KT Advance User Manual

Kestrel Technology, LLC

November 24, 2017

Contents

1 Quick Start

1.1 System Requirements

1.1.1 Platform: MacOSX or Linux

The KT Advance C Analyzer consists of three components that may be run on different platforms:

1. **Parser:** A Mac/Linux executable (parseFile) that takes as input a preprocessed C source file and produces a set of xml files that precisely represent the semantics of the C source file. This program is an extension of the CIL parser front end, developed by George Necula, at UC Berkeley [3], an now maintained by INRIA in France. Except for very simple programs (typically programs that do not include any standard libraries), it is recommended to run the parser on a Linux platform, as the parser pulls in definitions from the standard header files resident on the system.
2. **C Analyzer:** A Mac/Linux executable (ktadvance) that takes as input the semantics files produced by the parser as well as analysis results files, if available, and produces a set of xml files that hold analysis results. This executable will be wrapped in a license manager to protect the contained intellectual property. The C Analyzer operates solely on the semantics files produced by the parser without any other dependencies on the local system, and thus it can be run equally well on Mac or Linux.
3. **PyAdvance:** Python code, provided as source code to licensed users, that performs linking and provides various analyzer invocation, integration, and reporting services. All reporting scripts (scripts whose name typically start with `chc_report_` or `chc_show_` rely only on python code and thus can (theoretically) be run on any platform that has python installed, including Windows platforms. Thus if analysis has been performed on a server and the results saved, the analysis results can then be viewed and queried by anyone with access to these results.

1.1.2 Utility Programs

The front-end parser makes use of the utility `bear` to record and reply the actions performed by the Make file when compiling an application. This utility is usually available via a package manager.

1.1.3 Other Dependencies

The analyzer and python scripts make use of jar files; being able to extract these requires a working Java installation.

1.2 Organization

The `ktadvance` repository has three top directories:

1. `advance:` python scripts and programs to run the analysis and view the results; Section **??** describes these scripts and programs in more details.
2. `doc:` KT Advance User Manual (this document) and a Reference Manual that explains the analysis approach and describes in detail the data representation of all intermediate data artifacts and analysis results data.
3. `tests:` regression tests and other test cases, several of which have been pre-parsed and are ready for analysis. The reason for having pre-parsed applications is to provide reference applications that enable longitudional study of analyis performance. Several of the test directories have dedicated scripts for parsing and analysis, as described in Section **??**.

1.3 General Use Guidelines

The analysis consists of three phases that may be performed on different platforms.

1. **Parsing:** This phase takes as input the original source code, a Makefile (if there is more than one source file), and, in case of library includes, the library header files resident on the system. This phase produces as output a set of xml files that completely capture the semantics of the application, and are the sole input for the Analysis phase.
   Because of the dependency on the resident system library header files it is generally recommended to perform this phase of the analysis on a Linux system, because of its more standard library environment than MacOSX (the CIL parser also may have issues with some of the Darwin constructs on MacOSX).
   For several of the test cases in `tests/sard/kendra` and for all of the test cases in `tests/sard/zitser` and `tests/sard/juliet_v1.3` the parsing step has already been performed (on Linux) and the resulting artifacts are checked in in files named `semantics_linux.tar.gz`. These gzipped tar files contain all xml files necessary for the analysis, and thus to analyze these files the parsing phase can be skipped altogether.
2. **Analysis:** This phase takes as input the xml files produced by the parsing phase. As long as the source code is not modified, the analysis can be run several times without having to repeat the parsing step. The Analysis step can be run on either MacOSX or

Linux, independently of where the parsing step was performed, as it operates solely on the xml files produced and is not dependent on any external programs or library headers.

3. **Viewing Results:** All analysis results are saved in a directory with the name `semantics` in the analysis directory. Various reporting scripts are provided to process and view these results. These scripts rely only on python code and thus can be run on any platform once the analysis results have been produced.

1.4 Getting Started

All interactions with the KT Advance C Analyzer are performed via python scripts from the command line. All scripts have been tested to work with python 2.7. An effort has been made, however, to have all python code also compliant with python 3.x.

All scripts to interact with the analyzer are in the directory `ktadvance/advance/cmdline`. This directory has a few subdirectories with scripts dedicated to some of the test sets in the tests directory, as follows:

- **kendra:** scripts to analyze and report on the test cases in `tests/sard/kendra`;
- **zitser:** scripts to analyze and report on the test cases in `tests/sard/zitser`;
- **juliet:** scripts to analyze, score, and report on the test cases in `tests/sard/juliet_v1.3`.

Two other subdirectories have scripts to parse, analyze, and report on any c file or c application:

- **sfapp:** scripts to parse and analyze an application that consists of a single c file that be compiled directly with gcc (without a Make file).
- **mfapp:** scripts to parse and analyze an application that comes with a Makefile. It is expected that the Makefile exists (that is, a configure script has already been run, if necessary).

Sections 2 through 4 provide a detailed description and walkthrough of the scripts available in the test-specific cmdline directories and Sections 5 and 6 describe the scripts for the general case of applications consisting of a single c file and applications consisting of multiple c files, respectively. In general, user scripts start with the prefix `chc_` followed by some verb that indicates the action performed; most scripts have a –help command-line option that describes the arguments expected.

2 Running Test Cases: Kendra

The `ktadvance/tests/sard/kendra` directory contains a collection of very small test programs retrieved from the NIST Software Assurance Reference Dataset (`samate.nist.gov`). These programs are a subset of the collection of test cases developed by Kendra Kratkiewicz [2]. These test cases serve as a good first illustration of the KT Advance analysis approach and presentation of results. Section 2.3 has step-by-step instructions how to run these tests and some comments on particular cases.

2.1 Organization

The test cases are organized in groups of four related test cases, where the first three test cases have a given vulnerability with varying magnitude of overflow and in the fourth case that vulnerability is fixed (or absent). The names of the tests refer to the sequence numbers in the SARD repository, and the name of the group refers to the sequence number of the first test. For example, the test group id115Q contains the test cases id115.c, id116.c, id116.c, and id117.c .

The kendra tests are also used as regression tests for generating and discharging proof obligations. Each test directory has a reference file [testname].json (e.g., id115Q.json) that lists all proof obligations and their expected proof status, against which the analysis results are checked after each test run.

2.2 Summary of Scripts

Scripts to analyze and report on the kendra test cases are located in the `advance/cmdline/kendra` directory. They are (in alphabetical order):

- `chc_clean_kendraset.py`: removes the `semantics` for the given kendra set.
  Example: `python chc_clean_kendraset.py id115Q`
- `chc_kendra_dashboard.py`: outputs a summary of the results of all kendra sets (after analysis has been performed for all of them).
  Example: `python chc_kendra_dashboard.py`
- `chc_list_kendrasets.py`: outputs a list of all kendra sets.
  Example: `python chc_list_kendrasets.py`
- `chc_report_kendratest_file.py`: outputs a report of all proof obligations and their analysis results for a given kendra c file.
  Example: `python chc_report_kendratest_file.py id115.c`

- `chc_show_kendra_file_table.py`: outputs the entries for a particular kendra file in a given (file-level) data dictionary
  Example: `python chc_show_kendra_file_table id115.c --table predicate`
  Example: `python chc_show_kendra_file_table id115.c --list`
- `chc_show_kendra_function_table.py`: outputs the entries for a particular kendra file in a given (function-level) data dictionary
  Example: `python chc_show_kendra_function_table id115.c main --table ppo_type` *Example:* `python chc_show_kendra_function_table id115.c main --list`
- `chc_show_kendraset.py`: outputs a list of proof obligations for each file in the set
  Example: `python chc_show_kendraset.py id115Q`
- `chc_test_kendraset.py`: (parses and) analyzes the c files in the given test set
  Example: `python chc_test_kendraset.py id115Q`
  Example: `python chc_test_kendraset.py id115Q --verbose`

Below we give a more detailed walkthrough and illustration for invoking some of these scripts.

2.3 Running the Tests

Set the `PYTHONPATH` environment variable (or adapt for a different location of the ktadvance directory):

```
> export PYTHONPATH=$HOME/ktadvance
```

To see a list of test sets currently provided in the `tests/sard/kendra` directory:

```
> cd ktadvance/advance/cmdline/kendra
> python chc_list_kendratests.py
```

To run the analysis of a test set (staying in the cmdline/kendra directory):

```
> python chc_test_kendra_set.py id115Q
```

(or any other of the test set names provided in the list displayed earlier.) This will print the summary results for the four test programs included:

```
File          Parsing        PPO Gen        SPO Gen      PPO Results    SPO Results
------------------------------------------------------------------------------------
id115.c          ok             ok             ok             ok             ok
id116.c          ok             ok             ok             ok             ok
id117.c          ok             ok             ok             ok             ok
id118.c          ok             ok             ok             ok             ok
```

indicating that all stages ran without error, all proof obligations were correctly generated and the analysis results are as expected. To see more of the output while running the test, add –verbose as a command-line option.

To see which proof obligations are included in the test cases, with line numbers and expected proof status:

```
> python chc_show_kendra_test.py id115Q

id115.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id116.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id117.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id118.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      safe
      56  cast                   safe
```

When a test case has been analyzed the analysis results are saved in the `semantics/ktadvance` directory and are available for inspection and to reporting scripts. To see a full report, including code, proof justifications, and summary for an individual test file (note the filename):

```
> python chc_report_kendratest_file.py id115.c

Function main
--------------------------------------------------------------------------------
50  int main(int argc, char *argv[])
--------------------------------------------------------------------------------
Api:
  parameters:
    int[] argc
```

```
    ((char[] *) *) argv

  -- no assumptions

  -- no postcondition requests

  -- no postcondition guarantees

  -- no library calls
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
51  {
52    char buf[10];
53
54
55    /*  BAD  */
56    buf[4105] = 'A';
--------------------------------------------------------------------------------
<S>    1     56  index-lower-bound(4105) (safe)
                  index value 4105 is non-negative
<*>    2     56  index-upper-bound(4105,bound:10) (violation)
                  index value 4105 violates upper bound 10
<S>    3     56  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
--------------------------------------------------------------------------------

Primary Proof Obligations
functions    stmt   local     api    post  global    open   total
---------------------------------------------------------------
main          3       0       0       0       0       0       3
---------------------------------------------------------------
total         3       0       0       0       0       0       3
percent  100.00    0.00    0.00    0.00    0.00    0.00


Proof Obligation Statistics for file id115
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Primary Proof Obligations
                             stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
cast                            1       0       0       0       0       0       1
index-lower-bound               1       0       0       0       0       0       1
index-upper-bound               1       0       0       0       0       0       1
---------------------------------------------------------------------------------
total                           3       0       0       0       0       0       3
percent                    100.00    0.00    0.00    0.00    0.00    0.00
```

For each line of code that has associated proof obligations the report shows the proof obligation predicate, whether it is valid (safe, indicated by <S>) or violated (indicated by <*>), and the reason for the assessment. For the safe case, id118.c, the output shows that the proof obligation for the upper bound is indeed safe.

```
> python chc_report_kendratest_file.py id118.c

....

--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
51  {
52    char buf[10];
53
54
55    /*  OK  */
56    buf[9] = 'A';
--------------------------------------------------------------------------------
<S>    1     56  index-lower-bound(9) (safe)
                  index value 9 is non-negative
<S>    2     56  index-upper-bound(9,bound:10) (safe)
                  index value 9 is less than bound 10
<S>    3     56  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
--------------------------------------------------------------------------------

....
```

For a list of all proof obligation predicates and their meaning, please see the KT Advance Reference Manual in this directory.

To analyze all kendra test cases run

```
> python chc_test_kendra_sets.py
```

If all of them complete all analysis results are available for inspection and reporting. Below we discuss some special cases.

2.4 Some Kendra Examples Discussed

2.4.1 Id151Q: Library Function Postconditions and Macros

The tests in this set program call two library functions: assert and malloc. Proving the memory safety of this program requires knowledge of the semantics of both of these library functions, in this case in particular the postconditions of these functions. To obtain this information the analysis makes use of library function summaries (provided in `advance/summaries/cchsummaries.jar`).

The malloc summary includes a postcondition that states that the return value points to a newly allocated region of memory with size (in bytes) given by the first argument, or the return value is NULL. Proof obligation 34 uses this information to determine that the buffer access on line 60 violates its bounds. Similarly in program id154.c (the safe version) the same information is used to prove the buffer access safe.

The assert call is actually a macro that expands into a conditional expression that calls a function `__assert_fail`. In particular, the call

```
assert (buf != NULL);
```

expands into

```
((buf != ((void *)0)) ? (void) (0) : __assert_fail ("buf != NULL", "id151.c", 57,
                                                    __PRETTY_FUNCTION__))
```

which by CIL gets represented as

```
if ((caste(unsigned long,buf) != caste(unsigned long,caste((void *),0))))
  { }
else
  { __assert_fail("buf != NULL","id151.c",57, "main"); }
```

which explains the many proof obligations that get generated for this seemingly simple instruction.

The `__assert_fail` function has postcondition `false`, that is, it does not return. This means only the then branch, with condition `buf != NULL`, continues, based on which proof obligation 31 (not-null(buf)) can be proven safe.

The results for id151.c are

```
Function main
--------------------------------------------------------------------------------
52  int main(int argc, char *argv[])
--------------------------------------------------------------------------------
Api:
  parameters:
    int[] argc
    ((char[] *) *) argv

  postcondition guarantees:
   post-expr(eq,return-val,num-constant(0))

  library calls:
   assert:__assert_fail -- 1
   stdlib:malloc -- 1
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
53  {
54    char * buf;
55
56    buf = (char *) malloc(10 * sizeof(char));
--------------------------------------------------------------------------------
<S>    1     56  int-underflow(10,sizeof(char[]),op:mult,ikind:iulong) (safe)
                  underflow is well defined for unsigned types
<S>    2     56  int-overflow(10,sizeof(char[]),op:mult,ikind:iulong) (safe)
                  overflow is well defined for unsigned types
<S>    3     56  pointer-cast(tmp,from:void[],to:char[]) (safe)
                  cast to character type
<L>    4     56  initialized(tmp)     (safe)
                  assignedAt#56(rv:malloc)
--------------------------------------------------------------------------------
57    assert (buf != NULL);
--------------------------------------------------------------------------------
<S>    5     57  cast(buf,from:(char[] *),to:unsigned long[]) (safe)
                  casting a pointer to integer type unsigned long
<L>    6     57  initialized(buf)     (safe)
                  assignedAt#56
<S>    7     57  cast(caste((void[] *),0),from:(void[] *),to:unsigned long[]) (safe)
                  null-pointer cast
<S>    8     57  cast(0,from:int[],to:(void[] *)) (safe)
                  null-pointer cast
<S>    9     57  not-null(str(main))  (safe)
                  string literal
<S>   10     57  null-terminated(str(main)) (safe)
                  string literal
<S>   11     57  ptr-upper-bound(str(main),ntp(str(main)),op:pluspi,typ:char[]) (safe)
                  upperbound of constant string argument: main
<S>   12     57  initialized-range(str(main),len:ntp(str(main))) (safe)
                  constant string
<S>   13     57  not-null(str(id151.c)) (safe)
                  string literal
<S>   14     57  null-terminated(str(id151.c)) (safe)
                  string literal
<S>   15     57  ptr-upper-bound(str(id151.c),ntp(str(id151.c)),op:pluspi,typ:char[]) (safe)
                  upperbound of constant string argument: id151.c
<S>   16     57  initialized-range(str(id151.c),len:ntp(str(id151.c))) (safe)
                  constant string
<S>   17     57  not-null(str(buf != NULL)) (safe)
                  string literal
```

```
<S>   18     57  null-terminated(str(buf != NULL)) (safe)
                  string literal
<S>   19     57  ptr-upper-bound(str(buf != NULL),ntp(str(buf != NULL)),op:pluspi,typ:char[]) (safe)
                  upperbound of constant string argument: buf != NULL
<S>   20     57  initialized-range(str(buf != NULL),len:ntp(str(buf != NULL))) (safe)
                  constant string
<S>   21     57  valid-mem(str(buf != NULL)) (safe)
                  constant string is allocated by compiler
<S>   22     57  lower-bound(char[],str(buf != NULL)) (safe)
                  constant string is allocated by compiler
<S>   23     57  upper-bound(char[],str(buf != NULL)) (safe)
                  constant string is allocated by compiler
<S>   24     57  valid-mem(str(id151.c)) (safe)
                  constant string is allocated by compiler
<S>   25     57  lower-bound(char[],str(id151.c)) (safe)
                  constant string is allocated by compiler
<S>   26     57  upper-bound(char[],str(id151.c)) (safe)
                  constant string is allocated by compiler
<S>   27     57  valid-mem(str(main)) (safe)
                  constant string is allocated by compiler
<S>   28     57  lower-bound(char[],str(main)) (safe)
                  constant string is allocated by compiler
<S>   29     57  upper-bound(char[],str(main)) (safe)
                  constant string is allocated by compiler
--------------------------------------------------------------------------------
58
59    /*  BAD  */
60    buf[4105] = 'A';
--------------------------------------------------------------------------------
<L>   30     60  initialized(buf)     (safe)
                  assignedAt#56
<L>   31     60  not-null(buf)        (safe)
                  null has been explicitly excluded (either by assignment or by checking)
<L>   32     60  valid-mem(buf)       (safe)
                  all memory regions potentially pointed at are valid: addrof_heapregion_1
<S>   33     60  ptr-lower-bound(buf,4105,op:indexpi,typ:char[]) (safe)
                  add non-negative number:  value is 4105
<*>   34     60  ptr-upper-bound-deref(buf,4105,op:indexpi,typ:char[]) (violation)
                  increment is larger than or equal to the size of the memory region
                  returned by malloc: violates ((4105 * 1) < 10)
<S>   35     60  not-null((buf + 4105)) (safe)
                  arguments of pointer arithmetic are checked for null
<S>   36     60  valid-mem((buf + 4105)) (safe)
                  pointer arithmetic stays within memory region
<S>   37     60  lower-bound(char[],(buf + 4105)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy lowerbound
                  by inductive hypothesis
<S>   38     60  upper-bound(char[],(buf + 4105)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy upperbound
                  by inductive hypothesis
<S>   39     60  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
--------------------------------------------------------------------------------

Primary Proof Obligations
functions    stmt   local     api    post  global    open   total
----------------------------------------------------------------
main         33       6       0       0       0       0      39
----------------------------------------------------------------
total        33       6       0       0       0       0      39
```

```
percent   84.62   15.38    0.00    0.00    0.00    0.00


Proof Obligation Statistics for file id151
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Primary Proof Obligations
                             stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
cast                            4       0       0       0       0       0       4
initialized                     0       3       0       0       0       0       3
initialized-range               3       0       0       0       0       0       3
int-overflow                    1       0       0       0       0       0       1
int-underflow                   1       0       0       0       0       0       1
lower-bound                     4       0       0       0       0       0       4
not-null                        4       1       0       0       0       0       5
null-terminated                 3       0       0       0       0       0       3
pointer-cast                    1       0       0       0       0       0       1
ptr-lower-bound                 1       0       0       0       0       0       1
ptr-upper-bound                 3       0       0       0       0       0       3
ptr-upper-bound-deref           0       1       0       0       0       0       1
upper-bound                     4       0       0       0       0       0       4
valid-mem                       4       1       0       0       0       0       5
---------------------------------------------------------------------------------
total                          33       6       0       0       0       0      39
percent                     84.62   15.38    0.00    0.00    0.00    0.00
```

For id154.c (the safe version) the results are:

```
Function main
--------------------------------------------------------------------------------
52  int main(int argc, char *argv[])
--------------------------------------------------------------------------------
Api:
  parameters:
    int[] argc
    ((char[] *) *) argv

  postcondition guarantees:
   post-expr(eq,return-val,num-constant(0))

  library calls:
   assert:__assert_fail -- 1
   stdlib:malloc -- 1
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
53  {
54    char * buf;
55
56    buf = (char *) malloc(10 * sizeof(char));
--------------------------------------------------------------------------------
<S>    1     56  int-underflow(10,sizeof(char[]),op:mult,ikind:iulong) (safe)
                  underflow is well defined for unsigned types
<S>    2     56  int-overflow(10,sizeof(char[]),op:mult,ikind:iulong) (safe)
                  overflow is well defined for unsigned types
<S>    3     56  pointer-cast(tmp,from:void[],to:char[]) (safe)
                  cast to character type
<L>    4     56  initialized(tmp)     (safe)
```

```
                  assignedAt#56(rv:malloc)
--------------------------------------------------------------------------------
57    assert (buf != NULL);
--------------------------------------------------------------------------------
.......
--------------------------------------------------------------------------------
58
59    /*  OK  */
60    buf[9] = 'A';
--------------------------------------------------------------------------------
<L>   30     60  initialized(buf)     (safe)
                  assignedAt#56
<L>   31     60  not-null(buf)        (safe)
                  null has been explicitly excluded (either by assignment or by checking)
<L>   32     60  valid-mem(buf)       (safe)
                  all memory regions potentially pointed at are valid: addrof_heapregion_1
<S>   33     60  ptr-lower-bound(buf,9,op:indexpi,typ:char[]) (safe)
                  add non-negative number:  value is 9
<L>   34     60  ptr-upper-bound-deref(buf,9,op:indexpi,typ:char[]) (safe)
                  increment is less than the size of the memory region returned by
                  malloc: satisfies ((9 * 1) < 10)
<S>   35     60  not-null((buf + 9))  (safe)
                  arguments of pointer arithmetic are checked for null
<S>   36     60  valid-mem((buf + 9)) (safe)
                  pointer arithmetic stays within memory region
<S>   37     60  lower-bound(char[],(buf + 9)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy lowerbound
                  by inductive hypothesis
<S>   38     60  upper-bound(char[],(buf + 9)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy upperbound
                  by inductive hypothesis
<S>   39     60  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
--------------------------------------------------------------------------------
.......
```

2.4.2 Id167Q: Supporting Proof Obligations

The programs in this set illustrate the concept and use of *supporting proof obligations*. The safety of the array access in `function1` depends on the size of the array `buf` that is passed in as an argument. Since `function1` is not in a position to determine this size it must delegate the responsibility for the safety of the array access to the caller of the function. It does so by automatically generating the necessary conditions for safety on the argument value and advertising these as assumptions in its api. In this case there are two safety conditions: (1) the argument should not be null, and (2) the size of the argument should be at least 4106 (since it accesses index 4105).

The api assumptions generated by `function1` are converted by the calling function, `main`, into so-called supporting proof obligations that express the conditions applied to the actual arguments. In program id167.c the resulting proof obligation for the second api assumption, 4106 ¡ 10, evaluates to false, resulting in the report of a memory safety violation. In program id170.c both supporting proof obligations are shown valid.

Note that the violation in program 167.c is reported in `main` and not in `function1`, although the actual buffer overflow will happen in `function1`. In general a violation is placed at the highest applicable position in the call graph if there is a choice, because this will be the most likely position where a correction must be made. The program position of the actual buffer violation can be found by following the chain of assumptions and dependent ppo's/spos's that leads to the assumption that is being violated.

Results for id167.c:

```
Function function1
--------------------------------------------------------------------------------
50  void function1(char * buf)
--------------------------------------------------------------------------------
Api:
  parameters:
    (char[] *) buf

  api assumptions
   5  not-null(buf)
      --Dependent ppo's: [2]
   14  ptr-upper-bound-deref(buf,4105,op:pluspi,typ:char[])
      --Dependent ppo's: [5]
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
51  {
52    /*  BAD  */
53    buf[4105] = 'A';
--------------------------------------------------------------------------------
<S>    1     53  initialized(buf)     (safe)
                  buf is a function parameter
<A>    2     53  not-null(buf)        (safe)
                  condition not-null(buf) delegated to api
<L>    3     53  valid-mem(buf)       (safe)
                  all memory regions potentially pointed at are valid:
                  addr_in_(buf_1_)#init
<S>    4     53  ptr-lower-bound(buf,4105,op:indexpi,typ:char[]) (safe)
                  add non-negative number:  value is 4105
<A>    5     53  ptr-upper-bound-deref(buf,4105,op:indexpi,typ:char[]) (safe)
                  condition ptr-upperbound-deref(((buf +i 4105):char) delegated to api
<S>    6     53  not-null((buf + 4105)) (safe)
                  arguments of pointer arithmetic are checked for null
<S>    7     53  valid-mem((buf + 4105)) (safe)
                  pointer arithmetic stays within memory region
<S>    8     53  lower-bound(char[],(buf + 4105)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy lowerbound
                  by inductive hypothesis
<S>    9     53  upper-bound(char[],(buf + 4105)) (safe)
                  result of pointer arithmetic is guaranteed to satisfy upperbound
                  by inductive hypothesis
<S>   10     53  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
--------------------------------------------------------------------------------

Function main
--------------------------------------------------------------------------------
56  int main(int argc, char *argv[])
```

```
--------------------------------------------------------------------------------
Api:
  parameters:
    int[] argc
    ((char[] *) *) argv

  postcondition guarantees:
   post-expr(eq,return-val,num-constant(0))

  -- no library calls
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
57  {
58    char buf[10];
59
60
61    function1(buf);
--------------------------------------------------------------------------------
<S>    1     61  valid-mem(&(buf))    (safe)
                  address of a variable is a valid memory region
<S>    2     61  lower-bound(char[],&(buf)) (safe)
                  address of a variable
<S>    3     61  upper-bound(char[],&(buf)) (safe)
                  address of a variable
--------------------------------------------------------------------------------
Supporting Proof Obligations:
--------------------------------------------------------------------------------
57  {
58    char buf[10];
59
60
61    function1(buf);
--------------------------------------------------------------------------------
<S>    1    5   61   not-null(&(buf)) (safe)
                        address of variable buf
<*>    2   14   61   ptr-upper-bound-deref(&(buf),4105,op:pluspi,typ:char[]) (violation)
                        adding 4105 to the start of an array of length 10 violates
                        the upperbound

Primary Proof Obligations
functions       stmt   local     api    post  global    open   total
--------------------------------------------------------------------
function1          7       1       2       0       0       0      10
main               3       0       0       0       0       0       3
--------------------------------------------------------------------
total             10       1       2       0       0       0      13
percent        76.92    7.69   15.38    0.00    0.00    0.00

Supporting Proof Obligations
functions       stmt   local     api    post  global    open   total
--------------------------------------------------------------------
main               2       0       0       0       0       0       2
--------------------------------------------------------------------
total              2       0       0       0       0       0       2
percent       100.00    0.00    0.00    0.00    0.00    0.00


Proof Obligation Statistics for file id167
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Primary Proof Obligations
                              stmt   local     api    post  global    open   total
----------------------------------------------------------------------------------
cast                             1       0       0       0       0       0       1
initialized                      1       0       0       0       0       0       1
lower-bound                      2       0       0       0       0       0       2
not-null                         1       0       1       0       0       0       2
ptr-lower-bound                  1       0       0       0       0       0       1
ptr-upper-bound-deref            0       0       1       0       0       0       1
upper-bound                      2       0       0       0       0       0       2
valid-mem                        2       1       0       0       0       0       3
----------------------------------------------------------------------------------
total                           10       1       2       0       0       0      13
percent                      76.92    7.69   15.38    0.00    0.00    0.00

Supporting Proof Obligations
                              stmt   local     api    post  global    open   total
----------------------------------------------------------------------------------
not-null                         1       0       0       0       0       0       1
ptr-upper-bound-deref            1       0       0       0       0       0       1
----------------------------------------------------------------------------------
total                            2       0       0       0       0       0       2
percent                     100.00    0.00    0.00    0.00    0.00    0.00
```

3 Running Test Cases: Zitser

The test cases in the directory `ktadvance/tests/sard/zitser` are test cases number 1283 through 1310 from the NIST Software Assurance Reference Dataset (samate.nist.gov). These test cases were contributed by Misha Zitser, and are described in [4] All test cases have been pre-parsed and are ready for analysis.

3.1 Summary of Scripts

Scripts to analyze and report on the zitser test cases are located in the directory `ktadvance/advance/cmdline/zitser`. They are (in alphabetical order):

- `chc_analyze_zitser.py`: analyzes the given zitser test case.
  Example: `python chc_analyze_zitser.py id1283`
  Example: `python chc_analyze_zitser.py id1283 --verbose`
- `chc_analyze_zitser_set.py`: analyzes all zitser test cases (id1283-id1310); this may take 20-30 minutes.
  Example: `python chc_analyze_zitser_set.py`
- `chc_investigate_ppos.py`: outputs all open primary proof obligations per predicate, per file, per function for a given test case that has been analyzed; may optionally be restricted to a single predicate

Example: `python chc_investigate_ppos.py id1283`
Example: `python chc_investigate_ppos.py id1283 --predicate valid-mem`

- `chc_report_violations.py`: outputs a list of all (universal) violations found.
  Example: `python chc_report_violations.py id1283`
- `chc_report_zitser.py`: outputs a summary of the status of the proof obligations for the given zitser test case that has been analyzed.
  Example: `python chc_report_zitser.py id1283`
- `chc_report_zitser_file.py`: outputs a summary of the status for the proof obligations for the given zitser test case c file that has been analyzed (optionally with a listing of the code and the proof obligations associated with each line of code).
  Example: `python chc_report_zitser_file.py id1283 call_fb_realpath.c`
  Example: `python chc_report_zitser_file.py id1283 call_fb_realpath.c --showcode`
- `chc_report_zitser_function.py`: outputs the code and associated proof obligations with justification, if closed, and diagnostic or invariants, if open for a given function in a given zitser test case.
  Example: `python chc_report_zitser_function.py id1283 realpath-bad.c wu_realpath`
- `chc_show_zitser_file_table.py`: outputs the entries for a zitser test case file in a given (file-level) data dictionary
  Example: `python chc_show_zitser_file_table.py id1283 realpath-bad.c typ`
- `chc_show_zitser_function_table.py`: outputs the entries for a zitser test case function in a given (function-level) data dictionary
  Example: `python chc_show_zitser_function_table.py id1283 realpath-bad.c wu_realpath local_varinfo`
- `chc_zitser_dashboard.py`: outputs a summary of the results of all zitser test cases (after all test cases have been analyzed)
  Example: `python chc_zitser_dashboard.py`

3.2 Running the Tests

To analyze all 28 zitser test cases:

```
> export PYTHONPATH=$HOME/ktadvance
> cd ktadvance/advance/cmdline/zitser
> python chc_analyze_zitser_set.py
```

Since all zitser tests have been pre-parsed the analysis can be run on either Linux or MacOS; the results will always be the same, as the analysis has no dependencies on the compiler or resident standard header files.

To analyze a single zitser test case, say id1283:

```
> python chc_analyze_zitser.py id1283
```

or

```
> python chc_analyze_zitser.py id1283 --verbose
```

to have intermediate output printed to the console.

Once a test case is analyzed its results can be inspected. The command

```
> python chc_report_zitser.py id1283
```

outputs a summary of the status of all proof obligations:

```
Primary Proof Obligations
c files                   stmt   local     api    post  global    open   total
-------------------------------------------------------------------------------
call_fb_realpath            78      10       1       0       0      10      99
realpath-bad               685     440      59       0       0     196    1380
-------------------------------------------------------------------------------
total                      763     450      60       0       0     206    1479
percent                  51.59   30.43    4.06    0.00    0.00   13.93

Supporting Proof Obligations
c files                   stmt   local     api    post  global    open   total
-------------------------------------------------------------------------------
call_fb_realpath            10       0       0       0       0       1      11
realpath-bad                 7       1       2       0       0       1      11
-------------------------------------------------------------------------------
total                       17       1       2       0       0       2      22
percent                  77.27    4.55    9.09    0.00    0.00    9.09


Proof Obligation Statistics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Primary Proof Obligations
                               stmt   local     api    post  global    open   total
-----------------------------------------------------------------------------------
allocation-base                   0       2       0       0       0       0       2
cast                             47       6       0       0       0       4      57
common-base                       0       1       0       0       0       0       1
format-string                    33       0       0       0       0       0      33
global-mem                        5       0       0       0       0       2       7
index-lower-bound                 5       0       0       0       0       0       5
index-upper-bound                 5       0       0       0       0       0       5
initialized                      49     156       1       0       0       9     215
initialized-range                40       0       0       0       0      49      89
int-overflow                     12       2       0       0       0       5      19
int-underflow                    19       0       0       0       0       0      19
```

```
lower-bound                  82      66       0       0       0      17     165
no-overlap                    4       8       2       0       0       4      18
not-null                     58      56      46       0       0       1     161
null-terminated              40       1       0       0       0      47      88
pointer-cast                115       0       0       0       0       0     115
ptr-lower-bound               7       1       0       0       0       2      10
ptr-upper-bound              62       0       7       0       0      36     105
ptr-upper-bound-deref         3       0       4       0       0       3      10
signed-to-unsigned-cast      13       2       0       0       0       0      15
upper-bound                  82      66       0       0       0      17     165
valid-mem                    82      83       0       0       0      10     175
--------------------------------------------------------------------------------
total                       763     450      60       0       0     206    1479
percent                   51.59   30.43    4.06    0.00    0.00   13.93

Supporting Proof Obligations
                           stmt   local     api    post  global    open   total
--------------------------------------------------------------------------------
initialized                   0       0       0       0       0       2       2
no-overlap                    1       1       0       0       0       0       2
not-null                      3       0       1       0       0       0       4
ptr-upper-bound               7       0       1       0       0       0       8
ptr-upper-bound-deref         6       0       0       0       0       0       6
--------------------------------------------------------------------------------
total                        17       1       2       0       0       2      22
percent                   77.27    4.55    9.09    0.00    0.00    9.09
```

The command

```
> python chc_report_zitser_file.py id1283 realpath-bad.c
```

outputs the summary results for a single c file. Adding the option `--showcode` will produce a more detailed output that associates all proof obligations with the source code for all functions in the given c file. The command

```
> python chc_report_zitser_function.py id1283 realpath-bad.c fb_realpath
```

will output the source code for the function with the associated proof obligations and their status, as well as a summary of results for the function.

The script `chc_investigate_ppos.py` allows inspection of all open proof obligations or all proof obligations of a certain predicate, e.g.:

```
> python chc_investigate_ppos.py id1283 --predicate upper-bound
upper-bound
--------------------------------------------------------------------------------

  File: realpath-bad
    Function: fb_realpath
```

```
      138    284  upper-bound(char[],q) (open)
      172    292  upper-bound(char[],q) (open)
      181    293  upper-bound(char[const: ],caste((char[const: ] *),q)) (open)
      203    299  upper-bound(char[const: ],caste((char[const: ] *),q)) (open)
      223    305  upper-bound(char[],q) (open)
      231    313  upper-bound(char[],p) (open)
      254    315  upper-bound(char[],p) (open)
      265    316  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      320    325  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      431    359  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      464    365  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      567    390  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      586    391  upper-bound(char[],p) (open)
      609    397  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
      628    414  upper-bound(char[const: ],caste((char[const: ] *),p)) (open)
    Function: wu_realpath
      137    180  upper-bound(char[],ptr) (open)
      158    181  upper-bound(char[],ptr) (open)
```

....

To see a summary of the analysis results of all zitser test cases (after they have all been analyzed):

```
> python chc_zitser_dashboard.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Zitser test cases: id1283 - id1310
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Primary proof obligations

zitser testcase             stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
id1283                       763     450      60       0       0     206    1479
id1284                       640     347      51       0       0     164    1202
id1285                       871     294      35       0       0     271    1471
id1286                      1195     356      43       0       0     334    1928
id1287                       421     116      15       0       0     108     660
id1288                       378     118      16       0       0     111     623
id1289                       408     263      29       0       0     329    1029
id1290                       371     246      24       0       0     301     942
id1291                      1220    1099      58       0       0     399    2776
id1292                      1219    1102      59       0       0     398    2778
id1293                       807     805      47       0       0     262    1921
id1294                       926     848      51       0       0     272    2097
id1295                       302     164      32       0       0      99     597
id1296                       312     166      32       0       0      99     609
id1297                       355     521      16       0       0     219    1111
id1298                       725    1320      17       0       0     222    2284
id1299                       393     344      66       0       0     274    1077
id1300                       490     351      74       0       0     277    1192
id1301                       420     206      23       0       0     153     802
id1302                       255     168      16       0       0      98     537
```

```
id1303                   264    222     14      0      0    153    653
id1304                   291    252     14      0      0    156    713
id1305                   433    204     25      0      0    106    768
id1306                   478    213     34      0      0    112    837
id1307                   137     53      2      0      0    106    298
id1308                   138     53      2      0      0    107    300
id1309                   750    655     50      0      0    650   2105
id1310                   765    661     50      0      0    653   2129
----------------------------------------------------------------------------------
total                  15727  11597    955      0      0   6639  34918
percent                45.04  33.21   2.73   0.00   0.00  19.01

Secondary proof obligations

......

Proof obligation types

Primary proof obligations
                        stmt  local    api   post global   open  total
----------------------------------------------------------------------------------
allocation-base            0      2      2      0      0     10     14
cast                    1516    166      0      0      0    163   1845
common-base                0    193     16      0      0     13    222
common-base-type           0     91      0      0      0     10    101
format-string            593      0      0      0      0      2    595
global-mem                87      0      0      0      0     44    131
index-lower-bound         93     12      0      0      0      2    107
index-upper-bound         83     12      0      0      0     12    107
initialized              968   4581     48      0      0    824   6421
initialized-range        769      8      0      0      0    509   1286
int-overflow             218     26      0      0      0    145    389
int-underflow            316     21      8      0      0     44    389
lower-bound             1616   1178      0      0      0    690   3484
no-overlap                77     49     10      0      0     38    174
non-negative             179      0      0      0      0      0    179
not-null                1208   1709    785      0      0    600   4302
not-zero                   2      0      0      0      0      0      2
null-terminated          792     62      0      0      0    451   1305
pointer-cast            1839      0      0      0      0     40   1879
ptr-lower-bound          728     28     12      0      0    129    897
ptr-upper-bound          938     24     25      0      0    508   1495
ptr-upper-bound-deref    122      9     49      0      0    696    876
signed-to-unsigned-cast  165     36      0      0      0     74    275
unsigned-to-signed-cast   67     41      0      0      0    157    265
upper-bound             1574    910      0      0      0    932   3416
valid-mem               1598   2439      0      0      0    546   4583
width-overflow           179      0      0      0      0      0    179
----------------------------------------------------------------------------------
total                  15727  11597    955      0      0   6639  34918
percent                45.04  33.21   2.73   0.00   0.00  19.01

Secondary proof obligations
                        stmt  local    api   post global   open  total
----------------------------------------------------------------------------------
allocation-base            0      0      0      0      0     26     26
common-base                0      4      0      0      0      0      4
initialized                6      0      0      0      0     54     60
int-underflow              0      4      0      0      0      2      6
no-overlap                 4      2      0      0      0      0      6
```

```
not-null                        46      52      30       0       0      83     211
ptr-lower-bound                  0       4       0       0       0       2       6
ptr-upper-bound                 16       7       3       0       0      18      44
ptr-upper-bound-deref           10      10      10       0       0      19      49
------------------------------------------------------------------------------------
total                           82      83      43       0       0     204     412
percent                      19.90   20.15   10.44    0.00    0.00   49.51
```

4 Running Test Cases: Juliet

The `ktadvance/tests/sard/juliet_v1.3` directory contains a subset of the test cases from the Juliet Test Suite, developed by the NSA Center for Assured Software, and updated by NIST. The entire test suite is available from the NIST SARD repository, at `samate.nist.gov`. These tests are also part of the SATE V Ockham competition organized by NIST [1].

4.1 Organization

The test cases are organized in a similar way as the original Test Suite, except that the file hierarchy is extended by one level to create separate directories for each functional variant, that solely contain the flow variants for that functional variant. Furthermore, C++ variants have been removed.

Each functional variant directory contains the following files:

- `[name]_src.tar.gz`: the original c source files, including a reduced main file, `main_linux.c` that contains a main function that calls the functions in the flow variants.
- `semantics_linux.tar.gz`: the result of preprocessing and parsing the source files (on a linux platform) into KT Advance Analyzer input format The files in this tar file provide all necessary information for the analysis; the analysis can be run on any platforms; all platform dependencies are included in the files.
- `scorekey.json`: a specification of the proof obligations and their locations that are relevant to the vulnerability being tested by the test case, for both the good and the bad versions of the functions.

The tests currently provided are shown in Table 1. More tests will be added in the near future.

4.2 Summary of Scripts

Scripts to analyze, report on, and score the juliet test cases are located in the `advance/cmdline/juliet` directory. They are (in alphabetical order):

CWE121/s01/CWE129_large
CWE121/s01/CWE129_rand
CWE121/s01/CWE131_loop
CWE121/s01/char_type_overrun_memcpy
CWE121/s01/char_type_overrun_memmove
CWE121/s02/CWE193_char_alloca_loop
CWE121/s02/CWE193_char_alloca_ncpy
CWE121/s02/CWE193_char_declare_loop
CWE121/s03/CWE805_char_declare_loop
CWE121/s03/CWE805_char_declare_memcpy
CWE121/s03/CWE805_char_declare_memmove
CWE121/s03/CWE805_char_declare_ncpy
CWE122/s01/char_type_overrun_memcpy
CWE122/s01/char_type_overrun_memmove
CWE122/s05/CWE131_loop
CWE122/s05/CWE131_memcpy
CWE122/s06/CWE131_memmove
CWE122/s06/CWE135
CWE122/s06/c_CWE129_connect_socket
CWE122/s06/c_CWE129_fgets

Table 1: Current set of Juliet Tests provided in `tests/sard/juliet_v1.3`

- `chc_analyze_juliettest.py`: analyzes the given juliet test case
  Example: `python chc_analyze_juliettest.py CWE121/s01/CWE129_large`
- `chc_analyze_juliettest_sets.py`: analyzes all juliet test cases in the `juliet_v1.3` directory. This may take more than an hour.
  Example: `python chc_analyze_juliettest_sets.py`
- `chc_juliet_dashboard.py`: outputs a summary of the proof obligation status of the proof obligations related to the vulnerability being tested (as specified in the `scorekey.json` file). Can only be run after all test cases have been analyzed and scored.
  Example: `python chc_juliet_dashboard.py`
- `chc_list_juliettests.py`: outputs the list of juliet tests in the `tests/sard/juliet_v1.3` directory with date and times of their last analysis and scoring.
- `chc_parse_juliettest.py`: preprocesses and parses the given juliet test. This script is only used when adding new test sets; all existing tests have already been parsed.
  Example: `python chc_parse_juliettest.py CWE121/s01/CWE129_large`
- `chc_prepare_juliettest.py`: renames the flow variants into x[nn].c to reduce filename and function-name lengths in reports. This script is only used when adding new test sets.
  Example: `python chc_prepare_juliettest.py CWE121/s01/CWE129_large CWE121_Stack_Based_Buffer_Overflow__CWE129_large_`
- `chc_report_juliettest.py`: outputs a summary of the status of the proof obligations for all flow variants in the test.
  Example: `python chc_report_juliettest.py CWE121/s01/CWE129_large`
- `chc_report_juliettest_file.py`: outputs a summary of the status of the proof obligations for a given flow variant in the test.
  Example: `python chc_report_juliettest_file.py CWE121/s01/CWE129_large x01.c`
  Example:
  `python chc_report_juliettest_file.py CWE121/s01/CWE129_large x01.c --showcode`
- `chc_report_juliettest_function.py`: outputs the source code of the given function with all associated proof obligations and a summary of their status.
  Example:
  `python chc_report_juliettest_function.py CWE121/s01/CWE129_large x01.c goodB2G`
- `chc_report_juliettest_sets.py`: outputs a summary of the proof obligation status (for all proof obligations) for all juliet tests currently provided.
  Example: `python chc_report_juliettest_sets.py`
- `chc_report_juliettest_vulnerability.py`: outputs the source code for the given test with the proof obligations associated with the vulnerability tested (obtained from the scorekey)
  Example: `python chc_report_vulnerability.py CWE121/s01/CWE129_large 01`
- `chc_score_juliettest.py`: compares the analysis results with the desired results as specified in the file scorekey.json and records the results in the file summaryresults.json and outputs a comparison report.

Example: `python chc_score_juliettest.py CWE121/s01/CWE129_large`

- `chc_score_juliettest_sets.py`: runs the scoring on all juliet tests present. Can be run only after all tests have been analyzed
  Example: `python chc_score_juliettest_sets.py`

4.3 Running the Tests

Set the PYTHONPATH environment variable (or adapt for a different location of the ktadvance directory):

```
> export PYTHONPATH=$HOME/ktadvance
```

All tests provided in the `tests/sard/juliet_v1.3` directory have been pre-parsed and thus are ready for analysis. To analyze a Juliet test case:

```
> cd ktadvance/advance/cmdline/juliet
> python chc_analyze_juliettest.py CWE121/s01/CWE129_large
```

or, if multiple processors are available:

```
> python chc_analyze_juliettest.py CWE121/s01/CWE129_large --maxprocesses 16
```

which allows the invariant generation and proof obligation checking to be performed in parallel for different c files. The analysis will produce a semantics directory in the test directory that contains all analysis results. These analysis results can be inspected by various reporting scripts. To see a summary of the results for this test case:

```
> python chc_report_juliettest.py CWE121/s01/CWE129_large
Primary Proof Obligations
c files            stmt   local     api    post  global    open   total
-----------------------------------------------------------------------
io                  298      63      14       0       0      47     422
main_linux            7       3       0       0       0       1      11
std_thread           70      84      14       0       0      28     196
x01                  24      34       0       0       0       2      60
x02                  42      54       0       0       0       3      99
x03                  42      54       0       0       0       3      99
x04                  52      54       0       0       0       3     109
x05                  52      54       0       0       0       3     109
x06                  52      54       0       0       0       3     109
x07                  52      54       0       0       0       3     109
x08                  57      73       0       0       0       4     134
x09                 103      71       0       0       0       6     180
x10                 103      71       0       0       0       6     180
x11                  42      64       0       0       0       3     109
```

x12	55	74	0	0	0	5	134
x13	103	71	0	0	0	6	180
x14	103	71	0	0	0	6	180
x15	135	71	0	0	0	6	212
x16	24	34	0	0	0	2	60
x17	52	60	0	0	0	4	116
x18	24	34	0	0	0	2	60
x21	87	50	2	0	0	4	143
x22a	0	4	0	0	0	0	4
x22b	87	46	2	0	0	4	139
x31	24	40	0	0	0	2	66
x32	46	71	0	0	0	29	146
x34	46	44	0	0	0	8	98
x41	53	37	2	0	0	3	95
x42	49	44	0	0	0	5	98
x44	53	55	2	0	0	3	113
x45	49	44	0	0	0	5	98
x51a	0	3	0	0	0	0	3
x51b	53	34	2	0	0	3	92
x52a	0	3	0	0	0	0	3
x52b	3	0	0	0	0	0	3
x52c	53	34	2	0	0	3	92
x53a	0	3	0	0	0	0	3
x53b	3	0	0	0	0	0	3
x53c	3	0	0	0	0	0	3
x53d	53	34	2	0	0	3	92
x54a	0	3	0	0	0	0	3
x54b	3	0	0	0	0	0	3
x54c	3	0	0	0	0	0	3
x54d	3	0	0	0	0	0	3
x54e	53	34	2	0	0	3	92
x61a	46	44	0	0	0	5	95
x61b	3	0	0	0	0	0	3
x63a	9	0	0	0	0	0	9
x63b	49	50	5	0	0	6	110
x64a	12	0	0	0	0	0	12
x64b	49	53	3	0	0	11	116
x65a	0	21	0	0	0	0	21
x65b	53	34	2	0	0	3	92
x66a	15	3	0	0	0	0	18
x66b	64	44	6	0	0	8	122
x67a	0	3	0	0	0	3	6
x67b	49	41	0	0	0	5	95
x68a	0	3	0	0	0	0	3
x68b	49	41	0	0	0	5	95
total	2614	2122	60	0	0	267	5063
percent	51.63	41.91	1.19	0.00	0.00	5.27	

Supporting Proof Obligations

c files	stmt	local	api	post	global	open	total
x21	0	2	0	0	0	0	2
x22a	0	2	0	0	0	0	2
x41	0	2	0	0	0	0	2
x44	0	2	0	0	0	0	2
x51a	0	2	0	0	0	0	2
x52a	0	2	0	0	0	0	2
x52b	0	0	2	0	0	0	2
x53a	0	2	0	0	0	0	2

```
x53b                  0       0       2       0       0       0       2
x53c                  0       0       2       0       0       0       2
x54a                  0       2       0       0       0       0       2
x54b                  0       0       2       0       0       0       2
x54c                  0       0       2       0       0       0       2
x54d                  0       0       2       0       0       0       2
x63a                  1       1       0       0       0       0       2
x64a                  3       0       0       0       0       0       3
x65a                  0       2       0       0       0       0       2
x66a                  6       0       0       0       0       0       6
-----------------------------------------------------------------------
total                10      19      12       0       0       0      41
percent           24.39   46.34   29.27    0.00    0.00    0.00


Proof Obligation Statistics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Primary Proof Obligations
                              stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
allocation-base                  0       1       3       0       0       0       4
cast                            35       0       0       0       0       4      39
format-string                   17       0       0       0       0       0      17
global-mem                       1       0       0       0       0       0       1
index-lower-bound              295     248       0       0       0       0     543
index-upper-bound              150     348      20       0       0      25     543
initialized                    247    1225       3       0       0     161    1636
initialized-range              180       0       0       0       0       3     183
int-overflow                   156     130       0       0       0       0     286
int-underflow                  286       0       0       0       0       0     286
lower-bound                    216      40       0       0       0      13     269
no-overlap                       1       0       0       0       0       0       1
not-null                       197      22      31       0       0       6     256
not-zero                         1       0       0       0       0       0       1
null-terminated                180       1       0       0       0       2     183
pointer-cast                    36       0       0       0       0       6      42
ptr-lower-bound                  3       9       0       0       0       2      14
ptr-upper-bound                180       0       0       0       0       4     184
ptr-upper-bound-deref            0       0       3       0       0      11      14
signed-to-unsigned-cast          3       0       0       0       0       5       8
unsigned-to-signed-cast          2       0       0       0       0       2       4
upper-bound                    214      37       0       0       0      15     266
valid-mem                      214      61       0       0       0       8     283
---------------------------------------------------------------------------------
total                         2614    2122      60       0       0     267    5063
percent                      51.63   41.91    1.19    0.00    0.00    5.27

Supporting Proof Obligations
                              stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
index-upper-bound                0      19      12       0       0       0      31
not-null                         7       0       0       0       0       0       7
ptr-upper-bound-deref            3       0       0       0       0       0       3
---------------------------------------------------------------------------------
total                           10      19      12       0       0       0      41
percent                      24.39   46.34   29.27    0.00    0.00    0.00
```

The output of this report shows the number of and status of all proof obligations in all files included in the test (flow variants are renamed to x[nn].c). This report does not distinguish between proof obligations proven valid and proof obligations that are shown to be violated. The status is divided (currently) into four categories:

- `stmt`: the proof obligation was proven valid or a violation based on the information in the statement (or instruction) and its associated declarations. No invariants were necessary for the proof.
- `local`: the proof obligation was proven valid or a violation based on local invariants, that is, invariants generated within the function.
- `api`: the proof obligation was delegated to the api, possibly with the use of local invariants. This is done only if all variables in the proof obligations are shown to be in a linear equality relation with function parameters.
- `open`: the proof obligation could not be proven valid or a violation.

Thus juliet test CWE121/s01/CWE129_large has a total of 5063 proof obligations, 5.27% of which could not be proven valid or a violation.

To inspect the status of the proof obligations associated with the vulnerability being tested for, for a given file:

```
> python chc_report_juliettest_vulnerability CWE121/s01/CWE129_large 01
********** Function: CWE121_Stack_Based_Buffer_Overflow__CWE129_large_01_bad ***

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_01_bad
--------------------------------------------------------------------------------
22  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_01_bad()
Primary Proof Obligations:
--------------------------------------------------------------------------------
23  {
24      int data;
25      /* Initialize data */
26      data = -1;
27      /* POTENTIAL FLAW: Use an invalid index */
28      data = 10;
29      {
30          int i;
31          int buffer[10] = { 0 };
32          /* POTENTIAL FLAW: Attempt to write to an index of the array that
                   is above the upper bound
33          * This code does check to see if the array index is negative */
34          if (data >= 0)
35          {
36              buffer[data] = 1;
--------------------------------------------------------------------------------
<*>   12     36  index-upper-bound(data,bound:10) (violation)
                  index value, 10, is greater than or equal to length, 10
--------------------------------------------------------------------------------


********** Function: goodB2G **************************************************
```

```
Function goodB2G
--------------------------------------------------------------------------------
85  static void goodB2G()
Primary Proof Obligations:
--------------------------------------------------------------------------------
86  {
87      int data;
88      /* Initialize data */
89      data = -1;
90      /* POTENTIAL FLAW: Use an invalid index */
91      data = 10;
92      {
93          int i;
94          int buffer[10] = { 0 };
95          /* FIX: Properly validate the array index and prevent a buffer overflow */
96          if (data >= 0 && data < (10))
97          {
98              buffer[data] = 1;
--------------------------------------------------------------------------------
<X>   13     98  index-upper-bound(data,bound:10) (dead-code)
                  unreachable
--------------------------------------------------------------------------------


********** Function: goodG2B **************************************************

Function goodG2B
--------------------------------------------------------------------------------
55  static void goodG2B()
Primary Proof Obligations:
--------------------------------------------------------------------------------
56  {
57      int data;
58      /* Initialize data */
59      data = -1;
60      /* FIX: Use a value greater than 0, but less than 10 to avoid attempting to
61       * access an index of the array in the sink that is out-of-bounds */
62      data = 7;
63      {
64          int i;
65          int buffer[10] = { 0 };
66          /* POTENTIAL FLAW: Attempt to write to an index of the array that
                    is above the upper bound
67          * This code does check to see if the array index is negative */
68          if (data >= 0)
69          {
70              buffer[data] = 1;
--------------------------------------------------------------------------------
<L>   12     70  index-upper-bound(data,bound:10) (safe)
                  index value, 7, is less than length, 10
--------------------------------------------------------------------------------
```

which shows the code up to the location of the vulnerability together with the proof obligation(s) that are violated due to the vulnerability. In cases where the proof obligation is delegated to the caller, the callers are shown as well:

```
> python chc_report_juliettest_vulnerability CWE121/s01/CWE129_large 22
```

```
********** Function: CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badSink ***

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badSink
--------------------------------------------------------------------------------
25  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badSink(int data)
Primary Proof Obligations:
--------------------------------------------------------------------------------
26  {
27      if(CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badGlobal)
28      {
29          {
30              int i;
31              int buffer[10] = { 0 };
32              /* POTENTIAL FLAW: Attempt to write to an index of the array that
                      is above the upper bound
33              * This code does check to see if the array index is negative */
34              if (data >= 0)
35              {
36                  buffer[data] = 1;
--------------------------------------------------------------------------------
<A>   13     36  index-upper-bound(data,bound:10) (safe)
                  condition data < 10 delegated to api
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Callers
--------------------------------------------------------------------------------

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_bad
--------------------------------------------------------------------------------
27  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_bad()
Supporting Proof Obligations:
--------------------------------------------------------------------------------
28  {
29      int data;
30      /* Initialize data */
31      data = -1;
32      /* POTENTIAL FLAW: Use an invalid index */
33      data = 10;
34      CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badGlobal = 1; /* true */
35      CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_badSink(data);
--------------------------------------------------------------------------------
<*>    1   77   35   index-upper-bound(data,bound:10) (violation)
                        index value, 10, is greater than or equal to length, 10


********** Function: CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G1Sink ***

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G1Sink
--------------------------------------------------------------------------------
61  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G1Sink(int data)
Primary Proof Obligations:
--------------------------------------------------------------------------------
62  {
63      if(CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G1Global)
64      {
65          /* INCIDENTAL: CWE 561 Dead Code, the code below will never run */
66          printLine("Benign, fixed string");
67      }
68      else
69      {
```

```
70          {
71              int i;
72              int buffer[10] = { 0 };
73              /* FIX: Properly validate the array index and prevent a buffer overflow */
74              if (data >= 0 && data < (10))
75              {
76                  buffer[data] = 1;
--------------------------------------------------------------------------------
<L>   21     76  index-upper-bound(data,bound:10) (safe)
                  upper bound of index value range, 9, is less than length, 10
--------------------------------------------------------------------------------


********** Function: CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G2Sink ***

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G2Sink
--------------------------------------------------------------------------------
92  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G2Sink(int data)
Primary Proof Obligations:
--------------------------------------------------------------------------------
93  {
94      if(CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodB2G2Global)
95      {
96          {
97              int i;
98              int buffer[10] = { 0 };
99              /* FIX: Properly validate the array index and prevent a buffer overflow */
100              if (data >= 0 && data < (10))
101              {
102                  buffer[data] = 1;
--------------------------------------------------------------------------------
<L>   14    102  index-upper-bound(data,bound:10) (safe)
                  upper bound of index value range, 9, is less than length, 10
--------------------------------------------------------------------------------


********** Function: CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BSink ***

Function CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BSink
--------------------------------------------------------------------------------
118  void CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BSink(int data)
Primary Proof Obligations:
--------------------------------------------------------------------------------
119  {
120      if(CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BGlobal)
121      {
122          {
123              int i;
124              int buffer[10] = { 0 };
125              /* POTENTIAL FLAW: Attempt to write to an index of the array
                        that is above the upper bound
126              * This code does check to see if the array index is negative */
127              if (data >= 0)
128              {
129                  buffer[data] = 1;
--------------------------------------------------------------------------------
<A>   13    129  index-upper-bound(data,bound:10) (safe)
                  condition data < 10 delegated to api
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
Callers
--------------------------------------------------------------------------------

Function goodG2B
--------------------------------------------------------------------------------
78  static void goodG2B()
Supporting Proof Obligations:
--------------------------------------------------------------------------------
79  {
80      int data;
81      /* Initialize data */
82      data = -1;
83      /* FIX: Use a value greater than 0, but less than 10 to avoid attempting to
84       * access an index of the array in the sink that is out-of-bounds */
85      data = 7;
86      CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BGlobal = 1; /* true */
87      CWE121_Stack_Based_Buffer_Overflow__CWE129_large_22_goodG2BSink(data);
--------------------------------------------------------------------------------
<L>    1   11   87   index-upper-bound(data,bound:10) (safe)
                        index value, 7, is less than length, 10
```

To see all proof obligations associated with the code:

```
> python chc_report_juliettest_file.py CWE121/s01/CWE129_large x01.c --showcode
```

or for a single funciton:

```
> python chc_report_juliettest_function.py CWE121/s01/CWE129_large x01.c goodG2B
```

To check the analysis results against the scorekey:

```
> python chc_score_juliettest.py CWE121/s01/CWE129_large
```

```
x01.c

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_01_bad
     36   12: index-upper-bound        <*>  index value, 10, is greater than or equal to length, 10

  goodB2G
     98   13: index-upper-bound        <X>  unreachable

  goodG2B
     70   12: index-upper-bound        <L>  index value, 7, is less than length, 10

x02.c

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_02_bad
     41   12: index-upper-bound        <*>  index value, 10, is greater than or equal to length, 10

  goodB2G1
     84   13: index-upper-bound        <X>  unreachable
```

```
  goodB2G2
    118   13: index-upper-bound        <X>  unreachable

  goodG2B1
    159   12: index-upper-bound        <L>  index value, 7, is less than length, 10

......

x67b.c

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_67b_badSink
     37   13: index-upper-bound       ?

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_67b_goodB2GSink
     90   14: index-upper-bound        <L>  upper bound of index value range, 9, is less than length, 10

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_67b_goodG2BSink
     66   13: index-upper-bound       ?

x68b.c

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_68b_badSink
     36   13: index-upper-bound       ?

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_68b_goodB2GSink
     89   14: index-upper-bound        <L>  upper bound of index value range, 9, is less than length, 10

  CWE121_Stack_Based_Buffer_Overflow__CWE129_large_68b_goodG2BSink
     65   13: index-upper-bound       ?

Summary

test              violations                     safe-controls
         V    S    D    U    O               S    L    D    X    U    O
--------------------------------------------------------------------------------
01       1    0    0    0    0       |       0    1    0    1    0    0
02       1    0    0    0    0       |       0    1    0    2    0    0
03       1    0    0    0    0       |       0    1    0    2    0    0
04       1    0    0    0    0       |       0    2    0    2    0    0
05       1    0    0    0    0       |       0    2    0    2    0    0
06       1    0    0    0    0       |       0    2    0    2    0    0
07       1    0    0    0    0       |       0    2    0    2    0    0
08       1    0    0    0    0       |       0    3    0    1    0    0
09       0    0    0    1    0       |       0    4    0    0    0    0
10       0    0    0    1    0       |       0    4    0    0    0    0
11       1    0    0    0    0       |       0    2    0    2    0    0
12       0    0    0    1    0       |       0    3    0    2    0    0
13       0    0    0    1    0       |       0    4    0    0    0    0
14       0    0    0    1    0       |       0    4    0    0    0    0
15       0    0    0    1    0       |       0    4    0    0    0    0
16       1    0    0    0    0       |       0    1    0    1    0    0
17       0    0    0    1    0       |       0    2    0    0    0    0
18       1    0    0    0    0       |       0    1    0    1    0    0
21       1    0    0    0    0       |       0    3    0    0    0    0
22       1    0    0    0    0       |       0    3    0    0    0    0
31       1    0    0    0    0       |       0    1    0    1    0    0
32       0    0    0    1    0       |       0    1    0    0    1    0
34       0    0    0    1    0       |       0    1    0    0    1    0
41       1    0    0    0    0       |       0    2    0    0    0    0
```

```
42       0    0    0    1    0       |       0    1    0    0    1    0
44       1    0    0    0    0       |       0    2    0    0    0    0
45       0    0    0    1    0       |       0    1    0    0    1    0
51       1    0    0    0    0       |       0    2    0    0    0    0
52       1    0    0    0    0       |       0    2    0    0    0    0
53       1    0    0    0    0       |       0    2    0    0    0    0
54       1    0    0    0    0       |       0    2    0    0    0    0
61       0    0    0    1    0       |       0    1    0    0    1    0
63       1    0    0    0    0       |       0    2    0    0    0    0
64       0    0    0    1    0       |       0    1    0    0    1    0
65       1    0    0    0    0       |       0    2    0    0    0    0
66       0    0    0    1    0       |       0    1    0    0    1    0
67       0    0    0    1    0       |       0    1    0    0    1    0
68       0    0    0    1    0       |       0    1    0    0    1    0
--------------------------------------------------------------------------------
total   22    0    0   16    0       |       0   75    0   21    9    0
```

For each test this report shows the proof obligations associated with the vulnerability and their status. The summary shows how many of the violations were identified and how many of the safe controls were indeed proven safe. The column headers indicate:

- **violations**
  - **V:** violation proven
  - **S:** proven safe (should not happen)
  - **D:** delegated
  - **U:** unknown, open proof obligation
  - **O:** other
- **safe controls**
  - **S:** proven statement safe (no invariants necessary)
  - **L:** proven safe (using local invariants)
  - **D:** delegated
  - **U:** unknown, open proof obligation
  - **O:** other

To analyze and score all juliet tests present and see a summary of all results:

```
> python chc_analyze_juliettest_sets.py
........
> python chc_score_juliettest_sets.py
........
> python chc_juliet_dashboard.py
```

```
test                                              violations                       safe-controls
```

	V	S	D	U	O	S	L	D	X	U	O
CWE121/s01/char_type_overrun_memcpy	54	0	0	0	0	96	0	0	0	0	0
CWE121/s01/char_type_overrun_memmove	54	0	0	0	0	96	0	0	0	0	0
CWE121/s01/CWE129_large	22	0	0	16	0	0	75	0	21	9	0
CWE121/s01/CWE129_rand	0	0	10	28	0	0	96	0	0	9	0
CWE121/s01/CWE131_loop	25	0	0	9	0	0	38	0	0	9	0
CWE121/s02/CWE193_char_alloca_loop	18	0	0	16	0	0	26	0	0	21	0
CWE121/s02/CWE193_char_alloca_ncpy	18	0	0	16	0	0	28	0	0	19	0
CWE121/s02/CWE193_char_declare_loop	18	0	0	16	0	0	27	0	0	20	0
CWE121/s03/CWE805_char_declare_memcpy	36	0	0	30	0	0	52	0	0	40	0
CWE121/s03/CWE805_char_declare_memmove	36	0	0	30	0	0	54	0	0	38	0
CWE121/s03/CWE805_char_declare_ncpy	36	0	0	30	0	0	52	0	0	40	0
CWE121/s03/CWE805_char_declare_loop	36	0	0	26	0	0	49	0	0	38	0
CWE122/s01/char_type_overrun_memcpy	54	0	0	0	0	99	0	0	0	0	0
CWE122/s01/char_type_overrun_memmove	54	0	0	0	0	99	0	0	0	0	0
CWE122/s05/CWE131_loop	25	0	0	13	0	0	38	0	0	15	0
CWE122/s05/CWE131_memcpy	25	0	0	13	0	0	38	0	0	15	0
CWE122/s06/CWE131_memmove	25	0	0	13	0	0	38	0	0	15	0
CWE122/s06/CWE135	0	0	0	39	0	0	0	0	0	106	0
CWE122/s06/c_CWE129_connect_socket	0	0	0	38	0	0	90	0	0	17	0
CWE122/s06/c_CWE129_fgets	0	0	0	38	0	0	89	0	0	17	0
total	536	0	10	371	0	390	790	0	21	428	0

	violation	safe-control	total
ppos	917	1629	2546
identified	536	1201	1737
perc	58.5	73.7	68.2

5 Analyzing Your Own Files

The ktadvance/advance/cmdline/sfapp contains scripts that allow parsing and analyzing applications that consist of a single file and can be compiled directly by `gcc` without the help of a Makefile.

5.1 Summary of Scripts

Scripts to parse, analyze, and report on C applications that consist of a single file are located in the `advance/cmdline/sfapp` directory. They are (in alphabetical order):

- `chc_analyze_file.py`: analyzes the given c file.
  Example: `python chc_analyze_file.py $HOME/cfiletest test.c`
  Example: `python chc_analyze_file.py $HOME/cfiletest test.c --deletesemantics`

```
typedef struct _mystruct {
  int field1;
  char* field2;
} mystruct;


void initialize(mystruct *s) {
  s->field1 = 0;
  s->field2 = "Hello";
}

int main(int argc, char **argv) {

  int buffer[10];
  mystruct s;

  for (int i=0; i <= 10 ; i++) {
    buffer[i] = 0;
  }

  buffer[10] = 10;

  initialize(&s);
}
```

Figure 1: Example c file: `test.c`

- `chc_parse_file.py`: preprocesses and parses the given c file.
  Example: `python chc_parse_file.py $HOME/cfiletest/test.c`
  Example: `python chc_parse_file.py $HOME/cfiletest/test.c --savesemantics`
- `chc_report_file.py`: outputs a summary of the status of the proof obligations for the given c file.
  Examples:
  - `python chc_report_file.py $HOME/cfiletest test.c`
  - `python chc_report_file.py $HOME/cfiletest test.c --showcode`
  - `python chc_report_file.py $HOME/cfiletest test.c --showcode --violations`
  - `python chc_report_file.py $HOME/cfiletest test.c --showcode --open`

5.2 Running the Scripts

Set the PYTHONPATH environment variable (or adapt for a different location of the ktadvance directory) and go to the single-file cmdline directory:

```
> export PYTHONPATH=$HOME/ktadvance
> cd ktadvance/advance/cmdline/sfapp
```

Assume we would like to analyze the c file, `test.c`, shown in Figure 1 in the directory `cfiletest`. We first need to parse the file:

```
> python chc_parse_file.py $HOME/cfiletest/test.c
```

If we expect not to change the code, we can save the semantics, so it does not need to be parsed again:

```
> python chc_parse_file.py $HOME/cfiletest/test.c --savesemantics
```

This will create a semantics_linux.tar.gz (or semantics_mac.tar.gz) file in the `cfiletest` directory that gets unpacked by the analysis.

To analyze the program:

```
> python chc_analyze_file.py $HOME/cfiletest test.c
```

or, if we have a saved semantics file, it is usually a good idea to delete the current semantics directory:

```
> python chc_analyze_file.py $HOME/cfiletest test.c --deletesemantics
```

The analysis results will be saved in the directory `cfiletest/semantics/ktadvance` and can be inspected with some reporting scripts:

```
> python chc_report_file.py $HOME/cfiletest test.c

Primary Proof Obligations
functions          stmt   local     api    post  global    open   total
-----------------------------------------------------------------------
initialize            6       6       2       0       0       0      14
main                  6       5       0       0       0       1      12
```

```
---------------------------------------------------------------------
total              12      11       2       0       0       1      26
percent         46.15   42.31    7.69    0.00    0.00    3.85

Supporting Proof Obligations
functions        stmt   local     api    post  global    open   total
---------------------------------------------------------------------
main                1       0       0       0       0       0       1
---------------------------------------------------------------------
total               1       0       0       0       0       0       1
percent        100.00    0.00    0.00    0.00    0.00    0.00


Proof Obligation Statistics for file test
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Primary Proof Obligations
                              stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
index-lower-bound                1       1       0       0       0       0       2
index-upper-bound                1       0       0       0       0       1       2
initialized                      2       3       0       0       0       0       5
int-overflow                     0       1       0       0       0       0       1
int-underflow                    1       0       0       0       0       0       1
lower-bound                      2       2       0       0       0       0       4
not-null                         0       0       2       0       0       0       2
null-terminated                  1       0       0       0       0       0       1
pointer-cast                     1       0       0       0       0       0       1
upper-bound                      1       2       0       0       0       0       3
valid-mem                        2       2       0       0       0       0       4
---------------------------------------------------------------------------------
total                           12      11       2       0       0       1      26
percent                      46.15   42.31    7.69    0.00    0.00    3.85

Supporting Proof Obligations
                              stmt   local     api    post  global    open   total
---------------------------------------------------------------------------------
not-null                         1       0       0       0       0       0       1
---------------------------------------------------------------------------------
total                            1       0       0       0       0       0       1
percent                     100.00    0.00    0.00    0.00    0.00    0.00
```

which shows the numbers and status of proof obligations for the functions and kinds of proof obligations. To also show the code:

```
> python chc_report_file.py $HOME/cfiletest test.c --showcode
```

```
Function initialize
--------------------------------------------------------------------------------
8  void initialize(mystruct *s) {
--------------------------------------------------------------------------------
Api:
  parameters:
    (mystruct[] *) s

  api assumptions
   12  not-null(s)
      --Dependent ppo's: [2,7]

  -- no postcondition requests

  -- no postcondition guarantees

  -- no library calls
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
9    s->field1 = 0;
--------------------------------------------------------------------------------
<S>    1      9  initialized(s)       (safe)
                  s is a function parameter
<A>    2      9  not-null(s)          (safe)
                  condition not-null(s) delegated to api
<L>    3      9  valid-mem(s)         (safe)
                  all memory regions potentially pointed at are valid: addr_in_(s_1_)#init
<L>    4      9  lower-bound(struct _mystruct(1),s) (safe)
                  initial value of parameter s satisfies lower bound by inductive hypothesis
<L>    5      9  upper-bound(struct _mystruct(1),s) (safe)
                  initial value of parameter s satisfies upper bound by inductive hypothesis
--------------------------------------------------------------------------------
10    s->field2 = "Hello";
--------------------------------------------------------------------------------
<S>    6     10  initialized(s)       (safe)
                  s is a function parameter
<A>    7     10  not-null(s)          (safe)
                  condition not-null(s) delegated to api
<L>    8     10  valid-mem(s)         (safe)
                  all memory regions potentially pointed at are valid: addr_in_(s_1_)#init
<L>    9     10  lower-bound(struct _mystruct(1),s) (safe)
                  initial value of parameter s satisfies lower bound by inductive hypothesis
<L>   10     10  upper-bound(struct _mystruct(1),s) (safe)
                  initial value of parameter s satisfies upper bound by inductive hypothesis
<S>   11     10  pointer-cast(str(Hello),from:char[],to:char[]) (safe)
                  source and target type are the same
<S>   12     10  lower-bound(char[],caste((char[] *),str(Hello))) (safe)
                  constant string is allocated by compiler
<S>   13     10  valid-mem(caste((char[] *),str(Hello))) (safe)
                  constant string is allocated by compiler
<S>   14     10  null-terminated(caste((char[] *),str(Hello))) (safe)
                  string literal
--------------------------------------------------------------------------------

Function main
--------------------------------------------------------------------------------
13  int main(int argc, char **argv) {
--------------------------------------------------------------------------------
Api:
```

```
  parameters:
    int[] argc
    ((char[] *) *) argv

  -- no assumptions

  -- no postcondition requests

  postcondition guarantees:
   post-expr(eq,return-val,num-constant(0))

  -- no library calls
--------------------------------------------------------------------------------
Primary Proof Obligations:
--------------------------------------------------------------------------------
14
15    int buffer[10];
16    mystruct s;
17
18    for (int i=0; i <= 10 ; i++) {
--------------------------------------------------------------------------------
<L>    1     18  initialized(i)       (safe)
                  assignedAt#18
<L>    5     18  initialized(i)       (safe)
                  assignedAt#18
<S>    6     18  int-underflow(i,1,op:plusa,ikind:iint) (safe)
                  add non-negative number:  value is 1
<L>    7     18  int-overflow(i,1,op:plusa,ikind:iint) (safe)
                  result of addition is less than MAX: satisfies((10 + 1) < 32767)
--------------------------------------------------------------------------------
19      buffer[i] = 0;
--------------------------------------------------------------------------------
<L>    2     19  index-lower-bound(i) (safe)
                  lower bound of index value range is non-negative: 0

<?>    3     19  index-upper-bound(i,bound:10) (open)
                  --
                  argc:nv[lv:argc:int[]  test.c:13] : sx:(argc_4_)#init:nv[aux-argc:nv[lv:argc:int[]  test.c:13]_init]
                  argv:nv[lv:argv:((char[] *) *)  test.c:13] : sx:(argv_2_)#init:nv[aux-argv:nv[lv:argv:((char[] *) *)  test.
                  argv:nv[lv:argv:((char[] *) *)  test.c:13] : bv:(argv_2_)#init:nv[aux-argv:nv[lv:argv:((char[] *) *)  test.
                  argv:sv[lv:argv:((char[] *) *)  test.c:13] : regions:NULL(1)_s:2,addr_in_(argv_2_)#init_s:1
                  check_(ppo:2:1)_(ppo:3:1)_(ppo:4:1):nv[check(ppo:2,1;ppo:3,1;ppo:4,1)] : iv:[0;10]
                  check_(ppo:2:1)_(ppo:3:1)_(ppo:4:1):sv[check(ppo:2,1;ppo:3,1;ppo:4,1)] : syms:assignedAt#18
                  check_(ppo:3:2):nv[check(ppo:3,2)] : iv:10
                       i:nv[lv:i:int[]  test.c:18] : iv:[0;10]
                       i:sv[lv:i:int[]  test.c:18] : syms:assignedAt#18
<L>    4     19  initialized(i)       (safe)
                  assignedAt#18
--------------------------------------------------------------------------------
20    }
21
22    buffer[10] = 10;
--------------------------------------------------------------------------------
<S>    8     22  index-lower-bound(10) (safe)
                  index value 10 is non-negative
<*>    9     22  index-upper-bound(10,bound:10) (violation)
                  index value 10 violates upper bound 10
--------------------------------------------------------------------------------
23
24    initialize(&s);
```

```
--------------------------------------------------------------------------------
<S>   10     24  valid-mem(&(s))      (safe)
                  address of a variable is a valid memory region
<S>   11     24  lower-bound(struct _mystruct(1),&(s)) (safe)
                  address of a variable
<S>   12     24  upper-bound(struct _mystruct(1),&(s)) (safe)
                  address of a variable
--------------------------------------------------------------------------------
Supporting Proof Obligations:
--------------------------------------------------------------------------------
14
15    int buffer[10];
16    mystruct s;
17
18    for (int i=0; i <= 10 ; i++) {
19      buffer[i] = 0;
20    }
21
22    buffer[10] = 10;
23
24    initialize(&s);
--------------------------------------------------------------------------------
<S>    1   12   24   not-null(&(s)) (safe)
                        address of variable s
.......
```

which shows an open proof obligation for line 19 and a violation for line 22. It is also possible to only show proof obligations that are open or violated:

```
> python chc_report_file.py $HOME/cfiletest test.c --showcode --open --violations
```

or to get just a listing of all open proof obligations and violations:

```
> python chc_investigate_ppos.py $HOME/cfiletest test.c

* ==============================================================================
* KT Advance C Analyzer: Summary of open, delegated and violated proof obligations for test.c
* date       : 2017-11-28 14:17:29
* ==============================================================================
Open proof obligations:
================================================================================


index-upper-bound
--------------------------------------------------------------------------------

  File: test
    Function: main
         3     19  index-upper-bound(i,bound:10) (open)


Delegated proof obligations:
================================================================================
```

```
not-null
--------------------------------------------------------------------------------

  File: test
    Function: initialize
         2      9  not-null(s)          (safe)
         7     10  not-null(s)          (safe)


Violations:
================================================================================


index-upper-bound
--------------------------------------------------------------------------------

  File: test
    Function: main
         9     22  index-upper-bound(10,bound:10) (violation)
```

6 Analyzing Your Own Applications

References

[1] Paul E. Black and Athos Rebeiro. SATE V Ockham Sound Analysis Criteria. Technical Report NISTIR 8113, NIST, 2017.

[2] Kendra Kratkiewicz and Richard Lippmann. Using a diagnostic corpus of C programs to evaluate buffer overflow detection by static analysis tools. In Paul E. Black and Elizabeth Fong, editors, *Proceedings of Defining the State of the Art in Software Security Tools Workshop*, number 500-264 in NIST Special Publication, pages 102–111. NIST, September 2005.

[3] George C. Necula, Scott McPeak, Shree Prakash Rahul, and Westley Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. In R. Nigel Horspool, editor, *CC*, volume 2304 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2002.

[4] Misha Zitser, Richard Lippmann, and Tim Leek. Testing static analysis tools using exploitable buffer overflows from open source code. In Richard N. Taylor and Matthew B. Dwyer, editors, *SIGSOFT FSE*, pages 97–106. ACM, 2004.