aqua

Towards truly portable eBPF

Itay Shakury & Rafael D. Tinoco
OSS @ Aqua Security

Linux Plumbers 2021

@itaysk
@rafaeldtinoco

Hello

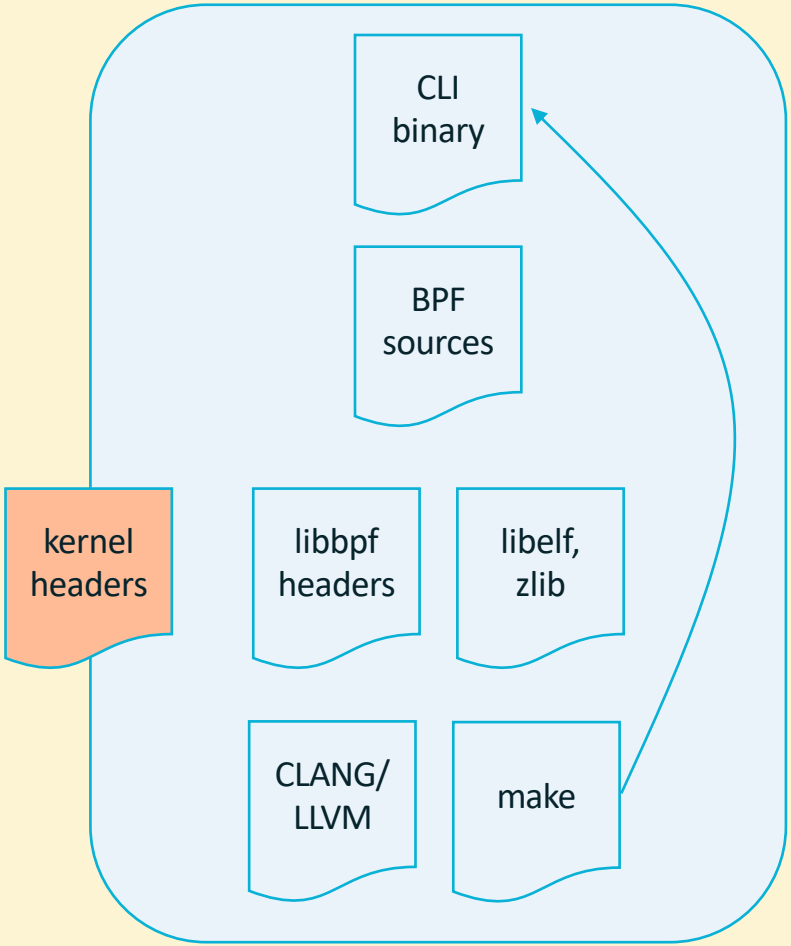
- Tracee – runtime security using eBPF
- Tell our story of building and shipping eBPF application
- Our POV: vendor not user, targeting common users
- Go -> eBPF
- User experience > developer productivity

```
$ docker run aquasec/tracee
```

aqua tracee

github.com/aquasecurity/tracee

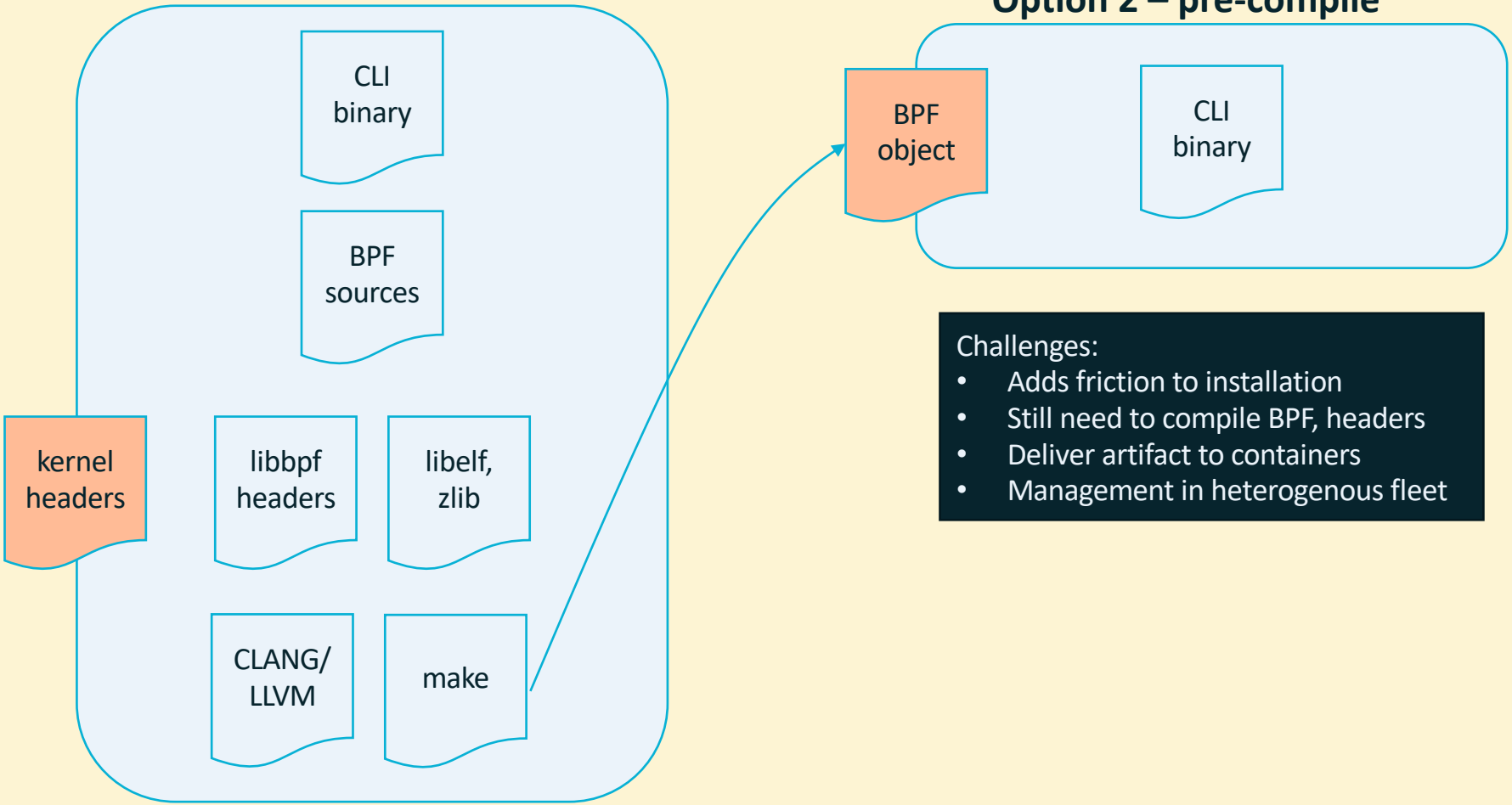
Option 1 – all in one image



Challenges:

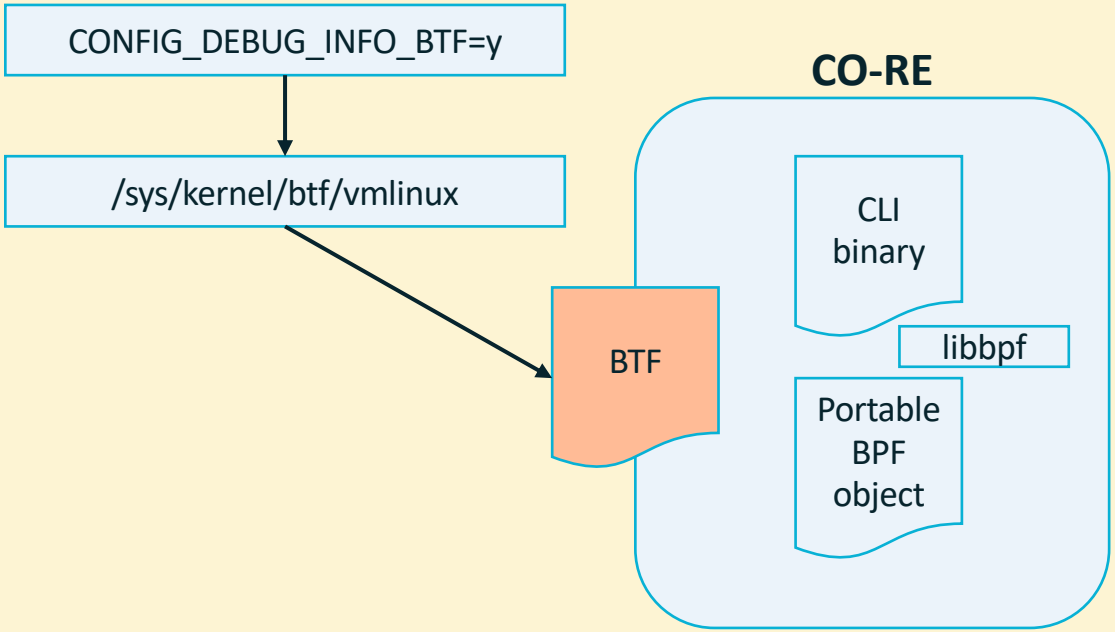
- Long startup time
- Big image (~155MB)
- Obtain correct headers
- Fragile header discovery

Option 2 – pre-compile



Challenges:

- Adds friction to installation
- Still need to compile BPF, headers
- Deliver artifact to containers
- Management in heterogenous fleet



Challenges:

- Portability aches
- libbpf in Go
- BTF prevalence

https://nakryiko.com/posts/bpf-portability-and-co-re/

```go
func main() {
  eventsChannel := make(chan []byte, 0)
  sig := make(chan os.Signal, 1)
  signal.Notify(sig, os.Interrupt)

  // initialize
  m, err := libbpfgo.NewModuleFromFile("myprobe.bpf.o")
  must(err)
  m.BPFLoadObject()
  prog, err := m.GetProgram("execve_handler")
  must(err)
  _, err = prog.AttachTracepoint("syscalls:sys_enter_execve")
  must(err)
  rb, _ := m.InitRingBuf("events", eventsChannel)

  // start
  fmt.Println("starting")
  rb.Start()
  go func() {
    for e := range eventsChannel {
      x, err := binary.ReadUvarint(bytes.NewReader(e))
      must(err)
      fmt.Printf("event: %v\n", x)
    }
  }()

  // wait
  <-sig
  fmt.Println("stopping")
  rb.Stop()
  m.Close()
}
```

aquasecurity / libbpfgo Public

Unwatch 12 | Unstar 92 | Fork 20

Code | Issues 20 | Pull requests 3 | Discussions | Actions | Projects | Wiki | Security | Insights

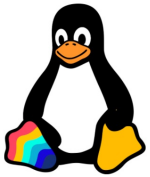
main | 4 branches | 4 tags | Go to file | Add file | Code

rafaeldtinoco libbpf: update submodule to v0.5.0 ef82a0b 7 days ago 103 commits

.github/workflows	Add back github action pr-libbpfgo workflow	4 months ago
docs	helpers/kernel_config: rename const to CUSTOM_OPTION_S...	9 days ago
helpers	helpers/kernel_config: rename const to CUSTOM_OPTION_S...	9 days ago
libbpf @ 5579664	libbpf: update submodule to v0.5.0	7 days ago
selftest	selftest: update all go.mod to v0.2.1-libbpf-0.4.0	7 days ago
.gitignore	examples: Add tcpconnect as a libbpfgo example (#45)	2 months ago
.gitmodules	Makefile: improvements (#28)	2 months ago
LICENSE	Create LICENSE	4 months ago
Makefile	examples: Add tcpconnect as a libbpfgo example (#45)	2 months ago
Readme.md	Update readme for semantic versioning (#71)	16 days ago
go.mod	Fix go module files to use new libbpfgo repository/module	4 months ago
go.sum	Fix go module files to use new libbpfgo repository/module	4 months ago
libbpf_cb.go	Fix eventsChannels race	6 months ago
libbpfgo.go	Add BPFLink.Destroy (#69)	16 days ago
libbpfgo_test.go	Makefile: improvements (#28)	2 months ago

Readme.md

libbpfgo



About

eBPF library for Go, wrapping libbpf

go linux golang ebpf bpf

Readme

Apache-2.0 License

Releases 4

v0.2.1-libbpf-0.4.0 Latest
16 days ago

+ 3 releases

Used by 54

+ 46

Contributors 8

Languages

Go 75.5% | C 12.8% | Makefile 8.3% | Shell 3.4%

BTF in the wild

BPF CO-RE (Compile Once – Run Everywhere)

Libbpf supports building BPF CO-RE-enabled applications, which, in contrast to BCC, do not require Clang/LLVM runtime being deployed to target servers and doesn't rely on kernel-devel headers being available.

It does rely on kernel to be built with BTF type information, though. Some major Linux distributions come with kernel BTF already built in:

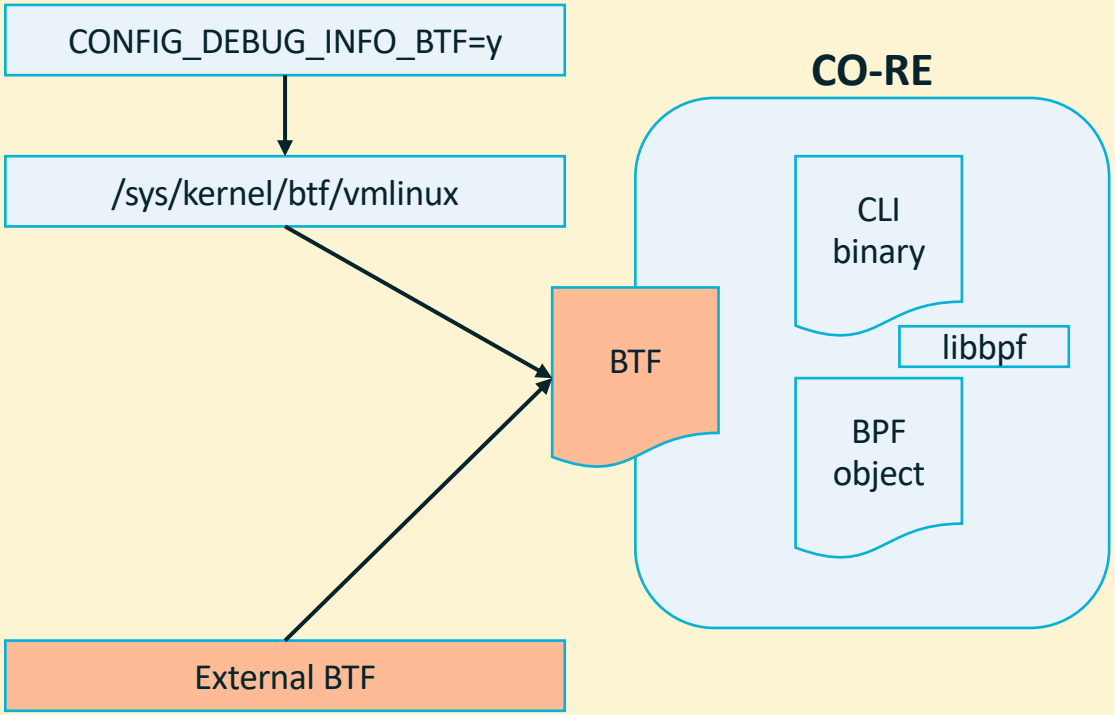
- Fedora 31+
- RHEL 8.2+
- OpenSUSE Tumbleweed (in the next release, as of 2020-06-04)
- Arch Linux (from kernel 5.7.1.arch1-1)
- Manjaro (from kernel 5.4 if compiled after 2021-06-18)
- Ubuntu 20.10
- Debian 11 (amd64/arm64)

If your kernel doesn't come with BTF built-in, you'll need to build custom kernel. You'll need:

- `pahole` 1.16+ tool (part of `dwarves` package), which performs DWARF to BTF conversion;
- kernel built with `CONFIG_DEBUG_INFO_BTF=y` option;
- you can check if your kernel has BTF built-in by looking for `/sys/kernel/btf/vmlinux` file:

```
$ ls -la /sys/kernel/btf/vmlinux
-r--r--r--. 1 root root 3541561 Jun  2 18:16 /sys/kernel/btf/vmlinux
```

https://github.com/libbpf/libbpf#bpf-co-re-compile-once--run-everywhere


CONFIG_DEBUG_INFO_BTF=y
/sys/kernel/btf/vmlinux
External BTF
BTF
CO-RE
CLI binary
libbpf
BPF object

How to load external BTF?

```
struct btf *btf__load_vmlinux_btf(void)
{
        struct {
                const char *path_fmt;
                bool raw_btf;
        } locations[] = {
                /* try canonical vmlinux BTF through sysfs first */
                { "/sys/kernel/btf/vmlinux", true /* raw BTF */ },
                /* fall back to trying to find vmlinux ELF on disk otherwise */
                { "/boot/vmlinux-%1$s" },
                { "/lib/modules/%1$s/vmlinux-%1$s" },
                { "/lib/modules/%1$s/build/vmlinux" },
                { "/usr/lib/modules/%1$s/kernel/vmlinux" },
                { "/usr/lib/debug/boot/vmlinux-%1$s" },
                { "/usr/lib/debug/boot/vmlinux-%1$s.debug" },
                { "/usr/lib/debug/lib/modules/%1$s/vmlinux" },
        };
```

```
commit 1373ff599556
Author: Shuyi Cheng <chengshuyi@linux.alibaba.com>
Date:   Tue Jul 13 09:42:37 2021

    libbpf: Introduce 'btf_custom_path' to 'bpf_obj_open_opts'
```

```
struct bpf_object_open_opts {
        ...
        const char *kconfig;
        /* Path to the custom BTF to be used for BPF CO-RE relocations.
         * This custom BTF completely replaces the use of vmlinux BTF
         * for the purpose of CO-RE relocations.
         * NOTE: any other BPF feature (e.g., fentry/fexit programs,
         * struct_ops, etc) will need actual kernel BTF at /sys/kernel/btf/vmlinux.
         */
        const char *btf_custom_path;
};
#define bpf_object_open_opts__last_field btf_custom_path
```

```
func NewModuleFromBufferArgs(args NewModuleArgs) (*Module, error) {
    C.set_print_fn()
    if err := bumpMemlockRlimit(); err != nil {
        return nil, err
    }
    if args.BTFObjPath == "" {
        args.BTFObjPath = "/sys/kernel/btf/vmlinux"
    }
    btfFile := C.CString(args.BTFObjPath)
    bpfName := C.CString(args.BPFObjName)
    bpfBuff := unsafe.Pointer(C.CBytes(args.BPFObjBuff))
    bpfBuffSize := C.size_t(len(args.BPFObjBuff))

    opts := C.struct_bpf_object_open_opts{}
    opts.object_name = bpfName
    opts.sz = C.sizeof_struct_bpf_object_open_opts
    opts.btf_custom_path = btfFile // instruct libbpf to use user provided kern

    if len(args.KConfigFilePath) > 2 {
        kConfigFile := C.CString(args.KConfigFilePath)
        opts.kconfig = kConfigFile // instruct libbpf to use user provided KCon
        defer C.free(unsafe.Pointer(kConfigFile))
    }
```

How to generate a BTF?

```
[user@host:~]$ pahole --help
Usage: pahole [OPTION...] FILE

  -a, --anon_include         include anonymous classes
  -A, --nested_anon_include  include nested (inside other structs) anonymous
                             classes
      --btf_base=PATH        Path to the base BTF file
      --btf_encode_force     Ignore those symbols found invalid when encoding
                             BTF.
      --btf_gen_all          Allow using all the BTF features supported by
                             pahole.
      --btf_gen_floats       Allow producing BTF_KIND_FLOAT entries.
  -B, --bit_holes=NR_HOLES   Show only structs at least NR_HOLES bit holes
  -c, --cacheline_size=SIZE  set cacheline size to SIZE
      --classes_as_structs   Use 'struct' when printing classes
      --count=COUNT          Print only COUNT input records
  -C, --class_name=CLASS_NAME   Show just this class
  -d, --recursive            recursive mode, affects several other flags
  -D, --decl_exclude=PREFIX  exclude classes declared in files with PREFIX
  -E, --expand_types         expand class members
  -f, --find_pointers_to=CLASS_NAME
                             Find pointers to CLASS_NAME
      --first_obj_only       Only process the first object file in the binary
      --fixup_silly_bitfields   Fix silly bitfields such as int foo:32
      --flat_arrays          Flat arrays
  -F, --format_path=FORMAT_LIST   List of debugging formats to try
      --header_type=TYPE     File header type
      --hex                  Print offsets and sizes in hexadecimal
  -H, --holes=NR_HOLES       show only structs with at least NR_HOLES holes
  -i, --contains=CLASS_NAME  Show classes that contains CLASS_NAME
  -I, --show_decl_info       Show the file and line number where the tags were
                             defined
  -j, --btf_encode_detached=FILENAME
                             Encode as BTF in a detached file
  -J, --btf_encode           Encode as BTF
      --kabi_prefix=STRING   When the prefix of the string is STRING, treat the
                             string as STRING.
  -l, --show_first_biggest_size_base_type_member
                             show first biggest size base_type member
  -m, --nr_methods           show number of methods
  -M, --show_only_data_members   show only the members that use space in the
                             class layout
```

```
# generate .BTF typeinfo from DWARF debuginfo
# ${1} - vmlinux image
# ${2} - file to dump raw BTF data into
gen_btf()
{
        local pahole_ver
        local extra_paholeopt=

        if ! [ -x "$(command -v ${PAHOLE})" ]; then
                echo >&2 "BTF: ${1}: pahole (${PAHOLE}) is not available"
                return 1
        fi

        pahole_ver=$(${PAHOLE} --version | sed -E 's/v([0-9]+)\.([0-9]+)/\1\2/')
        if [ "${pahole_ver}" -lt "116" ]; then
                echo >&2 "BTF: ${1}: pahole version $(${PAHOLE} --version) is too old, need at le
                return 1
        fi

        vmlinux_link ${1}

        if [ "${pahole_ver}" -ge "118" ] && [ "${pahole_ver}" -le "121" ]; then
                # pahole 1.18 through 1.21 can't handle zero-sized per-CPU vars
                extra_paholeopt="${extra_paholeopt} --skip_encoding_btf_vars"
        fi
        if [ "${pahole_ver}" -ge "121" ]; then
                extra_paholeopt="${extra_paholeopt} --btf_gen_floats"
        fi

        info "BTF" ${2}
        LLVM_OBJCOPY="${OBJCOPY}" ${PAHOLE} -J ${extra_paholeopt} ${1}

        # Create ${2} which contains just .BTF section but no symbols. Add
        # SHF_ALLOC because .BTF will be part of the vmlinux image. --strip-all
        # deletes all symbols including __start_BTF and __stop_BTF, which will
        # be redefined in the linker script. Add 2>/dev/null to suppress GNU
        # objcopy warnings: "empty loadable segment detected at ..."
        ${OBJCOPY} --only-section=.BTF --set-section-flags .BTF=alloc,readonly \
                --strip-all ${1} ${2} 2>/dev/null
        # Change e_type to ET_REL so that it can be used to link final vmlinux.
        # Unlike GNU ld, lld does not allow an ET_EXEC input.
        printf '\1' | dd of=${2} conv=notrunc bs=1 seek=16 status=none
}
```

BTF Generation Script - Ubuntu

```
# extract vmlinux file from ddeb package
dpkg --fsys-tarfile "${version}.ddeb" | \
    tar xvf - "./usr/lib/debug/boot/vmlinux-${version}" || \
{
    warn "could not deal with ${version}, cleaning and moving on..."
    rm -rf "${basedir}/ubuntu/${ubuntuver}/x86_64/usr"
    rm -rf "${version}.ddeb"
touch "${version}.failed"
    continue
}

mv "./usr/lib/debug/boot/vmlinux-${version}" "./${version}.vmlinux" || \
{
    warn "could not rename vmlinux ${version}, cleaning and moving on..."
    rm -rf "${basedir}/ubuntu/${ubuntuver}/x86_64/usr"
    rm -rf "${version}.ddeb"
touch "${version}.failed"
    continue

}

rm -rf "${basedir}/ubuntu/${ubuntuver}/x86_64/usr"

pahole -j "${version}.btf" "${version}.vmlinux"
# pahole "./${version}.btf" > "${version}.txt"
tar cvfJ "./${version}.btf.tar.xz" "${version}.btf"
```

Search or jump to... / Pull requests Issues Marketplace Explore

aquasecurity / btfhub Public

Unwatch 8 Star 30

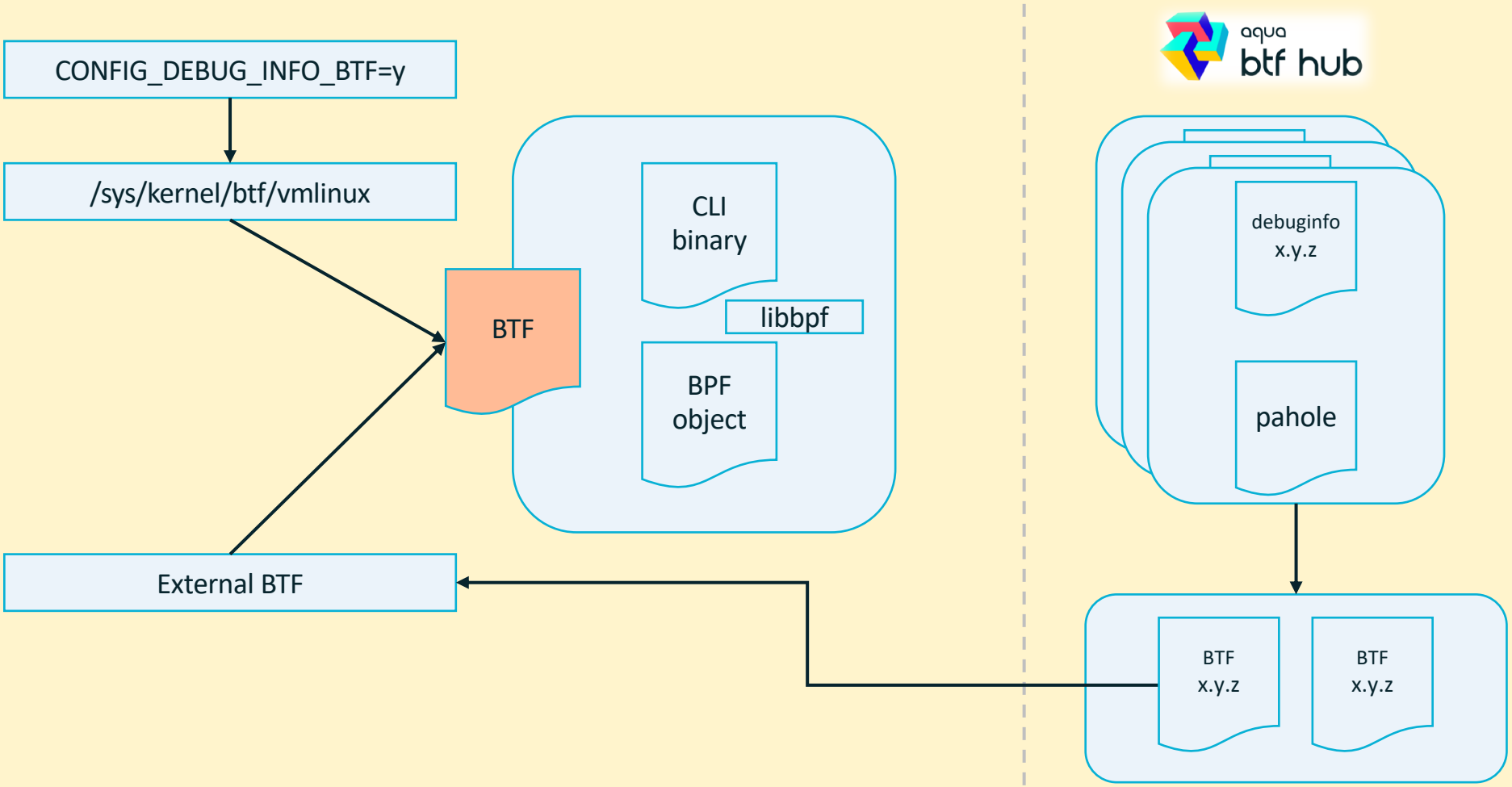
Code Issues 1 Pull requests Actions Projects Wiki Security Insights Settings

main btfhub / ubuntu / 18.04 / x86_64 / Go to file Add file

rafaeldtinoco ubuntu: sync latest bionic BTFs 9f4dc2a 5 days ago

..

4.15.0-1007-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1009-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-101-generic.btf.tar.xz	arch: create x86_64 directory for each of supported distros
4.15.0-1010-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1011-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1016-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1017-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1019-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1020-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1021-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1023-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1025-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1027-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1029-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1031-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1032-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1033-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1034-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1035-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1037-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1039-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1040-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1041-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1043-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1044-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1045-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel



https://github.com/aquasecurity/btfhub

README.md

CentOS

CentOS 7

Centos	RHEL	Release Date	RHEL Date	Kernel	BPF	BTF	HUB
7.0.1406	7.0	2014-07	2014-06-09	3.10.0-123	-	-	Y
7.1.1503	7.1	2015-03	2015-03-05	3.10.0-229	-	-	Y
7.2.1511	7.2	2015-11	2015-11-19	3.10.0-327	-	-	Y
7.3.1611	7.3	2016-11	2016-11-03	3.10.0-514	-	-	Y
7.4.1708	7.4	2017-08	2017-07-31	3.10.0-693	-	-	Y
7.5.1804	7.5	2018-04	2018-04-10	3.10.0-862	-	-	Y
7.6.1810	7.6	2018-10	2018-10-30	3.10.0-957	Y	-	Y
7.7.1908	7.7	2019-08	2019-08-06	3.10.0-1062	Y	-	Y
7.8.2003	7.8	2020-03	2020-03-31	3.10.0-1127	Y	-	Y
7.9.2009	7.9	2020-09	2020-09-29	3.10.0-1160	Y	-	Y

> **Note**: Latest centos7 kernels support BPF, and might support BTF, but they lack some eBPF features. With that, eBPF programs capable of running in those systems are very limited.
>
> Check out eBPF features your code use HERE

CentOS 8

Centos	RHEL	Release Date	RHEL Date	Kernel	BPF	BTF	HUB
8.0.1905	8.0	2019-09-24	2019-05-07	4.18.0-80	-	-	Y
8.1.1911	8.1	2020-01-15	2019-11-05	4.18.0-147	-	-	Y
8.2.2004	8.2	2020-06-15	2020-04-28	4.18.0-193	Y	Y	Y
8.3.2011	8.3	2020-12-07	2020-11-03	4.18.0-240	Y	Y	Y
8.4.2105	8.4	2021-06-03	2021-05-18	4.18.0-305	Y	Y	Y
...	...	...	...	...	Y	Y	Y

> **Note**: **ALL** latest CentOS 8 releases have BPF & BTF support enabled!

CentOS Stream 8

Stream	RHEL	Release Date	RHEL Date	Kernel	BPF	BTF	HUB
8.3	8.3	2021-01-14	2020-11-03	4.18.0-240	Y	Y	-
8.4	8.4	2021-01-14	2020-11-03	4.18.0-240	Y	Y	-

> **Note**: **ALL** CentOS Stream 8 releases have BPF & BTF support enabled

README.md

Alma

Alma	RHEL	Release Date	RHEL Date	Kernel	BPF	BTF	HUB
8.3	8.3	2021-03-30	2020-11-03	4.18.0-240	Y	Y	-
8.4	8.4	2021-05-26	2021-05-18	4.18.0-305	Y	Y	-
...	...	...	...	...	Y	Y	-

> **Note**: **ALL** Alma releases have BPF & BTF support enabled!

Fedora

Fedora	Release Date	Kernel	BPF	BTF	HUB
29	2018-10-30	4.18			Y
30	2019-05-07	5.0			Y
31	2019-10-29	5.3			Y
32	2020-04-28	5.6	Y	Y	-
33	2020-10-27	5.8	Y	Y	-
34	2021-04-27	5.11	Y	Y	-
...	-	-	Y	Y	-

> **Note**: All supported future Fedora releases will have BPF & BTF support enabled.

Ubuntu

Ubuntu Ver	Num	Release Date	Kernel	BPF	BTF	HUB
Bionic	18.04.2	2018-04-26	4.15.0	-	-	-
Bionic HWE	-	-	5.4.0	Y	-	Y
Focal	20.04.2	2020-04-23	5.4.0	Y	-	Y
Focal HWE	-	-	5.8.0	Y	-	Y
Groovy	20.10	2020-10-22	5.8.0	Y	Y	-
Groovy HWE	20.10	-	5.11.0	Y	Y	-
Hirsute	21.04	2021-04-22	5.11.0	Y	Y	-
...	...	...	...	Y	Y	-

> **Notes**: Bionic HWE, Focal and Focal HWE kernels need this HUB. All other future Ubuntu releases will have BPF & BTF support enabled.

Disclaimer

BTF Hub is open, feel free to engage

mauriciovasquezbernal commented on Aug 6 • edited Member

This PR is an experiment that uses https://github.com/aquasecurity/btfhub to run CO-RE based tools in systems without `CONFIG_DEBUG_INFO_BTF=y`.

How does it work?

The entrypoint script tries to download the BTF file for the current kernel, if it's successful it creates an ELF file with a .BTF section containing the BTF debug info and stores it at `/boot/vmlinux-$(uname -r)`

Testing done

I created a test cluster in ubuntu focal (using kubeadm) and tried to run some of the tools.

```
$ cat /boot/config-$(uname -r) | grep BTF
CONFIG_VIDEO_SONY_BTF_MPX=m
# CONFIG_DEBUG_INFO_BTF is not set

$ kubectl -n kube-system logs $PODNAME
OS detected: "Ubuntu 20.04.1 LTS"
Kernel detected: 5.4.0-80-generic
bcc detected: 0.21.0-1
Gadget image: docker.io/kinvolk/gadget:mauricio-btf-hub-poc
Deployment options:
INSPEKTOR_GADGET_OPTION_TRACELOOP_LOGLEVEL=info,json
INSPEKTOR_GADGET_OPTION_TRACELOOP=false
INSPEKTOR_GADGET_OPTION_TOOLS_MODE=auto
INSPEKTOR_GADGET_OPTION_HOOK_MODE=auto
Inspektor Gadget version: v0.2.1-115-g16a413c-dirty
Falling back to podinformer hook.
BTF is not available: Trying btfhub
Trying to download vmlinux from https://github.com/aquasecurity/btfhub/raw/main/ubuntu/20.04/5.4.0-80-gener
vmlinux downloaded. Using CO-RE based tools
Starting the Gadget Tracer Manager in the background...
Ready.
time="2021-08-11T01:17:18Z" level=info msg="Creating BPF map: /sys/fs/bpf/gadget/containers"
time="2021-08-11T01:17:18Z" level=info msg="Serving on gRPC socket /run/gadgettracermanager.socket"
time="2021-08-11T01:17:18Z" level=info msg="Starting Pod controller"
time="2021-08-11T01:17:18Z" level=info msg="starting trace controller manager

$ ./kubectl-gadget-linux-amd64 execsnoop
NODE            NAMESPACE        PODNAME          CONTAINERNAME   PCOMM           PID    PPID   RET ARGS
ubuntu-focal    default          mypod            mypod           cat             55050  29204    0 /bin/
```

https://github.com/kinvolk/inspektor-gadget/pull/221

CO-RE: Challenges

- ***PORTABILITY***
  - Kernel memory access
  - Diff stack sizes
  - Loop unrolling & complexity
  - Tail calls
- ***LIBBPF SUPPORT***
  - Destroy vs Detach
  - Missing legacy kprobes support
  - Destroy/detach changes
- ***BTF RELOCATIONS***
  - Quick Overview
    - BPF ELF Section Headers
    - BPF ELF Symbols Table
  - Kconfig file dependency
    - Kconfig relocations

aqua

CHALLENGE: PORTABILITY

(CO-RE and different kernel versions)

CO-RE: Challenges (portability: kernel memory access)

1. *LIBBPF NON-CO-RE*
   - bpf_**probe**_read(&pid, sizeof(pid), &task->pid);
2. *LIBBPF NON-CO-RE + BPF_PROG_TYPE_TRACING* (v5.4-rc3)
   - pid_t pid = **task->pid;**
3. ***LIBBPF CO-RE*** (same as bpf_probe_read() with __builtin_preserve_access_index())
   - bpf_**core**_read(&pid, sizeof(pid), &task->pid);
4. *LIBBPF CO-RE + BPF_PROG_TYPE_TRACING*
   - **__builtin_preserve_access_index()** LLVM built-in support: Accesses to aggregate data structures (structs, unions, arrays) in the argument will have appropriate CO-RE relocation information generated.
   - pid_t pid = **__builtin_preserve_access_index(({ task->pid; }));**

CO-RE: Challenges (portability: unrolling & complexity)

For the kprobe `security_sb_mount`, the `save_path_to_str_buf()` complexity is too big with the unroll logic + MAX_PATH_COMPONENTS of 80, even on higher kernels (like 5.4 in Ubuntu). Reducing to 64 did NOT help. Reducing to 48 DID help and it worked.

I checked Ubuntu kernel and it contains c04c0d2b968a ("bpf: increase complexity limit and maximum program size") commit with no reversions, which indicates that we might need to either split that logic into tails OR define less than 80 for 5.4.x kernels (if others are good with that number).

Based on the commit:

```
BPF_COMPLEXITY_LIMIT_INSNS is the kernel internal limit
and success to load the program no longer depends on program size,
but on 'smartness' of the verifier only.
```

it might be that the eBPF verifier in older kernels, like 5.4, is not *smart* enough to consider an unroll of 80 iterations, in the path resolution function, a logic less complex than it should.

So, we can do a:

```
// Otherwise, the sky is the limit (complexity limit of 1 million verified instructio
#define MAX_STR_ARR_ELEM      128
#define MAX_ARGS_STR_ARR_ELEM 128
#define MAX_PATH_PREF_SIZE    128
+ #if LINUX_VERSION_CODE < KERNEL_VERSION(5, 5, 0)
+ #define MAX_PATH_COMPONENTS   48
+ #else
#define MAX_PATH_COMPONENTS   80
+ #endif
#define MAX_BIN_CHUNKS        256
#endif
```

or change the defaults. Up to you! This small change fixes the issue for NON CO-RE runs in the Ubuntu 5.4 kernel.

```
static __always_inline int save_path_to_str_buf(buf_t *string_p, const struct path *path)
{
    struct path f_path;
    bpf_probe_read(&f_path, sizeof(struct path), path);
    char slash = '/';
    int zero = 0;
    struct dentry *dentry = f_path.dentry;
    struct vfsmount *vfsmnt = f_path.mnt;
    struct mount *mnt_parent_p;

    struct mount *mnt_p = real_mount(vfsmnt);
    bpf_probe_read(&mnt_parent_p, sizeof(struct mount*), &mnt_p->mnt_parent);

    u32 buf_off = (MAX_PERCPU_BUFSIZE >> 1);
    struct dentry *mnt_root;
    struct dentry *d_parent;
    struct qstr d_name;
    unsigned int len;
    unsigned int off;
    int sz;

    #pragma unroll
    for (int i = 0; i < MAX_PATH_COMPONENTS; i++) {
        mnt_root = get_mnt_root_ptr_from_vfsmnt(vfsmnt);
        d_parent = get_d_parent_ptr_from_dentry(dentry);
        if (dentry == mnt_root || dentry == d_parent) {
            if (dentry != mnt_root) {
                // We reached root, but not mount root - escaped?
                break;
            }
            if (mnt_p != mnt_parent_p) {
                // We reached root, but not global root - continue with mount point path
                bpf_probe_read(&dentry, sizeof(struct dentry*), &mnt_p->mnt_mountpoint);
                bpf_probe_read(&mnt_p, sizeof(struct mount*), &mnt_p->mnt_parent);
                bpf_probe_read(&mnt_parent_p, sizeof(struct mount*), &mnt_p->mnt_parent);
                vfsmnt = &mnt_p->mnt;
                continue;
            }
            // Global root - path fully parsed
            break;
        }
        // Add this dentry name to path
        d_name = get_d_name_from_dentry(dentry);
```

CO-RE: Challenges (portability: tail calls complexity)

```go
for e := range t.eventsToTrace {
    eU32 := uint32(e) // e is int32
    params := eventsParams[e]
    var paramsTypes uint64
    var paramsNames uint64
    for n, param := range params {
        paramsTypes = paramsTypes | (uint64(param.encType) << (8 * n))
        paramsNames = paramsNames | (uint64(param.encName) << (8 * n))
    }
    if err := paramsTypesBPFMap.Update(unsafe.Pointer(&eU32),
        unsafe.Pointer(&paramsTypes)); err != nil {
        return err
    }
    if err := paramsNamesBPFMap.Update(unsafe.Pointer(&eU32),
        unsafe.Pointer(&paramsNames)); err != nil {
        return err
    }
    if e == ExecveEventID || e == ExecveatEventID {
        event, ok := EventsIDToEvent[e]
        if !ok {
            continue
        }
        // execve functions require tail call on syscall enter as they perform extra work
        probFnName := fmt.Sprintf("syscall__%s", event.Name)
        err = t.initTailCall(eU32, "sys_enter_tails", probFnName)
        if err != nil {
            return err
        }
        // err = t.initTailCall(uint32(e), "sys_exit_tails", probFnName) // if ever needed
    }
}
```

```bpf
SEC("raw_tracepoint/sys_enter")
int tracepoint__raw_syscalls__sys_enter(struct bpf_raw_tracepoint_args *ctx)
{
    args_t args_tmp = {};
    int id = ctx->args[1];
    struct task_struct *task = (struct task_struct *)bpf_get_current_task();

    ...

    // call syscall handler, if exists
    // enter tail calls should never delete saved args
    bpf_tail_call(ctx, &sys_enter_tails, id);
    return 0;
}
```

```bpf
SEC("raw_tracepoint/sys_execve")
int syscall__execve(void *ctx)
{
    args_t args = {};
    u8 argnum = 0;

    bool delete_args = false;
    if (load_args(&args, delete_args, SYS_EXECVE) != 0)
        return -1;

    if (!event_chosen(SYS_EXECVE))
        return 0;

    buf_t *submit_p = get_buf(SUBMIT_BUF_IDX);
    if (submit_p == NULL)
        return 0;
    set_buf_off(SUBMIT_BUF_IDX, sizeof(context_t));

    context_t context = init_and_save_context(ctx,
                                              submit_p,
                                              SYS_EXECVE,
                                              2 /*argnum*/,
                                              0 /*ret*/);

    u64 *tags = bpf_map_lookup_elem(&params_names_map, &context.eventid);
    if (!tags) {
        return -1;
    }

    argnum += save_str_to_buf(submit_p,
            (void *)args.args[0] /*filename*/,
            DEC_ARG(0, *tags));

    argnum += save_str_arr_to_buf(submit_p,
            (const char *const *)args.args[1] /*argv*/,
            DEC_ARG(1, *tags));

    if (get_config(CONFIG_EXEC_ENV)) {
        argnum += save_str_arr_to_buf(submit_p,
                (const char *const *)args.args[2] /*envp*/,
                DEC_ARG(2, *tags));
    }

    context.argnum = argnum;
    save_context_to_buf(submit_p, (void*)&context);
    events_perf_submit(ctx);
    return 0;
}
```

CHALLENGE: LIBBPF SUPPORT

(1:1 libbpfgo & libbpf)

CO-RE: Challenges (libbpf support: link destroy vs detach)

- commit **d88b71d4a916** libbpf: remove unused bpf_link's destroy operation, add dealloc

```c
/* Release "ownership" of underlying BPF resource (typically, BPF program
 * attached to some BPF hook, e.g., tracepoint, kprobe, etc). Disconnected
 * link, when destructed through bpf_link__destroy() call won't attempt to
 * detach/unregisted that BPF resource. This is useful in situations where,
 * say, attached BPF program has to outlive userspace program that attached it
 * in the system. Depending on type of BPF program, though, there might be
 * additional steps (like pinning BPF program in BPF FS) necessary to ensure
 * exit of userspace program doesn't trigger automatic detachment and clean up
 * inside the kernel.
 */
void bpf_link__disconnect(struct bpf_link *link)
{
	link->disconnected = true;
}

int bpf_link__destroy(struct bpf_link *link)
{
	int err = 0;

	if (IS_ERR_OR_NULL(link))
		return 0;

	if (!link->disconnected && link->detach)
		err = link->detach(link);
	if (link->pin_path)
		free(link->pin_path);
	if (link->dealloc)
		link->dealloc(link);
	else
		free(link);

	return libbpf_err(err);
}
```

```go
84	// get BPF program from BPF object
85	bpfProgKsysSync, err = bpfModule.GetProgram("ksys_sync")
86	if err != nil {
87	    errExit(err)
88	}
89	// attach to BPF program to kprobe
90	bpfLinkKsysSync, err := bpfProgKsysSync.AttachKprobe("ksys_sync")
91	if err != nil {
92	    errExit(err)
93	}
94	// test detaching (libbpfgo PR #78 from Geyslan)
95	err = bpfLinkKsysSync.Detach()
96	if err != nil {
97	    errExit(err)
98	}
```

- **mine.go : 97 "invalid argument":** bpf_link__link_detach() shouldn't be used directly.

- link->**destroy()** usage is tricky:

  you may **disconnect bpf_link** and destroy internal resources only, keeping perf event fd opened and event enabled.

CO-RE: Challenges (libbpf support: legacy kprobe interface)

- commit **668ace0ea5ab** libbpf: use BPF perf link when supported by kernel

```
link = calloc(1, sizeof(*link));
if (!link)
        return libbpf_err_ptr(-ENOMEM);
link->link.detach = &bpf_link_perf_detach;
link->link.dealloc = &bpf_link_perf_dealloc;
link->perf_event_fd = pfd;

if (kernel_supports(prog->obj, FEAT_PERF_LINK)) {
        DECLARE_LIBBPF_OPTS(bpf_link_create_opts, link_opts,
                .perf_event.bpf_cookie = OPTS_GET(opts, bpf_cookie, 0));

        link_fd = bpf_link_create(prog_fd, pfd, BPF_PERF_EVENT, &link_opts);
        if (link_fd < 0) {
                err = -errno;
                pr_warn("prog '%s': failed to create BPF link for perf_event FD %d
                        prog->name, pfd,
                        err, libbpf_strerror_r(err, errmsg, sizeof(errmsg)));
                goto err_out;
        }
        link->link.fd = link_fd;
} else {
        if (OPTS_GET(opts, bpf_cookie, 0)) {
                pr_warn("prog '%s': user context value is not supported\n", prog-
                err = -EOPNOTSUPP;
                goto err_out;
        }

        if (ioctl(pfd, PERF_EVENT_IOC_SET_BPF, prog_fd) < 0) {
                err = -errno;
                pr_warn("prog '%s': failed to attach to perf_event FD %d: %s\n",
                        prog->name, pfd, libbpf_strerror_r(err, errmsg, sizeof(er
                if (err == -EPROTO)
                        pr_warn("prog '%s': try add PERF_SAMPLE_CALLCHAIN to or r
                                prog->name, pfd);
                goto err_out;
        }
        link->link.fd = pfd;
}
if (ioctl(pfd, PERF_EVENT_IOC_ENABLE, 0) < 0) {
        err = -errno;
        pr_warn("prog '%s': failed to enable perf_event FD %d: %s\n",
                prog->name, pfd, libbpf_strerror_r(err, errmsg, sizeof(errmsg)));
        goto err_out;
}

return &link->link;
```

DIFFERENT INTERFACES FOR EBPF LINK ATTACHMENTS TO PROBES AND TRACEPOINTS:

1. PERF_EVENT_IOC_SET_BPF (attaches program to existing kprobe tracepoint event) + PERF_EVENT_IOC_ENABLE (enables event specified by fd).

2. **BPF_LINK_CREATE (for-next tree)**

3. **LEGACY KPROBE_EVENTS (for-next tree)**

CO-RE: Challenges (libbpf support: legacy kprobe interface)

```
commit 155f556d64b1
Author: Rafael David Tinoco <rafaeldtinoco@ubuntu.com>
Date:   Tue Mar 23 01:09:52 2021

    libbpf: Add bpf object kern_version attribute setter

    Unfortunately some distros don't have their kernel version defined
    accurately in <linux/version.h> due to different long term support
    reasons.

    It is important to have a way to override the bpf kern_version
    attribute during runtime: some old kernels might still check for
    kern_version attribute during bpf_prog_load().
```

Kernel v4.15 needs eBPF **kern_version** attribute (we're currently supporting v4.19 and on).

```
commit ca304b40c20d
Author: Rafael David Tinoco <rafaeldtinoco@gmail.com>
Date:   Sun Sep 12 03:48:44 2021

    libbpf: Introduce legacy kprobe events support

    Allow kprobe tracepoint events creation through legacy interface, as the
    kprobe dynamic PMUs support, used by default, was only created in v4.17.

    Store legacy kprobe name in struct bpf_perf_link, instead of creating
    a new "subclass" off of bpf_perf_link. This is ok as it's just two new
    fields, which are also going to be reused for legacy uprobe support in
    follow up patches.
```

Kernel v4.19 still needs kprobe points to be added to **kprobe_events** (**legacy** kprobe support to libbpf) – thanks Andrii for reviewing and accepting it.

```
commit 46ed5fc33db9
Author: Andrii Nakryiko <andrii@kernel.org>
Date:   Tue Sep 21 18:00:35 2021

    libbpf: Refactor and simplify legacy kprobe code

    This patch also implicitly fixes the problem with invalid open() error
    handling present in poke_kprobe_events(), which (the function) this
    patch removes.

    Fixes: ca304b40c20d ("libbpf: Introduce legacy kprobe events support")
```

Note: Last days Andrii simplified legacy kprobe code and introduced legacy uprobe support, besides fixing some issues.

aqua

(quick pause: eBPF and relocations)

CO-RE: BPF Section Headers (quick overview)

```
Sections:
Idx Name                                    Size     VMA              Type
  0                                         00000000 0000000000000000
  1 .text                                   00000000 0000000000000000 TEXT
  2 kprobe/ksys_sync                        00000190 0000000000000000 TEXT
  3 tracepoint/syscalls/sys_enter_sync      00000190 0000000000000000 TEXT
  4 license                                 00000004 0000000000000000 DATA
  5 .maps                                   00000018 0000000000000000 DATA
  6 .BTF                                    00006999 0000000000000000
  7 .BTF.ext                                00
  8 .symtab                                 00
  9 .relkprobe/ksys_sync                    00
 10 .reltracepoint/syscalls/sys_enter_sync  00
 11 .rel.BTF                                00
 12 .rel.BTF.ext                            00
 13 .llvm_addrsig                           00
 14 .strtab                                 00
```

```
RELOCATION RECORDS FOR [kprobe/ksys_sync]:
OFFSET           TYPE                     VALUE
0000000000000068 R_BPF_64_64              CONFIG_ARCH_HAS_SYSCALL_WRAPPER
0000000000000148 R_BPF_64_64              events

RELOCATION RECORDS FOR [tracepoint/syscalls/sys_enter_sync]:
OFFSET           TYPE                     VALUE
0000000000000068 R_BPF_64_64              CONFIG_ARCH_HAS_SYSCALL_WRAPPER
0000000000000148 R_BPF_64_64              events
```

```
Symbol table '.symtab' contains 12 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    2 kprobe/ksys_sync
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 tracepoint/syscalls/sys_enter_sync
     7: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND CONFIG_ARCH_HAS_SYSCALL_WRAPPER
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 LICENSE
     9: 0000000000000000    24 OBJECT  GLOBAL DEFAULT    5 events
    10: 0000000000000000   400 FUNC    GLOBAL DEFAULT    2 ksys_sync
    11: 0000000000000000   400 FUNC    GLOBAL DEFAULT    3 tracepoint__sys_enter_sync
```

aqua

CHALLENGE: LIBBPF SUPPORT

(kconfig relocations)

CO-RE: Challenges (BPF relocations: kconfig & dead code)

- Kconfig relocations made with externs and eBPF map:

```
Relocation section '.relraw_tracepoint/sys_enter' at offset 0x1624e8 contains 50 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
0000000000000068  000008f800000001 R_BPF_64_64            0000000000000000 CONFIG_ARCH_HAS_SYSCALL_WRAPPER
0000000000000378  0000091700000001 R_BPF_64_64            0000000000000140 sys_32_to_64_map
00000000000003f0  0000090100000001 R_BPF_64_64            0000000000000000 config_map
0000000000000678  0000090900000001 R_BPF_64_64            0000000000000050 new_pidns_map
00000000000006c0  0000090100000001 R_BPF_64_64            0000000000000000 config_map
```

```
Symbol table '.symtab' contains 2373 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000001bc8     0 NOTYPE  LOCAL  DEFAULT    2 LBB0_101
...
  2291: 0000000000000000     0 SECTION LOCAL  DEFAULT   37 kprobe/security_file_mprotect
  2292: 0000000000000000     0 SECTION LOCAL  DEFAULT   38 kprobe/security_bpf
  2293: 0000000000000000     0 SECTION LOCAL  DEFAULT   39 kprobe/security_bpf_map
  2294: 0000000000000000     0 SECTION LOCAL  DEFAULT   40 kprobe/security_kernel_read_file
  2295: 0000000000000000     0 SECTION LOCAL  DEFAULT   41 classifier
  2296: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND CONFIG_ARCH_HAS_SYSCALL_WRAPPER
  2297: 0000000000000000     4 OBJECT  GLOBAL DEFAULT   44 KERNEL_VERSION
  2298: 0000000000000000     4 OBJECT  GLOBAL DEFAULT   43 LICENSE
  2299: 0000000000000078    20 OBJECT  GLOBAL DEFAULT   42 args_map
  2300: 000000000000012c    20 OBJECT  GLOBAL DEFAULT   42 bin_args_map
  2301: 00000000000001e0    20 OBJECT  GLOBAL DEFAULT   42 bufs
...
```

- Dead code elimination did not work for <= **v5.4 kernels** (constant coming from R/O map value). Verifier would not allow load because of bad accesses coming from dead branch.

```
SEC("raw_tracepoint/sys_enter")
int tracepoint__raw_syscalls__sys_enter(struct bpf_raw_tracepoint_args *ctx)
{
    args_t args_tmp = {};
    int id = ctx->args[1];
    struct task_struct *task = (struct task_struct *)bpf_get_current_task();

if (CONFIG_ARCH_HAS_SYSCALL_WRAPPER) {
    struct pt_regs regs = {};
    bpf_probe_read(&regs, sizeof(struct pt_regs), (void*)ctx->args[0]);

    if (is_x86_compat(task)) {
#if defined(bpf_target_x86)
        args_tmp.args[0] = regs.bx;
        args_tmp.args[1] = regs.cx;
        args_tmp.args[2] = regs.dx;
        args_tmp.args[3] = regs.si;
        args_tmp.args[4] = regs.di;
        args_tmp.args[5] = regs.bp;
#endif // bpf_target_x86
    } else {
        args_tmp.args[0] = PT_REGS_PARM1(&regs);
        args_tmp.args[1] = PT_REGS_PARM2(&regs);
        args_tmp.args[2] = PT_REGS_PARM3(&regs);
#if defined(bpf_target_x86)
        args_tmp.args[3] = regs.r10;
#else
        args_tmp.args[3] = PT_REGS_PARM4(&regs);
#endif
        args_tmp.args[4] = PT_REGS_PARM5(&regs);
        args_tmp.args[5] = PT_REGS_PARM6(&regs);
    }
} else { // NO CONFIG_ARCH_HAS_SYSCALL_WRAPPER
    args_tmp.args[0] = ctx->args[0];
    args_tmp.args[1] = ctx->args[1];
    args_tmp.args[2] = ctx->args[2];
    args_tmp.args[3] = ctx->args[3];
    args_tmp.args[4] = ctx->args[4];
    args_tmp.args[5] = ctx->args[5];
} // END CONFIG_ARCH_HAS_SYSCALL_WRAPPER
```

kconfig relocation

Issue: Wasn't **CO-RE**

dead code elimination

CO-RE: Challenges (BPF relocations: kconfig & dead code)

- Propose the dead code verifier fix to **stable v5.4 branch**:

```
commit 812ee47ad76e
Author: Andrii Nakryiko <andriin@fb.com>
Date:   Wed Oct 9 17:14:57 2019

    bpf: Track contents of read-only maps as scalars

    commit a23740ec43ba022dbfd139d0fe3eff193216272b upstream.

    Maps that are read-only both from BPF program side and user space side
    have their contents constant, so verifier can track referenced values
    precisely and use that knowledge for dead code elimination, branch
    pruning, etc. This patch teaches BPF verifier how to do this.

    Signed-off-by: Andrii Nakryiko <andriin@fb.com>
    Signed-off-by: Daniel Borkmann <daniel@iogearbox.net>
    Link: https://lore.kernel.org/bpf/20191009201458.2679171-2-andriin@fb.c
    Signed-off-by: Rafael David Tinoco <rafaeldtinoco@gmail.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
```

- And **fixed the CO-RE** issue we had.

- But there is more…

```
#ifdef CORE
#define get_kconfig(x) get_kconfig_val(x)
#else
#define get_kconfig(x) CONFIG_##x
#endif

if (get_kconfig(ARCH_HAS_SYSCALL_WRAPPER)) {
    struct pt_regs regs = {};
    bpf_probe_read(&regs, sizeof(struct pt_regs), (void*)ctx->args[0]);

    if (is_x86_compat(task)) {
#if defined(bpf_target_x86)
        args_tmp.args[0] = regs.bx;
        args_tmp.args[1] = regs.cx;
        args_tmp.args[2] = regs.dx;
        args_tmp.args[3] = regs.si;
        args_tmp.args[4] = regs.di;
        args_tmp.args[5] = regs.bp;
#endif // bpf_target_x86
    } else {
        args_tmp.args[0] = PT_REGS_PARM1(&regs);
        args_tmp.args[1] = PT_REGS_PARM2(&regs);
        args_tmp.args[2] = PT_REGS_PARM3(&regs);
#if defined(bpf_target_x86)
        args_tmp.args[3] = regs.r10;
#else
        args_tmp.args[3] = PT_REGS_PARM4(&regs);
#endif
        args_tmp.args[4] = PT_REGS_PARM5(&regs);
        args_tmp.args[5] = PT_REGS_PARM6(&regs);
    }
} else {
    bpf_probe_read(args_tmp.args, sizeof(6 * sizeof(u64)), (void *)ctx->args);
}
```

Wait! This is NEW!

ISSUE FIXED

CO-RE: Challenges (BPF relocations: kconfig dependency)

```
#ifdef CORE
#define get_kconfig(x) get_kconfig_val(x)
#else
#define get_kconfig(x) CONFIG_##x
#endif
```

```
if (get_kconfig(ARCH_HAS_SYSCALL_WRAPPER)) {
    struct pt_regs regs = {};
    bpf_probe_read(&regs, sizeof(struct pt_regs), (void*)ctx->args[0]);

    if (is_x86_compat(task)) {
#if defined(bpf_target_x86)
        args_tmp.args[0] = regs.bx;
        args_tmp.args[1] = regs.cx;
        args_tmp.args[2] = regs.dx;
        args_tmp.args[3] = regs.si;
        args_tmp.args[4] = regs.di;
        args_tmp.args[5] = regs.bp;
#endif // bpf_target_x86
    } else {
```

SOLUTION

- **Libbpf** relocations depend on:
  - **KCFG** extern
    - /proc/config.gz
    - /boot/config-$(uname –r)
  - **KSYM** extern (subsequent slides)
    - **RAW BTF** or ELF with **.BTF** sec

- Libbpf allows specifying kconfig file, but it is read as **extra kconfig options**, not a 'replacement' for existing kconfig.gz.

- **SOLUTION** was to **create our own kconfig_map**. (approach is like what libbpf does)

```go
// InitKernelConfig inits external KernelConfig object
func InitKernelConfig() (*KernelConfig, error) {
    config := KernelConfig{}

    // special case: user provided kconfig file (it MUST exist)

    osKConfigFilePath, err := checkEnvPath("LIBBPFGO_KCONFIG_FILE") // override /proc/config.gz
    if err != nil {
        return &config, err
    }
    if len(osKConfigFilePath) > 2 {
        if _, err := os.Stat(osKConfigFilePath); err != nil {
            return &config, err
        }
        config.kConfigFilePath = osKConfigFilePath
        if err := config.initKernelConfig(osKConfigFilePath); err != nil {
            return &config, err
        }

        return &config, nil
    }
```

aqua

CHALLENGE: LIBBPF SUPPORT

(ksym relocations in any env)

Tracee with BTF Hub

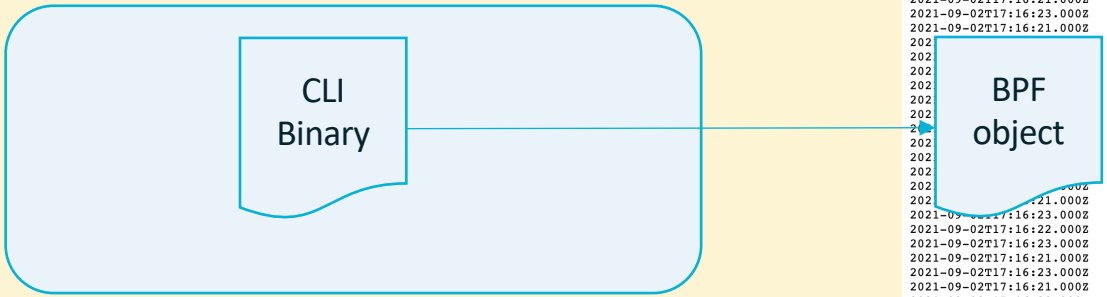
```
[rafaeldtinoco@bionic:~/.../aquasec-tracee/tracee-ebpf][btfhubdemo]$ sudo ./dist/tracee-ebpf --debug --trace uid=1000 --trace pid=new --trace event=execve
BTF: ubuntu 18.04 5.4.0-84-generic
BTF: vmlinux = false btfhub = true btfcached = false
BTF: bpfenv = false btfenv = false vmlinux = false btfhub = true btfcached = false
BTF: btfhub:  https://github.com/aquasecurity/btfhub/raw/main/ubuntu/18.04/x86_64/5.4.0-84-generic.btf.tar.xz
BTF: btf file is now cached
BTF: using btf from btfhub: %s /tmp/tracee/5.4.0-84-generic.btf
BPF: using embedded bpf object
unpacked CO:RE bpf object file into memory
TIME             UID    COMM             PID     TID     RET             EVENT               ARGS
05:41:39:618619  1000   bash             4011    4011    0               execve              pathname: /bin/cat, argv: [cat /proc/cmdline]
05:41:41:968996  1000   bash             4015    4015    0               execve              pathname: /sbin/ip, argv: [ip addr list]
05:41:44:197013  1000   bash             4017    4017    0               execve              pathname: /bin/ps, argv: [ps]
05:41:45:113015  1000   bash             4019    4019    0               execve              pathname: /bin/ls, argv: [ls --color=auto]
05:41:48:175616  1000   bash             4023    4023    0               execve              pathname: /bin/cat, argv: [cat /proc/cmdline]
05:41:53:136277  1000   bash             4025    4025    0               execve              pathname: /bin/true, argv: [/bin/true]

End of events stream
Stats: {eventCounter:6 errorCounter:0 lostEvCounter:0 lostWrCounter:0 lostNtCounter:0}
[rafaeldtinoco@bionic:~/.../aquasec-tracee/tracee-ebpf][btfhubdemo]$ sudo ./dist/tracee-ebpf --debug --trace uid=1000 --trace pid=new --trace event=execve
BTF: ubuntu 18.04 5.4.0-84-generic
BTF: vmlinux = false btfhub = true btfcached = true
BTF: bpfenv = false btfenv = false vmlinux = false btfhub = true btfcached = true
BTF: using btf from btfhub: %s /tmp/tracee/5.4.0-84-generic.btf
BPF: using embedded bpf object
unpacked CO:RE bpf object file into memory
TIME             UID    COMM             PID     TID     RET             EVENT               ARGS
05:42:05:436923  1000   bash             4044    4044    0               execve              pathname: /bin/ls, argv: [ls --color=auto]

End of events stream
Stats: {eventCounter:1 errorCounter:0 lostEvCounter:0 lostWrCounter:0 lostNtCounter:0}
```

aqua tracee

cross-compile BPF



Build other eBPF programs #100

Open **afdesk** opened this issue on May 15 · 16 comments

afdesk commented on May 15

Hello!

Motivation

I am interested in using `driverkit` to build other eBPF programs besides Falco (currently working with `Tracee`).

Feature

I would like to propose some changes to make `driverkit` more generic, and I can implement those if the maintainers agree:

- add custom templates for the ebpf builders
- use different docker images for the ebpf builders
- add options to disable some actions for the `driverkit` docker process (for prepare driver config and makefile templates)

And the main change: implement a way to use linux headers for all the kernel releases.

Now `driverkit` is looking for distros from ubuntu repositories (https://mirrors.edge.kernel.org/ubuntu/pool/main/l/linux) with next versions of kernels: ... 4.4.x, 4.15.x, 5.4.x, 5.8.x, 5.11.x.
We need to support all the kernel versions.

Happy to hear what you think and if you have other suggestions for supporting this use case!

4

https://github.com/falcosecurity/driverkit/issues/100

download.falco.org/?prefix=driver/17f5df52a7d9ed6bb12d3b1768460def8439936d/

2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-37-generic_41.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-37-generic_41.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-39-generic_43.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-39-generic_43.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-40-generic_44.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-40-generic_44.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-42-generic_46.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-42-generic_46.o
202	676.5 kB	falco_ubuntu-generic_5.4.0-45-generic_49.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-45-generic_49.o
202	676.5 kB	falco_ubuntu-generic_5.4.0-47-generic_51.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-47-generic_51.o
202	676.5 kB	falco_ubuntu-generic_5.4.0-48-generic_52.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-48-generic_52.o
2	676.5 kB	falco_ubuntu-generic_5.4.0-51-generic_56.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-51-generic_56.o
202	676.5 kB	falco_ubuntu-generic_5.4.0-52-generic_57.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-52-generic_57.o
202	676.5 kB	falco_ubuntu-generic_5.4.0-53-generic_59.ko
202	3.2 MB	falco_ubuntu-generic_5.4.0-53-generic_59.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-54-generic_60.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-54-generic_60.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-58-generic_64.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-58-generic_64.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-59-generic_65.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-59-generic_65.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-60-generic_67.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-60-generic_67.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-62-generic_70.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-62-generic_70.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-64-generic_72.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-64-generic_72.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-65-generic_73.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-65-generic_73.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-66-generic_74.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-66-generic_74.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-67-generic_75.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-67-generic_75.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-70-generic_78.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-70-generic_78.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-71-generic_79.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-71-generic_79.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-72-generic_80.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-72-generic_80.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-73-generic_82.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-73-generic_82.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-74-generic_83.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-74-generic_83.o
2021-08-07T04:42:48.000Z	681.6 kB	falco_ubuntu-generic_5.4.0-75-generic_84.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-75-generic_84.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-76-generic_85.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-76-generic_85.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-77-generic_86.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-77-generic_86.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-78-generic_87.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-78-generic_87.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-79-generic_88.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-79-generic_88.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-80-generic_90.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-80-generic_90.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-81-generic_91.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-81-generic_91.o
2021-08-27T16:46:16.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-83-generic_93.ko
2021-08-27T16:46:15.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-83-generic_93.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-84-generic_94.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-84-generic_94.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-25-generic_26.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-26-generic_27.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-28-generic_30.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-29-generic_31.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-33-generic_36.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-34-generic_37.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-36-generic_40.ko

download.falco.org/?prefix=driver/17f5df52a7d9ed6bb12d3b1768460def8439936d/

2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-37-generic_41.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-37-generic_41.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-39-generic_43.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-39-generic_43.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-40-generic_44.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-40-generic_44.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-42-generic_46.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-42-generic_46.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-45-generic_49.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-45-generic_49.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-47-generic_51.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-47-generic_51.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-48-generic_52.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-48-generic_52.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-51-generic_56.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-51-generic_56.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-52-generic_57.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-52-generic_57.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-53-generic_59.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-53-generic_59.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-54-generic_60.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-54-generic_60.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-58-generic_64.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-58-generic_64.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-59-generic_65.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-59-generic_65.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-60-generic_67.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-60-generic_67.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.4.0-62-generic_70.ko
2021-09-02T17:16:21.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-62-generic_70.o
2021-09		
2021-08-07T04:42:48.000Z	681.6 kB	falco_ubuntu-generic_5.4.0-75-generic_84.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-75-generic_84.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-76-generic_85.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-76-generic_85.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-77-generic_86.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-77-generic_86.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-78-generic_87.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-78-generic_87.o
2021-08-07T04:42:48.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-79-generic_88.ko
2021-08-07T04:42:46.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-79-generic_88.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-80-generic_90.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-80-generic_90.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-81-generic_91.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-81-generic_91.o
2021-08-27T16:46:16.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-83-generic_93.ko
2021-08-27T16:46:15.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-83-generic_93.o
2021-09-02T17:16:23.000Z	681.5 kB	falco_ubuntu-generic_5.4.0-84-generic_94.ko
2021-09-02T17:16:22.000Z	3.2 MB	falco_ubuntu-generic_5.4.0-84-generic_94.o
2021-09-02T17:16:23.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-25-generic_26.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-26-generic_27.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-28-generic_30.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-29-generic_31.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-33-generic_36.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-34-generic_37.ko
2021-07-24T05:11:13.000Z	676.5 kB	falco_ubuntu-generic_5.8.0-36-generic_40.ko

Fix the *artifact*
for the *environment*
(cross-compilation)

aquasecurity / btfhub Public

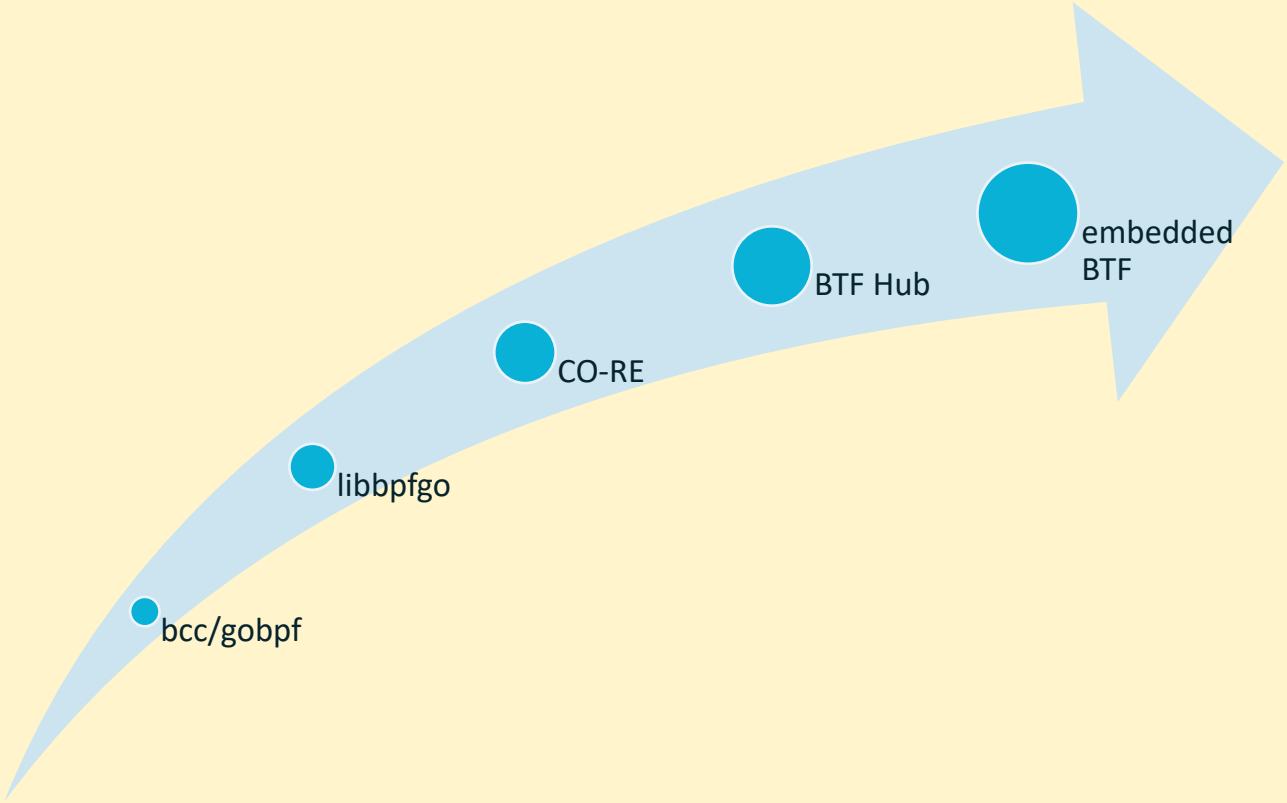
Code | Issues 1 | Pull requests | Actions | Projects | Wiki | Security | Insights

main | btfhub / ubuntu / 18.04 / x86_64 /

rafaeldtinoco ubuntu: sync latest bionic BTFs

..	
4.15.0-1007-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1009-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-101-generic.btf.tar.xz	arch: create x86_64 directory for each of supported distros
4.15.0-1010-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1011-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1016-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1031-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1032-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1033-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1034-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1035-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1037-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1039-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1040-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1041-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1043-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1044-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel
4.15.0-1045-aws.btf.tar.xz	ubuntu: bionic: add btfs for aws kernel

Fix the *environment*
for the *artifact*
(external BTF)


bcc/gobpf
libbpfgo
CO-RE
BTF Hub
embedded
BTF

Truly portable eBPF application

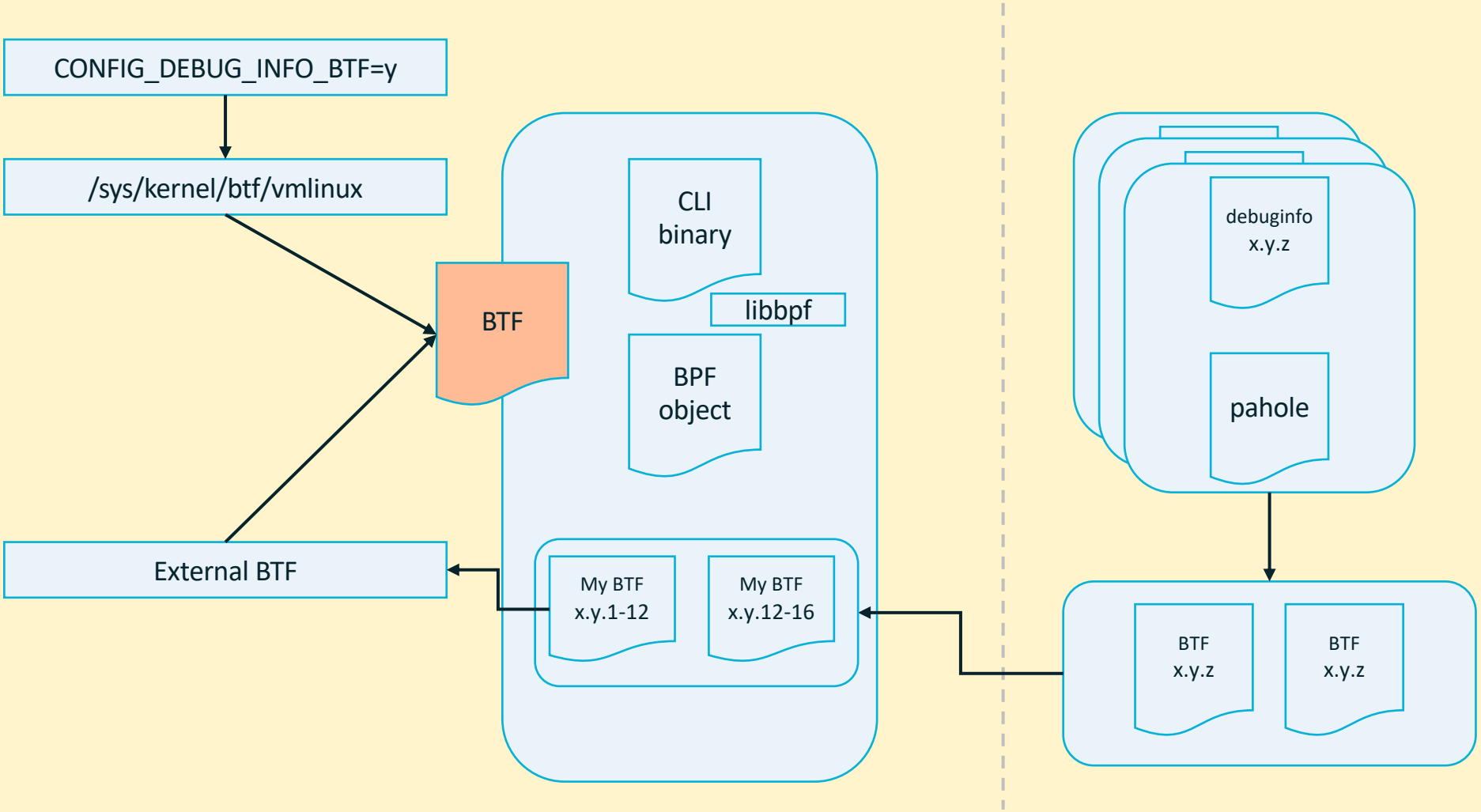
```
[user@host:~/.../focal/x86_64]$ pahole -C task_struct ./5.4.0-26-generic.btf -E > 5.4.0-26-generic.btf.task-struct
[user@host:~/.../focal/x86_64]$ pahole -C task_struct ./5.4.0-28-generic.btf -E > 5.4.0-28-generic.btf.task-struct
[user@host:~/.../focal/x86_64]$ pahole -C task_struct ./5.4.0-70-generic.btf -E > 5.4.0-70-generic.btf.task-struct
```

✅ Smaller BTFs

```
[user@host:~/.../tracee-ebpf/dist][rafaeldtinoco]$ pahole \
> -F btf ./tracee.bpf.core.o | \
> grep "^[a-zA-Z]" | sed 's:{::g' | \
> sort | head -20
struct __call_single_node
struct __sk_buff
struct address_space
struct arch_hw_breakpoint
struct arch_tlbflush_unmap_batch
struct blocking_notifier_head
struct bpf_cgroup_storage
struct bpf_cgroup_storage_key
struct bpf_map
struct bpf_map_def
```

```
[user@host:~/.../focal/x86_64]$ diff ./5.4.0-26-generic.btf.task-struct 5.4.0-70-generic.btf.task-struct | h
ss -l diff --theme GitHub
144,149c144,145
<               struct sched_rt_entity * parent;                                    /*   624     8 */
<               struct rt_rq *     rt_rq;                                           /*   632     8 */
<               /* --- cacheline 10 boundary (640 bytes) --- */
<               struct rt_rq *     my_q;                                            /*   640     8 */
<       } rt; /*   576    72 */
<       struct task_group *        sched_task_group;                                /*   648     8 */
---
>       } rt; /*   576    48 */
>       struct task_group *        sched_task_group;                                /*   624     8 */
152,158c148,158
<                       long unsigned int __rb_parent_color;                        /*   656     8 */
<                       struct rb_node * rb_right;                                  /*   664     8 */
<                       struct rb_node * rb_left;                                   /*   672     8 */
```

✅ Less BTFs


CONFIG_DEBUG_INFO_BTF=y
/sys/kernel/btf/vmlinux
BTF
External BTF
CLI binary
libbpf
BPF object
My BTF x.y.1-12
My BTF x.y.12-16
debuginfo x.y.z
pahole
BTF x.y.z
BTF x.y.z

Let us know what you think

- https://github.com/aquasecurity/tracee
- https://github.com/aquasecurity/libbpfgo
- https://github.com/aquasecurity/btfhub
- Itay Shakury @itaysk
- Rafael D. Tinoco @rafaeldtinoco
- Yaniv Agman @AgmanYaniv
- Grant Seltzer @GrantSeltzer

aqua

TORWARDS TRULLY PORTABLE eBPF

Itay Shakury & Rafael Tinoco
Aqua Security

Linux Plumbers 2021

@itaysk
@rafaeldtinoco

References

https://nakryiko.com/posts/bpf-portability-and-co-re/
https://nakryiko.com/posts/btf-dedup/
https://lwn.net/Articles/801479/
https://github.com/libbpf/libbpf#bpf-co-re-compile-once--run-everywhere
https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md
Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges and Applications

Images and Icons:
https://commons.wikimedia.org/
https://www.flaticon.com/