the way to d_path helper

jiri olsa

bpf_d_path helper

returns string with full path of struct path object

```
* long bpf_d_path(struct path *path, char *buf, u32 sz)
*     Description
*             Return full path for given 'struct path' object, which
*             needs to be the kernel BTF 'path' object. The path is
*             returned in the provided buffer 'buf' of size 'sz' and
*             is zero terminated.
*
*     Return
*             On success, the strictly positive length of the string,
*             including the trailing NUL character. On error, a negative
*             value.
```

selftest vfs_getattr

```
SEC("fentry/vfs_getattr")

int BPF_PROG(prog_stat, struct path *path, struct kstat *stat,
             __u32 request_mask, unsigned int query_flags)
{
      ...
      ret = bpf_d_path(path, buf, MAX_PATH_LEN);
```

selftest filp_close

```
SEC("fentry/filp_close")

int BPF_PROG(prog_close, struct file *file, void *id)
{
	...
	ret = bpf_d_path(&file->f_path, buf, MAX_PATH_LEN);
```

bpftrace filp_close

```
int filp_close(struct file *filp, fl_owner_t id);
```

```
# bpftrace -e 'kfunc:filp_close \
  { printf("%s\n", dpath(args->filp->f_path)); }'

Attaching 1 probe...
pipe:[109233]
/dev/pts/7
/dev/pts/7
/etc/ld.so.cache
/usr/lib64/libcap.so.2.26
...
```

bpftrace fget

```
struct file *fget(unsigned int fd);


# bpftrace -e 'kretfunc:fget \
  { printf("%s\n", dpath(retval->f_path)); }'

Attaching 1 probe...
/etc/ld.so.cache
/usr/lib64/libselinux.so.1
/usr/lib64/libselinux.so.1
...
```

probe types

- BPF_PROG_TYPE_TRACING
- bpftrace’s kfunc probes

read dentry objects via bpf_probe_read

https://github.com/iovisor/bcc/issues/237#issuecomment-252128055

```
struct task_struct *curr = (struct task_struct *)bpf_get_current_task();

bpf_probe_read(&files, sizeof(files), &curr->files);
bpf_probe_read(&fdt, sizeof(fdt), &files->fdt);
bpf_probe_read(&f, sizeof(f), &fdt[fd]);
bpf_probe_read(&de, sizeof(de), &f->f_path.dentry);
bpf_probe_read(&dn, sizeof(dn), &de->d_name);
...
```

get_fd_path / get_file_path / file_path

from Wenbo Zhang

https://lore.kernel.org/bpf/cover.1574162990.git.ethercflow@gmail.com/

```
* int bpf_get_file_path(char *path, u32 size, int fd)
*     Description
*             Get **file** atrribute from the current task by *fd*, then call
*             **d_path** to get it's absolute path and copy it as string into
*             *path* of *size*.
```

from file descriptor to file object

call d_path on file’s path object

problems

from file descriptor to file object – racy

call d_path on file’s path object – locking

```
> What are the locking conditions guaranteed to that XXXXX?  Note that d_path()
> is *NOT* lockless - call it from an interrupt/NMI/etc. and you are XXXXX.
> It can grab rename_lock and mount_lock; usually it avoids that, so you won't
> see them grabbed on every call, but after the first seqlock mismatch it will
> fall back to grabbing the spinlock in question.  And then there's ->d_dname()
> with whatever things _that_ chooses to do....
```

d_path needs

- BTF arguments
- list of allowed probes

BTF arguments

```
BPF_CALL_3(bpf_d_path, struct path *, path, char *, buf, u32, sz)
{
        long len;
        char *p;

        if (!sz)
                return 0;

        p = d_path(path, buf, sz);
        if (IS_ERR(p)) {
                len = PTR_ERR(p);
        } else {
                len = buf + sz - p;
                memmove(buf, p, len);
        }

        return len;
}
```

BTF arguments

```
SEC("fentry/vfs_getattr")

int BPF_PROG(prog_stat, struct path *path, struct kstat *stat,
             __u32 request_mask, unsigned int query_flags)
{
      ...
      ret = bpf_d_path(path, buf, MAX_PATH_LEN);
```

BTF arguments + offset

- BTF ID + offset
- verifier accepts now for helper arguments

BTF arguments + offset

```
SEC("fentry/filp_close")

int BPF_PROG(prog_close, struct file *file, void *id)
{
        ...
        ret = bpf_d_path(&file->f_path, buf, MAX_PATH_LEN);
```

```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        ...
```

list of allowed probes

d_path can be called only from certain places (d’oh..)

- **helper provides list of functions**
- **verifier checks this list against probe**

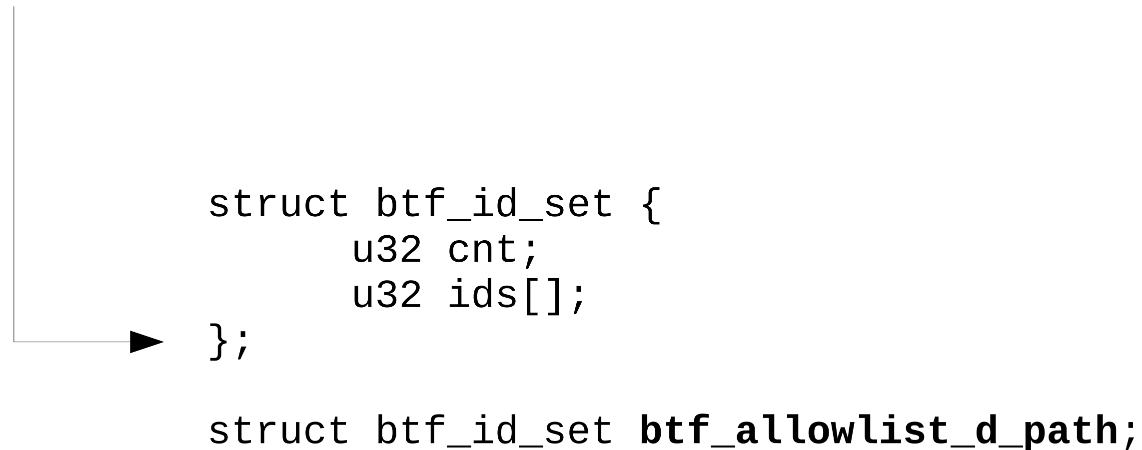
list of allowed probes

- list of names.. (d’oh)
- list of BTF IDs defined directly in the code
- macros in btf_ids.h header

list of allowed probes

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

BTF_SET_START(btf_allowlist_d_path)
BTF_ID(func, vfs_truncate)
BTF_ID(func, vfs_fallocate)
BTF_ID(func, dentry_open)
BTF_SET_END(btf_allowlist_d_path)
```



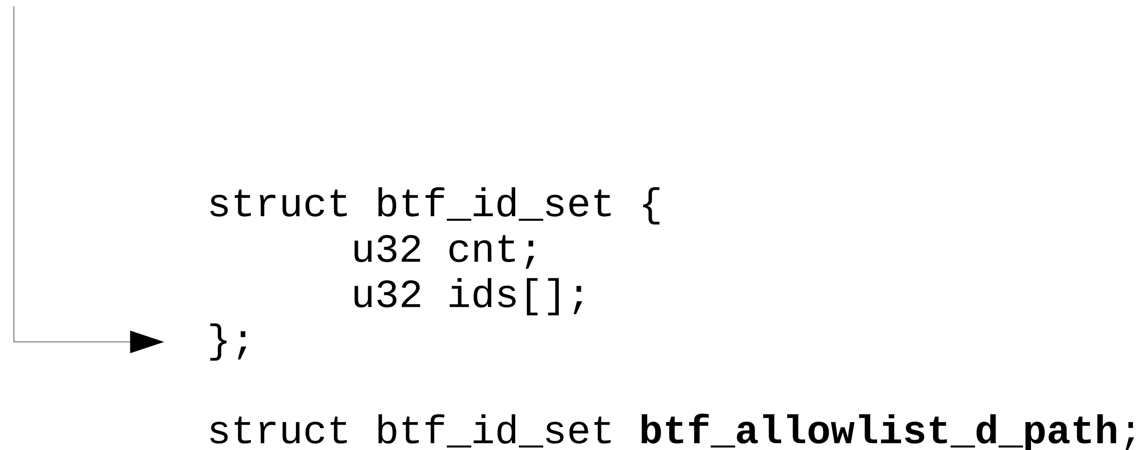
```
struct btf_id_set {
      u32 cnt;
      u32 ids[];
};

struct btf_id_set btf_allowlist_d_path;
```

set – sorted

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

BTF_SET_START(btf_allowlist_d_path)
BTF_ID(func, vfs_truncate)
BTF_ID(func, vfs_fallocate)
BTF_ID(func, dentry_open)
BTF_SET_END(btf_allowlist_d_path)
```



list – untouched

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

BTF_ID_LIST(bpf_ctx_convert_btf_id)
BTF_ID(struct, bpf_ctx_convert)

BTF_ID_LIST(btf_bpf_prog_id)
BTF_ID(struct, bpf_prog)

BTF_ID_LIST(btf_netlink_sock_id)
BTF_ID(struct, netlink_sock)
```

```
extern u32 bpf_ctx_convert_btf_id[];
extern u32 btf_bpf_prog_id[];
extern u32 netlink_sock[];
```

list

- get rid of runtime BTF ID resolving for:
  - helpers btf_ids argument array
  - btf_sock_ids array
  - bpf iterators arguments
  - bpf_ctx_convert struct

types

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

BTF_ID_LIST(list)
BTF_ID(struct,  task_struct)
BTF_ID(union,   thread_union)
BTF_ID(typedef, int32_t)
BTF_ID(func,    file_close)

BTF_ID_UNUSED
```

local / global

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

BTF_ID_LIST(list)
BTF_ID_LIST_GLOBAL(list)

BTF_SET_START(set)
BTF_SET_END

BTF_SET_START_GLOBAL(set)
BTF_SET_END
```

config

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

BTF_SET_START(btf_allowlist_d_path)

#ifdef CONFIG_FOO
BTF_ID(func, foo)
#endif

BTF_ID(func, vfs_fallocate)
BTF_ID(func, dentry_open)

BTF_SET_END(btf_allowlist_d_path)
```

.BTF_ids section

object.o

vmlinux

.BTF_ids ELF section

```
__BTF_ID__set1__set:
.zero 4
__BTF_ID__type1__name1__3:
.zero 4
__BTF_ID__type2__name2__4:
.zero 4
```

```
.zero 4
```

```
.zero 4
```

.BTF_ids ELF section

```
__BTF_ID__set1__set:
.zero 4
__BTF_ID__type1__name1__3:
.zero 4
__BTF_ID__type2__name2__4:
.zero 4
__BTF_ID__type3__name3__5:
.zero 4
__BTF_ID__type4__name4__6:
.zero 4
__BTF_ID__type5__name5__7:
.zero 4
__BTF_ID__type6__name6__8:
.zero 4
__BTF_ID__type7__name7__9:
...
```

```
$ make
...
GEN     modules.builtin
LD      .tmp_vmlinux.btf
BTF     .btf.vmlinux.bin.o
LD      .tmp_vmlinux.kallsyms1
KSYM    .tmp_vmlinux.kallsyms1.o
LD      .tmp_vmlinux.kallsyms2
KSYM    .tmp_vmlinux.kallsyms2.o
LD      vmlinux
BTFIDS  vmlinux
SORTTAB vmlinux
…
```

```
$ make
...
GEN      modules.builtin
LD       .tmp_vmlinux.btf
BTF      .btf.vmlinux.bin.o
LD       .tmp_vmlinux.kallsyms1
KSYM     .tmp_vmlinux.kallsyms1.o
LD       .tmp_vmlinux.kallsyms2
KSYM     .tmp_vmlinux.kallsyms2.o
LD       vmlinux
BTFIDS   vmlinux
SORTTAB  vmlinux
…
```

```
$ make
...
GEN     modules.builtin
LD      .tmp_vmlinux.btf
BTF     .btf.vmlinux.bin.o
LD      .tmp_vmlinux.kallsyms1
KSYM    .tmp_vmlinux.kallsyms1.o
LD      .tmp_vmlinux.kallsyms2
KSYM    .tmp_vmlinux.kallsyms2.o
LD      vmlinux
BTFIDS  vmlinux
SORTTAB vmlinux
…
```

$ pahole vmlinux → .BTF data

$ resolve_btfids vmlinux

.BTF_ids ELF section

```
__BTF_ID__set__set:
.zero 4
__BTF_ID__type1__name1__3:
.zero 4
__BTF_ID__type2__name2__4:
.zero 4
__BTF_ID__type3__name3__5:
.zero 4
__BTF_ID__type4__name4__6:
.zero 4
__BTF_ID__type5__name5__7:
.zero 4
__BTF_ID__type6__name6__8:
.zero 4
__BTF_ID__type7__name7__9:
```

resolve_btfids

- resolves .BTF_ids section
- in tools/bpf/resolve_btfids
- built on kernel compilation start for CONFIG_DEBUG_INFO_BTF option

future

- more functions for d_path list
- allow d_path for LSM probes
- allow d_path for raw tracepoints

documentation

- Documentation/bpf/btf.rst
- tools/bpf/resolve_btfids/main.c
- include/linux/btf_ids.h

thanks, questions..