KLEE

07/12/2016

Kalev Alpernas

Slides and demos based on:

http://klee.github.io/tutorials/

https://feliam.wordpress.com/2010/10/07/the-symbolic-maze/

Running KLEE Using Docker

- We run a Docker image of KLEE
  - YMMV if compiled from source
- Creating a new persistent container:

```
docker run -ti --name=<name> --ulimit='stack=-1:-1' klee/klee
```

- Listing available containers:

```
docker ps -a
```

- Restarting a container:

```
docker start -ai cont_name
```

- Removing a container:

```
docker rm cont_name
```

Some More Details on the KLEE Docker Image

- Inside a container, you can install whatever you need the regular way
  - e.g. `sudo apt-get install emacs`
  - User/pass: `klee`/`klee`

- git is already installed, so you can `git clone` the course repository directly in the container

- You can also use the `--volume` option of `klee run` to mount a directory

  `--volume /home/softprod/tmp:/home/klee/tmp`

Running KLEE

- Let’s check the `get_sign` function

```
int get_sign(int x)
{
  if (x == 0)
    return 0;
  if (x < 0)
    return -1;
  else
    return 1;
}
```

Running KLEE

- Let's check the `get_sign` function
- We need to run it on a *symbolic* input

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;



  return get_sign(a);
}
```

Running KLEE

- Let's check the `get_sign` function
- We need to run it on a *symbolic* input
- We use `klee_make_symbolic` To mark `a` as a symbolic variable

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Running KLEE

- First, need to produce LLVM IR (bitcode)

```
clang -I /home/klee/klee_src/include -emit-llvm -c -g get_sign.c
```

Running KLEE

- First, need to produce LLVM IR (bitcode)

```
clang -I /home/klee/klee_src/include -emit-llvm -c -g get_sign.c
```

- Next, we run klee

```
klee get_sign.bc
```

Running KLEE

- First, need to produce LLVM IR (bitcode)

```
clang -I /home/klee/klee_src/include -emit-llvm -c -g get_sign.c
```

- Next, we run klee

```
klee get_sign.bc
```

- How many different paths do we expect?

Running KLEE

- We have 3 execution paths:
  - `a == 0`
  - `a < 0`
  - `a > 0`

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Running KLEE

- We have 3 execution paths:
  - `a == 0`
  - `a < 0`
  - `a > 0`

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Running KLEE

- We have 3 execution paths:
  - `a == 0`
  - `a < 0`
  - `a > 0`

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Running KLEE

- We have 3 execution paths:
  - `a == 0`
  - `a < 0`
  - `a > 0`

- Indeed KLEE produces 3 testcases

```
ls klee-last | grep ktest
```

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Running KLEE

- We have 3 execution paths:
  - `a == 0`
  - `a < 0`
  - `a > 0`
- Indeed KLEE produces 3 testcases

  ```
  ls klee-last | grep ktest
  ```
- We can run the testcases

```
int get_sign(int x)
{
  if (x == 0) return 0;
  if (x < 0) return -1;
  else return 1;
}
int main()
{
  int a;
  klee_make_symbolic(
    &a,
    sizeof(a),
    "a");
  return get_sign(a);
}
```

Using KLEE to Find Errors

- KLEE finds some errors on its own
  - Array out of bounds, division by 0, etc.
  - Error kinds we needed to specifically request from CBMC to look for

- We can also add annotation to the C program

  `klee_assert(<boolean expression>)`

- If KLEE detects a violation of the assertion, it reports that testcase as a bug

Maze Example