eBPF Docs

@docs.ebpf.io

About Me

- Isovalent @ Cisco
- cilium/cilium contributor and cilium/ebpf reviewer
- Doing eBPF “stuff” since about early 2021

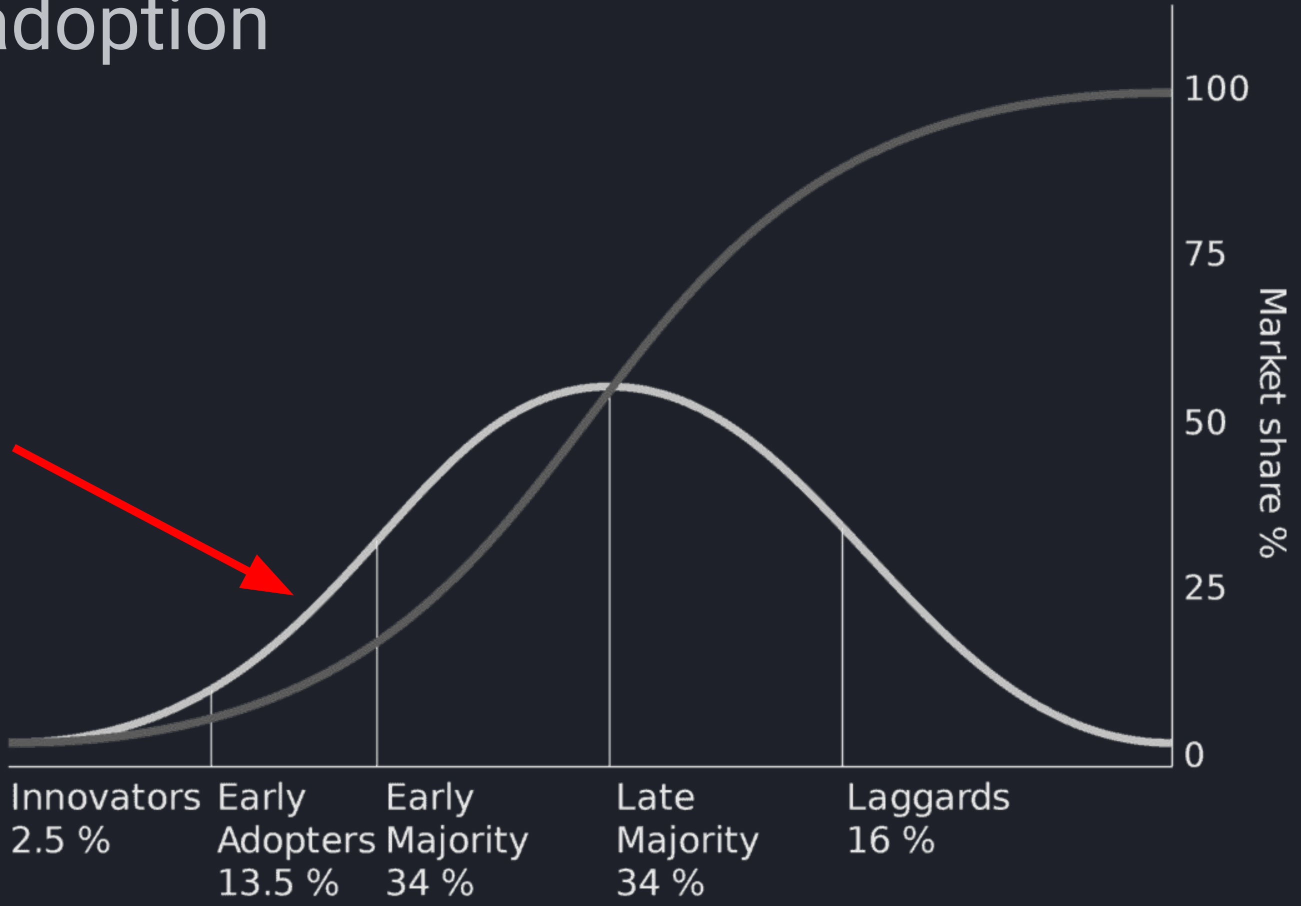
eBPF is great

- Do cool things with the kernel
- Safe(er)
- Fast

eBPF is great, but…

- ...it is complex
- ...it is quickly evolving
- ...not well documented

eBPF adoption



Resources for learning

- Blogs / articles
  - Lacking detail or age quickly
- Videos
  - Not searchable
- Kernel sources
  - Not beginner friendly
- Docs... 🤔

Shoutouts and kudos

- https://docs.kernel.org/bpf/
- https://man7.org/linux/man-pages/man7/bpf-helpers.7.html
- https://github.com/iovisor/bcc/tree/master/docs
- https://docs.cilium.io/en/latest/reference-guides/bpf/index.html

Motivation

- Helping people on slack and stackoverflow
- The same questions kept coming up
- Multiple links required or no links available
- Missing docs for certain questions
  - Can I use X in kernel Y (I need to support kernels going back to vX.YY)?
  - Can I use X in program type Y?
  - What does field X on context Y do?
  - Can I use loops in eBPF?
  - …

Goals

- One stop shop
- For eBPF devs, not kernel devs
- Internal linking (wikipedia style)
- Linkable content, separate pages, and bookmarks
- Searchable content, internally and via search engines
- Answers to practical questions

Welcome to the eBPF Docs! eBPF is an amazing technology which enables its users to extend the functionality of operating systems in a fast and secure way. eBPF is powerful, but also very complex, especially for newcomers.

This site aims to provide technical documentation for eBPF. If you are looking for specific information, we recommend you to use the search feature in to top right. You can use the navigation bar on the left for a hierarchical view, or use the condensed table of contents below to jump to a particular general topic.

Quick links

Intro to eBPF

An introduction to eBPF on Linux

→ eBPF on Linux

eBPF Concepts

An overview core eBPF concepts

→ Concepts

Program Types

An overview of eBPF program types

→ Program types

eBPF Map

An overview of eBPF map types

→ Map types

Helper Functions

An overview of eBPF helper functions

→ Helper functions

Syscall Commands

An overview of eBPF syscall commands

→ Syscall commands

KFuncs

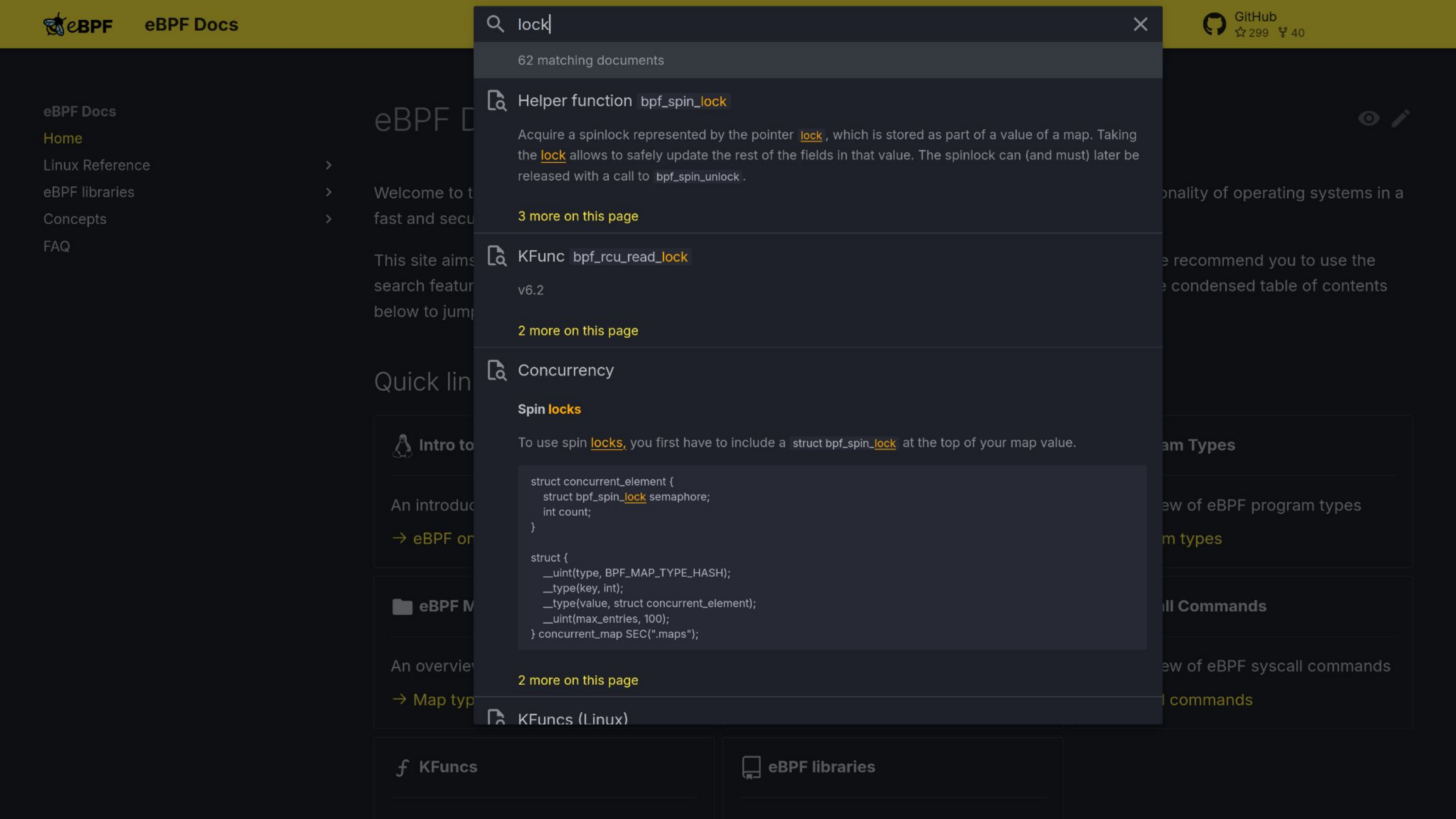
An overview of eBPF KFuncs

→ KFuncs

eBPF libraries

Libraries to help you with eBPF

→ eBPF libraries



lock

62 matching documents

Helper function `bpf_spin_lock`

Acquire a spinlock represented by the pointer `lock`, which is stored as part of a value of a map. Taking the lock allows to safely update the rest of the fields in that value. The spinlock can (and must) later be released with a call to `bpf_spin_unlock`.

3 more on this page

KFunc `bpf_rcu_read_lock`

v6.2

2 more on this page

Concurrency

Spin locks

To use spin locks, you first have to include a `struct bpf_spin_lock` at the top of your map value.

```
struct concurrent_element {
    struct bpf_spin_lock semaphore;
    int count;
}

struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __type(key, int);
    __type(value, struct concurrent_element);
    __uint(max_entries, 100);
} concurrent_map SEC(".maps");
```

2 more on this page

KFuncs (Linux)

Program type `BPF_PROG_TYPE_XDP`

v4.8

XDP (Express Data Path) programs can attach to network devices and are called for every incoming (ingress) packet received by that network device. XDP programs can take quite a large number of actions, most prominent of which are manipulation of the packet, dropping the packet, redirecting it and letting it pass to the network stack.

Notable use cases for XDP programs are for DDoS protection, Load Balancing, and high-throughput packet filtering. If loaded with native driver support, XDP programs will be called just after receiving the packet but before allocating memory for a socket buffer. This call site makes XDP programs extremely performant, especially in use cases where traffic is forwarded or dropped a lot in comparison to other eBPF program types or techniques which run after the relatively expensive socket buffer allocation process has taken place, only to discard it.

Usage

XDP programs are typically put into an ELF section prefixed with `xdp`. The XDP program is called by the kernel with a `xdp_md` context. The return value indicates what action the kernel should take with the packet, the following values are permitted:

- `XDP_ABORTED` - Signals that a unrecoverable error has taken place. Returning this action will cause the kernel to trigger the `xdp_exception` tracepoint and print a line to the trace log. This allows for debugging of such occurrences. It is also expensive, so should not be used without consideration in production.
- `XDP_DROP` - Discards the packet. It should be noted that since we drop the packet very early, it will

Helper functions

Not all helper functions are available in all program types. These are the helper calls available for XDP programs:

Supported helper functions

- `bpf_cgrp_storage_delete`
- `bpf_cgrp_storage_get`
- `bpf_check_mtu`
- `bpf_csum_diff`
- `bpf_dynptr_data`
- `bpf_dynptr_from_mem`
- `bpf_dynptr_read`
- `bpf_dynptr_write`
- `bpf_fib_lookup`
- `bpf_for_each_map_elem`
- `bpf_get_current_pid_tgid` v6.10
- `bpf_get_current_task`
- `bpf_get_current_task_btf`
- `bpf_get_ns_current_pid_tgid` v6.10
- `bpf_get_numa_node_id`
- `bpf_get_prandom_u32`
- `bpf_get_smp_processor_id`
- `bpf_jiffies64`
- `bpf_kptr_xchg`
- `bpf_ktime_get_boot_ns`
- `bpf_ktime_get_ns`
- `bpf_ktime_get_tai_ns`

KFuncs

Supported kfuncs

- bpf_arena_alloc_pages
- bpf_arena_free_pages
- bpf_cast_to_kern_ctx
- bpf_cgroup_acquire
- bpf_cgroup_ancestor
- bpf_cgroup_from_id
- bpf_cgroup_release
- bpf_crypto_decrypt
- bpf_crypto_encrypt
- bpf_ct_change_status
- bpf_ct_change_timeout
- bpf_ct_insert_entry
- bpf_ct_release
- bpf_ct_set_nat_info
- bpf_ct_set_status
- bpf_ct_set_timeout
- bpf_dynptr_adjust
- bpf_dynptr_clone
- bpf_dynptr_from_xdp
- bpf_dynptr_is_null
- bpf_dynptr_is_rdonly
- bpf_dynptr_size
- bpf_dynptr_slice
- bpf_dynptr_slice_rdwr

Map types (Linux)

Generic map types

These map types are not limited to a very specific use case but can be used in a number of different use cases to store data.

- `BPF_MAP_TYPE_HASH`
- `BPF_MAP_TYPE_ARRAY`
- `BPF_MAP_TYPE_PERCPU_HASH`
- `BPF_MAP_TYPE_PERCPU_ARRAY`
- `BPF_MAP_TYPE_QUEUE`
- `BPF_MAP_TYPE_STACK`
- `BPF_MAP_TYPE_LRU_HASH`
- `BPF_MAP_TYPE_LRU_PERCPU_HASH`
- `BPF_MAP_TYPE_LPM_TRIE`
- `BPF_MAP_TYPE_BLOOM_FILTER`
- `BPF_MAP_TYPE_ARENA`

Map in map

These map types hold references to other map types as their values.

- `BPF_MAP_TYPE_ARRAY_OF_MAPS`
- `BPF_MAP_TYPE_HASH_OF_MAPS`

Map type `BPF_MAP_TYPE_HASH`

v3.19

The hash map type is a generic map type with no restrictions on the structure of the key and value. Hash-maps are implemented using a hash table, allowing for lookups with arbitrary keys.

Attributes

While the size of the key and value are essentially unrestricted both `value_size` and `key_size` must be at least zero and their combined size no larger than `KMALLOC_MAX_SIZE`. `KMALLOC_MAX_SIZE` is the maximum size which can be allocated by the kernel memory allocator, its exact value being dependant on a number of factors. If this edge case is hit a `-E2BIG` error number is returned to the map create syscall.

Syscall commands

The following syscall commands work with this map type:

- `BPF_MAP_LOOKUP_ELEM`
- `BPF_MAP_LOOKUP_AND_DELETE_ELEM`
- `BPF_MAP_UPDATE_ELEM`
- `BPF_MAP_GET_NEXT_KEY`
- `BPF_MAP_LOOKUP_BATCH`
- `BPF_MAP_LOOKUP_AND_DELETE_BATCH`

Flags

Back to top

The following flags are supported by this map type.

`BPF_F_NO_PREALLOC`

v4.6

Hash maps are pre-allocated by default, this means that even a completely empty hash map will use the same amount of kernel memory as a full map.

If this flag is set, pre-allocation is disabled. Users might consider this for large maps since allocating large amounts of memory takes a lot of time during creation and might be undesirable.

Warning

The patch set[1] does note that not pre-allocating may cause issues in some edge-cases, which was the original reason for defaulting to pre-allocation.

`BPF_F_NUMA_NODE`

v4.14

When set, the `numa_node` attribute is respected during map creation.

`BPF_F_RDONLY`

v4.15

Setting this flag will make it so the map can only be read via the syscall interface, but not written to.

For details please check the generic description.

Map helpers

These are helpers with the primary purpose involves the interaction with a map.

Generic map helpers

These helpers can be used on a lot of different maps, especially the generic map types like array and hash maps.

- `bpf_map_lookup_elem`
- `bpf_map_update_elem`
- `bpf_map_delete_elem`
- `bpf_for_each_map_elem`
- `bpf_map_lookup_percpu_elem`
- `bpf_spin_lock`
- `bpf_spin_unlock`

Perf event array helpers

These helpers are used with `BPF_MAP_TYPE_PERF_EVENT_ARRAY` maps.

- `bpf_perf_event_read`
- `bpf_perf_event_output`
- `bpf_perf_event_read_value`
- `bpf_skb_output`
- `bpf_xdp_output`

Tail call helpers

Probe and trace helpers

These helpers are used in probing and tracing functions like kprobes, tracepoints and uprobes.

- `bpf_get_attach_cookie`

Memory helpers

These helpers are used to read from or write to kernel or userspace memory.

- `bpf_probe_read`
- `bpf_probe_write_user`
- `bpf_probe_read_str`
- `bpf_get_stack`
- `bpf_probe_read_user`
- `bpf_probe_read_kernel`
- `bpf_probe_read_user_str`
- `bpf_probe_read_kernel_str`
- `bpf_copy_from_user`
- `bpf_copy_from_user_task`
- `bpf_copy_from_user_task`
- `bpf_find_vma`

Process influencing helpers

These helpers are used to influence processes.

Helper function `bpf_ringbuf_output`

v5.8

Definition

> Copyright (c) 2015 The Libbpf Authors. All rights reserved.

Copy *size* bytes from *data* into a ring buffer *ringbuf*. If **BPF_RB_NO_WAKEUP** is specified in *flags*, no notification of new data availability is sent. If **BPF_RB_FORCE_WAKEUP** is specified in *flags*, notification of new data availability is sent unconditionally. If **0** is specified in *flags*, an adaptive notification of new data availability is sent.

An adaptive notification is a notification sent whenever the user-space process has caught up and consumed all available payloads. In case the user-space process is still processing a previous payload, then no notification is needed as it will process the newly added payload automatically.

Returns

0 on success, or a negative error in case of failure.

```
static long (* const bpf_ringbuf_output)(void *ringbuf, void *data, __u64 size, __u64 flags) = (void *) 130;
```

Usage

The `ringbuf` must be a pointer to the ring buffer map. `data` is a pointer to the data that needs to be copied into the ring buffer. The `size` argument specifies the number of bytes to be copied. The `flags`

Usage

The `ringbuf` must be a pointer to the ring buffer map. `data` is a pointer to the data that needs to be copied into the ring buffer. The `size` argument specifies the number of bytes to be copied. The `flags` argument defines how the notification of the new data availability should be handled.

This function incurs an extra memory copy operation in comparison to using `bpf_ringbuf_reserve` / `bpf_ringbuf_submit` / `bpf_ringbuf_discard`, but allows submitting records of lengths unknown to the verifier.

Program types

This helper call can be used in the following program types:

- `BPF_PROG_TYPE_CGROUP_DEVICE`
- `BPF_PROG_TYPE_CGROUP_SKB`
- `BPF_PROG_TYPE_CGROUP_SOCK`
- `BPF_PROG_TYPE_CGROUP_SOCKOPT`
- `BPF_PROG_TYPE_CGROUP_SOCK_ADDR`
- `BPF_PROG_TYPE_CGROUP_SYSCTL`
- `BPF_PROG_TYPE_FLOW_DISSECTOR`
- `BPF_PROG_TYPE_KPROBE`
- `BPF_PROG_TYPE_LSM`
- `BPF_PROG_TYPE_LWT_IN`
- `BPF_PROG_TYPE_LWT_OUT`
- `BPF_PROG_TYPE_LWT_SEG6LOCAL`
- `BPF_PROG_TYPE_LWT_XMIT`

- `BPF_PROG_TYPE_PERF_EVENT`
- `BPF_PROG_TYPE_RAW_TRACEPOINT`
- `BPF_PROG_TYPE_RAW_TRACEPOINT_WRITABLE`
- `BPF_PROG_TYPE_SCHED_ACT`
- `BPF_PROG_TYPE_SCHED_CLS`
- `BPF_PROG_TYPE_SK_LOOKUP`
- `BPF_PROG_TYPE_SK_MSG`
- `BPF_PROG_TYPE_SK_REUSEPORT`
- `BPF_PROG_TYPE_SK_SKB`
- `BPF_PROG_TYPE_SOCKET_FILTER`
- `BPF_PROG_TYPE_SOCK_OPS`
- `BPF_PROG_TYPE_STRUCT_OPS`
- `BPF_PROG_TYPE_SYSCALL`
- `BPF_PROG_TYPE_TRACEPOINT`
- `BPF_PROG_TYPE_TRACING`
- `BPF_PROG_TYPE_XDP`

Example

```c
// Copy data into the ring buffer
bpf_ringbuf_output(&my_ringbuf, &my_data, sizeof(my_data), 0);
```

August 25, 2024   January 25, 2023   GitHub

cGroup resource statistic KFuncs

These KFuncs are used to update or flush cGroup resource statistics efficiently.

- `cgroup_rstat_updated`
- `cgroup_rstat_flush`

Key signature verification KFuncs

These KFuncs are used to verify PKCS#7 signed data against keys from a key-ring.

- `bpf_lookup_user_key`
- `bpf_lookup_system_key`
- `bpf_key_put`
- `bpf_verify_pkcs7_signature`

File related kfuncs

- `bpf_get_file_xattr`

CPU mask KFuncs

- `bpf_cpumask_create`
- `bpf_cpumask_release`
- `bpf_cpumask_acquire`
- `bpf_cpumask_first`

KFunc `bpf_crypto_ctx_create`

v6.10

Create a mutable BPF crypto context.

Definition

Allocates a crypto context that can be used, acquired, and released by a BPF program. The crypto context returned by this function must either be embedded in a map as a kptr, or freed with `bpf_crypto_ctx_release`. As crypto API functions use `GFP_KERNEL` allocations, this function can only be used in sleepable BPF programs.

`params`: pointer to struct bpf_crypto_params which contains all the details needed to initialise crypto context.

`params__sz`: size of steuct bpf_crypto_params usef by bpf program

`err`: integer to store error code when NULL is returned.

Returns

Returns an allocated crypto context on success, may return NULL if no memory is available.

`struct bpf_crypto_ctx *bpf_crypto_ctx_create(const struct bpf_crypto_params *params, u32 params__sz, int *err)`

Note

This kfunc returns a pointer to a refcounted object. The verifier will then ensure that the pointer to the object is eventually

Returns

Returns an allocated crypto context on success, may return NULL if no memory is available.

`struct bpf_crypto_ctx *bpf_crypto_ctx_create(const struct bpf_crypto_params *params, u32 params__sz, int *err)`

Note

This kfunc returns a pointer to a refcounted object. The verifier will then ensure that the pointer to the object is eventually released using a release kfunc, or transferred to a map using a referenced kptr (by invoking `bpf_kptr_xchg`). If not, the verifier fails the loading of the BPF program until no lingering references remain in all possible explored states of the program.

Note

The pointer returned by the kfunc may be NULL. Hence, it forces the user to do a NULL check on the pointer returned from the kfunc before making use of it (dereferencing or passing to another helper).

Note

This function may sleep, and therefore can only be used from sleepable programs.

Usage

This kfunc is used to allocate a new BPF crypto context which can then be used in `bpf_crypto_encrypt` and `bpf_crypto_decrypt` to encrypt or decrypt network packets. The creation allocates memory and thus may sleep, so this must be done outside of packet processing context in a syscall program.

The created context can be stored and shared with network programs via a map containing a kernel pointer.

Usage

This kfunc is used to allocate a new BPF crypto context which can then be used in `bpf_crypto_encrypt` and `bpf_crypto_decrypt` to encrypt or decrypt network packets. The creation allocates memory and thus may sleep, so this must be done outside of packet processing context in a syscall program.

The created context can be stored and shared with network programs via a map containing a kernel pointer.

Program types

The following program types can make use of this kfunc:

- `BPF_PROG_TYPE_SYSCALL`

Example

Example

```
/* Copyright (c) 2024 Meta Platforms, Inc. and affiliates. */

#include "vmlinux.h"
#include "bpf_tracing_net.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>
#include <bpf/bpf_tracing.h>
#include "bpf_misc.h"
#include "bpf_kfuncs.h"

struct bpf_crypto_ctx *bpf_crypto_ctx_create(const struct bpf_crypto_params *params,
                          u32 params__sz, int *err) __ksym;
struct bpf_crypto_ctx *bpf_crypto_ctx_acquire(struct bpf_crypto_ctx *ctx) __ksym;
void bpf_crypto_ctx_release(struct bpf_crypto_ctx *ctx) __ksym;

struct __crypto_ctx_value {
    struct bpf_crypto_ctx __kptr * ctx;
};
```

Example

Back to top

Example

```c
/* Copyright (c) 2024 Meta Platforms, Inc. and affiliates. */

#include "vmlinux.h"
#include "bpf_tracing_net.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>
#include <bpf/bpf_tracing.h>
#include "bpf_misc.h"
#include "bpf_kfuncs.h"

struct bpf_crypto_ctx *bpf_crypto_ctx_create(const struct bpf_crypto_params *params,
                        u32 params__sz, int *err) __ksym;
struct bpf_crypto_ctx *bpf_crypto_ctx_acquire(struct bpf_crypto_ctx *ctx) __ksym;
void bpf_crypto_ctx_release(struct bpf_crypto_ctx *ctx) __ksym;

struct __crypto_ctx_value {
    struct bpf_crypto_ctx __kptr * ctx;
};

struct array_map {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __type(key, int);
    __type(value, struct __crypto_ctx_value);
    __uint(max_entries, 1);
} __crypto_ctx_map SEC(".maps");

static inline int crypto_ctx_insert(struct bpf_crypto_ctx *ctx)
{
    struct __crypto_ctx_value local, *v;
    struct bpf_crypto_ctx *old;
    u32 key = 0;
    int err;

    local.ctx = NULL;
    err = bpf_map_update_elem(&__crypto_ctx_map, &key, &local, 0);
    if (err) {
        bpf_crypto_ctx_release(ctx);
        return err;
    }

    v = bpf_map_lookup_elem(&__crypto_ctx_map, &key);
    if (!v) {
        bpf_crypto_ctx_release(ctx);
```

Linux eBPF concepts

This is an index of Linux specific eBPF concepts and features. For more generic eBPF concepts that are not Linux specific, see the eBPF concepts page.

Maps

Maps allow for data storage and communication

→ Maps

Verifier

The verifier checks the safety of eBPF programs

→ Verifier

Functions

This page explains how functions work for eBPF on Linux

→ Functions

Concurrency

This page explains the effects of concurrency on eBPF programs and how to handle it

→ Concurrency

Pinning

Pinning allows the file system to reference eBPF objects and keep them alive

→ Pinning

Tail calls

Tail calls allow for the chaining of eBPF programs

→ Tail calls

Loops

Loops in eBPF are not trivial, this page explains how to use different types of loops

→ Loops

Timers

Timers allow for the scheduling of eBPF functions to execute at a later time

→ Timers

Resource Limit

This page explains how the Linux kernel counts and restricts the resources used by eBPF

→ Resource Limit

Loops in BPF

Loops in programming is a common concept, however, in BPF they can be a bit more complicated than in most environments. This is due to the verifier and the guaranteed "safe" nature of BPF programs.

Unrolling

Before v5.3 loops in BPF bytecode were not allowed because the verifier wasn't smart enough to determine if a loop would always terminate. The workaround for a long time was to unroll loops in the compiler. Unrolling loops increases the size of a program and can only be done if the amount of iterations is known at compile time. To unroll a loop you can use the `#pragma unroll` pragma as such:

```
#pragma unroll
for (int i = 0; i < 10; i++) {
    // do something
}
```

Bounded loops

Since v5.3 the verifier is smart enough to determine if a loop will stop or not. These are referred to as "bounded loops". Users of this feature still have to be careful though, because its easy to write a loop which makes your program too complex for the verifier to handle. The verifier will check every possible permutation of a loop, so if you have a loop that goes up to 100 times with a body of 20 instructions and a few branches, then that loop counts for a few thousand instructions towards the complexity limit.

A common mistake is to use variables with a huge range as the bounds for a loop. For example:

BPF Token

v6.9

BPF Token is a mechanism for delegating some of the BPF subsystem functionalities to an unprivileged process (e.g., container) within user-namespace from a privileged process (e.g., container runtime) in the init-namespace.

eBPF and the Linux Capabilities

When eBPF was first introduced in the Linux kernel, it required `CAP_SYS_ADMIN` to load programs, create maps, etc. However, having `CAP_SYS_ADMIN` for eBPF applications gives too many privileges beyond just interacting with the BPF subsystem.

That's why `CAP_BPF` has been introduced since v5.8. It allows more granular control of which eBPF functionalities the process can use. For example, to load network-related eBPF programs such as TC or XDP, it requires `CAP_BPF + CAP_NET_ADMIN`; to load tracing-related eBPF programs like kprobe or raw_tracepoint, it requires `CAP_BPF + CAP_PERFMON` and so on.

eBPF and User Namespaces

User Namespace is a namespace that isolates UID, GID, keys, and capabilities. Within the User Namespace, a process can behave like having the root privilege, but only for the resources governed by the User Namespaces.

For example, the process within the User Namespace can even have a `CAP_NET_ADMIN` and create a new Network Namespace and network devices within it. However, it cannot connect a `veth` device with

Dynptrs (Dynamic Pointers)

A "dynptr" or "dynamic pointer" is a concept in the Linux eBPF verifier. It is a pointer with additional metadata so that certain safety check can be performed at runtime. This is useful for situations where it might be difficult to statically prove the safety of certain actions.

For example, consider the situation where you have a map with a setting that instructs a program to read or write an arbitrary spot in a packet. Since the map value can be any value and the size of the packet is variable as well, it is challenging to statically prove all cases (though likely not impossible). By using dynptrs, we shift the burden of proof to runtime. If the program tries to access memory outside of the packet, the helper function or kfunc will return an error instead.

To a eBPF program, a dynptr is just an opaque pointer. The verifier will not allow the program to dereference it directly. Instead, the program must use helper functions or kfuncs to access the memory it points to. These functions will perform the necessary safety checks.

Helper functions and kfuncs

The following functions create or manipulate dynptrs:

- `bpf_dynptr_from_mem` - Creates a dynptr for a map value or global variable.
- `bpf_dynptr_read` - Attempts to read from a dynptr.
- `bpf_dynptr_write` - Attempts to write to a dynptr.
- `bpf_dynptr_data` - Returns a pointer to the underlying data of a dynptr of a given length at a given offset. The verifier knows about the length and offset and will enforce bounds checks statically.
- `bpf_ringbuf_reserve_dynptr` - Reserves a sample in a ring buffer as dynptr, allowing for runtime

Libbpf eBPF side

Libbpf contains a number of C header files containing mostly pre-processor macros, forward declarations and type definitions that make it easier to write eBPF programs. This is an index into these useful definitions.

bpf_helper_defs.h

The `bpf_helper_defs.h` file is automatically generated from the kernel sources. It contains forward declarations for every type that is used by eBPF helper functions and somewhat special forward declarations for the helper functions themselves.

For example, the `bpf_map_lookup_elem` function is declared as:

```
static void *(* const bpf_map_lookup_elem)(void *map, const void *key) = (void *) 1;
```

The normal forward declaration of this function would be

`void *bpf_map_lookup_elem(void *map, const void *key);` .

But what the special declaration does is it casts a pointer of value `1` to a const static function pointer. This causes the compiler to emit a `call 1` instruction which the kernel recognizes as a call to the `bpf_map_lookup_elem` function.

It is entirely possible to copy parts of this file if you are only interested in specific helper functions and their types and even modify their definitions to suit your needs. Though for most people it will be best to include the whole file.

bpf_helpers.h

The `bpf_helpers.h` file is the single most useful file in the eBPF side of the libbpf library. It contains a lot of generic and basic definitions you will use in almost any eBPF program. It also includes the `bpf_helper_defs.h` file, so you don't need to include it separately.

The file contains definitions for the following:

- BTF map macros / types
  - `__uint`
  - `__uint`
  - `__array`
  - `__ulong`
  - `enum libbpf_pin_type`
- Attributes
  - `__always_inline`
  - `__noinline`
  - `__weak`
  - `__hidden`
  - `__kconfig`
  - `__ksym`
  - `__kptr_untrusted`
  - `__kptr`
  - `__percpu_kptr`
- Global function attributes

bpf_core_read.h

The `bpf_core_read.h` file contains macros for CO-RE(Compile Once Run Everywhere) operations.

The file contains definitions for the following:

- CO-RE memory access
  - `BPF_CORE_READ`
  - `BPF_CORE_READ_INTO`
  - `bpf_core_read`
  - `BPF_CORE_READ_STR_INTO`
  - `bpf_core_read_str`
  - `BPF_CORE_READ_USER`
  - `BPF_CORE_READ_USER_INTO`
  - `bpf_core_read_user`
  - `BPF_CORE_READ_USER_STR_INTO`
  - `bpf_core_read_user_str`
  - `BPF_CORE_READ_BITFIELD`
  - `BPF_CORE_READ_BITFIELD_PROBED`
  - `BPF_CORE_WRITE_BITFIELD`
- CO-RE queries
  - `bpf_core_field_exists`
  - `bpf_core_field_size`
  - `bpf_core_field_offset`
  - `bpf_core_type_id_local`

Behind the curtain

- Mkdocs + Material for MkDocs + Plugins
- Docker
- GHA + Github pages
- Tools + data files

GitHub Pages

Material for MkDocs

Documentation that simply works

Tools: version-finder + feature-gen

- YAML config
- git grep
- git bisect
- <!-- [FEATURE_TAG](...) --><!-- [/FEATURE_TAG] -->

Program type `BPF_PROG_TYPE_XDP`

v4.8

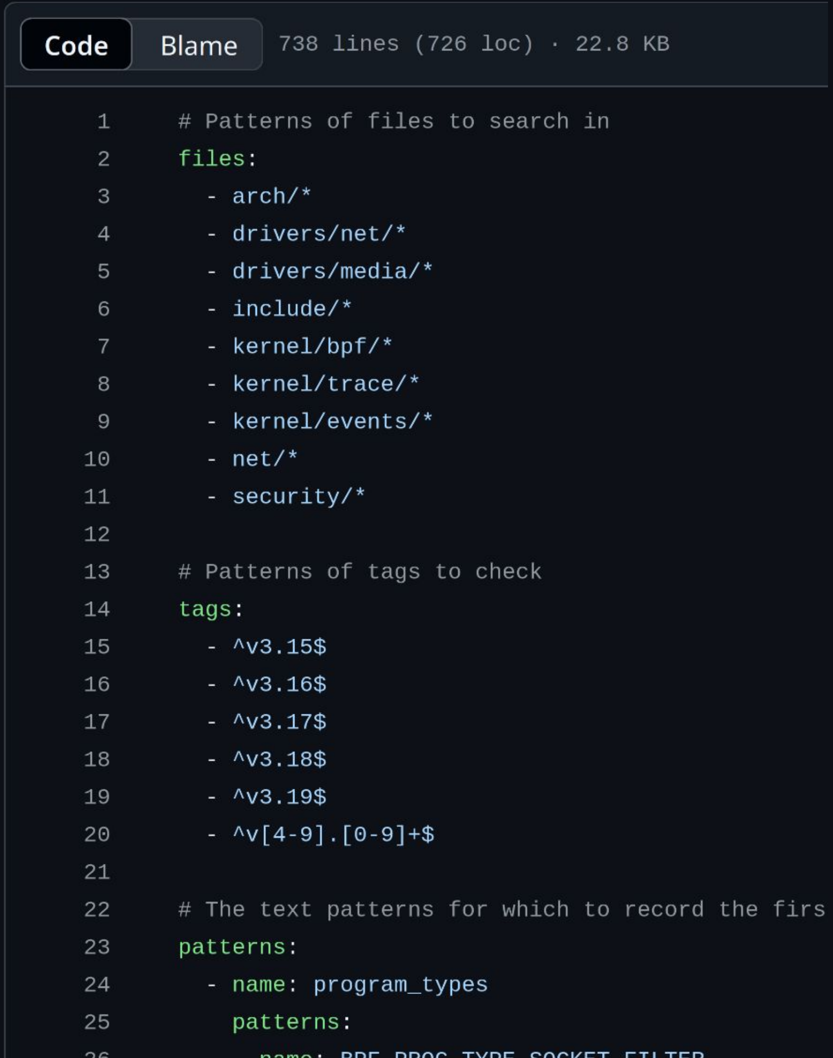
XDP (Express Data Path) programs can attach to network devices and are called for every incoming (ingress) packet received by that network device. XDP programs can take quite a large number of actions, most prominent of which are manipulation of the packet, dropping the packet, redirecting it and letting it pass to the network stack.

Notable use cases for XDP programs are for DDoS protection, Load Balancing, and high-throughput packet filtering. If loaded with native driver support, XDP programs will be called just after receiving the packet but before allocating memory for a socket buffer. This call site makes XDP programs extremely performant, especially in use cases where traffic is forwarded or dropped a lot in comparison to other eBPF program types or techniques which run after the relatively expensive socket buffer allocation process has taken place, only to discard it.

Usage

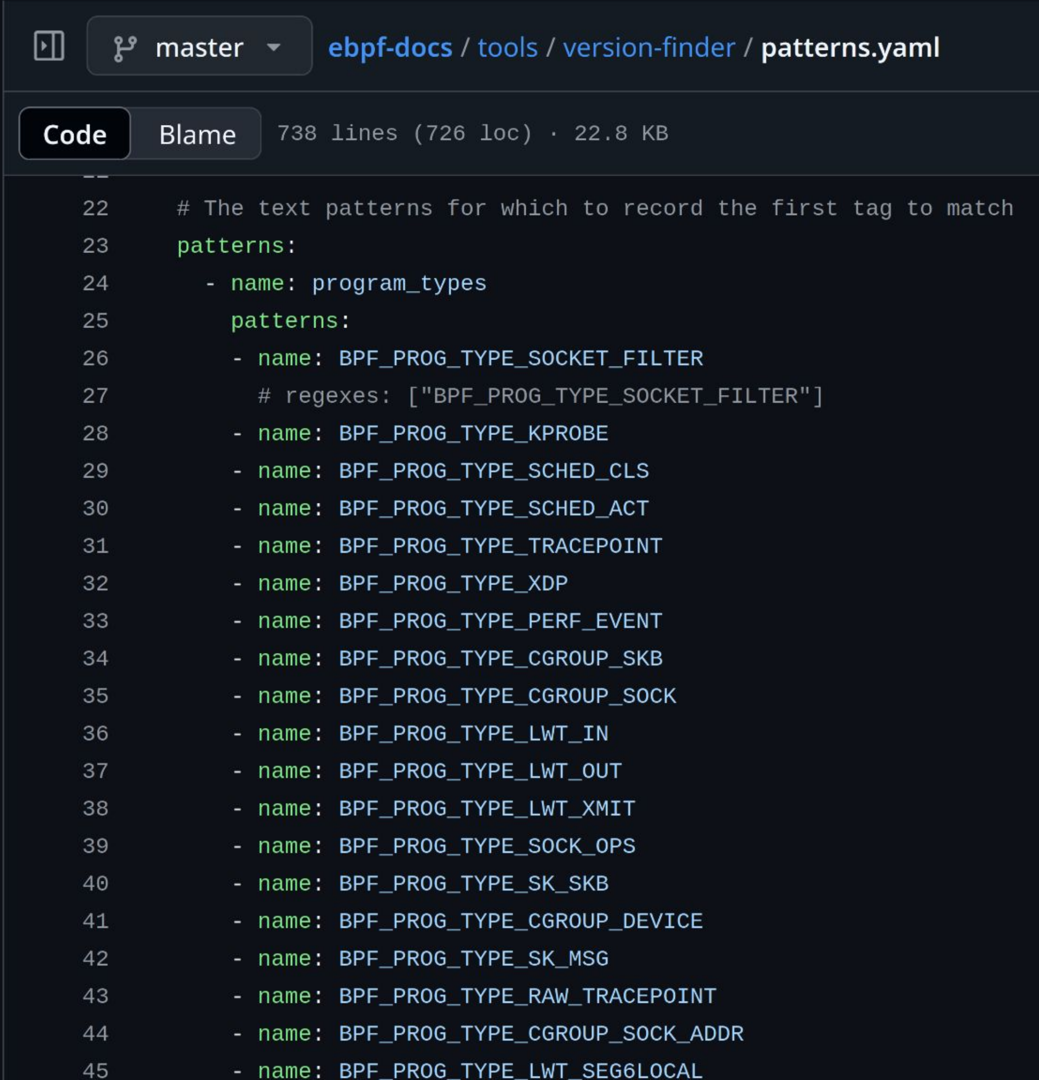
XDP programs are typically put into an ELF section prefixed with `xdp`. The XDP program is called by the kernel with a `xdp_md` context. The return value indicates what action the kernel should take with the packet, the following values are permitted:

- `XDP_ABORTED` - Signals that a unrecoverable error has taken place. Returning this action will cause the kernel to trigger the `xdp_exception` tracepoint and print a line to the trace log. This allows for debugging of such occurrences. It is also expensive, so should not be used without consideration in production.
- `XDP_DROP` - Discards the packet. It should be noted that since we drop the packet very early, it will



738 lines (726 loc) · 22.8 KB

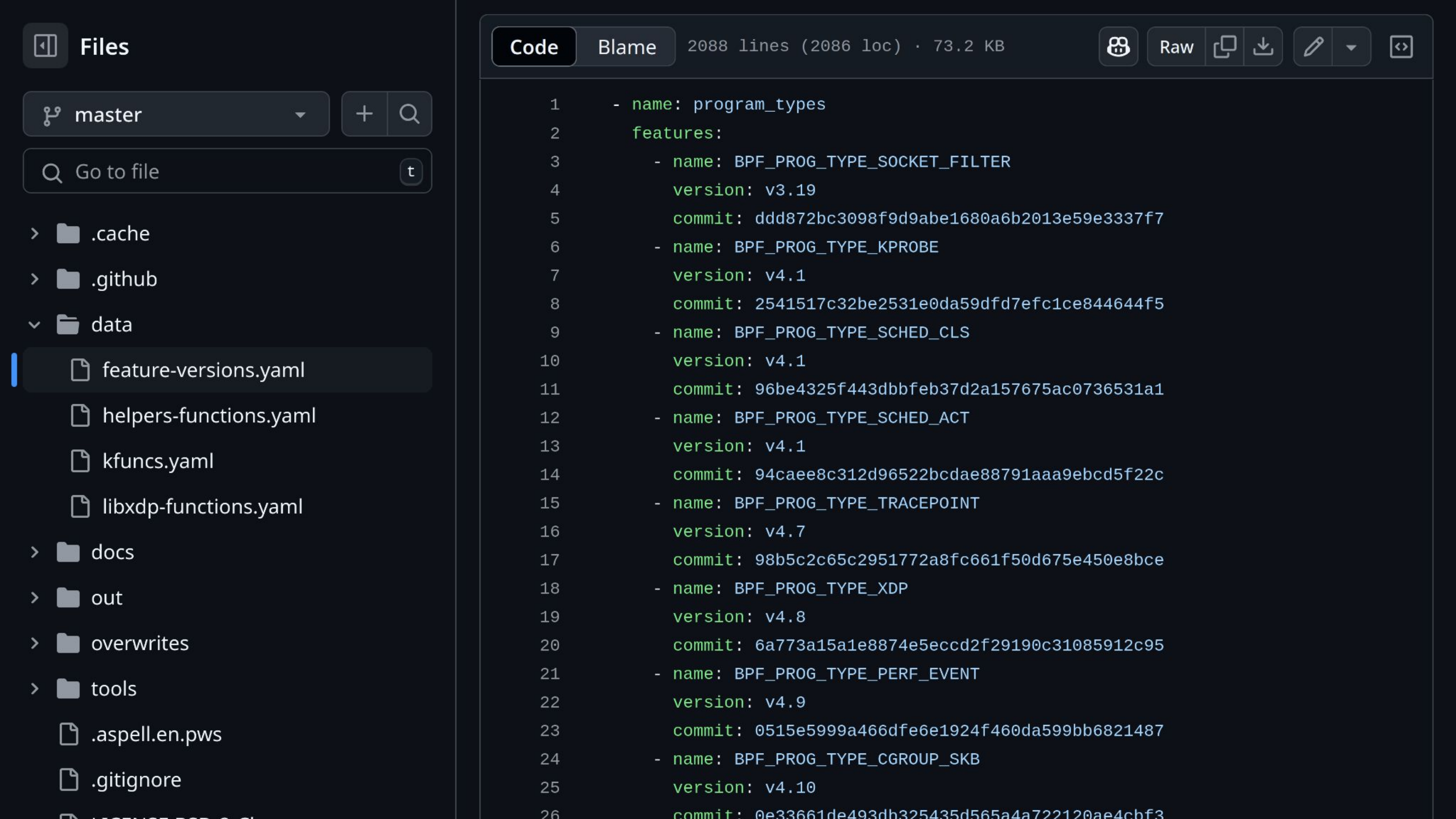
```yaml
# Patterns of files to search in
files:
  - arch/*
  - drivers/net/*
  - drivers/media/*
  - include/*
  - kernel/bpf/*
  - kernel/trace/*
  - kernel/events/*
  - net/*
  - security/*

# Patterns of tags to check
tags:
  - ^v3.15$
  - ^v3.16$
  - ^v3.17$
  - ^v3.18$
  - ^v3.19$
  - ^v[4-9].[0-9]+$

# The text patterns for which to record the firs
patterns:
  - name: program_types
    patterns:
```



ebpf-docs / tools / version-finder / patterns.yaml

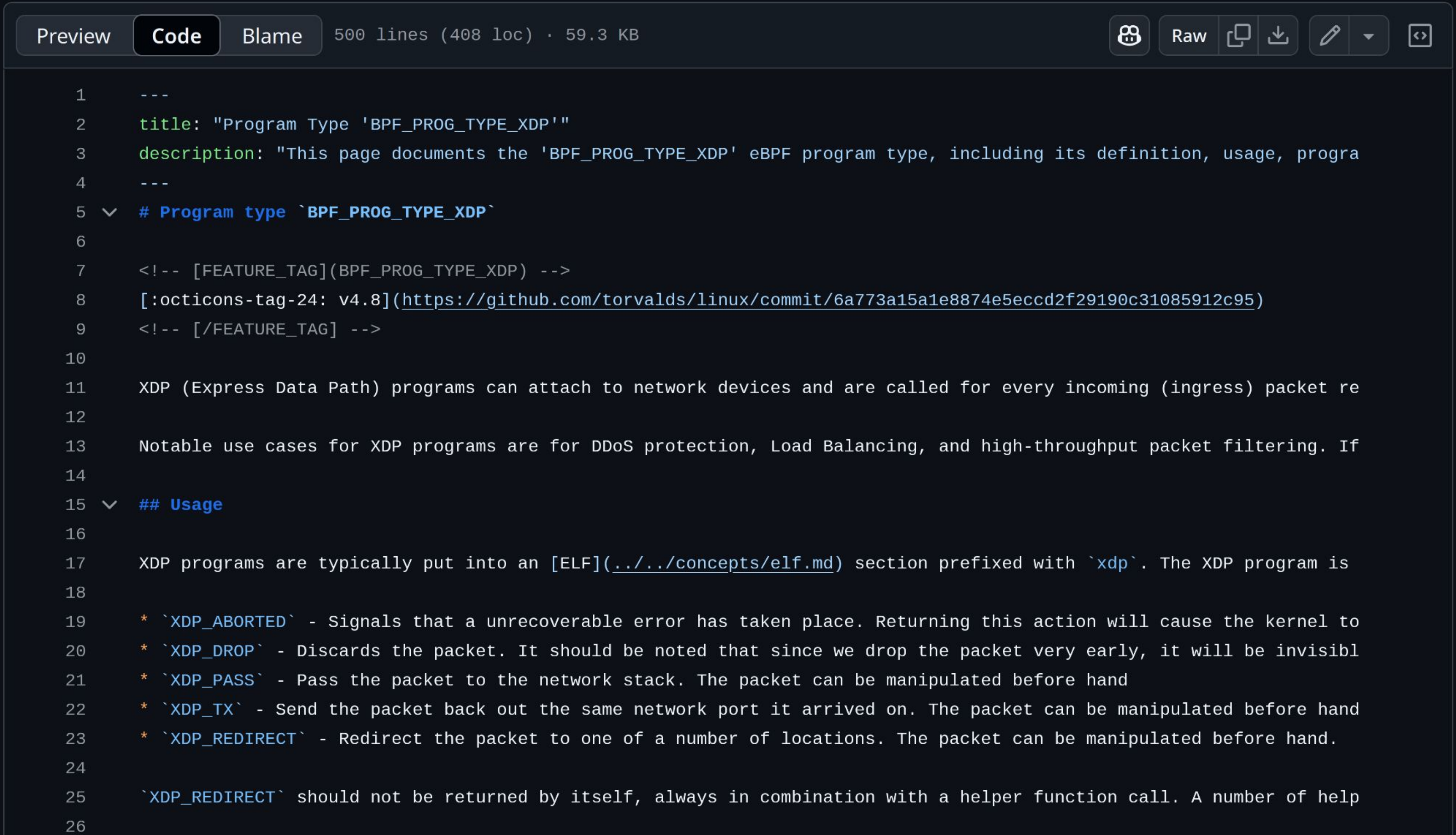
738 lines (726 loc) · 22.8 KB

```yaml
# The text patterns for which to record the first tag to match
patterns:
  - name: program_types
    patterns:
    - name: BPF_PROG_TYPE_SOCKET_FILTER
      # regexes: ["BPF_PROG_TYPE_SOCKET_FILTER"]
    - name: BPF_PROG_TYPE_KPROBE
    - name: BPF_PROG_TYPE_SCHED_CLS
    - name: BPF_PROG_TYPE_SCHED_ACT
    - name: BPF_PROG_TYPE_TRACEPOINT
    - name: BPF_PROG_TYPE_XDP
    - name: BPF_PROG_TYPE_PERF_EVENT
    - name: BPF_PROG_TYPE_CGROUP_SKB
    - name: BPF_PROG_TYPE_CGROUP_SOCK
    - name: BPF_PROG_TYPE_LWT_IN
    - name: BPF_PROG_TYPE_LWT_OUT
    - name: BPF_PROG_TYPE_LWT_XMIT
    - name: BPF_PROG_TYPE_SOCK_OPS
    - name: BPF_PROG_TYPE_SK_SKB
    - name: BPF_PROG_TYPE_CGROUP_DEVICE
    - name: BPF_PROG_TYPE_SK_MSG
    - name: BPF_PROG_TYPE_RAW_TRACEPOINT
    - name: BPF_PROG_TYPE_CGROUP_SOCK_ADDR
    - name: BPF_PROG_TYPE_LWT_SEG6LOCAL
```



2088 lines (2086 loc) · 73.2 KB

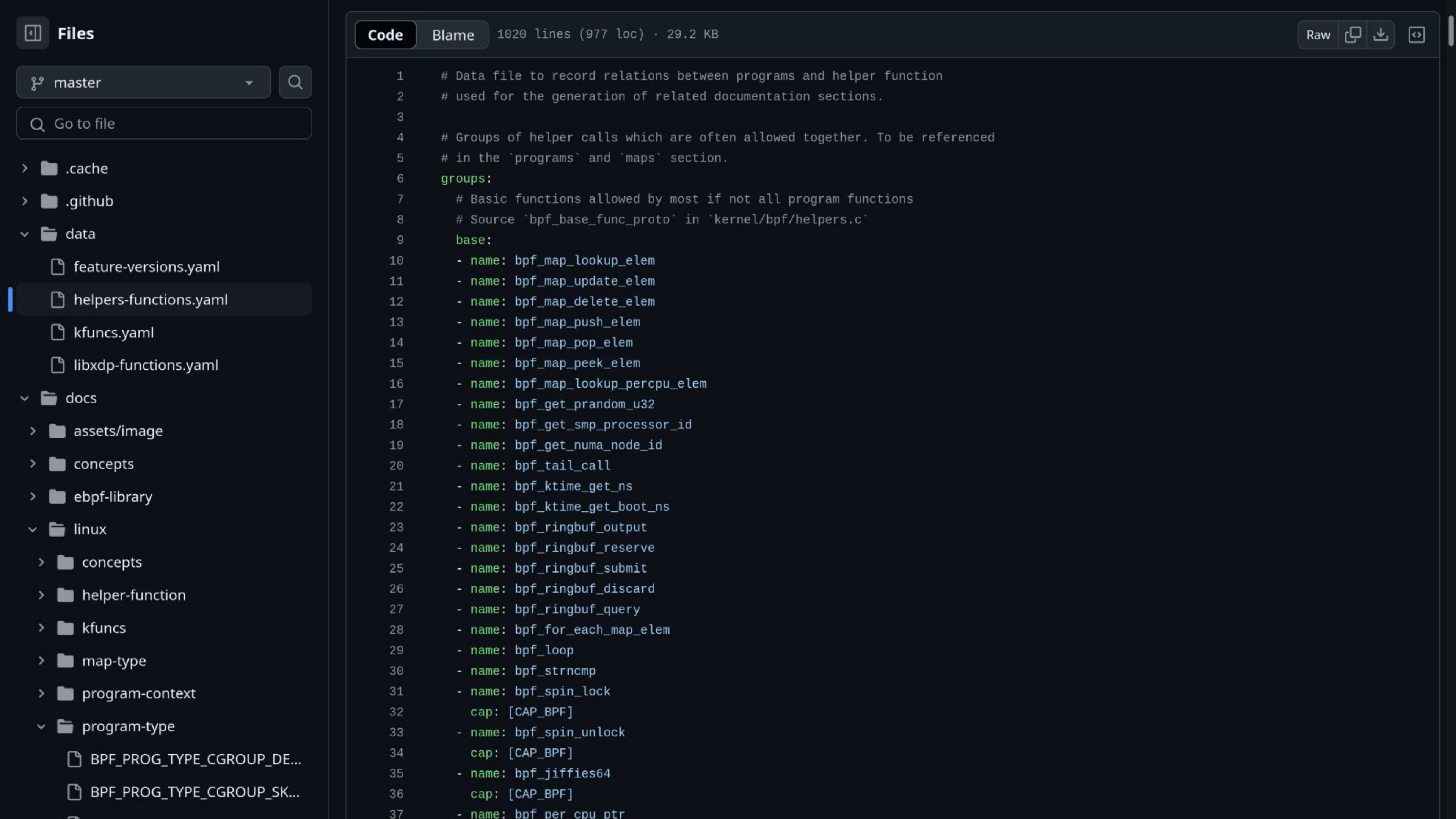
```yaml
- name: program_types
  features:
    - name: BPF_PROG_TYPE_SOCKET_FILTER
      version: v3.19
      commit: ddd872bc3098f9d9abe1680a6b2013e59e3337f7
    - name: BPF_PROG_TYPE_KPROBE
      version: v4.1
      commit: 2541517c32be2531e0da59dfd7efc1ce844644f5
    - name: BPF_PROG_TYPE_SCHED_CLS
      version: v4.1
      commit: 96be4325f443dbbfeb37d2a157675ac0736531a1
    - name: BPF_PROG_TYPE_SCHED_ACT
      version: v4.1
      commit: 94caee8c312d96522bcdae88791aaa9ebcd5f22c
    - name: BPF_PROG_TYPE_TRACEPOINT
      version: v4.7
      commit: 98b5c2c65c2951772a8fc661f50d675e450e8bce
    - name: BPF_PROG_TYPE_XDP
      version: v4.8
      commit: 6a773a15a1e8874e5eccd2f29190c31085912c95
    - name: BPF_PROG_TYPE_PERF_EVENT
      version: v4.9
      commit: 0515e5999a466dfe6e1924f460da599bb6821487
    - name: BPF_PROG_TYPE_CGROUP_SKB
      version: v4.10
      commit: 0e33661de493db325435d565a4a722120ae4cbf3
```



```
---
title: "Program Type 'BPF_PROG_TYPE_XDP'"
description: "This page documents the 'BPF_PROG_TYPE_XDP' eBPF program type, including its definition, usage, progra
---
# Program type `BPF_PROG_TYPE_XDP`

<!-- [FEATURE_TAG](BPF_PROG_TYPE_XDP) -->
[:octicons-tag-24: v4.8](https://github.com/torvalds/linux/commit/6a773a15a1e8874e5eccd2f29190c31085912c95)
<!-- [/FEATURE_TAG] -->

XDP (Express Data Path) programs can attach to network devices and are called for every incoming (ingress) packet re

Notable use cases for XDP programs are for DDoS protection, Load Balancing, and high-throughput packet filtering. If

## Usage

XDP programs are typically put into an [ELF](../../concepts/elf.md) section prefixed with `xdp`. The XDP program is

* `XDP_ABORTED` - Signals that a unrecoverable error has taken place. Returning this action will cause the kernel to
* `XDP_DROP` - Discards the packet. It should be noted that since we drop the packet very early, it will be invisibl
* `XDP_PASS` - Pass the packet to the network stack. The packet can be manipulated before hand
* `XDP_TX` - Send the packet back out the same network port it arrived on. The packet can be manipulated before hand
* `XDP_REDIRECT` - Redirect the packet to one of a number of locations. The packet can be manipulated before hand.

`XDP_REDIRECT` should not be returned by itself, always in combination with a helper function call. A number of help

```

Tools: helper-ref-gen

- Read kernel sources, manually maintain data file
- Recreate hierarchy of “groups”
- Flatten and apply “since” metadata
- Render on helper pages and program pages


Files

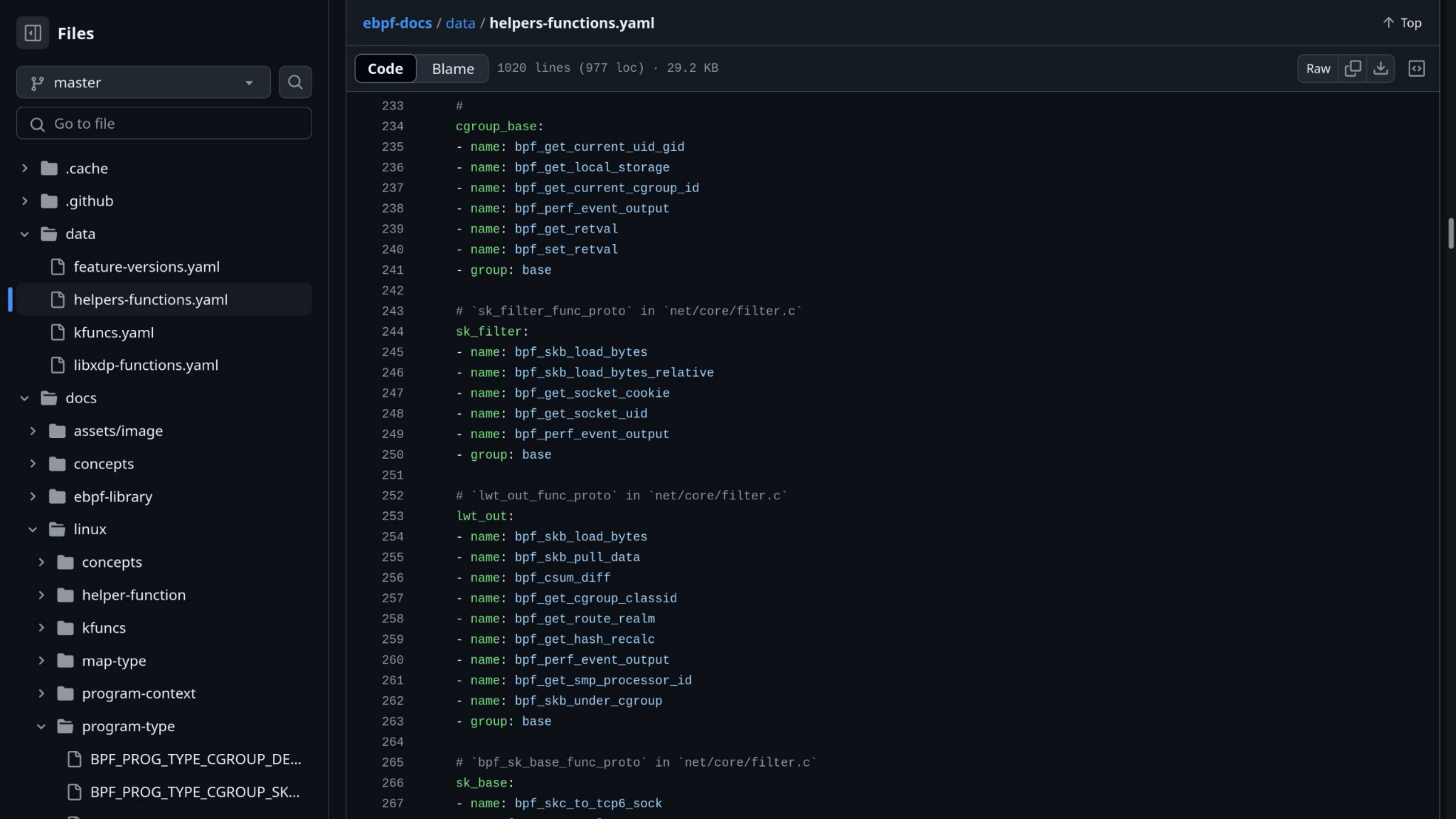
master

Go to file

- .cache
- .github
- data
  - feature-versions.yaml
  - helpers-functions.yaml
  - kfuncs.yaml
  - libxdp-functions.yaml
- docs
  - assets/image
  - concepts
  - ebpf-library
  - linux
    - concepts
    - helper-function
    - kfuncs
    - map-type
    - program-context
    - program-type
      - BPF_PROG_TYPE_CGROUP_DE...
      - BPF_PROG_TYPE_CGROUP_SK...

Code Blame 1020 lines (977 loc) · 29.2 KB Raw

```yaml
# Data file to record relations between programs and helper function
# used for the generation of related documentation sections.

# Groups of helper calls which are often allowed together. To be referenced
# in the `programs` and `maps` section.
groups:
  # Basic functions allowed by most if not all program functions
  # Source `bpf_base_func_proto` in `kernel/bpf/helpers.c`
  base:
  - name: bpf_map_lookup_elem
  - name: bpf_map_update_elem
  - name: bpf_map_delete_elem
  - name: bpf_map_push_elem
  - name: bpf_map_pop_elem
  - name: bpf_map_peek_elem
  - name: bpf_map_lookup_percpu_elem
  - name: bpf_get_prandom_u32
  - name: bpf_get_smp_processor_id
  - name: bpf_get_numa_node_id
  - name: bpf_tail_call
  - name: bpf_ktime_get_ns
  - name: bpf_ktime_get_boot_ns
  - name: bpf_ringbuf_output
  - name: bpf_ringbuf_reserve
  - name: bpf_ringbuf_submit
  - name: bpf_ringbuf_discard
  - name: bpf_ringbuf_query
  - name: bpf_for_each_map_elem
  - name: bpf_loop
  - name: bpf_strncmp
  - name: bpf_spin_lock
    cap: [CAP_BPF]
  - name: bpf_spin_unlock
    cap: [CAP_BPF]
  - name: bpf_jiffies64
    cap: [CAP_BPF]
  - name: bpf_per_cpu_ptr
```


Files

master

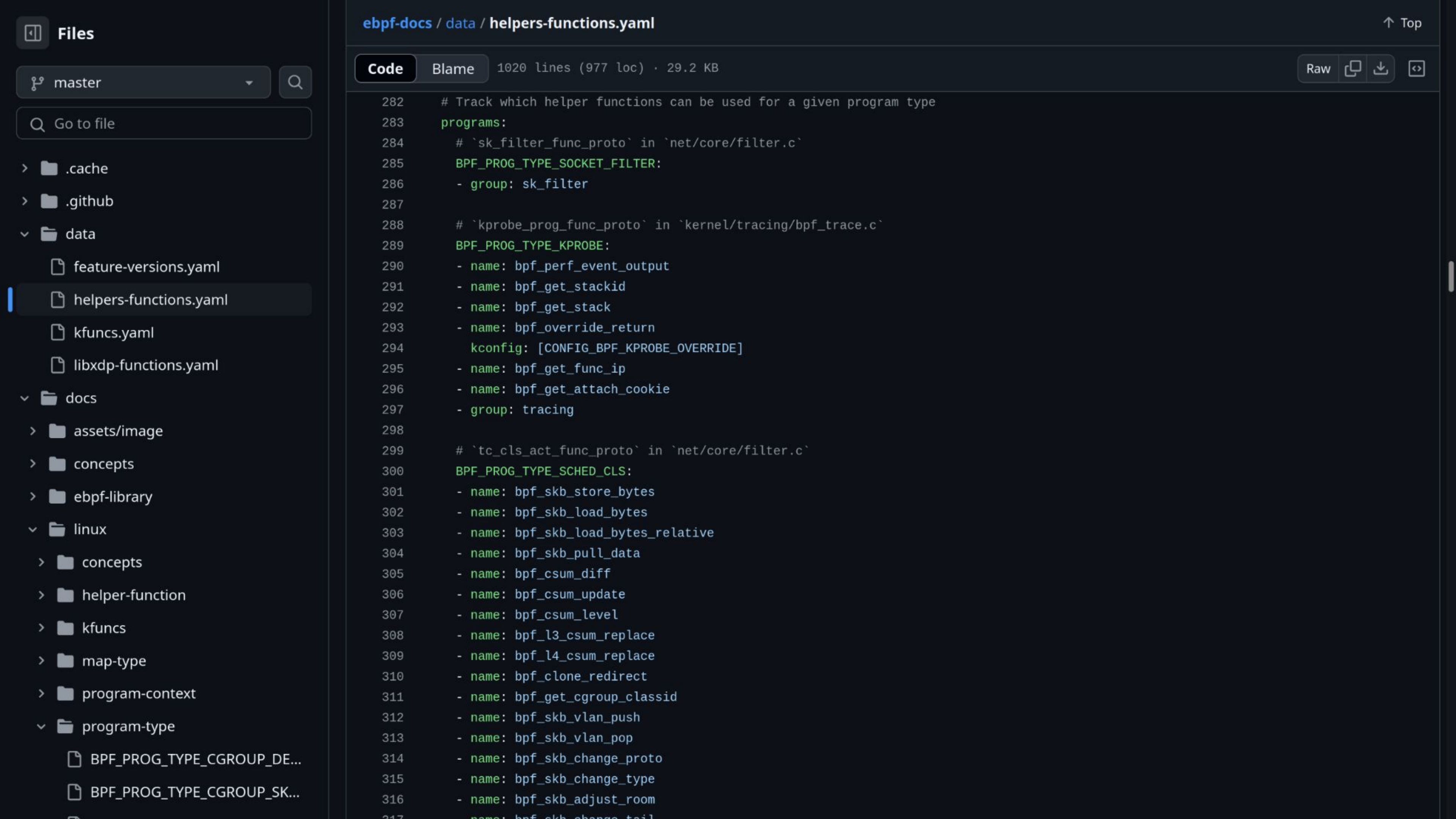
Go to file

- .cache
- .github
- data
  - feature-versions.yaml
  - helpers-functions.yaml
  - kfuncs.yaml
  - libxdp-functions.yaml
- docs
  - assets/image
  - concepts
  - ebpf-library
  - linux
    - concepts
    - helper-function
    - kfuncs
    - map-type
    - program-context
    - program-type
      - BPF_PROG_TYPE_CGROUP_DE...
      - BPF_PROG_TYPE_CGROUP_SK...

ebpf-docs / data / helpers-functions.yaml

Code Blame 1020 lines (977 loc) · 29.2 KB Raw

```yaml
    #
    cgroup_base:
    - name: bpf_get_current_uid_gid
    - name: bpf_get_local_storage
    - name: bpf_get_current_cgroup_id
    - name: bpf_perf_event_output
    - name: bpf_get_retval
    - name: bpf_set_retval
    - group: base

    # `sk_filter_func_proto` in `net/core/filter.c`
    sk_filter:
    - name: bpf_skb_load_bytes
    - name: bpf_skb_load_bytes_relative
    - name: bpf_get_socket_cookie
    - name: bpf_get_socket_uid
    - name: bpf_perf_event_output
    - group: base

    # `lwt_out_func_proto` in `net/core/filter.c`
    lwt_out:
    - name: bpf_skb_load_bytes
    - name: bpf_skb_pull_data
    - name: bpf_csum_diff
    - name: bpf_get_cgroup_classid
    - name: bpf_get_route_realm
    - name: bpf_get_hash_recalc
    - name: bpf_perf_event_output
    - name: bpf_get_smp_processor_id
    - name: bpf_skb_under_cgroup
    - group: base

    # `bpf_sk_base_func_proto` in `net/core/filter.c`
    sk_base:
    - name: bpf_skc_to_tcp6_sock
```



Files

master

Go to file

- .cache
- .github
- data
  - feature-versions.yaml
  - helpers-functions.yaml
  - kfuncs.yaml
  - libxdp-functions.yaml
- docs
  - assets/image
  - concepts
  - ebpf-library
  - linux
    - concepts
    - helper-function
    - kfuncs
    - map-type
    - program-context
    - program-type
      - BPF_PROG_TYPE_CGROUP_DE...
      - BPF_PROG_TYPE_CGROUP_SK...

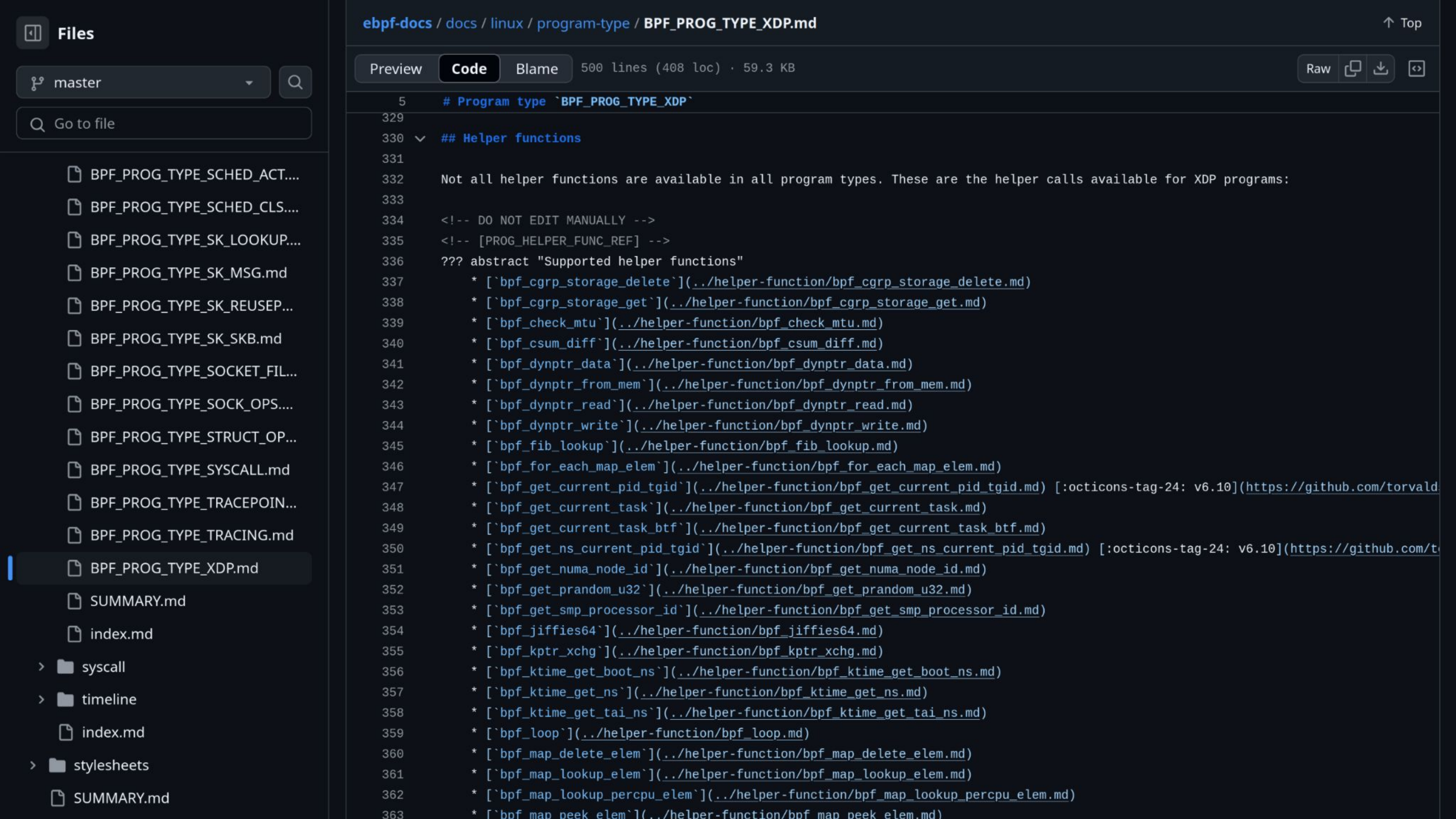
ebpf-docs / data / helpers-functions.yaml

Top

Code Blame 1020 lines (977 loc) · 29.2 KB

Raw

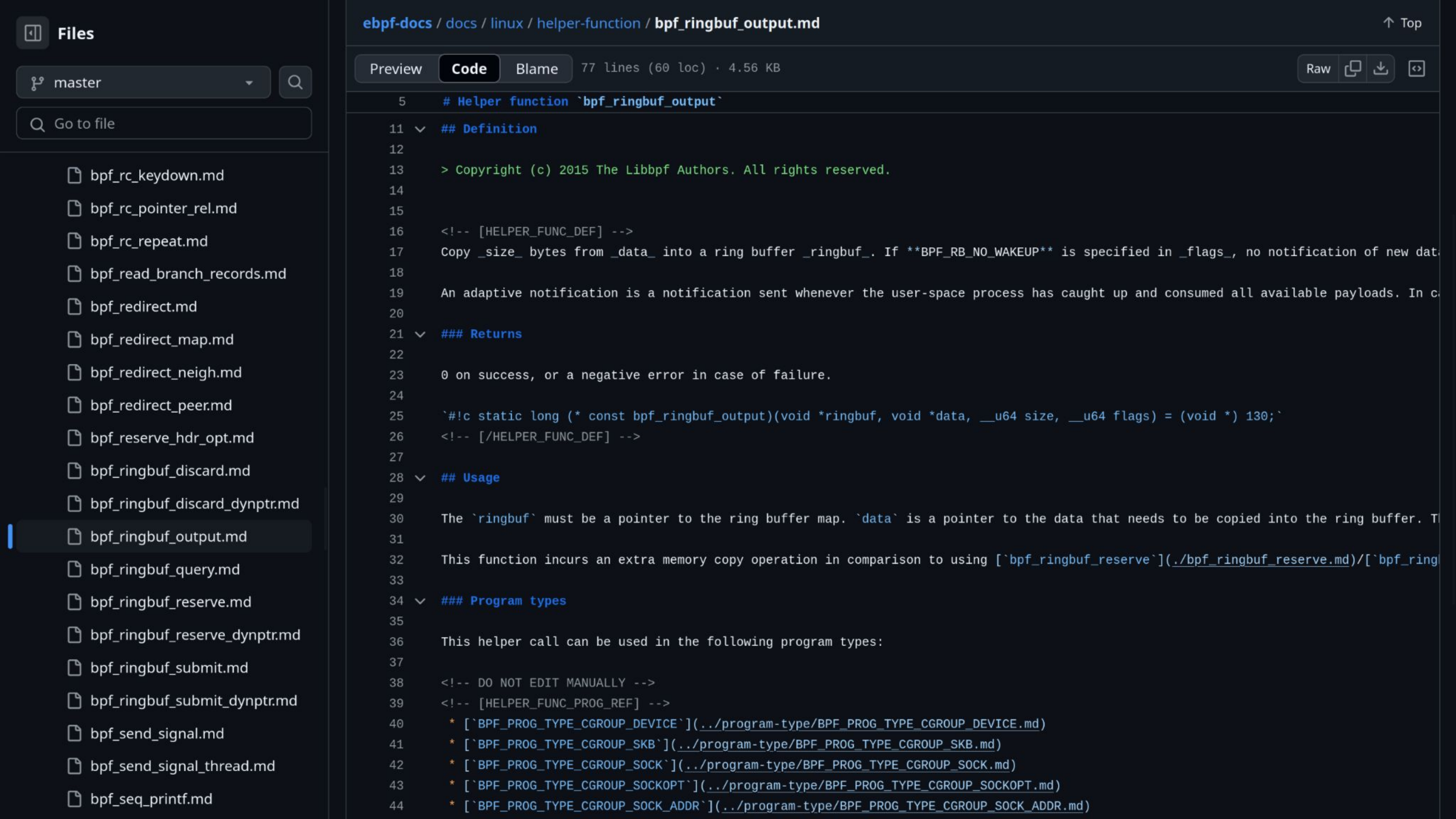
```yaml
# Track which helper functions can be used for a given program type
programs:
  # `sk_filter_func_proto` in `net/core/filter.c`
  BPF_PROG_TYPE_SOCKET_FILTER:
  - group: sk_filter

  # `kprobe_prog_func_proto` in `kernel/tracing/bpf_trace.c`
  BPF_PROG_TYPE_KPROBE:
  - name: bpf_perf_event_output
  - name: bpf_get_stackid
  - name: bpf_get_stack
  - name: bpf_override_return
    kconfig: [CONFIG_BPF_KPROBE_OVERRIDE]
  - name: bpf_get_func_ip
  - name: bpf_get_attach_cookie
  - group: tracing

  # `tc_cls_act_func_proto` in `net/core/filter.c`
  BPF_PROG_TYPE_SCHED_CLS:
  - name: bpf_skb_store_bytes
  - name: bpf_skb_load_bytes
  - name: bpf_skb_load_bytes_relative
  - name: bpf_skb_pull_data
  - name: bpf_csum_diff
  - name: bpf_csum_update
  - name: bpf_csum_level
  - name: bpf_l3_csum_replace
  - name: bpf_l4_csum_replace
  - name: bpf_clone_redirect
  - name: bpf_get_cgroup_classid
  - name: bpf_skb_vlan_push
  - name: bpf_skb_vlan_pop
  - name: bpf_skb_change_proto
  - name: bpf_skb_change_type
  - name: bpf_skb_adjust_room
  - name: bpf_skb_change_tail
```


Files
master
Go to file
BPF_PROG_TYPE_SCHED_ACT....
BPF_PROG_TYPE_SCHED_CLS....
BPF_PROG_TYPE_SK_LOOKUP....
BPF_PROG_TYPE_SK_MSG.md
BPF_PROG_TYPE_SK_REUSEP...
BPF_PROG_TYPE_SK_SKB.md
BPF_PROG_TYPE_SOCKET_FIL...
BPF_PROG_TYPE_SOCK_OPS....
BPF_PROG_TYPE_STRUCT_OP...
BPF_PROG_TYPE_SYSCALL.md
BPF_PROG_TYPE_TRACEPOIN...
BPF_PROG_TYPE_TRACING.md
BPF_PROG_TYPE_XDP.md
SUMMARY.md
index.md
syscall
timeline
index.md
stylesheets
SUMMARY.md
ebpf-docs / docs / linux / program-type / BPF_PROG_TYPE_XDP.md
Top
Preview
Code
Blame
500 lines (408 loc) · 59.3 KB
Raw

```
# Program type `BPF_PROG_TYPE_XDP`

## Helper functions

Not all helper functions are available in all program types. These are the helper calls available for XDP programs:

<!-- DO NOT EDIT MANUALLY -->
<!-- [PROG_HELPER_FUNC_REF] -->
??? abstract "Supported helper functions"
    * [`bpf_cgrp_storage_delete`](../helper-function/bpf_cgrp_storage_delete.md)
    * [`bpf_cgrp_storage_get`](../helper-function/bpf_cgrp_storage_get.md)
    * [`bpf_check_mtu`](../helper-function/bpf_check_mtu.md)
    * [`bpf_csum_diff`](../helper-function/bpf_csum_diff.md)
    * [`bpf_dynptr_data`](../helper-function/bpf_dynptr_data.md)
    * [`bpf_dynptr_from_mem`](../helper-function/bpf_dynptr_from_mem.md)
    * [`bpf_dynptr_read`](../helper-function/bpf_dynptr_read.md)
    * [`bpf_dynptr_write`](../helper-function/bpf_dynptr_write.md)
    * [`bpf_fib_lookup`](../helper-function/bpf_fib_lookup.md)
    * [`bpf_for_each_map_elem`](../helper-function/bpf_for_each_map_elem.md)
    * [`bpf_get_current_pid_tgid`](../helper-function/bpf_get_current_pid_tgid.md) [:octicons-tag-24: v6.10](https://github.com/torvald
    * [`bpf_get_current_task`](../helper-function/bpf_get_current_task.md)
    * [`bpf_get_current_task_btf`](../helper-function/bpf_get_current_task_btf.md)
    * [`bpf_get_ns_current_pid_tgid`](../helper-function/bpf_get_ns_current_pid_tgid.md) [:octicons-tag-24: v6.10](https://github.com/t
    * [`bpf_get_numa_node_id`](../helper-function/bpf_get_numa_node_id.md)
    * [`bpf_get_prandom_u32`](../helper-function/bpf_get_prandom_u32.md)
    * [`bpf_get_smp_processor_id`](../helper-function/bpf_get_smp_processor_id.md)
    * [`bpf_jiffies64`](../helper-function/bpf_jiffies64.md)
    * [`bpf_kptr_xchg`](../helper-function/bpf_kptr_xchg.md)
    * [`bpf_ktime_get_boot_ns`](../helper-function/bpf_ktime_get_boot_ns.md)
    * [`bpf_ktime_get_ns`](../helper-function/bpf_ktime_get_ns.md)
    * [`bpf_ktime_get_tai_ns`](../helper-function/bpf_ktime_get_tai_ns.md)
    * [`bpf_loop`](../helper-function/bpf_loop.md)
    * [`bpf_map_delete_elem`](../helper-function/bpf_map_delete_elem.md)
    * [`bpf_map_lookup_elem`](../helper-function/bpf_map_lookup_elem.md)
    * [`bpf_map_lookup_percpu_elem`](../helper-function/bpf_map_lookup_percpu_elem.md)
    * [`bpf_map_peek_elem`](../helper-function/bpf_map_peek_elem.md)
```

Tools: helper-def-scraper

- Grab `bpf_helper_defs.h` from libbpf
- Parse out the comments
- Convert to markdown
- Insert between <!-- [HELPER_FUNC_DEF] and <!-- [/HELPER_FUNC_DEF] -->



bpf_rc_keydown.md
bpf_rc_pointer_rel.md
bpf_rc_repeat.md
bpf_read_branch_records.md
bpf_redirect.md
bpf_redirect_map.md
bpf_redirect_neigh.md
bpf_redirect_peer.md
bpf_reserve_hdr_opt.md
bpf_ringbuf_discard.md
bpf_ringbuf_discard_dynptr.md
bpf_ringbuf_output.md
bpf_ringbuf_query.md
bpf_ringbuf_reserve.md
bpf_ringbuf_reserve_dynptr.md
bpf_ringbuf_submit.md
bpf_ringbuf_submit_dynptr.md
bpf_send_signal.md
bpf_send_signal_thread.md
bpf_seq_printf.md

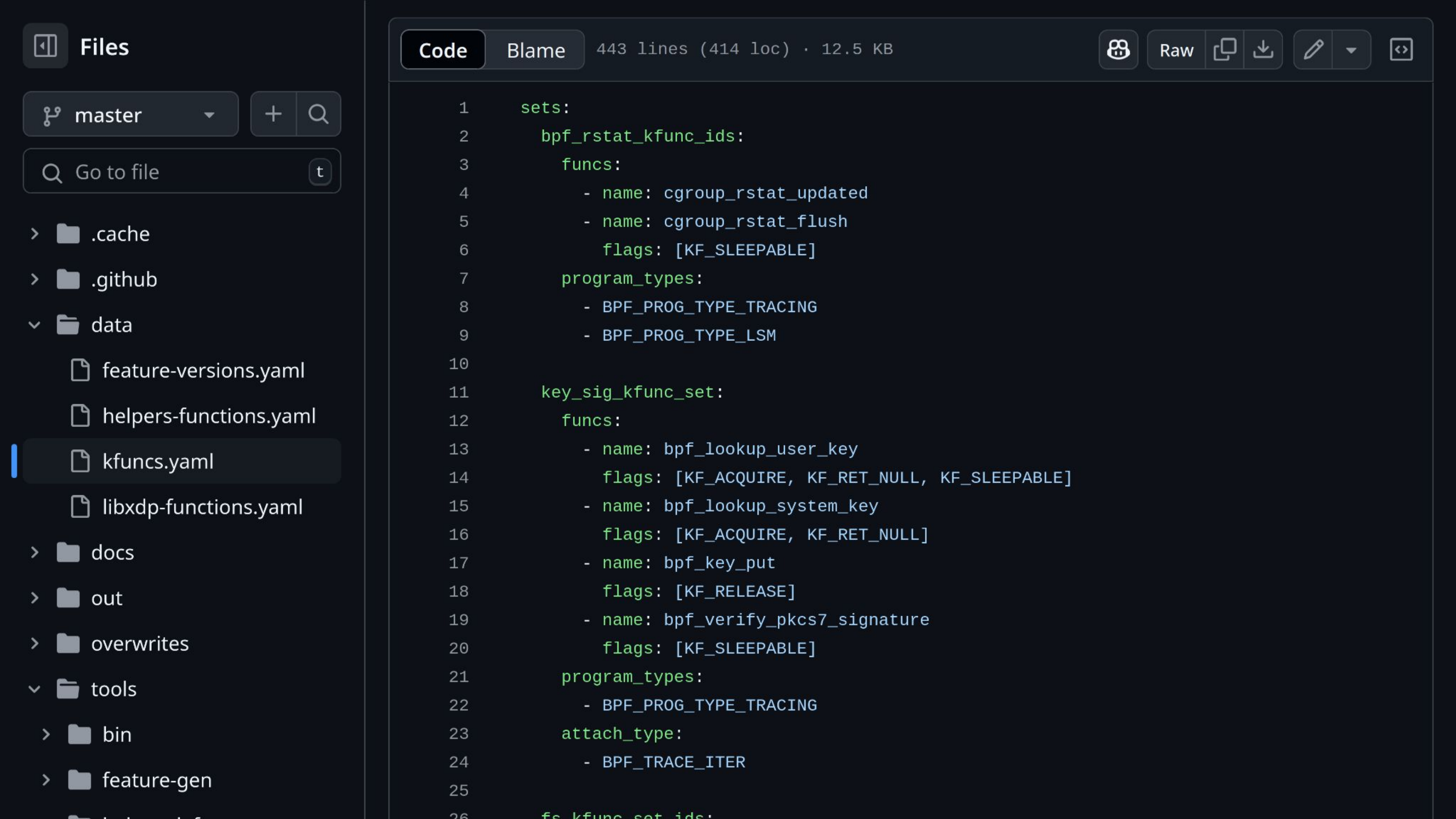
ebpf-docs / docs / linux / helper-function / bpf_ringbuf_output.md

77 lines (60 loc) · 4.56 KB

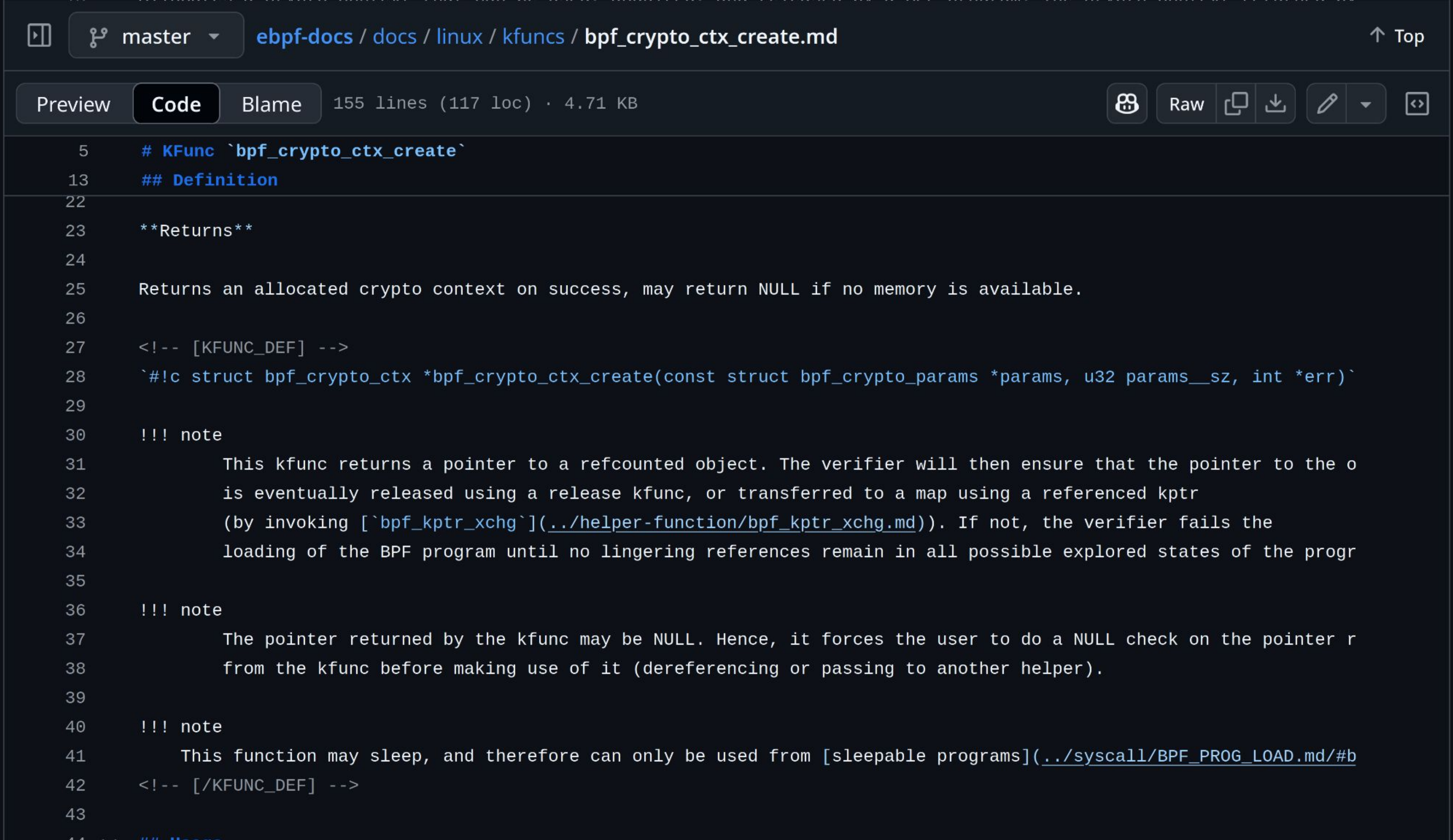
```
# Helper function `bpf_ringbuf_output`
## Definition

> Copyright (c) 2015 The Libbpf Authors. All rights reserved.


<!-- [HELPER_FUNC_DEF] -->
Copy _size_ bytes from _data_ into a ring buffer _ringbuf_. If **BPF_RB_NO_WAKEUP** is specified in _flags_, no notification of new dat

An adaptive notification is a notification sent whenever the user-space process has caught up and consumed all available payloads. In c

### Returns

0 on success, or a negative error in case of failure.

`#!c static long (* const bpf_ringbuf_output)(void *ringbuf, void *data, __u64 size, __u64 flags) = (void *) 130;`
<!-- [/HELPER_FUNC_DEF] -->

## Usage

The `ringbuf` must be a pointer to the ring buffer map. `data` is a pointer to the data that needs to be copied into the ring buffer. T

This function incurs an extra memory copy operation in comparison to using [`bpf_ringbuf_reserve`](./bpf_ringbuf_reserve.md)/[`bpf_ring

### Program types

This helper call can be used in the following program types:

<!-- DO NOT EDIT MANUALLY -->
<!-- [HELPER_FUNC_PROG_REF] -->
 * [`BPF_PROG_TYPE_CGROUP_DEVICE`](../program-type/BPF_PROG_TYPE_CGROUP_DEVICE.md)
 * [`BPF_PROG_TYPE_CGROUP_SKB`](../program-type/BPF_PROG_TYPE_CGROUP_SKB.md)
 * [`BPF_PROG_TYPE_CGROUP_SOCK`](../program-type/BPF_PROG_TYPE_CGROUP_SOCK.md)
 * [`BPF_PROG_TYPE_CGROUP_SOCKOPT`](../program-type/BPF_PROG_TYPE_CGROUP_SOCKOPT.md)
 * [`BPF_PROG_TYPE_CGROUP_SOCK_ADDR`](../program-type/BPF_PROG_TYPE_CGROUP_SOCK_ADDR.md)
```

Tools: kfunc-gen

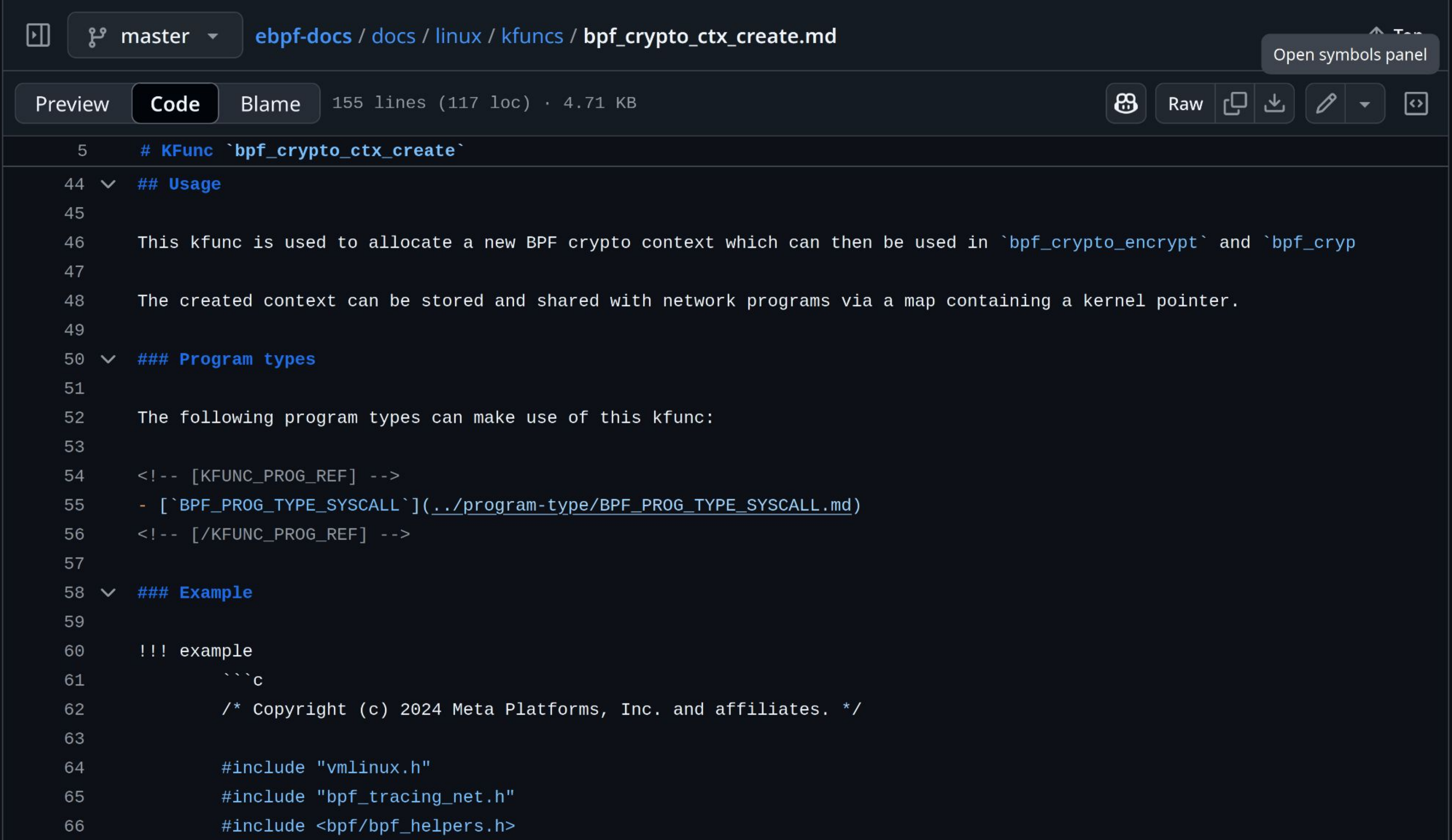
- Manually curate all kfunc sets (program types + KF_* flags)
- Compile a vmlinux with all kfuncs
- Use `bpf_kfunc` decl tag to validate data file
- Generate program <-> kfunc ref from data file
- Generate flag related warnings from data file
- Generate function signature from vmlinux BTF



```yaml
sets:
  bpf_rstat_kfunc_ids:
    funcs:
      - name: cgroup_rstat_updated
      - name: cgroup_rstat_flush
        flags: [KF_SLEEPABLE]
    program_types:
      - BPF_PROG_TYPE_TRACING
      - BPF_PROG_TYPE_LSM

  key_sig_kfunc_set:
    funcs:
      - name: bpf_lookup_user_key
        flags: [KF_ACQUIRE, KF_RET_NULL, KF_SLEEPABLE]
      - name: bpf_lookup_system_key
        flags: [KF_ACQUIRE, KF_RET_NULL]
      - name: bpf_key_put
        flags: [KF_RELEASE]
      - name: bpf_verify_pkcs7_signature
        flags: [KF_SLEEPABLE]
    program_types:
      - BPF_PROG_TYPE_TRACING
    attach_type:
      - BPF_TRACE_ITER

```


master
ebpf-docs / docs / linux / kfuncs / bpf_crypto_ctx_create.md
Top
Preview
Code
Blame
155 lines (117 loc) · 4.71 KB
Raw

```
# KFunc `bpf_crypto_ctx_create`
## Definition

**Returns**

Returns an allocated crypto context on success, may return NULL if no memory is available.

<!-- [KFUNC_DEF] -->
`#!c struct bpf_crypto_ctx *bpf_crypto_ctx_create(const struct bpf_crypto_params *params, u32 params__sz, int *err)`

!!! note
        This kfunc returns a pointer to a refcounted object. The verifier will then ensure that the pointer to the o
        is eventually released using a release kfunc, or transferred to a map using a referenced kptr
        (by invoking [`bpf_kptr_xchg`](../helper-function/bpf_kptr_xchg.md)). If not, the verifier fails the
        loading of the BPF program until no lingering references remain in all possible explored states of the progr

!!! note
        The pointer returned by the kfunc may be NULL. Hence, it forces the user to do a NULL check on the pointer r
        from the kfunc before making use of it (dereferencing or passing to another helper).

!!! note
    This function may sleep, and therefore can only be used from [sleepable programs](../syscall/BPF_PROG_LOAD.md/#b
<!-- [/KFUNC_DEF] -->
```


master
ebpf-docs / docs / linux / kfuncs / bpf_crypto_ctx_create.md
Open symbols panel
Preview
Code
Blame
155 lines (117 loc) · 4.71 KB
Raw

~~~markdown
# KFunc `bpf_crypto_ctx_create`
## Usage

This kfunc is used to allocate a new BPF crypto context which can then be used in `bpf_crypto_encrypt` and `bpf_cryp

The created context can be stored and shared with network programs via a map containing a kernel pointer.

### Program types

The following program types can make use of this kfunc:

<!-- [KFUNC_PROG_REF] -->
- [`BPF_PROG_TYPE_SYSCALL`](../program-type/BPF_PROG_TYPE_SYSCALL.md)
<!-- [/KFUNC_PROG_REF] -->

### Example

!!! example
        ```c
        /* Copyright (c) 2024 Meta Platforms, Inc. and affiliates. */

        #include "vmlinux.h"
        #include "bpf_tracing_net.h"
        #include <bpf/bpf_helpers.h>
~~~

Files

master

Go to file

- BPF_PROG_TYPE_SCHED_ACT....
- BPF_PROG_TYPE_SCHED_CLS....
- BPF_PROG_TYPE_SK_LOOKUP....
- BPF_PROG_TYPE_SK_MSG.md
- BPF_PROG_TYPE_SK_REUSEP...
- BPF_PROG_TYPE_SK_SKB.md
- BPF_PROG_TYPE_SOCKET_FIL...
- BPF_PROG_TYPE_SOCK_OPS....
- BPF_PROG_TYPE_STRUCT_OP...
- BPF_PROG_TYPE_SYSCALL.md
- BPF_PROG_TYPE_TRACEPOIN...
- BPF_PROG_TYPE_TRACING.md
- BPF_PROG_TYPE_XDP.md
- SUMMARY.md
- index.md
- syscall
- timeline
- index.md
- stylesheets
- SUMMARY.md

ebpf-docs / docs / linux / program-type / BPF_PROG_TYPE_XDP.md

Top

Preview Code Blame 500 lines (408 loc) · 59.3 KB

Raw

```
# Program type `BPF_PROG_TYPE_XDP`

## KFuncs

<!-- [PROG_KFUNC_REF] -->
??? abstract "Supported kfuncs"
    - [`bpf_arena_alloc_pages`](../kfuncs/bpf_arena_alloc_pages.md)
    - [`bpf_arena_free_pages`](../kfuncs/bpf_arena_free_pages.md)
    - [`bpf_cast_to_kern_ctx`](../kfuncs/bpf_cast_to_kern_ctx.md)
    - [`bpf_cgroup_acquire`](../kfuncs/bpf_cgroup_acquire.md)
    - [`bpf_cgroup_ancestor`](../kfuncs/bpf_cgroup_ancestor.md)
    - [`bpf_cgroup_from_id`](../kfuncs/bpf_cgroup_from_id.md)
    - [`bpf_cgroup_release`](../kfuncs/bpf_cgroup_release.md)
    - [`bpf_crypto_decrypt`](../kfuncs/bpf_crypto_decrypt.md)
    - [`bpf_crypto_encrypt`](../kfuncs/bpf_crypto_encrypt.md)
    - [`bpf_ct_change_status`](../kfuncs/bpf_ct_change_status.md)
    - [`bpf_ct_change_timeout`](../kfuncs/bpf_ct_change_timeout.md)
    - [`bpf_ct_insert_entry`](../kfuncs/bpf_ct_insert_entry.md)
    - [`bpf_ct_release`](../kfuncs/bpf_ct_release.md)
    - [`bpf_ct_set_nat_info`](../kfuncs/bpf_ct_set_nat_info.md)
    - [`bpf_ct_set_status`](../kfuncs/bpf_ct_set_status.md)
    - [`bpf_ct_set_timeout`](../kfuncs/bpf_ct_set_timeout.md)
    - [`bpf_dynptr_adjust`](../kfuncs/bpf_dynptr_adjust.md)
    - [`bpf_dynptr_clone`](../kfuncs/bpf_dynptr_clone.md)
    - [`bpf_dynptr_from_xdp`](../kfuncs/bpf_dynptr_from_xdp.md)
    - [`bpf_dynptr_is_null`](../kfuncs/bpf_dynptr_is_null.md)
    - [`bpf_dynptr_is_rdonly`](../kfuncs/bpf_dynptr_is_rdonly.md)
    - [`bpf_dynptr_size`](../kfuncs/bpf_dynptr_size.md)
    - [`bpf_dynptr_slice`](../kfuncs/bpf_dynptr_slice.md)
    - [`bpf_dynptr_slice_rdwr`](../kfuncs/bpf_dynptr_slice_rdwr.md)
    - [`bpf_iter_bits_destroy`](../kfuncs/bpf_iter_bits_destroy.md)
    - [`bpf_iter_bits_new`](../kfuncs/bpf_iter_bits_new.md)
    - [`bpf_iter_bits_next`](../kfuncs/bpf_iter_bits_next.md)
    - [`bpf_iter_css_destroy`](../kfuncs/bpf_iter_css_destroy.md)
    - [`bpf_iter_css_new`](../kfuncs/bpf_iter_css_new.md)
    - [`bpf_iter_css_next`](../kfuncs/bpf_iter_css_next.md)
```

Future work

- Keep up with changes
- Libbpf userspace
- More concept pages
- Tutorial style pages
- Fill the gaps
- Verifier logs
  - How to read them
  - Common errors and solutions
  - Collecting logs at https://github.com/parttimenerd/ebpf-verifier-errors