Handbook

Tracing skb_clone()

1. Experiment setup

- VMware Workstation Pro
- Ubuntu 18.04.5

2. BCC installation

Install with source code

1. Changing source

Edit sources.list文件

`sudo vim /etc/apt/sources.list`

Content:

```
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic main restricted universe
multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-updates main restricted universe
multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-updates main restricted
universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-backports main restricted
universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-backports main restricted
universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-security main restricted universe
multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-security main restricted
universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-proposed main restricted universe
multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-proposed main restricted
universe multiverse
```

Update the software

```
sudo apt-get update
sudo apt-get upgrade
```

2. Install dependencies

The tool chain includes:

- LLVM 3.7.1 or higher，support compling
- BPF(default: on) Clang，same with LLVM

- cmake(>=3.1), gcc(>=4.7), flex, bison
- LuaJIT, 如果需要Lua

  Run the following:

```
sudo apt-get -y install bison build-essential cmake flex git libedit-dev \
  libllvm6.0 llvm-6.0-dev libclang-6.0-dev python zlib1g-dev libelf-dev libfl-
dev python3-distutils
```

```
forward@ubuntu:~$ sudo apt-get -y install bison build-essential cmake flex git libedit-dev \
>   libllvm6.0 llvm-6.0-dev libclang-6.0-dev python zlib1g-dev libelf-dev libfl-dev python3-distutils
Reading package lists... Done
Building dependency tree
Reading state information... Done
libllvm6.0 is already the newest version (1:6.0-1ubuntu2).
The following additional packages will be installed:
```

Finish installation:

```
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (12.4ubuntu1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.5) ...
Processing triggers for systemd (237-3ubuntu10.53) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for install-info (6.5.0.dfsg.1-2) ...
```

3. Install and compile BCC

由于bcc的默认安装目录为/usr/share，执行以下指令：

```
cd /usr/share
sudo git clone https://github.com/iovisor/bcc.git
```

```
forward@ubuntu:/usr/share$ sudo git clone https://github.com/iovisor/bcc.git
Cloning into 'bcc'...
remote: Enumerating objects: 25362, done.
remote: Counting objects: 100% (5757/5757), done.
remote: Compressing objects: 100% (733/733), done.
remote: Total 25362 (delta 5270), reused 5091 (delta 5022), pack-reused 19605
Receiving objects: 100% (25362/25362), 16.33 MiB | 5.53 MiB/s, done.
Resolving deltas: 100% (16798/16798), done.
```

Note:

BCC的master分支不再支持LLVM7或更早版本，需要使用**tag v0.24.0**，执行以下指令：

```
cd bcc
sudo git checkout v0.24.0
```

```
forward@ubuntu:/usr/share/bcc$ sudo git checkout v0.24.0
Note: checking out 'v0.24.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 8f40d6f5 update debian changelog for release v0.24.0
```

Run cmake, automatically generates makefile:

```
sudo mkdir build; cd build
sudo cmake ..
```

```
forward@ubuntu:/usr/share/bcc/build$ sudo cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
Cloning into '/usr/share/bcc/src/cc/libbpf'...
Submodule path 'src/cc/libbpf': checked out '22411acc4b2c846868fd570b2d9f3b016d2af2cb'
-- Latest recognized Git tag is v0.24.0
-- Git HEAD is 8f40d6f57a8d94e7aee74ce358572d34d31b4ed4
-- Revision is 0.24.0-8f40d6f5
-- Performing Test HAVE_NO_PIE_FLAG
-- Performing Test HAVE_NO_PIE_FLAG - Success
-- Performing Test HAVE_REALLOCARRAY_SUPPORT
-- Performing Test HAVE_REALLOCARRAY_SUPPORT - Success
-- Found LLVM: /usr/lib/llvm-6.0/include 6.0.0 (Use LLVM_ROOT envronment variable for another version of LLVM)
-- Found BISON: /usr/bin/bison (found version "3.0.4")
-- Found FLEX: /usr/bin/flex (found version "2.6.4")
-- Found LibElf: /usr/lib/x86_64-linux-gnu/libelf.so
-- Performing Test ELF_GETSHDRSTRNDX
-- Performing Test ELF_GETSHDRSTRNDX - Success
-- Could NOT find LibDebuginfod (missing: LIBDEBUGINFOD_LIBRARIES LIBDEBUGINFOD_INCLUDE_DIRS)
-- Using static-libstdc++
-- Could NOT find LuaJIT (missing: LUAJIT_LIBRARIES LUAJIT_INCLUDE_DIR)
CMake Warning at tests/python/CMakeLists.txt:10 (message):
  Recommended test program 'netperf' not found


CMake Warning at tests/python/CMakeLists.txt:16 (message):
  Recommended test program 'iperf' or 'iperf3' not found


-- Configuring done
-- Generating done
-- Build files have been written to: /usr/share/bcc/build
```

Run make:

`sudo make`

```
forward@ubuntu:/usr/share/bcc/build$ sudo make
Scanning dependencies of target bcc-loader-static
[  0%] Building CXX object src/cc/CMakeFiles/bcc-loader-static.dir/bcc_syms.cc.o
[  0%] Building C object src/cc/CMakeFiles/bcc-loader-static.dir/bcc_elf.c.o
[  1%] Building C object src/cc/CMakeFiles/bcc-loader-static.dir/bcc_perf_map.c.o
[  1%] Building C object src/cc/CMakeFiles/bcc-loader-static.dir/bcc_proc.c.o
[  1%] Building CXX object src/cc/CMakeFiles/bcc-loader-static.dir/common.cc.o
[  2%] Linking CXX static library libbcc-loader-static.a
[  2%] Built target bcc-loader-static
Scanning dependencies of target bpf-static
```

Finishing：

```
[ 99%] Building CXX object tests/cc/CMakeFiles/test_libbcc.dir/test_usdt_args.cc.o
[ 99%] Building CXX object tests/cc/CMakeFiles/test_libbcc.dir/test_usdt_probes.cc.o
[100%] Building CXX object tests/cc/CMakeFiles/test_libbcc.dir/utils.cc.o
[100%] Building CXX object tests/cc/CMakeFiles/test_libbcc.dir/test_parse_tracepoint.cc.o
[100%] Linking CXX executable test_libbcc
[100%] Built target test_libbcc
```

Run make install:

`sudo make install`

```
forward@ubuntu:/usr/share/bcc/build$ sudo make install
[sudo] password for forward:
[  2%] Built target bcc-loader-static
[ 11%] Built target bpf-static
[ 13%] Built target clang_frontend
[ 14%] Built target api-static
```

Finishing:

```
-- Installing: /usr/share/bcc/tools/old/syncsnoop
-- Installing: /usr/share/bcc/tools/old/tcpaccept
-- Installing: /usr/share/bcc/tools/old/tcpconnect
-- Installing: /usr/share/bcc/tools/old/wakeuptime
forward@ubuntu:/usr/share/bcc/build$
```

Binding python3, execute:

```
sudo cmake -DPYTHON_CMD=python3 ..
pushd src/python/
sudo make
sudo make install
popd
```

```
forward@ubuntu:/usr/share/bcc/build$ sudo cmake -DPYTHON_CMD=python3 ..
-- Latest recognized Git tag is v0.24.0
-- Git HEAD is 8f40d6f57a8d94e7aee74ce358572d34d31b4ed4
-- Revision is 0.24.0-8f40d6f5
-- Found LLVM: /usr/lib/llvm-6.0/include 6.0.0 (Use LLVM_ROOT envronment variable for another version of LLVM)
-- Could NOT find LibDebuginfod (missing: LIBDEBUGINFOD_LIBRARIES LIBDEBUGINFOD_INCLUDE_DIRS)
-- Using static-libstdc++
-- Could NOT find LuaJIT (missing: LUAJIT_LIBRARIES LUAJIT_INCLUDE_DIR)
CMake Warning at tests/python/CMakeLists.txt:10 (message):
  Recommended test program 'netperf' not found


CMake Warning at tests/python/CMakeLists.txt:16 (message):
  Recommended test program 'iperf' or 'iperf3' not found


-- Configuring done
-- Generating done
-- Build files have been written to: /usr/share/bcc/build
forward@ubuntu:/usr/share/bcc/build$ pushd src/python/
/usr/share/bcc/build/src/python /usr/share/bcc/build
forward@ubuntu:/usr/share/bcc/build/src/python$ sudo make
[100%] Built target bcc_py_python3
forward@ubuntu:/usr/share/bcc/build/src/python$ sudo make install
[100%] Built target bcc_py_python3
Install the project...
-- Install configuration: "Release"
running install
running build
running build_py
running install_lib
copying build/lib/bcc/table.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/__init__.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/libbcc.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/syscall.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/utils.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/containers.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/disassembler.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/perf.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/tcp.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/usdt.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/version.py -> /usr/lib/python3/dist-packages/bcc
byte-compiling /usr/lib/python3/dist-packages/bcc/table.py to table.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/libbcc.py to libbcc.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/syscall.py to syscall.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/utils.py to utils.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/containers.py to containers.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/disassembler.py to disassembler.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/perf.py to perf.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/tcp.py to tcp.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/usdt.py to usdt.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/version.py to version.cpython-36.pyc
running install_egg_info
Removing /usr/lib/python3/dist-packages/bcc-0.24.0_8f40d6f5.egg-info
Writing /usr/lib/python3/dist-packages/bcc-0.24.0_8f40d6f5.egg-info
forward@ubuntu:/usr/share/bcc/build/src/python$ popd
/usr/share/bcc/build
```

BCC installation is finished。

注意修改python软链接指向python3，即可直接使用/usr/share/bcc/tools中的现有工具。

```
cd /usr/bin
sudo rm python
sudo ln -s python3 python
```

例如执行funccount工具, verify the installation

```
forward@ubuntu:/usr/share/bcc/tools$ sudo ./funccount "vfs_*"
[sudo] password for forward:
Tracing 67 functions for "b'vfs_*'"... Hit Ctrl-C to end.
^C
FUNC                                    COUNT
b'vfs_statfs'                               1
b'vfs_readlink'                             3
b'vfs_statx_fd'                            18
b'vfs_statx'                               30
b'vfs_getattr'                             46
b'vfs_getattr_nosec'                       46
b'vfs_open'                               113
b'vfs_write'                              556
b'vfs_read'                               767
b'vfs_writev'                            1155
Detaching...
```

3. skb_clone experiment

1. Requirement

   Request any file from the Internet using curl，using bcc to measure how many skb_clone() has been called and output the process id。

2. Thoughts

   In order to realize the requirement, we have to go through the following steps。

   2.1. 使用kprobe对内核进行动态tracing

   ```
   from bcc import BPF

   BPF(text = """
   #C语言代码段
   """)

   #对bpf的处理代码
   ```

   根据BPF程序结构，bcc有如下两种写法：

   - C函数以`kprobe__`开头，其余部分被视为要监测的内核函数名，如`kprobe__skb_clone()`
   - C函数为自定义函数，利用`b.attach_kprobe(event="func_1", fn_name="func_2")`建立

   kprobe，内核系统出现func_1操作时执行func_2函数，即自定义函数。

   2.2. 获取调用skb_clone()函数的进程id

   提供如下两种思路：

   - 从`trace_fields()`中获取诸如pid和时间戳的信息

- 使用`BPF_PERF_OUTPUT()`接口

2.3. 记录skb_clone()函数的调用次数

需要用变量count记录次数，实现count的更新累加，并能返回给用户态。

利用BPF map对象，实现用户态和内核态的数据交互，这里使用BPF_HASH。

3. 实验过程

根据两种获取信息的思路，提供两种编写方式：

3.1. `trace_fields()`

核心代码如下：

```
#include <uapi/linux/ptrace.h>

//创建一个BPF map对象
BPF_HASH(count);

//出现skb_clone操作时执行以下函数，记录次数
int do_trace(struct pt_regs *ctx){
    u64 cur_count = 1, key = 0;
    u64 *cur_ptr;
    cur_ptr = count.lookup(&key);
    if (cur_ptr == NULL){

        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

    }else{

        cur_count = *cur_ptr + 1;
        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

        }
    return 0;
}
```

其中，

- `count.lookup(&key)`：查看hash中key对应的值，如果存在返回值的指针
- `count.update(&key, &cur_count)`：将value与key关联，并覆盖之前的value，即更新次数
- `bpf_trace_printk()`：打印的信息会由`trace_fields()`中的`msg`字段取得

当出现skb_clone()操作时，就会执行do_trace()函数，用变量cur_count记录次数。

每出现一次skb_clone()操作时，`cur_ptr = count.lookup(&key)`就会查看hash中key对应的值，返回值的指针，`cur_count = *cur_ptr + 1`即可更新cur_count数值，并通过hash：`count.update(&key, &cur_count)`实现次数的更新覆盖。

最后，输出相关信息即可。

3.2. `BPF_PERF_OUTPUT()`

该方法的逻辑思路一致，区别在于不再从`trace_fields()`中获取信息。

需要自己定义C struct，将data从kernel传到user space。

```
struct data_t{
    u32 pid;
    u64 ts;
    u64 cur_count;
    char comm[TASK_COMM_LEN];
};
```

我们选择记录pid、时间戳、次数以及进程名，通过相应bpf函数获取。

获取并计算得出相应数据后，通过`events.perf_submit(ctx, &data, sizeof(data))`提交，使得user space能够通过per ring buffer读取。

输出信息部分代码如下：

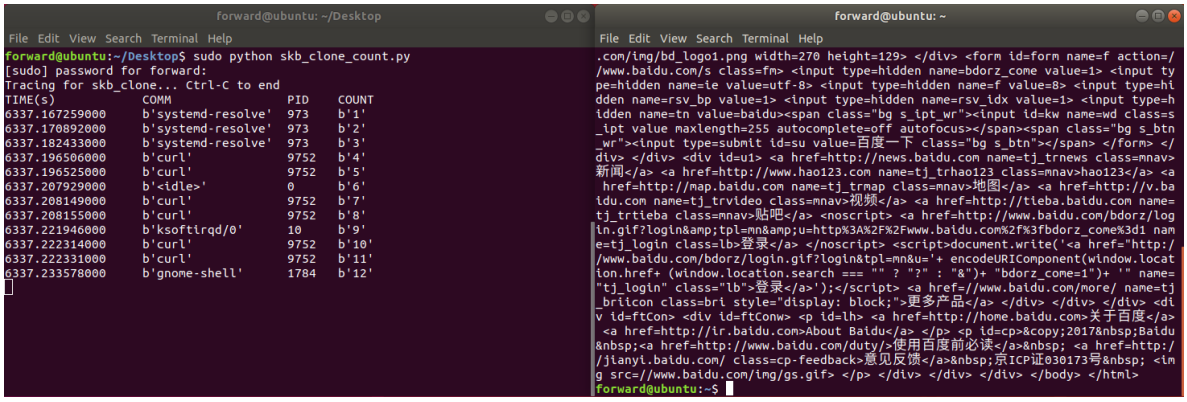
```
start = 0
def print_event(cpu, data, size):
    global start
    event = b["events"].event(data)  #获取由bpf函数perf_submit输出的数据
    if start == 0:
        start = event.ts
    time_s = (float(event.ts-start)) / 1000000000
    print("%-18.9f %-19s %-6d %d"%(time_s, event.comm, event.pid,
event.cur_count))

#打开perf缓冲区，并指定一个回调函数，当有数据时，自动使用该函数处理
b["events"].open_perf_buffer(print_event)
while True:
    try:
        b.perf_buffer_poll()  #等待perf缓冲区内容
    except  KeyboardInterrupt:
        exit()
```

全部示例代码参考附录。

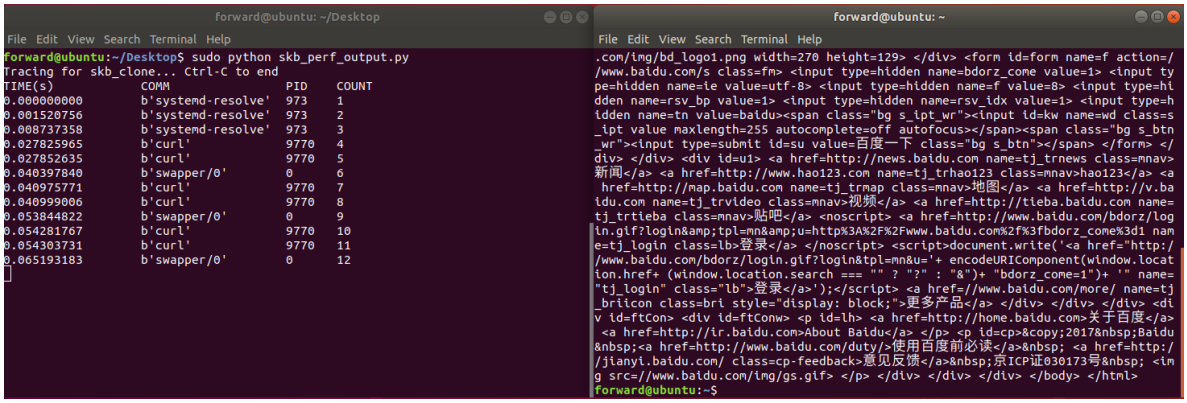
四、skb_clone实验结果

以`curl www.baidu.com`为例，监测`skb_clone()`的调用情况。

1. `trace_fields`



2. `BPF_PERF_OUTPUT()`



示例中两种方法测出的调用次数为12，可以多次测量观察差别。

不同版本、不同环境下会有不同，可以根据具体情况、结合输出内容深入分析相应网络行为。

五、参考资料及建议

- bcc仓库

  https://github.com/iovisor/bcc

- 推荐使用手册中对应版本，版本不一致的可参考仓库中相关资料

六、代码附录

1. `skb_clone_count.py`

```
from __future__ import print_function
from bcc import BPF

b = BPF(text="""
    #include <uapi/linux/ptrace.h>

    BPF_HASH(count);

    int do_trace(struct pt_regs *ctx){
        u64 cur_count = 1, key = 0;
        u64 *cur_ptr;
        cur_ptr = count.lookup(&key);
        if (cur_ptr == NULL){

            count.update(&key, &cur_count);
            bpf_trace_printk("%d\\n", cur_count);

        }else{

            cur_count = *cur_ptr + 1;
            count.update(&key, &cur_count);
            bpf_trace_printk("%d\\n", cur_count);

            }
        return 0;
    }
""")

b.attach_kprobe(event=("skb_clone"), fn_name="do_trace")
print("Tracing for skb_clone... Ctrl-C to end")
print("%-18s %-19s %-6s %s" % ("TIME(s)", "COMM", "PID", "COUNT"))
```

```
while True:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except KeyboardInterrupt:
        exit()
    print("%-18.9f %-19s %-6d %s" %(ts, task, pid, msg))
```

2. `skb_perf_output.py`

```
from __future__ import print_function
from bcc import BPF

prog = """

#include <linux/sched.h>
#include <uapi/linux/ptrace.h>

struct data_t{
    u32 pid;
    u64 ts;
    u64 cur_count;
    char comm[TASK_COMM_LEN];
};

BPF_PERF_OUTPUT(events);
BPF_HASH(count);

int do_trace(struct pt_regs *ctx){
    struct data_t data = {};

    data.pid = bpf_get_current_pid_tgid();
    data.ts = bpf_ktime_get_ns();
    bpf_get_current_comm(&data.comm, sizeof(data.comm));

    u64 cur_count = 1, key = 0;
    u64 *cur_ptr;
    cur_ptr = count.lookup(&key);

    if (cur_ptr == NULL){

        count.update(&key, &cur_count);
        data.cur_count = cur_count;
        events.perf_submit(ctx, &data, sizeof(data));

    }else{

        cur_count = *cur_ptr + 1;
        count.update(&key, &cur_count);
        data.cur_count = cur_count;
        events.perf_submit(ctx, &data, sizeof(data));

        }
    return 0;
}
"""

b = BPF(text = prog)
```

```
b.attach_kprobe(event = "skb_clone", fn_name="do_trace")
print("Tracing for skb_clone... Ctrl-C to end")
print("%-18s %-19s %-6s %s" % ("TIME(s)", "COMM", "PID", "COUNT"))

start = 0
def print_event(cpu, data, size):
    global start
    event = b["events"].event(data)
    if start == 0:
        start = event.ts
    time_s = (float(event.ts-start)) / 1000000000
    print("%-18.9f %-19s %-6d %d" % (time_s, event.comm, event.pid,
event.cur_count))

b["events"].open_perf_buffer(print_event)
while True:
    try:
        b.perf_buffer_poll()
    except  KeyboardInterrupt:
        exit()
```