# (e)BPF, perf, tracing

28.02.2019
Maciej Borzęcki
CEHUG Łódź

# This talk is about

- Berkeley Packet Filter
- extended BPF
- SECCOMP
- Tracing
- Having fun while reading code

# BPF

# BPF design goals

- It must be protocol independent. The kernel should not have to be modified to add new protocol support
- It must be general. The instruction set should be rich enough to handle unforeseen uses
- Packet data references should be minimized
- Decoding an instruction should consist of a single C switch statement
- The abstract machine registers should reside in physical registers

McCanne, Steven; Jacobson, Van (1992-12-19). "The BSD Packet Filter: A New Architecture for User-level Packet Capture" (PDF).

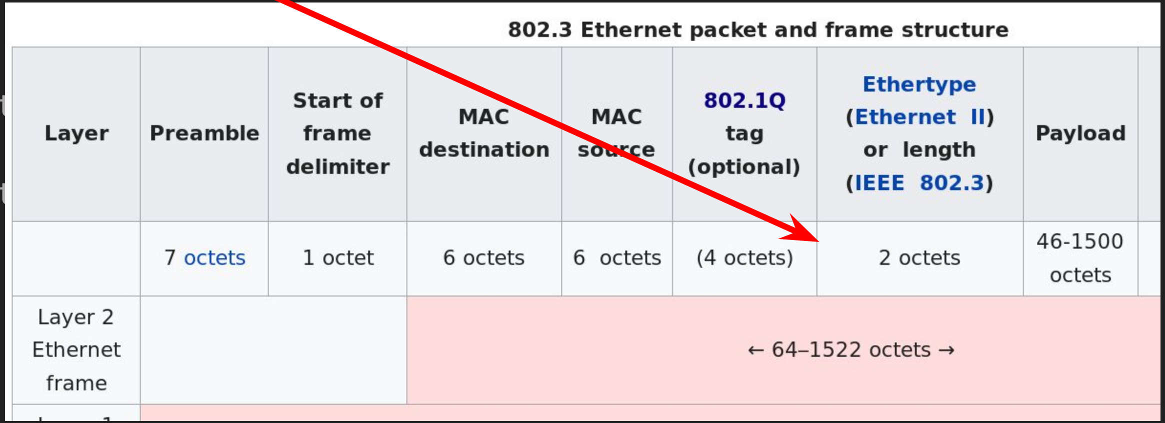
# Demo - tcpdump

# BPF

```
$ tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800        jt 2 jf 12  <-- EtherType 0x0800 IPv4
(002) ldb      [23]
(003) jeq      #0x6          jt 4 jf 12  <-- IPv4 Protocol - TCP
(004) ldh      [20]
(005) jset     #0x1fff       jt 12   jf 6
(006) ldxb     4*([14]&0xf)                 <-- Internet Header Length
(007) ldh      [x + 14]                     <-- TCP source port
(008) jeq      #0x50         jt 11   jf 9
(009) ldh      [x + 16]                     <-- TCP destination port
(010) jeq      #0x50         jt 11   jf 12
(011) ret      #262144                      <-- accept 256k of packet data
(012) ret      #0                           <-- ignore
```

# BPF

```
$ tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800          jt 2 jf 12  <-- EtherType 0x0800 IPv4
(002) ldb      [23]
(003) jeq      #0x6            jt 4 jf 12  <-- IPv4 Protocol - TCP
(004) ldh      [20]
(005) jset     #0x1fff         jt 12   jf 6
(006) ldxb     4*([14]&0xf)
(007) ldh      [x + 14]
(008) jeq      #0x50           jt
(009) ldh      [x + 16]
(010) jeq      #0x50           jt
(011) ret      #262144
(012) ret      #0
```

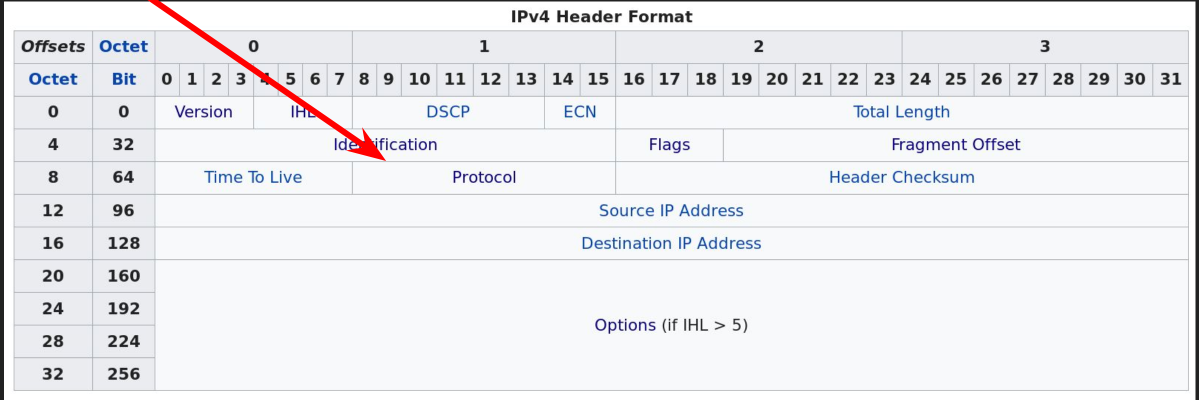


**802.3 Ethernet packet and frame structure**

| Layer | Preamble | Start of frame delimiter | MAC destination | MAC source | 802.1Q tag (optional) | Ethertype (Ethernet II) or length (IEEE 802.3) | Payload |
|---|---|---|---|---|---|---|---|
| | 7 octets | 1 octet | 6 octets | 6 octets | (4 octets) | 2 octets | 46-1500 octets |
| Layer 2 Ethernet frame | | | ← 64–1522 octets → | | | | |

# BPF

```
$ tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800          jt 2 jf 12  <-- EtherType 0x0800 IPv4
(002) ldb      [23]
(003) jeq      #0x6            jt 4 jf 12  <-- IPv4 Protocol - TCP
(004) ldh      [20]
(005) jset     #0x1fff         jt 12   jf 6
(006) ldxb     4*([14]&0xf)                   <-- Internet Header Length
(007) ldh      [x + 14]                       <-- TCP source port
(008) jeq      #0x50
(009) ldh      [x + 16]
(010) jeq      #0x50
(011) ret      #262144
(012) ret      #0
```

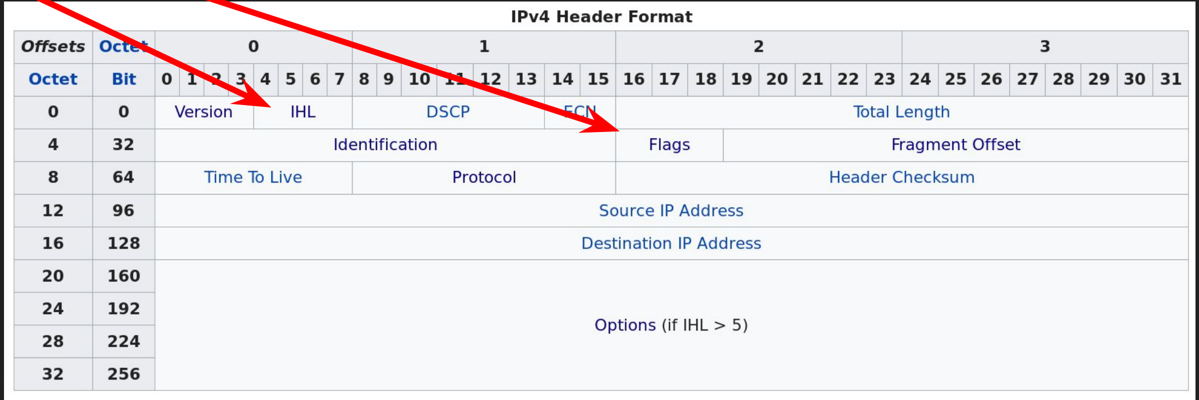


IPv4 Header Format

| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Version | | | | IHL | | | | DSCP | | | | | | ECN | | Total Length | | | | | | | | | | | | | | | |
| 4 | 32 | Identification | | | | | | | | | | | | | | | | Flags | | | Fragment Offset | | | | | | | | | | | | |
| 8 | 64 | Time To Live | | | | | | | | Protocol | | | | | | | | Header Checksum | | | | | | | | | | | | | | | |
| 12 | 96 | Source IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 128 | Destination IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | 160 | Options (if IHL > 5) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | 192 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | 224 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | 256 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

# BPF

```
$ tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800         jt 2 jf 12  <-- EtherType 0x0800 IPv4
(002) ldb      [23]
(003) jeq      #0x6           jt 4 jf 12  <-- IPv4 Protocol - TCP
(004) ldh      [20]
(005) jset     #0x1fff        jt 12   jf 6
(006) ldxb     4*([14]&0xf)                  <-- Internet Header Length
(007) ldh      [x + 14]                      <-- TCP source port
(008) jeq      #0x50
(009) ldh      [x + 16]
(010) jeq      #0x50
(011) ret      #262144
(012) ret      #0
```

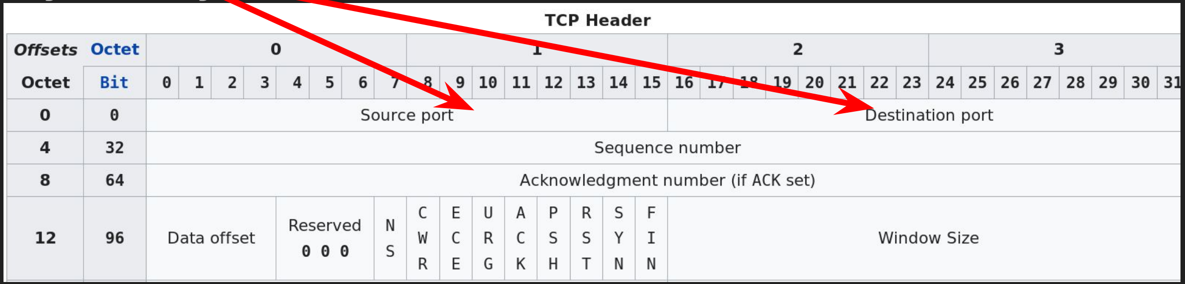


IPv4 Header Format

| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Version | | | | IHL | | | | DSCP | | | | | | ECN | | Total Length | | | | | | | | | | | | | | | |
| 4 | 32 | Identification | | | | | | | | | | | | | | | | Flags | | | Fragment Offset | | | | | | | | | | | | |
| 8 | 64 | Time To Live | | | | | | | | Protocol | | | | | | | | Header Checksum | | | | | | | | | | | | | | | |
| 12 | 96 | Source IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 128 | Destination IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | 160 | Options (if IHL > 5) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | 192 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | 224 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | 256 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

# BPF

```
$ tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800         jt 2 jf 12  <-- EtherType 0x0800 IPv4
(002) ldb      [23]
(003) jeq      #0x6           jt 4 jf 12  <-- IPv4 Protocol - TCP
(004) ldh      [20]
(005) jset     #0x1fff        jt 12   jf 6
(006) ldxb     4*([14]&0xf)                  <-- Internet Header Length
(007) ldh      [x + 14]                      <-- TCP source port
(008) jeq      #0x50          jt 11   jf 9
(009) ldh      [x + 16]                      <-- TCP destination port
(010) jeq      #0x50          jt 11   jf 12
(011) ret      #262144
(012) ret      #0
```



TCP Header

| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Source port | | | | | | | | | | | | | | | | Destination port | | | | | | | | | | | | | | | |
| 4 | 32 | Sequence number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 64 | Acknowledgment number (if ACK set) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 96 | Data offset | | | | Reserved<br>0 0 0 | | | NS | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size | | | | | | | | | | | | | | | |

# BPF abstract machine

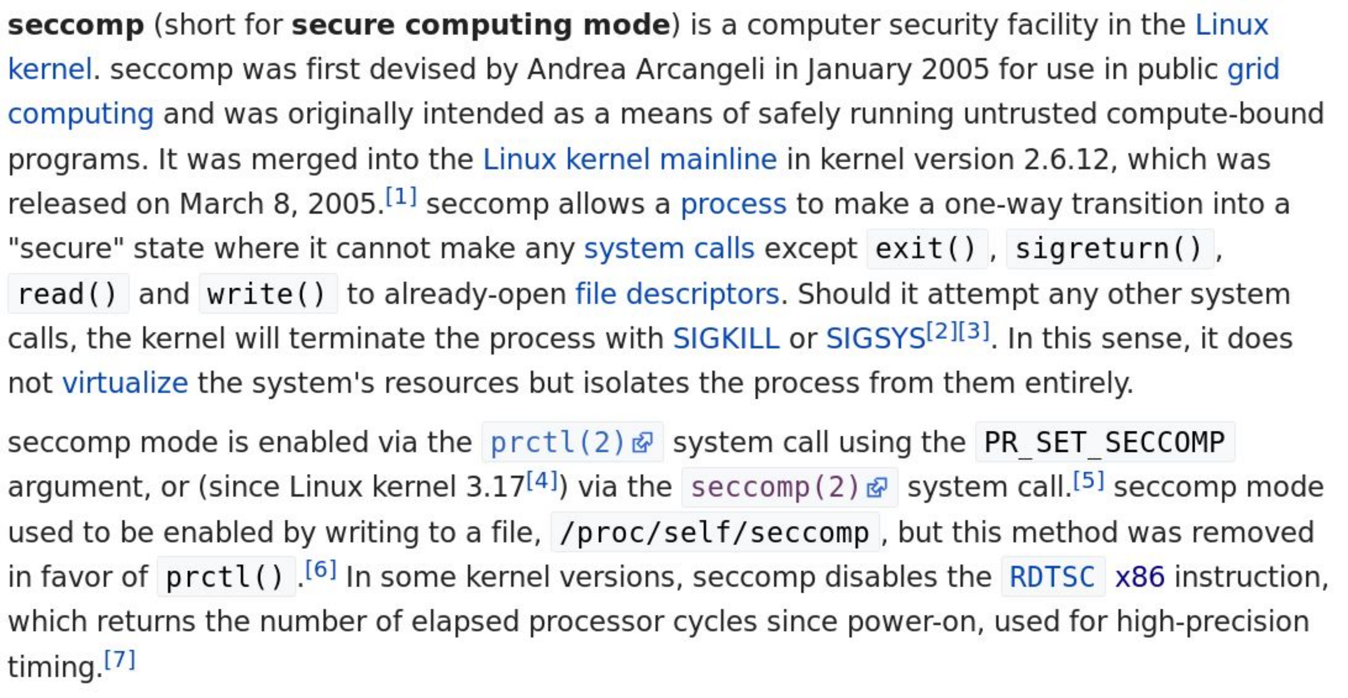
- 2 32bit registers, A (accumulator), X (index)
- 16 elements stack, each 32bit
- Basic instruction set
    - load/store (ld*, ld*, st*, stx*)
    - ALU on A, X or K (constant)
    - Jump (conditional, unconditional)
    - opcode:16b jt:8b jf:8b k:32b
- Forward jumps only

# BPF

```
$ tcpdump -d 'ip6 and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x86dd        jt 2 jf 9  <-- EtherType 0x86dd IPv6
(002) ldb      [20]
(003) jeq      #0x6           jt 4 jf 9  <-- IPv6 Protocol - TCP
(004) ldh      [54]
(005) jeq      #0x50          jt 8 jf 6  <-- TCP source port
(006) ldh      [56]
(007) jeq      #0x50          jt 8 jf 9  <-- TCP destination port
(008) ret      #262144
(009) ret      #0
```

# BPF use case - SECCOMP

# SECCOMP



**seccomp** (short for **secure computing mode**) is a computer security facility in the Linux kernel. seccomp was first devised by Andrea Arcangeli in January 2005 for use in public grid computing and was originally intended as a means of safely running untrusted compute-bound programs. It was merged into the Linux kernel mainline in kernel version 2.6.12, which was released on March 8, 2005.[1] seccomp allows a process to make a one-way transition into a "secure" state where it cannot make any system calls except `exit()`, `sigreturn()`, `read()` and `write()` to already-open file descriptors. Should it attempt any other system calls, the kernel will terminate the process with SIGKILL or SIGSYS[2][3]. In this sense, it does not virtualize the system's resources but isolates the process from them entirely.

seccomp mode is enabled via the `prctl(2)` system call using the `PR_SET_SECCOMP` argument, or (since Linux kernel 3.17[4]) via the `seccomp(2)` system call.[5] seccomp mode used to be enabled by writing to a file, `/proc/self/seccomp`, but this method was removed in favor of `prctl()`.[6] In some kernel versions, seccomp disables the `RDTSC` x86 instruction, which returns the number of elapsed processor cycles since power-on, used for high-precision timing.[7]

# seccomp

- seccomp() system call
- Control which system calls can be executed
- One way street
- Once enabled cannot be dropped
- Preserved across fork() and clone()
- Preserved across execve()
- Filters are BPF programs

# seccomp

- seccomp() system call
- Control which system calls can be executed
- One way street
- Once enabled cannot be dropped
- Preserved across fork() and clone()
- Preserved across execve()
- Filters are BPF programs

# seccomp - libseccomp & Go

```
// default action - ALLOW
filter, err := seccomp.NewFilter(seccomp.ActAllow)

// find syscall number - read()
sysNum, err := seccomp.GetSyscallFromName("read")

// first arg == 3
cond, err := seccomp.MakeCondition(0, seccomp.CompareEqual, 3)

// read(3, ..) = -EPERM
err = filter.AddRuleConditional(sysNum, seccomp.ActErrno.SetReturnCode(1),
       []seccomp.ScmpCondition{cond})
```

# seccomp - BPF program pseudocode

```
# filter for arch x86_64 (3221225534 = 0xc000003e)
if ($arch == 3221225534)
  # filter for syscall "read" (0) [priority: 65533]
  if ($syscall == 0)
      if ($a0.hi32 == 0)
      if ($a0.lo32 == 3)
      action ERRNO(1);
  # default action
  action ALLOW;
# invalid architecture action
action KILL;
```

# seccomp - BPF program disassembly

```
 line  OP   JT   JF   K
=================================
 0000: 0x20 0x00 0x00 0x00000004   ld  $data[4]
 0001: 0x15 0x00 0x0a 0xc000003e   jeq 3221225534 true:0002 false:0012
 0002: 0x20 0x00 0x00 0x00000000   ld  $data[0]
 0003: 0x35 0x00 0x01 0x40000000   jge 1073741824 true:0004 false:0005
 0004: 0x15 0x00 0x07 0xffffffff   jeq 4294967295 true:0005 false:0012
 0005: 0x15 0x00 0x04 0x00000000   jeq 0     true:0006 false:0010
 0006: 0x20 0x00 0x00 0x00000014   ld  $data[20]
 0007: 0x15 0x00 0x02 0x00000000   jeq 0     true:0008 false:0010
 0008: 0x20 0x00 0x00 0x00000010   ld  $data[16]
 0009: 0x15 0x01 0x00 0x00000003   jeq 3     true:0011 false:0010
 0010: 0x06 0x00 0x00 0x7fff0000   ret ALLOW
 0011: 0x06 0x00 0x00 0x00050001   ret ERRNO(1)
 0012: 0x06 0x00 0x00 0x00000000   ret KILL
```

# Demo

# seccomp - BPF demo

```
int main(int argc, char *argv[]) {
  ...
  char *bpf_prog = read_prog(argv[1], &bpf_prog_size);

  prctl(PR_SET_NO_NEW_PRIVS, 1);
  struct sock_fprog prog = {
      .len = bpf_prog_size / sizeof(struct sock_filter),
      .filter = (struct sock_filter *)bpf_prog,
  };
  seccomp(SECCOMP_SET_MODE_FILTER, SECCOMP_FILTER_FLAG_LOG, &prog);
  /* filter attached */
  int fd = open("/proc/self/exe", 0);
  assert(fd == 3);
  char buf;
  /* THIS SHOULD FAIL */
  ssize_t rd = read(fd, &buf, 1);
  assert(rd == -1);
  perror("read() failed");
  ...
}
```

# seccomp & BPF - kernel side

- kernel/seccomp.c: seccomp_prepare_filter() ← when setting up the filter
- net/core/filter.c: bpf_check_classic()
- kernel/seccomp.c: seccomp_run_filters() ← when the filter is applied
- BPF_PROG_RUN(), returns seccomp action

# seccomp & BPF - kernel side

```
static u32 seccomp_run_filters(const struct seccomp_data *sd,
                        struct seccomp_filter **match)
{
    u32 ret = SECCOMP_RET_ALLOW;
    struct seccomp_filter *f =
             READ_ONCE(current->seccomp.filter);
    ...
    /*
     * All filters in the list are evaluated and the lowest BPF return
     * value always takes priority (ignoring the DATA).
     */
    for (; f; f = f->prev) {
       u32 cur_ret = BPF_PROG_RUN(f->prog, sd);

       if (ACTION_ONLY(cur_ret) < ACTION_ONLY(ret)) {
             ret = cur_ret;
             *match = f;
       }
    }
    return ret;
}
```

```
struct seccomp_data {
    int nr;
    __u32 arch;
    __u64 instruction_pointer;
    __u64 args[6];
};
```

# seccomp - BPF program disassembly

```
 line  OP   JT   JF   K
=================================
 0000: 0x20 0x00 0x00 0x00000004   ld  $data[4]
 0001: 0x15 0x00 0x0a 0xc000003e   jeq 3221225534 true:0002 false:0012
 0002: 0x20 0x00 0x00 0x00000000   ld  $data[0]
 0003: 0x35 0x00 0x01 0x40000000   jge 1073741824 true:0004 false:0005
 0004: 0x15 0x00 0x07 0xffffffff   jeq 4294967295 true:0005 false:0012
 0005: 0x15 0x00 0x04 0x00000000   jeq 0     true:0006 false:0010
 0006: 0x20 0x00 0x00 0x00000014   ld  $data[20]
 0007: 0x15 0x00 0x02 0x00000000   jeq 0     true:0008 false:0010
 0008: 0x20 0x00 0x00 0x00000010   ld  $data[16]
 0009: 0x15 0x01 0x00 0x00000003   jeq 3     true:0011 false:0010
 0010: 0x06 0x00 0x00 0x7fff0000   ret ALLOW
 0011: 0x06 0x00 0x00 0x00050001   ret ERRNO(1)
 0012: 0x06 0x00 0x00 0x00000000   ret KILL
```

```
struct seccomp_data {
    int nr;
    __u32 arch;
    __u64 instruction_pointer;
    __u64 args[6];
};
```

# eBPF

# eBPF

- 64bit registers
- r0 - r10
- C compatible ABI
  - Parameters passed in r1 - r5 (r1 carries call context)
  - r6 - r9 callee saved
  - return value in r0
  - r10 frame pointer
- No overhead calling to/from C code
- JITed
- Kernel provides helpers
- Verifier

# eBPF - kernel helper

```
/* prototype for BPF verifier */
const struct bpf_func_proto bpf_get_prandom_u32_proto = {
  .func         = bpf_user_rnd_u32,
  .gpl_only     = false,
  .ret_type     = RET_INTEGER,
};
/* for the actual call from eBPF */
BPF_CALL_0(bpf_user_rnd_u32)
{
  /* Should someone ever have the rather unwise idea to use some
   * of the registers passed into this function, then note that
   * this function is called from native eBPF and classic-to-eBPF
   * transformations. Register assignments from both sides are
   * different, f.e. classic always sets fn(ctx, A, X) here.
   */
  struct rnd_state *state;
  u32 res;
  state = &get_cpu_var(bpf_user_rnd_state);
  res = prandom_u32_state(state);
  put_cpu_var(bpf_user_rnd_state);
  return res;
}
```

# eBPF in the kernel

- cgroups
  - Firewall
  - Device (former device cgroup)
- tc - traffic control
  - classifier
- xtables
- tracing, events
- More peculiar use cases
  - PPP
  - ISDN
  - LIRC

# Demo - tc-bpf

# tc-bpf

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

#ifndef __section
# define __section(x)  __attribute__((section(x), used))
#endif

__section("classifier") int cls_main(struct __sk_buff *skb)
{
	return -1;
}

char __license[] __section("license") = "GPL";
```

```
$ clang -O2 -emit-llvm -c hello.c -o - | llc -march=bpf -filetype=obj -o hello.o
$ llvm-objdump -S -no-show-raw-insn hello.o
```

# Demo - bpftool

# systemd cgroup packet filter

- Installed for cgroup by systemd
- When service IP address white/black list is used

```
[Unit]
Description=Journal Service
...
[Service]
ExecStart=/usr/lib/systemd/systemd-journald
...
SystemCallFilter=@system-service    ← SECCOMP
SystemCallErrorNumber=EPERM
SystemCallArchitectures=native
...
IPAddressDeny=any                   ← cgroup packet filter
```

# systemd cgroup packet filter

```
$ bpftool prog list

$ bpftool cgroup tree

$ bpftool prog dump xlated id <id>

$ bpftool prog dump jited id <id>
```

# systemd cgroup packet filter - kernel side

```
int __cgroup_bpf_run_filter_skb(struct sock *sk,
                   struct sk_buff *skb,
                   enum bpf_attach_type type)
{
    int ret;

    if (!sk || !sk_fullsock(sk))
       return 0;
    if (sk->sk_family != AF_INET && sk->sk_family != AF_INET6)
       return 0;
    ...
    /* compute pointers for the bpf prog */
    bpf_compute_and_save_data_end(skb, &saved_data_end);
    ret = BPF_PROG_RUN_ARRAY(cgrp->bpf.effective[type], skb,
                    bpf_prog_run_save_cb);
    bpf_restore_data_end(skb, saved_data_end);
    __skb_pull(skb, offset);
    skb->sk = save_sk;
    return ret == 1 ? 0 : -EPERM;
}
```

# Tracing

# Tracing

- functrace
- Trace events
- Kprobes
- Uprobes
- USDT

# Tracing subsystem

- /sys/debug/kernel/tracing
- Privileged access only
- See Documentation/trace/

# perf

- Swiss Army knife of all tracing/profiling
- Profiling

```
$ perf stat
```

```
$ perf record/report
```

- **Tracing**

```
$ perf trace
```

```
$ perf record -e …
```

```
$ perf probe
```

# Demo - SECCOMP

# perf trace

```
$ sudo perf trace ./demo prog.bpf
     ? (        ): demo/17292  ... [continued]: execve()) = 0
     2.480 ( 0.139 ms): demo/17292 brk(                     ) = 0x1d39000
     ...
     6.878 ( 0.038 ms): demo/17292 seccomp(op: FILTER, flags: 0x2, uargs: 0x7ffd734af360) = 0
     12.306 ( 0.017 ms): demo/17292 openat(dfd: CWD, filename: 0x4020c0) = 3
     12.306 ( 0.042 ms): demo/17292  ... [continued]: read()) = -1 EPERM Operation not permitted
     ...
```

# kprobe - perf probe & trace

```
$ sudo perf probe -v seccomp_run_filters
Added new event:
  probe:seccomp_run_filters (on seccomp_run_filters)

You can now use it in all perf tools, such as:

      perf record -e probe:seccomp_run_filters -aR sleep 1

$ sudo perf trace ./demo prog.bpf
      ? (        ): demo/17292  ... [continued]: execve()) = 0
      2.480 ( 0.139 ms): demo/17292 brk(                     ) = 0x1d39000
      ...
      6.878 ( 0.038 ms): demo/17292 seccomp(op: FILTER, flags: 0x2, uargs: 0x7ffd734af360) = 0
      12.306 ( 0.017 ms): demo/17292 openat(dfd: CWD, filename: 0x4020c0) = 3
      12.306 ( 0.042 ms): demo/17292  ... [continued]: read()) = -1 EPERM Operation not permitted
      ...
```

# kretprobe - perf probe & trace

```
$ sudo perf probe -v 'seccomp_run_filters%return' '$retval:x32'
You can now use it in all perf tools, such as:

     perf record -e probe:seccomp_run_filters__return -aR sleep 1

$ sudo perf trace ./demo prog.bpf
     0.000 demo/17473 syscalls:sys_enter_read:fd: 3</home/guest/seccomp/prog.bpf>, buf: 0xcc4a80,..
     1.004 demo/17473 syscalls:sys_enter_seccomp:op: FILTER, flags: 0x2, uargs: 0x7fffc57e0e40
     1.035 demo/17473 syscalls:sys_exit_seccomp:0x0
     ...
     1.060 demo/17473 probe:seccomp_run_filters__return:(ffffffff8617e850 <- ffffffff8617ef4e)
                           arg1=0x50001
     1.090 demo/17473 syscalls:sys_exit_read:0xffffffffffffffff
     ...
```

# Perf record

```
$ sudo perf record -e 'probe:seccomp_run_filters,syscalls:*' demo ./prog.bpf

$ sudo perf script
```

# Demo - simple tracing with event trace

# Event trace

```
$ sudo perf trace -e 'tcp:*' demo http://ifconfig.co/json > /dev/null

$ sudo perf list |grep skb

$ sudo perf trace -e 'tcp:*,skb:*' demo http://ifconfig.co/json > /dev/null

$ sudo perf trace -e 'tcp:*,skb:*,udp:*' ./a.out http://ifconfig.co/json > /dev/null
```

# Event trace

- Generic event trace
- Manually defined trace points
- Manually invoked
- Helper macro TRACE_EVENT(....)
- Invoked as trace_<name>()

- Example:
  - include/trace/events/skb.h, TRACE_EVENT(kfree_skb)
  - net/core/skbuff.c, trace_kfree_skb()

# Demo - deeper tracing with uprobes & events

# uprobe

- Userspace probes
- Dynamically patched userspace code
- No modifications needed

```
$ perf probe -x <binary> <symbol> <format>

$ sudo perf probe -x /usr/lib/libsoup-2.4.so.1.8.0 soup_message_body_new

$ sudo perf probe -x /usr/lib/libsoup-2.4.so.1.8.0 soup_message_new

$ sudo perf probe -x /usr/lib/libgobject-2.0.so g_object_unref

$ sudo perf probe -x /usr/lib/libsoup-2.4.so.1.8.0 \

    'soup_message_body_new%return' '$retval:u32'
```

# uprobe - simple tracing

```
$ sudo perf trace \
      -e 'probe_libsoup:soup_session_send_message,probe_libsoup:soup_session_send_message__return' \
      ./demo http://ifconfig.co/json > /dev/null

$ sudo perf trace -g -e … ./demo http://ifconfig.co/json > /dev/null
```

# Demo - Userspace Statically Defined Trace

# USDT - Statically Defined Trace

```
#include <sys/sdt.h>
...
void foo(size_t cnt) {
  DTRACE_PROBE1(demo, foo, cnt);
  if (cnt % 2 == 0) printf("foo\n");
  else printf("oof\n");
}
void bar(size_t cnt) {
  DTRACE_PROBE1(demo, bar, cnt);
  if (cnt % 2 == 0) printf("bar\n");
  else printf("rab\n");
}
int main(int argc, char *argv[]) {
  int cnt = 0;
  while (1) {
     foo(cnt); bar(cnt); sleep(1); cnt++;
  }
  return 0;
}
```

# USDT

```
$ perf buildid-cache --add demo

$ perf list | grep sdt_demo

$ perf probe add sdt_demo:foo; perf probe add sdt_demo:bar

$ perf trace -e sdt_demo:foo,sdt_demo:bar ./demo
```

# Demo - eBPF + bcc + tracing

# Use eBPF & kprobe hook to figure out DNS server

- DNS, UDP port 53
- Limit to AF_INET (IPv4)
- Packets are send with sendmsg(), with socket of SOCK_DGRAM, AF_INET

```
$ grep udp_send < /proc/kallsyms
0000000000000000 t udp_send_skb.isra.4
0000000000000000 T udp_sendmsg           ← USE THIS ONE
0000000000000000 T udp_sendpage
0000000000000000 t udp_sendmsg.cold.15
0000000000000000 r __ksymtab_udp_sendmsg
0000000000000000 r __kstrtab_udp_sendmsg
```

# udp_sendmsg

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg, size_t len)
{
  struct inet_sock *inet = inet_sk(sk);
  ...
  DECLARE_SOCKADDR(struct sockaddr_in *, usin, msg->msg_name);
  ...
  int connected = 0;
  __be32 daddr, faddr, saddr;
  __be16 dport;
  ...
  if (usin) {
      ...
      daddr = usin->sin_addr.s_addr;
      dport = usin->sin_port;
  } else {
      if (sk->sk_state != TCP_ESTABLISHED)
            return -EDESTADDRREQ;
      daddr = inet->inet_daddr;
      dport = inet->inet_dport;
      connected = 1;
  }
 ...
}
```

# bcc - BPF Compiler Collection

- Wrapper around calls to llvm
- Constrained C variant
- Hooks intro tracing subsystem
- Receiving data from the tracing subsystem
- Processing of simple trace_pipe
- C++, Python bindings

# bcc - kprobe

```
#include <uapi/linux/ptrace.h>
#include <net/sock.h>
#include <linux/socket.h>
#include <linux/in.h>
#include <linux/in6.h>

struct data_t {
      u32 family;
      u32 pid;
      u32 port;
      struct in_addr in;
};

// BPF table for pushing out data
BPF_PERF_OUTPUT(events);
```

# bcc - kprobe

```
/* UDP in IPv4 */
int kprobe__udp_sendmsg(struct pt_regs *ctx, struct sock *sk, struct msghdr *msg, size_t len) {
     struct data_t event = {};
     u32 tgid = bpf_get_current_pid_tgid() >> 32;
     event.pid = tgid;
     event.family = sk->__sk_common.skc_family;
     if (sk->sk_state == TCP_ESTABLISHED) {
          event.port = sk->__sk_common.skc_dport;
          event.in.s_addr = sk->__sk_common.skc_daddr;
     } else {
          struct sockaddr_in *addr = (struct sockaddr_in *)msg->msg_name;
          event.port = addr->sin_port;
          event.in = addr->sin_addr;
     }

     events.perf_submit(ctx, &event, sizeof(event));
     return 0;
}
```

# Thank you!