Kernel Extensions Large Language Model Agent

Yusheng Zheng, Yiwei Yang, Maolin Chen,
Andrew Quinn

IMPERIAL
UC SANTA CRUZ

eBPF '24 Workshop
August 4, 2024
SIGCOMM 2024
— SYDNEY —

eBPF May be hard to write

- For no kernel background junior developer
  - DevOps debugging the kernel network stack
  - Security engineer monitoring the system
  - Developer write network function in xdp or socket
- Similar For AI and LLM

Hallucinations

- Wrong API and helpers

2. Create a BPF program (process_monitor.c):

c 复制代码

```c
#include <uapi/linux/ptrace.h>
#include <linux/sched.h>

// Define a hash map to count forks per process
BPF_HASH(fork_counter, u32, u64);

int trace_fork(struct pt_regs *ctx) {
    u32 pid = bpf_get_current_pid_tgid() >> 32;
    u64 *counter, zero = 0;

    // Lookup or initialize the counter for the current process
    counter = fork_counter.lookup_or_init(&pid, &zero);
    (*counter)++;

    if (*counter >= 20) {
        // If the counter exceeds 20, kill the process
        bpf_trace_printk("Killing process %d\n", pid);
        bpf_kill_pid(pid);
    }

    return 0;
}
```

https://chatgpt.com/share/6f3b6aca-e1b3-427b-b18d-38d8bcdf85c1

Hallucinations

- Wrong attach point

```
tracepoint:xhci-hcd:xhci_stop_device
tracepoint:xhci-hcd:xhci_urb_dequeue
tracepoint:xhci-hcd:xhci_urb_enqueue
tracepoint:xhci-hcd:xhci_urb_giveback
root@yunwei-server:/home/yunwei37# bpftrace -l  | wc -l
119752
root@yunwei-server:/home/yunwei37#
```

Here's a simple example of how you can write a `bpftrace` script to detect new USB device attachments:

bash 复制代码

```
#!/usr/bin/env bpftrace

tracepoint:usb:usb_device_add
{
    printf("New USB device added: vendor=%x product=%x\n", args->vendor, args->product);
}
```

https://chatgpt.com/share/796109a6-c1ce-4309-af70-934dad15ecb5

Verifier limitation

```
// Check for HTTP GET request
if (http_payload[0] == 'G' && http_payload[1] == 'E' && http_payload[2] == 'T' && http_
    // Pointer to the start of the URL
    char *url_start = http_payload + 4;

    // Pointer to the end of the URL (space character after the URL)
    char *url_end = url_start;

    // Find the end of the URL
    while (url_end < (char *)data_end && *url_end != ' ') {
        url_end++;
    }
```

https://chatgpt.com/share/f7e8f41c-c191-41f5-ba28-0e7e2856d889

Incorrect behavio

- Successful pass the compiler and verifier
  - Not consider IPv6
  - Port Byte Order：

write a bpftrace program to trace tcp_connect events and displaying the source and destination IP addresses with their respective ports.

Here's a `bpftrace` program that traces `tcp_connect` events and displays the source and destination IP addresses along with their respective ports:

c 复制代码

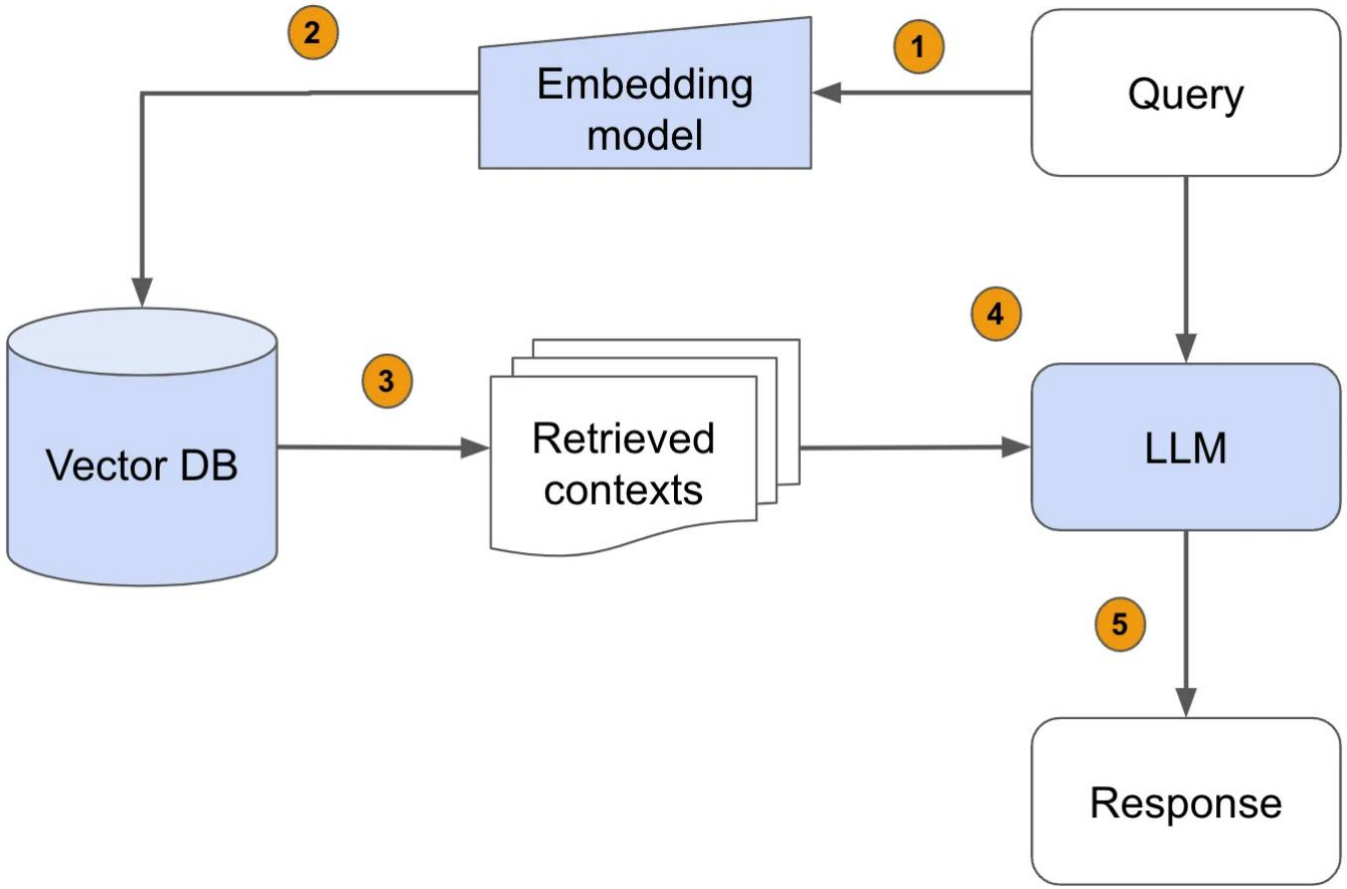
```
#!/usr/bin/env bpftrace

tracepoint:tcp:tcp_connect
{
    $sk = (struct sock *)args->sk;

    $saddr = ntop(((struct sockaddr_in *)(((struct inet_sock *)$sk)->inet_saddr))->sin_addr
    $daddr = ntop(((struct sockaddr_in *)(((struct inet_sock *)$sk)->inet_daddr))->sin_addr

    $sport = ((struct inet_sock *)$sk)->inet_sport;
    $dport = ((struct inet_sock *)$sk)->inet_dport;

    printf("TCP connect: src IP %s port %d -> dst IP %s port %d\n", $saddr, $sport, $daddr,
}
```

https://chatgpt.com/share/5c793353-3ab2-4907-8f82-ec4bccc62a1f

Retrieval-Augmented Generation (RAG)

fine—tuning or RAG?

- Data insufficient
- eBPF need kernel info when deployed



Agent Workflow

- Plans：
  - Workflow
  - Feedback
  - ReAct(Thought, Action, Observation)
- Tools：
  - clang
  - Seahorn
  - bpftrace
- Memory：
  - Short term in-context memory

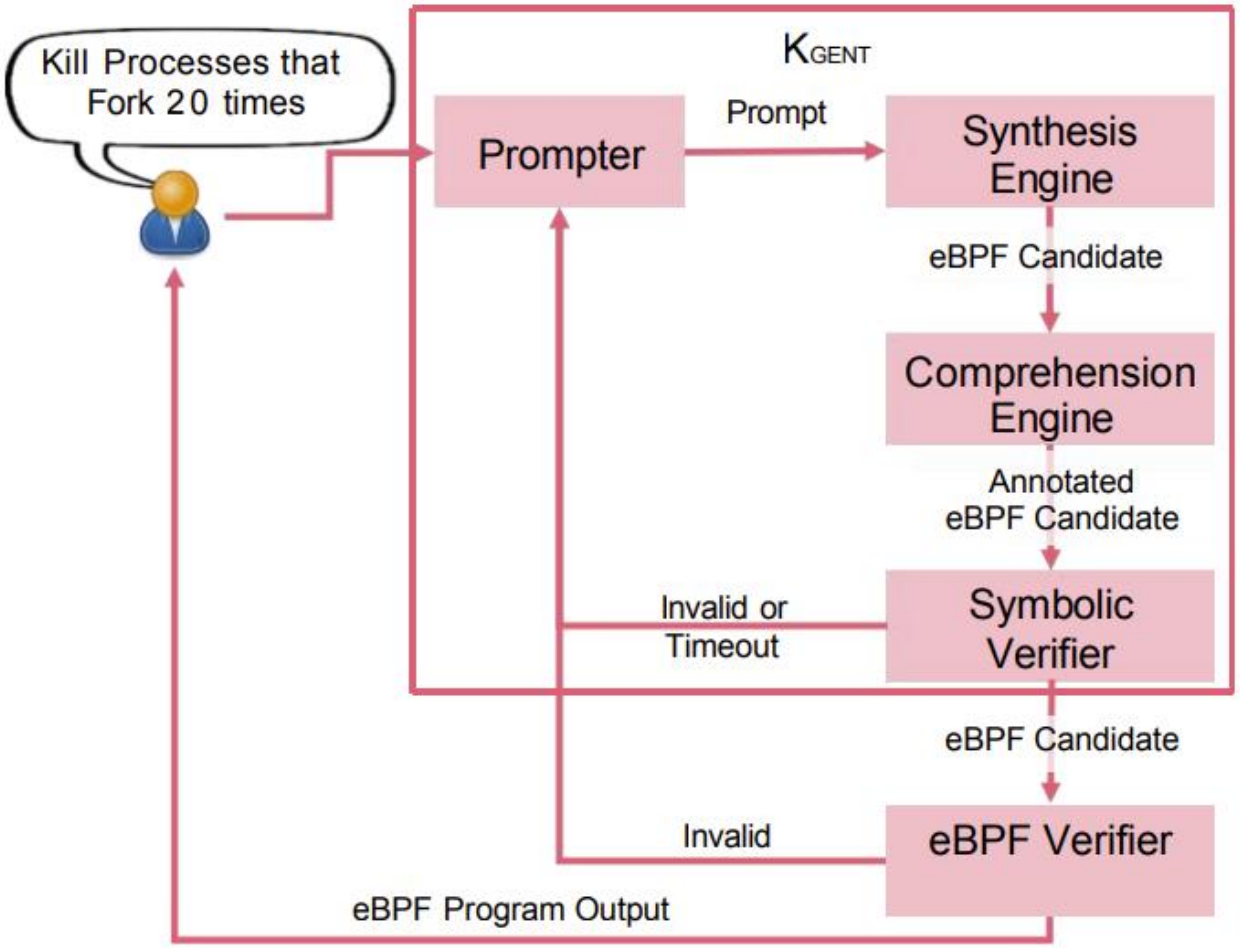


Figure 1: The Workflow of KGENT

Example :

User :

"Trace tcp_connect events for both IPv4 and IPv6 connection attempts, display the source and destination IP addresse."

Prompt template

In context learning

```
Write a bpftrace program that traces or profile the following user request:

### User Request

{user_request}

### Examples

Here are some simple examples to help you get started with bpftrace:

{examples}

Here are some complex examples may be related to your user request:

{complex_examples}

Use a tool provided to execute your bpftrace program.
You should only write the bpftrace program itself.
```

Vector DB

- Query：Trace tcp_connect events for both IPv4 and IPv6 connection attempts, display the source and destination IP addresse.
- Results：

````
Traces the TCP SYN backlog size and creates a histogram of the backlog sizes,
also indicating if any SYN packets are being dropped.

```
#ifndef BPFTRACE_HAVE_BTF
#include <net/sock.h>
#endif

BEGIN
{
   printf("Tracing SYN backlog size. Ctrl-C to end.\n");
}

kprobe:tcp_v4_syn_recv_sock,
kprobe:tcp_v6_syn_recv_sock
{
   $sock = (struct sock *)arg0;
   @backlog[$sock->sk_max_ack_backlog & 0xffffffff] =
       hist($sock->sk_ack_backlog);
   if ($sock->sk_ack_backlog > $sock->sk_max_ack_backlog) {
      time("%H:%M:%S dropping a SYN.\n");
   }
}

END
{
   printf("\n@backlog[backlog limit]: histogram of backlog size\n");
}

```
````

Vector DB: attach and helpers Spec

- Generated Spec with LLM from source code once
- Get related attach and helper Spec from vector DB

```c
/* Do all connect socket setups that can be done AF independent. */
static void tcp_connect_init(struct sock *sk)
{
        const struct dst_entry *dst = __sk_dst_get(sk);
        struct tcp_sock *tp = tcp_sk(sk);
        __u8 rcv_wscale;
        u32 rcv_wnd;

        /* We'll fix this up when we get a response from the other end.
         * See tcp_input.c:tcp_rcv_state_process case TCP_SYN_SENT.
         */
        tp->tcp_header_len = sizeof(struct tcphdr);
        if (READ_ONCE(sock_net(sk)->ipv4.sysctl_tcp_timestamps))
                tp->tcp_header_len += TCPOLEN_TSTAMP_ALIGNED;

        tcp_ao_connect_init(sk);
```

Kernel source code

GPT-4

```
{
    "kretprobe:tcp_connect_init": {
        "description": "initializes TCP connection parameters for a
        client socket before connection. It takes a sock structure
        pointer and adjusts the TCP header length, records
        user-defined MSS, and initializes PMTU and CA parameters. It
        selects and limits the receive window, sets sequence numbers
        and timestamps, clears socket errors, resets flags, purges the
        write queue, and sets retransmission timeouts. This modifies
        the sock and tcp_sock structures, ensuring the TCP connection
        setup is properly configured before sending SYN packets",
        "proto":"static void tcp_connect_init(struct sock *sk)",
        "pre": {
            "sk": "!= null;->__sk_common.skc_rcv_saddr != 0;
            ->__sk_common.skc_daddr != 0;->__sk_common.skc_num >=
            0->__sk_common.skc_dportr != 0;"
        }
    }
}
```

Spec and description

Symbolic Verifier

```
1  kprobe:tcp_connect {
2      $sk = (struct sock *) arg0;
3      assume($sk != 0);
4      assume($sk->___sk_common.skc_rcv_saddr != 0);
5      assume($sk->___sk_common.skc_daddr != 0);
6      assume($sk->___sk_common.skc_num >= 0);
7      assume($sk->___sk_common.skc_dport >= 0);
8      assume(sizeof($sk->___sk_common.skc_rcv_saddr) == 4
       ↪  || sizeof($sk->___sk_common.skc_rcv_saddr) == 16);
9      assume(sizeof($sk->___sk_common.skc_daddr) == 4 ||
       ↪  sizeof($sk->___sk_common.skc_daddr) == 16);
10     $saddr = ntop(2, $sk->___sk_common.skc_rcv_saddr);
11     $daddr = ntop(2, $sk->___sk_common.skc_daddr);
12     $sport = ($sk->___sk_common.skc_num);
13     $dport = ($sk->___sk_common.skc_dport);
14     printf("TCP connect: %s:%d -> %s:%d\n", $saddr, $sport,
       ↪  $daddr, $dport);
15     assert($dport ==
       ↪  bswap($sk->___sk_common.skc_dport));
16     assert($sport == bswap($sk->___sk_common.skc_num));
17 }
```

- Read from template
- Use LLM to generate Hoare constraints
- Verify by SeaHorn for semantic correctness
- Output feedback to LLM

Results

- For bpftrace:

System	A	FP	FN
GPT-4 few shot	30%	2.5%	67.5%
GPT-4+Feedback	60%	7.5%	32.5%
GPT-4+Feedback+Symbex	77.5%	5%	17.5%
Human	72.5%	2.5%	25%
KGENT	80%	2.5%	17.5%

Table 1: The Breakdown Accuracy Analysis of KGENT

- Codellama (Based on llama2)
  bpftrace: 40%
- GPT-4 libbpf feedback: 37.5%

Tested with gpt-4 API in 2023-09

Limitations and Future work

How to generate larger and better eBPF?

- Limitations:
  - Small programs and tasks: <100 lines
  - Context window limit from LLM
  - Dataset small
- Possible solution:
  - Split the task
  - Like AutoGPT
  - Ask user in iteration

Limitations and Future work

- Verification may not cover all behavior
  - More background and description
    - Generate better Hoare contract.
    - Use more Software Engineering efforts like counterexample generation.
  - Test driven development
- Security Vulnerabilities
  - Might contain security flaws
  - eBPF runs in the kernel

Thanks

Opensource repo：

- https://github.com/eunomia-bpf/GPTtrace
- https://github.com/eunomia-bpf/KEN