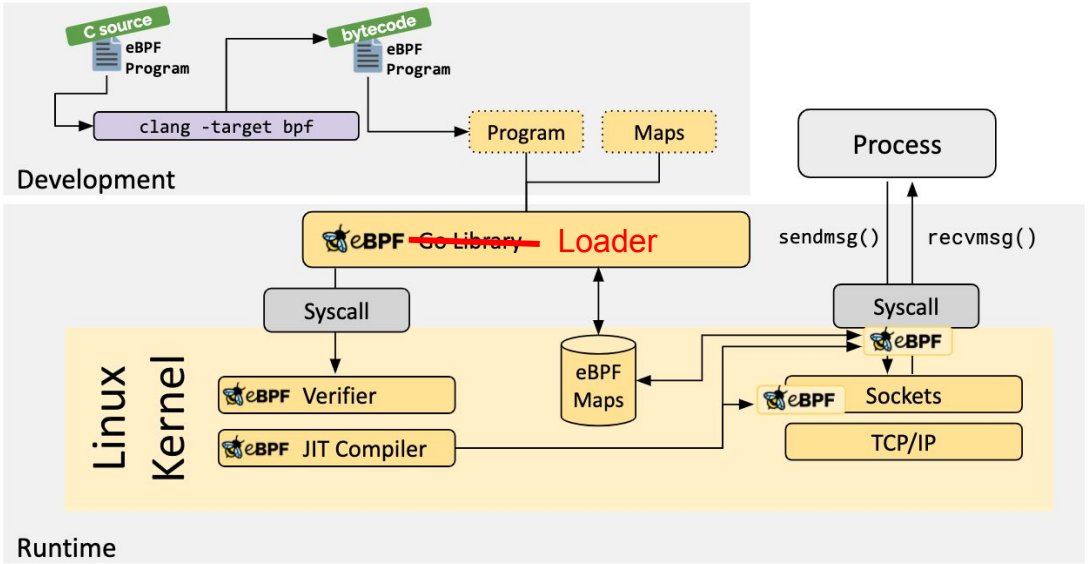
eBPF Loader Deep Dive

Dylan Reimerink @ Isovalent

What is an eBPF loader (library)?

- Load eBPF objects into the kernel
- Loaders:
  - IP / TC
  - bpftool
  - bpftrace
  - <your app>
- Abstraction / Library / SDK
- Loader libraries:
  - Libbpf
  - Aya
  - BCC
  - ebpf-go (cilium/ebpf)



Example program

```c
#include [...]

struct flow_key
{
    __be32 saddr;
    __be32 daddr;
};

struct flow_value
{
    __u32 bytes;
    __u32 packets;
};

struct
{
    __uint(type, BPF_MAP_TYPE_HASH);
    __type(key, struct flow_key);
    __type(value, struct flow_value);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
    __uint(max_entries, 200);
    __uint(map_flags, BPF_F_NO_PREALLOC);
} flow_map SEC(".maps");
```

```c
__attribute__((noinline)) int handle_ipv4(void *data,
void *data_end);

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

    struct ethhdr *l2 = data;
    if (data + sizeof(l2) > data_end)
    {
        return XDP_PASS;
    }

    switch (l2->h_proto)
    {
    case bpf_htons(ETH_P_IP):
        return handle_ipv4(data + sizeof(l2), data_end);
    }

    return XDP_PASS;
}
```

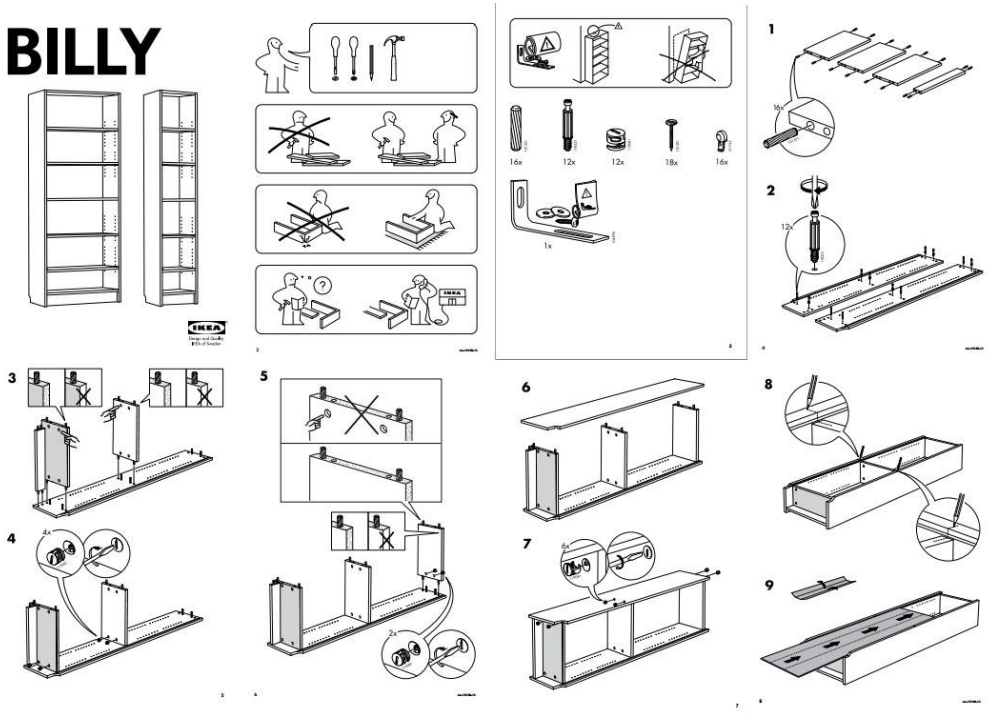
What do I get?

- Executable and Linkable Format
-
```
$ clang -o out.o helloworld.c
$ file out.o
out.o: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2
$ chmod +x out.o && ./out.o
Hello, World!
```
-
```
$ clang -target bpf -Wall -O2 -g -c example.c -I/usr/include -o example
$ file example
example: ELF 64-bit LSB relocatable, eBPF, version 1 (SYSV), with debug_info, not stripped
$ chmod +x out.o && ./out.o
zsh: exec format error: ./example
```

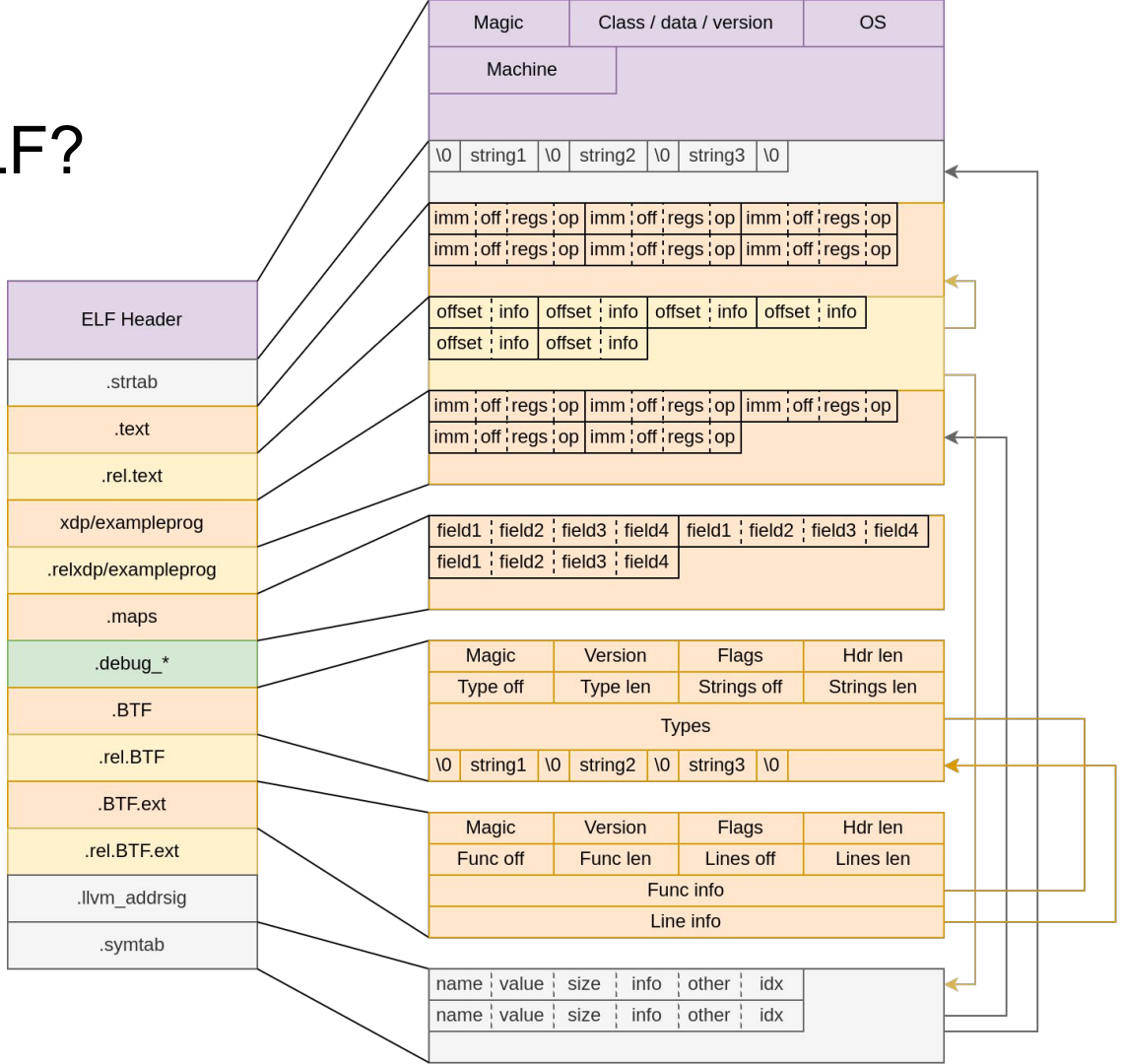
What is an ELF?

Executable

Relocatable



What is an ELF?



BPF Syscall

```
$ man bpf
BPF(2)                          Linux Programmer's Manual

NAME
       bpf - perform a command on an extended BPF map or program

SYNOPSIS
       #include <linux/bpf.h>

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

```
union bpf_attr {
    /* [...] */
    struct { /* anonymous struct used by BPF_MAP_*_ELEM commands */
        __u32       map_fd;
        __aligned_u64   key;
        union {
            __aligned_u64 value;
            __aligned_u64 next_key;
        };
        __u64       flags;
    };
    /* [...] */
} __attribute__((aligned(8)));
```

BPF Commands

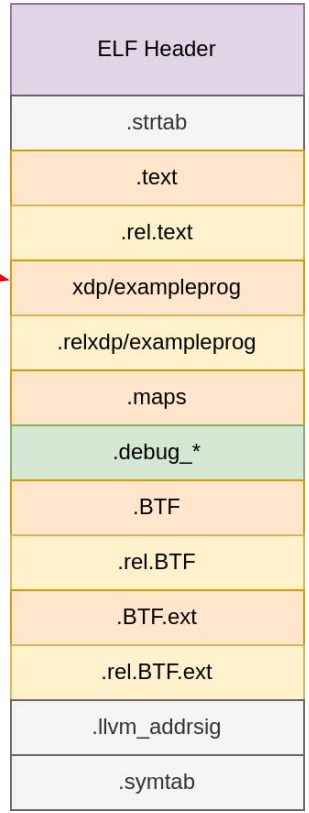
- Programs
  - BPF_PROG_LOAD
  - BPF_PROG_{ATTACH,DETACH}
  - BPF_PROG_{TEST_RUN,RUN}
  - BPF_PROG_QUERY
  - BPF_PROG_BIND_MAP
- Maps
  - BPF_MAP_CREATE
  - BPF_MAP_{LOOKUP,UPDATE,DELETE}_ELEM
  - BPF_MAP_GET_NEXT_KEY}
  - BPF_MAP_{LOOKUP,UPDATE,DELETE}_BATCH
  - BPF_MAP_LOOKUP_AND_DELETE_{ELEM,BATCH}
  - BPF_MAP_FREEZE
- BTF
  - BPF_BTF_LOAD

BPF Commands

- Link
  - BPF_LINK_{CREATE,UPDATE,DETACH}
- Generic
  - BPF_OBJ_PIN
  - BPF_OBJ_GET
  - BPF_OBJ_GET_INFO_BY_FD
  - BPF_{PROG,MAP,BTF,LINK}_GET_NEXT_ID
  - BPF_{PROG,MAP,BTF,LINK}_GET_FD_BY_ID
- Misc
  - BPF_RAW_TRACEPOINT_OPEN
  - BPF_TASK_FD_QUERY
  - BPF_ENABLE_STATS
  - BPF_ITER_CREATE

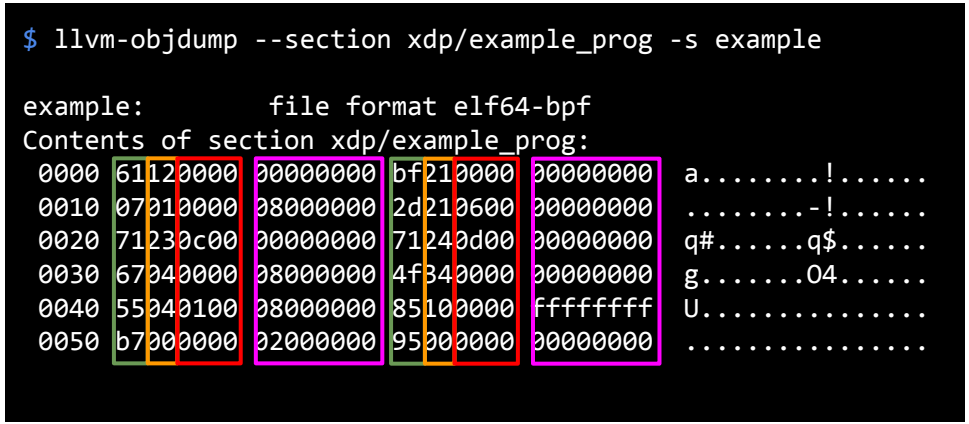
Programs

```c
SEC("xdp/example_prog")
int example_prog(struct xdp_md *ctx)
{
    void *data = (void *)(unsigned long)ctx->data;
    void *data_end = (void *)(unsigned long)ctx->data;

    struct ethhdr *l2 = data;
    if (data + sizeof(l2) > data_end)
    {
        return XDP_PASS;
    }

    switch (l2->h_proto)
    {
    case bpf_htons(ETH_P_IP):
        return handle_ipv4(data + sizeof(l2), data_end);
    }

    return XDP_PASS;
}
```



Programs

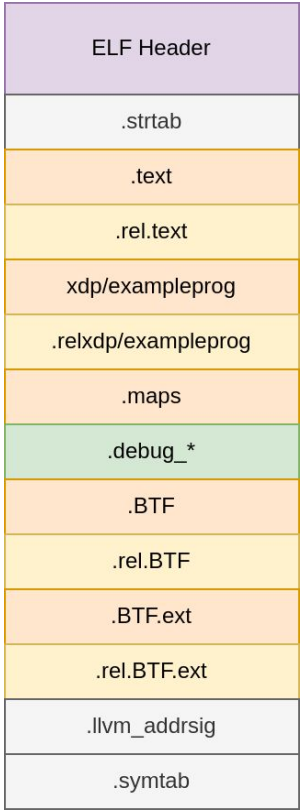
- Opcode (1 byte)
- Src + Dst resisters (1 byte)
- Offset (2 bytes)
- Intermediate (4 bytes)
- 64-bit intermediate values use 2 instructions



Programs

```
$ llvm-objdump --section xdp/example_prog -s example

example:        file format elf64-bpf

Disassembly of section xdp/example_prog:

0000000000000000 <example_prog>:
       0:       61 12 00 00 00 00 00 00 r2 = *(u32 *)(r1 + 0)
       1:       bf 21 00 00 00 00 00 00 r1 = r2
       2:       07 01 00 00 08 00 00 00 r1 += 8
       3:       2d 21 06 00 00 00 00 00 if r1 > r2 goto +6 <LBB1_3>
       4:       71 23 0c 00 00 00 00 00 r3 = *(u8 *)(r2 + 12)
       5:       71 24 0d 00 00 00 00 00 r4 = *(u8 *)(r2 + 13)
       6:       67 04 00 00 08 00 00 00 r4 <<= 8
       7:       4f 34 00 00 00 00 00 00 r4 |= r3
       8:       55 04 01 00 08 00 00 00 if r4 != 8 goto +1 <LBB1_3>
       9:       85 10 00 00 ff ff ff ff call -1

0000000000000050 <LBB1_3>:
      10:       b7 00 00 00 02 00 00 00 r0 = 2
      11:       95 00 00 00 00 00 00 00 exit
```

Programs




```
$ llvm-objdump --section .relxdp/example_prog -r example

example:        file format elf64-bpf

RELOCATION RECORDS FOR [xdp/example_prog]:
OFFSET           TYPE                     VALUE
0000000000000048 R_BPF_64_32              handle_ipv4
```

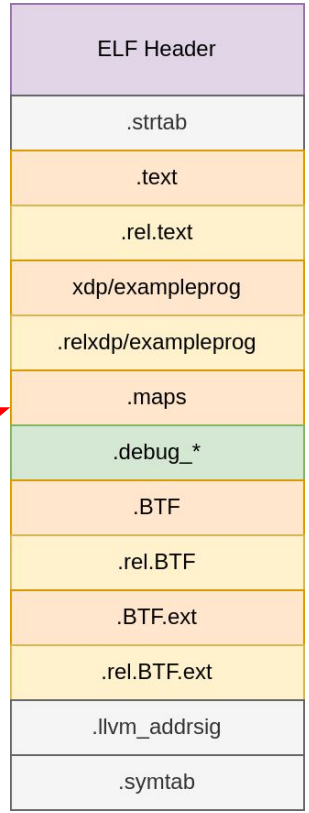
```
$ llvm-objdump --syms example

example:        file format elf64-bpf

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 example.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000118 l       .text  0000000000000000 LBB0_4
00000000000000d8 l       .text  0000000000000000 LBB0_3
0000000000000000 l    d  xdp/example_prog       0000000000000000 xdp/example_prog
0000000000000050 l       xdp/example_prog       0000000000000000 LBB1_3
0000000000000000 g     F .text  0000000000000128 handle_ipv4
0000000000000000 g     O .maps  0000000000000030 flow_map
0000000000000000 g     F xdp/example_prog       0000000000000060 example_prog
```

Programs

```
struct { /* anonymous struct used by BPF_PROG_LOAD
command */
        __u32           prog_type;
        __u32           insn_cnt;
        __aligned_u64   insns;
        __aligned_u64   license;
        __u32           log_level;
        __u32           log_size;
        __aligned_u64   log_buf;
        __u32           kern_version;
        __u32           prog_flags;
        char            prog_name[BPF_OBJ_NAME_LEN];
        __u32           prog_ifindex;
        __u32           expected_attach_type;
        __u32           prog_btf_fd;
        __u32           func_info_rec_size;
        __aligned_u64   func_info;
        __u32           func_info_cnt;
        __u32           line_info_rec_size;
        __aligned_u64   line_info;
        __u32           line_info_cnt;
        __u32           attach_btf_id;
        union {
            __u32       attach_prog_fd;
            __u32       attach_btf_obj_fd;
        };
        __u32           core_relo_cnt;
        __aligned_u64   fd_array;
        __aligned_u64   core_relos;
        __u32           core_relo_rec_size;
};
```

Maps

```c
#include [...]

struct flow_key
{
    __be32 saddr;
    __be32 daddr;
};

struct flow_value
{
    __u32 bytes;
    __u32 packets;
};

struct
{
    __uint(type, BPF_MAP_TYPE_HASH);
    __type(key, struct flow_key);
    __type(value, struct flow_value);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
    __uint(max_entries, 200);
    __uint(map_flags, BPF_F_NO_PREALLOC);
} flow_map SEC(".maps");
```



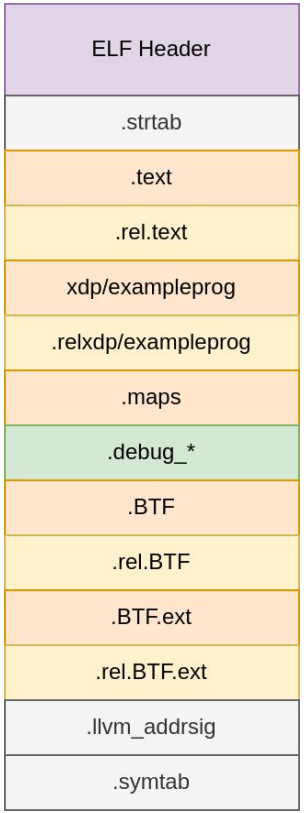
Maps

```
__attribute__((noinline)) int handle_ipv4(void *data, void *data_end) {
    struct iphdr *l3 = data;
    if (data + sizeof(l3) > data_end)
        return XDP_PASS;

    struct flow_key key = {
        .daddr = l3->daddr,
        .saddr = l3->saddr,
    };

    struct flow_value *value = bpf_map_lookup_elem(&flow_map, &key);
    if (!value) {
        struct flow_value new_value = {
            .bytes = sizeof(struct ethhdr) + l3->tot_len,
            .packets = 1,
        };

        bpf_map_update_elem(&flow_map, &key, &new_value, BPF_ANY);

        return XDP_PASS;
    }

    value->bytes += sizeof(struct ethhdr) + l3->tot_len;
    value->packets++;
    return XDP_PASS;
}
```

Maps

```
$ llvm-objdump --section=.text -d example
Disassembly of section .text:
0000000000000000 <handle_ipv4>:
       0:       bf 16 00 00 00 00 00 00 r6 = r1
       1:       07 01 00 00 08 00 00 00 r1 += 8
       2:       2d 21 20 00 00 00 00 00 if r1 > r2 goto +32 <LBB0_4>
       3:       61 61 0c 00 00 00 00 00 r1 = *(u32 *)(r6 + 12)
       4:       63 1a f8 ff 00 00 00 00 *(u32 *)(r10 - 8) = r1
       5:       61 61 10 00 00 00 00 00 r1 = *(u32 *)(r6 + 16)
       6:       63 1a fc ff 00 00 00 00 *(u32 *)(r10 - 4) = r1
       7:       bf a2 00 00 00 00 00 00 r2 = r10
       8:       07 02 00 00 f8 ff ff ff r2 += -8
       9:       18 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r1 = 0 ll
      11:       85 00 00 00 01 00 00 00 call 1
      12:       55 00 0e 00 00 00 00 00 if r0 != 0 goto +14 <LBB0_3>
      13:       69 61 02 00 00 00 00 00 r1 = *(u16 *)(r6 + 2)
      14:       b7 02 00 00 01 00 00 00 r2 = 1
      15:       63 2a f4 ff 00 00 00 00 *(u32 *)(r10 - 12) = r2
      16:       07 01 00 00 0e 00 00 00 r1 += 14
      17:       63 1a f0 ff 00 00 00 00 *(u32 *)(r10 - 16) = r1
      18:       bf a2 00 00 00 00 00 00 r2 = r10
      19:       07 02 00 00 f8 ff ff ff r2 += -8
      20:       bf a3 00 00 00 00 00 00 r3 = r10
      21:       07 03 00 00 f0 ff ff ff r3 += -16
      22:       18 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r1 = 0 ll
      24:       b7 04 00 00 00 00 00 00 r4 = 0
      25:       85 00 00 00 02 00 00 00 call 2
      26:       05 00 08 00 00 00 00 00 goto +8 <LBB0_4>

00000000000000d8 <LBB0_3>:
      27:       69 61 02 00 00 00 00 00 r1 = *(u16 *)(r6 + 2)
      28:       61 02 04 00 00 00 00 00 r2 = *(u32 *)(r0 + 4)
      29:       07 02 00 00 01 00 00 00 r2 += 1
      30:       63 20 04 00 00 00 00 00 *(u32 *)(r0 + 4) = r2
      31:       61 02 00 00 00 00 00 00 r2 = *(u32 *)(r0 + 0)
      32:       0f 21 00 00 00 00 00 00 r1 += r2
      33:       07 01 00 00 0e 00 00 00 r1 += 14
      34:       63 10 00 00 00 00 00 00 *(u32 *)(r0 + 0) = r1

0000000000000118 <LBB0_4>:
      35:       b7 00 00 00 02 00 00 00 r0 = 2
      36:       95 00 00 00 00 00 00 00 exit
```

Maps




```
$ llvm-objdump --section=.rel.text -r example

example:        file format elf64-bpf

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE                     VALUE
0000000000000048 R_BPF_64_64              flow_map
00000000000000b0 R_BPF_64_64              flow_map
```

```
$ llvm-objdump --syms example

example:        file format elf64-bpf

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 example.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000118 l       .text  0000000000000000 LBB0_4
00000000000000d8 l       .text  0000000000000000 LBB0_3
0000000000000000 l    d  xdp/example_prog       0000000000000000 xdp/example_prog
0000000000000050 l       xdp/example_prog       0000000000000000 LBB1_3
0000000000000000 g     F .text  0000000000000128 handle_ipv4
0000000000000000 g     O .maps  0000000000000030 flow_map
0000000000000000 g     F xdp/example_prog       0000000000000060 example_prog
```

Maps

```
struct { /* anonymous struct used by BPF_MAP_CREATE command */
   __u32   map_type;     /* one of enum bpf_map_type */
   __u32   key_size;     /* size of key in bytes */
   __u32   value_size;   /* size of value in bytes */
   __u32   max_entries;  /* max number of entries in a map */
   __u32   map_flags;    /* BPF_MAP_CREATE related
                          * flags defined above. */
   __u32   inner_map_fd; /* fd pointing to the inner map */
   __u32   numa_node;    /* numa node (effective only if
                          * BPF_F_NUMA_NODE is set).
                          */
   char    map_name[BPF_OBJ_NAME_LEN];
   __u32   map_ifindex;              /* ifindex of netdev to create on */
   __u32   btf_fd;                   /* fd pointing to a BTF type data */
   __u32   btf_key_type_id;          /* BTF type_id of the key */
   __u32   btf_value_type_id;        /* BTF type_id of the value */
   __u32   btf_vmlinux_value_type_id; /* BTF type_id of a kernel
                                       * struct stored as the map value *\

       /* [...] */
   };
```

Maps

```c
#include [...]

struct flow_key
{
   __be32 saddr;
   __be32 daddr;
};

struct flow_value
{
   __u32 bytes;
   __u32 packets;
};

struct
{
   __uint(type, BPF_MAP_TYPE_HASH);
   __type(key, struct flow_key);
   __type(value, struct flow_value);
   __uint(pinning, LIBBPF_PIN_BY_NAME);
   __uint(max_entries, 200);
   __uint(map_flags, BPF_F_NO_PREALLOC);
} flow_map SEC(".maps");
```

```
$ llvm-objdump -j .maps -s example

example:        file format elf64-bpf
Contents of section .maps:
 0000 00000000 00000000 00000000 00000000  ................
 0010 00000000 00000000 00000000 00000000  ................
 0020 00000000 00000000 00000000 00000000  ................
```

```c
/*
* Helper structure used by eBPF C program
* to describe BPF map attributes to libbpf loader
*/
struct bpf_map_def {
   unsigned int type;
   unsigned int key_size;
   unsigned int value_size;
   unsigned int max_entries;
   unsigned int map_flags;
} __attribute__((deprecated("use BTF-defined maps in
.maps section")));
```

BTF

- BPF Type Format
- Derived from DWARF debug symbols
- Smaller than DWARF
- Type validation in the verifier
  - Spinlocks (v5.1)
  - Callbacks
    - Map iterators (v5.13)
    - Timers (v5.15)
  - Kfuncs (v5.13)
- Flexible map attributes
- Debug information
- CO-RE

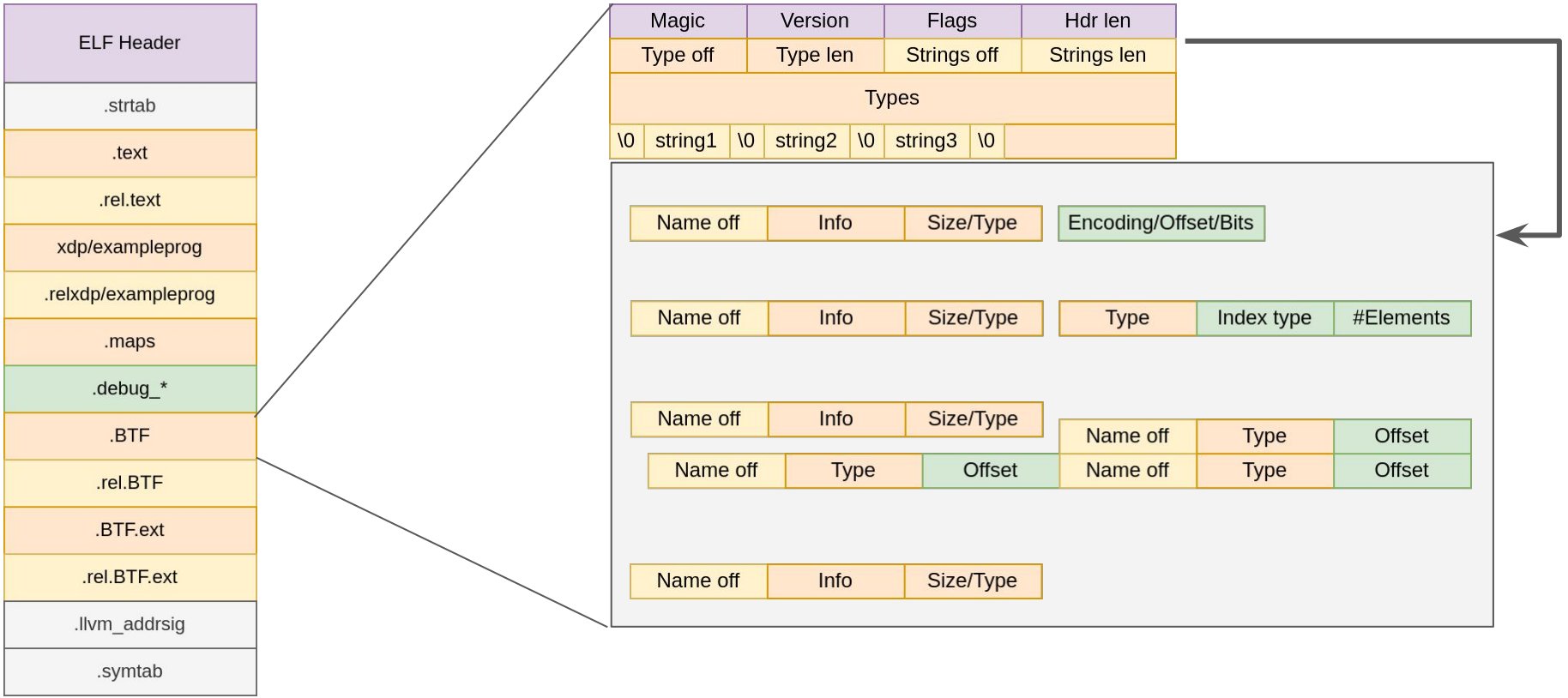
BTF

```
$  bpftool btf dump file example
[1] PTR '(anon)' type_id=3
[2] INT 'int' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
[3] ARRAY '(anon)' type_id=2 index_type_id=4 nr_elems=1
[4] INT '__ARRAY_SIZE_TYPE__' size=4 bits_offset=0 nr_bits=32
encoding=(none)
[5] PTR '(anon)' type_id=6
[6] STRUCT 'flow_key' size=8 vlen=2
        'saddr' type_id=7 bits_offset=0
        'daddr' type_id=7 bits_offset=32
[7] TYPEDEF '__be32' type_id=8
[8] TYPEDEF '__u32' type_id=9
[9] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32
encoding=(none)
[10] PTR '(anon)' type_id=11
[11] STRUCT 'flow_value' size=8 vlen=2
        'bytes' type_id=8 bits_offset=0
        'packets' type_id=8 bits_offset=32
[12] PTR '(anon)' type_id=13
[13] ARRAY '(anon)' type_id=2 index_type_id=4 nr_elems=200
[14] STRUCT '(anon)' size=48 vlen=6
        'type' type_id=1 bits_offset=0
        'key' type_id=5 bits_offset=64
        'value' type_id=10 bits_offset=128
        'pinning' type_id=1 bits_offset=192
        'max_entries' type_id=12 bits_offset=256
        'map_flags' type_id=1 bits_offset=320
[15] VAR 'flow_map' type_id=14, linkage=global
[16] PTR '(anon)' type_id=0
[17] FUNC_PROTO '(anon)' ret_type_id=2 vlen=2
        'data' type_id=16
        'data_end' type_id=16
[18] FUNC 'handle_ipv4' type_id=17 linkage=global
[19] PTR '(anon)' type_id=20
[20] STRUCT 'xdp_md' size=24 vlen=6
        'data' type_id=8 bits_offset=0
        'data_end' type_id=8 bits_offset=32
        'data_meta' type_id=8 bits_offset=64
        'ingress_ifindex' type_id=8 bits_offset=96
        'rx_queue_index' type_id=8 bits_offset=128
        'egress_ifindex' type_id=8 bits_offset=160
[21] FUNC_PROTO '(anon)' ret_type_id=2 vlen=1
        'ctx' type_id=19
[22] FUNC 'example_prog' type_id=21 linkage=global
[23] DATASEC '.maps' size=0 vlen=1
        type_id=15 offset=0 size=48 (VAR 'flow_map')
```

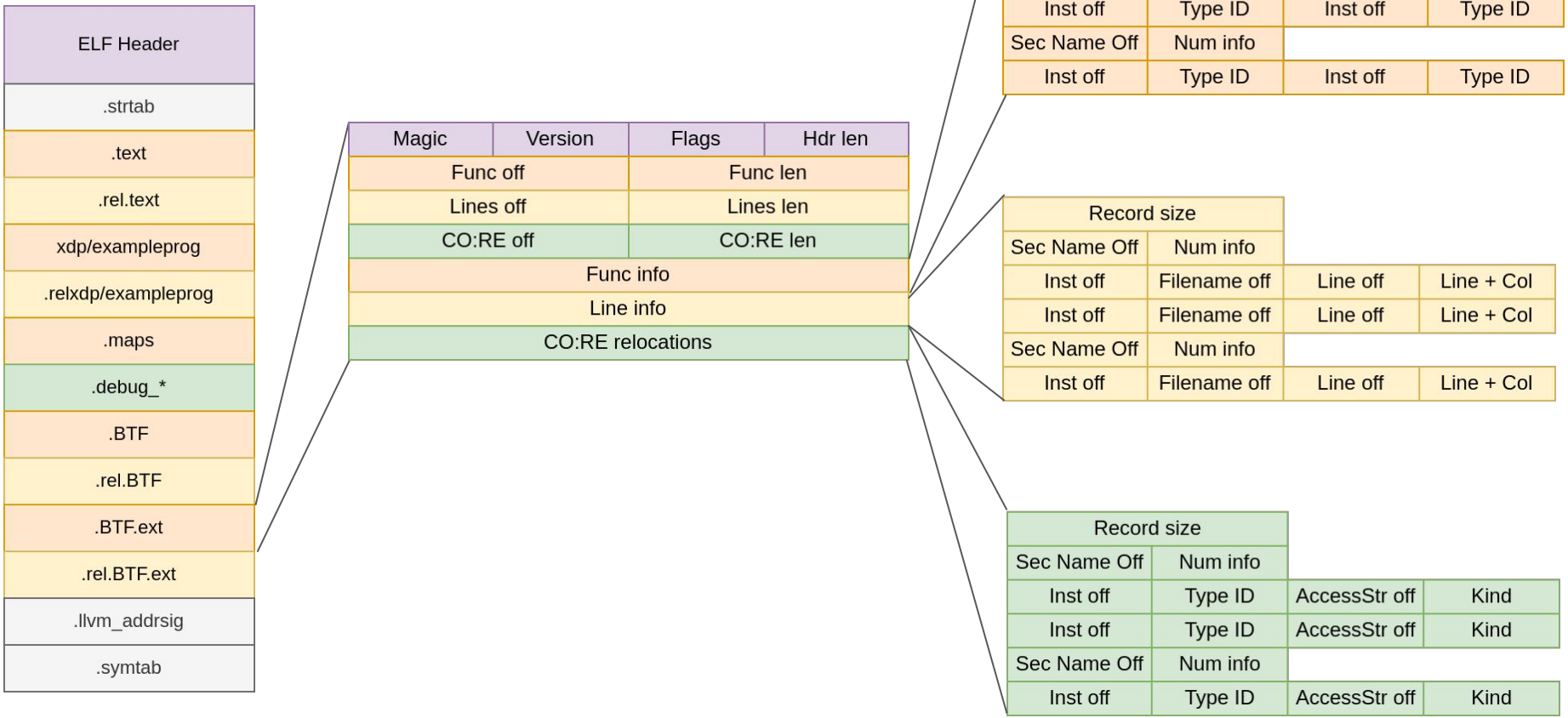
BTF

```
[23] DATASEC '.maps' size=0 vlen=1
    [15] VAR 'flow_map' type_id=14, linkage=global
        [14] STRUCT '(anon)' size=48 vlen=6
        'type' type_id=1 bits_offset=0
            [1] PTR '(anon)' type_id=3
                [3] ARRAY '(anon)' type_id=2 index_type_id=4 nr_elems=1 (__uint = 1)
        'key' type_id=5 bits_offset=64
            [5] PTR '(anon)' type_id=6
                [6] STRUCT 'flow_key' size=8 vlen=2
                'saddr' type_id=7 bits_offset=0
                    [7] TYPEDEF '__be32' type_id=8
                        [8] TYPEDEF '__u32' type_id=9
                            [9] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32 encoding=(none)
                'daddr' type_id=7 bits_offset=32
                    [7] TYPEDEF '__be32' type_id=8
        'value' type_id=10 bits_offset=128
            [10] PTR '(anon)' type_id=11
                [11] STRUCT 'flow_value' size=8 vlen=2
                'bytes' type_id=8 bits_offset=0
                    [8] TYPEDEF '__u32' type_id=9
                        [9] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32 encoding=(none)
                'packets' type_id=8 bits_offset=32
                    [8] TYPEDEF '__u32' type_id=9
                        [9] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32 encoding=(none)
        [...]
```

BTF


ELF Header
.strtab
.text
.rel.text
xdp/exampleprog
.relxdp/exampleprog
.maps
.debug_*
.BTF
.rel.BTF
.BTF.ext
.rel.BTF.ext
.llvm_addrsig
.symtab
Magic
Version
Flags
Hdr len
Type off
Type len
Strings off
Strings len
Types
\0
string1
\0
string2
\0
string3
\0
Name off
Info
Size/Type
Encoding/Offset/Bits
Name off
Info
Size/Type
Type
Index type
#Elements
Name off
Info
Size/Type
Name off
Type
Offset
Name off
Type
Offset
Name off
Type
Offset
Name off
Info
Size/Type

BTF


ELF Header
.strtab
.text
.rel.text
xdp/exampleprog
.relxdp/exampleprog
.maps
.debug_*
.BTF
.rel.BTF
.BTF.ext
.rel.BTF.ext
.llvm_addrsig
.symtab
Magic
Version
Flags
Hdr len
Func off
Func len
Lines off
Lines len
CO:RE off
CO:RE len
Func info
Line info
CO:RE relocations
Record size
Sec Name Off
Num info
Inst off
Type ID
Inst off
Type ID
Inst off
Type ID
Inst off
Type ID
Sec Name Off
Num info
Inst off
Type ID
Inst off
Type ID
Record size
Sec Name Off
Num info
Inst off
Filename off
Line off
Line + Col
Inst off
Filename off
Line off
Line + Col
Sec Name Off
Num info
Inst off
Filename off
Line off
Line + Col
Record size
Sec Name Off
Num info
Inst off
Type ID
AccessStr off
Kind
Inst off
Type ID
AccessStr off
Kind
Sec Name Off
Num info
Inst off
Type ID
AccessStr off
Kind

BTF

```
struct { /* anonymous struct for BPF_BTF_LOAD */
    __aligned_u64   btf;
    __aligned_u64   btf_log_buf;
    __u32           btf_size;
    __u32           btf_log_size;
    __u32           btf_log_level;
};
```

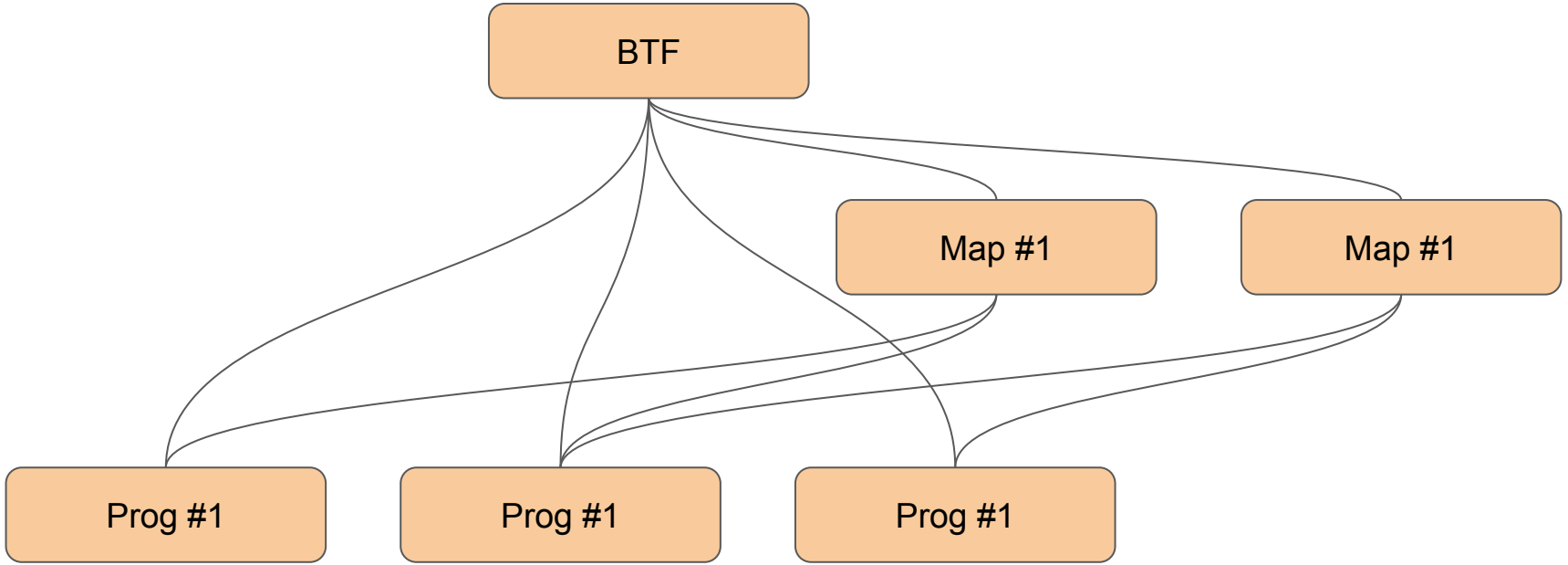
BTF

```
struct { /* anonymous struct used by BPF_MAP_CREATE command */
   __u32   map_type;     /* one of enum bpf_map_type */
   __u32   key_size;     /* size of key in bytes */
   __u32   value_size;   /* size of value in bytes */
   __u32   max_entries;  /* max number of entries in a map */
   __u32   map_flags;    /* BPF_MAP_CREATE related
                          * flags defined above. */
   __u32   inner_map_fd; /* fd pointing to the inner map */
   __u32   numa_node;    /* numa node (effective only if
                          * BPF_F_NUMA_NODE is set).
                          */
   char    map_name[BPF_OBJ_NAME_LEN];
   __u32   map_ifindex;              /* ifindex of netdev to create on */
   __u32   btf_fd;                   /* fd pointing to a BTF type data */
   __u32   btf_key_type_id;          /* BTF type_id of the key */
   __u32   btf_value_type_id;        /* BTF type_id of the value */
   __u32   btf_vmlinux_value_type_id; /* BTF type_id of a kernel
                                       * struct stored as the map value *\

       /* [...] */
   };
```

BTF

```
struct { /* anonymous struct used by BPF_PROG_LOAD command */
        __u32           prog_type;
        __u32           insn_cnt;
        __aligned_u64   insns;
        __aligned_u64   license;
        __u32           log_level;
        __u32           log_size;
        __aligned_u64   log_buf;
        __u32           kern_version;
        __u32           prog_flags;
        char            prog_name[BPF_OBJ_NAME_LEN];
        __u32           prog_ifindex;
        __u32           expected_attach_type;
        __u32           prog_btf_fd;
        __u32           func_info_rec_size;
        __aligned_u64   func_info;
        __u32           func_info_cnt;
        __u32           line_info_rec_size;
        __aligned_u64   line_info;
        __u32           line_info_cnt;
        __u32           attach_btf_id;
        union {
                __u32   attach_prog_fd;
                __u32   attach_btf_obj_fd;
        };
        __u32           core_relo_cnt;
        __aligned_u64   fd_array;
        __aligned_u64   core_relos;
        __u32           core_relo_rec_size;
};
```

Object load order



CO-RE

- Compile Once - Run Everywhere
- All you need to use it - https://nakryiko.com/posts/bpf-core-reference-guide
- LibBPF macros
  - BPF_CORE_READ{_STR,_INTO,_STR_INTO}()
  - bpf_core_field_{exists,size,offset}()
  - bpf_core_type_size()
  - bpf_core_enum_value{_exists}()
- Compiler builtins
  - __builtin_preserve_field_info
  - __builtin_preserve_type_info
  - __builtin_preserve_enum_value
  - __builtin_preserve_access_index
  - __builtin_btf_type_id

CO-RE

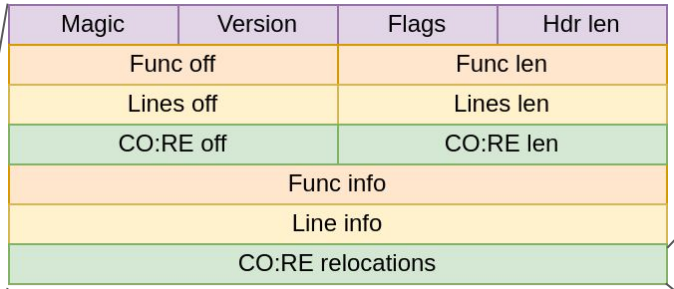
```
#include "vmlinux.h"
#include "bpf_core_read.h"
#include "bpf_tracing.h"

struct
{
   __uint(type, BPF_MAP_TYPE_PERF_EVENT_ARRAY);
   __type(key, int);
   __type(value, __u32);
   __uint(max_entries, 1024);
} closed_sockets SEC(".maps");

SEC("kprobe/tcp_close")
int BPF_KPROBE(record_close, struct sock *sk, long timeout)
{

   u64 cookie = BPF_CORE_READ(sk, __sk_common.skc_cookie.counter);

   bpf_perf_event_output(ctx, &closed_sockets, BPF_F_CURRENT_CPU, &cookie, sizeof(u64));
   return 0;
}
```

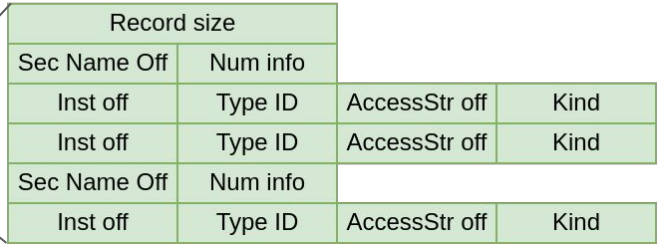
CO-RE

```
 0: MovReg dst: r6 src: r1
 1: LdXMemDW dst: r3 src: r6 off: 112 imm: 0
 2: MovImm dst: r1 imm: 104
 3: AddReg dst: r3 src: r1
 4: MovReg dst: r7 src: rfp
 5: AddImm dst: r7 imm: -8
 6: MovReg dst: r1 src: r7
 7: MovImm dst: r2 imm: 8
 8: Call FnProbeReadKernel
 9: MovReg dst: r1 src: r6
10: LoadMapPtr dst: r2 fd: 0 <closed_sockets>
12: LdImmDW dst: r3 imm: 4294967295
14: MovReg dst: r4 src: r7
15: MovImm dst: r5 imm: 8
16: Call FnPerfEventOutput
17: MovImm dst: r0 imm: 0
18: Exit
```

```
int BPF_KPROBE(record_close, struct sock *sk, long
timeout) {

   u64 cookie = BPF_CORE_READ(
      sk,
     __sk_common.skc_cookie.counter
   );
   // u64 cookie = {
   //   u64 __r;
   //   bpf_probe_read_kernel(
   //     __r,
   //     sizeof(sk->__sk_common.skc_cookie.counter),
   //     sk + __builtin_preserve_access_index(
   //         sk->__sk_common.skc_cookie.counter
   //     ));
   //   __r;
   // };

   bpf_perf_event_output([...]);
   return 0;
}
```

BTF





Instruction offset: 16 bytes, Inst # 2
Type ID: 18
Accessor String: 0:1:17:0
Kind: 0 (byte_offset)

sk->__sk_common.skc_cookie.counter

0	*sk
1	.__sk_common
17	.skc_cookie
0	.counter

Q/A

Thank you