eBPF CO-RE

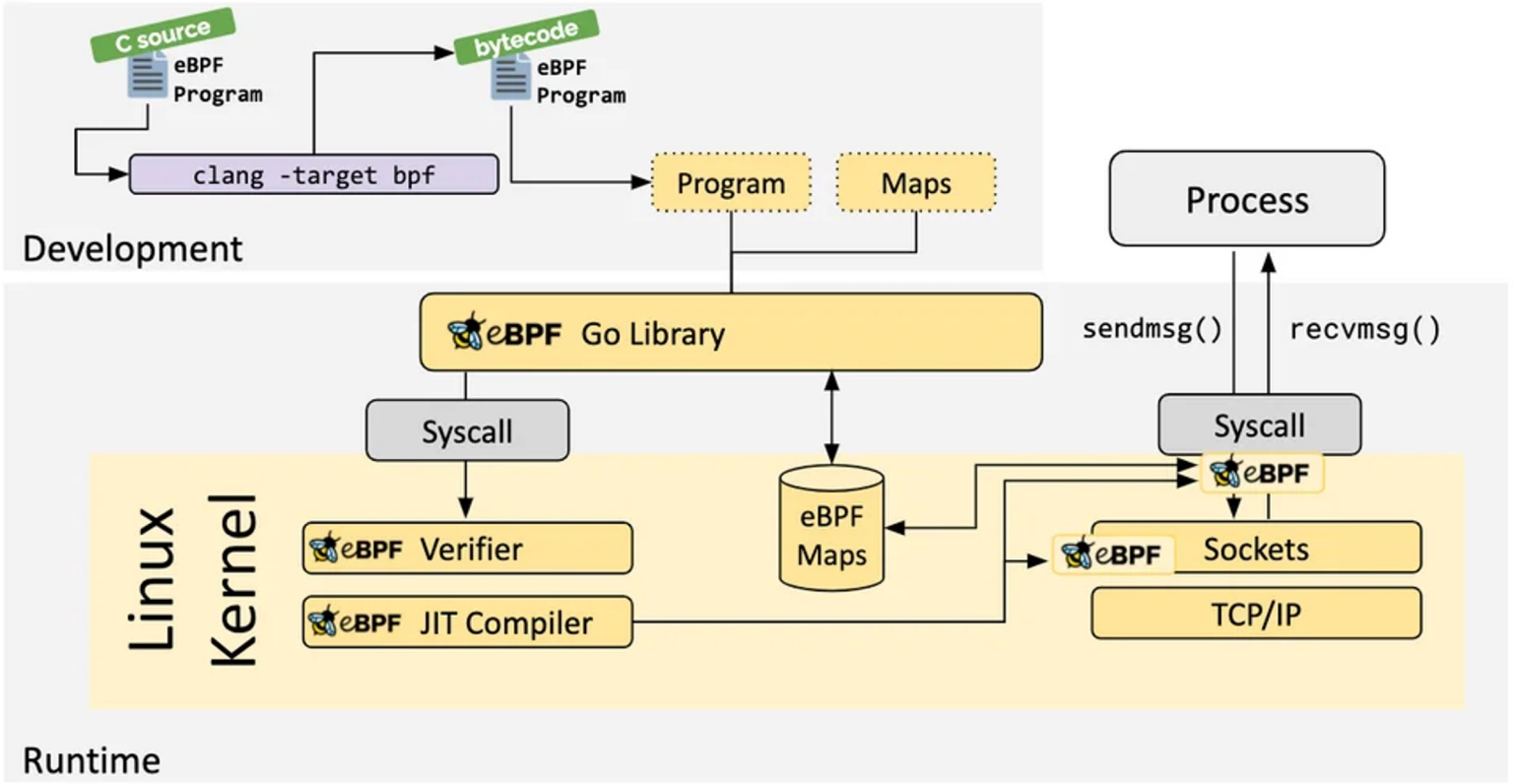
05-241019 永冨 敬士

目次

eBPFの目的

- カーネル空間でプログラムを動かしたい
  - 計測・ネットワーク制御
- 従来の方法: カーネルモジュール
  - プログラムの安全性は保証されていない
  - 柔軟にロード出来ない
- eBPFの利点
  - カーネル側でのプログラムの検証
  - システムコールでの柔軟なeBPFプログラムのロード

eBPFのアーキテクチャ



https://ebpf.io/ja/what-is-ebpf/#ローダと検証器のアーキテクチャ

目次

以降に出てくる例は次のgithubリポジトリ

https://github.com/taka231/ebpf_loader

のexamplesディレクトリ以下にある例です。

例1: 単にパケットをdropするプログラム

- xdpはNICレベルでパケットの操作を行うeBPFのプログラムの種類
  - 引数にパケットの情報
  - 戻り値にパケットの操作
- コンパイルはclangで-target bpfオプションをつけて行う

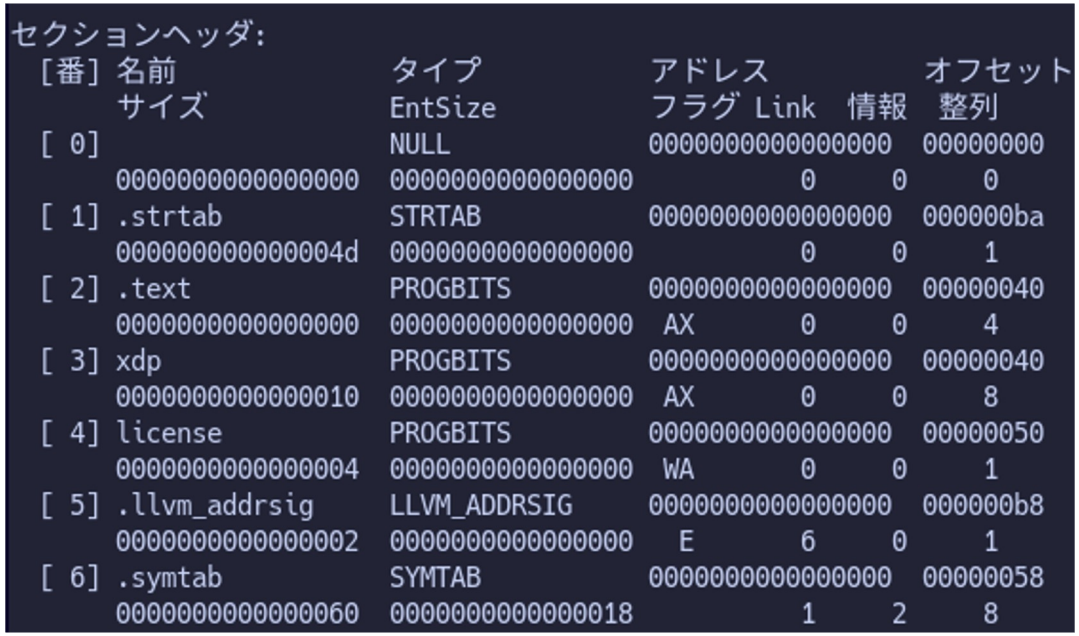
```c
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("xdp")
int xdp_prog_simple(struct xdp_md *ctx) {
    return XDP_DROP;
}

char _license[] SEC("license") = "GPL";
```

例1のプログラムを実行する手順

1. 実行して得られたelf形式のバイナリから、xdpセクションの中身を取り出す



```
セクションヘッダ:
  [番] 名前              タイプ            アドレス          オフセット
       サイズ            EntSize           フラグ Link  情報  整列
  [ 0]                   NULL              0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .strtab           STRTAB            0000000000000000  000000ba
       000000000000004d  0000000000000000           0     0     1
  [ 2] .text             PROGBITS          0000000000000000  00000040
       0000000000000000  0000000000000000  AX       0     0     4
  [ 3] xdp               PROGBITS          0000000000000000  00000040
       0000000000000010  0000000000000000  AX       0     0     8
  [ 4] license           PROGBITS          0000000000000000  00000050
       0000000000000004  0000000000000000  WA       0     0     1
  [ 5] .llvm_addrsig     LLVM_ADDRSIG      0000000000000000  000000b8
       0000000000000002  0000000000000000   E       6     0     1
  [ 6] .symtab           SYMTAB            0000000000000000  00000058
       0000000000000060  0000000000000018           1     2     8
```

- 左図は例1のプログラムのコンパイル生成物をreadelfした結果
- xdpセクションのボディに、eBPFバイトコードがそのまま入っている

例1のプログラムを実行する手順

2. bpfシステムコールでeBPFプログラムをロードする

```
int bpf(int cmd, union bpf_attr *attr, unsigned int size);
```

- bpfシステムコールは、複数のサブコマンドからなっている
- プログラムのロードには、BPF_PROG_LOADを用いる

man bpfより→

```
struct {    /* Used by BPF_PROG_LOAD */
    __u32         prog_type;
    __u32         insn_cnt;
    __aligned_u64 insns;      /* 'const struct bpf_insn *' */
    __aligned_u64 license;    /* 'const char *' */
    __u32         log_level;  /* verbosity level of verifier */
    __u32         log_size;   /* size of user buffer */
    __aligned_u64 log_buf;    /* user supplied 'char *'
                                 buffer */
    __u32         kern_version;
                              /* checked when prog_type=kprobe
                                 (since Linux 4.1) */
};
```

例1のプログラムを実行する手順

3. プログラムをxdpにアタッチする

- bpfシステムコールのサブコマンドであるBPF_LINK_CREATEを用いる
  - 以前はattach先によって複数のシステムコールを呼ぶ必要があったが、今は大体bpfのサブコマンドで出来る模様

例2: mapを使ったrelocationを伴うプログラム

- mapのkey 0に対応する値が1の場合にパケットをdropし、その他の場合はpassするプログラム
- mapを用いているので、システムコールで生成したmapのfdをeBPFプログラムのバイトコードに埋め込む工程が必要となる。
- mapの生成にはBPF_MAP_CREATEサブコマンドを用いる

```c
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __uint(max_entries, 1);
    __type(key, __u32);
    __type(value, __u32);
} drop_flag SEC(".maps");

SEC("xdp")
int xdp_prog_map(struct xdp_md *ctx) {
    __u32 key = 0;
    __u32 *flag = bpf_map_lookup_elem(&drop_flag, &key);

    if (flag && *flag == 1) {
        return XDP_DROP;
    }

    return XDP_PASS;
}

char _license[] SEC("license") = "GPL";
```

例2: mapを使ったrelocationを伴うプログラム

- readelfの結果を見ると.relxdp と.mapsが増えている
- .relxdpにはxdpセクションの relocation情報が入っている
- .mapsには上記eBPFプログラム中で宣言したmapの情報が入っている

```
セクションヘッダ:
  [番] 名前              タイプ            アドレス          オフセット
       サイズ            EntSize           フラグ Link  情報  整列
  [ 0]                   NULL              0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .strtab           STRTAB            0000000000000000  00000163
       0000000000000062  0000000000000000           0     0     1
  [ 2] .text             PROGBITS          0000000000000000  00000040
       0000000000000000  0000000000000000  AX       0     0     4
  [ 3] xdp               PROGBITS          0000000000000000  00000040
       0000000000000070  0000000000000000  AX       0     0     8
  [ 4] .relxdp           REL               0000000000000000  00000150
       0000000000000010  0000000000000010   I       8     3     8
  [ 5] .maps             PROGBITS          0000000000000000  000000b0
       0000000000000020  0000000000000000  WA       0     0     8
  [ 6] license           PROGBITS          0000000000000000  000000d0
       0000000000000004  0000000000000000  WA       0     0     1
  [ 7] .llvm_addrsig     LLVM_ADDRSIG      0000000000000000  00000160
       0000000000000003  0000000000000000   E       8     0     1
  [ 8] .symtab           SYMTAB            0000000000000000  000000d8
       0000000000000078  0000000000000018           1     2     8
```

例2: mapを使ったrelocationを伴うプログラム

- https://docs.kernel.org/bpf/llvm_reloc.htmlより
- r_offsetは書き換える対象の命令のoffset
- r_infoはrelocation typeとsymbol index(elfのsymbol tableのindex)を含む

```
typedef struct
{
  Elf64_Addr    r_offset;  // Offset from the beginning of section.
  Elf64_Xword   r_info;    // Relocation type and symbol index.
} Elf64_Rel;
```

Enum	ELF Reloc Type	Description	BitSize	Offset	Calculation
0	R_BPF_NONE	None			
1	R_BPF_64_64	ld_imm64 insn	32	r_offset + 4	S + A
2	R_BPF_64_ABS64	normal data	64	r_offset	S + A
3	R_BPF_64_ABS32	normal data	32	r_offset	S + A
4	R_BPF_64_NODYLD32	.BTF[.ext] data	32	r_offset	S + A
10	R_BPF_64_32	call insn	32	r_offset + 4	(S + A) / 8 - 1

- 以下は例2のプログラムの.relxdpセクションの中身をparseしたもの
  [Elf64Rel { r_offset: 32, rel_type: RBpf64_64, sym_idx: 3 }]

例2: mapを使ったrelocationを伴うプログラム

- R_BPF_64_64は即値代入命令の即値を書き換えるrelocation type
  - (非公式の)命令仕様書
- 単純にimmをBPF_MAP_CREATEの戻り値のfdで書き換えると、ロード時にmap_ptrを要求しているがscalarが代入されたというエラーが出る
- srcの値によって何を代入するかが表される仕様となっている
  - これを1にするとmap fdの代入となる

```
3: (07) r2 += -4                                ; R2_w=fp-4
4: (18) r1 = 0x3                                ; R1_w=3
6: (85) call bpf_map_lookup_elem#1
R1 type=scalar expected=map_ptr
processed 6 insns (limit 1000000) max_states_per_insn 0
```

目次

BPF CO-REとは？

- eBPFのプログラムはカーネル構造体に依存
- カーネル内の型定義はバージョンによって異なる可能性がある
- ある環境でコンパイルしたeBPFプログラムを、別の環境でもコンパイルし直さずに動かしたい

⇨ BPF CO-RE(Compile Once, Run Everywhere)はこの目的のためのlibbpf(bpfのライブラリ)の機能

- コンパイラが吐くelfバイナリ内のBTF(BPF Type Format)とカーネルの型定義を比較して、バイトコード列を書き換える
- カーネルの型定義は、カーネルのビルド時に生成され、BTFバイナリとして/sys/kernel/btf/vmlinuxに置かれる

例3: CO-RE

- IPv6であった場合はパケットをdropするプログラム
- カーネルの構造体定義では無く、vmlinux.hを読み込んでいる
- vmlinux.hは、bpftoolコマンドを使って/sys/kernel/btf/vmlinuxをC言語のヘッダファイルに変換したもの
- 今回は変換したvmlinux.hの中のethhdr構造体のfieldを入れ替えたものを用いて誤ったobject fileを生成した

```c
#include "vmlinux.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>
#include <bpf/bpf_core_read.h>

#define ETH_P_IPV6  0x86DD

SEC("xdp")
int xdp_drop_ipv6(struct xdp_md *ctx) {
    void *data = (void *)(long)ctx->data;
    void *data_end = (void *)(long)ctx->data_end;

    struct ethhdr *eth = data;
    if ((void *)(eth + 1) > data_end)
        return XDP_PASS;

    __u16 proto = bpf_ntohs(eth->h_proto);
    if (proto == ETH_P_IPV6)
        return XDP_DROP;

    return XDP_PASS;
}

char _license[] SEC("license") = "GPL";
```

例3: CO-RE

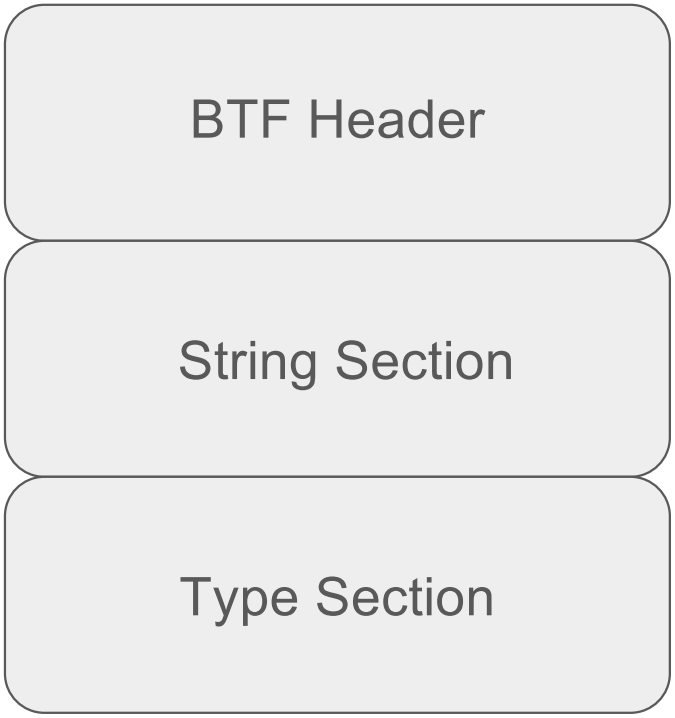
左図はreadelfの結果を抜粋したもの

- 今回用いるのは.BTFと.BTF.ext
- .BTFは主にeBPFプログラム中で使われている型の情報を持つ
- .BTF.extはCO-RE relocationの情報を持つ

```
[12] .debug_addr       PROGBITS         0000000000000000  00000500
     0000000000000018  0000000000000000           0     0     1
[13] .rel.debug_addr   REL              0000000000000000  00000e60
     0000000000000020  0000000000000010   I      24    12     8
[14] .BTF              PROGBITS         0000000000000000  00000518
     00000000000003a8  0000000000000000           0     0     4
[15] .rel.BTF          REL              0000000000000000  00000e80
     0000000000000010  0000000000000010   I      24    14     8
[16] .BTF.ext          PROGBITS         0000000000000000  000008c0
     00000000000000ec  0000000000000000           0     0     4
[17] .rel.BTF.ext      REL              0000000000000000  00000e90
     00000000000000b0  0000000000000010   I      24    16     8
```

例3: CO-RE

.BTFの構造は以下のようになっている



- BTF Headerには２つのsectionのoffsetとlenの情報が入っている
- String Sectionにはnull文字で区切られた文字列が入っている(elfの表現と同じ)
- Type Sectionには型定義の情報(可変長)のVecが入っている
  - https://docs.kernel.org/bpf/btf.html#type-encoding

例3: CO-RE

.BTF.extの構造は次のようになっている

- BTF EXT Headerは3つのSectionのoffsetとlenを持つ(CO-RE Infoはoptional)
- CO-RE InfoにはSection毎のrelocation情報が入っている
  - https://docs.kernel.org/bpf/btf.html#btf-ext-section

```
struct btf_ext_info_sec {
    __u32   sec_name_off; /* offset to section name */
    __u32   num_info;
    /* Followed by num_info * record_size number of bytes */
    __u8    data[0];
};
```

例3: CO-RE

CO-RE relocationの詳細

- relocationの種類

```
enum bpf_core_relo_kind {
        BPF_CORE_FIELD_BYTE_OFFSET = 0,  /* field byte offset */
        BPF_CORE_FIELD_BYTE_SIZE   = 1,  /* field size in bytes */
        BPF_CORE_FIELD_EXISTS      = 2,  /* field existence in target kernel */
        BPF_CORE_FIELD_SIGNED      = 3,  /* field signedness (0 - unsigned, 1 - signed) */
        BPF_CORE_FIELD_LSHIFT_U64  = 4,  /* bitfield-specific left bitshift */
        BPF_CORE_FIELD_RSHIFT_U64  = 5,  /* bitfield-specific right bitshift */
        BPF_CORE_TYPE_ID_LOCAL     = 6,  /* type ID in local BPF object */
        BPF_CORE_TYPE_ID_TARGET    = 7,  /* type ID in target kernel */
        BPF_CORE_TYPE_EXISTS       = 8,  /* type existence in target kernel */
        BPF_CORE_TYPE_SIZE         = 9,  /* type size in bytes */
        BPF_CORE_ENUMVAL_EXISTS    = 10, /* enum value existence in target kernel */
        BPF_CORE_ENUMVAL_VALUE     = 11, /* enum value integer value */
        BPF_CORE_TYPE_MATCHES      = 12, /* type match in target kernel */
};
```

例3: CO-RE

これらの情報を用いたrelocationの手順 (field byte offsetの場合)

1. .BTF.extのCO-RE relocation情報から、命令のoffset, type_id(BTFのtype sectionのインデックス), access_str_off, relocation kindを取得する
2. relocation kindがfield byte offsetならば、type_idでstructの定義を取得し、access_str_offからstructの何番目のfieldかを特定する。
3. 同名のstruct, fieldの型をvmlinuxのBTFから探し、offsetを取得する
4. 命令のoffsetをこのoffsetに置き換える

例3: CO-RE

```
0000000000000000 <xdp_drop_ipv6>:
       0:       b7 00 00 00 02 00 00 00 r0 = 0x2
       1:       61 12 04 00 00 00 00 00 r2 = *(u32 *)(r1 + 0x4)
                0000000000000008:  CO-RE <byte_off> [2] struct xdp_md::data_end (0:1)
       2:       61 11 00 00 00 00 00 00 r1 = *(u32 *)(r1 + 0x0)
                0000000000000010:  CO-RE <byte_off> [2] struct xdp_md::data (0:0)
       3:       bf 13 00 00 00 00 00 00 r3 = r1
       4:       07 03 00 00 0e 00 00 00 r3 += 0xe
       5:       2d 23 04 00 00 00 00 00 if r3 > r2 goto +0x4 <xdp_drop_ipv6+0x50>
       6:       69 11 00 00 00 00 00 00 r1 = *(u16 *)(r1 + 0x0)
                0000000000000030:  CO-RE <byte_off> [8] struct ethhdr::h_proto (0:0)
       7:       b7 00 00 00 01 00 00 00 r0 = 0x1
       8:       15 01 01 00 86 dd 00 00 if r1 == 0xdd86 goto +0x1 <xdp_drop_ipv6+0x50>
       9:       b7 00 00 00 02 00 00 00 r0 = 0x2
      10:       95 00 00 00 00 00 00 00 exit
```