# eBPF - The World of Wonders

*The Sky-People have sent us a message, that they can take whatever they want, and no one can stop them*

# Demystifying eBPF with XDP

# About Me

Open Source Enthusiast , expert in Linux, Kubernetes and Containers.

Working as Lead SRE/DevSecOps @Cisco AI.

My GitHub: https://github.com/ansilh

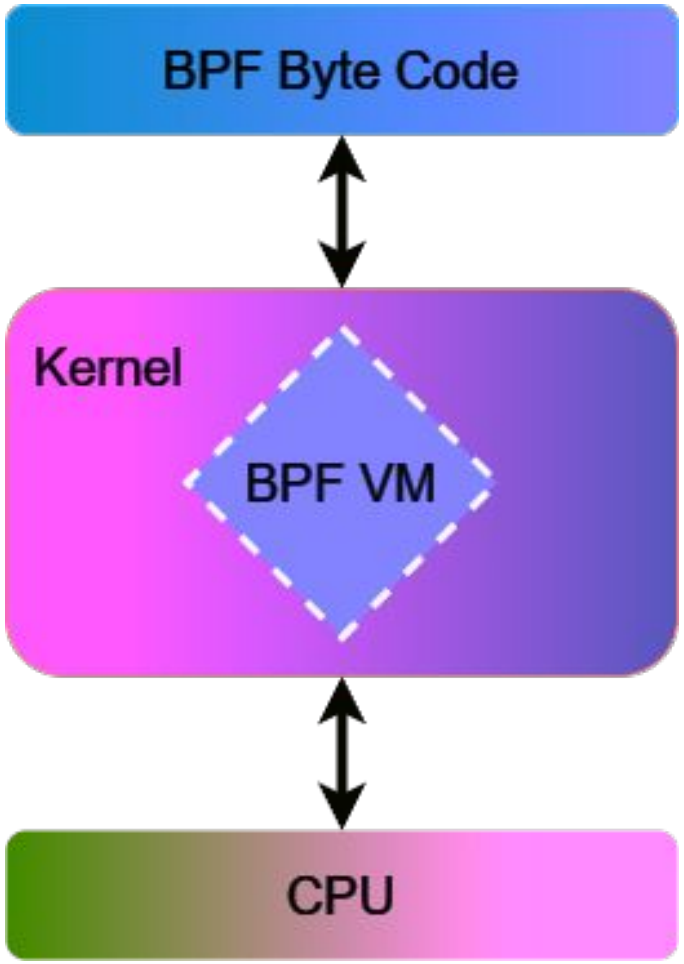
My LinkedIn: https://www.linkedin.com/in/ansilh/

# Linux – The Universe

*Sky People - The User space*

*Pandora - The Kernel space*

# The famous "*tcpdump*"

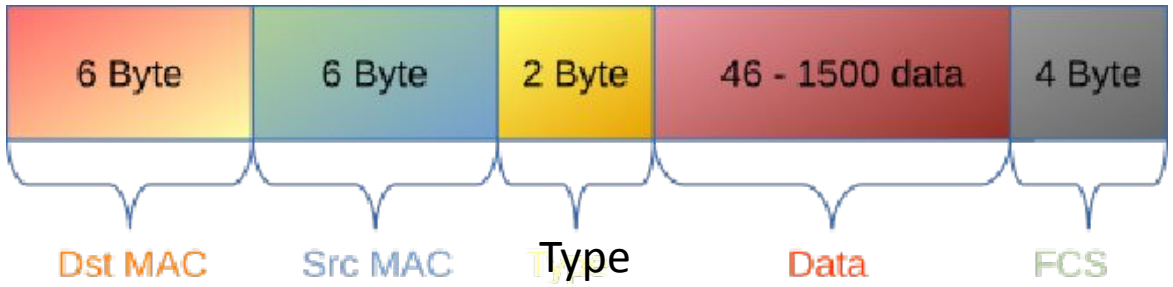


# The "*Object Code*"

```
[root@localhost ~]# tcpdump -i ens33 arp -d
(000) ldh      [12]
(001) jeq      #0x806           jt 2    jf 3
(002) ret      #262144
(003) ret      #0
[root@localhost ~]#
```

```
[root@localhost ~]# tcpdump -i ens33 arp -ddd
4
40 0 0 12
21 0 1 2054
6 0 0 262144
6 0 0 0
```

# Packet "*Ethernet type II*" frame



# Loadable Kernel Modules

```c
/*
 * hello.c - The simplest kernel module.
 */
#include <linux/module.h> /* Needed by all modules */
#include <linux/printk.h> /* Needed for pr_info() */
int init_module(void)
{
    pr_info("Hello world 1.\n");
    /* A non 0 return means init_module failed; module can't be loaded.
    return 0;
}

void cleanup_module(void)
{
   pr_info("Goodbye world 1.\n");
}
MODULE_LICENSE("GPL");
```

# Hello *LKM*

- sudo apt-get install make gcc
- make
- sudo insmod ./hello.ko
- dmesg | grep Hello
- lsmod | grep hello

# Demo

# *Helper Functions*

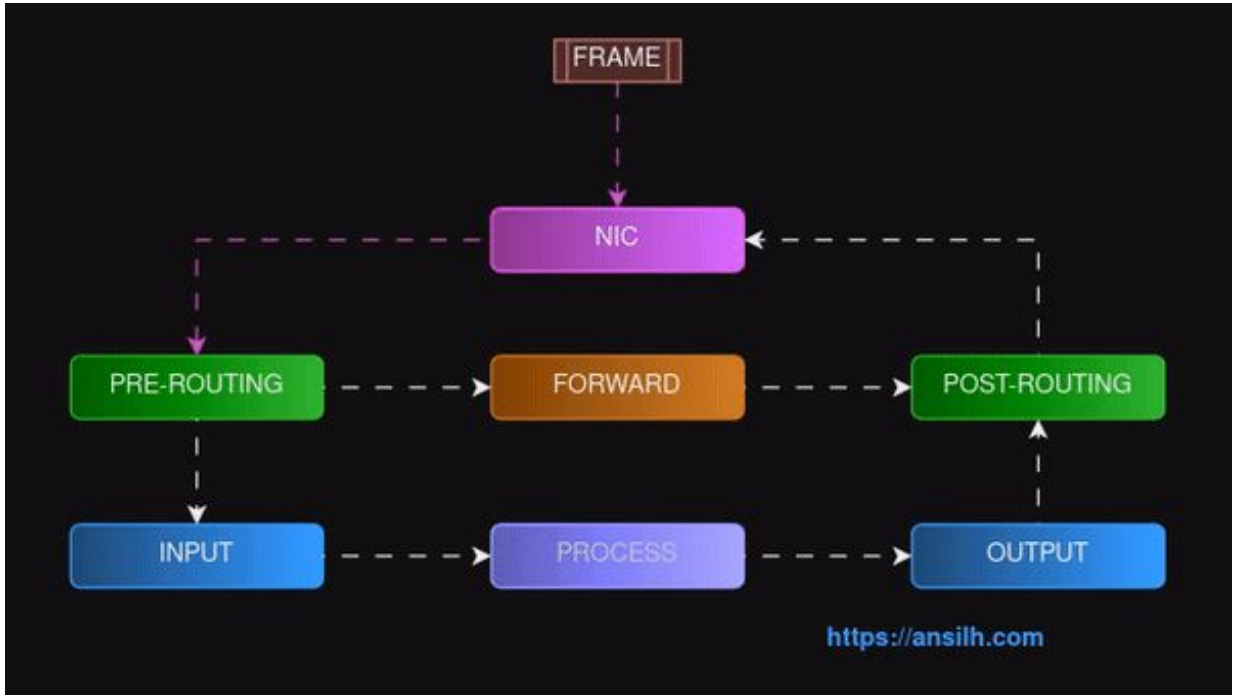
- Functions available with-in the kernel
- In previous example we used the helper function **pr_info** from header file **<linux/printk.h>**

# *Hooks*

- Hooks are pre-defined points in Kernel where you can register a function to it. When ever kernel reaches that hook, the registered function in that hook gets executed

# Packet "*Netfilter*" hooks

The *netfilter* hooks are a framework inside the Linux kernel that allows kernel modules to register callback functions at different locations of the Linux network stack.



# Demo *Firewall*

```c
#define BLOCK_PORT 80

// Callback function
static unsigned int hook_func(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
    // Access packet information using skb
    struct iphdr *iph = ip_hdr(skb);
    struct tcphdr *tcph = tcp_hdr(skb);

    if(ntohs(tcph->dest) == BLOCK_PORT){
      pr_info("Blocking packet from %pI4 to port %u\n",&iph->saddr, ntohs(tcph->dest));
      return NF_DROP; // Drop packets to port 80
    }

    return NF_ACCEPT; // Allow the packet to continue
}
```

# Demo *Firewall*

```c
// Netfilter hook options
static struct nf_hook_ops nf_hook_ops = {
    .hook = hook_func, // Our callback function
    .pf = PF_INET, // IPv4 protocol family
    .hooknum = NF_INET_PRE_ROUTING, // Hook point in the netfilter framework
    .priority = NF_IP_PRI_FIRST, // Invocation priority
};

static int __init drop_pkt_module_init(void) {
    return nf_register_net_hook(&init_net, &nf_hook_ops); //init_net indicates the root network na
}

static void __exit drop_pkt_module_exit(void) {
    nf_unregister_net_hook(&init_net, &nf_hook_ops);
}

module_init(drop_pkt_module_init);
module_exit(drop_pkt_module_exit);
MODULE_LICENSE("GPL");
```

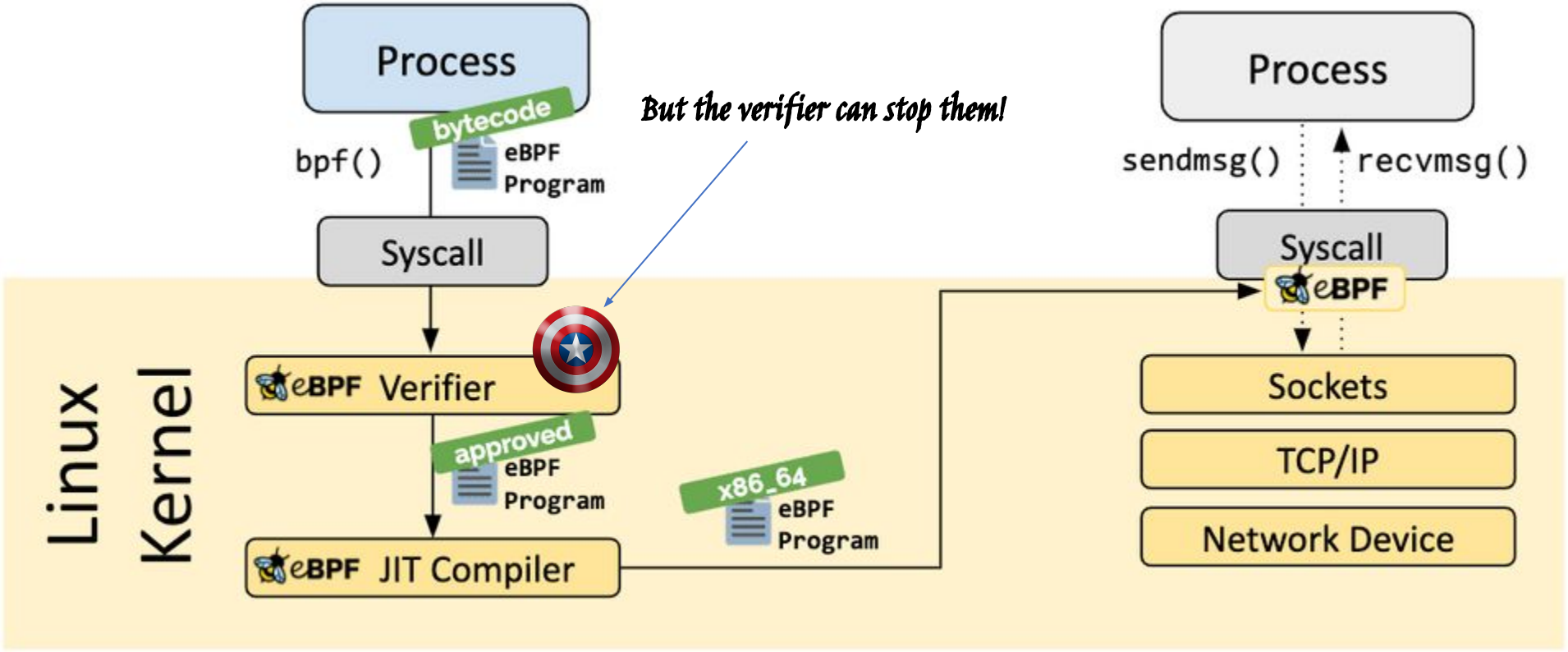
# The *LKM inside Kernel*

# eBPF

Become the one!

# eBPF Intro

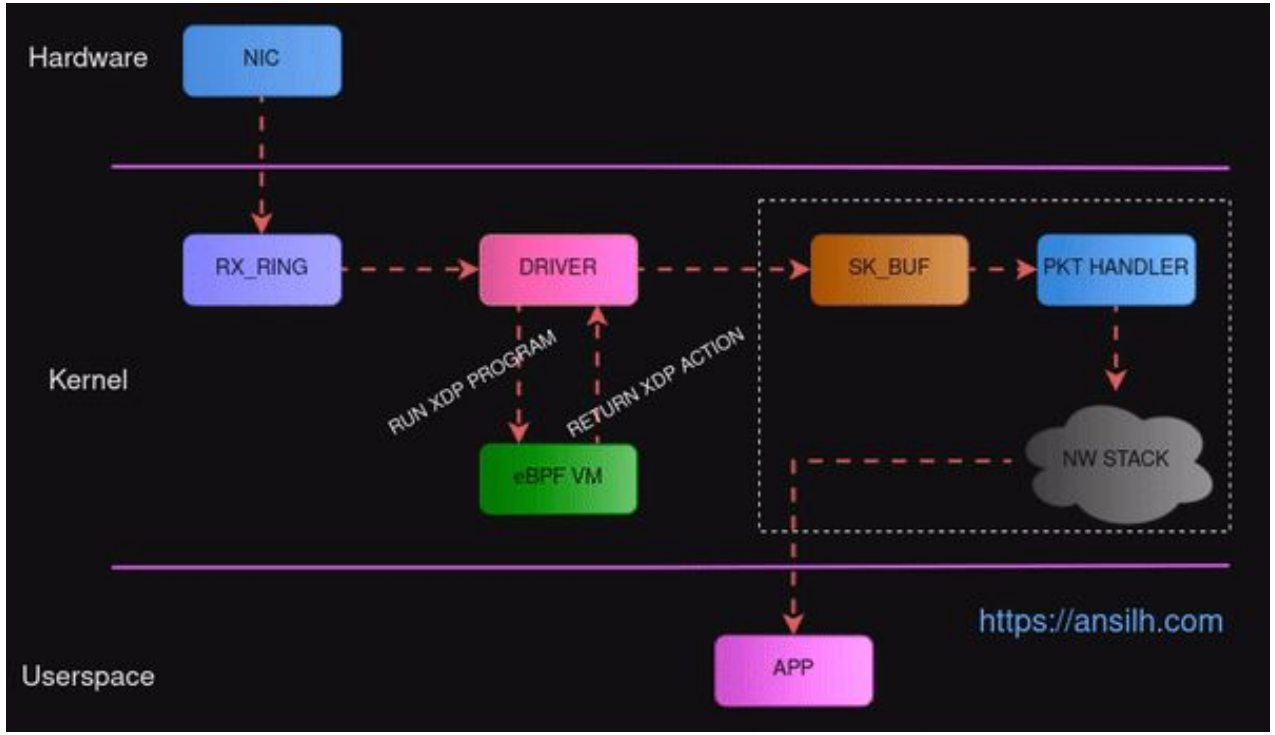
*The Sky-People have sent us a message, that they can take whatever they want, and no one can stop them*



# The Connection

# XDP Packet flow

- The packets comes to the NIC.
- NIC places those packets to the RX_RING
- When the buffer gets full*, an interrupt is fired to CPU.
- Once the CPU get’s interrupted, the loaded NIC driver code gets executed.
- The driver code reads the packets from the queue.
- **The initial path of packet after RX_RING is where we use XDP hook.**



# Type of *XDP*

- **Generic XDP**
  - XDP program loaded into the kernel as part of the network path.
- **Native XDP**
  - XDP program loaded by the driver in it’s initial receive path
- **Offloaded XDP**
  - XDP program loads directly to the NIC and handled by the NIC controller.

# *XDP* actions

Once the packet is received by the XDP program, it can do one of below;

- **XDP_DROP** : No processing, just drop the packet.
- **XDP_PASS** : Pass the packet to the next network stack component.
- **XDP_TX** : Forward the packet to the same network interface.
- **XDP_REDIRECT** : Froward the packet to another NIC and bypass all kernel

# *eBPF* program for *XDP*

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

SEC("xdp_drop")
int xdp_drop_prog(struct xdp_md *ctx)
{
    return XDP_DROP;
}

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

# Demo