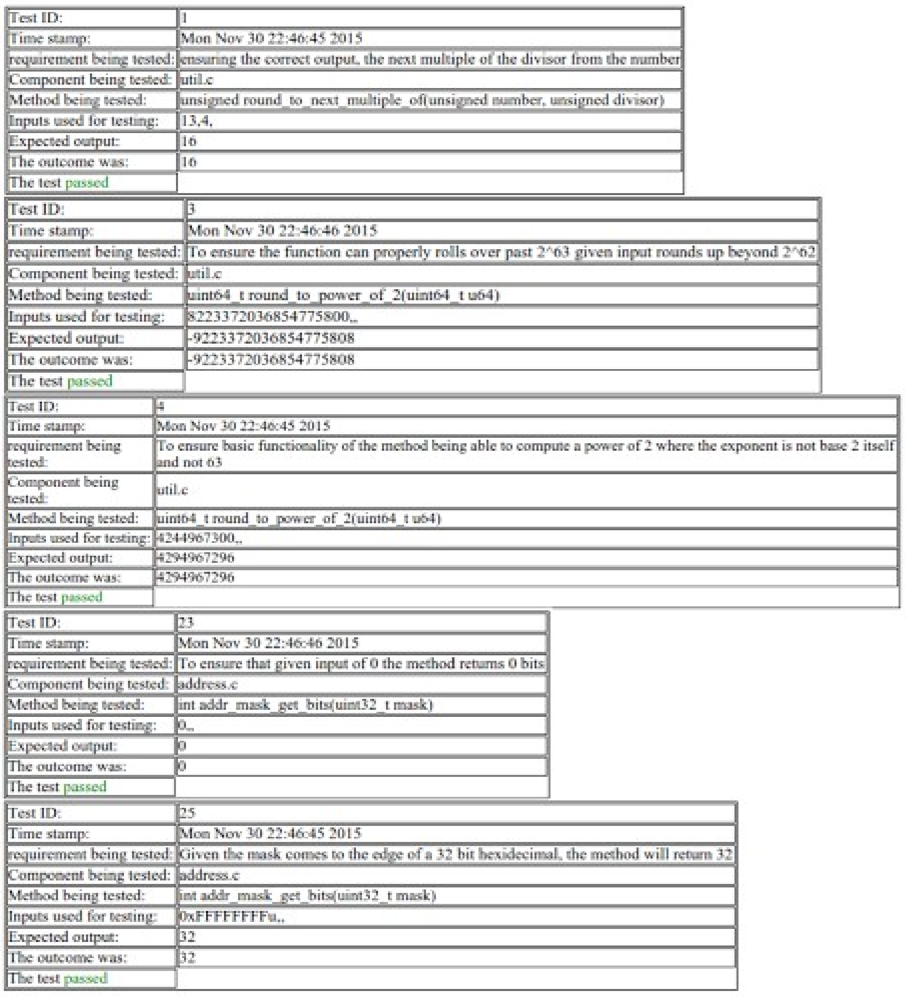
Faults Injected

1. in util.c: round_to_next_multiple_of:
   'number += divisor – 1' changed to 'number = divisor - 1'
2. in util.c: round_to_power_of_2:
   values of variable 'low' and 'lg2' switched
3. in address.c: addr_mask_get_bits:
   'if(mask=0)' changed to 'if (mask=1)'
4. in address.c: addr_mask_get_bits:
   'if (mask=0xFFFFFFFFu)' changed to 'if (mask == 0x0FFFFFFFu)'
5. in address.c: addr_mask_get_bits:
   'for (i=1; i<=32; ++i)' changed to 'for (i = 1; i<=32; i+=2)'

Results of Faulted Tests

Tests 1, 3, 4, 23, and 25 all failed as a result of our fault injections. This was expected since all the methods changed were tested by those tests. We had no other changes in the results.

<u>No Faults</u>



Test ID:	1
Time stamp:	Mon Nov 30 22:46:45 2015
requirement being tested:	ensuring the correct output, the next multiple of the divisor from the number
Component being tested:	util.c
Method being tested:	unsigned round_to_next_multiple_of(unsigned number, unsigned divisor)
Inputs used for testing:	13,4,
Expected output:	16
The outcome was:	16
The test passed	

Test ID:	3
Time stamp:	Mon Nov 30 22:46:46 2015
requirement being tested:	To ensure the function can properly rolls over past 2^63 given input rounds up beyond 2^62
Component being tested:	util.c
Method being tested:	uint64_t round_to_power_of_2(uint64_t u64)
Inputs used for testing:	8223372036854775800,,
Expected output:	-9223372036854775808
The outcome was:	-9223372036854775808
The test passed	

Test ID:	4
Time stamp:	Mon Nov 30 22:46:45 2015
requirement being tested:	To ensure basic functionality of the method being able to compute a power of 2 where the exponent is not base 2 itself and not 63
Component being tested:	util.c
Method being tested:	uint64_t round_to_power_of_2(uint64_t u64)
Inputs used for testing:	4244967300,,
Expected output:	4294967296
The outcome was:	4294967296
The test passed	

Test ID:	23
Time stamp:	Mon Nov 30 22:46:46 2015
requirement being tested:	To ensure that given input of 0 the method returns 0 bits
Component being tested:	address.c
Method being tested:	int addr_mask_get_bits(uint32_t mask)
Inputs used for testing:	0,,
Expected output:	0
The outcome was:	0
The test passed	

Test ID:	25
Time stamp:	Mon Nov 30 22:46:45 2015
requirement being tested:	Given the mask comes to the edge of a 32 bit hexidecimal, the method will return 32
Component being tested:	address.c
Method being tested:	int addr_mask_get_bits(uint32_t mask)
Inputs used for testing:	0xFFFFFFFFu,,
Expected output:	32
The outcome was:	32
The test passed	

With Faults

Test ID:	1
Time stamp:	Mon Nov 30 22:52:29 2015
requirement being tested:	ensuring the correct output, the next multiple of the divisor from the number
Component being tested:	util.c
Method being tested:	unsigned round_to_next_multiple_of(unsigned number, unsigned divisor)
Inputs used for testing:	13,4,
Expected output:	16
the outcome was:	0
The test failed	

Test ID:	3
Time stamp:	Mon Nov 30 22:52:29 2015
requirement being tested:	To ensure the function can properly rolls over past 2^63 given input rounds up beyond 2^62
Component being tested:	util.c
Method being tested:	uint64_t round_to_power_of_2(uint64_t u64)
Inputs used for testing:	8223372036854775800,,
Expected output:	-9223372036854775808
the outcome was:	62
The test failed	

Test ID:	4
Time stamp:	Mon Nov 30 23:02:19 2015
requirement being tested:	To ensure basic functionality of the method being able to compute a power of 2 where the exponent is not base 2 itself and not 63
Component being tested:	util.c
Method being tested:	uint64_t round_to_power_of_2(uint64_t u64)
Inputs used for testing:	4244967300,,
Expected output:	4294967296
the outcome was:	31
The test failed	

Test ID:	23
Time stamp:	Mon Nov 30 22:52:30 2015
requirement being tested:	To ensure that given input of 0 the method returns 0 bits
Component being tested:	address.c
Method being tested:	int addr_mask_get_bits(uint32_t mask)
Inputs used for testing:	0,,
Expected output:	0
the outcome was:	-1
The test failed	

Test ID:	25
Time stamp:	Mon Nov 30 23:53:06 2015
requirement being tested:	Given the mask comes to the edge of a 32 bit hexidecimal, the method will return 32
Component being tested:	address.c
Method being tested:	int addr_mask_get_bits(uint32_t mask)
Inputs used for testing:	0xFFFFFFFFu,,
Expected output:	32
the outcome was:	-1
The test failed	

Chapter 6: Final Thoughts and Conclusion

What We Learned

One of the biggest takeaways from this project was how difficult it is to work with open sourced projects like Tor. Even though Tor was very well documented and suppported, we still had difficulty deciphering simple helper methods such as those from utils.c and address.c (the best example of course being the digit_to_num method).

There were also some general lessons such as the difficulty of looking over and working with C code for those of us in the group that were less familiar with it and how sticking with a difficult project can sometimes be the better option if you're willing to make compromises. For us those compromises were mostly testing helper methods rather than the more complicated encryption methods that we were more interested in, but didn't fully understand. However, we saved ourselves a lot of time by not switching projects and, although we fell behind a bit due to our deliberations, we ultimately did enjoy staying with Tor.

It's also possible we actually found an issue in the code, if we didn't misunderstand the method and from the fault injections we were able to see first hand how small changes to the code can affect its functionality.

Evaluation of Team

Our team worked pretty well together and kept a pretty good line of communication through quick team meetings and use of Hangouts whenever we worked on things separately. There were almost no instances of miscommunication in terms of making test cases or the framework. We probably could have done better at completing the deliverables in a more timely fashion and including more detail, but overall we believe we did a good job and our proud of our work.

Evaluation of Assignments

There were times when the requirements for our deliverables were really unclear. For instance, as a lot of what went into Deliverable 3 was applicable to Deliverable 4, we weren't sure what needed to be included. It would have been helpful to have a rubric for certain assignments. It would have also been helpful to have a layout of what our test report was supposed to look like, as that was also unclear.
We did enjoy working with open source software and it definitely increased the interest of some of our group members who had never done anything with it before.

Conclusion

There is still plenty that can be done with Tor and the test cases we made don't even scrape the barrel of utils.c and address.c, much less the much larger project overall. Our group found working with Tor to be challenging, but interesting and enjoyable regardless.