

11/21/2013

CSCI 362
SECTION 1



int team = 4;

Robert Bryant | Jeremy Jones | Lynn Kitchner | Evan Kosin | Davida Mitchell

TABLE OF CONTENTS

INTRODUCTION	3
I. Selecting a Project and Building Source Code	3
II. Snort	4
TEST PLAN	5
I. Testing Process	5
II. Requirements	5
III. Tested Items	6
IV. Requirements Traceability	8
V. Testing Schedule	9
VI. Testing Recording Procedures	10
VII. Hardware and Software Requirements	11
VIII. Constraints	11
TEST CASES	12
I. Initial Test Inputs	12
II. Constructing Initial Test Cases	13
III. Determining Methods to Test	14
TESTING FRAMEWORK	16
I. Dependencies	17

II. File Structure	17
III. Acquiring the Framework	18
IV. Building Snort Source Code	18
V. Architecture of Framework	19
VI. Using Test Case Template	20
VII. Running Testing Framework	22
VIII. C++ Code Created	22
IX. Interpreting Report	23
X. Experience Creating Framework	24
REMAINING TEST CASES	25
TEST BY INJECTING FAULTS	27
I. Running Faults Script	27
II. Fault #1	28
III. Fault #2	28
IV. Fault #3	29
V. Fault #4	30
VI. Fault #5	31
OVERALL EXPERIENCE	32
APPENDIX	33
1. Test Cases	33
2. Methods	49

INTRODUCTION

The purpose of this project is to test units of FOSS, or Free Open Source Software. The testing of these units will be black-box only, which is to test the functionality of the chosen methods. Testing of these methods will happen with a series of test cases, executed by an automated testing framework. These test cases are to be defined as a given set of inputs with expected outcomes, testing only a functional requirement. To better understand the evolution of the project, this document goes through the phases of our testing process.

I. Selecting a Project and Building Source Code

We picked Celestia, a space visualization software for educational purposes, as our first choice and began to build it. Building Celestia requires installing a lot of dependencies. It took a long time just to be able to have everything set up to build. When building, we ran into some issues. Celestia has errors in its code. This led us to change our project to OO4Kids, a simplified version of OpenOffice.org to allow ease of use for children. We then began trying to build OO4Kids with no results. OO4Kids has a very large repository, which took a long time to check out.

When building, we found that the build files for this project are incorrect and unable to be edited directly. We made a group decision to change our project again. This time we

chose Snort, an IDS (intrusion detection system). Snort is small and has few dependencies. Some of our team has already successfully built it. Snort has no existing tests since it is already tried and true. We've found that working with open source software in general is cumbersome. Not enough of the projects are built well. They all have specialized build environments that are not well documented.

II. Snort

Snort is an open source software originally created by Martin Roesch in 1998. Three years later Roesch founded Sourcefire, a network security company which currently develops Snort. Sourcefire was acquired by CISCO in July 2013. Snort is a network intrusion prevention and detection system which works by performing protocol analysis, content searching, and content matching. More information about this software, as well as it's source code, can be found on Snort's homepage, www.snort.org.

TEST PLAN

The purpose of this test plan is to guide the group through the phases of our testing process. With a clear set of objectives, this test plan will allow us to stay on schedule well before the final delivery dates. This plan communicates our testing objectives and will allow us to later adapt to change if necessary.

I. Testing Process

Phase I - Analyze source code to determine which methods to test

Phase II - Begin constructing test cases for methods

Phase III - Build and design automated testing framework

Phase IV - Creating rest of total required test cases

Phase V - Test by injecting faults into source code

Phase VI - Document results through visual presentation

II. Requirements

R1 Returns the percentage of the first argument as a part of the second argument.

R2 Returns the length of a null terminated string in the argument or
SNORT_STRNLEN_ERROR / -1 if not null terminated.

-
- R3 Returns the correct offset between gmt and local for the current time if it is standard time.
- R4 Returns the correct offset between gmt and local in unix time.
- R5 Returns the correct offset between gmt and local in unix time when the current time is during daylight savings time.
- R6 Returns the correct offset between gmt and local for the epoch (00:00:00, Jan 1, 1970).
- R7 Returns a flat string from an array of strings.
- R8 Returns if the function was able to copy a string.
- 0 = SNORT_STRNCPY_SUCCESS
- 1 = SNORT_STRNCPY_TRUNCATION
- 1 = SNORT_STRNCPY_ERROR
- R9 Does not compile if the correct parameters are not passed in.

III. Tested Items

We chose five methods from the file util.c for our component testing. These methods can be found in the appendix of this document.

- M1 double CalcPct(uint64_t cnt, uint64_t total);
- CalcPct calculates a percentage $\text{cnt}/\text{total} * 100$
- M2 int SnortStrnlen(const char *buf, int buf_size);
- Snort Strnlen determines whether a buffer is null terminated, if it is returns string length, if not returns -1.

M3 `int gm2local(time_t t);`

gm2local calculates an offset of the time zone you are in from GMT.

M4 `char* copy_argv(char **argv);`

copy_argv converts an array of strings into a flat string.

M5 `int SnortStrncpy(char *dst, const char *src, size_t dst_size);`

SnortStrncpy copies a string from one memory location to a new memory location.

IV. Requirements Traceability

Item \ Req	R1	R2	R3	R4	R5	R6	R7	R8	R9
M1	X								X
M2		X							X
M3			X	X	X	X			X
M4							X		X
M5								X	X

V. Testing Schedule

TIME PERIOD	OBJECTIVE
10/01 - 10/08	Write Test Plan Write up test cases (5) Define requirements
10/08 - 10/15	Begin to design testing framework Decide what will be tested Begin presentation design
10/15 - 10/22	Write up test cases (20) Begin writing scripts for testing automation
10/22 - 10/29	Complete creating testing framework Create architectural description Create 'How To'

10/29 - 11/05	Begin designing poster
	Begin to create presentation
11/05 - 11/12	Complete test cases (25)
	Create faults in code and test
	Final formatting of presentation
	Final details of poster
11/12 - 11/19	All phases complete
	Present to class

VI. Testing Recording Procedures

A report will be created every time the test script is run. This will be viewed as an HTML web page. For more information see the section titled *Interpreting Report* in this document.

The results of each test will be recorded in a chart containing:

testCaseNumber

requirement

inputs

expected outcomes

actual

pass/fail marker

VII. Hardware and Software Requirements

The following technologies are required to run the testing framework as well as building the source code for Snort:

- Virtual Machine
- Ubuntu 12.04/ Mint 15
- GCC and G++ versions included in the distos
- Python 2.7

VIII. Constraints

Factors that may restrict our abilities that will have to overcome are mostly technical.

Most of the team is a new user to GNU/Linux and the entire team is not familiar with C.

We will also be working around different school and work schedules to complete the phases of this project.

TEST CASES

I. Initial Test Inputs

This section shows the initial test cases created for CalcPct for purposes of planning.

Testing: double CalcPct(uint64_t cnt, uint64_t total);

Inputs	Expected Outcome
0, 0	0.0
1, 0	0.0
0, 1	0.0
1, 1	100.0
2, 1	200.0
1, 0xFFFFFFFFFFFFFFFF(in decimal)	0.0
A, 0	DNC
0, A	DNC
A, A	DNC
,	DNC
, 1	DNC

1,	DNC
-1, -1	1.0 w/ compiler warning

*DNC = does not compile

II. Constructing Initial Test Cases

These are 5 of the final test cases. They are all testing the same method, double CalcPct(uint64_t cnt, uint64_t total). More information about these text files can be found in the framework section of this document.

testCase101.txt

```
testNumber|101
requirement|Returns the correct value based on the
calculations
component|util.c
method|CalcPct
testInputs|0,0
oracle|0
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase102.txt

```
testNumber|102
requirement|Returns the correct value based on the
calculations
component|util.c
method|CalcPct
testInputs|1,1
oracle|100
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase103.txt

```
testNumber|103
```

```
requirement|Returns the correct value based on the
calculations
component|util.c
method|CalcPct
testInputs|0.5,0.5
oracle|0
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase104.txt

```
testNumber|104
requirement|Returns the correct value based on the
calculations
component|util.c
method|CalcPct
testInputs|1,2
oracle|50
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase105.txt

```
testNumber|105
requirement|Returns the correct value based on the
calculations
component|util.c
method|CalcPct
testInputs|2,1
oracle|200
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

III. Determining Methods to Test

Most files made heavy use of structs which complicated testing. To solve this problem, we have designed the testing script to insert a user defined portion of code which could be used to create dummy objects or instantiate structs. *util.c* is the file where we will probably find all tested methods for this project due to it's simplicity in dependencies.

TESTING FRAMEWORK

This testing suite will allow a user to create test cases that will be read and run through our testing harness. Templates are available to create your own test cases following our specified format. All test cases will be run through the component with the specified inputs, created into source code and run. The outputs are collected and inserted into a report with specific information about each test case.

I. Dependencies

You must check out the entire repository to run this testing suite. These are the required tools to be able to run this testing suite, and specifically to run it to test Snort.

TEST SUITE:

Unix Based System

BASH or BASH Compatible Shell

g++

cstdlib

Python 2.x

HTML5/CSS3 Compatible Browser

Subversion

C or C++

Snort Source Code

II. File Structure

Our repository is created within a version control system called Subversion. This is the structure of the repository. There are no spaces in any folder or file names. Check out the repository at the highest folder level. Always remember to update.

/TEAM4_TESTING

Team4_Deliverable1.pdf

Team4_Deliverable2.pdf

Team4_Deliverable3.pdf

Team4_Deliverable4.pdf

Team4_Deliverable5.pdf

/INT_TEAM_EQUALS_4

docs/

README.TXT

oracles/

project/

contains all source code for Snort

reports/

empty until script is run, then html report will be located here

resources/

css/

stylesheets for report

img/

images for report

js/

javascript files for report

templateCPP.cpp - *template to be used for created the C++*

code

testCaseTemplate.txt - *template for creating a test case*

scripts/

build - *script that will build all Snort source code for user*

faults - *script that activates injection of faults*

runAllTests - *script that runs testing framework*

temp/

testCases/

testCase101.txt

testCase102.txt

...

testCase201.txt

...

testCasesExecutables/

III. Acquiring the Framework

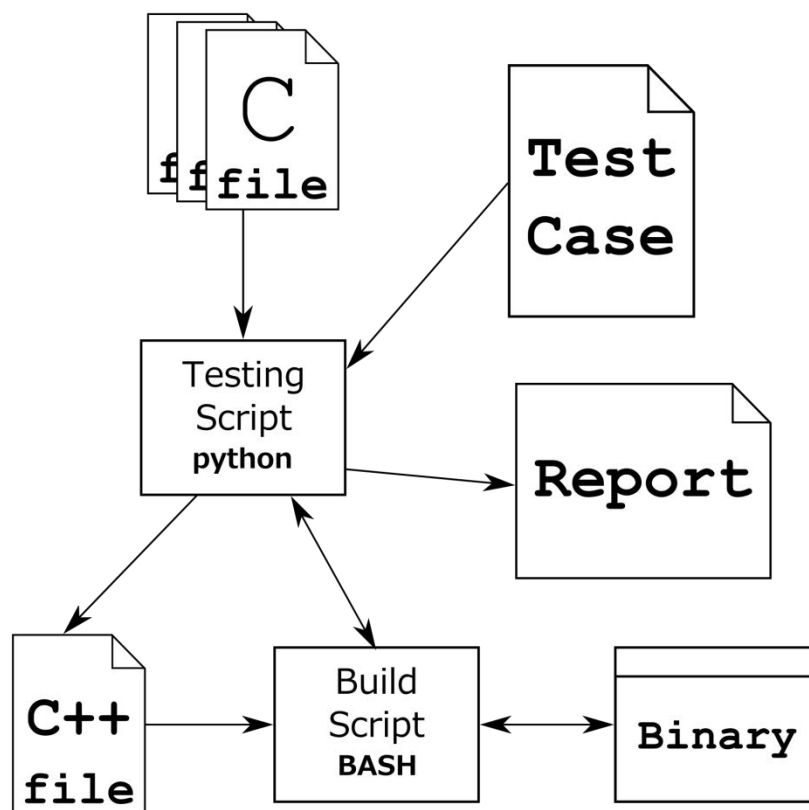
Use username/password to access team4 subversion. Checkout all files from the SVN repository from the top level (/TEAM4).

Location of repository: <https://svn.cs.cofc.edu/repos/CSCI362201302/team4>

IV. Building Snort Source Code

We have made a script to automatically build the source code for you. To do this, run the script found at 'scripts/build' inside the project folder. Alternatively, we have also made the source code available to you in our repository. This source code is contained in 'project/' subfolder. A useful tutorial can be found on the Xmodulo blog, located at <http://xmodulo.com/2013/08/how-to-compile-and-install-snort-from-source-code-on-ubuntu.html>.

V. Architecture of Framework



The chart above describes the architecture of our testing framework. Our testing script is written in a Python. The C files, from the Snort source code and test case text files are sent into the testing script. The testing script takes all of the information from the test case file to use to create each test. We have written our script to all for each

method to be searched for within the source code. From there, a C++ file is created to send the specified inputs, outputs, method, and dependencies. Everything is sent back to the script, which then creates an HTML report.

For more information about the C++ file, see section *C++ Code Created* below.

VI. Using Test Case Template

Reference the test case template located in 'resources/testCaseTemplate.txt'.

testnumber

Should be same as file name. Replace template with this number when saving.

requirement

Requirement being tested by the test case.

component

Component being tested. Needs to match that component's file name (case sensitive).

method

Name of method being tested. Only include the name, no parameters (case sensitive).

testinputs

Arguments used to call the function.

oracle

Expected outcome.

linenumber

Line number method is found in the component. These are inclusive.

outputtype

Data type the function is expected to output. The return type should explicitly match the return type of the function being tested, as it will be used to instantiate a variable for that return type.

~DEPENDENCIES~

Required dependencies should be placed here. Anything located at this point will be directly inserted into the resulting C++ file for compilation including:

- non-Local Variables

- non-Local Objects

- inherited Classes

- instantiated Classes

- included Interfaces

- required Libraries

- required #DEFINE

Note: If the component being testing requires the use of a C library, the equivalent C++ library needs to be included instead. To find this, you may go

<http://cplusplus.com/reference/clibrary>.

After each test case file is created, it must be saved as “testCase[number].txt”. The testing script will only grab files, from the ‘testCase/’ folder with this specific standard of file name. All other files in this folder will be ignored. Commit this text file to “testCase/” folder to be read by the script.

VII. Running Testing Framework

Make sure to update from the repository before running the testing suite. The script for the testing suite is located under ‘scripts/’. From the terminal, you can type ‘./runAllTests’.


The script can also be run from your local version of the repository in your file browser. You can click on the file called ‘runAllTests’ under the ‘scripts/’ folder. A message will appear, allowing you to either ‘run’ or ‘run in terminal’.

Running the script empties the report folder each time. After you run the script, all of the resource files needed for the report display will be piped over to this folder from the ‘resources/’ folder. The script will pop up in an internet browser window. The “report.html” file can also be found in this reports folder for later viewing. If you rerun the script, a new report will be generated.

VIII. C++ Code Created

Units are tested through direct compilation of the function. The function is extracted directly from the file being tested and is inserted into a C++ template. Any dependencies are inserted before the declaration of main in the template file and if an output is specified, the cout statement is replaced with the user defined code. This file is compiled for testing.

IX. Interpreting Report

 int team = 4;					
Test Case	Method	Inputs	Oracle	Output	Results
testCase101	CalcPct	0,0	0	0	Pass
testCase102	CalcPct	1,1	100	100	Pass
testCase103	CalcPct	0.5,0.5	0	0	Pass
testCase104	CalcPct	1,2	50	50	Pass
testCase105	CalcPct	2,1	200	200	Pass
testCase106	CalcPct	1,0xFFFFFFFF	5.42101e-18	5.42101e-18	Pass
testCase107	CalcPct	"A",1	DNC	DNC	Pass
testCase108	CalcPct	1,"A"	DNC	DNC	Pass
testCase109	CalcPct	"A","A"	DNC	DNC	Pass
testCase110	CalcPct	,	DNC	DNC	Pass
testCase111	CalcPct	,1	DNC	DNC	Pass
testCase112	CalcPct	1,	DNC	DNC	Pass
testCase113	CalcPct	-1,-1	100	100	Pass

Requirement: Does not compile if the correct parameters are not passed in.

Test Case

Name of the test case being run.

Method

The function or method being tested.

Inputs

The inputs passed to the function or method being tested.

Oracle

The expected output of the tested function or method.

Output

The actual output of the tested function or method.

Results

Pass if actual output matches expected output. Else, fail.

Requirement

This will show up when you mouse over the test case number. It is the functional requirement being tested.

X. Experience Creating Framework

Initially, during development of the script, we ran into a significant issue. The step which compiled the tested code was failing with the complaint that g++ was unable to find `main(int argc, char **argv)`. This left Evan confused, as the error did not appear to fit the context of any known problem. After considerable debugging and thinking, the conclusion was reached that the file needed to be closed. This solved the problem and the remainder of the script was smooth sailing.

From there, we were able to spruce up the look of our HTML report to make it more appealing to the user. We made many changes to our original design throughout the process. Evan and Lynn worked to create the CSS, HTML and Javascript behind the

design. With two visions as to where the design should go, we had to make a decision as to how the final report would look. Davida developed a logo to use as our team logo, which we displayed on the report.

REMAINING TEST CASES

Most of the work since creating the framework has been polishing. The specifications required 5 test cases for deliverable #3 and 25 test cases for deliverable #4. We already had more than 25 test cases by the third deliverable. We have updated our previous test case to have some specific requirements. Evan has made some modifications to the script and has been working on a search method that will make the framework more robust by removing the need for line numbers in the test cases. He also added the ability to have a custom output in case the return is a struct that needs to be checked in a manner that "cout" would not work well with.

We encountered a strange problem while trying to test another function. In "snprintf.c" they used the old K&R style function declarations where the parameter types come after the parameter list. None of us had seen that style before so we were a bit confused. Our script wouldn't compile the object files for these test cases as well. After some playing around we figured out that the g++ compiler doesn't support this type of declaration. We changed the output and compiled it with gcc and it worked fine. The problem that this introduces is that we assumed everything that would compile with gcc would compile with g++ and we built our framework accordingly. In the end we decided

to not support K&R style declaration as we would have to add a flag in the test case file and add more complexity to the script. Being that we are not supporting it we had to throw that function out.

At this point we have noticed that our framework is kind of limited. We found a very simple function at first to test and designed our framework around it. The way we have designed our framework may be too simple currently. We chose the easy story and assumed that all other stories would be of the same complexity which has made us make the compromises that we have had to. Creating test cases that are testing complex functions proves to be difficult. We may need to redesign a bit to allow those functions to be tested. Operating in an agile style has gotten things done, but using the waterfall development method could allow for a little more of the long term thinking. We are currently testing three functions. We plan to test five.

TEST BY INJECTING FAULTS

We did a final test by injecting errors, or faults, into the source code to make some of our test cases fail. For these tests we have modified the code of each of the tested functions.

I. Running Faults Script

We created separate source files with each fault activated as well as a file with all the faults activated. We also have a script that will activate each or all faults. In our scripts folder is “faults” and the following arguments tell it what to do:

Y or y - *activate all faults*

N or n - *restore original source code*

{1-5} - *activate the fault with the corresponding number*

**note: these arguments do not take a tack ("") in front of them*

II. Fault #1

In the CalcPct function of util.c we changed the operator from division to multiplication in line 149.

FAILED TEST CASES:

testCase104

testCase106

testCase113

Although this drastically changed the calculation of this function only 3 test cases failed, but two of those (104 and 106) are really the test cases that are most important in testing this function as they actually do a calculation that is meaningful. We had six test cases (107-112) that do not compile to begin with so they never got a chance to execute the fault. Three of our test cases (101-103) were using arguments that would give a false positive with this fault.

III. Fault #2

In the SnortStrnlen function we change the equality check in line 1695 to an inequality check. This line originally checked to make sure the length argument was in fact the size of the buffer.

FAILED TEST CASES:

testCase207

testCase210

This modification only cause two failures, but once again they are very significant failures. The change we made drastically changes the logic of the function so it is basically unusable. testCase207 failed testing because it was supposed to pass the equality check and now it is going around it. testCase210 is the opposite as it was supposed to fail which lets the function proceed.

IV. Fault #3

At line 297 in util.c in the function gmt2local we commented out two lines. These lines check to see if the input is 0 and if so to change it to the current time.

FAILED TEST CASES:

testCase309 (fails during normal execution during standard time also)

This is a tricky fault because whether or not it is currently daylight savings time or standard time will make certain test cases pass or fail depending on which requirements they have. 0 as the argument to this function tells it to grab the current time, but there is no way for it to calculate the offset at the epoch. The modification that we made reverses that situation where there is a way to get the offset at the epoch, but not at the current time. The epoch was during standard time which it is also currently so it passes those tests. If it was during daylight savings time testCase301 and testCase312 would fail. 312 would fail in normal execution as well during DST since its requirement is to calculate the offset at the epoch. Another thing of note is all the oracles for the test cases for this function are calculated using Eastern Time and if this test was run with a different time zone all the tests would fail.

V. Fault #4

The return variable was named 'buf' so we just extended it to 'buffer', but we didn't declare it anywhere so it is an invalid token.

FAILED TEST CASES:

testCase401

testCase403

testCase404

testCase405

This modification will cause any test case to fail unless 'buffer' is declared in dependencies. The test cases that passed were not supposed to compile to begin with so they are showing false positives. This shows the limitations of our framework as we do not have the capacity to distinguish between different compilation errors.

VI. Fault #5

In line 1638 of util.c in the SnortStrncpy function we changed the '`<=`' to '`<`'. This checks to make sure that the destination has at least 1 space in it to truncate the string into.

FAILED TEST CASES:

None

The only test case that this change would have affected is testCase506. All other test cases either fail to compile or pass the check anyway. We can venture to guess that the strncpy method called 5 lines afterwards returns NULL since the whole function returns SNORT_STRNCPY_ERROR (-1) and that is the only other place where that value is returned. The interesting thing is that return has comments above it saying "Not sure if this ever happens but might as well be on the safe side" so our fault is being caught by something that the original coder wasn't quite sure of.

OVERALL EXPERIENCE

We began our group exercises with the thought that we would work as a surgical team, where each person had a distinct role. We thought that this is how it would continue for the entire semester. This was not the case, however. The lines between each role began to disappear, and team members began to put forth their skills in all areas necessary. It was not simply one person doing documentation, and one person writing the script for example. We each had a part in all pieces of the project. This way may have been inefficient, but it really worked towards our benefit to create a quality project that we all felt proud to put our names on.

Although we had great success, we have all agreed that if we were to do a group project similar to this again, we would like to take on more of an Agile process. For this to occur, we would categorize items into stories. These stories would be then broken up into tasks necessary to complete each story. These tasks would be assigned to each person.

This project helped us to further develop or skills as software engineers. For some, it was their first experiences with Ubuntu. For others, it was their first experience creating an HTML page with CSS and Javascript. Everyone already knew Python, but we used it in a way we never had before. Overall, everyone has developed their skills as a member of a team. This experience alone will be invaluable for our future careers.

APPENDIX

I. Test Cases

testCase101.txt

testCase101.txt

```
testnumber|101
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|0,0
oracle|0
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase102.txt

```
testnumber|102
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|1,1
oracle|100
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase103.txt

```
testnumber|103
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|0.5,0.5
```

```
oracle|0
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase104.txt

```
testnumber|104
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|1,2
oracle|50
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase105.txt

```
testnumber|105
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|2,1
oracle|200
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase106.txt

```
testnumber|106
requirement|Returns the percentage of the first argument as a
part of the second argument.
component|util.c
method|CalcPct
testinputs|1,0xFFFFFFFFFFFFFFFF
oracle|5.42101e-18
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase107.txt

```
testnumber|107
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|CalcPct
testinputs|"A",1
oracle|DNC
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase108.txt

```
testnumber|108
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|CalcPct
testinputs|1,"A"
oracle|DNC
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase109.txt

```
testnumber|109
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|CalcPct
testinputs|"A","A"
oracle|DNC
linenumber|139-155
outputtype|double
~DEPENDENCIES~
#include <cstdint>
```

testCase110.txt

```
testnumber|110
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|CalcPct
```

```
testinputs| ,  
oracle|DNC  
linenumber|139-155  
outputtype|double  
~DEPENDENCIES~  
#include <cstdint>
```

testCase111.txt

```
testnumber|111  
requirement|Does not compile if the correct parameters are not  
passed in.  
component|util.c  
method|CalcPct  
testinputs| ,1  
oracle|DNC  
linenumber|139-155  
outputtype|double  
~DEPENDENCIES~  
#include <cstdint>
```

testCase112.txt

```
testnumber|112  
requirement|Does not compile if the correct parameters are not  
passed in.  
component|util.c  
method|CalcPct  
testinputs|1,  
oracle|DNC  
linenumber|139-155  
outputtype|double  
~DEPENDENCIES~  
#include <cstdint>
```

testCase113.txt

```
testnumber|113  
requirement|Returns the percentage of the first argument as a  
part of the second argument.  
component|util.c  
method|CalcPct  
testinputs|-1,-1  
oracle|100  
linenumber|139-155  
outputtype|double  
~DEPENDENCIES~  
#include <cstdint>
```

testCase201.txt

```
testnumber|201
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
method|SnortStrnlen
testinputs|NULL,NULL
oracle|-1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase202.txt

```
testnumber|202
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
method|SnortStrnlen
testinputs|NULL,1
oracle|-1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase203.txt

```
testnumber|203
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs|'a',NULL
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase204.txt

```
testnumber|204
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
```

```
method|SnortStrnlen
testinputs|"a",NULL
oracle|-1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase205.txt

```
testnumber|205
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs|'a',0
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase206.txt

```
testnumber|206
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
method|SnortStrnlen
testinputs|"a",0
oracle|-1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase207.txt

```
testnumber|207
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
method|SnortStrnlen
testinputs|"a",1
oracle|-1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
```

```
#define SNORT_STRNLEN_ERROR -1
```

testCase208.txt

```
testnumber|208
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs|'a',1
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase209.txt

```
testnumber|209
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs|'a',2
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase210.txt

```
testnumber|210
requirement|Returns the length of a null terminated string in
the argument or SNORT_STRNLEN_ERROR / -1 if not null terminated.
component|util.c
method|SnortStrnlen
testinputs|"a",2
oracle|1
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase211.txt

```
testnumber|211
requirement|Does not compile if the correct parameters are not
passed in.
```

```
component|util.c
method|SnortStrnlen
testinputs| ,
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase212.txt

```
testnumber|212
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs| ,0
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase213.txt

```
testnumber|213
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrnlen
testinputs|"a",
oracle|DNC
linenumber|1682-1699
outputtype|int
~DEPENDENCIES~
#define SNORT_STRNLEN_ERROR -1
```

testCase301.txt

```
testnumber|301
requirement|Returns the correct offset between gmt and local for
the current time if it is standard time.
component|util.c
method|gmt2local
testinputs|0
oracle|-18000
linenumber|279-315
outputtype|int
```

~DEPENDENCIES~

#include <time.h>

testCase302.txt

testnumber|302

requirement|Returns the correct offset between gmt and local in unix time.

component|util.c

method|gmt2local

testinputs|8192

oracle|-18000

linenumber|279-315

outputtype|int

~DEPENDENCIES~

#include <time.h>

testCase303.txt

testnumber|303

requirement|Returns the correct offset between gmt and local in unix time.

component|util.c

method|gmt2local

testinputs|31536000

oracle|-18000

linenumber|279-315

outputtype|int

~DEPENDENCIES~

#include <time.h>

testCase304.txt

testnumber|304

requirement|Returns the correct offset between gmt and local in unix time.

component|util.c

method|gmt2local

testinputs|1

oracle|-18000

linenumber|279-315

outputtype|int

~DEPENDENCIES~

#include <time.h>

testCase305.txt

testnumber|305

```
requirement|Returns the correct offset between gmt and local in
unix time.
component|util.c
method|gmt2local
testinputs|-1
oracle|-18000
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase306.txt

```
testnumber|306
requirement|Returns the correct offset between gmt and local in
unix time.
component|util.c
method|gmt2local
testinputs|2147483648
oracle|-18000
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase307.txt

```
testnumber|307
requirement|Returns the correct offset between gmt and local in
unix time.
component|util.c
method|gmt2local
testinputs|86399
oracle|-18000
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase308.txt

```
testnumber|308
requirement|Returns the correct offset between gmt and local in
unix time.s
component|util.c
method|gmt2local
testinputs|15897600
oracle|-14400
```

```
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase309.txt

```
testnumber|309
requirement|Returns the correct offset between gmt and local in
unix time when the current time is during daylight savings time.
component|util.c
method|gmt2local
testinputs|0
oracle|-14400
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase310.txt

```
testnumber|310
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|gmt2local
testinputs|"a"
oracle|DNC
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase311.txt

```
testnumber|311
requirement|Returns the correct offset between gmt and local in
unix time.
component|util.c
method|gmt2local
testinputs|'a'
oracle|-18000
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase312.txt

```
testnumber|312
requirement|Returns the correct offset between gmt and local for
the epoch (00:00:00, Jan 1, 1970).
component|util.c
method|gmt2local
testinputs|0
oracle|-18000
linenumber|279-315
outputtype|int
~DEPENDENCIES~
#include <time.h>
```

testCase401.txt

```
testnumber|401
requirement|Returns a flat string from an array of strings.
component|util.c
method|copy_argv
testinputs|input
oracle|a a
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[3] = {"a","a"};
```

```
int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase402.txt

```
testnumber|402
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|copy_argv
testinputs|input
oracle|DNC
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[];

int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase403.txt

```
testnumber|403
requirement|Returns a flat string from an array of strings.
component|util.c
method|copy_argv
testinputs|input
oracle|a a
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[2] = {"a a"} ;

int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase404.txt

```
testnumber|404
requirement|Returns a flat string from an array of strings.
component|util.c
method|copy_argv
testinputs|input
oracle|a aa
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[3] = {"a","aa"} ;

int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase405.txt

```
testnumber|405
requirement|Returns a flat string from an array of strings.
component|util.c
method|copy_argv
testinputs|input
oracle|a
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[3] = {"a", ""};

int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase406.txt

```
testnumber|406
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|copy_argv
testinputs|input
oracle|DNC
linenumber|320-376
outputtype|char*
~DEPENDENCIES~
#include <string.h>
```

```
char *input[] = {'a','a'};

int FatalError(const char *format,...){
    exit(1);
return 0;
}
```

testCase501.txt

```
testnumber|501
requirement|Returns if the function was able to copy a string.
0 = SNORT_STRNCPY_SUCCESS, 1 = SNORT_STRNCPY_TRUNCATION, -1 =
SNORT_STRNCPY_ERROR.
component|util.c
```

```
method|SnortStrncpy
testinputs|dst,"Hello",3
oracle|1
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char dst[3];
```

testCase502.txt

```
testnumber|502
requirement|Returns if the function was able to copy a string.
0 = SNORT_STRNCPY_SUCCESS, 1 = SNORT_STRNCPY_TRUNCATION, -1 =
SNORT_STRNCPY_ERROR.
component|util.c
method|SnortStrncpy
testinputs|dst,"Hello",9
oracle|0
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char dst[9];
```

testCase503.txt

```
testnumber|503
requirement|Returns a flat string from an array of strings.
component|util.c
method|SnortStrncpy
testinputs|"Hello",9
oracle|DNC
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char dst[9];
```

testCase504.txt

```
testnumber|504
requirement|Does not compile if the correct parameters are not
passed in.
component|util.c
method|SnortStrncpy
testinputs|dst,9
oracle|DNC
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char dst[9];
```

testCase505.txt

```
testnumber|505
requirement|Returns if the function was able to copy a string.
0 = SNORT_STRNCPY_SUCCESS, 1 = SNORT_STRNCPY_TRUNCATION, -1 =
SNORT_STRNCPY_ERROR.
component|util.c
method|SnortStrncpy
testinputs|dst,"Hello",6
oracle|0
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char dst[6];
```

testCase506.txt

```
testnumber|506
requirement|Returns if the function was able to copy a string.
0 = SNORT_STRNCPY_SUCCESS, 1 = SNORT_STRNCPY_TRUNCATION, -1 =
SNORT_STRNCPY_ERROR.
component|util.c
method|SnortStrncpy
testinputs|dst,"Hello",0
oracle|-1
```

```
outputtype|int
~DEPENDENCIES~
#include <string.h>
#define SNORT_STRNCPY_SUCCESS 0
#define SNORT_STRNCPY_TRUNCATION 1
#define SNORT_STRNCPY_ERROR -1

char * dst;
```

2. Methods

CalcPct

```
/* Function: CalcPct(uint64_t, uint64_t)
 *
 * Purpose: Calculate the percentage of a value compared to a
total
 *
 * Arguments: cnt => the numerator in the equation
 *            total => the denominator in the calculation
 *
 * Returns: pct -> the percentage of cnt to value
 */

double CalcPct(uint64_t cnt, uint64_t total)
{
    double pct = 0.0;

    if (total == 0.0)
    {
        pct = (double)cnt;
    }
    else
    {
        pct = (double)cnt / (double)total;
    }

    pct *= 100.0;

    return pct;
}
```

SnortStrnlen

```
/* Determines whether a buffer is '\0' terminated and returns
the
* string length if so
*
* returns the string length if '\0' terminated
* returns SNORT_STRNLEN_ERROR if not '\0' terminated
*/

int SnortStrnlen(const char *buf, int buf_size)
{
    int i = 0;

    if (buf == NULL || buf_size <= 0)
        return SNORT_STRNLEN_ERROR;

    for (i = 0; i < buf_size; i++)
    {
        if (buf[i] == '\0')
            break;
    }

    if (i == buf_size)
        return SNORT_STRNLEN_ERROR;

    return i;
}
```

gmt2local

```
/* Function: gmt2local(time_t)
*
* Purpose: Figures out how to adjust the current clock reading
based on the
*          timezone you're in.  Ripped off from TCPdump.
*
* Arguments: time_t => offset from GMT
*
* Returns: offset seconds from GMT
*/

int gmt2local(time_t t)
```

```
{
    register int dt, dir;
    register struct tm *gmt, *loc;
    struct tm sgmt;

    if(t == 0)
        t = time(NULL);

    gmt = &sgmt;
    *gmt = *gmtime(&t);
    loc = localtime(&t);

    dt = (loc->tm_hour - gmt->tm_hour) * 60 * 60 +
        (loc->tm_min - gmt->tm_min) * 60;

    dir = loc->tm_year - gmt->tm_year;

    if(dir == 0)
        dir = loc->tm_yday - gmt->tm_yday;

    dt += dir * 24 * 60 * 60;

    return(dt);
}
```

***copy_argv**

```
/* Purpose: Copies a 2D array (like argv) into a flat string.
Stolen from
*          TCPDump.
*
* Arguments: argv => 2D array to flatten
*
* Returns: Pointer to the flat string
*/
```

```
char *copy_argv(char **argv)
{
    char **p;
    u_int len = 0;
    char *buf;
    char *src, *dst;
    //void ftlerr(char *,...);
```

```
p = argv;
if(*p == 0)
    return 0;

while(*p)
    len += strlen(*p++) + 1;

buf = (char *) calloc(1,len);

if(buf == NULL)
{
    FatalError("calloc() failed: %s\n", strerror(errno));
}
p = argv;
dst = buf;

while((src = *p++) != NULL)
{
    while((*dst++ = *src++) != '\0');
    dst[-1] = ' ';
}

dst[-1] = '\0';

/* Check for an empty string */
dst = buf;
while (isspace((int)*dst))
    dst++;

if (strlen(dst) == 0)
{
    free(buf);
    buf = NULL;
}

return buf;
}
```

SnortStrncpy

```
/* Guaranteed to be '\0' terminated even if truncation occurs.
 *
 * Arguments:  dst - the string to contain the copy
 *            src - the string to copy from
```

```
*          dst_size - the size of the destination buffer
*          including the null byte.
*
* returns SNORT_STRNCPY_SUCCESS if successful
* returns SNORT_STRNCPY_TRUNCATION on truncation
* returns SNORT_STRNCPY_ERROR on error
*
* Note: Do not set dst[0] = '\0' on error since it's possible
that
* dst and src are the same pointer - it will at least be null
* terminated in any case
*/
int SnortStrncpy(char *dst, const char *src, size_t dst_size)
{
    char *ret = NULL;

    if (dst == NULL || src == NULL || dst_size <= 0)
        return SNORT_STRNCPY_ERROR;

    dst[dst_size - 1] = '\0';

    ret = strncpy(dst, src, dst_size);

    /* Not sure if this ever happens but might as
    * well be on the safe side */
    if (ret == NULL)
        return SNORT_STRNCPY_ERROR;

    if (dst[dst_size - 1] != '\0')
    {
        /* result was truncated */
        dst[dst_size - 1] = '\0';
        return SNORT_STRNCPY_TRUNCATION;
    }

    return SNORT_STRNCPY_SUCCESS;
}
```