Creating a testing framework for functions within Tor and our outcomes

The Monkeycatfishcoalition

Aaron Allsbrook / Spenser Black / Callum Brill / Stefan Moser
College of Charleston / The Monkeycatfishcoalition

What is Tor? Why did we choose it?

What is Tor?

Tor (The Onion Router) is a work of free, open source software that allows a user to maintain privacy and anonymity on the Internet. Tor works by connecting a user through a series of virtual tunnel connections with traffic directed through a volunteer network of more than six thousand relays. In order for an online activity to be traced, the tracer must work through each encrypted layer within the network. This allows internet activities, such as instant messages, visiting websites, and posting to websites, not easily traceable back to a single source. Tor's primary intention is to enable freedom of interacting with confidential materials, engaging in confidential communication, and protecting the privacy its users.

Why did we choose Tor?

We chose Tor because of its general widespread application towards users compared to most other H/FOSS project options. The intention of private, protected online use and encouragement of browsing with limited censorship was also very appealing. Tor also has a wide variety of users ranging from casual civilians to military and national security officials.

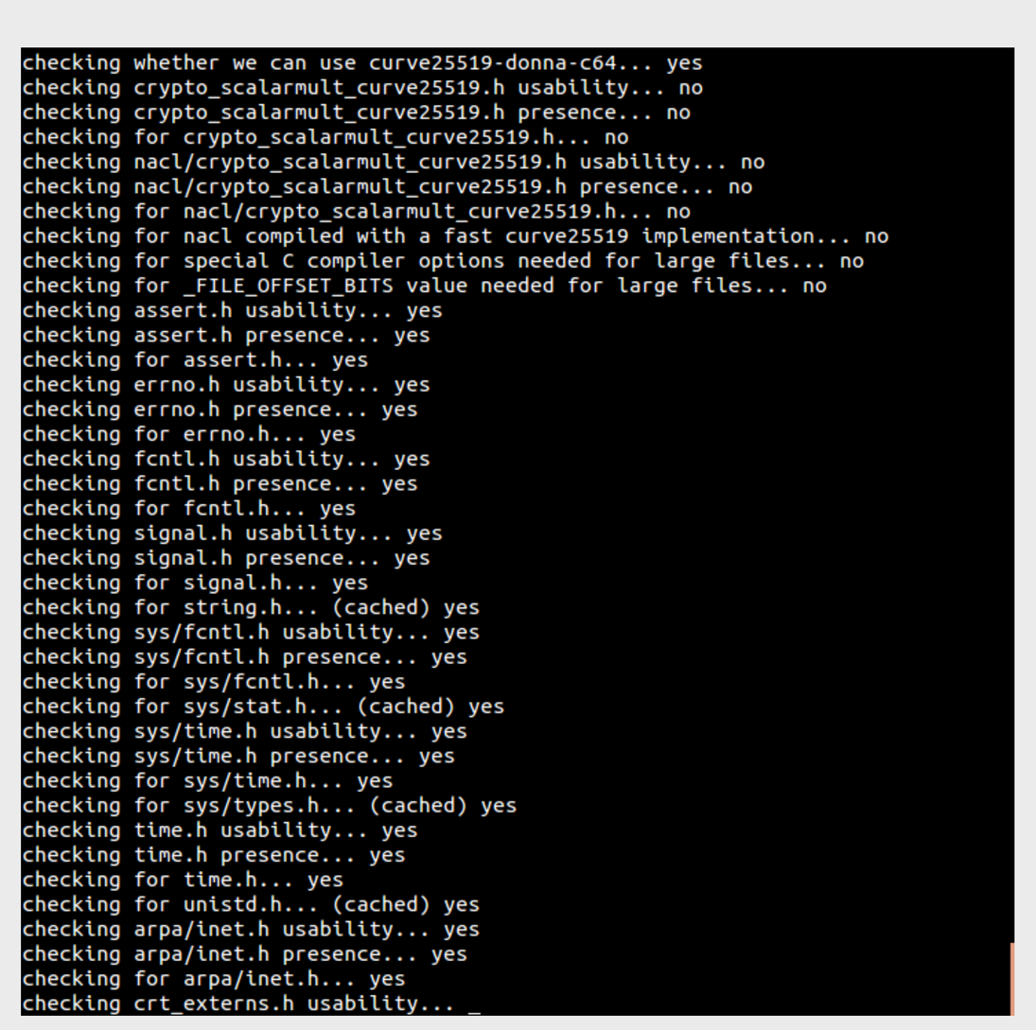
Methods Tested

Methods that underwent testing:

- round_to_next_multiple_of(int a, int b);
- round_to_power_of_2(int a);
- digit_to_num(char a);
- tor_strisnonupper(char* a);
- addr_mask_get_bits(uint32_t a);
- tor_llround(int a);

Why?

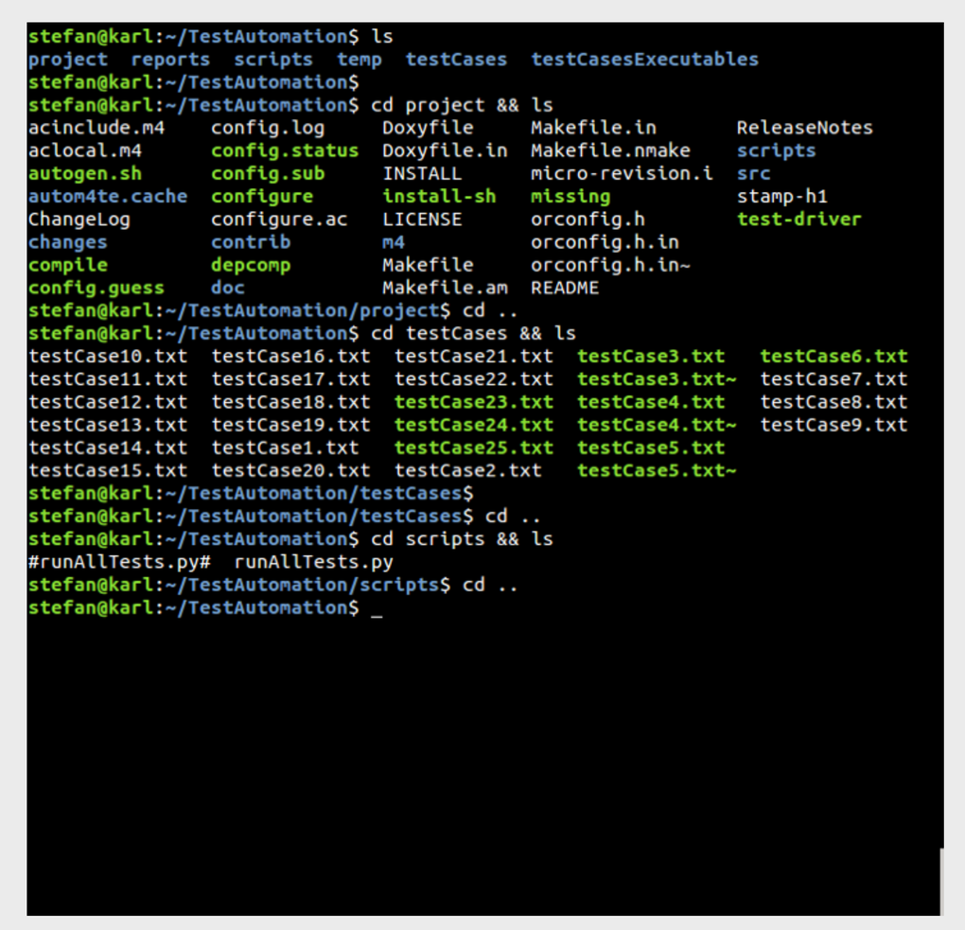
We chose these simple helper methods due to their accessibility and low number of dependencies.



Framework and Results

Results

The results of our testing framework concluded that the functions within Tor are well written and most of our cases passed. However, some cases required some kind of exception handling given invalid input. However this is really only true for cases that, given a user is running Tor, would not normally occur. In particular, the method *digit_to_num* which accepts a character variable, would continue to give valid output given that the input variable was not a character. Aside from *digit_to_num,* the remaining tested functions seemed to produce proper predicted output.



Test ID:	10
Time stamp:	Mon Nov 30 20:05:00 2015
requirement being tested:	given a char, return the char's numeric value
Component being tested:	util.c
Method being tested:	static int digit_to_num(char d)
Inputs used for testing:	a,,
Expected output:	error
results in a segmentation fault	

Test ID:	18
Time stamp:	Mon Nov 30 20:05:00 2015
requirement being tested:	Testing digit_to_num to ensure correct results
Component being tested:	util.c
Method being tested:	digit_to_num(char d)
Inputs used for testing:	3,,
Expected output:	3
The outcome was:	3
The test passed	

Test ID:	25
Time stamp:	Mon Nov 30 20:05:00 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	

file:///home/stefan/TestAutomation/reports/report.html

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:	11
Time stamp:	Mon Nov 30 20:30:46 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	

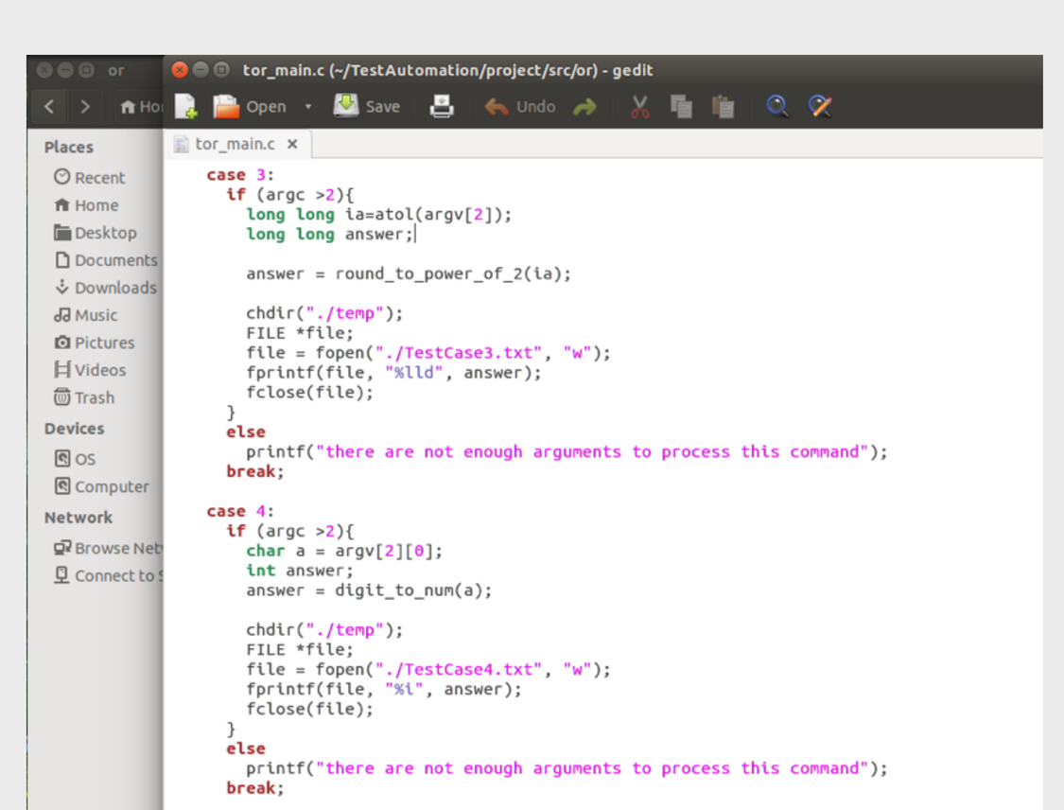
Test ID:	14
Time stamp:	Mon Nov 30 20:30:46 2015
requirement being tested:	Testing tor_llround to ensure correct results
Component being tested:	util.c
Method being tested:	int64_t tor_llround(double d)
Inputs used for testing:	5.3,,
Expected output:	5
The outcome was:	5
The test passed	

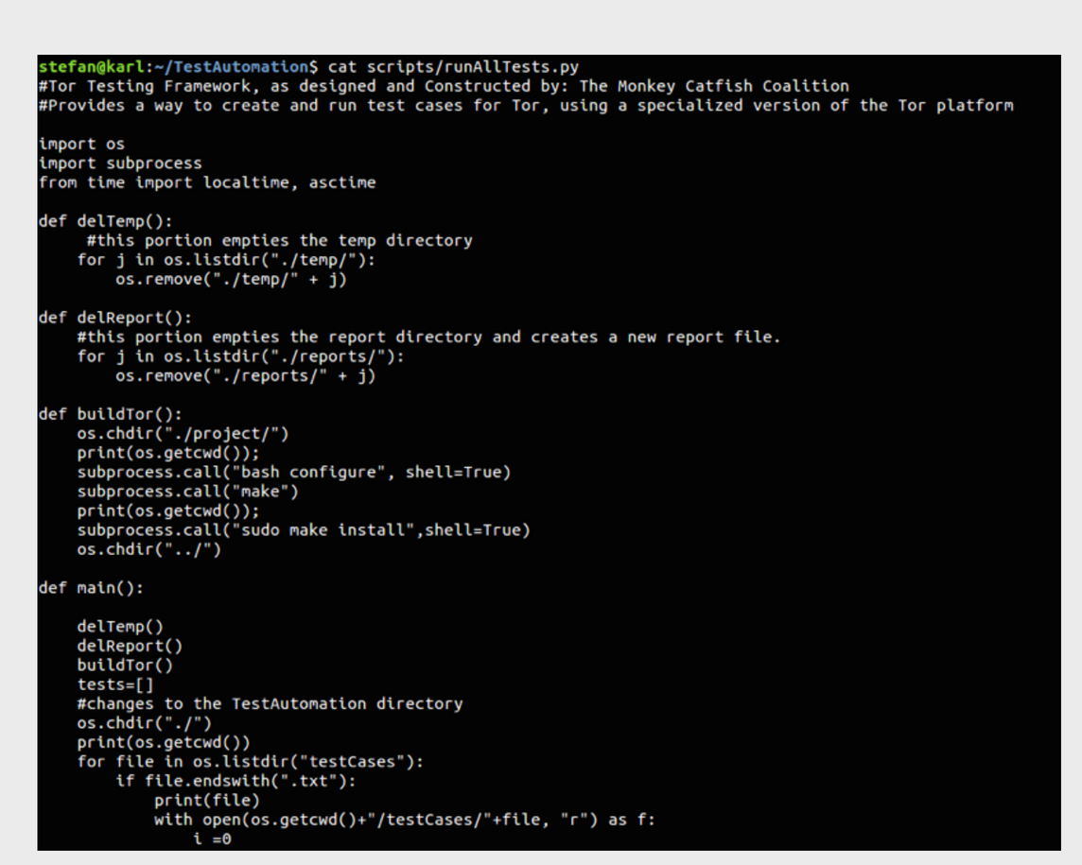
Test ID:	15
Time stamp:	Mon Nov 30 20:30:46 2015
requirement being tested:	Testing tor_llround to ensure correct results
Component being tested:	util.c
Method being tested:	int64_t tor_llround(double d)
Inputs used for testing:	-0.1,,
Expected output:	0

In the end, 16 of our 25 test cases passed. The remaining 9 failing tests are primarily due to the *digit_to_num* method and with our group still a little unfamiliar with the C language. But again, most of the inputs that yielded incorrect results are most likely unrealistic cases.

Framework

Our framework is a little different from the project specifications given that Tor is written in C. Our lead architect, Callum Brill, 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 temporary .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.





Thoughts On Results of implemented Faults

With the implementation of faults, we were able to further experience errors in our test cases. Although the fault injections were not entirely too complex (switching some variables and conditionals), all methods with faults still yielded correct output for at least one case each.
It should also be mentioned that depending on the specific requirements of whichever function was faulted, the incorrect output could have been misunderstood as correct output.

For example, *round_to_power_of_2* yields a number that was very close to the natural log base 2 of the inputted integer.

Original Test Case:

Test ID:	4
Time stamp:	Mon Nov 30 20:05:01 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	

Fault Injection:

Test ID:	4
Time stamp:	Mon Nov 30 19:55:31 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	

Goals and experiences

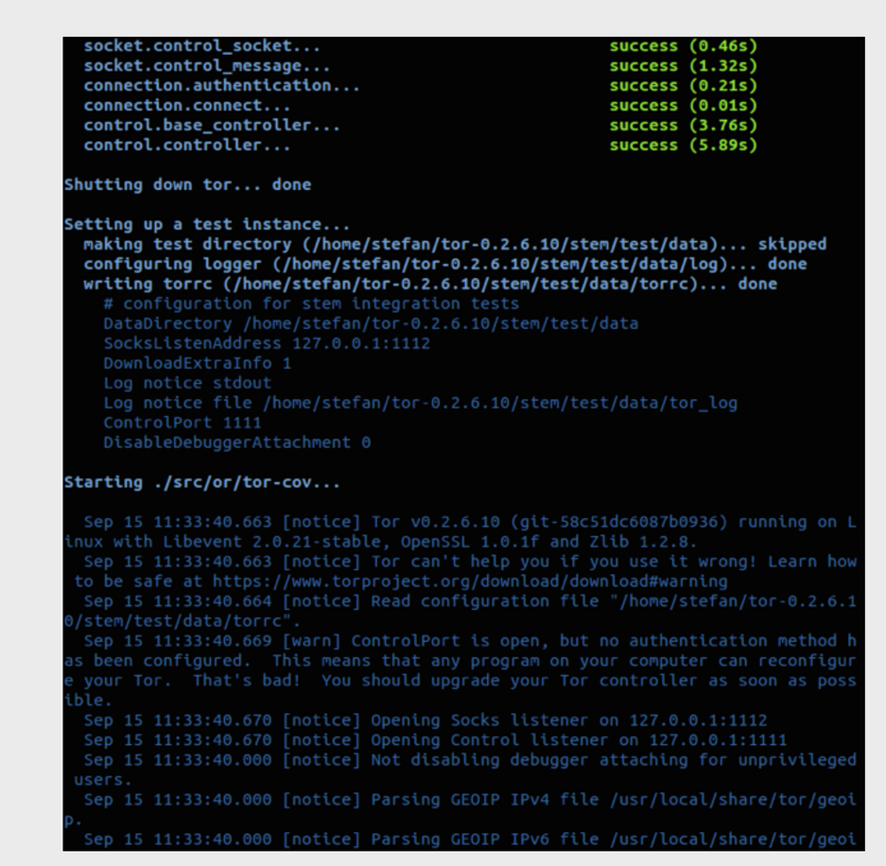
Goals and experiences

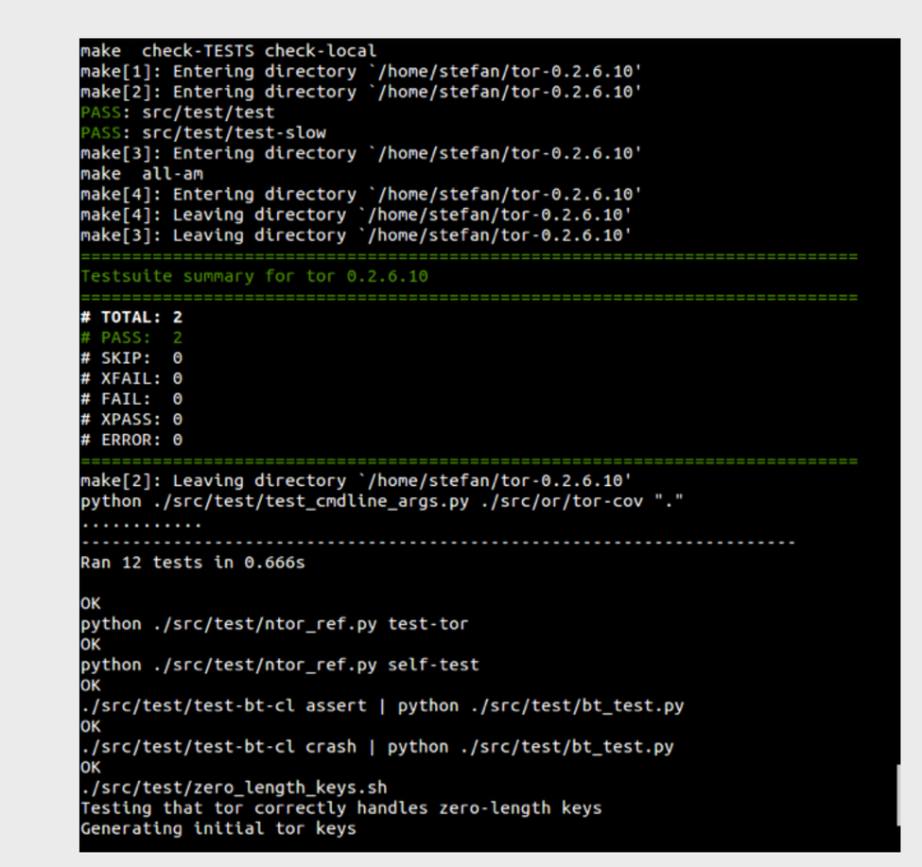
Our goal was to create a testing framework for utility functions within Tor's core source code. Most of the accessible functions were merely mathematical or involved string/character manipulation. This made them easier to predict and trace proper output. However, Tor's core is written in the C language. Prior to this project, most members of our group had little experience with C and its complexity. This was coupled by the fact that although most of Tor's code is easily accessible, readable, and well documented, some of it is not. This made some test case opportunities hard to predict and understand.

For example, we predicted faults for a few test cases to the function *digit_to_num*. But for these inputs that were meant to be invalid, we receive a more unforeseen output than we expected. This could be due to the fact that C is holding some data in memory locations that the function outputs even if the input is invalid.

Test ID:	8
Time stamp:	Mon Nov 30 20:05:00 2015
requirement being tested:	given a char, return the char's numeric value
Component being tested:	util.c
Method being tested:	static int digit_to_num(char d)
Inputs used for testing:	11,,
Expected output:	error
the outcome was:	1
The test failed	

Stem/Unit Tests in Tor





Disclosure

What was learned

We learned that working with an open source project as large and as complex as Tor is difficult. Although Tor is well documented, we still found difficulty and initially bit off more than we could chew. But although most of our tests cases were of simple helper methods, we were still able to complete the framework and thoroughly test our 25 test cases.