2017年11月7日 星期二

Seda 安装配置笔记

1、安装环境

- VMWare Workstation 12.0 虚拟机
- 2核4线程，8G内存
- Ubuntu-14.04.5-desktop-amd64

2、安装依赖软件

- 在命令行下输入：

  ```
  ~$ sudo apt-get install cmake libxml2-dev libncurses5-dev python-pygraphviz libarmadillo-dev realpath time libboost-all-dev libx11-dev vim git g++ build-essential coccinelle python-pip
  ~$ sudo apt-get update
  ```

- 下载dlib，移动到/usr/local/include，并进行编译和配置

  ```
  ~$ wget http://dlib.net/files/dlib-18.18.zip
  ~$ unzip dlib-18.18.zip
  ~$ sudo cp -r dlib-18.18/dlib/ /usr/local/include
  ~$ cd /usr/local/include/dlib/all
  /usr/local/include/dlib/all$ sudo g++ -O3 -shared -fPIC -o /usr/local/lib/libdlib.so source.cpp
  ```

- 下载并安装所需的python包

  ```
  ~$ sudo pip install termcolor unidiff pygraphviz
  ```

- 下载llvm与clang，并切换到指定版本

  ```
  ~$ git clone https://github.com/llvm-mirror/llvm.git llvm
  ~$ git -C llvm checkout release_36
  ~$ git clone https://github.com/llvm-mirror/clang.git llvm/tools/clang
  ~$ git -C llvm/tools/clang checkout release_36
  ```

```
~$ cd llvm
~/llvm$ mkdir build && cd build
~llvm/build$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DBUILD_SHARED_LIBS=ON -DLLVM_LIBDIR_SUFFIX=64 -DLLVM_ENABLE_RTTI=ON ..
```

- （Optional）修改~/llvm/build/CMakeCache.txt文件中的一行，可以在bitcode文件中有更好的命名。

```
- CMAKE_CXX_FLAGS_RELWITHDEBINFO:STRING=-O2 -g -DNDEBUG
+ CMAKE_CXX_FLAGS_RELWITHDEBINFO:STRING=-O2 -g -D_DEBUG
```

- 安装llvm与clang

```
~llvm/build$ sudo make install
```

- 添加共享库路径。在/etc/ld.so.conf.d文件夹下新建一个文件

```
~$ cd /etc/ld.so.conf.d
/etc/ld.so.conf.d$ sudo vim libLLVM.conf
```

- 在文件里写入之前安装的llvm/build/lib64路径（/home/lt/llvm是我机器上llvm的路径，要根据自己的实际情况修改）

```
/home/lt/llvm/build/lib64
```

- 保存后，执行ldconfig

```
~$ sudo ldconfig
```

3、安装配置Seda

- 编译安装Seda：

```
~$ cd seda/src
~/seda/src$ mkdir build && cd build
~/seda/src/build$ cmake ..
~/seda/src/build$ make install
```

- 初始配置seda（只配置一次）

```
~$ cd seda
~/seda$ git config --global core.abbrev 12
~/seda$ cp doc/config.template config
```

- 每次新打开一个bash，要先执行下面的命令

```
~$ cd seda
~/seda$ source envsetup.sh
```

- 在seda根目录下，执行seda指令来获取可用的命令

```
~$ cd seda
~/seda$ seda
```

4、测试seda能否正常工作

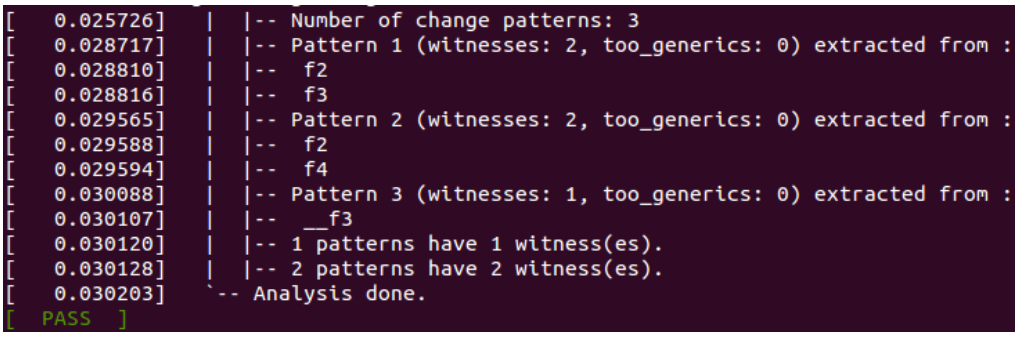
- 新打开一个bash，执行source envsetup.sh

```
~$ cd seda
~/seda$ source envsetup.sh
```

- 执行test 20-example测试

```
~/seda$ seda test 20-example
```

- 正确的话，输出的最后几行如下



- 测试semantic patch能否正常生成

```
~/seda$ cd tests/20-example/graphs/
~/seda/tests/20-example/graphs$ seda coccigen .
```

- 正确的话，会输出如下的结果

```
//# pattern-1, witnesses: 2
@r0@
identifier i, fn;
@@
struct ops i = {
    .f2 = fn,
};
@@
identifier r0.fn;
identifier x;
@@
- int fn(struct dentry * x)
+ int fn(struct dentry * x, struct inode * i)
 {
     <...
- x->d_sb
+ i->i_sb
     ...>
 }
//# pattern-2, witnesses: 2
@r1@
identifier i, fn;
@@
struct ops i = {
    .f2 = fn,
};
@@
identifier d;
identifier i;
identifier r1.fn;
@@
 int fn(struct dentry * d, struct inode * i)
 {
     <...
- d_inode(d)
+ i
     ...>
 }
```