≡ KLEE

master branch

The 2nd International KLEE Workshop on Symbolic Execution is coming!
Join us from 14-15 September 2020 in London.

#SYMBOLIC ENVIRONMENT

Using Symbolic Environment

As mentioned in the overview of KLEE's basic command-line options, KLEE provides several options as part of its symbolic environment. Their usage, however, is often not easily understood by new users. This tutorial provides basic usage examples for `-sym-arg` and `-sym-files`, which are perhaps the most essential among the options.

`-sym-arg` Usage

We note that `-sym-arg <N>` provides a command-line argument of length N to the program under test. Its variant `-sym-args <MIN> <MAX> <N>` provides at least MIN arguments and at most MAX symbolic arguments, each with maximum length N. To demonstrate `-sym-arg`, let us first consider the following program `password.c` that checks its command-line argument for a match with a hard-coded password.

```
#include <stdio.h>
```

```
  if (buf[0] == 'h' && buf[1] == 'e' &&
      buf[2] == 'l' && buf[3] == 'l' &&
      buf[4] == 'o')
    return 1;
  return 0;
}

int main(int argc, char **argv) {
  if (argc < 2)
     return 1;

  if (check_password(argv[1])) {
    printf("Password found!\n");
    return 0;
  }

  return 1;
}
```

To enable symbolic environment, KLEE has to be given the `-posix-runtime` option. We run KLEE given the bitcode of `password.c` as input and using `-sym-arg` option as follows.

```
$ clang -c -g -emit-llvm password.c
$ klee -posix-runtime password.bc -sym-arg 5
KLEE: NOTE: Using model: /home/klee/klee/build/Release+Debug+Asser
KLEE: output directory is "/home/klee/klee-out-1"
KLEE: Using STP solver backend
KLEE: WARNING: undefined reference to function: __errno_location
KLEE: WARNING: undefined reference to function: printf
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not
```

```
Password found!

KLEE: done: total instructions = 817
KLEE: done: completed paths = 6
KLEE: done: generated tests = 6
```

As can be seen, due to the command-line argument being symbolic, KLEE executed six paths, with one of the path having the command-line argument match the password.

`-sym-files` Usage

The option `-sym-files <NUM> <N>` creates NUM symbolic files, where the first file is named 'A', the second 'B', and so on, each with size N. Its sibling options `-sym-stdin` and `-sym-stdout` make the standard input and output symbolic, respectively.

Let us now consider a password checker, still called `password.c` that reads a string from a file specified by the user and checks if it matches a hard-coded password. If the file name is not specified, or if there is an error when opening the file, it reads the string from the standard input.

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int check_password(int fd) {
  char buf[5];
  if (read(fd, buf, 5) != -1) {
    if (buf[0] == 'h' && buf[1] == 'e' &&
      return 1;
  }
  return 0;
}

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

  if (argc >= 2) {
    if ((fd = open(argv[1], O_RDONLY)) != -1) {
      if (check_password(fd)) {
        printf("Password found in %s\n", argv[1]);
        close(fd);
        return 0;
      }
      close(fd);
      return 1;
    }
  }

  if (check_password(0)) {
    printf("Password found in standard input\n");
    return 0;
  }

  return 1;
}
```

We now run the program using KLEE. For the program not to get stuck trying to read data, we need to provide some input. In our first run, we provide a symbolic standard input using `-sym-stdin`

≡ KLEE master branch

```
$ clang -c -g -emit-llvm password.c
$ klee -posix-runtime password.bc -sym-stdin 10
KLEE: NOTE: Using model: /home/klee/klee/build/Release+Debug+Asse
KLEE: output directory is "/home/klee/klee-out-0"
KLEE: Using STP solver backend
KLEE: WARNING: undefined reference to function: __errno_location
KLEE: WARNING: undefined reference to function: printf
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not
KLEE: WARNING ONCE: calling external: syscall(4, 35171856, 3526822
KLEE: WARNING ONCE: calling external: printf(35146128) at [no debu
Password found in standard input

KLEE: done: total instructions = 1283
KLEE: done: completed paths = 6
KLEE: done: generated tests = 6
```

We have now discovered the password using KLEE.

Our program can also read the password from a disk file, but we want to read a file with symbolic content so that KLEE executes the path where the password check is successful. The `-sym-files` option provides several such files named 'A', 'B', 'C', and so on. By specifying the option `-sym-files 1 10` below, we ask KLEE to provide one symbolic file of size 10 bytes, and that file is named 'A' by KLEE. We therefore provide this file name as an argument to our program.

```
$ klee -posix-runtime password.bc A -sym-files 1 10
KLEE: NOTE: Using model: /home/klee/klee/build/Release+Debug+Asser
KLEE: output directory is "/home/klee/klee-out-0"
KLEE: Using STP solver backend
KLEE: WARNING: undefined reference to function: __errno_location
KLEE: WARNING: undefined reference to function: printf
```

master branch

```
KLEE: WARNING ONCE: calling external: printf(37924528, 3785
Password found in A

KLEE: done: total instructions = 2868
KLEE: done: completed paths = 6
KLEE: done: generated tests = 6
```

The password was successfully read from the symbolic file A in one of the execution paths.

master branch

Resources

Mailing List

Doxygen

GitHub

TravisCI

Documentation for KLEE master branch