



第二届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

# libbpf-bootstrap框架开发实 践

西邮Linux兴趣小组  
鹏

高

中国·西安

# 学习eBPF

若仅了解Linux系统编程，则视内核为黑盒。

若一头扎进Linux内核源码，初学者容易失去重点。

可以通过eBPF，对内核窥探一二。

对于学生，eBPF不仅是工具，也是学习内核的途径。



第二届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

# 1.libbpf-bootstrap框架

# 2.eBPF在安卓下的开发探索

中国·西安



第二届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

# 1.libbpf-bootstrap框架

中国·西安

## (1) 框架对比

BCC:

python

BCC 依赖于运行时编译，需要将整个庞大的 LLVM/Clang 库嵌入到自身中。  
读写BPF Map时需要编写半面向对象的C代码，和内核中发生的不完全匹配。  
开发和测试的迭代变得痛苦，可能在运行时有很多编译错误。

libbpf:

C/C++库，作为内核的一部分进行维护，位于tools/lib/bpf目录下

BPF CO-RE

libbpf提供了一个eBPF loader，用于处理LLVM生成的ELF文件，将其载入内核

## (2) libbpf-bootstrap框架特点

基于 libbpf 开发出来eBPF内核层代码，  
通过bpftool工具直接生成用户层代码操作接口。  
支持C语言和Rust语言  
xxx.bpf.c内核层代码，被 clang 编译器编译成 xxx.tmp.bpf.o  
bpftool 工具通过 xxx.tmp.bpf.o 自动生成 xxx.skel.h 头文件  
xxx.skel.h 头文件中包含了 xxx.bpf.c 对应的elf文件数据，  
以及用户层需要的 open, load, attach 等接口。

```
static inline struct minimal_bpf *open(const struct bpf_object_open_opts *opts = nullptr);  
static inline struct minimal_bpf *open_and_load();  
static inline int load(struct minimal_bpf *skel);  
static inline int attach(struct minimal_bpf *skel);  
static inline void detach(struct minimal_bpf *skel);  
static inline void destroy(struct minimal_bpf *skel);  
static inline const void *elf_bytes(size_t *sz);
```

## (3) libbpf-bootstrap框架生命周期

### open:

从 clang 编译器编译得到的eBPF程序elf文件中抽取 maps, eBPF程序, 全局变量等; 但是还未在内核中创建, 可以对 maps, 全局变量 进行修改。

### load:

maps, 全局变量 在内核中被创建, eBPF字节码程序加载到内核中, 并进行校验; 但这个阶段, eBPF程序虽然存在内核中, 但还不会被运行, 还可以对内核中的maps进行初始状态的赋值。

### attach:

eBPF程序被attach到挂接点, eBPF相关功能开始运行, 比如: eBPF程序被触发运行, 更新maps, 全局变量等。

### destroy:

eBPF程序被 detached, eBPF用到的资源将会被释放。

## (3) libbpf-bootstrap 框架优势

### BPF CO-RE:

一次编译，可以运行在不同版本的内核中

需要内核打开 CONFIG\_DEBUG\_INFO\_BTFF 配置

```
#define bpf_core_read(dst, sz, src) \
    bpf_probe_read_kernel(dst, sz, (const void *) __builtin_preserve_access_index(src))
```

\_\_builtin\_preserve\_access\_index 宏 clang 编译器编译时增加结构体字段重定位的信息

### 编写便捷简单:

提供了简单的示例程序和 Makefile 文件

不需要过多的配置和构建，方便开发



## (3) libbpf-bootstrap框架优势

### 全局变量 (Linux 5.5+)

可以用于配置BPF程序、存放轻量的统计数据、在内核和用户空间传递数据。

通过BPF skeleton来访问这些变量,在用户空间对这些变量的修改, 会立刻反映到内核空间。

常作为命令行参数进行过滤的媒介。

### BPF ring buffer map (Linux 5.8+)

相比 perf buffer, 两者底层都使用epoll

perf-buffer per-CPU 环形缓冲区

ring buffer 分配 CPU 共享的缓冲区

内存效率更高、保证事件顺序, 性能也更高。

## (4) 示例程序-进程调度监控

B站 bailu12311

tracepoint 利用 tp/sched/sched\_process\_exec和

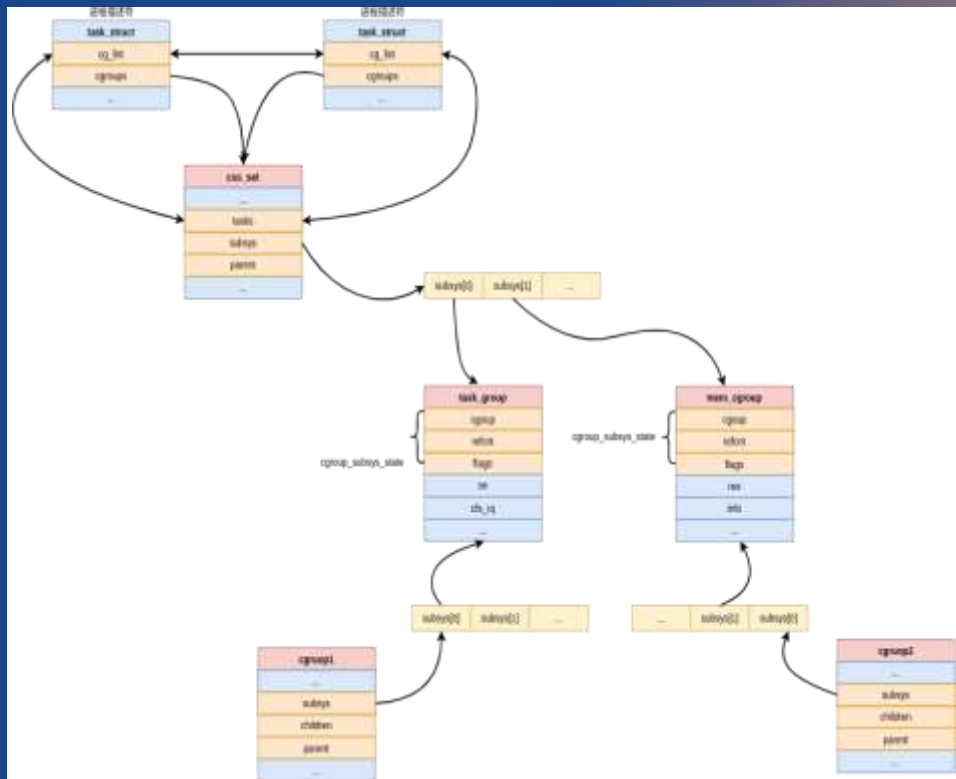
tp/sched/sched\_process\_exit

判断为宿主机/容器 task\_struct cgroup中的container\_id

通过bpf\_send\_signal(9)将有阻断策略的进程杀死

ringbuffer map 用于从内核向用户空间发送数据

ID	用户id	组id	宿主机/容器	PPID	PID	进程名	进程所属路径	调度状态	运行...	所耗时间 (m...
61957	0	0	宿主机	26707	26711	sleep		执行结束	0	1002
61967	0	0	宿主机	2201	26706	sh		执行结束	0	1015
61958	0	0	宿主机	26707	26712	sed	/usr/bin/sed	正在执行	0	0
61959	0	0	宿主机	26707	26712	sed		执行结束	0	0
61960	0	0	宿主机	26707	26713	cat	/usr/bin/cat	正在执行	0	0
61961	0	0	宿主机	26707	26713	cat		执行结束	0	0
61962	0	0	宿主机	26707	26714	cpuUsage.sh		执行结束	0	0
61963	0	0	宿主机	26707	26715	cat	/usr/bin/cat	正在执行	0	0





第二届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

## 2. eBPF在安卓下的开发实践

中国·西安

# (1) AOSP源码开发流程

内核态:

- `DEFINE_BPF_MAP(name_of_my_map, ARRAY, int, uint32_t, 10);`  
定义MAP用来实现用户程序和内核互传数据的共享缓存。
- `DEFINE_BPF_PROG("PROGTYPE/PROGNAME", AID_*, AID_*, PROGFUNC)(.args..),`  
定义了一个BPF函数, 这个函数编译后, 可以加载进内核, 实现钩子函数的功能。
- `LICENSE("GPL")` 许可协议声明

用户态:

- `bpf_obj_get()`
- `bpf_attach_tracepoint()/bpf_attach_kprobe()`

编写android.bp文件

## (2) 如何加载?

源码/system/bpf/libbpf\_android/Loader.cpp

安卓规定bpfloader是唯一可以加载bpf程序的程序

bootloader在系统启动时自动扫描并解析 /system/etc/bpf/\*.o btf格式目标文件，读取并校验 critical、license、bpfloader\* 相关 section 数据。

loadProg解析bpf程序文件，实现 loadCodeSections 函数创建Bpf程序和 createMaps 函数创建 Map。

在 /sys/fs/bpf 目录生成相应的/sys/fs/bpf/prog\_FILENAME\_PROGTYPE\_PROGNAME和 /sys/fs/bpf/map\_FILENAME\_MAPNAME 文件

```
// /sys/fs/bpf/prog_<filename>_<mapname>
    progPinLoc = string(BPF_FS_PATH) + "prog_" + fname + "_" + cs[i].name;
// Format of pin location is /sys/fs/bpf/map_<filename>_<mapname>
    mapPinLoc = string(BPF_FS_PATH) + "map_" + fname + "_" + string(mapNames[i]);
```

首先把编译好的bpf.o程序放到 /system/etc/bpf/ 目录下，这就要求我们需要有 /system 目录的可写权限，重新挂载根目录。加载器文件需要通过 /sys/fs/bpf 目录下的文件来和BPF程序进行交互。

## (3) 问题

map的类型定义错误

只要是使用结构体作value值的话，且map在内核态进行了lookup操作，那么map使用ARRAY类型就会导致map无法产生，但是如果是只进行了update操作没有lookup的话就可以使用ARRAY类型也可以使用HASH。

一些辅助函数无法使用

prog无法产生的原因

- 1.没有进行错误处理
- 2.用户态读取map地址出错

构建较繁琐，开发较复杂，编译时间较长，不容易排错

## (4) libbpf-bootstrap交叉编译

依赖:libelf, zlib

交叉编译工具链 aarch64-linux-gnu

交叉编译libbpf-bootstrap

```
make EXTRA_CFLAGS="-IXXX" EXTRA_LDFLAGS="-LXXX" ARCH=arm64
```

EXTRA\_CFLAGS 指定 libelf 和 zlib 的头文件路径

EXTRA\_LDFLAGS 指定 libelf 和 zlib 的库文件路径



## (5) libbpf-bootstrap-android

西邮 刘佳欢:

<https://github.com/linuxkerneltravel/libbpf-bootstrap-android>

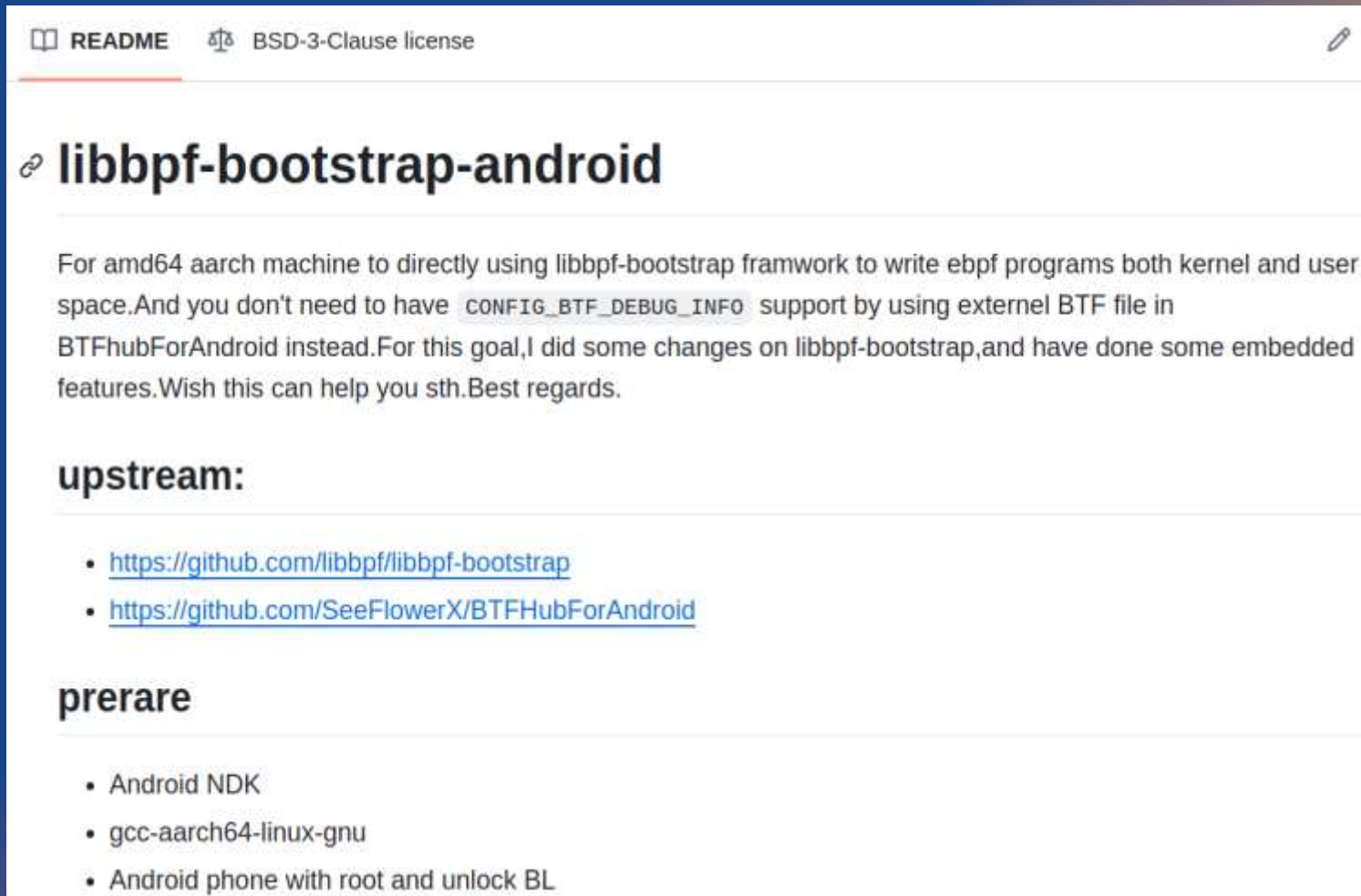
准备:

Android NDK

gcc-aarch64-linux-gnu

Android phone with root and unlock BL

ebpf程序源码可以无需任何改动  
就运行在安卓kernel



The screenshot shows the GitHub repository page for **libbpf-bootstrap-android**. At the top, it indicates the license is BSD-3-Clause. The main heading is **libbpf-bootstrap-android**. The description states: "For amd64 aarch machine to directly using libbpf-bootstrap framework to write ebpf programs both kernel and user space. And you don't need to have `CONFIG_BT_DEBUG_INFO` support by using external BTF file in BTFHubForAndroid instead. For this goal, I did some changes on libbpf-bootstrap, and have done some embedded features. Wish this can help you sth. Best regards." Below this, under the heading **upstream:**, there are two links: <https://github.com/libbpf/libbpf-bootstrap> and <https://github.com/SeeFlowerX/BTFHubForAndroid>. Under the heading **prerare** (likely a typo for prerequisites), there is a list of requirements: Android NDK, gcc-aarch64-linux-gnu, and Android phone with root and unlock BL.

README BSD-3-Clause license

### libbpf-bootstrap-android

For amd64 aarch machine to directly using libbpf-bootstrap framework to write ebpf programs both kernel and user space. And you don't need to have `CONFIG_BT_DEBUG_INFO` support by using external BTF file in BTFHubForAndroid instead. For this goal, I did some changes on libbpf-bootstrap, and have done some embedded features. Wish this can help you sth. Best regards.

**upstream:**

- <https://github.com/libbpf/libbpf-bootstrap>
- <https://github.com/SeeFlowerX/BTFHubForAndroid>

**prerare**

- Android NDK
- gcc-aarch64-linux-gnu
- Android phone with root and unlock BL



## (6) 示例程序-系统调用监控

参数指定pid的系统调用，设置程序监测的时长

监控信息

'PID, command, syscall\_number, syscall\_name, syscall\_runtime'

'进程id 进程名 系统调用号 系统调用名 系统调用运行时间'

pid=4379	comm=InteractionJank	syscall_number=113	syscall_name=clock_gettime	syscall_runtime_ns=2239
pid=4379	comm=InteractionJank	syscall_number=64	syscall_name=write	syscall_runtime_ns=2448
pid=4379	comm=InteractionJank	syscall_number=64	syscall_name=write	syscall_runtime_ns=1822
pid=4379	comm=InteractionJank	syscall_number=174	syscall_name=getuid	syscall_runtime_ns=1041
pid=4379	comm=InteractionJank	syscall_number=22	syscall_name=epoll_pwait	syscall_runtime_ns=1719
pid=4661	comm=ZAS_DataCollect	syscall_number=29	syscall_name=ioctl	syscall_runtime_ns=7865
pid=4661	comm=ZAS_DataCollect	syscall_number=64	syscall_name=write	syscall_runtime_ns=39844
pid=4661	comm=ZAS_DataCollect	syscall_number=64	syscall_name=write	syscall_runtime_ns=3282
pid=4661	comm=ZAS_DataCollect	syscall_number=174	syscall_name=getuid	syscall_runtime_ns=1146
pid=4661	comm=ZAS_DataCollect	syscall_number=22	syscall_name=epoll_pwait	syscall_runtime_ns=3386
pid=4379	comm=jank	syscall_number=63	syscall_name=read	syscall_runtime_ns=2865
pid=21714	comm=encent.mobileqq	syscall_number=64	syscall_name=write	syscall_runtime_ns=6823
pid=21714	comm=encent.mobileqq	syscall_number=215	syscall_name=munmap	syscall_runtime_ns=10469
pid=21714	comm=encent.mobileqq	syscall_number=64	syscall_name=write	syscall_runtime_ns=3959
pid=21714	comm=encent.mobileqq	syscall_number=66	syscall_name=writev	syscall_runtime_ns=20573
pid=21714	comm=encent.mobileqq	syscall_number=64	syscall_name=write	syscall_runtime_ns=4375
pid=21714	comm=encent.mobileqq	syscall_number=67	syscall_name=pread64	syscall_runtime_ns=269166
pid=21714	comm=encent.mobileqq	syscall_number=67	syscall_name=pread64	syscall_runtime_ns=2396
pid=21714	comm=encent.mobileqq	syscall_number=222	syscall_name=mmap	syscall_runtime_ns=7708
pid=21714	comm=encent.mobileqq	syscall_number=215	syscall_name=munmap	syscall_runtime_ns=7135

## (6) 示例程序-系统调用监控

android设备没有打开CONFIG\_FTRACE\_SYSCALLS=y配置，所以并不支持syscalls类型的tracepoint。  
但可以使用tp/raw\_syscalls/sys\_enter和tp/raw\_syscalls/sys\_exit为挂载点,系统调用的统一入口。

获取到系统调用号，在arm64下找到对应的系统调用名。

定义全局变量，用于存储目标进程的PID

Map用于enter和exit之间数据交互，key为pid，value为时间戳

syscall\_runtime 计算为 (sys\_exit\_time - sys\_enter\_time)

```
struct trace_event_raw_sys_enter {  
    struct trace_entry ent;  
    long int id;  
    long unsigned int args[6];  
    char __data[0];  
};  
  
struct trace_event_raw_sys_exit {  
    struct trace_entry ent;  
    long int id;  
    long int ret;  
    char __data[0];  
};
```

## 参考:

libbpf-bootstrap: <https://github.com/libbpf/libbpf-bootstrap>

西邮 刘佳欢: <https://github.com/linuxkerneltravel/libbpf-bootstrap-android>

B站 张口就问: [https://space.bilibili.com/384754914/?spm\\_id\\_from=333.999.0.0](https://space.bilibili.com/384754914/?spm_id_from=333.999.0.0)

西邮 屈雷皓: [https://blog.csdn.net/m0\\_68715848?type=blog](https://blog.csdn.net/m0_68715848?type=blog)

B站 bailu12311: <https://space.bilibili.com/324170812>



第二届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

感谢 聆听  
感谢 大家

中国·西安