Protocol Analysis and Defeating DDoS in FreeBSD Kernel

Playing with Packet Filter Hooks

Nebi Şenol YILMAZ
Senior Consultant / Managing Partner
Secrove Information Security Consulting
nsyilmaz@secrove.com
Twitter: @nsyilmaz
Linkedin: nsyilmaz

Agenda

- Why DoS/DDoS?
- Mitigation
- Why BSD?
- Kernel Level Programming
- Pfil Hooks
- White Hat
- Black Hat

Why DoS/DDoS?

Why DoS/DDoS?

Why DoS/DDoS?


WE ARE ANONYMOUS WE ARE LEGION
WE DO NOT FORGET WE DO NOT FORGIVE EXPECT US
?

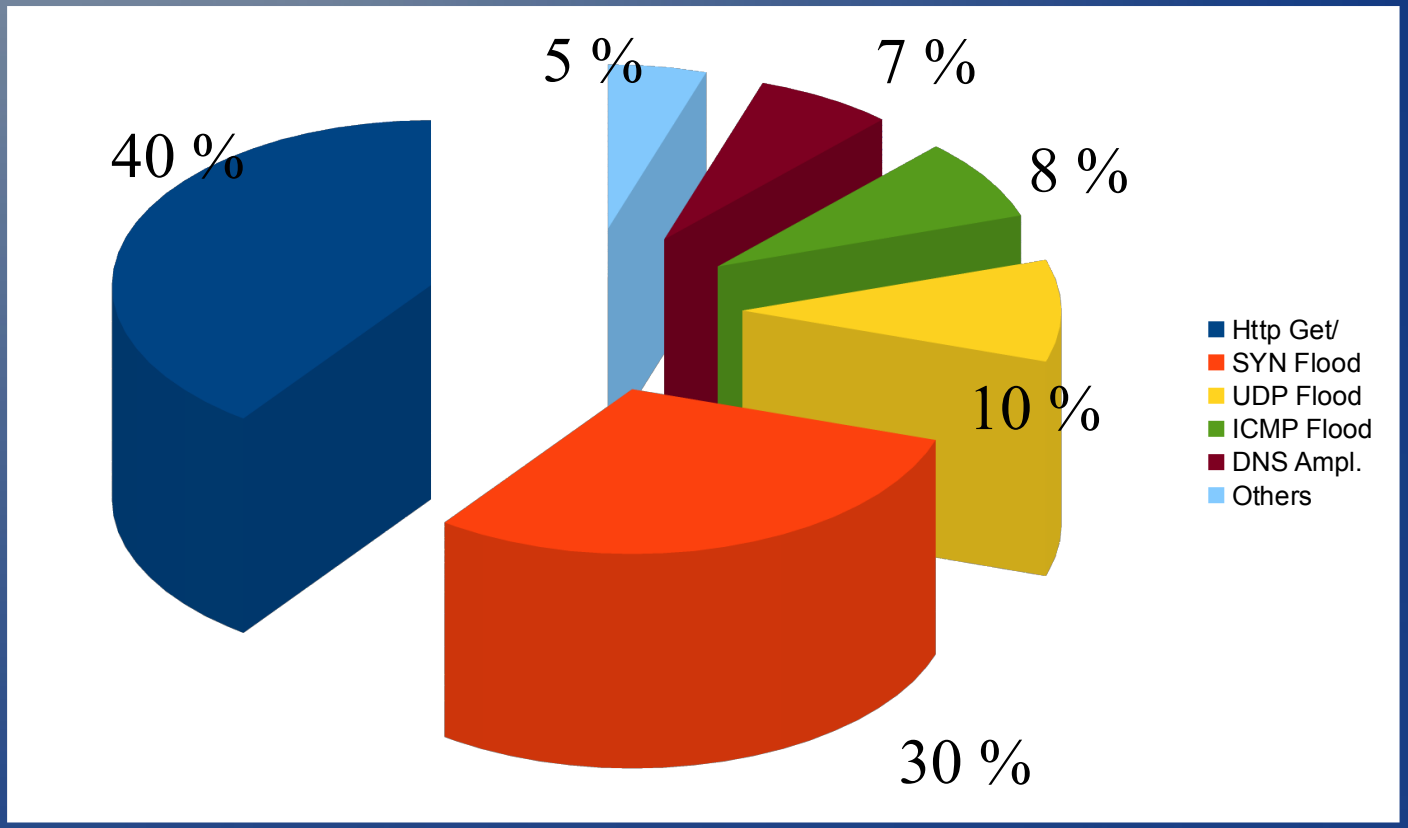
Why DoS/DDoS?

Takes part in real-life experience...

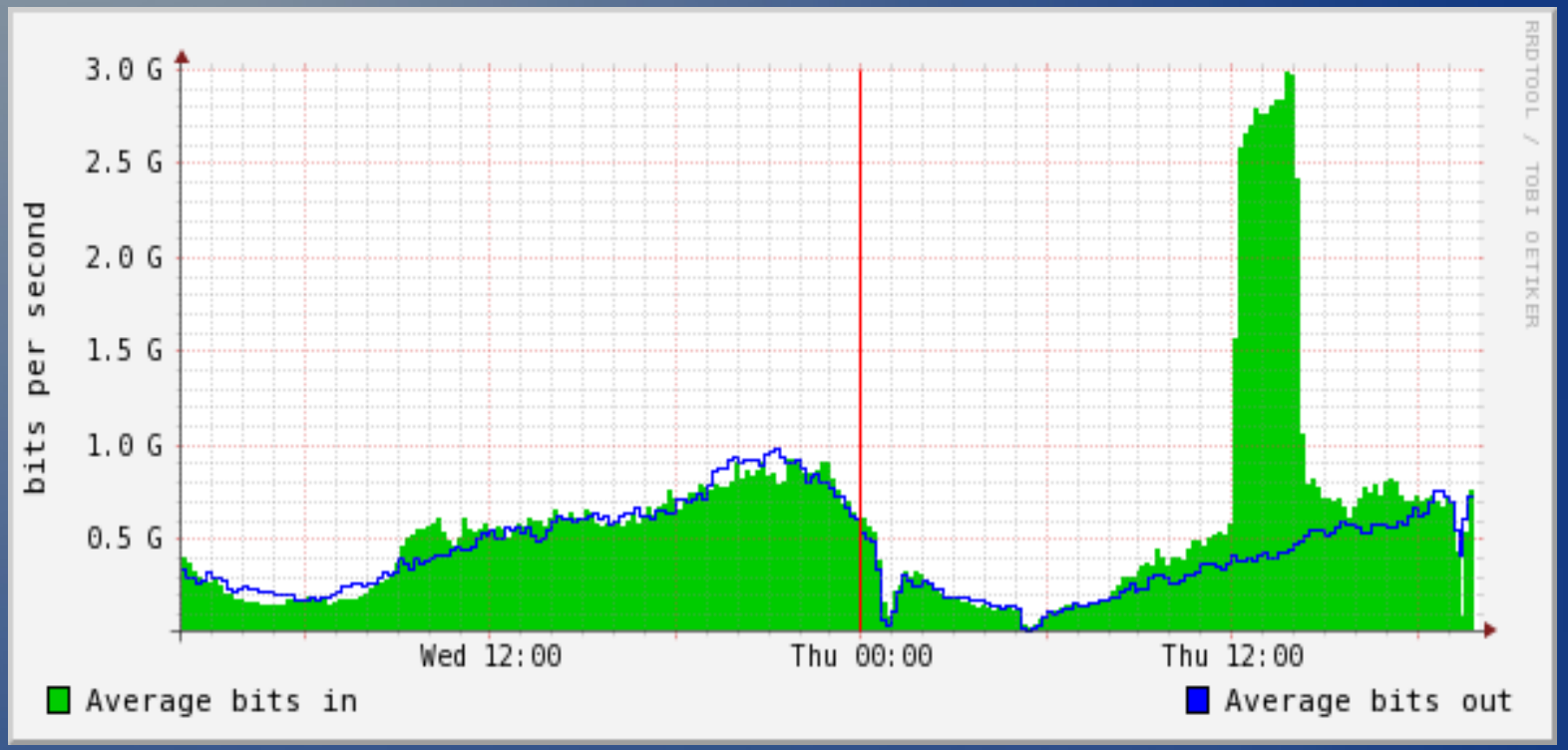
- WikiLeaks and VISA, MasterCard and PayPal
- Sony PlayStation, WordPress
- Attacks on stock exchanges (Hong-Kong)
- Political protests via attacks (US, Israel, TR)
- Finally, makes you offline... (TTNet)

Why DoS/DDoS?

So, how...?



Why DoS/DDoS?



Configuration vs Bandwidth

Mitigation

DoS / DDoS is NOT just a configuration or bandwidth problem...

- Awareness
- Know-how
- Communication & Cooperation

Specific solutions should be implemented other than traditional IPS or Firewall devices.

Mitigation

- Packet Accounting
- Syn Cookies (into seq. num.)
- Syn Caches (160 vs 736 bytes TCB)
- Syn Proxy
- TCP Backlog
- SYN-Received Timer

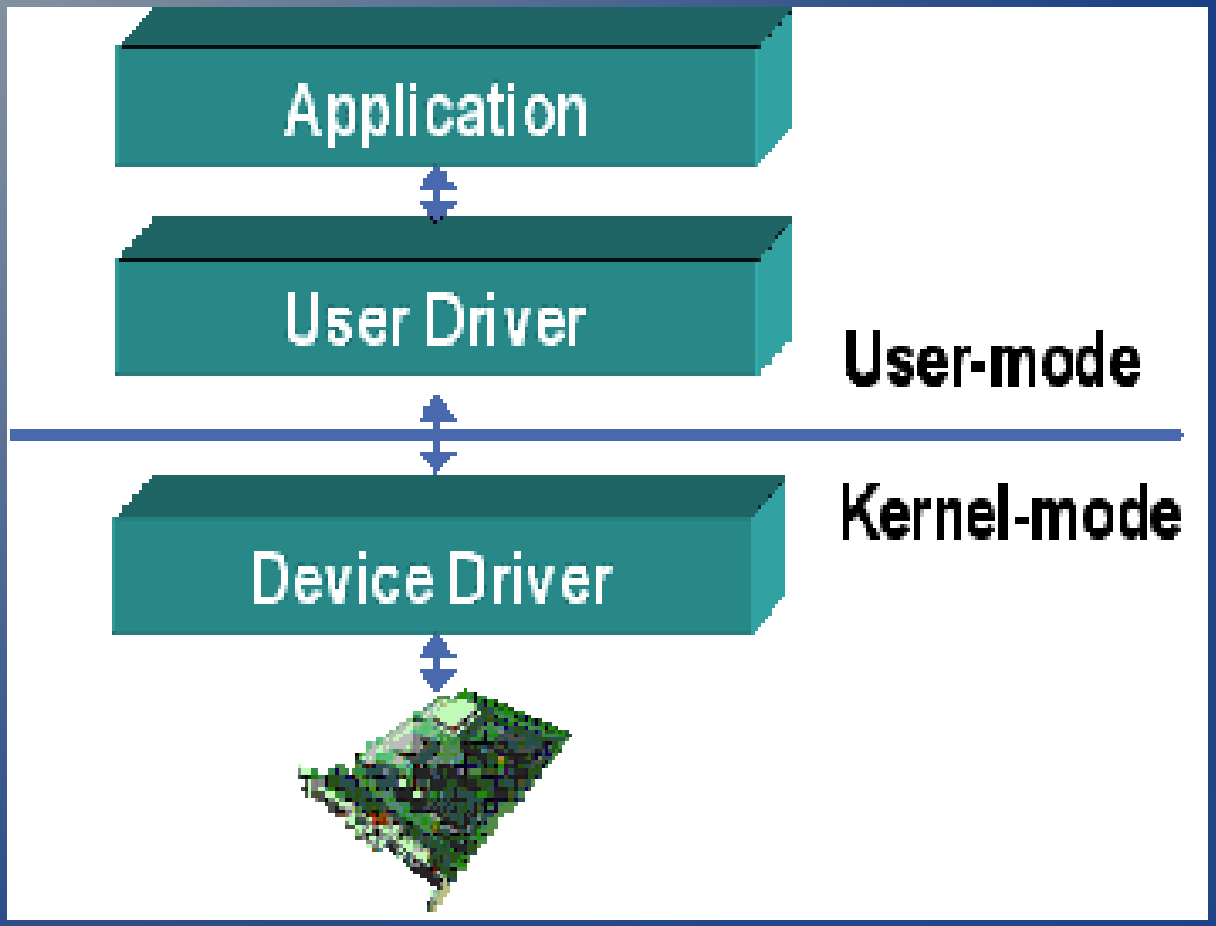
Why BSD?

- Complete OS
- Ancestor of well-known OS'
- Widespread adoption TCP/IP implementation
- Relatively much more stable
- Strong development environment (C)

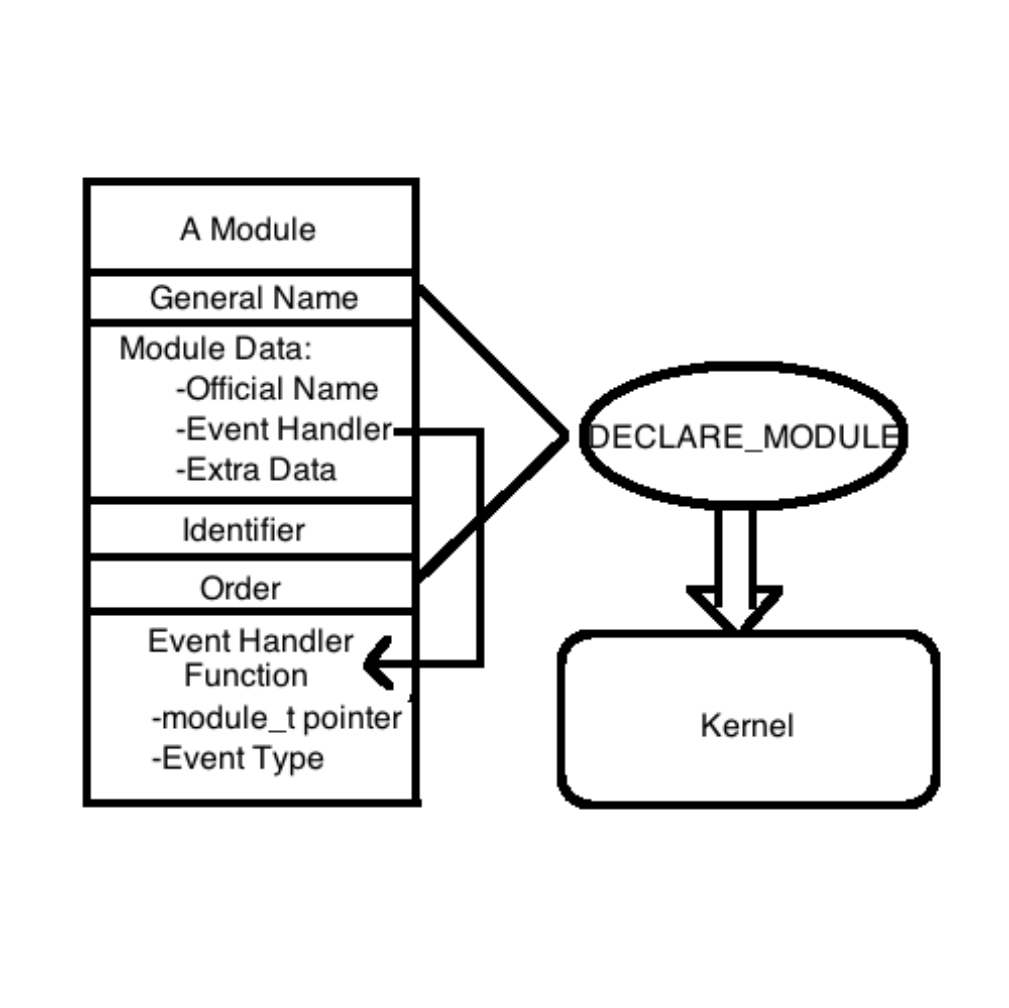
so...

- Technology companies are running on
- Security appliances are built on

Kernel Level Programming



Kernel Level Programming



Kernel Level Programming

- int handler(struct module *module, int event, void *arg);

- struct moduledata {

```
        const char      *name;              /* module name */
        modeventhand_t  evhand;             /* event handler */
        void            *priv;              /* extra data */
 }
```

- DECLARE_MODULE(name, moduledata_t data, sub, order);

```
#include <sys/param.h>
#include <sys/module.h>
#include <sys/kernel.h>
#include <sys/systm.h>


static int handler(struct module *module, int event, void *arg) {
        int e = 0;
        switch (event) {
        case MOD_LOAD:
                uprintf("Module Loaded\n");
                Break;

        case MOD_UNLOAD:
                uprintf("Module Unloaded\n");
                break;

        default:
                e = EOPNOTSUPP;
                break;
        }

        return(e);
}



static moduledata_t nopcon_conf = {
    "nopcon",
     handler,
     NULL
};



DECLARE_MODULE(nopcon, nopcon_conf, SI_SUB_DRIVERS, SI_ORDER_MIDDLE);
```

```
static int nopcon_function(unsigned int opt){

	switch (opt){
		case 1:
			printf("Function called on load...\n");
			break;

		case 2:
			printf("Function called on unload...\n");
			break;

		default:
			printf("undefined opt...\n");
			break;
	}


return 0;
}

static int handler(struct module *module, int event, void *arg) {
	int e = 0;
	switch (event) {
	case MOD_LOAD:
		uprintf("Module Loaded\n");
		nopcon_function(1);
		break;

	case MOD_UNLOAD:
		uprintf("Module Unloaded\n");
		nopcon_function(2);
		break;

	default:
		e = EOPNOTSUPP;
		break;
	}

	return(e);
}
```

Kernel Level Programming

- Load & unload, not so exciting...

- Let the kernel do something event driven...

```
struct callout callOut;

static void nopcon_function(void *opt){

	printf("Timer called...\n");
	callout_reset(&callOut, 1*hz, nopcon_function, NULL);

}


static void initTimer(void){
	callout_init(&callOut,1);
	callout_reset(&callOut, 1*hz, nopcon_function, NULL);
}

static void deinitTimer(void){
	callout_stop(&callOut);
}


static int handler(struct module *module, int event, void *arg) {
	int e = 0;
	switch (event) {
	case MOD_LOAD:
		uprintf("Module Loaded\n");
		initTimer();
		break;

	case MOD_UNLOAD:
		uprintf("Module Unloaded\n");
		deinitTimer();
		break;

	default:
		e = EOPNOTSUPP;
		break;
	}

	return(e);
}
```

Kernel Level Programming

- Let's do something different other than calling a function :P

Pfil Hooks


ORIGINAL MOTION PICTURE SOUNDTRACK
Hook
MUSIC COMPOSED AND CONDUCTED BY JOHN WILLIAMS

Pfil Hooks

The pfil framework allows for a specified function to be invoked for every incoming or outgoing packet for a particular network I/O stream.

These hooks may be used to implement a firewall or perform packet transformations.

When a filter is invoked, the packet appears just as if it “came off the wire”. That is, all protocol fields are in network byte order.

The latest pfil input and output lists were implemented as <sys/queue.h> TAILQ structures.

Pfil Hooks

- The hook function should be added to hook list.

- void pfil_add_hook(

```
 int (*func)(),              // name of the hook function
 void *arg,                  // arguments
 int flags,                  // flags
 struct pfil_head *);        // head of the hook list
```

Pfil Hooks

- The hook function should be implemented in this form:

- Int (*func)(
  void *arg, // argument
  struct mbuf **mp, // socket buffer
  struct ifnet *, // network interface
  int dir, // flow direction
  struct inpcb *); // protocol control block

Pfil Hooks

- Head of the hook list should be determined

- struct pfil_head *pfil_head_get(
  int af, // address family
  u_long dlt); // data link type

Pfil Hooks

How the hooks are invoked...?

Let's check out the network stack in kernel iplementation...

Input: “netinet/ip_input.c”
Output: “netinet/ip_output.c”

Pfil Hooks

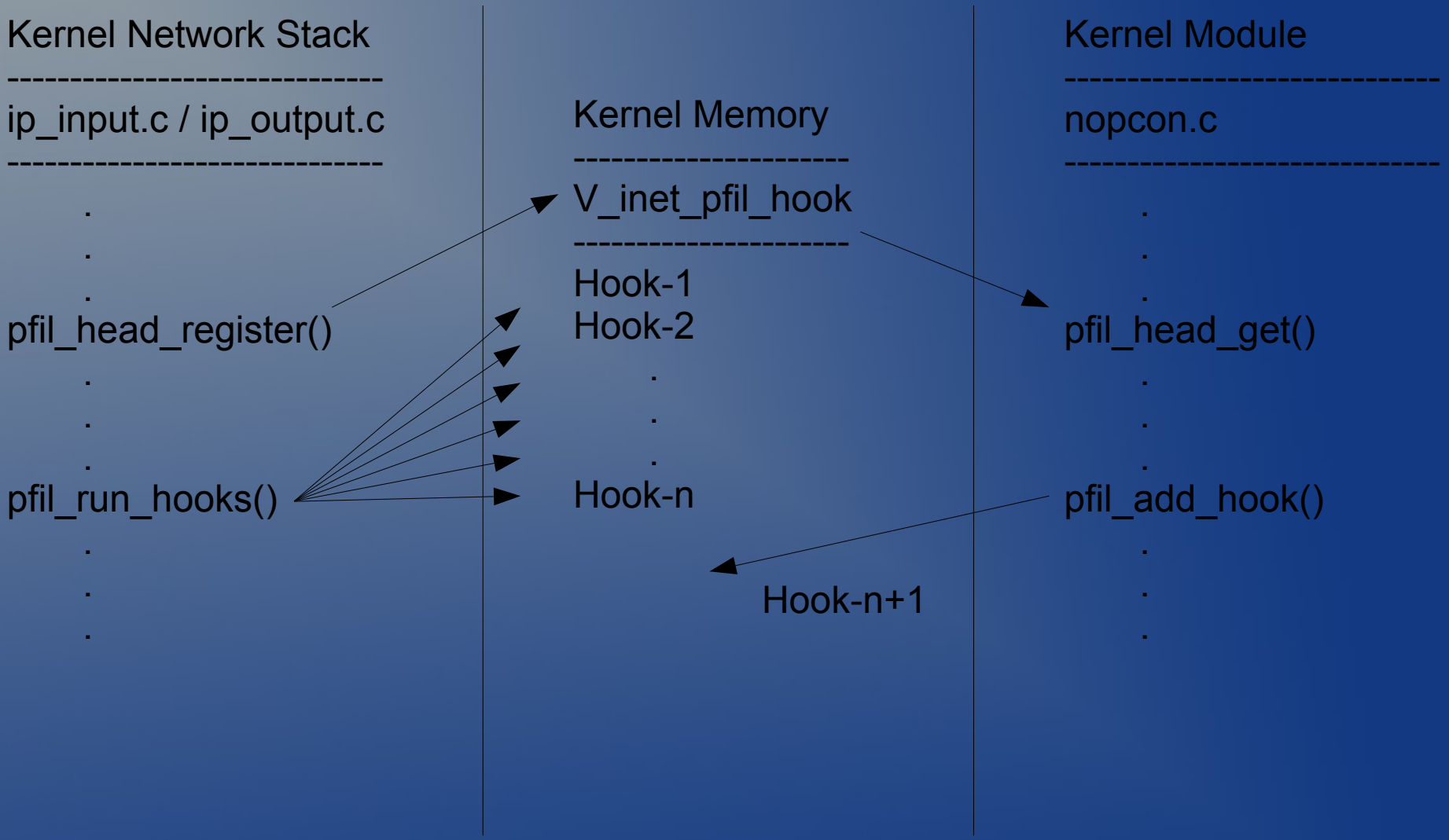
/usr/src/sys/netinet/ip_input.c

```
/*
 * IP initialization: fill in IP protocol switch table.
 * All protocols not implemented in kernel go to raw IP protocol handler.
 */
void
ip_init(void)
{
                        .
                        .
                        .
        /* Initialize packet filter hooks. */
        V_inet_pfil_hook.ph_type = PFIL_TYPE_AF;
        V_inet_pfil_hook.ph_af = AF_INET;
        if ((i = pfil_head_register(&V_inet_pfil_hook)) != 0)
                printf("%s: WARNING: unable to register pfil hook, "
                        "error %d\n", __func__, i);
                        .
                        .
                        .
}

/*
 * Ip input routine.  Checksum and byte swap header.  If fragmented
 * try to reassemble.  Process options.  Pass to next level.
 */
void
ip_input(struct mbuf *m)
{
                        .
                        .
                        .
        /* Jump over all PFIL processing if hooks are not active. */
        if (!PFIL_HOOKED(&V_inet_pfil_hook))
                goto passin;

        odst = ip->ip_dst;
        if (pfil_run_hooks(&V_inet_pfil_hook, &m, ifp, PFIL_IN, NULL) != 0)
                return;
                        .
                        .
                        .
}
```

/usr/src/sys/netinet/ip_output.c

```
/*
 * IP output.  The packet in mbuf chain m contains a skeletal IP
 * header (with len, off, ttl, proto, tos, src, dst).
 * ip_len and ip_off are in host format.
 * The mbuf chain containing the packet will be freed.
 * The mbuf opt, if present, will not be freed.
 * In the IP forwarding case, the packet will arrive with options already
 * inserted, so must have a NULL opt pointer.
 */
int
ip_output(struct mbuf *m, struct mbuf *opt, struct route *ro, int flags,
    struct ip_moptions *imo, struct inpcb *inp)
{
                        .
                        .
                        .
        /* Jump over all PFIL processing if hooks are not active. */
        if (!PFIL_HOOKED(&V_inet_pfil_hook))
                goto passout;

        /* Run through list of hooks for output packets. */
        odst.s_addr = ip->ip_dst.s_addr;
        error = pfil_run_hooks(&V_inet_pfil_hook, &m, ifp, PFIL_OUT, inp);
        if (error != 0 || m == NULL)
                goto done;
                        .
                        .
                        .
}
```

Pfil Hooks


Kernel Network Stack
ip_input.c / ip_output.c
pfil_head_register()
pfil_run_hooks()
Kernel Memory
V_inet_pfil_hook
Hook-1
Hook-2
Hook-n
Hook-n+1
Kernel Module
nopcon.c
pfil_head_get()
pfil_add_hook()

Pfil Hooks

```
static int nopcon_hook(void *arg, struct mbuf **mp, struct ifnet *ifp, int dir, struct inpcb *inp){

        struct ip *ip;
        struct tcphdr *tcp;
        struct udphdr *udp;
        struct icmphdr *icmp;

        unsigned int hlen;
        unsigned int totlen;


        ip = mtod(*mp, struct ip *);
        hlen = ip->ip_hl << 2;
        totlen = ip->ip_len;


        tcp = (struct tcphdr *)((unsigned char *)ip + hlen);
        udp = (struct udphdr *)((unsigned char *)ip + hlen);
        icmp = (struct icmphdr *)((unsigned char *)ip + hlen);

return 0;

}
static int handler(struct module *module, int event, void *arg) {
        int e = 0;
        struct pfil_head *pfh;

        switch (event) {
        case MOD_LOAD:
                uprintf("Module Loaded\n");
                pfh = pfil_head_get(PFIL_TYPE_AF, AF_INET);
                pfil_add_hook(nopcon_hook, NULL, PFIL_IN | PFIL_WAITOK, pfh);
                break;

        case MOD_UNLOAD:
                uprintf("Module Unloaded\n");
                pfh = pfil_head_get(PFIL_TYPE_AF, AF_INET);
                pfil_remove_hook(nopcon_hook, NULL, PFIL_IN | PFIL_WAITOK, pfh);
                break;

        default:
                e = EOPNOTSUPP;
                break;
        }

        return(e);
}
```

White Hat

Session / Packet accounting samples for DDoS mitigation...

Black Hat

Session / Packet alteration on wire...

Questions / Answers

?

Thanks ...