LINUX KERNEL EBPF UND DNS

(DDI User Group 2022)

CARSTEN STROTMANN

Created: 2022-06-23 Thu 16:02

WER SPRICH HIER?

Carsten Strotmann

dnsworkshop.de

blog.defaultroutes.de

DNS(SEC)/DANE/DHCP/IPv6 Trainer und Helfer

RIPE/IETF

EBPF

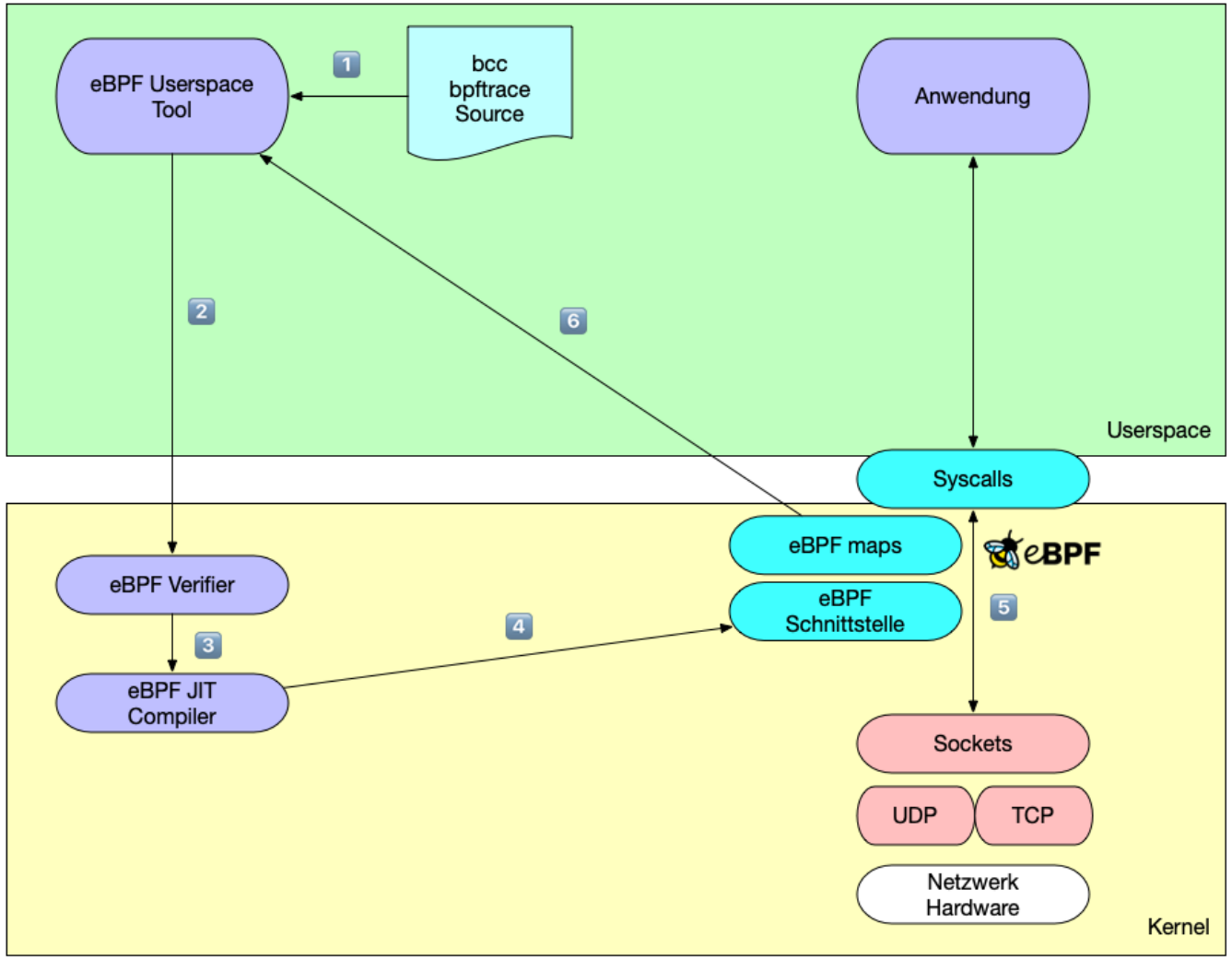
WAS IST EBPF UND BCC?

- eBPF ist die *extended Berkeley Packet Filter* virtuelle Maschine im Linux Kernel
- BCC ist die *BPF Compiler Collection*, eine Sammlung von Tools und eBPF Programmen
- eBPF ist eine Weiterentwicklung der originalen Berkeley Packet Filter Technologie
  https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

DIE EBPF IDEE

- eBPF erlaubt es dem Benutzer, Programme im Betriebssystem-Kern innerhalb einer Sandbox auszuführen
  - eBPF ermöglicht es, die Funktionen des Betriebssystem-Kerns sicher und effizient zu erweitern, ohne den Quell-Code des Kernels ändern oder Module laden zu müssen
  - eBPF Programme können Netzwerk-Pakete (und andere Datenstrukturen) innerhalb des Linux-Kernels überwachen und verändern
  - eBPF Programme sind **keine** Kernel Module, es ist nicht notwendig ein Kernel-Entwickler zu sein um eBPF benutzen zu können
    - Wissen über Programmierung in der Sprache "C" ist jedoch von Vorteil

EBPF


eBPF Userspace Tool
1
bcc bpftrace Source
Anwendung
2
6
Userspace
Syscalls
eBPF maps
eBPF
eBPF Schnittstelle
5
eBPF Verifier
4
3
eBPF JIT Compiler
Sockets
UDP
TCP
Netzwerk Hardware
Kernel

EBPF EINSATZGEBIETE

- Einsatzgebiete für eBPF
  - Netzwerk Sicherheit (erweiterte Firewall Funktionen)
  - Host Security
  - Forensische Analysen
  - Fehlerdiagnose
  - Geschwindigkeit-Messungen
- eBPF ist in modernen Linux Systemen verfügbar (ab Kernel 3.18+) und wird derzeit von Microsoft auf das Windows Betriebssystem portiert
  - Blog Post "Making eBPF work on Windows" https://cloudblogs.microsoft.com/opensource/2021/05/10/making-ebpf-work-on-windows/

DIE WURZELN VON BPF

- Der originale BSD Packet Filter (BPF) wurde von Steven McCanne und Van Jacobson am Lawrence Berkeley Laboratory entwickelt (https://www.tcpdump.org/papers/bpf-usenix93.pdf)
  - BPF wurde auf fast alle Unix/Linux Systeme und viele non-Unix Betriebssysteme portiert (z.B. Windows, BeOS/Haiku, OS/2 …)
  - BPF ist die Basis-Technologie hinter bekannten Netzwerk-Sniffing Werkzeugen wie `tcpdump` und *Wireshark*

BPF AM BEISPIEL VON TCPDUMP

- Wird ein Werkzeug auf Basis vom BPF verwendet, so wird der Filter in einen Bytecode für die BPF virtuelle Maschine im Linux-Kernel übersetzt und in den Kernel geladen
  - Das Betriebssystem ruft das BPF-Programm für jedes Netzwerk-Paket auf, welches den Netzwerk-Stack durchläuft
  - Nur Pakete welche auf den Filter-Ausdruck passen werden an das Programm im Userspace weitergeleitet (`tcpdump` in dieses Beispiel)
  - BPF reduziert die Menge an Daten welche zwischen dem Kernel und dem Userspace ausgetauscht werden müssen

BPF AM BEISPIEL VON TCPDUMP

`tcpdump` kann angewiesen werden den BPF Quellcode des `tcpdump` Filters auszugeben:

```
# tcpdump -d port 53 and host 1.1.1.1
Warning: assuming Ethernet
(000) ldh      [12]
(001) jeq      #0x86dd          jt 19   jf 2
(002) jeq      #0x800           jt 3    jf 19
(003) ldb      [23]
(004) jeq      #0x84            jt 7    jf 5
(005) jeq      #0x6             jt 7    jf 6
(006) jeq      #0x11            jt 7    jf 19
(007) ldh      [20]
(008) jset     #0x1fff          jt 19   jf 9
(009) ldxb     4*([14]&0xf)
(010) ldh      [x + 14]
(011) jeq      #0x35            jt 14   jf 12
(012) ldh      [x + 16]
(013) jeq      #0x35            jt 14   jf 19
(014) ld       [26]
(015) jeq      #0x1010101       jt 18   jf 16
(016) ld       [30]
(017) jeq      #0x1010101       jt 18   jf 19
(018) ret      #262144
(019) ret      #0
```

EBPF VS. BPF

- Während BPF (heute auch cBPF = classic BPF genannt) Netzwerk Pakete im Betriebssystem-Kern filtert, kann eBPF noch auf weitere Datenstrukturen Filter anwenden:
  - Kernel Systemcalls
  - Kernel Tracepoints
  - Kernel Funktionen
  - Userspace Tracepoints
  - Userspace Funktionen

EBPF UND DER LINUX KERNEL

- Die erste Version von eBPF wurde im Linux Kernel 3.18 eingeführt
  - die meisten neuen Kernel Versionen seither haben weitere, neue eBPF Funktionen implementiert
  - Linux Distributionen (Red Hat/Canonical/Suse) haben zum Teil eBPF Funktionen auf ältere LTS Kernel Versionen zurückportiert
  - Eine Übersicht der eBPF Funktionen nach Linux Kernel Version aufgeschlüsselt: https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

DIE EBPF ARCHITEKTUR

DIE EBPF VIRTUELLE MASCHINE

- eBPF Programme werden für eine virtuelle CPU Architektur übersetzt
- Der Programmcode wird in den Linux Kernel geladen und dort geprüft
- Auf einigen CPU Architekturen (amd64, AARCH64) wird der eBPF Bytecode in nativen Maschinencode re-compiliert (Just in Time Compiler = JIT)

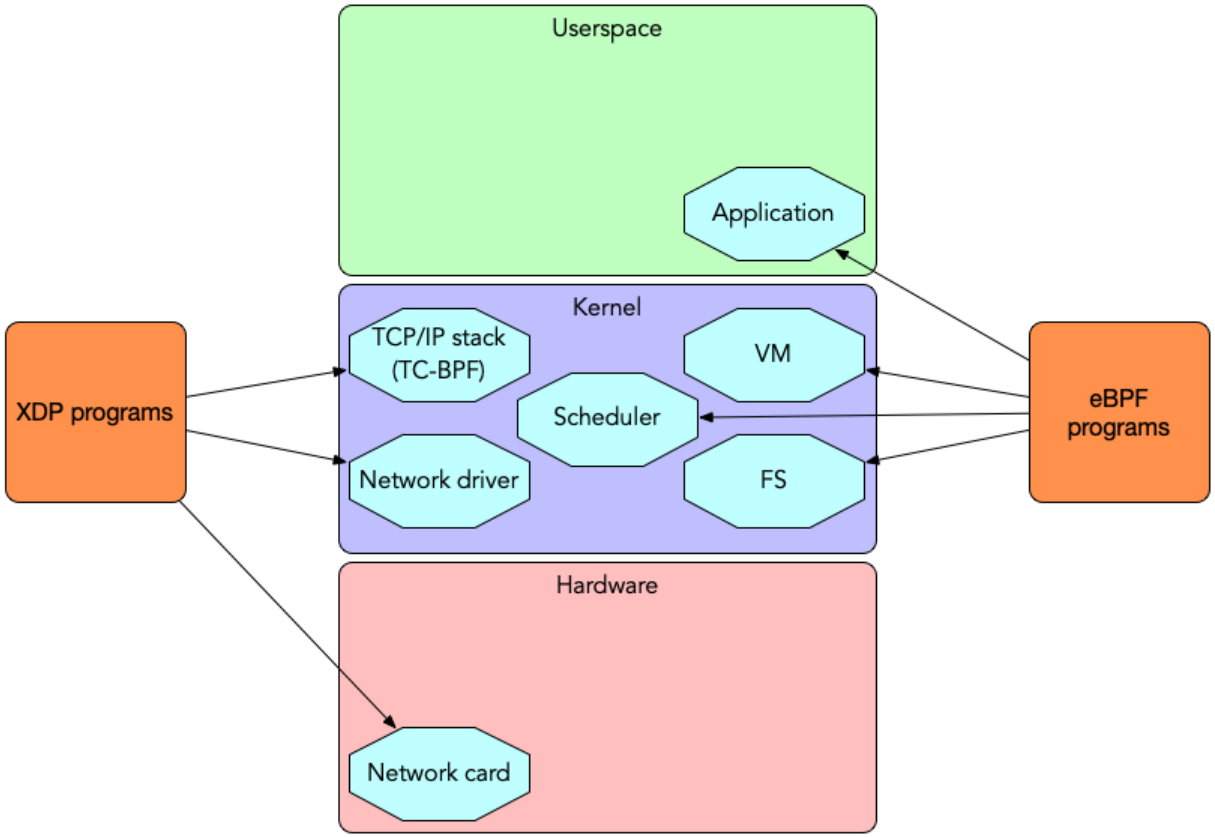
XDP - EXPRESS DATA PATH

- Der *express data path* (XDP) innerhalb des Linux-Kernels ist eine Infrastruktur, um auf unterster Ebene Kontrolle über Netzwerk-Pakete auszuüben
  - Der normale Datenfluss im Linux Netzwerk-Stack kann via XDP umgangen werden
  - eBPF Programme können in den eXpress Data Path (XDP) geladen werden

XDP / EBPF HARDWARE OFFLOADING

- XDP eBPF Programm können auf verschiedenen Ebenen in den Linux Kernel geladen werden
  - **Offload XDP**: direkt in die Netzwerk-Hardware (ASIC/FPGA, benötigt Unterstützung für XDP in der Hardware, z.B. vorhanden in den Netronome Netzwerkadaptern)
  - **Native XDP**: In den Linux Kernel Netzwerk-Treiber der Netzwerkschnittstelle (benötigt Unterstützung durch den Treiber)
  - **Generic XDP**: In den Linux Kernel Netzwerk-Stack (weniger Performance, aber ohne besondere Unterstützung von Hardware oder Treibern möglich)

XDP / EBPF AUSFÜHRUNGS-EBENEN

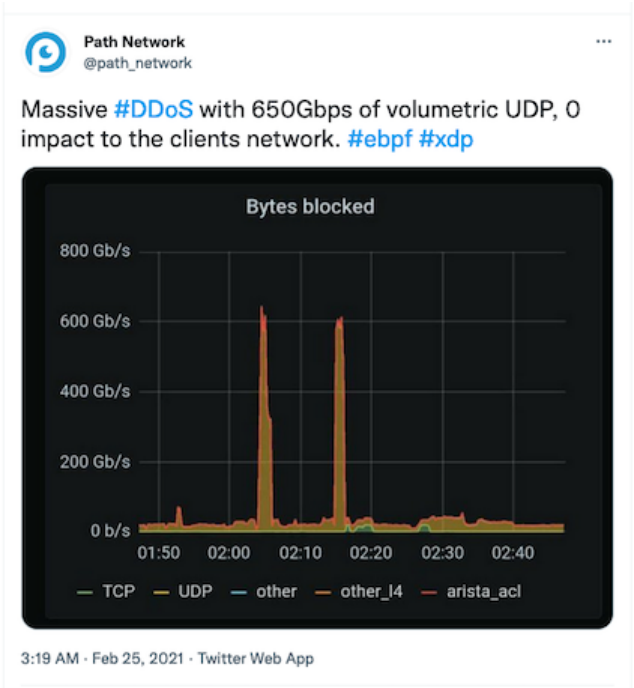


XDP FUNKTIONEN

- XDP Programme können
  - **lesen**: Netzwerk-Pakete und Statistiken sammeln
  - **verändern**: Den Inhalt der Netzwerkpakete ändern
  - **verwerfen**: Ausgewählte Netzwerk-Pakete können verworfen werden (Firewall)
  - **umleiten**: Netzwerkpakete können auf dem gleichen oder anderen Netzwerkschnittstellen umgeleitet werden (Switching/Routing)
  - **durchlassen**: Das Netzwerkpaket wird an den Linux TCP/IP Stack zur normalen Bearbeitung übergeben

XDP VS DDOS ANGRIFFE

- XDP kann unerwünschten Netzwerk-Verkehr schon sehr früh im Netzwerk-Stack verwerfen (z.B. innerhalb der Netzwerk-Hardware). Dies ist ein guter Schutz gegen DDoS Angriffe



EBPF/XDP SUPPORT IN DNS SOFTWARE

- Der Open-Source DNS Load-Balancer DNSdist von PowerDNS kann DNS Pakete via eBPF und XDP filtern oder per Rate-Limiting beschränken
- Der Knot Resolver (seit Version 5.2.0) kann mittels ePBF und XDP den Linux TCP/IP Stack für DNS Pakete umgehen und die DNS-Pakete direkt an den Knot DNS Resolver Prozess im Userspace weiterleiten (https://knot-resolver.readthedocs.io/en/stable/daemon-bindings-net_xdpsrv.html)

EBPF BENUTZEN

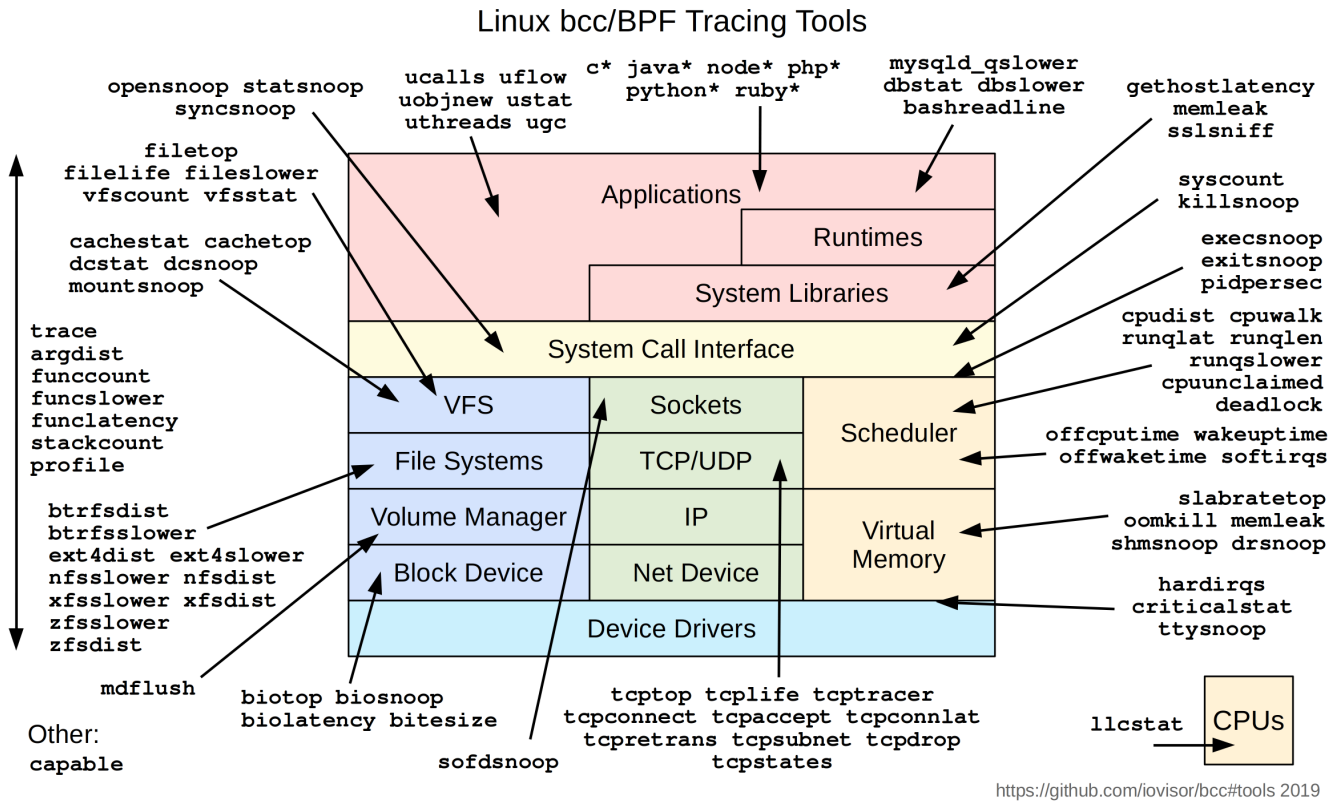
EBPF PROGRAMME ERSTELLEN

- eBPF Programme können auf verschiedene Weise erstellt werden
  - Als Low level eBPF Assembler Code
  - Mit einem High Level Compiler (mit Hilfe von LLVM): C / GO / Rust / Lua / Python …
  - Spezialisierte eBPF Script-Sprachen: z.B. `bpftrace`

BCC

- BCC ist die BPF Compiler Sammlung
  - Website https://github.com/iovisor/bcc
  - BCC übersetzt C oder Python Quellcode in eBPF Programme und kann diese in den Linux Kernel laden

FERTIGE BCC EBPF PROGRAMME



https://github.com/iovisor/bcc#tools 2019

BEISPIEL DER BENUTZUNG EINES MIT BCC ERSTELLTEN EPBF WERKZEUGS (1/2)

- Die Syscalls eines BIND 9 Prozesses auswerten mit dem Programm `syscount`

```
# syscount-bpfcc -p `pgrep named` -i 10
Tracing syscalls, printing top 10... Ctrl+C to quit.
[07:34:19]
SYSCALL                   COUNT
futex                       547
getpid                      121
sendto                      113
read                         56
write                        31
epoll_wait                   31
openat                       23
close                        20
epoll_ctl                    20
recvmsg                      20
```

BEISPIEL DER BENUTZUNG EINES MIT BCC ERSTELLTEN EBPF WERKZEUGS (2/2)

- Die Linux Capabilities von laufenden Prozessen anzeigen

```
# capable-bpfcc | grep named
07:36:17  0      29378  (named)          24   CAP_SYS_RESOURCE     1
07:36:17  0      29378  (named)          24   CAP_SYS_RESOURCE     1
07:36:17  0      29378  (named)          12   CAP_NET_ADMIN        1
07:36:17  0      29378  (named)          21   CAP_SYS_ADMIN        1
07:36:17  0      29378  named            6    CAP_SETGID           1
07:36:17  0      29378  named            6    CAP_SETGID           1
07:36:17  0      29378  named            7    CAP_SETUID           1
07:36:17  109    29378  named            24   CAP_SYS_RESOURCE     1
```

DIE EBPF SKRIPTSPRACHE "BPFTRACE"

- `bpftrace` ist eine kleine Skripting-Sprace ähnlich wie `awk` oder `dtrace`
  - Website https://bpftrace.org
- `bpftrace` Programme binden sich an eBPF-Probes und führen Funktionen aus, wann immer ein System-Ereignis gemeldet wird (systemcall, function-call)
- `bpftrace` hat Hilfs-Strukturen eingebaut um direkt mit eBPF Datanstrukturen arbeiten zu können
- `bpftrace` erlaubt es eBPF Programme im Vergleich zu BCC kompakter zu schreiben

INSTRUMENTIERUNG DES LINUX TCP/IP STACKS MITTELS EBPF

BCC UND BPFTRACE PROGRAMME

- Es gibt hunderte kleine eBPF Programme welche es dem Benutzer erlauben tief in den Linux Netzwerk Stack zu schauen the Linux network stack
  - Die BCC Beispielprogramme
  - Die `bpftrace` Beispielprogramme
  - Die Programme aus dem eBPF Büchern und den dazugehörigen Webseiten

BEISPIEL: GETHOSTLATENCY

- Das BCC Programm `gethostlatency` misst die Latenz der client-seitigen DNS Namensauflösung durch Systemaufrufe wie `getaddrinfo` oder `gethostbyname`

```
# gethostlatency-bpfcc
TIME      PID    COMM                  LATms HOST
10:21:58  19183  ping                 143.22 example.org
10:22:18  19184  ssh                    0.03 host.example.de
10:22:18  19184  ssh                   60.59 host.example.de
10:22:35  19185  ping                  23.44 isc.org
10:22:49  19186  ping                4459.72 yahoo.co.kr
```

BEISPIEL: NETQTOP

- `netqtop` - Gibt Statistiken über die Warteschlangen einer Netzwerkschnittstelle aus. Mit diesem Programm können Informationen bei der Überlastung einer Netzwerkschnittstelle gesammelt werden.

```
# netqtop-bpfcc -n eth0 -i 10
Mon Nov 15 07:43:29 2021
TX
 QueueID    avg_size   [0, 64)    [64, 512)  [512, 2K)  [2K, 16K)  [16K, 64K)
 0          297.82     2          48         1          4          0
 Total      297.82     2          48         1          4          0

RX
 QueueID    avg_size   [0, 64)    [64, 512)  [512, 2K)  [2K, 16K)  [16K, 64K)
 0          70.95      43         34         0          0          0
 Total      70.95      43         34         0          0          0
--------------------------------------------------------------------------------
```

BEISPIEL: TCPTRACER

- Dieses Programm zeig den Status von TCP Verbindungen im System (A = accept, C = connected, X = closed) sowie der Quell- und Ziel IP-Adressen und -Ports.

```
# tcptracer-bpfcc -p $(pgrep named)
Tracing TCP established connections. Ctrl-C to end.
T  PID    COMM             IP SADDR            DADDR            SPORT  DPORT
C  29404  isc-net-0000     4  127.0.0.1        127.0.0.1        41555  953
A  29378  isc-socket-0     4  127.0.0.1        127.0.0.1        953    41555
X  29404  isc-socket-0     4  127.0.0.1        127.0.0.1        41555  953
X  29378  isc-socket-0     4  127.0.0.1        127.0.0.1        953    41555
C  29378  isc-net-0000     4  46.101.109.138   192.33.4.12      43555  53
C  29378  isc-net-0000     4  46.101.109.138   192.33.4.12      33751  53
X  29378  isc-socket-0     4  46.101.109.138   192.33.4.12      43555  53
X  29378  isc-socket-0     4  46.101.109.138   192.33.4.12      33751  53
C  29378  isc-net-0000     4  46.101.109.138   193.0.14.129     38145  53
C  29378  isc-net-0000     4  46.101.109.138   192.33.14.30     40905  53
X  29378  isc-socket-0     4  46.101.109.138   193.0.14.129     38145  53
X  29378  isc-socket-0     4  46.101.109.138   192.33.14.30     40905  53
```

BEISPIEL: TCPCONNLAT

- `tcpconnlat` gibt die Latenz einer TCP-basierten Verbindung aus, hier eine ausgehende DNS Anfrage über TCP eines BIND 9 resolvers (in Beispiel eine Abfrage von `microsoft.com txt`, wobei die Antwort zu groß für ein 1232 Byte UDP Paket ist)
  - `isc-net-0000` ist der interne Name des BIND 9 Threads

```
# tcpconnlat-bpfcc
PID    COMM         IP SADDR            DADDR            DPORT LAT(ms)
29378  isc-net-0000 4  46.101.109.138   193.0.14.129     53    37.50
29378  isc-net-0000 4  46.101.109.138   192.52.178.30    53    14.01
29378  isc-net-0000 4  46.101.109.138   199.9.14.201     53    8.48
29378  isc-net-0000 4  46.101.109.138   192.42.93.30     53    1.90
29378  isc-net-0000 4  46.101.109.138   40.90.4.205      53    14.27
29378  isc-net-0000 4  46.101.109.138   199.254.48.1     53    19.21
29378  isc-net-0000 4  46.101.109.138   192.48.79.30     53    7.66
29378  isc-net-0000 4  46.101.109.138   192.41.162.30    53    7.97
29396  isc-net-0000 4  127.0.0.1        127.0.0.1        53    0.06
```

BEISPIEL: UDPLIFE

- `udplife` ist ein `bpftrace` um die UDP Roundtrip-Time (hier DNS round trip time) einer UDP Kommunikation auszugeben (Programm von Brendan Gregg, siehe Links)

```
# udplife.bt
Attaching 8 probes...
PID    COMM       LADDR           LPORT RADDR           RPORT   TX_B   RX_B MS
29378 isc-net-00 46.101.109.138  0     199.19.57.1     16503     48    420 268
29378 isc-net-00 46.101.109.138  0     51.75.79.143    81        49     43 13
29378 isc-net-00 46.101.109.138  0     199.6.1.52      16452     48    408 24
29378 isc-net-00 46.101.109.138  0     199.249.120.1   81        44     10 9
29378 isc-net-00 46.101.109.138  0     199.254.31.1    32891     64     30 273
29378 isc-net-00 46.101.109.138  0     65.22.6.1       32891     64     46 266
```

BEISPIELE FÜR DEN EINSATZ VON EBPF: "SERVER AGNOSTIC DNS AUGMENTATION USING EBPF"

- Eine master thesis von Tom Carpay (mit Unterstützung von NLnet Labs)
- eBPF Funktionen welche von Tom Carpay getestet wurden
  - Umschreiben von DNS Query-Names via eBPF
  - DNS Response-Rate Limiting im Linux Kernel unabhängig von DNS Server Produkt
- https://www.nlnetlabs.nl/downloads/publications/DNS-augmentation-with-eBPF.pdf

BIND 9 INSTRUMENTIEREN

BEISPIEL: LOGGING VON DNS FORWARDING-ENTSCHEIDUNGEN

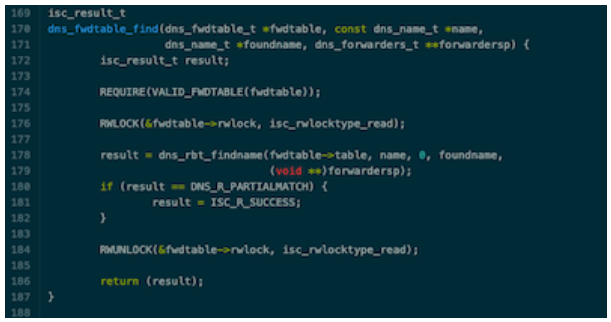
- Ein BIND 9 DNS Resolver mit einer Forward-Zone konfiguriert.

```
zone "dnslab.org" {
        type forward;
        forwarders { 1.1.1.1; 8.8.8.8; };
};
```

- Das BIND 9 Logging System ist sehr mächtig, hat jedoch keine Funktion um DNS Forwarding Entscheidungen zu loggen
- Ziel: Ein `bpftrace` Skript erstellen um BIND 9 DNS Forwarding-Entscheidungen auszugeben

SCHRITT 1 - USE THE ~~FORCE~~ SOURCE

- Der BIND 9 Quellcode ist auf dem ISC Github Server offen verfügbar und durchsuchbar: https://gitlab.isc.org
- Eine Suche im BIND 9 Quellcode nach *forwarding* findet die Funktion `dns_fwdtable_find` in `/lib/dns/forward.c`. Das schaut erfolgversprechend aus:



```
isc_result_t
dns_fwdtable_find(dns_fwdtable_t *fwdtable, const dns_name_t *name,
                  dns_name_t *foundname, dns_forwarders_t **forwardersp) {
        isc_result_t result;

        REQUIRE(VALID_FWDTABLE(fwdtable));

        RWLOCK(&fwdtable->rwlock, isc_rwlocktype_read);

        result = dns_rbt_findname(fwdtable->table, name, 0, foundname,
                                  (void **)forwardersp);
        if (result == DNS_R_PARTIALMATCH) {
                result = ISC_R_SUCCESS;
        }

        RWUNLOCK(&fwdtable->rwlock, isc_rwlocktype_read);

        return (result);
}
```

SCHRITT 2 - EIN PROOF-OF-CONCEPT TEST

- Die Funktion `dns_fwdtable_find` nimmt als Eingangsparameter einen Domain-Namen und liefert den Wert `0` wenn der Namen via Forwarding aufgelöst werden muss, und einen Wert > 0 wenn Forwarding nicht verwendet wird
    - Ein `bpftrace` one-liner gibt uns die Information ob diese Funktion für diese Aufgabe benutzbar ist:

```
bpftrace -e 'uretprobe:/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so:dns_fwdtable_find { print(retval) }'
```

SCHRITT 2 - EIN PROOF-OF-CONCEPT TEST

```
root@ebpf-test:~# bpftrace -e 'uretprobe:/lib/x86_64-linux-gnu/libd
16.22-Debian.so:dns_fwdtable_find { print(retval) }'
Attaching 1 probe...
0
23
23
23
23
23
23
23
```

```
root@ebpf-test:~# dig @localhost ns200a.dnslab.org +short
167.172.136.154
root@ebpf-test:~# dig @localhost isc.org +short
149.20.1.66
root@ebpf-test:~#
```

SCHRITT 3 - DAS BPFTRACE SKRIPT PLANEN

- Nun da wir die Funktion für die Aufgabe gefunden und verifiziert haben können wir ein bpftrace Skript schreiben
- Das Skript wird
  - Den Domain Namen, welcher der Funktion `dns_fwdtable_find` übergeben wird, beim Eintritt in die Funktion speichern
  - Den Rückgabewert der Funktion (`retval`) auf den Wert Null (`0`) prüfen und den Domain Namen ausgeben wenn Forwarding benutzt wird

HERAUSFORDERUNG - KAMPF MIT STRUCTS

- Der Domain-Name welcher auf Forwarding geprüft werden soll wird der Funktion als Datenstruktur (struct) vom Typ `dns_name_t` übergeben
  - Es ist leider kein einfacher Zeiger auf eine Zeichenkette, welche wir ausdrucken können
- Eine Suche durch die Dokumentation des ISC BIND 9 Quellcodes findet die Datenstruktur `dns_name_t`. Das 2te Feld ist ein `unsigned char * ndata`, dies scheint der Domain-Name zu sein

HERAUSFORDERUNG - KAMPF MIT STRUCTS

- Die Definition des Datenstruktur `dns_name_t` befindet sich in der Datei `lib/dns/include/dns/name.h`

```
 96
 97  /*%
 98   * Clients are strongly discouraged from using this type directly,  with
 99   * the exception of the 'link' and 'list' fields which may be used directly
100   * for whatever purpose the client desires.
101   */
102  struct dns_name {
103          unsigned int   magic;
104          unsigned char *ndata;
105          unsigned int   length;
106          unsigned int   labels;
107          unsigned int   attributes;
108          unsigned char *offsets;
109          isc_buffer_t  *buffer;
110          ISC_LINK(dns_name_t) link;
111          ISC_LIST(dns_rdataset_t) list;
112  };
```

HERAUSFORDERUNG - KAMPF MIT STRUCTS

- `bpftrace` beutzt eine der Programmiersprache C ähnliche Syntax, daher können wir die Definition der Datenstruktur aus dem BIND 9 Quellcode direkt in das `bpftrace` Skript importieren
    - Die verkettete Liste und das Feld `isc_buffer_t` wird für unser Skript nicht benötigt und da diese Felder keine eingebauten Datentypen beschreiben kommentieren wir diese aus:

```
#!/usr/bin/bpftrace

struct dns_name {
        unsigned int   magic;
        unsigned char *ndata;
        unsigned int   length;
        unsigned int   labels;
        unsigned int   attributes;
        unsigned char *offsets;
//      isc_buffer_t  *buffer;
//      ISC_LINK(dns_name_t) link;
//      ISC_LIST(dns_rdataset_t) list;
};
[...]
```

EINEN TEXT BEIM START DES SKRIPTS AUSGEBEN

- Die `BEGIN` Pseudo-Probe wird beim Start des Skriptes aktiv und gibt eine Nachricht auf das Terminal aus um dem Benutzer darüber zu informieren das das Skript erfolgreich gestartet wurde

```
[...]
BEGIN
{
  print("Waiting for forward decision...\n");
}
[...]
```

DEN FUNKTIONSAUFRUF ÜBERWACHEN

- Diese Probe wird aktiv wenn die Funktion im BIND 9 aufgerufen wird
  - Es ist eine `uprobe` (User-Space Entry-Probe)
  - Die Probe instrumentalisiert die Funktion `dns_fwdtable_find` in der dynamischen Bibliotheks-Datei `/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so`
  - Das 2te Argument des Funktionsaufrufs (`arg1`) wird in ein `struct dns_name` gecastet und das Feld `ndata` referenziert
  - Der Inhalt des Feldes wird in der Variable `@dns_name[tid]` (indiziert mit der Thread ID (`tid`) des laufenden BIND 9 Threads im Prozess) gespeichert

```
[...]
uprobe:/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so:dns_fwdtable_find
{
  @dns_name[tid] = ((struct dns_name *)arg1)->ndata
}
[...]
```

DEN RÜCKSPRUNG AUS DER FUNKTION ÜBERWACHEN

- Die 3te Probe wird beim Verlassen der Funktion aktiv (`uretprobe` - User-space Funktion *Return* Probe)
  - Gleiche Bibliotheks-Datei und Funktion wie zuvor
- Ist der Rückgabewert der Dunktion Null `0` (Domain Name muss über Forwarding aufgelöst werden) wird der Wert der Variable `@dns_name[tid]` in eine Zeichenkette gewandelt und ausgegeben
- Die Variable `@dns_name[tid]` wird nicht weiter benötigt und gelöscht

```
uretprobe:/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so:dns_fwdtable_find
{
 if (retval == 0) {
    printf("Forwarded domain name: %s\n", str(@dns_name[tid]));
 }
 delete(@dns_name[tid]);
}
```

DAS VOLLSTÄNDIGE SKRIPT

```
#!/usr/bin/bpftrace

struct dns_name {
        unsigned int   magic;
        unsigned char *ndata;
        unsigned int   length;
        unsigned int   labels;
        unsigned int   attributes;
        unsigned char *offsets;
//      isc_buffer_t  *buffer;
//      ISC_LINK(dns_name_t) link;
//      ISC_LIST(dns_rdataset_t) list;
};

BEGIN
{
  print("Waiting for forward decision...\n");
}
uprobe:/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so:dns_fwdtable_find
{
  @dns_name[tid] = ((struct dns_name *)arg1)->ndata
}

uretprobe:/lib/x86_64-linux-gnu/libdns-9.16.22-Debian.so:dns_fwdtable_find
{
 if (retval == 0) {
    printf("Forwarded domain name: %s\n", str(@dns_name[tid]));
 }
 delete(@dns_name[tid]);
}
```

THE SKRIPT BEI DER ANWENDUNG

- Immer wenn ein Domain-Namen im BIND 9 Resolver aufgelöst wird, wird auch das `bpftrace` Skript aktiv
  - In diesem Beispiel werden alle Anfragen an die Domain `dnslab.org` per Forwarding weitergeleitet, jedoch nicht die Anfragen an `ietf.org`

```
root@ebpf-test:~# ./forward.bt
Attaching 3 probes...
Waiting for forward decision...

Forwarded domain name: zone203dnslaborg
Forwarded domain name: dnslaborg
Forwarded domain name: dnslaborg
Forwarded domain name: dnslaborg
Forwarded domain name: dnslaborg

root@ebpf-test:~# dig zone203.dnslab.org +short
137.184.150.214
root@ebpf-test:~# dig ietf.org +short
4.31.198.44
root@ebpf-test:~#
```

PAKET FILTER MIT EBPF

EBPF ALS NETZWERK FIREWALL

- Mittels eBPF können sehr effiziente Firewall-Systeme gebaut werden
  - eBPF kann Netzwerk-Verkehr stoppen, bevor dieser den Linux TCP/IP Stack oder die Anwendung erreicht
  - Da eBPF pro Netzwerk-Paket ein volles Programm ausführt, können komplexe Filter definiert werden
    - Filter auf Basis von DNS Query Namen
    - DNSSEC Daten in der Antwort verfügbar?
    - Quell-IP des antwortenden autoritativen DNS Servers (blockieren bekannter "bösartiger" DNS Server)
    - EDNS data (DNS Pakete mit DNS Cookies bevorzugen)
    - …

XDP FIREWALL

- Die XDP Firewall ist ein neues Projekt welches mittels XDP und eBPF eine generische Firewall erzeugt
  - Source-Code https://github.com/gamemann/XDP-Firewall
  - Eine Beispiel-Regel um DNS Pakete auf Port 53 zu blocken

```
interface = "eth0";
updatetime = 15;

filters = (
    {
        enabled = true,
        action = 0,
        udp_enabled = true,
        udp_dport = 53
    }
);
```

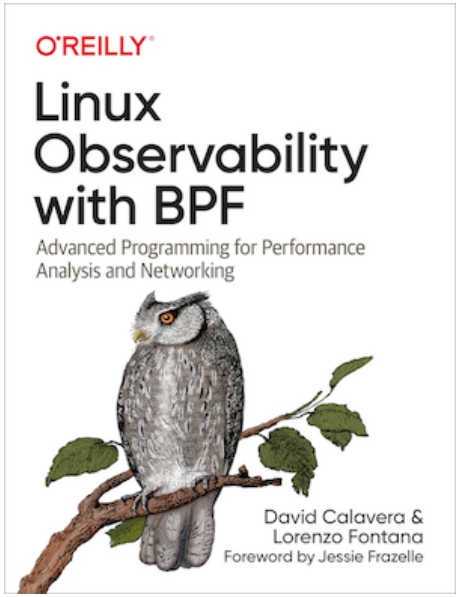
EBPF FIREWALL BEISPIEL: BLOCK-NON-DNS

- Unser Hands-One Beispiel heute zeigt einen simplen eBPF Netzwerk-Filter
  - Der Filter blockiert jeglichen UDP Verkehr an ein Interface (hier das Loopback-Interface) mit der Ausnahme von UDP DNS Paketen (Port 53)
  - Dies hilft bei der Abwehr von non-DNS DDoS Angriffen auf einem autoritativen DNS Resolver

LITERATUR

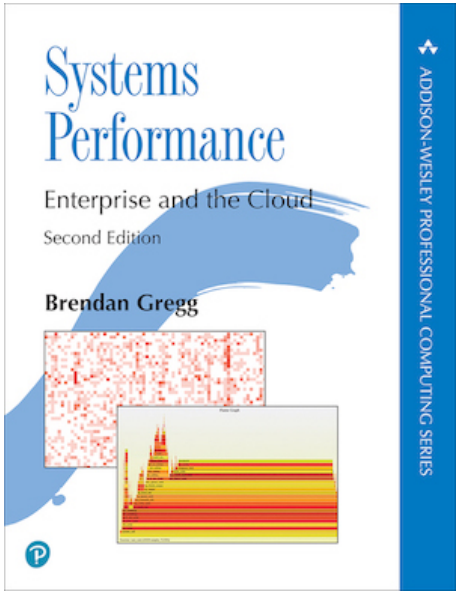
BUCH: LINUX OBSERVABILITY WITH BPF

Von David Calavera, Lorenzo Fontana (November 2019)



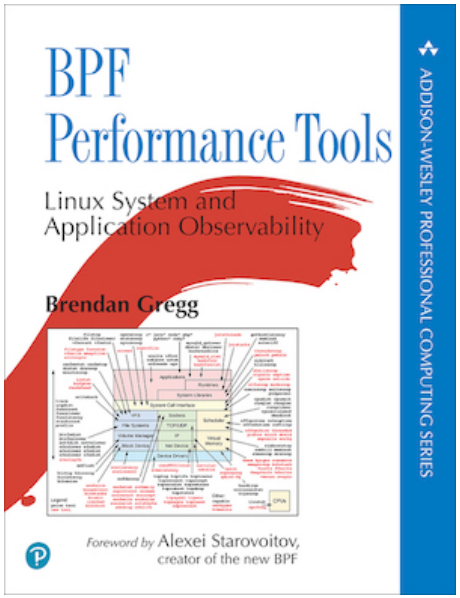
BUCH: SYSTEMS PERFORMANCE (2ND ED.)

Von Brendan Gregg (Dezember 2020)



BUCH: BPF PERFORMANCE TOOLS

Von Brendan Gregg (Dezember 2019)



LINKS

EBPF

- eBPF support and Linux kernel versions https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md
- Awesome BPF https://github.com/zoidbergwill/awesome-ebpf
- Using user-space tracepoints with BPF https://lwn.net/Articles/753601/
- Extending systemd Security Features with eBPF https://kinvolk.io/blog/2021/04/extending-systemd-security-features-with-ebpf/
- Absolute Beginner's Guide to BCC, XDP, and eBPF https://dev.to/satrobit/absolute-beginner-s-guide-to-bcc-xdp-and-ebpf-47oi

EBPF

- Linux Extended BPF (eBPF) Tracing Tools
  https://www.brendangregg.com/ebpf.html
- Performance Implications of Packet Filtering with Linux eBPF
  https://www.net.in.tum.de/fileadmin/bibtex/publications/pa
  Packet-Filtering-eBPF-XDP.pdf
- eBPF for perfomance analysis and networking
  https://marioskogias.github.io/students/debeule.pdf
    - BPF and XDP Reference Guide https://docs.cilium.io/en/v1.10/bpf/

BCC

- Intro to Kernel and Userspace Tracing Using BCC, Part 1 of 3 https://blogs.oracle.com/linux/post/intro-to-bcc-1

BPFTRACE

- bpftrace Reference Guide https://github.com/iovisor/bpftrace/blob/master/docs/refer
- Kernel analysis with bpftrace https://lwn.net/Articles/793749
- The bpftrace One-Liner Tutorial https://github.com/iovisor/bpftrace/blob/master/docs/tutor
- Full-system dynamic tracing on Linux using eBPF and bpftrac https://www.joyfulbikeshedding.com/blog/2019-01-31-full-s tracing-on-linux-using-ebpf-and-bpftrace.html
- bpftrace Cheat Sheet https://www.brendangregg.com/BPF/b sheet.html

NETWORK SCRIPTS

- udplife https://github.com/brendangregg/bpf-perf-tools-book/blob/master/exercinses/Ch10_Networking/udplife.bt
- How SKBs work http://vger.kernel.org/~davem/skb_data.html

EBPF PROMETHEUS EXPORTER

- eBPF exporter https://blog.cloudflare.com/introducing-ebpf_exporter/

EXPRESS DATA PATH (XDP)

- Introduction to: XDP and BPF building blocks https://people.netfilter.org/hawk/presentations/ebplane201
- A practical introduction to XDP https://www.linuxplumbersconf.org/event/2/contributions/7 lpc2018-xdp-tutorial.pdf
- eBPF/XDP https://www.slideshare.net/Netronome/ebpfxdp-s
- XDP Packet filter and UDP https://fly.io/blog/bpf-xdp-packet-
- XDP Firewall https://github.com/gamemann/XDP-Firewall

EXPRESS DATA PATH (XDP)

- How to filter packets super fast: XDP & eBPF! https://jvns.ca/blog/2017/04/07/xdp-bpf-tutorial/
- Load XDP programs using the ip (iproute2) command https://medium.com/@fntlnz/load-xdp-programs-using-the-ip-iproute2-command-502043898263
- L4Drop: XDP DDoS Mitigations https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/
- How to drop a packet in Linux in more ways than one https://codilime.com/blog/how-to-drop-a-packet-in-linux-in-more-ways-than-one/
- eBPFsnitch https://github.com/harporoeder/ebpfsnitch
- XDP minimal example https://ruderich.org/simon/notes/xdp-minimal-example
- Why is the kernel community replacing iptables with BPF? https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables

VIELEN DANK!

Kontakt:

carsten@dnsworkshop.de