black hat USA 2022

eBPF ELFs JMPing Through the Windows

Richard Johnson

Trellix

Whoami

Richard Johnson

Senior Principal Security Researcher, Trellix
Vulnerability Research & Reverse Engineering

Owner, Fuzzing IO
Advanced Fuzzing and Crash Analysis Training

Contact
rjohnson@fuzzing.io
@richinseattle

Shout out to the Trellix Interns!

Kasimir Schulz	Andrea Fioraldi
@abraxus7331	@andreafioraldi

Outline

What is eBPF

eBPF is a virtual CPU architecture and VM aka “Berkley Packet Filter” extended to a more general purpose execution engine as an alternative to native kernel modules

eBPF programs are compiled from C into the virtual CPU instructions via LLVM and can run in emulated or JIT execution modes and includes a static verifier as part of the loader

Execution is sandboxed and highly restricted in what memory it can access and how many instructions each eBPF program may contain

eBPF is designed for high speed inspection and modification of network packets and program execution

Origins of eBPF

Berkeley Packet Filter technology was developed in 1992 as a way to filter network packets

BPF was reimplemented for most Unix style operating systems and also ported to userland

Most users have interacted with BPF via tcpdump, wireshark, winpcap, or npcap

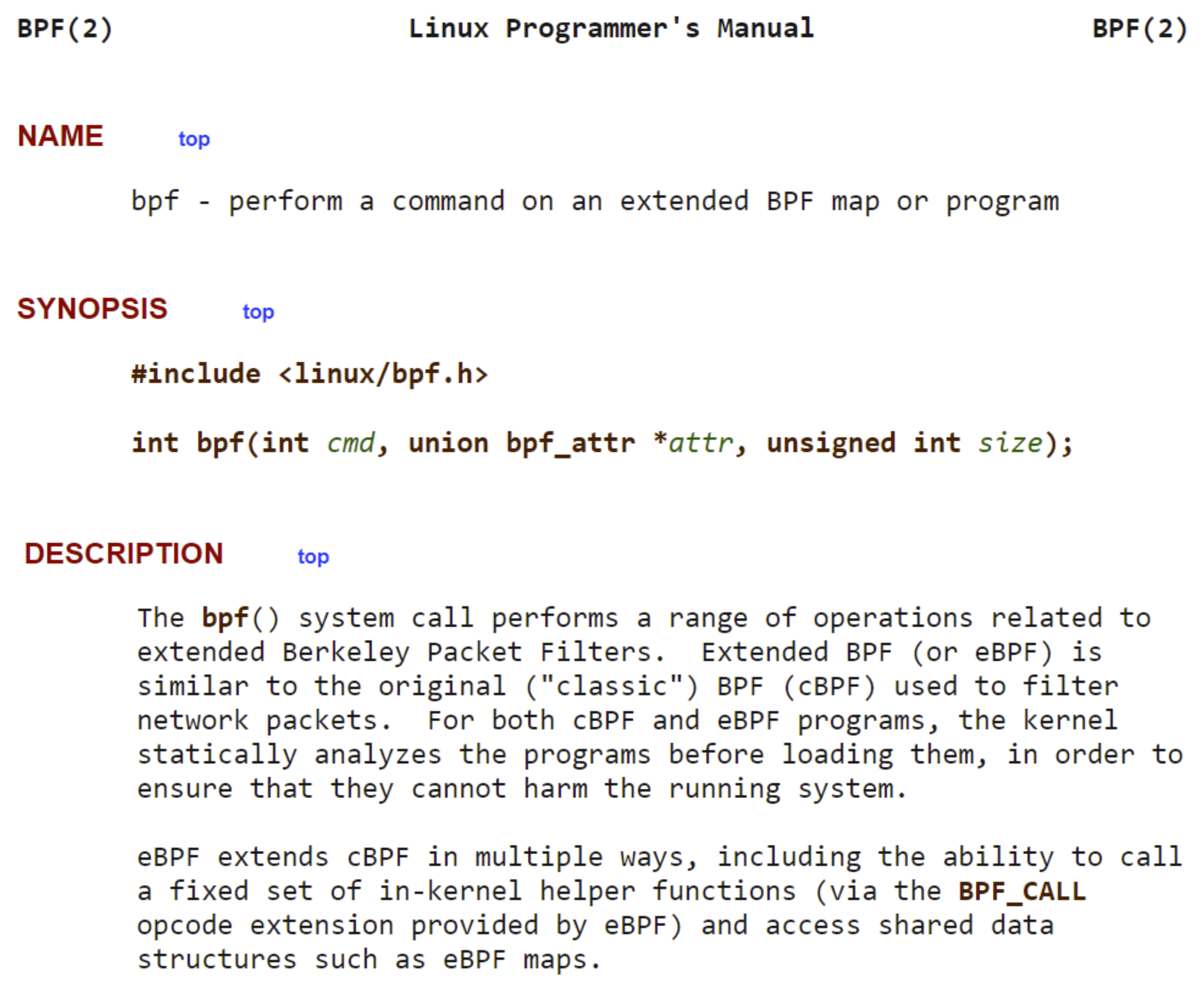
Using tcpdump and supplying a filter string like “dst host 10.10.10.10 and (tcp port 80 or tcp port 443)” automatically compiles into a BPF filter for high performance.

We now call this older BPF interface cBPF or Classic BPF

Origins of eBPF

In December 2014, Linux kernel 3.18 was released with the addition of the bpf() system call which implements the eBPF API

eBPF extends BPF instructions to 64bit and adds the concept of BPF Maps which are arrays of persistent data structures that can be shared between eBPF programs and userspace daemons



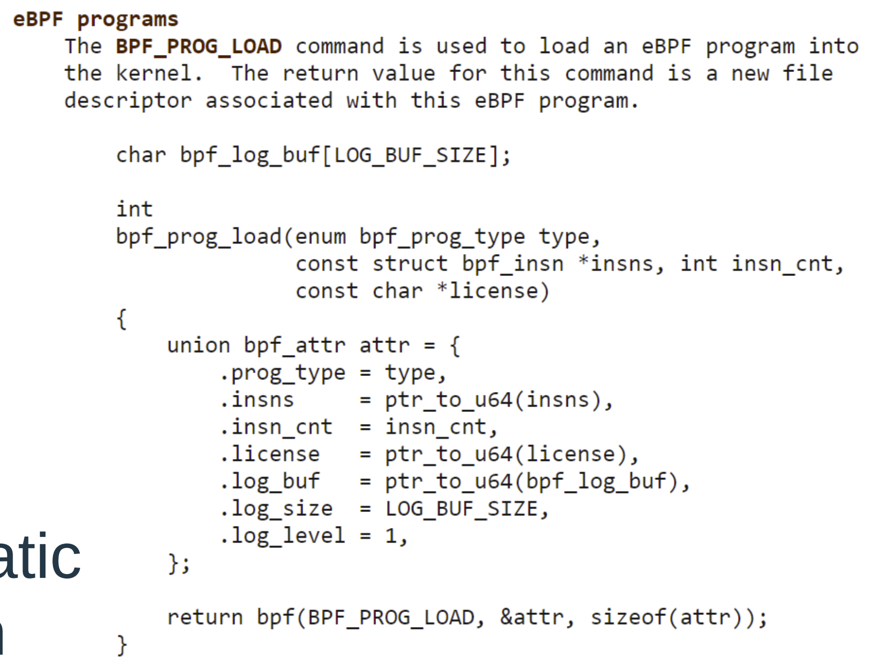
```
BPF(2)                  Linux Programmer's Manual                  BPF(2)

NAME       top

       bpf - perform a command on an extended BPF map or program

SYNOPSIS       top

       #include <linux/bpf.h>

       int bpf(int cmd, union bpf_attr *attr, unsigned int size);

DESCRIPTION       top

       The bpf() system call performs a range of operations related to
       extended Berkeley Packet Filters.  Extended BPF (or eBPF) is
       similar to the original ("classic") BPF (cBPF) used to filter
       network packets.  For both cBPF and eBPF programs, the kernel
       statically analyzes the programs before loading them, in order to
       ensure that they cannot harm the running system.

       eBPF extends cBPF in multiple ways, including the ability to call
       a fixed set of in-kernel helper functions (via the BPF_CALL
       opcode extension provided by eBPF) and access shared data
       structures such as eBPF maps.
```

Origins of eBPF

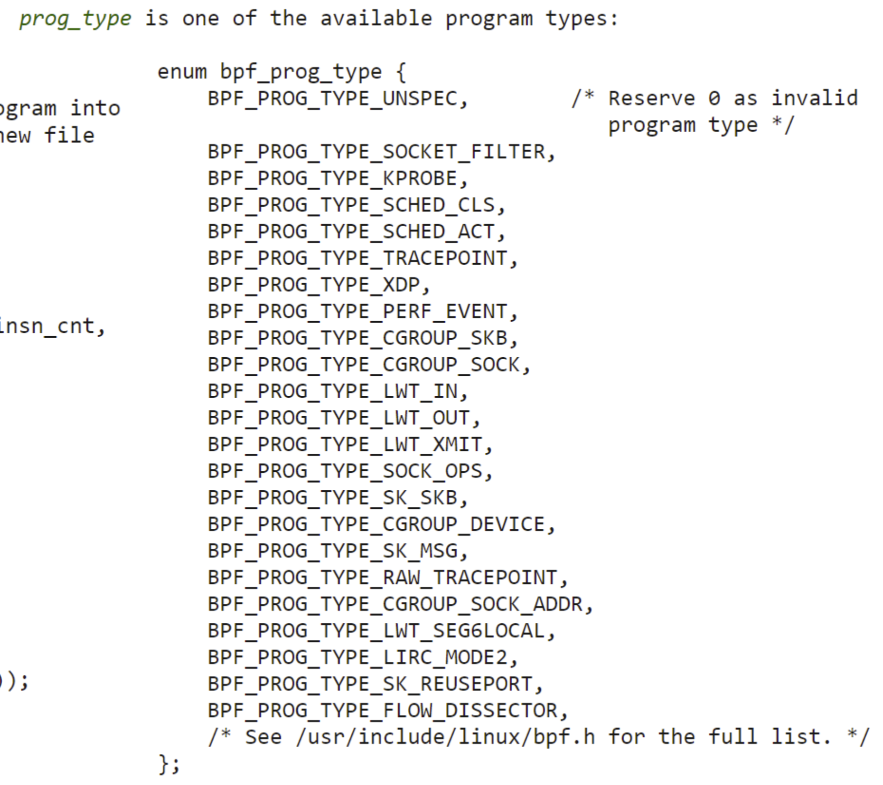
eBPF extended the original BPF concept to allow users to write general purpose programs and call out to kernel provided APIs

Each eBPF program is a single function, but they may tail call into others

All eBPF programs must pass a static verifier that ensures safe execution within the VM

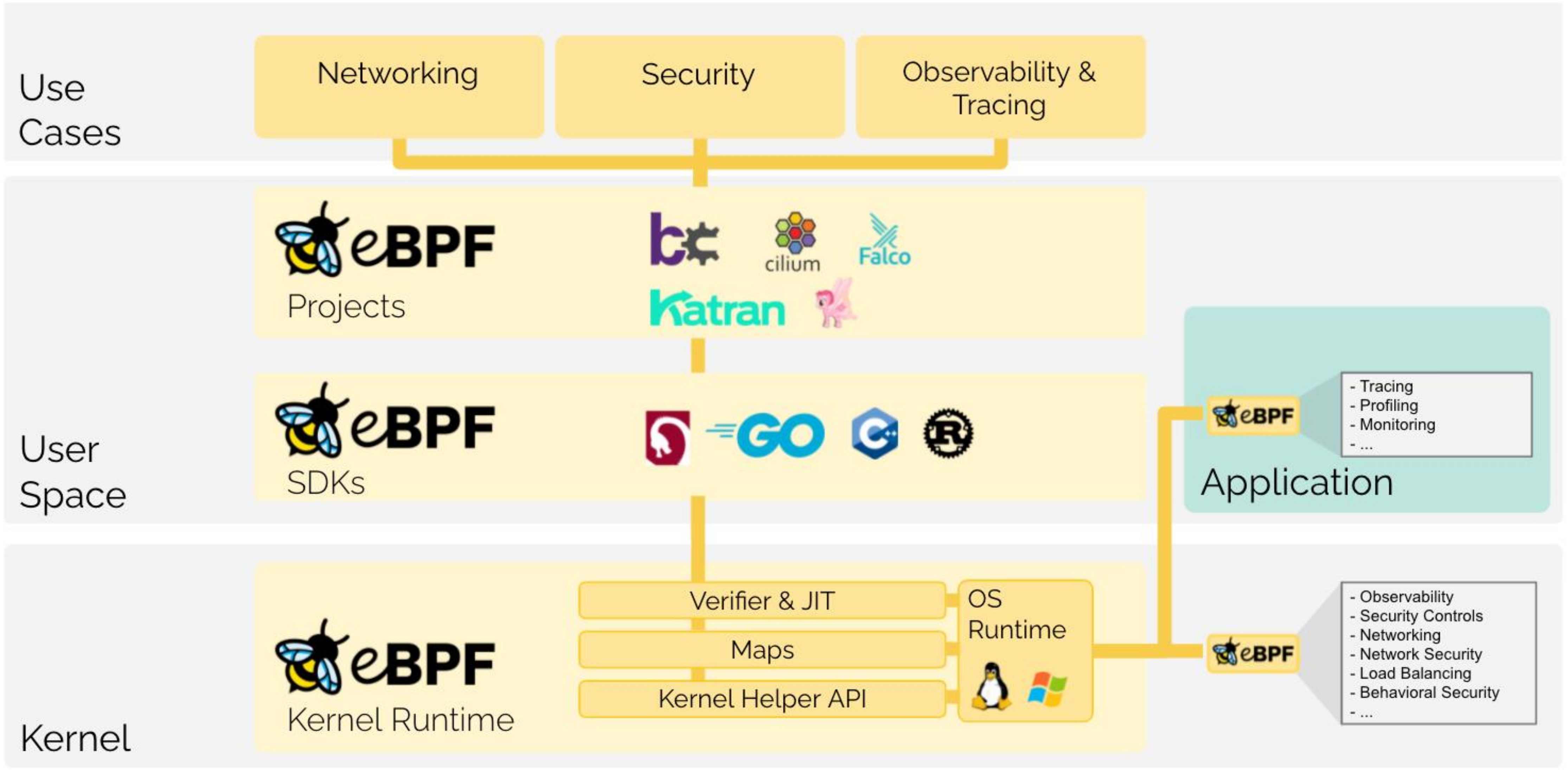


```
eBPF programs
    The BPF_PROG_LOAD command is used to load an eBPF program into
    the kernel.  The return value for this command is a new file
    descriptor associated with this eBPF program.

        char bpf_log_buf[LOG_BUF_SIZE];

        int
        bpf_prog_load(enum bpf_prog_type type,
                      const struct bpf_insn *insns, int insn_cnt,
                      const char *license)
        {
            union bpf_attr attr = {
                .prog_type = type,
                .insns     = ptr_to_u64(insns),
                .insn_cnt  = insn_cnt,
                .license   = ptr_to_u64(license),
                .log_buf   = ptr_to_u64(bpf_log_buf),
                .log_size  = LOG_BUF_SIZE,
                .log_level = 1,
            };

            return bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
        }
```



```
prog_type is one of the available program types:

    enum bpf_prog_type {
        BPF_PROG_TYPE_UNSPEC,        /* Reserve 0 as invalid
                                        program type */
        BPF_PROG_TYPE_SOCKET_FILTER,
        BPF_PROG_TYPE_KPROBE,
        BPF_PROG_TYPE_SCHED_CLS,
        BPF_PROG_TYPE_SCHED_ACT,
        BPF_PROG_TYPE_TRACEPOINT,
        BPF_PROG_TYPE_XDP,
        BPF_PROG_TYPE_PERF_EVENT,
        BPF_PROG_TYPE_CGROUP_SKB,
        BPF_PROG_TYPE_CGROUP_SOCK,
        BPF_PROG_TYPE_LWT_IN,
        BPF_PROG_TYPE_LWT_OUT,
        BPF_PROG_TYPE_LWT_XMIT,
        BPF_PROG_TYPE_SOCK_OPS,
        BPF_PROG_TYPE_SK_SKB,
        BPF_PROG_TYPE_CGROUP_DEVICE,
        BPF_PROG_TYPE_SK_MSG,
        BPF_PROG_TYPE_RAW_TRACEPOINT,
        BPF_PROG_TYPE_CGROUP_SOCK_ADDR,
        BPF_PROG_TYPE_LWT_SEG6LOCAL,
        BPF_PROG_TYPE_LIRC_MODE2,
        BPF_PROG_TYPE_SK_REUSEPORT,
        BPF_PROG_TYPE_FLOW_DISSECTOR,
        /* See /usr/include/linux/bpf.h for the full list. */
    };
```

Applications of eBPF



Linux eBPF Applications

Microsoft Defender for Endpoint

Falco

cilium

ProcMon-for-Linux

vmware Carbon Black EDR

red canary

aqua tracee

More projects on https://ebpf.io/projects

Prior eBPF Research

Evil eBPF – Jeff Dileo, DEF CON 27 (2019)
- Use of BPF_MAPS as IPC
- Discussed the unprivileged interface BPF_PROG_TYPE_SOCKET_FILTER
- Outlined a technique for ROP chain injection

With Friends like eBPF, who needs enemies – Guillaume Fournier, et al, BH USA 2021
- eBPF Rootkit demonstrations hooking syscall returns and userspace APIs
- Exfiltration over replaced HTTPS request packets

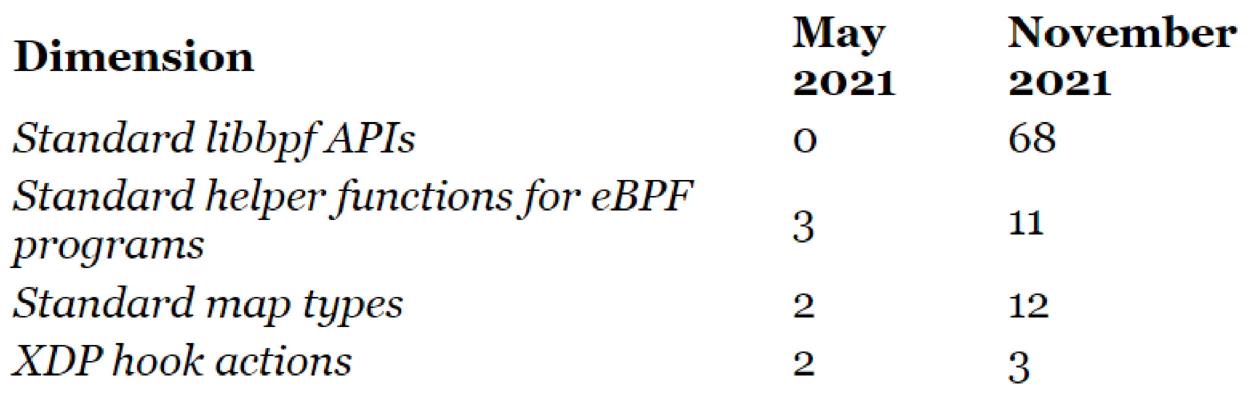
Extra Better Program Finagling (eBPF) – Richard Johnson, Toorcon 2021
- Showed hooks on Linux for tracing intercepting process creation
- Preempt loading libc with attacker controlled library (undebuggable from userland)
- Hook all running processes
- Provide a method for pivoting hooks into systemd-init
- Fuzzed and previewed crashes in ubpf and PREVAIL verifier

eBPF for Windows Timeline

eBPF for Windows was announced in May 2021 https://cloudblogs.microsoft.com/opensource/2021/05/10/making-ebpf-work-on-windows/
“So far, two hooks (XDP and socket bind) have been added, and though these are networking-specific hooks, we expect many more hooks and helpers, not just networking-related, will be added over time.”

August 2021 Microsoft, Netflix, Google, Facebook, and Isovalent announce the eBPF Foundation as part of the Linux Foundation

November 2021 added libbpf compatibility and additional BPF_MAPS support



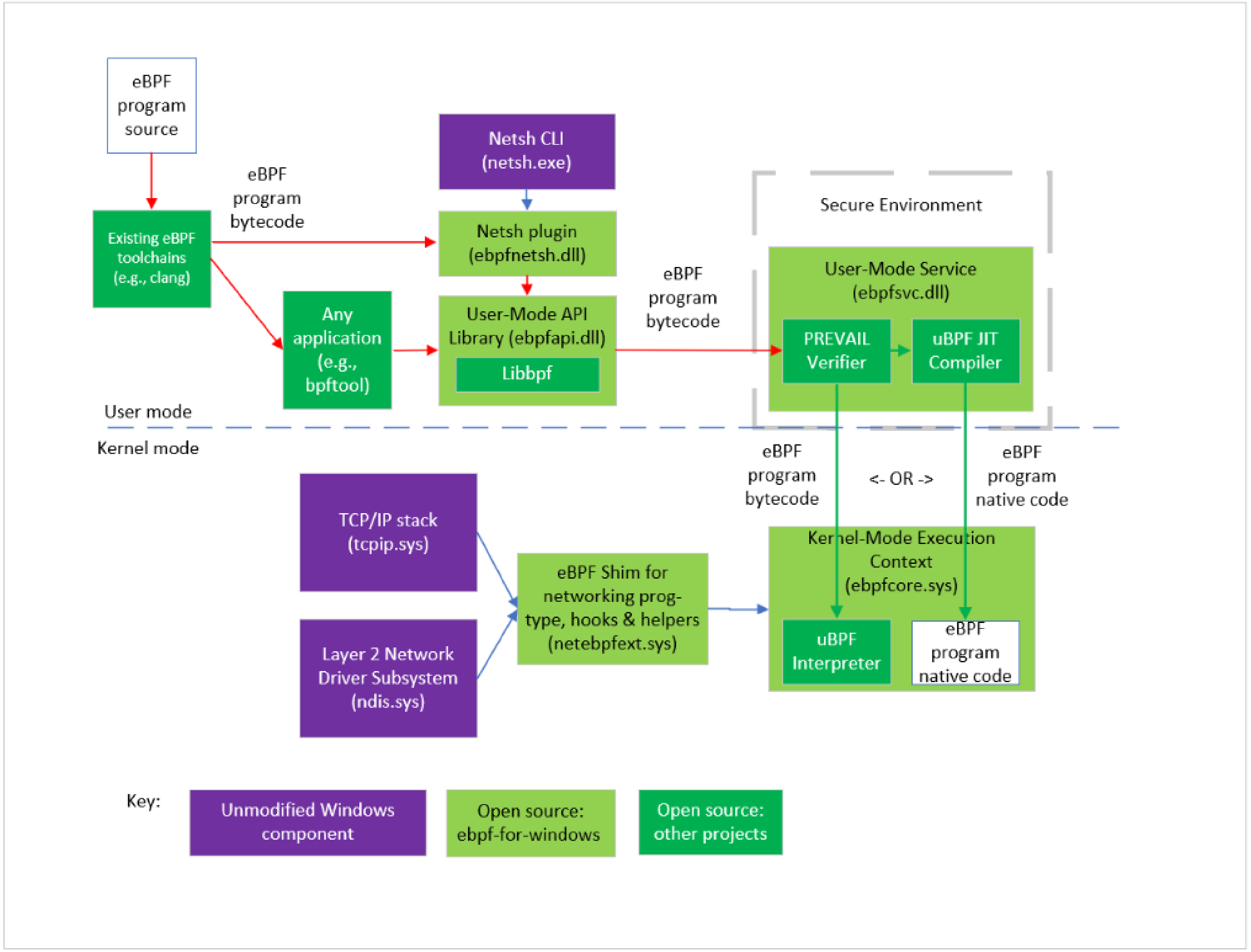
Dimension	May 2021	November 2021
Standard libbpf APIs	0	68
Standard helper functions for eBPF programs	3	11
Standard map types	2	12
XDP hook actions	2	3

https://cloudblogs.microsoft.com/opensource/2021/11/29/progress-on-making-ebpf-work-on-windows/

February 2022 Microsoft released a blog discussing efforts to port Cillium L4LB load balancer from Linux to Windows https://cloudblogs.microsoft.com/opensource/2022/02/22/getting-linux-based-ebpf-programs-to-run-with-ebpf-for-windows/

eBPF for Windows Architecture

Unlike the Linux eBPF system which is entirely contained in the kernel and used via system calls, the Windows version splits the system into several components and imports several opensource projects including the IO Visor uBPF VM and the PREVAIL static verifier*



eBPF program source
Existing eBPF toolchains (e.g., clang)
eBPF program bytecode
Netsh CLI (netsh.exe)
Netsh plugin (ebpfnetsh.dll)
Any application (e.g., bpftool)
User-Mode API Library (ebpfapi.dll)
Libbpf
eBPF program bytecode
Secure Environment
User-Mode Service (ebpfsvc.dll)
PREVAIL Verifier
uBPF JIT Compiler
User mode
Kernel mode
eBPF program bytecode
<- OR ->
eBPF program native code
TCP/IP stack (tcpip.sys)
Layer 2 Network Driver Subsystem (ndis.sys)
eBPF Shim for networking prog-type, hooks & helpers (netebpfext.sys)
Kernel-Mode Execution Context (ebpfcore.sys)
uBPF Interpreter
eBPF program native code
Key:
Unmodified Windows component
Open source: ebpf-for-windows
Open source: other projects

eBPF for Windows

eBPF for Windows is currently capable of performing introspection and modification of network packets and exposes a libbpf api compatibility layer for portability

eBPF for Windows is shipped as a standalone component with claims that is for easier serviceability

eBPF for Windows is MIT Licensed and may be shipped as a component of third party applications which may extend any of the layers

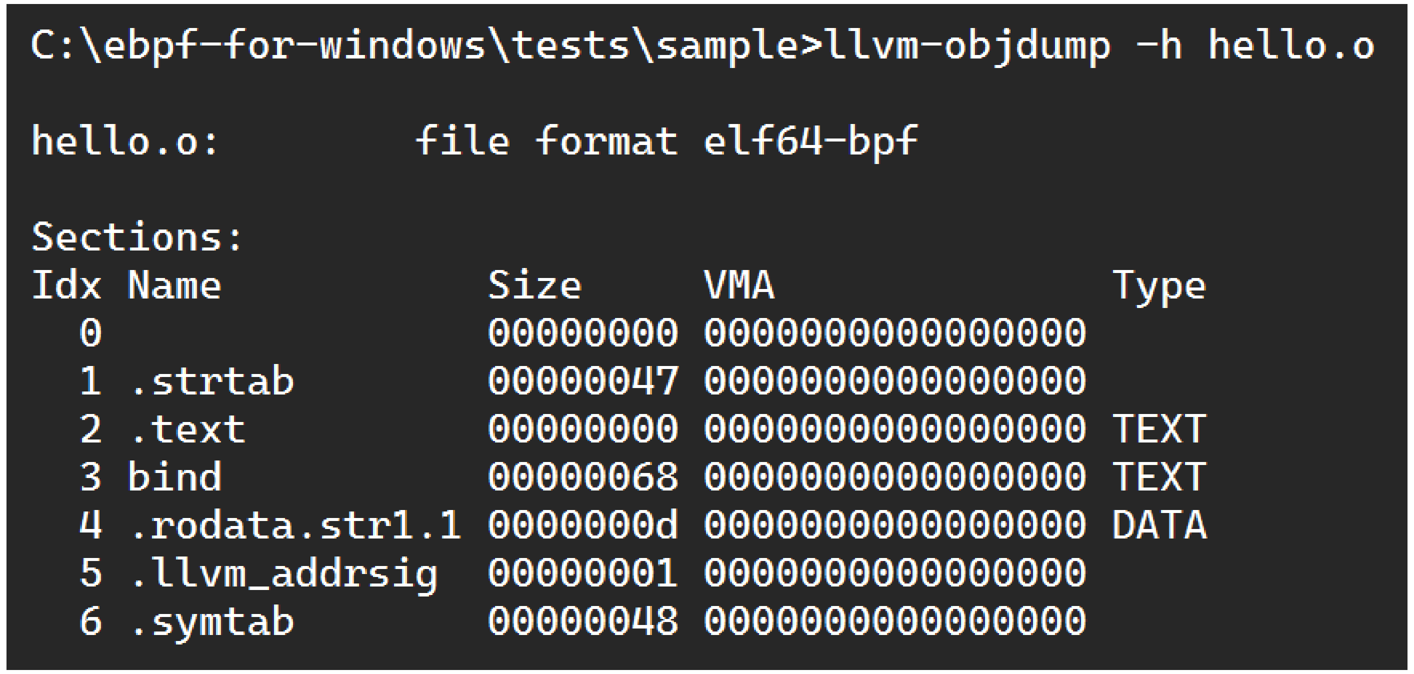
Creating eBPF Programs on Windows

On Windows, eBPF programs can be compiled from C source using LLVM

```
#include "bpf_helpers.h"

SEC("bind")
int hello(void *ctx) {
    bpf_printk("Hello world\n");
    return 0;
}

C:\ebpf-for-windows\tests\sample>clang -target bpf -O2 -Werror -c hello.c \
-I..\..\include -I..\..\external\bpftool
```

Creating eBPF Programs on Windows

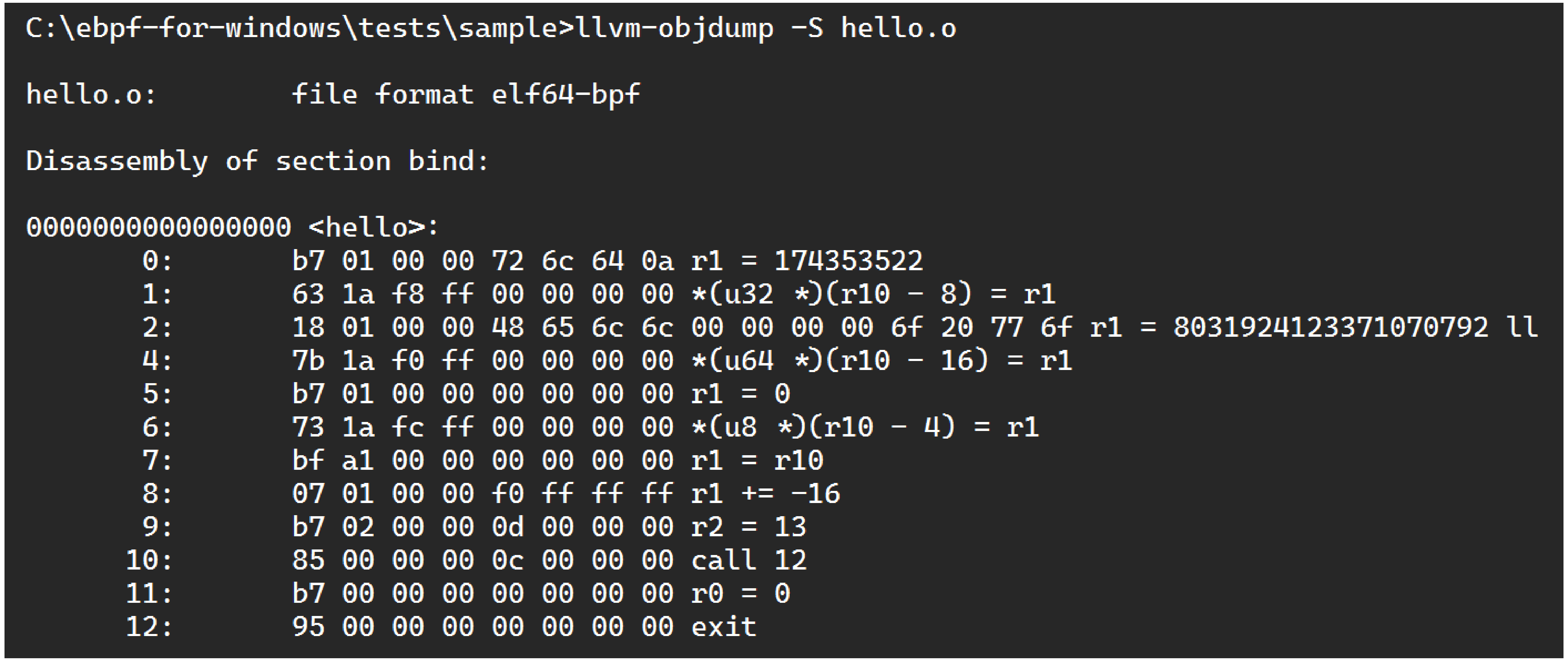
The resulting output is an ELF object with eBPF bytecode stored in ELF sections



```
C:\ebpf-for-windows\tests\sample>llvm-objdump -h hello.o

hello.o:        file format elf64-bpf

Sections:
Idx Name            Size     VMA              Type
  0                 00000000 0000000000000000
  1 .strtab         00000047 0000000000000000
  2 .text           00000000 0000000000000000 TEXT
  3 bind            00000068 0000000000000000 TEXT
  4 .rodata.str1.1  0000000d 0000000000000000 DATA
  5 .llvm_addrsig   00000001 0000000000000000
  6 .symtab         00000048 0000000000000000
```

Creating eBPF Programs on Windows

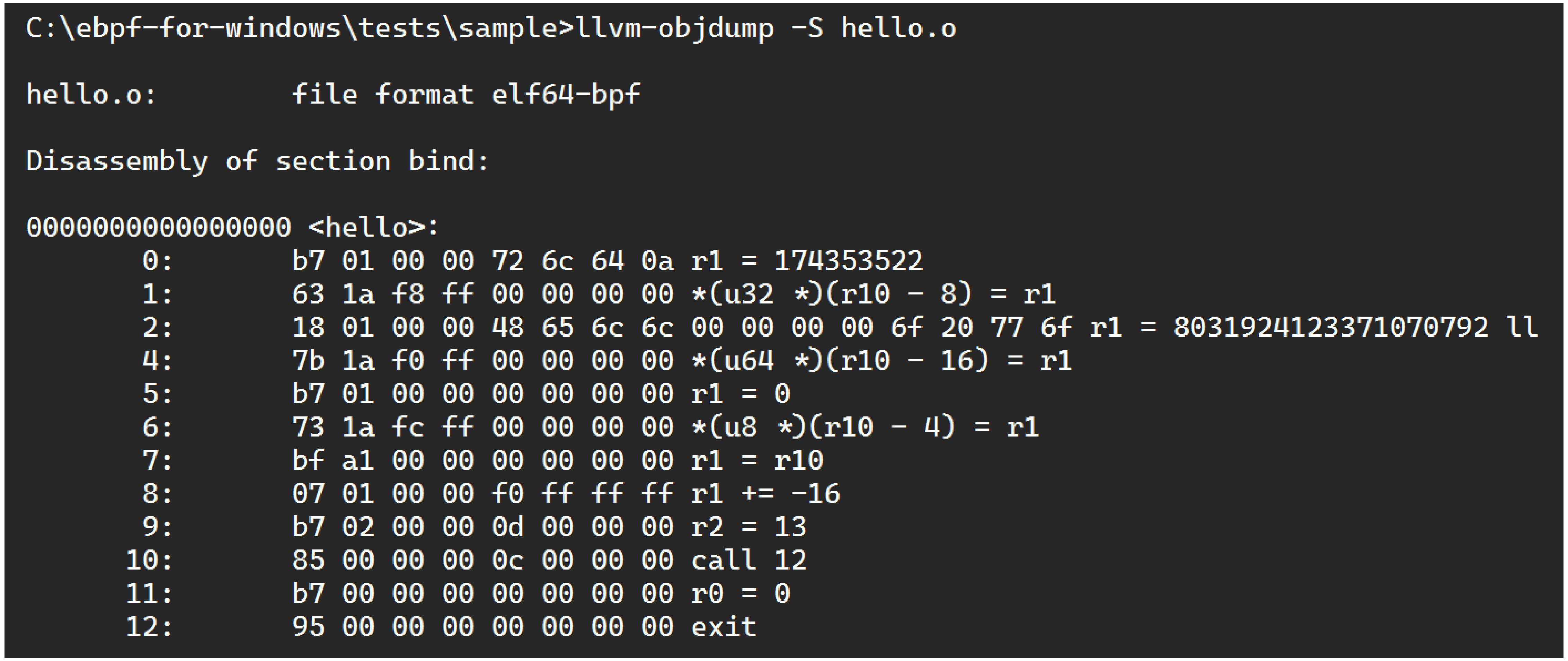
The resulting output is an ELF object with eBPF bytecode stored in ELF sections



```
C:\ebpf-for-windows\tests\sample>llvm-objdump -S hello.o

hello.o:        file format elf64-bpf

Disassembly of section bind:

0000000000000000 <hello>:
       0:       b7 01 00 00 72 6c 64 0a r1 = 174353522
       1:       63 1a f8 ff 00 00 00 00 *(u32 *)(r10 - 8) = r1
       2:       18 01 00 00 48 65 6c 6c 00 00 00 00 6f 20 77 6f r1 = 8031924123371070792 ll
       4:       7b 1a f0 ff 00 00 00 00 *(u64 *)(r10 - 16) = r1
       5:       b7 01 00 00 00 00 00 00 r1 = 0
       6:       73 1a fc ff 00 00 00 00 *(u8 *)(r10 - 4) = r1
       7:       bf a1 00 00 00 00 00 00 r1 = r10
       8:       07 01 00 00 f0 ff ff ff r1 += -16
       9:       b7 02 00 00 0d 00 00 00 r2 = 13
      10:       85 00 00 00 0c 00 00 00 call 12
      11:       b7 00 00 00 00 00 00 00 r0 = 0
      12:       95 00 00 00 00 00 00 00 exit
```

Creating eBPF Programs on Windows

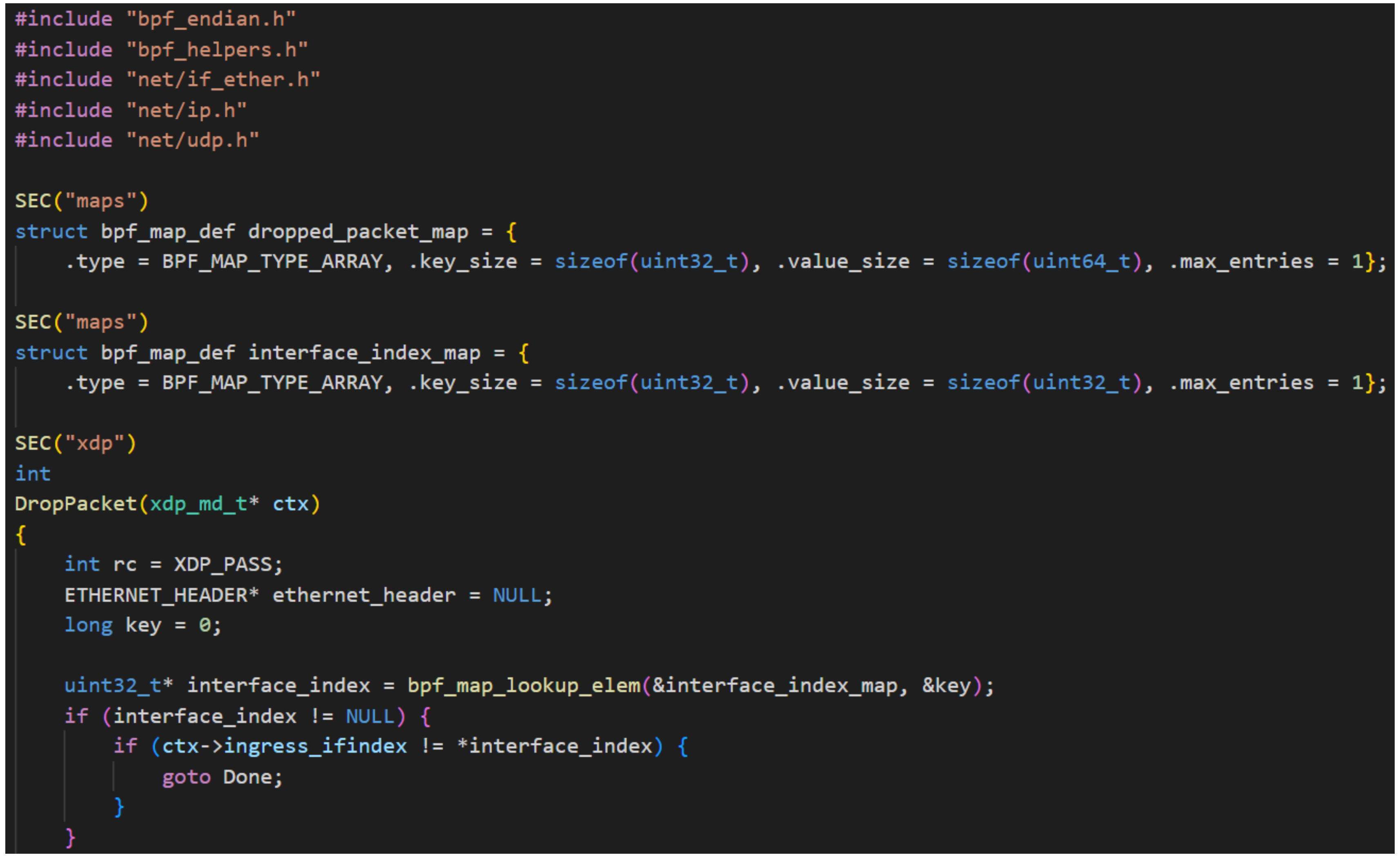
The resulting output is an ELF object with eBPF bytecode stored in ELF sections



```
C:\ebpf-for-windows\tests\sample>llvm-objdump -S hello.o

hello.o:        file format elf64-bpf

Disassembly of section bind:

0000000000000000 <hello>:
       0:       b7 01 00 00 72 6c 64 0a r1 = 174353522
       1:       63 1a f8 ff 00 00 00 00 *(u32 *)(r10 - 8) = r1
       2:       18 01 00 00 48 65 6c 6c 00 00 00 00 6f 20 77 6f r1 = 8031924123371070792 ll
       4:       7b 1a f0 ff 00 00 00 00 *(u64 *)(r10 - 16) = r1
       5:       b7 01 00 00 00 00 00 00 r1 = 0
       6:       73 1a fc ff 00 00 00 00 *(u8 *)(r10 - 4) = r1
       7:       bf a1 00 00 00 00 00 00 r1 = r10
       8:       07 01 00 00 f0 ff ff ff r1 += -16
       9:       b7 02 00 00 0d 00 00 00 r2 = 13
      10:       85 00 00 00 0c 00 00 00 call 12
      11:       b7 00 00 00 00 00 00 00 r0 = 0
      12:       95 00 00 00 00 00 00 00 exit
```

Creating eBPF Programs on Windows

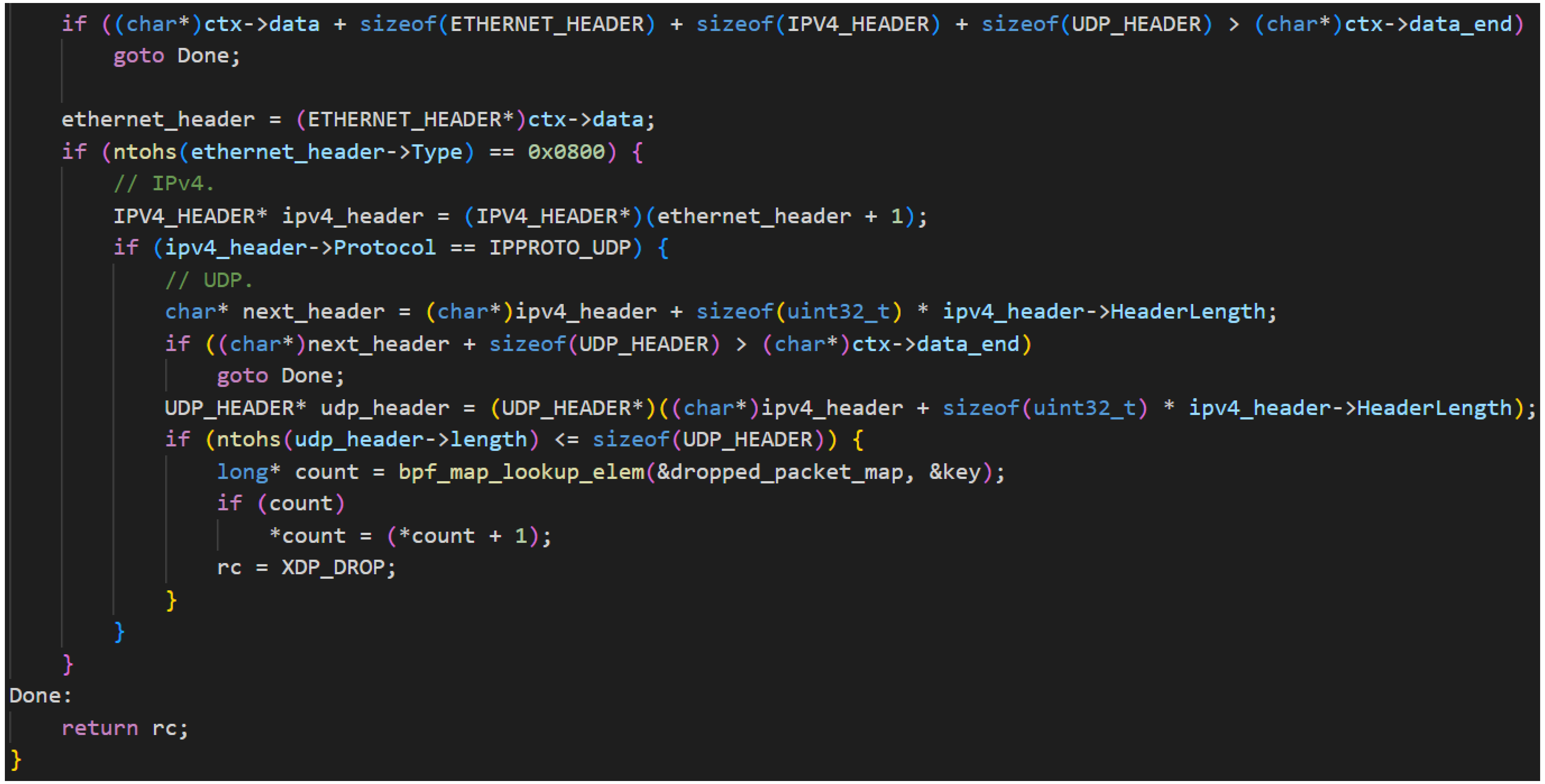
Here's an example of a more practical eBPF program for dropping certain packets



```c
#include "bpf_endian.h"
#include "bpf_helpers.h"
#include "net/if_ether.h"
#include "net/ip.h"
#include "net/udp.h"

SEC("maps")
struct bpf_map_def dropped_packet_map = {
    .type = BPF_MAP_TYPE_ARRAY, .key_size = sizeof(uint32_t), .value_size = sizeof(uint64_t), .max_entries = 1};

SEC("maps")
struct bpf_map_def interface_index_map = {
    .type = BPF_MAP_TYPE_ARRAY, .key_size = sizeof(uint32_t), .value_size = sizeof(uint32_t), .max_entries = 1};

SEC("xdp")
int
DropPacket(xdp_md_t* ctx)
{
    int rc = XDP_PASS;
    ETHERNET_HEADER* ethernet_header = NULL;
    long key = 0;

    uint32_t* interface_index = bpf_map_lookup_elem(&interface_index_map, &key);
    if (interface_index != NULL) {
        if (ctx->ingress_ifindex != *interface_index) {
            goto Done;
        }
    }
```

Creating eBPF Programs on Windows

Here’s an example of a more practical eBPF program for dropping certain packets



```
    if ((char*)ctx->data + sizeof(ETHERNET_HEADER) + sizeof(IPV4_HEADER) + sizeof(UDP_HEADER) > (char*)ctx->data_end)
        goto Done;

    ethernet_header = (ETHERNET_HEADER*)ctx->data;
    if (ntohs(ethernet_header->Type) == 0x0800) {
        // IPv4.
        IPV4_HEADER* ipv4_header = (IPV4_HEADER*)(ethernet_header + 1);
        if (ipv4_header->Protocol == IPPROTO_UDP) {
            // UDP.
            char* next_header = (char*)ipv4_header + sizeof(uint32_t) * ipv4_header->HeaderLength;
            if ((char*)next_header + sizeof(UDP_HEADER) > (char*)ctx->data_end)
                goto Done;
            UDP_HEADER* udp_header = (UDP_HEADER*)((char*)ipv4_header + sizeof(uint32_t) * ipv4_header->HeaderLength);
            if (ntohs(udp_header->length) <= sizeof(UDP_HEADER)) {
                long* count = bpf_map_lookup_elem(&dropped_packet_map, &key);
                if (count)
                    *count = (*count + 1);
                rc = XDP_DROP;
            }
        }
    }
Done:
    return rc;
}
```

eBPF for Windows Program Types

BPF_PROG_TYPE_XDP
"Program type for handling incoming packets as early as possible.
Attach type(s): BPF_XDP"

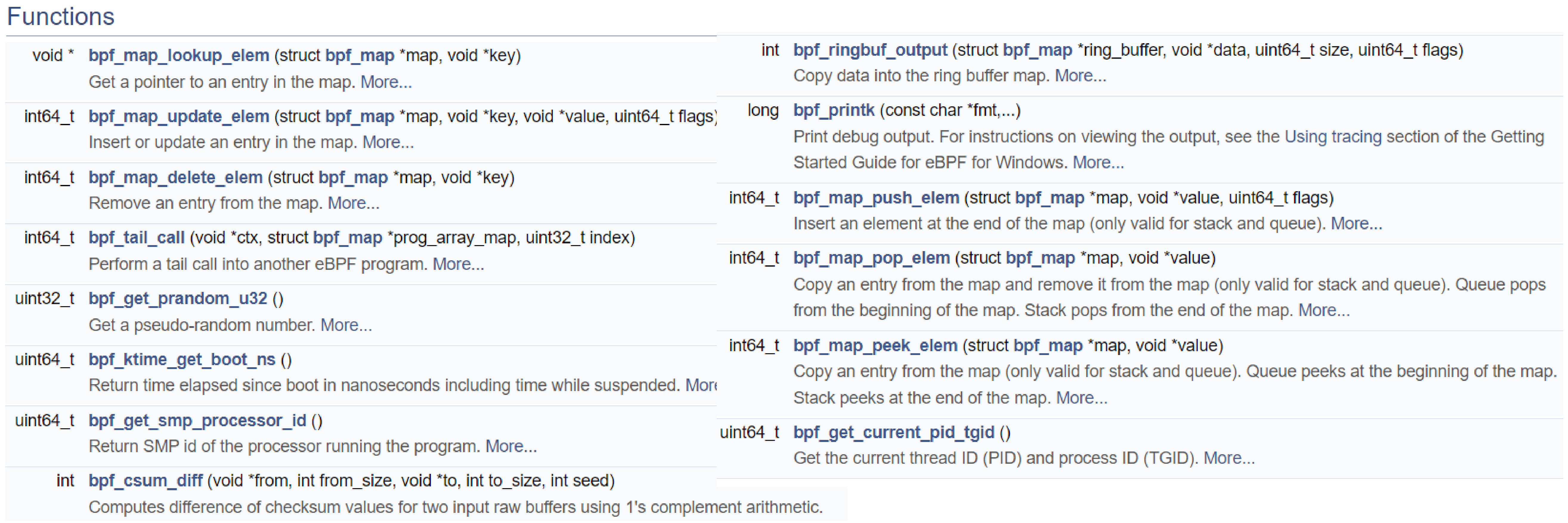
BPF_PROG_TYPE_BIND
"Program type for handling socket bind() requests.
Attach type(s): BPF_ATTACH_TYPE_BIND"

BPF_PROG_TYPE_CGROUP_SOCK_ADDR
"Program type for handling various socket operations
Attach type(s): BPF_CGROUP_INET4_CONNECT BPF_CGROUP_INET6_CONNECT
BPF_CGROUP_INET4_RECV_ACCEPT BPF_CGROUP_INET6_RECV_ACCEPT"

BPF_PROG_TYPE_SOCK_OPS
"Program type for handling socket event notifications such as connection established
Attach type(s): BPF_CGROUP_SOCK_OPS"

black hat USA 2022

eBPF for Windows libbpf API

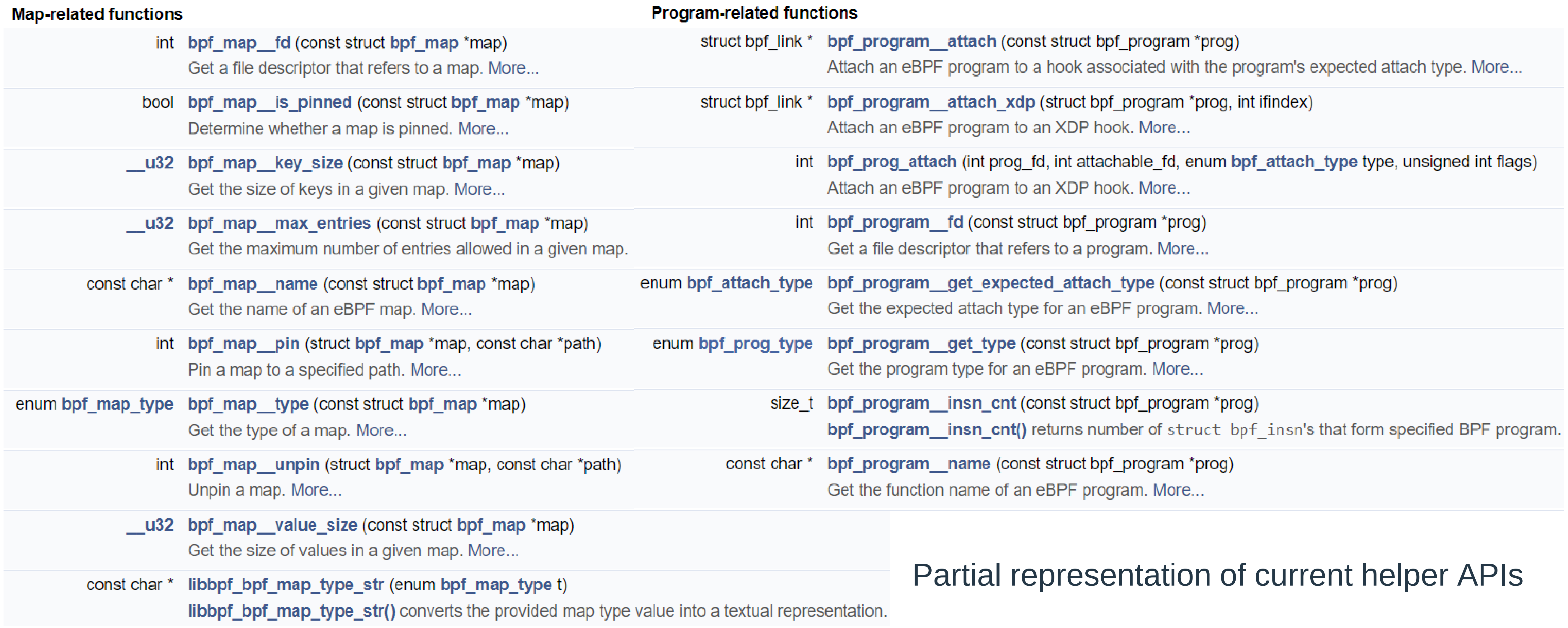


Functions

void *	**bpf_map_lookup_elem** (struct **bpf_map** *map, void *key) Get a pointer to an entry in the map. More...
int64_t	**bpf_map_update_elem** (struct **bpf_map** *map, void *key, void *value, uint64_t flags) Insert or update an entry in the map. More...
int64_t	**bpf_map_delete_elem** (struct **bpf_map** *map, void *key) Remove an entry from the map. More...
int64_t	**bpf_tail_call** (void *ctx, struct **bpf_map** *prog_array_map, uint32_t index) Perform a tail call into another eBPF program. More...
uint32_t	**bpf_get_prandom_u32** () Get a pseudo-random number. More...
uint64_t	**bpf_ktime_get_boot_ns** () Return time elapsed since boot in nanoseconds including time while suspended. Mor
uint64_t	**bpf_get_smp_processor_id** () Return SMP id of the processor running the program. More...
int	**bpf_csum_diff** (void *from, int from_size, void *to, int to_size, int seed) Computes difference of checksum values for two input raw buffers using 1's complement arithmetic.
int	**bpf_ringbuf_output** (struct **bpf_map** *ring_buffer, void *data, uint64_t size, uint64_t flags) Copy data into the ring buffer map. More...
long	**bpf_printk** (const char *fmt,...) Print debug output. For instructions on viewing the output, see the Using tracing section of the Getting Started Guide for eBPF for Windows. More...
int64_t	**bpf_map_push_elem** (struct **bpf_map** *map, void *value, uint64_t flags) Insert an element at the end of the map (only valid for stack and queue). More...
int64_t	**bpf_map_pop_elem** (struct **bpf_map** *map, void *value) Copy an entry from the map and remove it from the map (only valid for stack and queue). Queue pops from the beginning of the map. Stack pops from the end of the map. More...
int64_t	**bpf_map_peek_elem** (struct **bpf_map** *map, void *value) Copy an entry from the map (only valid for stack and queue). Queue peeks at the beginning of the map. Stack peeks at the end of the map. More...
uint64_t	**bpf_get_current_pid_tgid** () Get the current thread ID (PID) and process ID (TGID). More...

Partial representation of current helper APIs

eBPF for Windows libbpf API



Map-related functions

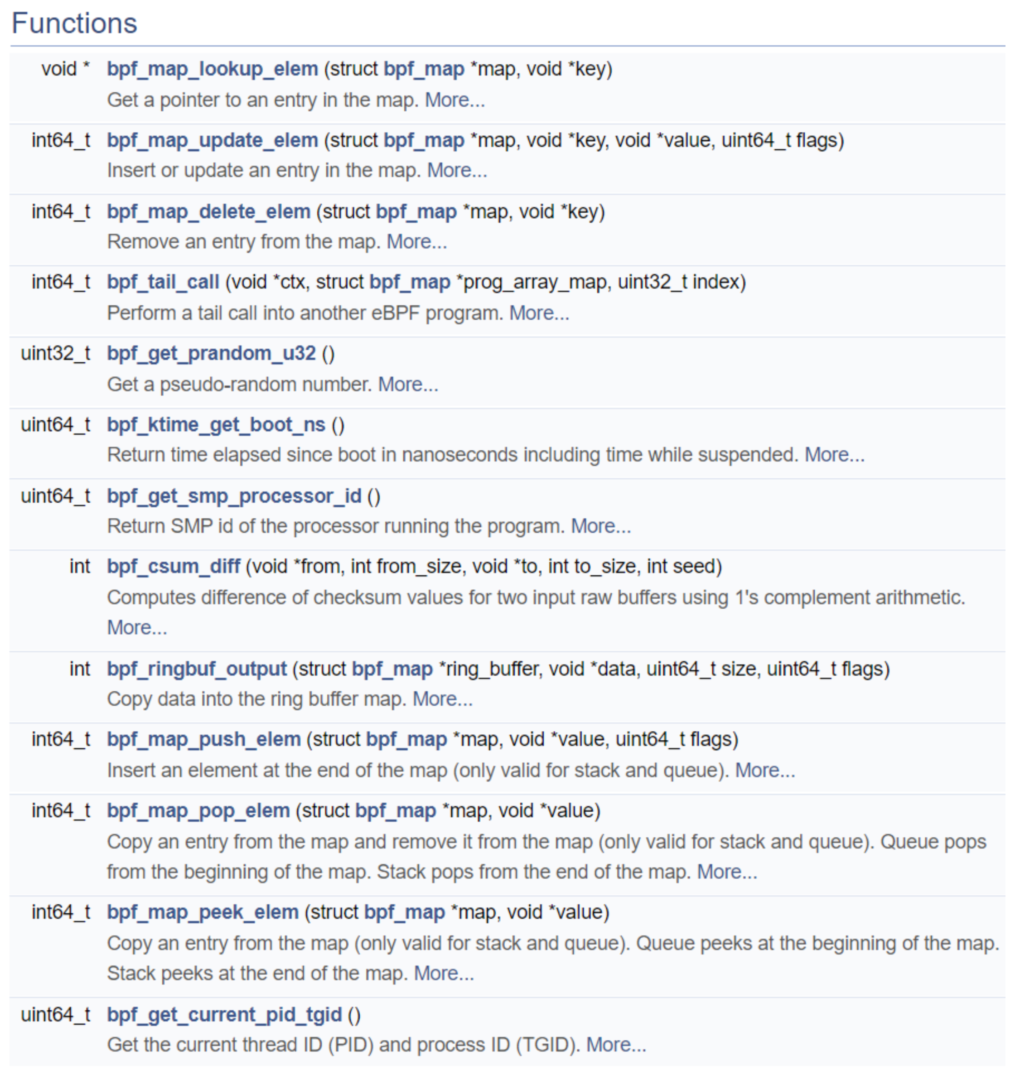
int	bpf_map__fd (const struct bpf_map *map) Get a file descriptor that refers to a map. More...
bool	bpf_map__is_pinned (const struct bpf_map *map) Determine whether a map is pinned. More...
__u32	bpf_map__key_size (const struct bpf_map *map) Get the size of keys in a given map. More...
__u32	bpf_map__max_entries (const struct bpf_map *map) Get the maximum number of entries allowed in a given map.
const char *	bpf_map__name (const struct bpf_map *map) Get the name of an eBPF map. More...
int	bpf_map__pin (struct bpf_map *map, const char *path) Pin a map to a specified path. More...
enum bpf_map_type	bpf_map__type (const struct bpf_map *map) Get the type of a map. More...
int	bpf_map__unpin (struct bpf_map *map, const char *path) Unpin a map. More...
__u32	bpf_map__value_size (const struct bpf_map *map) Get the size of values in a given map. More...
const char *	libbpf_bpf_map_type_str (enum bpf_map_type t) libbpf_bpf_map_type_str() converts the provided map type value into a textual representation.

Program-related functions

struct bpf_link *	bpf_program__attach (const struct bpf_program *prog) Attach an eBPF program to a hook associated with the program's expected attach type. More...
struct bpf_link *	bpf_program__attach_xdp (struct bpf_program *prog, int ifindex) Attach an eBPF program to an XDP hook. More...
int	bpf_prog_attach (int prog_fd, int attachable_fd, enum bpf_attach_type type, unsigned int flags) Attach an eBPF program to an XDP hook. More...
int	bpf_program__fd (const struct bpf_program *prog) Get a file descriptor that refers to a program. More...
enum bpf_attach_type	bpf_program__get_expected_attach_type (const struct bpf_program *prog) Get the expected attach type for an eBPF program. More...
enum bpf_prog_type	bpf_program__get_type (const struct bpf_program *prog) Get the program type for an eBPF program. More...
size_t	bpf_program__insn_cnt (const struct bpf_program *prog) bpf_program__insn_cnt() returns number of `struct bpf_insn`'s that form specified BPF program.
const char *	bpf_program__name (const struct bpf_program *prog) Get the function name of an eBPF program. More...

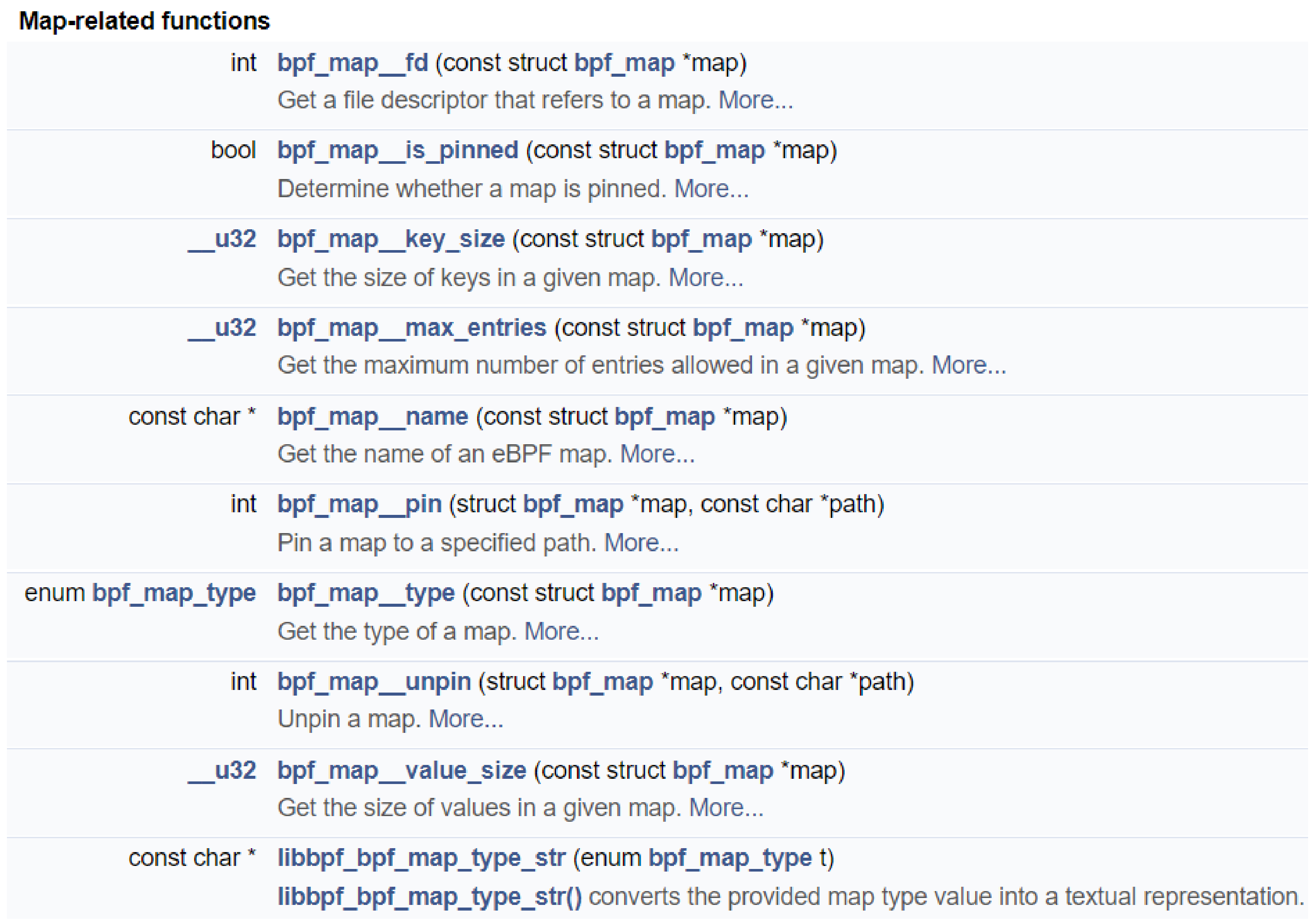
Partial representation of current helper APIs

eBPF for Windows libbpf API



Functions

void *	**bpf_map_lookup_elem** (struct **bpf_map** *map, void *key) Get a pointer to an entry in the map. More...
int64_t	**bpf_map_update_elem** (struct **bpf_map** *map, void *key, void *value, uint64_t flags) Insert or update an entry in the map. More...
int64_t	**bpf_map_delete_elem** (struct **bpf_map** *map, void *key) Remove an entry from the map. More...
int64_t	**bpf_tail_call** (void *ctx, struct **bpf_map** *prog_array_map, uint32_t index) Perform a tail call into another eBPF program. More...
uint32_t	**bpf_get_prandom_u32** () Get a pseudo-random number. More...
uint64_t	**bpf_ktime_get_boot_ns** () Return time elapsed since boot in nanoseconds including time while suspended. More...
uint64_t	**bpf_get_smp_processor_id** () Return SMP id of the processor running the program. More...
int	**bpf_csum_diff** (void *from, int from_size, void *to, int to_size, int seed) Computes difference of checksum values for two input raw buffers using 1's complement arithmetic. More...
int	**bpf_ringbuf_output** (struct **bpf_map** *ring_buffer, void *data, uint64_t size, uint64_t flags) Copy data into the ring buffer map. More...
int64_t	**bpf_map_push_elem** (struct **bpf_map** *map, void *value, uint64_t flags) Insert an element at the end of the map (only valid for stack and queue). More...
int64_t	**bpf_map_pop_elem** (struct **bpf_map** *map, void *value) Copy an entry from the map and remove it from the map (only valid for stack and queue). Queue pops from the beginning of the map. Stack pops from the end of the map. More...
int64_t	**bpf_map_peek_elem** (struct **bpf_map** *map, void *value) Copy an entry from the map (only valid for stack and queue). Queue peeks at the beginning of the map. Stack peeks at the end of the map. More...
uint64_t	**bpf_get_current_pid_tgid** () Get the current thread ID (PID) and process ID (TGID). More...



Map-related functions

int	**bpf_map__fd** (const struct **bpf_map** *map) Get a file descriptor that refers to a map. More...
bool	**bpf_map__is_pinned** (const struct **bpf_map** *map) Determine whether a map is pinned. More...
__u32	**bpf_map__key_size** (const struct **bpf_map** *map) Get the size of keys in a given map. More...
__u32	**bpf_map__max_entries** (const struct **bpf_map** *map) Get the maximum number of entries allowed in a given map. More...
const char *	**bpf_map__name** (const struct **bpf_map** *map) Get the name of an eBPF map. More...
int	**bpf_map__pin** (struct **bpf_map** *map, const char *path) Pin a map to a specified path. More...
enum **bpf_map_type**	**bpf_map__type** (const struct **bpf_map** *map) Get the type of a map. More...
int	**bpf_map__unpin** (struct **bpf_map** *map, const char *path) Unpin a map. More...
__u32	**bpf_map__value_size** (const struct **bpf_map** *map) Get the size of values in a given map. More...
const char *	**libbpf_bpf_map_type_str** (enum **bpf_map_type** t) **libbpf_bpf_map_type_str()** converts the provided map type value into a textual representation.

eBPF for Windows Security Model

eBPF for Windows allows unsigned code to run in the kernel

Current DACLs require Administrative access to interact with the trusted service in userland or the driver directly via IOCTLs to load eBPF programs

When eBPF bytecode is loaded by the service, a static verifier checks to ensure the program will terminate within a certain number of instructions and not access out of bounds memory.

The VM engine then can JIT code to x64 and pass native instructions to the kernel or run in an interpreted mode executing the eBPF bytecode in the kernel* (Debug mode only)

eBPF for Windows Static Verifier

On Linux, the kernel has it’s own static verifier that runs when eBPF code is loaded via system calls

On Windows, an opensource component called PREVAIL has been used

PREVAIL has stronger security guarantees and uses abstract interpretation for a sound analysis

Modern advancements in eBPF such as loops and tail calls are allowed

eBPF for Windows Execution Engine

On Linux, the original kernel implementation of the eBPF bytecode execution engine is GPL licensed

On Windows, an opensource third party component from the IO Visor Project called uBPF is used (https://github.com/iovisor/ubpf)

uBPF (Userspace eBPF VM) is BSD licensed and can run in user or kernel contexts

uBPF can be leveraged by other projects as a replacement for Lua or Javascript

eBPF for Windows Security Guarantees

The combination of the static verifier and sandboxed execution attempt to provide the following security guarantees:

- eBPF Programs will terminate within a reasonable amount of time (limited by instruction counts, loops are unrolled, etc)
- eBPF Programs will not read memory outside the bounds specified at compile time
- Registers are checked for value ranges, uninitialized use
- Stack references are contained to memory written by the program
- Arguments to function calls are type checked
- Pointers must be checked for NULL before dereferencing
- eBPF for Windows can also be run in a secure HVCI mode*

eBPF for Windows Attack Scenarios

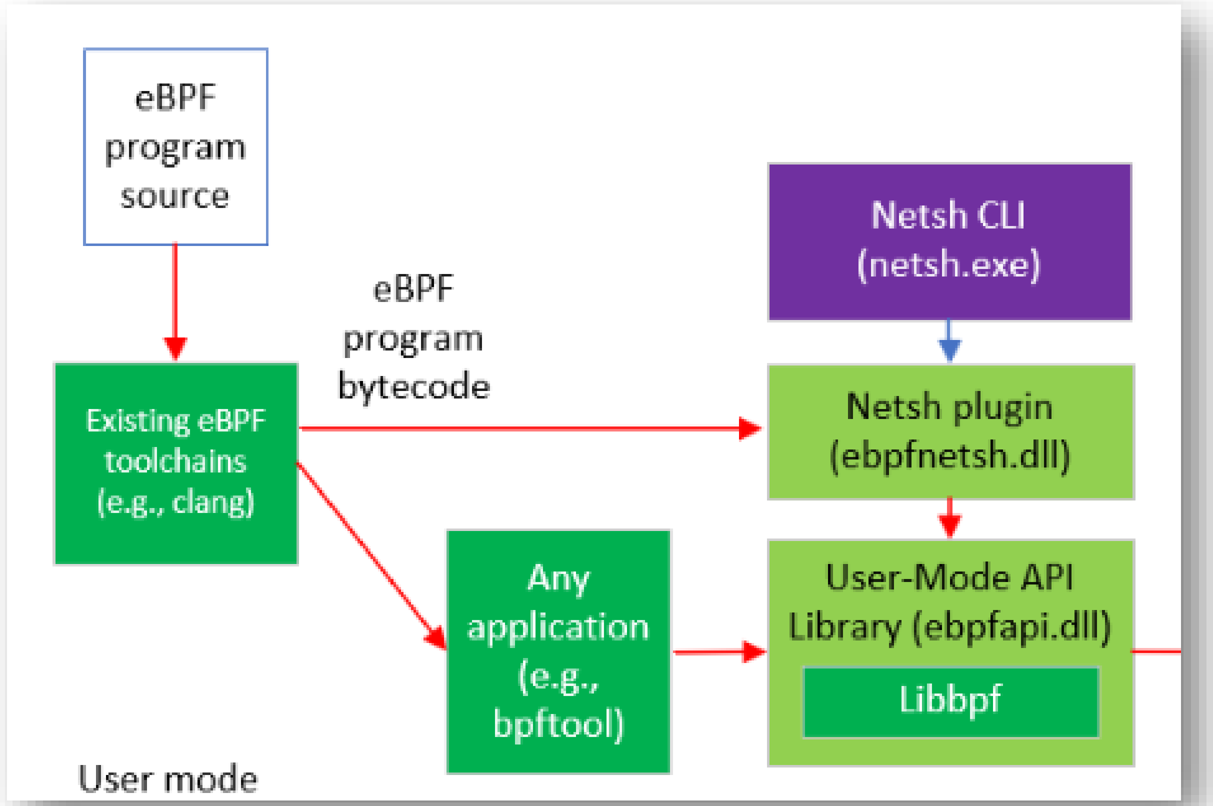
Valid attack scenarios include:

- Code execution as Administrator due to parsing errors on loading 3rd party modules
- Code execution in the trusted service via RPC API implementation errors
- Code execution in the trusted service via static verifier or JIT compiler bugs
- Code execution in the kernel via static verifier, JIT compiler, or interpreter bugs
- Code execution in the kernel via IOCTL implementation errors
- Code execution in the kernel via shim hook implementation errors

eBPF4Win API (ebpfapi.dll)

The initial set of components in the eBPF for Windows stack involve the user facing API contained in ebpfapi.dll that allows loading and unloading programs, creating and deleting maps, and so on.

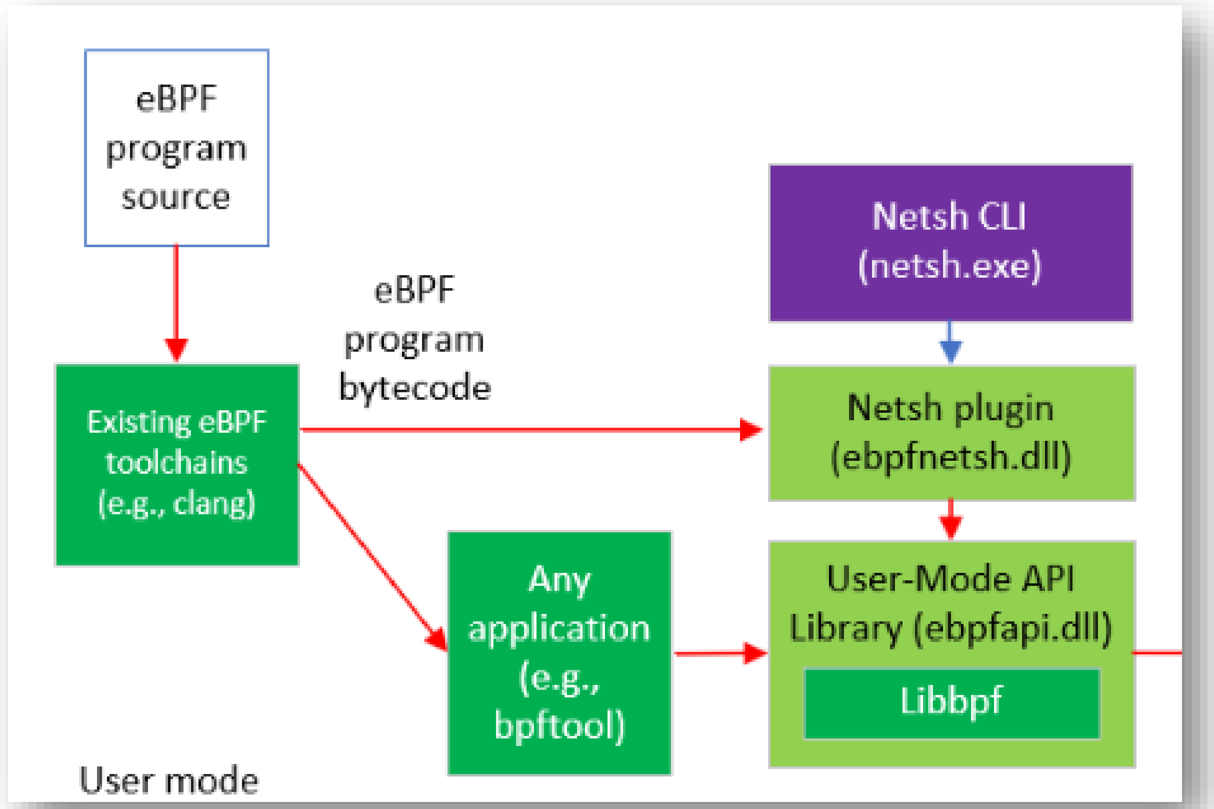
ebpfapi.dll is exposed through the bpftool.exe and netsh interfaces and contains the API set shown previously for loading programs, manipulating maps, and the ability to verify ELF sections from file path or memory


eBPF program source
eBPF program bytecode
Netsh CLI (netsh.exe)
Existing eBPF toolchains (e.g., clang)
Netsh plugin (ebpfnetsh.dll)
Any application (e.g., bpftool)
User-Mode API Library (ebpfapi.dll)
Libbpf
User mode

Fuzzing ebpfapi.dll

To fuzz the ELF loading API, we used a combination of fuzzing the PREVAIL verifier code on Linux and cross fuzzing as well as directly harnessing ebpfapi.dll APIs with libfuzzer

We will show some of the cross fuzzing results later but here is the first vulnerability we submitted to Microsoft..


eBPF program source
eBPF program bytecode
Netsh CLI (netsh.exe)
Existing eBPF toolchains (e.g., clang)
Netsh plugin (ebpfnetsh.dll)
Any application (e.g., bpftool)
User-Mode API Library (ebpfapi.dll)
Libbpf
User mode

EbpfApi Arbitrary Code Execution

Our first vulnerability is a heap corruption which calls free() on user controlled data during the parsing of the ELF object containing an eBPF program. Initial corruption occurs during the parsing of ELF relocation sections.

```
CommandLine: bpftool.exe prog load crash.o xdp

===========================================================
VERIFIER STOP 000000000000000F: pid 0x2D24: corrupted suffix pattern

        00000267F2D91000 : Heap handle
        00000267F3AA2FC0 : Heap block
        0000000000000038 : Block size
        00000267F3AA2FF8 : corruption address
===========================================================

...

0:000> db 00000267F3AA2FF8 l20
00000267`f3aa2ff8  41 41 41 41 00 d0 d0 d0-?? ?? ?? ?? ?? ?? ?? ??  AAAA....????????
00000267`f3aa3008  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

EbpfApi Arbitrary Code Execution

This attack would involve an Administrator loading a malicious prebuilt eBPF program or compiling a malicious project file which contained header data for an undersized relocation section which, when free()'d by the destructor for the relocation object would allow an attacker arbitrary code execution

```
0:000> k
 # Child-SP          RetAddr               Call Site
...
07 0000003c`c56ff060 00007ffc`151185ca     verifier!AVrfp_ucrt_free+0x4d
08 (Inline Function) --------`--------     EbpfApi!std::_Deallocate+0x2a
09 (Inline Function) --------`--------     EbpfApi!std::allocator<ebpf_inst>::deallocate+0x2e
0a (Inline Function) --------`--------     EbpfApi!std::vector<ebpf_inst,std::allocator<ebpf_inst> >::_Tidy+0x40
0b (Inline Function) --------`--------     EbpfApi!std::vector<ebpf_inst,std::allocator<ebpf_inst> >::{dtor}+0x40
0c 0000003c`c56ff090 00007ffc`15144778     EbpfApi!raw_program::~raw_program+0x7a
0d 0000003c`c56ff0c0 00007ffc`15144fac     EbpfApi!read_elf+0x9a8
0e 0000003c`c56ff550 00007ffc`15114fa0     EbpfApi!read_elf+0xbc
0f 0000003c`c56ff790 00007ffc`1510151b     EbpfApi!load_byte_code+0x140
10 0000003c`c56ffa50 00007ffc`1510374d     EbpfApi!_initialize_ebpf_object_from_elf+0x16b
11 0000003c`c56ffb30 00007ffc`1513c81e     EbpfApi!ebpf_object_open+0x1ed
```

EbpfApi Arbitrary Code Execution

Due to the looping nature of ELF parsing and arbitrary control of sizes and contents, we have high confidence this vulnerability can be exploited in practice

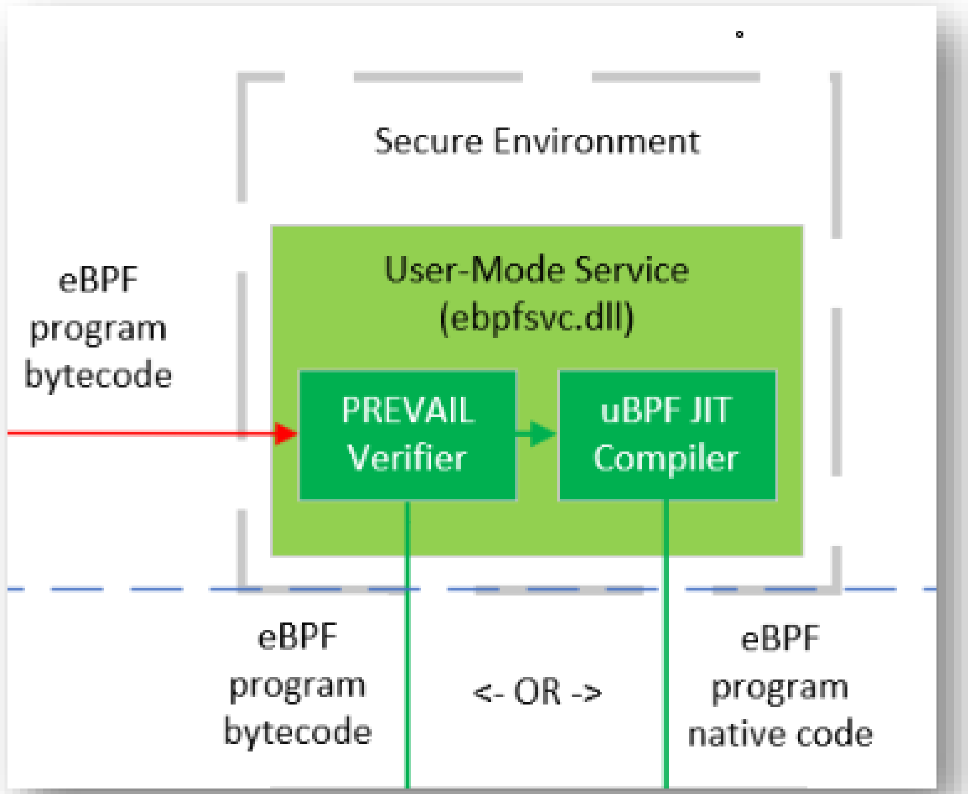
```
0:000> !heap -p -a 000001e45c188c98
    address 000001e45c188c98 found in
    _HEAP @ 1e45c100000
              HEAP_ENTRY Size Prev Flags            UserPtr UserSize - state
        000001e45c188c10 000b 0000  [00]   000001e45c188c60    00038 - (busy)
        7ffc18c044c1 verifier!AVrfDebugPageHeapAllocate+0x0000000000000431
        ...
        7ffc1513caef EbpfApi!operator new+0x000000000000001f
        7ffc151425f4 EbpfApi!std::vector<ebpf_inst,std::allocator<ebpf_inst> >::_Range_construct_or_tidy<ebpf_inst *
                                                                                                __ptr64>+0x0000000000000064
        7ffc15142c67 EbpfApi!ELFIO::relocation_section_accessor_template<ELFIO::section const
                                                                    >::generic_get_entry_rela<ELFIO::Elf64_Rela>+0x0000000000000177
        7ffc15144258 EbpfApi!read_elf+0x0000000000000488
        7ffc15144fac EbpfApi!read_elf+0x00000000000000bc
        7ffc15114fa0 EbpfApi!load_byte_code+0x0000000000000140
        7ffc1510151b EbpfApi!_initialize_ebpf_object_from_elf+0x000000000000016b
        7ffc1510374d EbpfApi!ebpf_object_open+0x00000000000001ed
```

eBPF4Win Service (ebpfsvc.dll)

The eBPF for Windows Service contains PREVAIL and uBPF code bases and exposes an RPC based API

The RPC service exports a single API for verifying and loading a program:

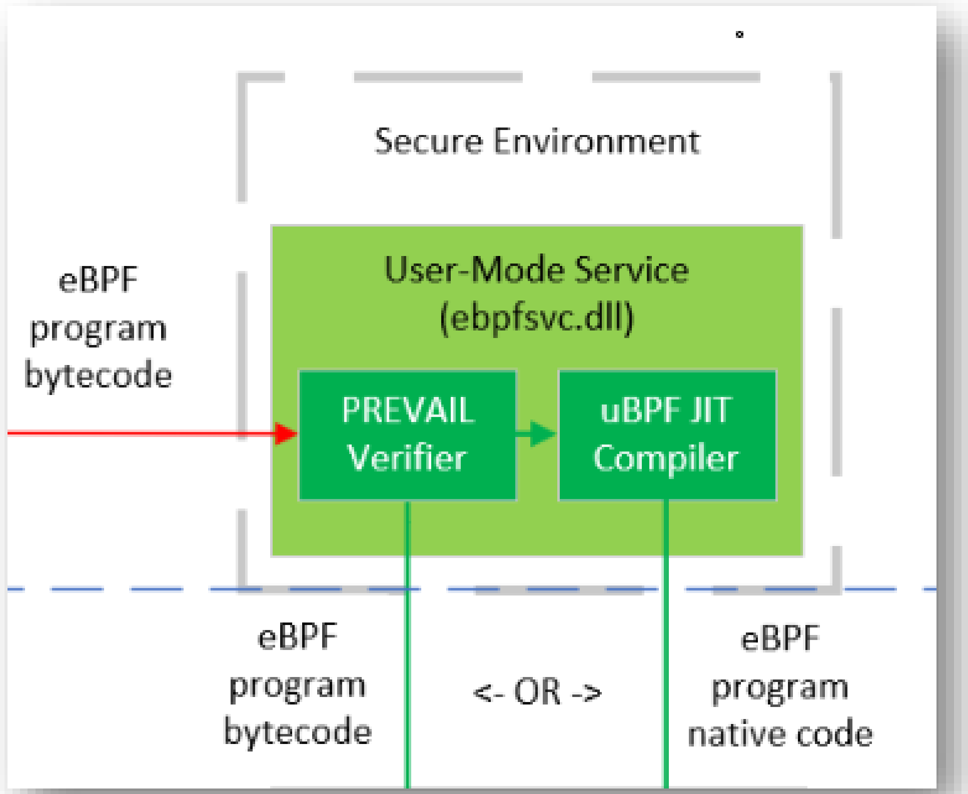
```
ebpf_result_t verify_and_load_program(
            [ in, ref ] ebpf_program_load_info * info,
            [ out, ref ] uint32_t * logs_size,
            [ out, size_is(, *logs_size), ref ] char** logs);
```



eBPF4Win Service (ebpfsvc.dll)

The **verify_and_load_program** RPC API is called through the internal API **ebpf_program_load_bytes** function that is ultimately exposed as part of the libbpf API **bpf_prog_load**

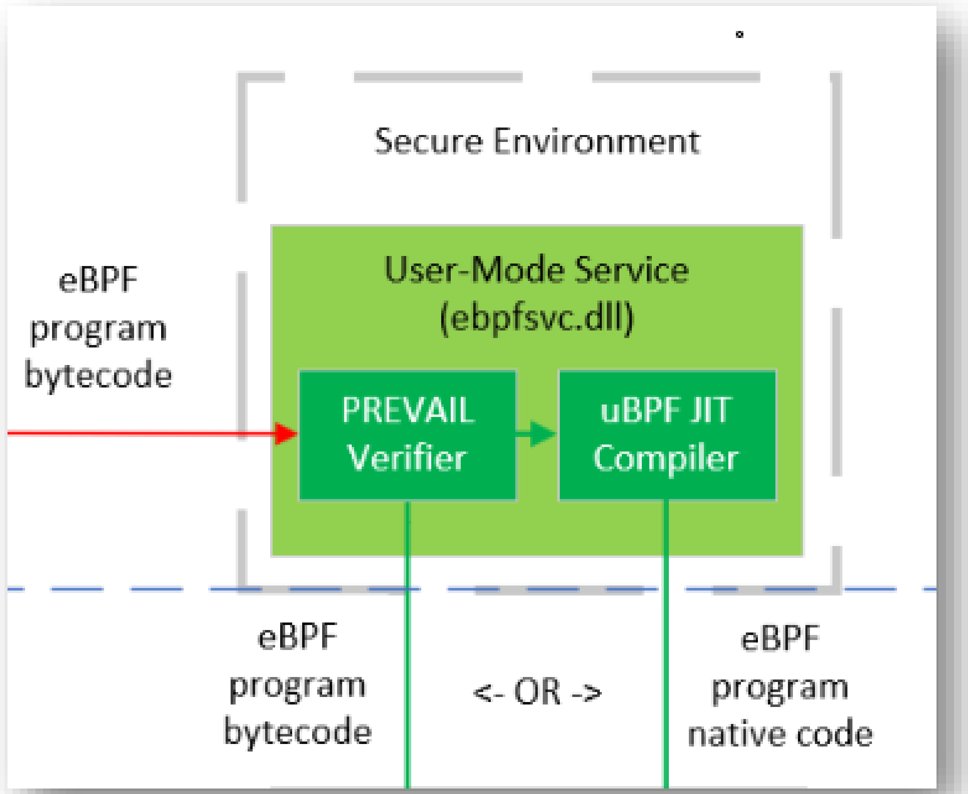
It is also called by the **ebpf_object_load** function which is contained in EbpfAPI and is how netsh and bpftool load programs via the service



PREVAIL Static Verifier

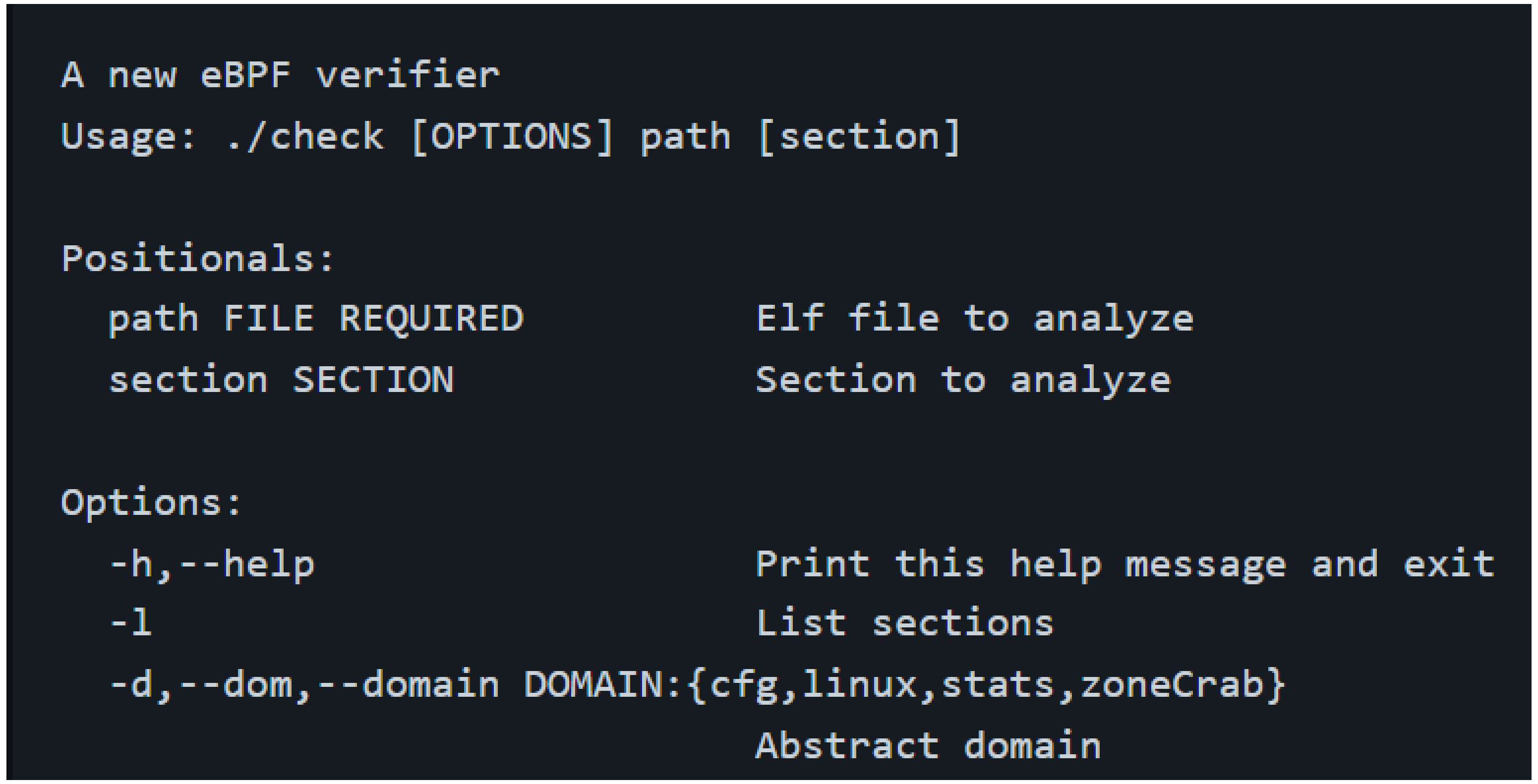
The PREVAIL Static Verifier is “a Polynomial-Runtime EBPF Verifier using an Abstract Interpretation Layer”

Designed to be faster and more precise than the Linux static verifier and it is dual licensed MIT and Apache so it can be used anywhere alongside uBPF

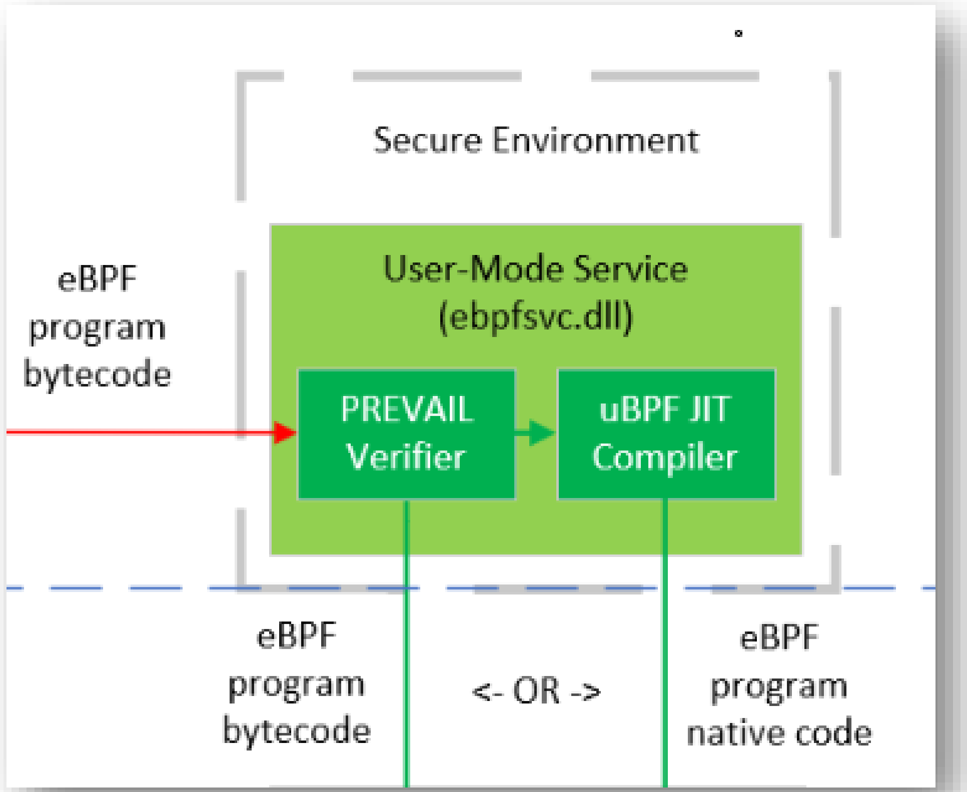


PREVAIL Static Verifier

It includes a simple standalone tool called ‘check’ which is easily fuzzed with a file fuzzing approach

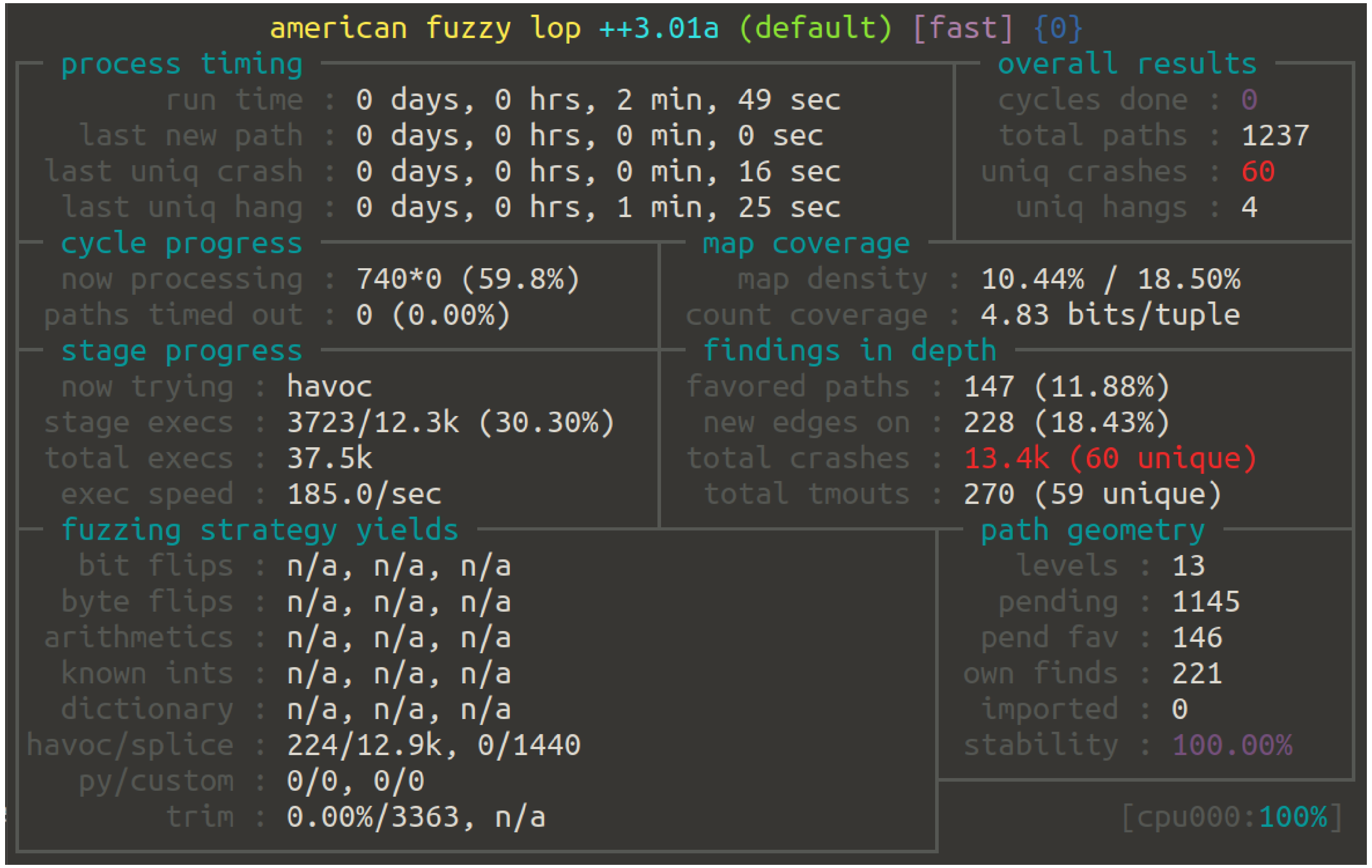


```
A new eBPF verifier
Usage: ./check [OPTIONS] path [section]

Positionals:
  path FILE REQUIRED           Elf file to analyze
  section SECTION              Section to analyze

Options:
  -h,--help                    Print this help message and exit
  -l                           List sections
  -d,--dom,--domain DOMAIN:{cfg,linux,stats,zoneCrab}
                               Abstract domain
```



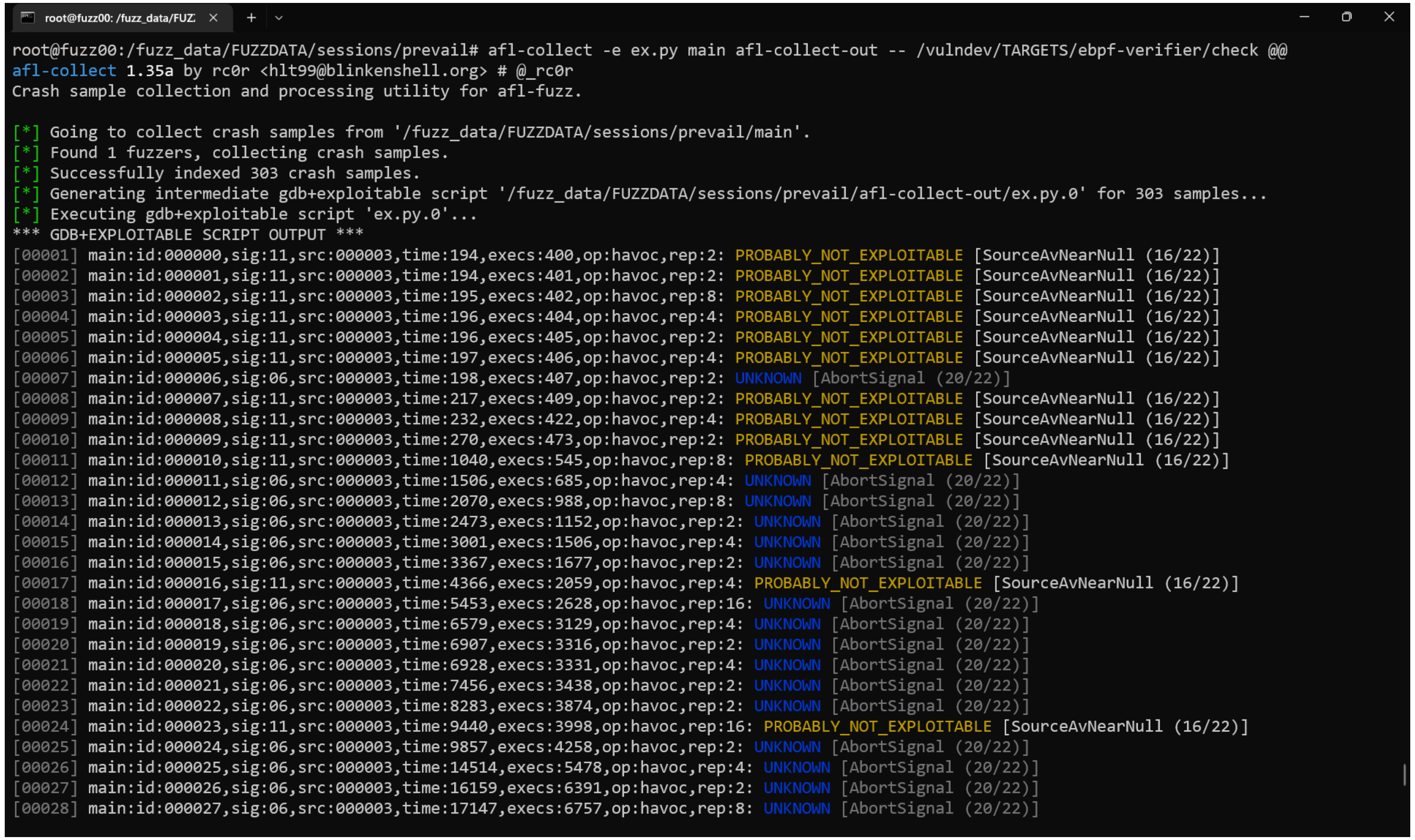
black hat USA 2022

Fuzzing PREVAIL



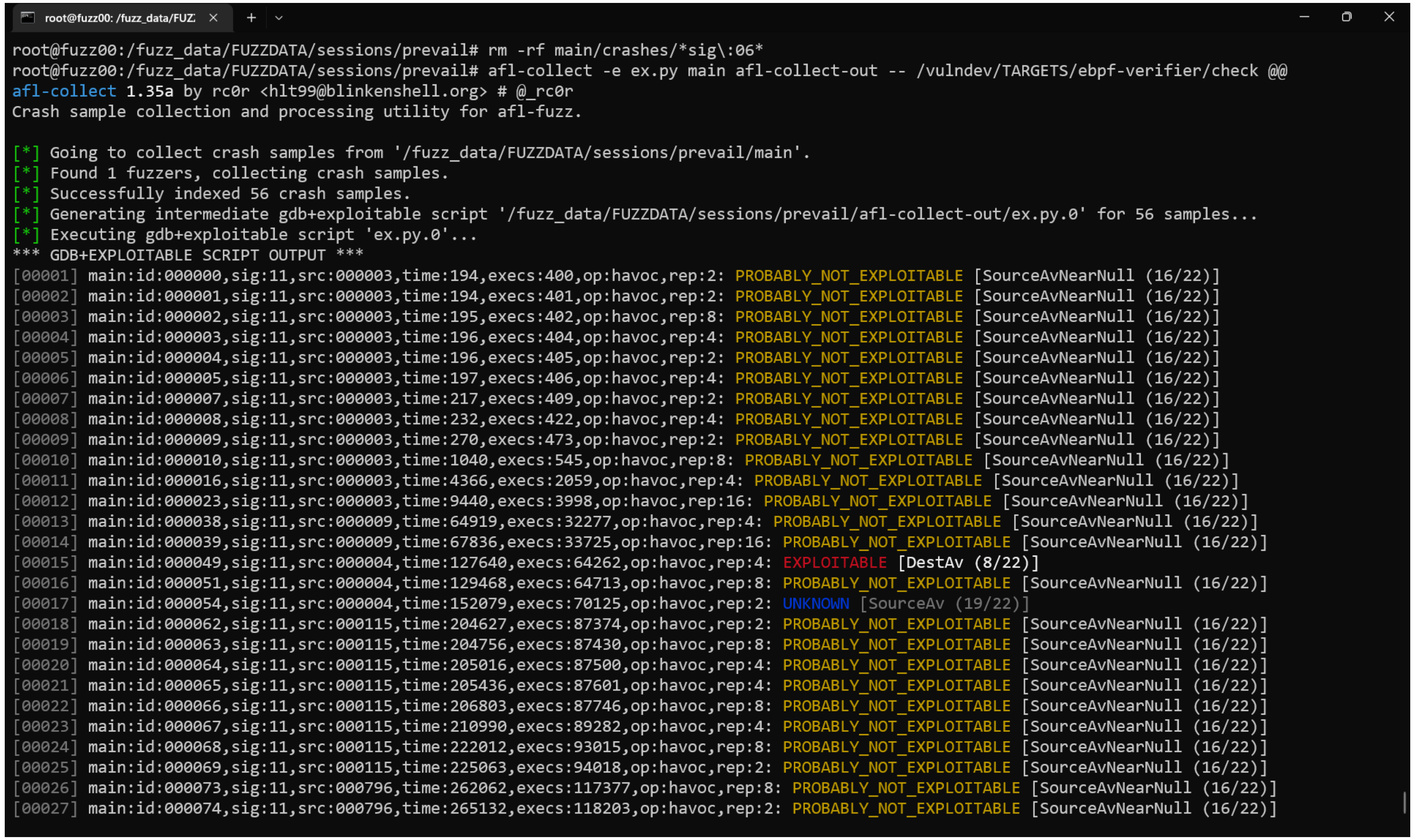
```
              american fuzzy lop ++3.01a (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 2 min, 49 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec       │  total paths : 1237  │
│ last uniq crash : 0 days, 0 hrs, 0 min, 16 sec      │ uniq crashes : 60    │
│  last uniq hang : 0 days, 0 hrs, 1 min, 25 sec      │   uniq hangs : 4     │
├─ cycle progress ───────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 740*0 (59.8%)    │    map density : 10.44% / 18.50%      │
│ paths timed out : 0 (0.00%)        │ count coverage : 4.83 bits/tuple      │
├─ stage progress ───────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc                │ favored paths : 147 (11.88%)          │
│ stage execs : 3723/12.3k (30.30%)  │  new edges on : 228 (18.43%)          │
│ total execs : 37.5k                │ total crashes : 13.4k (60 unique)     │
│  exec speed : 185.0/sec            │  total tmouts : 270 (59 unique)       │
├─ fuzzing strategy yields ──────────┴───────────┬─ path geometry ───────────┤
│   bit flips : n/a, n/a, n/a                    │    levels : 13            │
│  byte flips : n/a, n/a, n/a                    │   pending : 1145          │
│ arithmetics : n/a, n/a, n/a                    │  pend fav : 146           │
│  known ints : n/a, n/a, n/a                    │ own finds : 221           │
│  dictionary : n/a, n/a, n/a                    │  imported : 0             │
│havoc/splice : 224/12.9k, 0/1440                │ stability : 100.00%       │
│   py/custom : 0/0, 0/0                         ├───────────────────────────┘
│        trim : 0.00%/3363, n/a                  │          [cpu000:100%]
└────────────────────────────────────────────────┘
```

Fuzzing PREVAIL



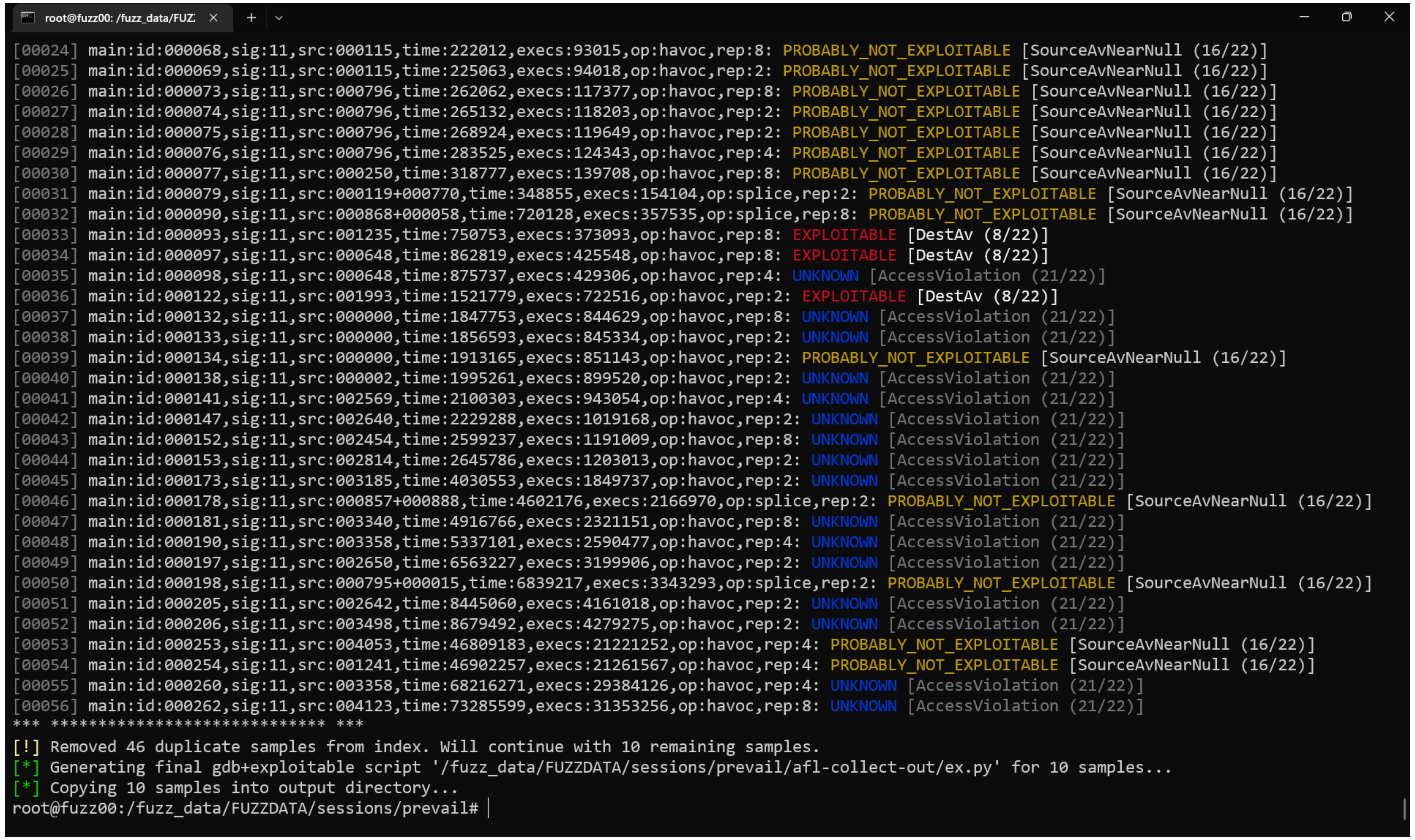
```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/prevail# afl-collect -e ex.py main afl-collect-out -- /vulndev/TARGETS/ebpf-verifier/check @@
afl-collect 1.35a by rc0r <hlt99@blinkenshell.org> # @_rc0r
Crash sample collection and processing utility for afl-fuzz.

[*] Going to collect crash samples from '/fuzz_data/FUZZDATA/sessions/prevail/main'.
[*] Found 1 fuzzers, collecting crash samples.
[*] Successfully indexed 303 crash samples.
[*] Generating intermediate gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/prevail/afl-collect-out/ex.py.0' for 303 samples...
[*] Executing gdb+exploitable script 'ex.py.0'...
*** GDB+EXPLOITABLE SCRIPT OUTPUT ***
[00001] main:id:000000,sig:11,src:000003,time:194,execs:400,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00002] main:id:000001,sig:11,src:000003,time:194,execs:401,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00003] main:id:000002,sig:11,src:000003,time:195,execs:402,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00004] main:id:000003,sig:11,src:000003,time:196,execs:404,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00005] main:id:000004,sig:11,src:000003,time:196,execs:405,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00006] main:id:000005,sig:11,src:000003,time:197,execs:406,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00007] main:id:000006,sig:06,src:000003,time:198,execs:407,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00008] main:id:000007,sig:11,src:000003,time:217,execs:409,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00009] main:id:000008,sig:11,src:000003,time:232,execs:422,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00010] main:id:000009,sig:11,src:000003,time:270,execs:473,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00011] main:id:000010,sig:11,src:000003,time:1040,execs:545,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00012] main:id:000011,sig:06,src:000003,time:1506,execs:685,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00013] main:id:000012,sig:06,src:000003,time:2070,execs:988,op:havoc,rep:8: UNKNOWN [AbortSignal (20/22)]
[00014] main:id:000013,sig:06,src:000003,time:2473,execs:1152,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00015] main:id:000014,sig:06,src:000003,time:3001,execs:1506,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00016] main:id:000015,sig:06,src:000003,time:3367,execs:1677,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00017] main:id:000016,sig:11,src:000003,time:4366,execs:2059,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00018] main:id:000017,sig:06,src:000003,time:5453,execs:2628,op:havoc,rep:16: UNKNOWN [AbortSignal (20/22)]
[00019] main:id:000018,sig:06,src:000003,time:6579,execs:3129,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00020] main:id:000019,sig:06,src:000003,time:6907,execs:3316,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00021] main:id:000020,sig:06,src:000003,time:6928,execs:3331,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00022] main:id:000021,sig:06,src:000003,time:7456,execs:3438,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00023] main:id:000022,sig:06,src:000003,time:8283,execs:3874,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00024] main:id:000023,sig:11,src:000003,time:9440,execs:3998,op:havoc,rep:16: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00025] main:id:000024,sig:06,src:000003,time:9857,execs:4258,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00026] main:id:000025,sig:06,src:000003,time:14514,execs:5478,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00027] main:id:000026,sig:06,src:000003,time:16159,execs:6391,op:havoc,rep:2: UNKNOWN [AbortSignal (20/22)]
[00028] main:id:000027,sig:06,src:000003,time:17147,execs:6757,op:havoc,rep:8: UNKNOWN [AbortSignal (20/22)]
```

Fuzzing PREVAIL



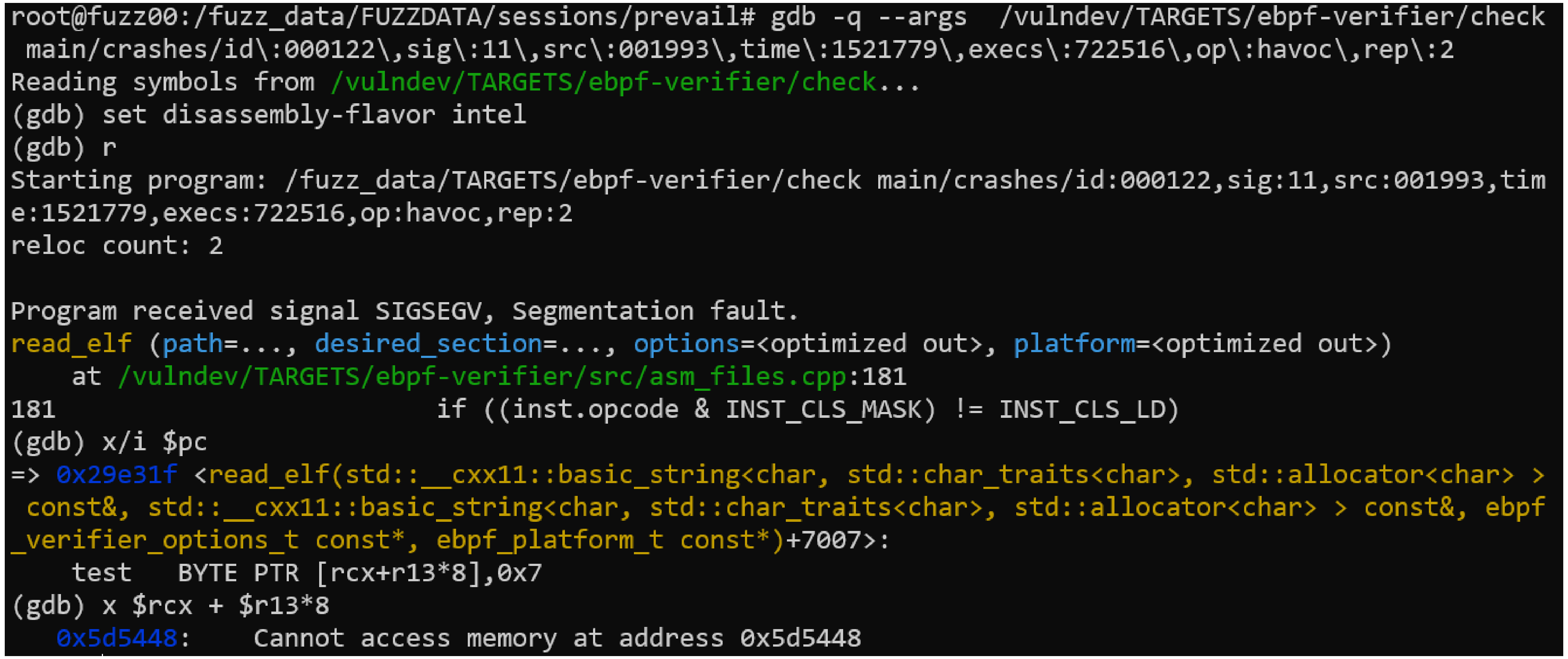
```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/prevail# rm -rf main/crashes/*sig\:06*
root@fuzz00:/fuzz_data/FUZZDATA/sessions/prevail# afl-collect -e ex.py main afl-collect-out -- /vulndev/TARGETS/ebpf-verifier/check @@
afl-collect 1.35a by rc0r <hlt99@blinkenshell.org> # @_rc0r
Crash sample collection and processing utility for afl-fuzz.

[*] Going to collect crash samples from '/fuzz_data/FUZZDATA/sessions/prevail/main'.
[*] Found 1 fuzzers, collecting crash samples.
[*] Successfully indexed 56 crash samples.
[*] Generating intermediate gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/prevail/afl-collect-out/ex.py.0' for 56 samples...
[*] Executing gdb+exploitable script 'ex.py.0'...
*** GDB+EXPLOITABLE SCRIPT OUTPUT ***
[00001] main:id:000000,sig:11,src:000003,time:194,execs:400,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00002] main:id:000001,sig:11,src:000003,time:194,execs:401,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00003] main:id:000002,sig:11,src:000003,time:195,execs:402,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00004] main:id:000003,sig:11,src:000003,time:196,execs:404,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00005] main:id:000004,sig:11,src:000003,time:196,execs:405,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00006] main:id:000005,sig:11,src:000003,time:197,execs:406,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00007] main:id:000007,sig:11,src:000003,time:217,execs:409,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00008] main:id:000008,sig:11,src:000003,time:232,execs:422,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00009] main:id:000009,sig:11,src:000003,time:270,execs:473,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00010] main:id:000010,sig:11,src:000003,time:1040,execs:545,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00011] main:id:000016,sig:11,src:000003,time:4366,execs:2059,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00012] main:id:000023,sig:11,src:000003,time:9440,execs:3998,op:havoc,rep:16: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00013] main:id:000038,sig:11,src:000009,time:64919,execs:32277,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00014] main:id:000039,sig:11,src:000009,time:67836,execs:33725,op:havoc,rep:16: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00015] main:id:000049,sig:11,src:000004,time:127640,execs:64262,op:havoc,rep:4: EXPLOITABLE [DestAv (8/22)]
[00016] main:id:000051,sig:11,src:000004,time:129468,execs:64713,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00017] main:id:000054,sig:11,src:000004,time:152079,execs:70125,op:havoc,rep:2: UNKNOWN [SourceAv (19/22)]
[00018] main:id:000062,sig:11,src:000115,time:204627,execs:87374,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00019] main:id:000063,sig:11,src:000115,time:204756,execs:87430,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00020] main:id:000064,sig:11,src:000115,time:205016,execs:87500,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00021] main:id:000065,sig:11,src:000115,time:205436,execs:87601,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00022] main:id:000066,sig:11,src:000115,time:206803,execs:87746,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00023] main:id:000067,sig:11,src:000115,time:210990,execs:89282,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00024] main:id:000068,sig:11,src:000115,time:222012,execs:93015,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00025] main:id:000069,sig:11,src:000115,time:225063,execs:94018,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00026] main:id:000073,sig:11,src:000796,time:262062,execs:117377,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00027] main:id:000074,sig:11,src:000796,time:265132,execs:118203,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
```

Fuzzing PREVAIL



```
[00024] main:id:000068,sig:11,src:000115,time:222012,execs:93015,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00025] main:id:000069,sig:11,src:000115,time:225063,execs:94018,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00026] main:id:000073,sig:11,src:000796,time:262062,execs:117377,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00027] main:id:000074,sig:11,src:000796,time:265132,execs:118203,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00028] main:id:000075,sig:11,src:000796,time:268924,execs:119649,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00029] main:id:000076,sig:11,src:000796,time:283525,execs:124343,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00030] main:id:000077,sig:11,src:000250,time:318777,execs:139708,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00031] main:id:000079,sig:11,src:000119+000770,time:348855,execs:154104,op:splice,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00032] main:id:000090,sig:11,src:000868+000058,time:720128,execs:357535,op:splice,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00033] main:id:000093,sig:11,src:001235,time:750753,execs:373093,op:havoc,rep:8: EXPLOITABLE [DestAv (8/22)]
[00034] main:id:000097,sig:11,src:000648,time:862819,execs:425548,op:havoc,rep:8: EXPLOITABLE [DestAv (8/22)]
[00035] main:id:000098,sig:11,src:000648,time:875737,execs:429306,op:havoc,rep:4: UNKNOWN [AccessViolation (21/22)]
[00036] main:id:000122,sig:11,src:001993,time:1521779,execs:722516,op:havoc,rep:2: EXPLOITABLE [DestAv (8/22)]
[00037] main:id:000132,sig:11,src:000000,time:1847753,execs:844629,op:havoc,rep:8: UNKNOWN [AccessViolation (21/22)]
[00038] main:id:000133,sig:11,src:000000,time:1856593,execs:845334,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00039] main:id:000134,sig:11,src:000000,time:1913165,execs:851143,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00040] main:id:000138,sig:11,src:000002,time:1995261,execs:899520,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00041] main:id:000141,sig:11,src:002569,time:2100303,execs:943054,op:havoc,rep:4: UNKNOWN [AccessViolation (21/22)]
[00042] main:id:000147,sig:11,src:002640,time:2229288,execs:1019168,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00043] main:id:000152,sig:11,src:002454,time:2599237,execs:1191009,op:havoc,rep:8: UNKNOWN [AccessViolation (21/22)]
[00044] main:id:000153,sig:11,src:002814,time:2645786,execs:1203013,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00045] main:id:000173,sig:11,src:003185,time:4030553,execs:1849737,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00046] main:id:000178,sig:11,src:000857+000888,time:4602176,execs:2166970,op:splice,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00047] main:id:000181,sig:11,src:003340,time:4916766,execs:2321151,op:havoc,rep:8: UNKNOWN [AccessViolation (21/22)]
[00048] main:id:000190,sig:11,src:003358,time:5337101,execs:2590477,op:havoc,rep:4: UNKNOWN [AccessViolation (21/22)]
[00049] main:id:000197,sig:11,src:002650,time:6563227,execs:3199906,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00050] main:id:000198,sig:11,src:000795+000015,time:6839217,execs:3343293,op:splice,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00051] main:id:000205,sig:11,src:002642,time:8445060,execs:4161018,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00052] main:id:000206,sig:11,src:003498,time:8679492,execs:4279275,op:havoc,rep:2: UNKNOWN [AccessViolation (21/22)]
[00053] main:id:000253,sig:11,src:004053,time:46809183,execs:21221252,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00054] main:id:000254,sig:11,src:001241,time:46902257,execs:21261567,op:havoc,rep:4: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00055] main:id:000260,sig:11,src:003358,time:68216271,execs:29384126,op:havoc,rep:4: UNKNOWN [AccessViolation (21/22)]
[00056] main:id:000262,sig:11,src:004123,time:73285599,execs:31353256,op:havoc,rep:8: UNKNOWN [AccessViolation (21/22)]
*** ****************************** ***
[!] Removed 46 duplicate samples from index. Will continue with 10 remaining samples.
[*] Generating final gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/prevail/afl-collect-out/ex.py' for 10 samples...
[*] Copying 10 samples into output directory...
root@fuzz00:/fuzz_data/FUZZDATA/sessions/prevail#
```

Fuzzing PREVAIL



```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/prevail# gdb -q --args  /vulndev/TARGETS/ebpf-verifier/check
 main/crashes/id\:000122\,sig\:11\,src\:001993\,time\:1521779\,execs\:722516\,op\:havoc\,rep\:2
Reading symbols from /vulndev/TARGETS/ebpf-verifier/check...
(gdb) set disassembly-flavor intel
(gdb) r
Starting program: /fuzz_data/TARGETS/ebpf-verifier/check main/crashes/id:000122,sig:11,src:001993,tim
e:1521779,execs:722516,op:havoc,rep:2
reloc count: 2

Program received signal SIGSEGV, Segmentation fault.
read_elf (path=..., desired_section=..., options=<optimized out>, platform=<optimized out>)
    at /vulndev/TARGETS/ebpf-verifier/src/asm_files.cpp:181
181                             if ((inst.opcode & INST_CLS_MASK) != INST_CLS_LD)
(gdb) x/i $pc
=> 0x29e31f <read_elf(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
 const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, ebpf
_verifier_options_t const*, ebpf_platform_t const*)+7007>:
    test   BYTE PTR [rcx+r13*8],0x7
(gdb) x $rcx + $r13*8
   0x5d5448:    Cannot access memory at address 0x5d5448
```

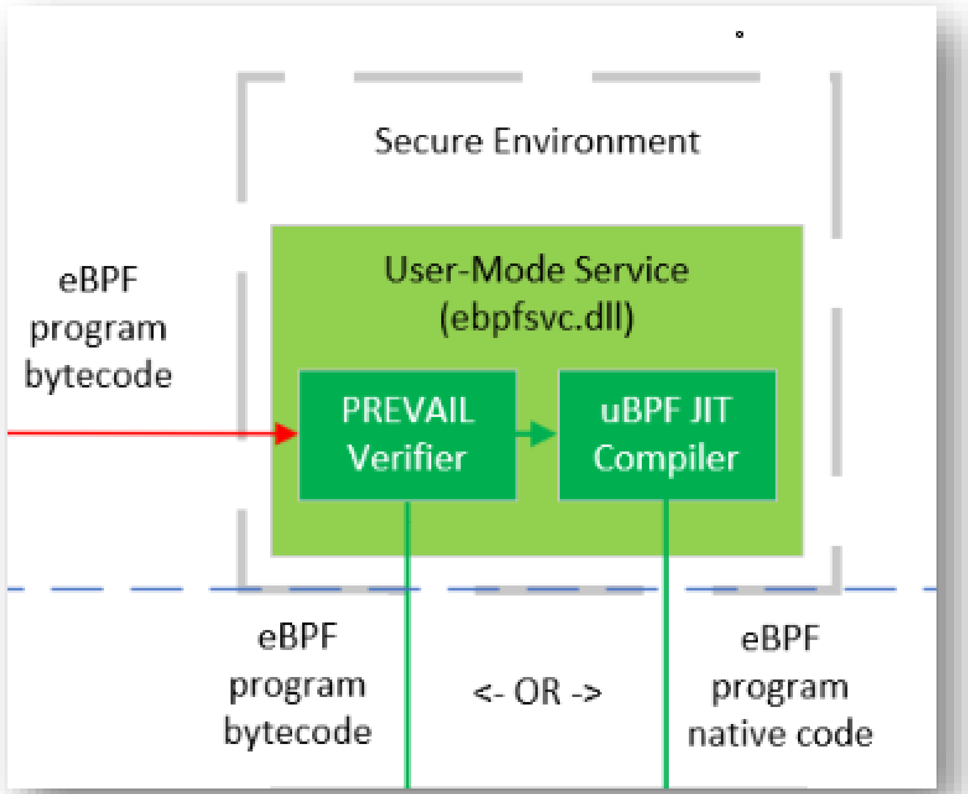
Information Classification: General

#BHUSA @BlackHatEvents

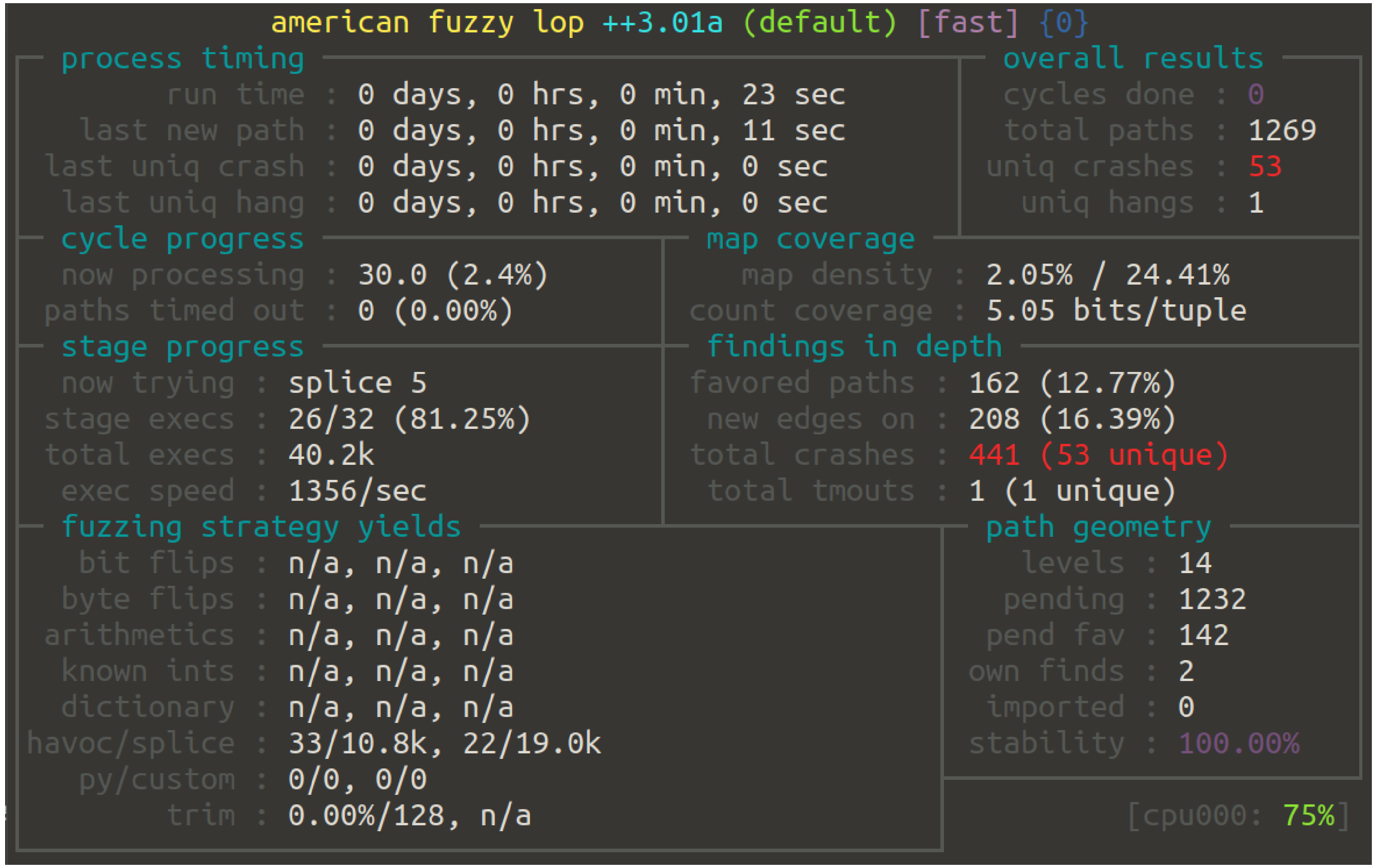
uBPF

uBPF (Userspace BPF) is an independent reimplementation of the eBPF bytecode interpreter and JIT engine that is BSD licensed and can run in user or kernel contexts

Similar to PREVAIL, uBPF comes with a simple reference implementation of the VM with the ability to load and run eBPF programs. It does not have any helper functions or maps available and is only a virtual CPU and execution environment



Fuzzing uBPF

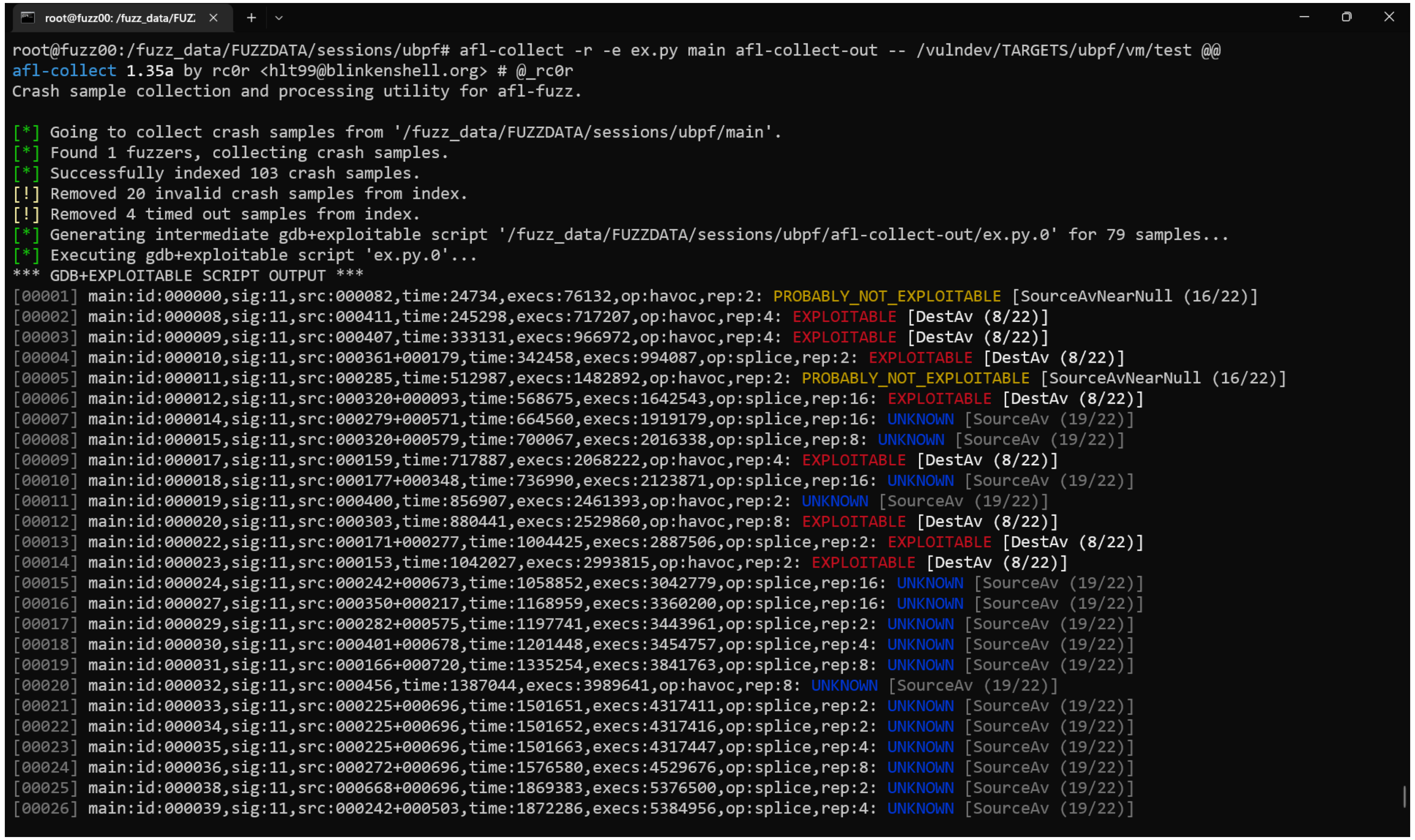


```
                american fuzzy lop ++3.01a (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 23 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 11 sec      │  total paths : 1269  │
│ last uniq crash : 0 days, 0 hrs, 0 min, 0 sec       │ uniq crashes : 53    │
│  last uniq hang : 0 days, 0 hrs, 0 min, 0 sec       │   uniq hangs : 1     │
├─ cycle progress ────────────────┬─ map coverage ────┴──────────────────────┤
│  now processing : 30.0 (2.4%)   │    map density : 2.05% / 24.41%         │
│ paths timed out : 0 (0.00%)     │ count coverage : 5.05 bits/tuple        │
├─ stage progress ────────────────┼─ findings in depth ─────────────────────┤
│  now trying : splice 5          │ favored paths : 162 (12.77%)            │
│ stage execs : 26/32 (81.25%)    │  new edges on : 208 (16.39%)            │
│ total execs : 40.2k             │ total crashes : 441 (53 unique)         │
│  exec speed : 1356/sec          │  total tmouts : 1 (1 unique)            │
├─ fuzzing strategy yields ───────┴──────────────┬─ path geometry ──────────┤
│   bit flips : n/a, n/a, n/a                    │    levels : 14           │
│  byte flips : n/a, n/a, n/a                    │   pending : 1232         │
│ arithmetics : n/a, n/a, n/a                    │  pend fav : 142          │
│  known ints : n/a, n/a, n/a                    │ own finds : 2            │
│  dictionary : n/a, n/a, n/a                    │  imported : 0            │
│havoc/splice : 33/10.8k, 22/19.0k               │ stability : 100.00%      │
│   py/custom : 0/0, 0/0                         ├──────────────────────────┘
│        trim : 0.00%/128, n/a                   │          [cpu000: 75%]
└────────────────────────────────────────────────┘
```

Information Classification: General

#BHUSA @BlackHatEvents

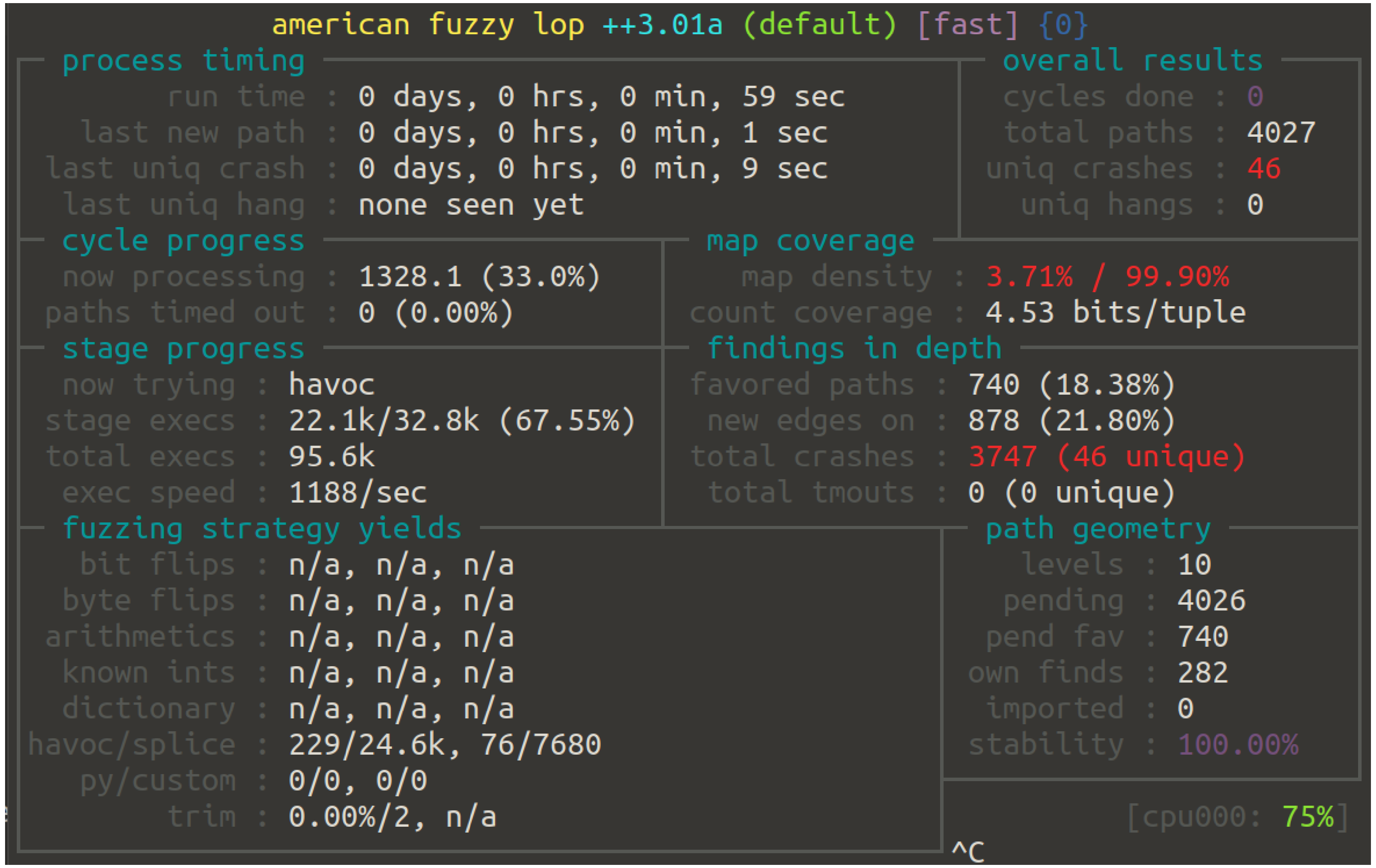
Fuzzing uBPF



root@fuzz00: /fuzz_data/FUZ

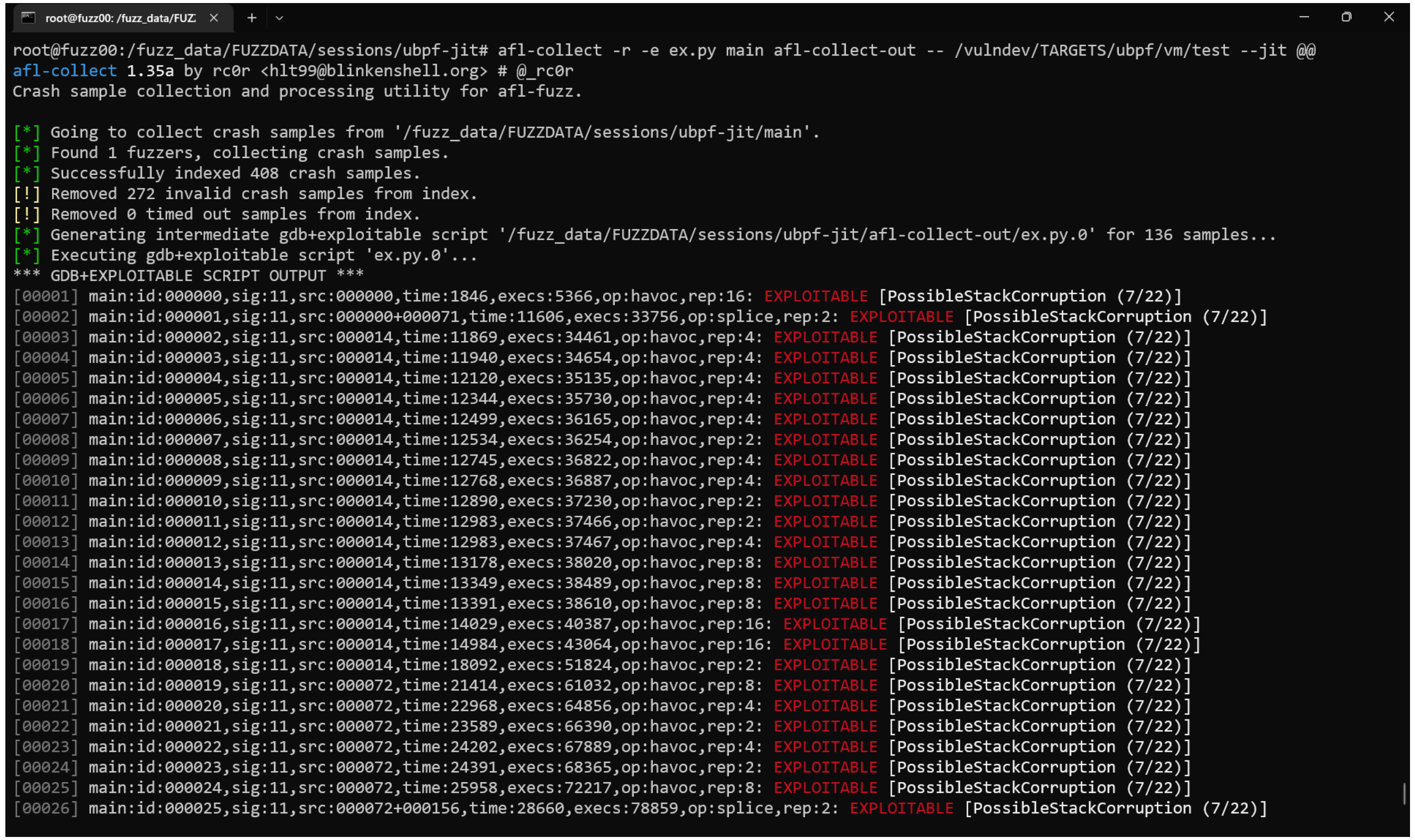
```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/ubpf# afl-collect -r -e ex.py main afl-collect-out -- /vulndev/TARGETS/ubpf/vm/test @@
afl-collect 1.35a by rc0r <hlt99@blinkenshell.org> # @_rc0r
Crash sample collection and processing utility for afl-fuzz.

[*] Going to collect crash samples from '/fuzz_data/FUZZDATA/sessions/ubpf/main'.
[*] Found 1 fuzzers, collecting crash samples.
[*] Successfully indexed 103 crash samples.
[!] Removed 20 invalid crash samples from index.
[!] Removed 4 timed out samples from index.
[*] Generating intermediate gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/ubpf/afl-collect-out/ex.py.0' for 79 samples...
[*] Executing gdb+exploitable script 'ex.py.0'...
*** GDB+EXPLOITABLE SCRIPT OUTPUT ***
[00001] main:id:000000,sig:11,src:000082,time:24734,execs:76132,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00002] main:id:000008,sig:11,src:000411,time:245298,execs:717207,op:havoc,rep:4: EXPLOITABLE [DestAv (8/22)]
[00003] main:id:000009,sig:11,src:000407,time:333131,execs:966972,op:havoc,rep:4: EXPLOITABLE [DestAv (8/22)]
[00004] main:id:000010,sig:11,src:000361+000179,time:342458,execs:994087,op:splice,rep:2: EXPLOITABLE [DestAv (8/22)]
[00005] main:id:000011,sig:11,src:000285,time:512987,execs:1482892,op:havoc,rep:2: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00006] main:id:000012,sig:11,src:000320+000093,time:568675,execs:1642543,op:splice,rep:16: EXPLOITABLE [DestAv (8/22)]
[00007] main:id:000014,sig:11,src:000279+000571,time:664560,execs:1919179,op:splice,rep:16: UNKNOWN [SourceAv (19/22)]
[00008] main:id:000015,sig:11,src:000320+000579,time:700067,execs:2016338,op:splice,rep:8: UNKNOWN [SourceAv (19/22)]
[00009] main:id:000017,sig:11,src:000159,time:717887,execs:2068222,op:havoc,rep:4: EXPLOITABLE [DestAv (8/22)]
[00010] main:id:000018,sig:11,src:000177+000348,time:736990,execs:2123871,op:splice,rep:16: UNKNOWN [SourceAv (19/22)]
[00011] main:id:000019,sig:11,src:000400,time:856907,execs:2461393,op:havoc,rep:2: UNKNOWN [SourceAv (19/22)]
[00012] main:id:000020,sig:11,src:000303,time:880441,execs:2529860,op:havoc,rep:8: EXPLOITABLE [DestAv (8/22)]
[00013] main:id:000022,sig:11,src:000171+000277,time:1004425,execs:2887506,op:splice,rep:2: EXPLOITABLE [DestAv (8/22)]
[00014] main:id:000023,sig:11,src:000153,time:1042027,execs:2993815,op:havoc,rep:2: EXPLOITABLE [DestAv (8/22)]
[00015] main:id:000024,sig:11,src:000242+000673,time:1058852,execs:3042779,op:splice,rep:16: UNKNOWN [SourceAv (19/22)]
[00016] main:id:000027,sig:11,src:000350+000217,time:1168959,execs:3360200,op:splice,rep:16: UNKNOWN [SourceAv (19/22)]
[00017] main:id:000029,sig:11,src:000282+000575,time:1197741,execs:3443961,op:splice,rep:2: UNKNOWN [SourceAv (19/22)]
[00018] main:id:000030,sig:11,src:000401+000678,time:1201448,execs:3454757,op:splice,rep:4: UNKNOWN [SourceAv (19/22)]
[00019] main:id:000031,sig:11,src:000166+000720,time:1335254,execs:3841763,op:splice,rep:8: UNKNOWN [SourceAv (19/22)]
[00020] main:id:000032,sig:11,src:000456,time:1387044,execs:3989641,op:havoc,rep:8: UNKNOWN [SourceAv (19/22)]
[00021] main:id:000033,sig:11,src:000225+000696,time:1501651,execs:4317411,op:splice,rep:2: UNKNOWN [SourceAv (19/22)]
[00022] main:id:000034,sig:11,src:000225+000696,time:1501652,execs:4317416,op:splice,rep:2: UNKNOWN [SourceAv (19/22)]
[00023] main:id:000035,sig:11,src:000225+000696,time:1501663,execs:4317447,op:splice,rep:4: UNKNOWN [SourceAv (19/22)]
[00024] main:id:000036,sig:11,src:000272+000696,time:1576580,execs:4529676,op:splice,rep:8: UNKNOWN [SourceAv (19/22)]
[00025] main:id:000038,sig:11,src:000668+000696,time:1869383,execs:5376500,op:splice,rep:2: UNKNOWN [SourceAv (19/22)]
[00026] main:id:000039,sig:11,src:000242+000503,time:1872286,execs:5384956,op:splice,rep:4: UNKNOWN [SourceAv (19/22)]
```

Fuzzing uBPF JIT



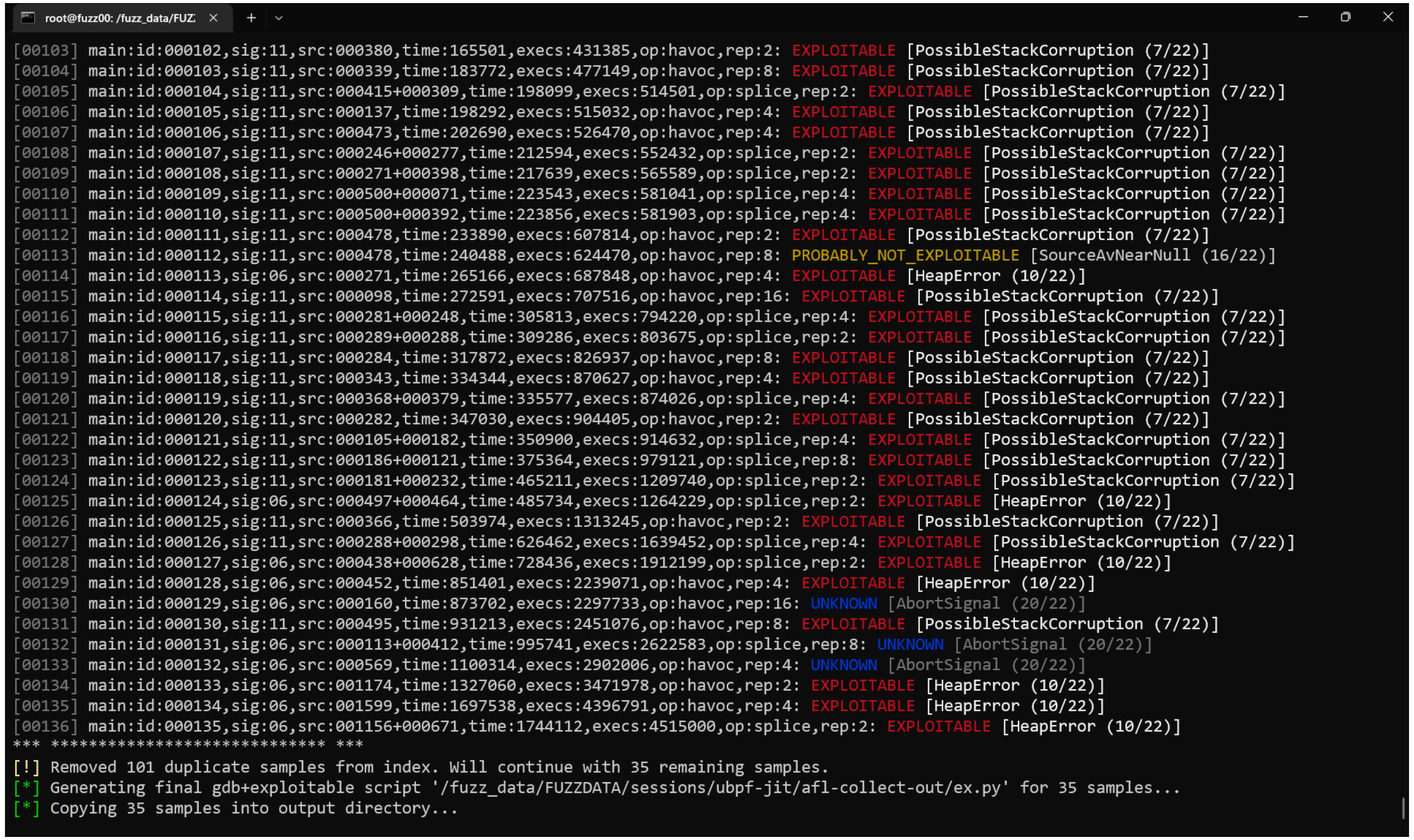
```
             american fuzzy lop ++3.01a (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 59 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec       │  total paths : 4027  │
│ last uniq crash : 0 days, 0 hrs, 0 min, 9 sec       │ uniq crashes : 46    │
│  last uniq hang : none seen yet                     │   uniq hangs : 0     │
├─ cycle progress ────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 1328.1 (33.0%) │    map density : 3.71% / 99.90%       │
│ paths timed out : 0 (0.00%)     │ count coverage : 4.53 bits/tuple      │
├─ stage progress ────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc             │ favored paths : 740 (18.38%)          │
│ stage execs : 22.1k/32.8k (67.55%) │  new edges on : 878 (21.80%)       │
│ total execs : 95.6k             │ total crashes : 3747 (46 unique)      │
│  exec speed : 1188/sec          │  total tmouts : 0 (0 unique)          │
├─ fuzzing strategy yields ───────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                      │    levels : 10         │
│  byte flips : n/a, n/a, n/a                      │   pending : 4026       │
│ arithmetics : n/a, n/a, n/a                      │  pend fav : 740        │
│  known ints : n/a, n/a, n/a                      │ own finds : 282        │
│  dictionary : n/a, n/a, n/a                      │  imported : 0          │
│havoc/splice : 229/24.6k, 76/7680                 │ stability : 100.00%    │
│   py/custom : 0/0, 0/0                           ├────────────────────────┘
│        trim : 0.00%/2, n/a                       │          [cpu000: 75%]
└──────────────────────────────────────────────────┘^C
```

Fuzzing uBPF JIT



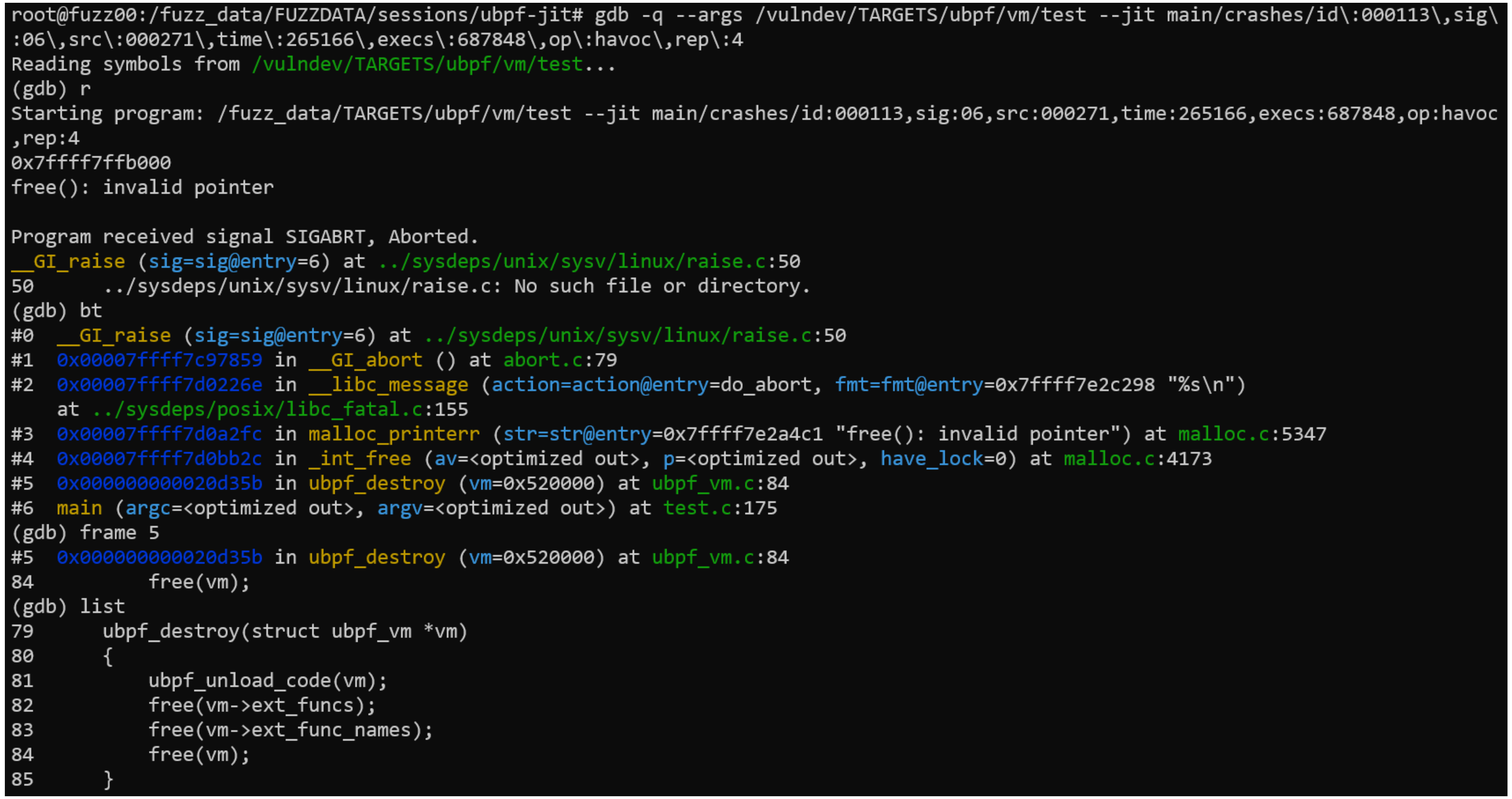
```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/ubpf-jit# afl-collect -r -e ex.py main afl-collect-out -- /vulndev/TARGETS/ubpf/vm/test --jit @@
afl-collect 1.35a by rc0r <hlt99@blinkenshell.org> # @_rc0r
Crash sample collection and processing utility for afl-fuzz.

[*] Going to collect crash samples from '/fuzz_data/FUZZDATA/sessions/ubpf-jit/main'.
[*] Found 1 fuzzers, collecting crash samples.
[*] Successfully indexed 408 crash samples.
[!] Removed 272 invalid crash samples from index.
[!] Removed 0 timed out samples from index.
[*] Generating intermediate gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/ubpf-jit/afl-collect-out/ex.py.0' for 136 samples...
[*] Executing gdb+exploitable script 'ex.py.0'...
*** GDB+EXPLOITABLE SCRIPT OUTPUT ***
[00001] main:id:000000,sig:11,src:000000,time:1846,execs:5366,op:havoc,rep:16: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00002] main:id:000001,sig:11,src:000000+000071,time:11606,execs:33756,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00003] main:id:000002,sig:11,src:000014,time:11869,execs:34461,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00004] main:id:000003,sig:11,src:000014,time:11940,execs:34654,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00005] main:id:000004,sig:11,src:000014,time:12120,execs:35135,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00006] main:id:000005,sig:11,src:000014,time:12344,execs:35730,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00007] main:id:000006,sig:11,src:000014,time:12499,execs:36165,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00008] main:id:000007,sig:11,src:000014,time:12534,execs:36254,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00009] main:id:000008,sig:11,src:000014,time:12745,execs:36822,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00010] main:id:000009,sig:11,src:000014,time:12768,execs:36887,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00011] main:id:000010,sig:11,src:000014,time:12890,execs:37230,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00012] main:id:000011,sig:11,src:000014,time:12983,execs:37466,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00013] main:id:000012,sig:11,src:000014,time:12983,execs:37467,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00014] main:id:000013,sig:11,src:000014,time:13178,execs:38020,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00015] main:id:000014,sig:11,src:000014,time:13349,execs:38489,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00016] main:id:000015,sig:11,src:000014,time:13391,execs:38610,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00017] main:id:000016,sig:11,src:000014,time:14029,execs:40387,op:havoc,rep:16: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00018] main:id:000017,sig:11,src:000014,time:14984,execs:43064,op:havoc,rep:16: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00019] main:id:000018,sig:11,src:000014,time:18092,execs:51824,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00020] main:id:000019,sig:11,src:000072,time:21414,execs:61032,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00021] main:id:000020,sig:11,src:000072,time:22968,execs:64856,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00022] main:id:000021,sig:11,src:000072,time:23589,execs:66390,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00023] main:id:000022,sig:11,src:000072,time:24202,execs:67889,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00024] main:id:000023,sig:11,src:000072,time:24391,execs:68365,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00025] main:id:000024,sig:11,src:000072,time:25958,execs:72217,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00026] main:id:000025,sig:11,src:000072+000156,time:28660,execs:78859,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
```

Fuzzing uBPF JIT



```
[00103] main:id:000102,sig:11,src:000380,time:165501,execs:431385,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00104] main:id:000103,sig:11,src:000339,time:183772,execs:477149,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00105] main:id:000104,sig:11,src:000415+000309,time:198099,execs:514501,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00106] main:id:000105,sig:11,src:000137,time:198292,execs:515032,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00107] main:id:000106,sig:11,src:000473,time:202690,execs:526470,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00108] main:id:000107,sig:11,src:000246+000277,time:212594,execs:552432,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00109] main:id:000108,sig:11,src:000271+000398,time:217639,execs:565589,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00110] main:id:000109,sig:11,src:000500+000071,time:223543,execs:581041,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00111] main:id:000110,sig:11,src:000500+000392,time:223856,execs:581903,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00112] main:id:000111,sig:11,src:000478,time:233890,execs:607814,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00113] main:id:000112,sig:11,src:000478,time:240488,execs:624470,op:havoc,rep:8: PROBABLY_NOT_EXPLOITABLE [SourceAvNearNull (16/22)]
[00114] main:id:000113,sig:06,src:000271,time:265166,execs:687848,op:havoc,rep:4: EXPLOITABLE [HeapError (10/22)]
[00115] main:id:000114,sig:11,src:000098,time:272591,execs:707516,op:havoc,rep:16: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00116] main:id:000115,sig:11,src:000281+000248,time:305813,execs:794220,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00117] main:id:000116,sig:11,src:000289+000288,time:309286,execs:803675,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00118] main:id:000117,sig:11,src:000284,time:317872,execs:826937,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00119] main:id:000118,sig:11,src:000343,time:334344,execs:870627,op:havoc,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00120] main:id:000119,sig:11,src:000368+000379,time:335577,execs:874026,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00121] main:id:000120,sig:11,src:000282,time:347030,execs:904405,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00122] main:id:000121,sig:11,src:000105+000182,time:350900,execs:914632,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00123] main:id:000122,sig:11,src:000186+000121,time:375364,execs:979121,op:splice,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00124] main:id:000123,sig:11,src:000181+000232,time:465211,execs:1209740,op:splice,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00125] main:id:000124,sig:06,src:000497+000464,time:485734,execs:1264229,op:splice,rep:2: EXPLOITABLE [HeapError (10/22)]
[00126] main:id:000125,sig:11,src:000366,time:503974,execs:1313245,op:havoc,rep:2: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00127] main:id:000126,sig:11,src:000288+000298,time:626462,execs:1639452,op:splice,rep:4: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00128] main:id:000127,sig:06,src:000438+000628,time:728436,execs:1912199,op:splice,rep:2: EXPLOITABLE [HeapError (10/22)]
[00129] main:id:000128,sig:06,src:000452,time:851401,execs:2239071,op:havoc,rep:4: EXPLOITABLE [HeapError (10/22)]
[00130] main:id:000129,sig:06,src:000160,time:873702,execs:2297733,op:havoc,rep:16: UNKNOWN [AbortSignal (20/22)]
[00131] main:id:000130,sig:11,src:000495,time:931213,execs:2451076,op:havoc,rep:8: EXPLOITABLE [PossibleStackCorruption (7/22)]
[00132] main:id:000131,sig:06,src:000113+000412,time:995741,execs:2622583,op:splice,rep:8: UNKNOWN [AbortSignal (20/22)]
[00133] main:id:000132,sig:06,src:000569,time:1100314,execs:2902006,op:havoc,rep:4: UNKNOWN [AbortSignal (20/22)]
[00134] main:id:000133,sig:06,src:001174,time:1327060,execs:3471978,op:havoc,rep:2: EXPLOITABLE [HeapError (10/22)]
[00135] main:id:000134,sig:06,src:001599,time:1697538,execs:4396791,op:havoc,rep:4: EXPLOITABLE [HeapError (10/22)]
[00136] main:id:000135,sig:06,src:001156+000671,time:1744112,execs:4515000,op:splice,rep:2: EXPLOITABLE [HeapError (10/22)]
*** ***************************** ***
[!] Removed 101 duplicate samples from index. Will continue with 35 remaining samples.
[*] Generating final gdb+exploitable script '/fuzz_data/FUZZDATA/sessions/ubpf-jit/afl-collect-out/ex.py' for 35 samples...
[*] Copying 35 samples into output directory...
```

Fuzzing uBPF JIT

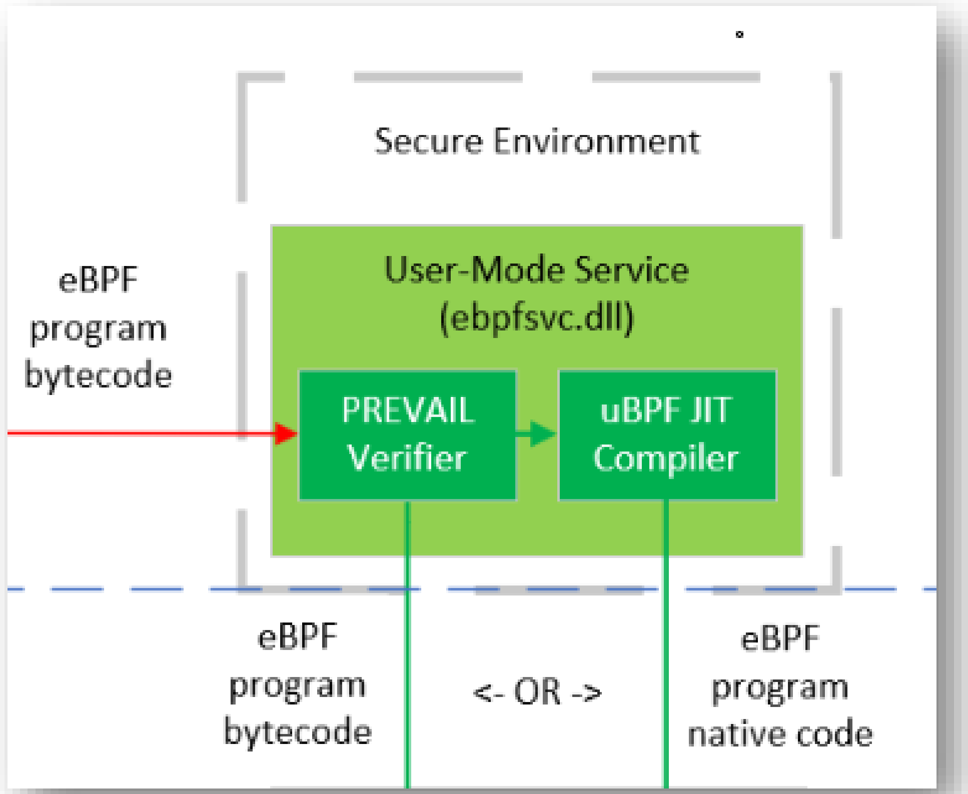


```
root@fuzz00:/fuzz_data/FUZZDATA/sessions/ubpf-jit# gdb -q --args /vulndev/TARGETS/ubpf/vm/test --jit main/crashes/id\:000113\,sig\
:06\,src\:000271\,time\:265166\,execs\:687848\,op\:havoc\,rep\:4
Reading symbols from /vulndev/TARGETS/ubpf/vm/test...
(gdb) r
Starting program: /fuzz_data/TARGETS/ubpf/vm/test --jit main/crashes/id:000113,sig:06,src:000271,time:265166,execs:687848,op:havoc
,rep:4
0x7ffff7ffb000
free(): invalid pointer

Program received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:50
50      ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
(gdb) bt
#0  __GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:50
#1  0x00007ffff7c97859 in __GI_abort () at abort.c:79
#2  0x00007ffff7d0226e in __libc_message (action=action@entry=do_abort, fmt=fmt@entry=0x7ffff7e2c298 "%s\n")
    at ../sysdeps/posix/libc_fatal.c:155
#3  0x00007ffff7d0a2fc in malloc_printerr (str=str@entry=0x7ffff7e2a4c1 "free(): invalid pointer") at malloc.c:5347
#4  0x00007ffff7d0bb2c in _int_free (av=<optimized out>, p=<optimized out>, have_lock=0) at malloc.c:4173
#5  0x000000000020d35b in ubpf_destroy (vm=0x520000) at ubpf_vm.c:84
#6  main (argc=<optimized out>, argv=<optimized out>) at test.c:175
(gdb) frame 5
#5  0x000000000020d35b in ubpf_destroy (vm=0x520000) at ubpf_vm.c:84
84          free(vm);
(gdb) list
79      ubpf_destroy(struct ubpf_vm *vm)
80      {
81          ubpf_unload_code(vm);
82          free(vm->ext_funcs);
83          free(vm->ext_func_names);
84          free(vm);
85      }
```

Fuzzing ebpfsvc.dll

Our initial attempts at fuzzing involved cross fuzzing using the pile of crashes we had found in the individual components but we were hitting crashes too early in the API

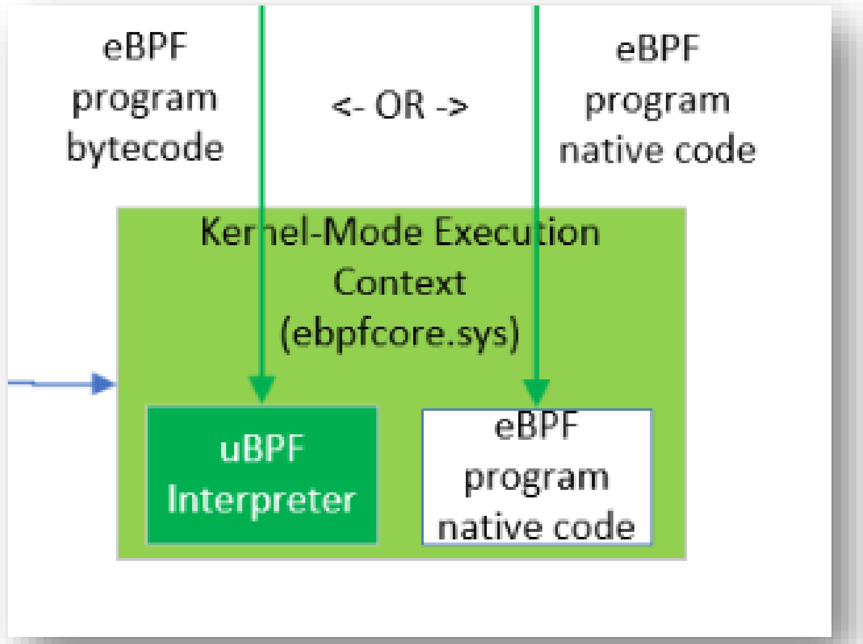
We began fuzzing with WTF but this coincided with the checkin of Microsoft's own libfuzzer harness for PREVAIL which found many of the same bugs so no new bugs were found



eBPF4Win Kernel (ebpfcore.sys)

In addition to the RPC interface exposed by the eBPFSvc the kernel module exposes a set of IOCTLs for manipulating programs and maps

Currently the ACL on the Device Object requires Administrator privileges so the impact is limited at this point in time, however this is meant to be proactive vulnerability analysis so we will fuzz the IOCTL layer



ebpfcore.sys IOCTL Interface

IOCTL Functions

0x0 resolve_helper	0x10 get_ec_function
0x1 resolve_map	0x11 get_program_info
0x2 create_program	0x12 get_pinned_map_info
0x3 create_map	0x13 get_link_handle_by_id
0x4 load_code	0x14 get_map_handle_by_id
0x5 map_find_element	0x15 get_program_handle_by_id
0x6 map_update_element	0x16 get_next_link_id
0x7 map_update_element_with_handle	0x17 get_next_map_id
0x8 map_delete_element	0x18 get_next_program_id
0x9 map_get_next_key	0x19 get_object_info
0xa query_program_info	0x1a get_next_pinned_program_path
0xb update_pinning	0x1b bind_map
0xc get_pinned_object	0x1c ring_buffer_map_query_buffer
0xd link_program	0x1d ring_buffer_map_async_query
0xe unlink_program	0x1e load_native_module
0xf close_handle	0x1f load_native_programs

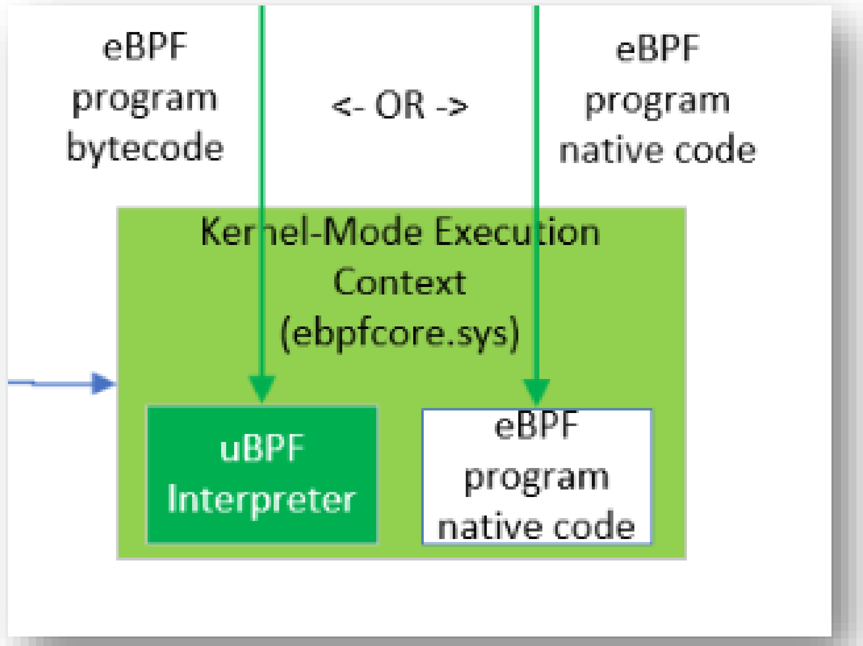
Fuzzing ebpfcore.sys

The majority of attack surface is available via fuzzing the IOCTL interface for ebpfcore.sys

To fuzz kernel attack surface a more sophisticated technique was used

Emulation and snapshot based fuzzing was used leveraging the WTF fuzzer tool from Axel Souchet

Multiple IOCTL requests can be sent in sequence between memory restoration from snapshot



Snapshot Fuzzing

An advanced fuzzing technique that uses emulators to continue code execution of a snapshot of a live system to allow researchers to fuzz specific areas of code.

Benefits:

- Allows researchers to create small and quick fuzzing loops in complex programs.
- Allows researchers to create large amounts of complexity in the program before fuzzing so that the fuzzer does not need to set up complexity.
- Allows researchers to fuzz "hard to reach" areas of code.

WTF Fuzzer

WTF Fuzzer

Advantages

- Distributed
- Code-Coverage Guided
- Customizable
- Cross Platform

Tradeoffs

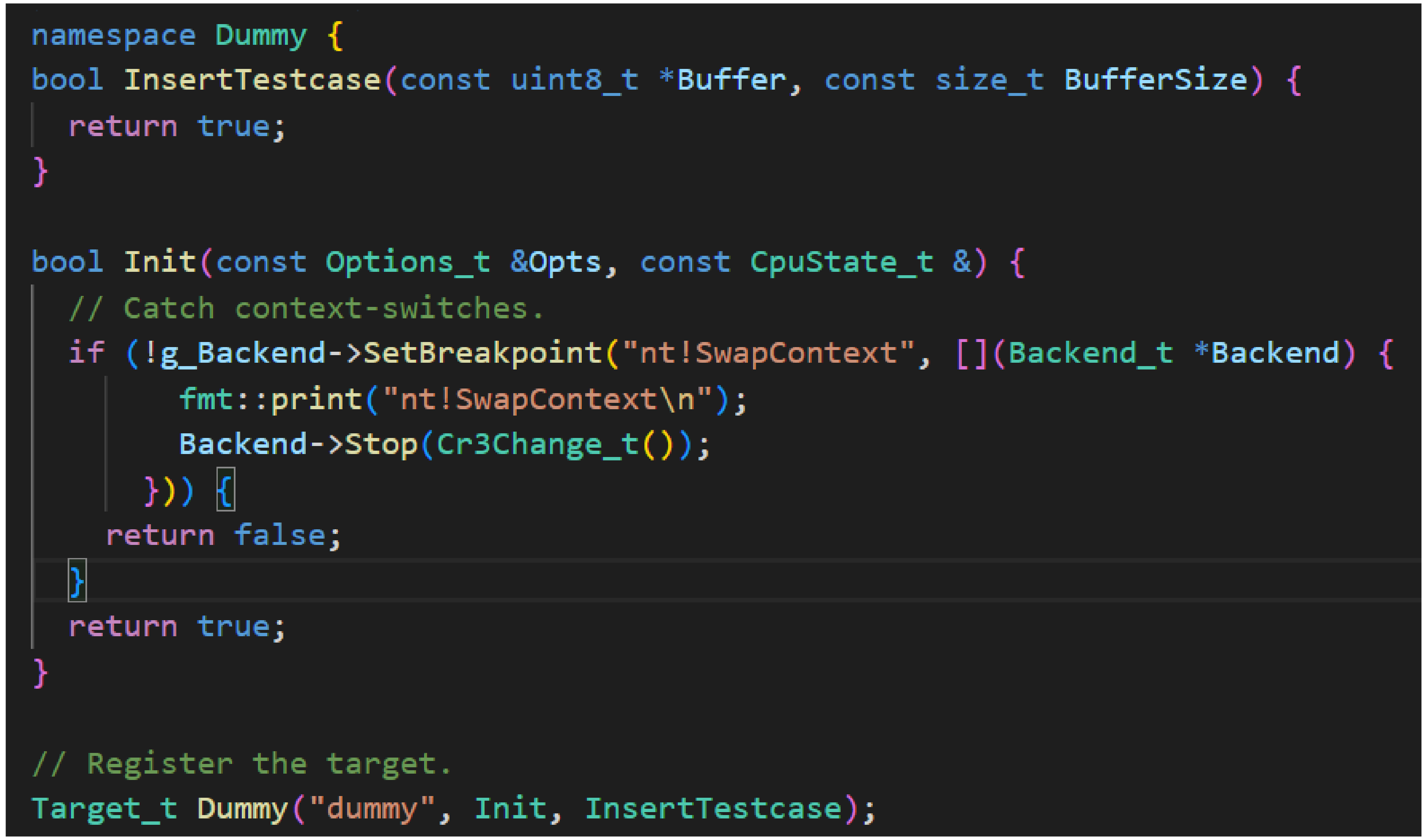
- Out of the box cannot handle:
  - Task Switching
  - Device IO
- Still in Development

WTF Fuzzer

To write a fuzzer with WTF, a few functions must be implemented

Init() sets up breakpoints in the emulator to handle events

InsertTestcase() is called with fuzzed data

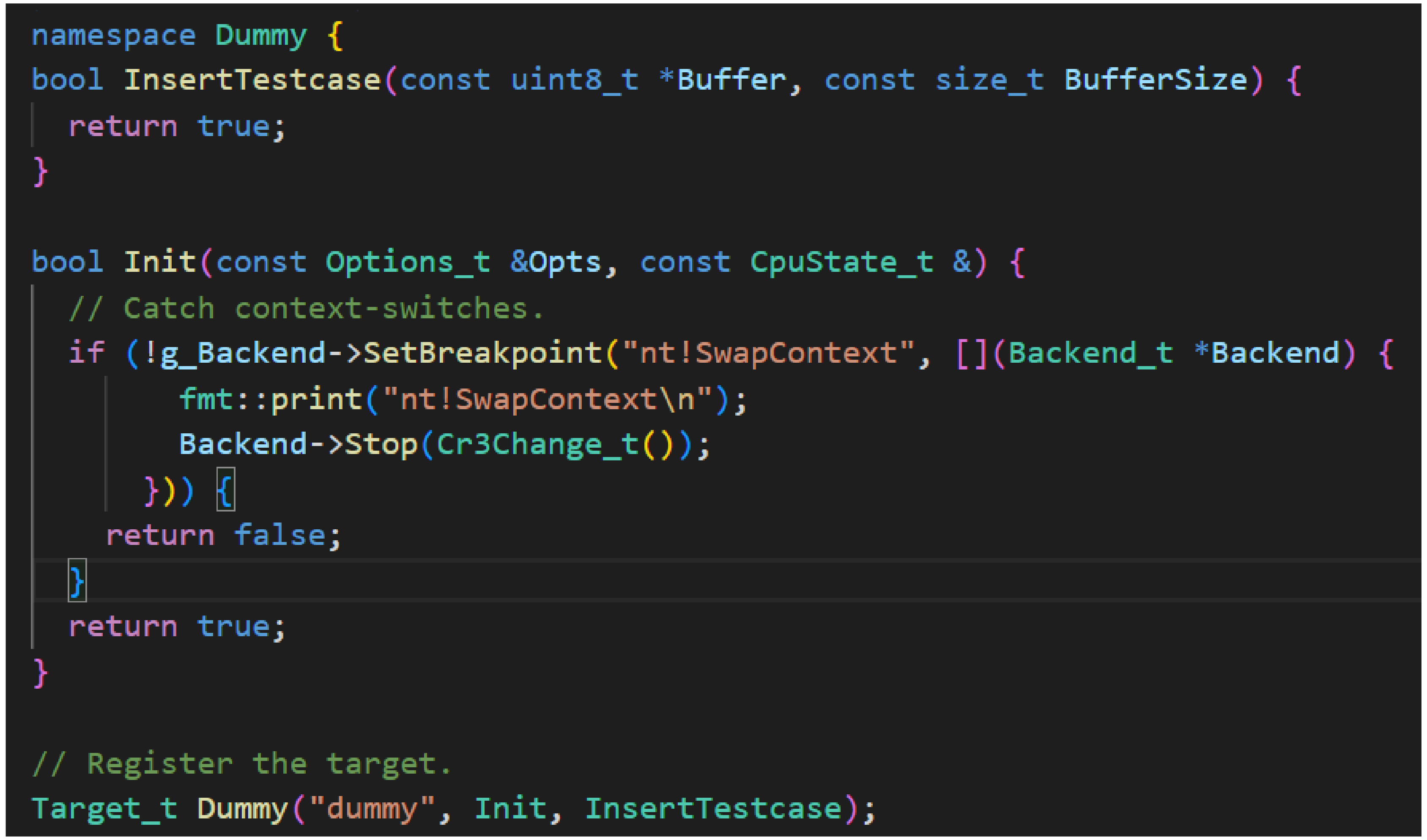


```
namespace Dummy {
bool InsertTestcase(const uint8_t *Buffer, const size_t BufferSize) {
  return true;
}

bool Init(const Options_t &Opts, const CpuState_t &) {
  // Catch context-switches.
  if (!g_Backend->SetBreakpoint("nt!SwapContext", [](Backend_t *Backend) {
        fmt::print("nt!SwapContext\n");
        Backend->Stop(Cr3Change_t());
      })) {
    return false;
  }
  return true;
}

// Register the target.
Target_t Dummy("dummy", Init, InsertTestcase);
```

WTF Fuzzer

There are also optional callbacks for custom data generators and the snapshot restore event

For multi-packet or IOCTL requests, the user implements a serialization format

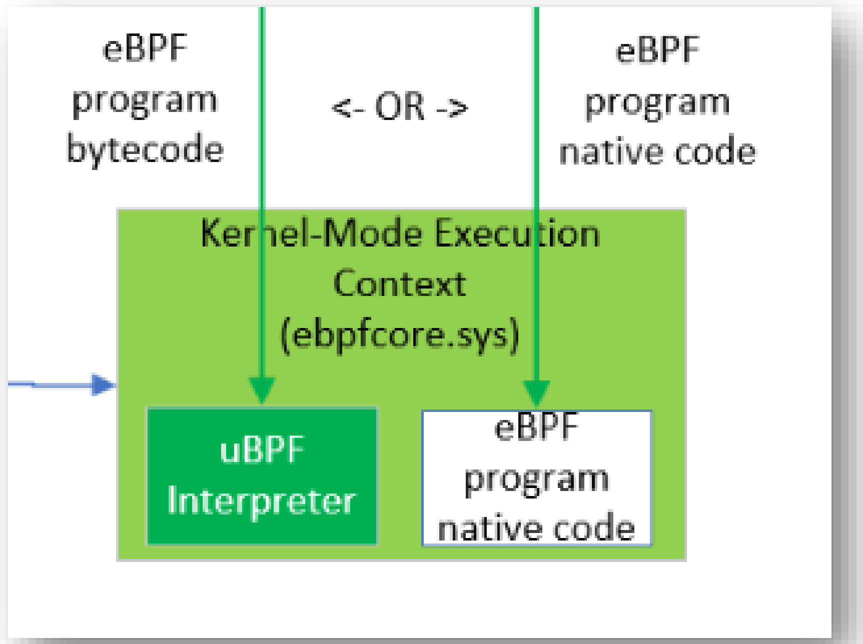


```
namespace Dummy {
bool InsertTestcase(const uint8_t *Buffer, const size_t BufferSize) {
  return true;
}

bool Init(const Options_t &Opts, const CpuState_t &) {
  // Catch context-switches.
  if (!g_Backend->SetBreakpoint("nt!SwapContext", [](Backend_t *Backend) {
        fmt::print("nt!SwapContext\n");
        Backend->Stop(Cr3Change_t());
      })) {
    return false;
  }
  return true;
}

// Register the target.
Target_t Dummy("dummy", Init, InsertTestcase);
```

WTF vs ebpfcore.sys

We created a harness based on the excellent tlv_server harness that is included with WTF. The original is designed to simulate sending multiple network packets to an interface.

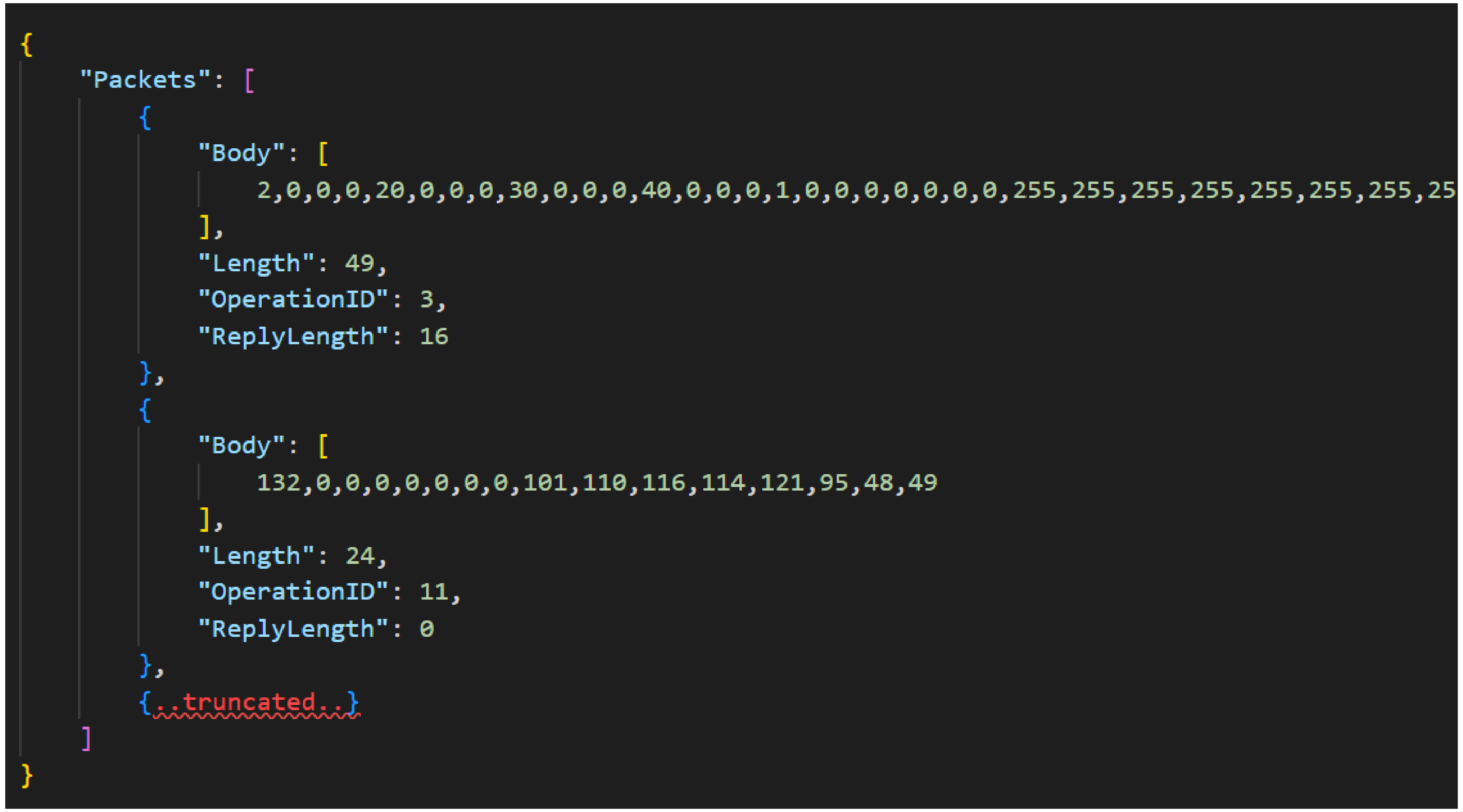
We forked this code and had it send IOCTL requests via DeviceIOControlFile calls instead



WTF vs ebpfcore.sys

For multi IOCTL requests we created a JSON based serialization format

The serialized testcase contains an array of requests that include the bytes of the data in the Body along with the Length, IOCTL OperationID, and expected ReplyLength



```
{
    "Packets": [
        {
            "Body": [
                2,0,0,0,20,0,0,0,30,0,0,0,40,0,0,0,1,0,0,0,0,0,0,0,255,255,255,255,255,255,255,25
            ],
            "Length": 49,
            "OperationID": 3,
            "ReplyLength": 16
        },
        {
            "Body": [
                132,0,0,0,0,0,0,0,101,110,116,114,121,95,48,49
            ],
            "Length": 24,
            "OperationID": 11,
            "ReplyLength": 0
        },
        {..truncated..}
    ]
}
```

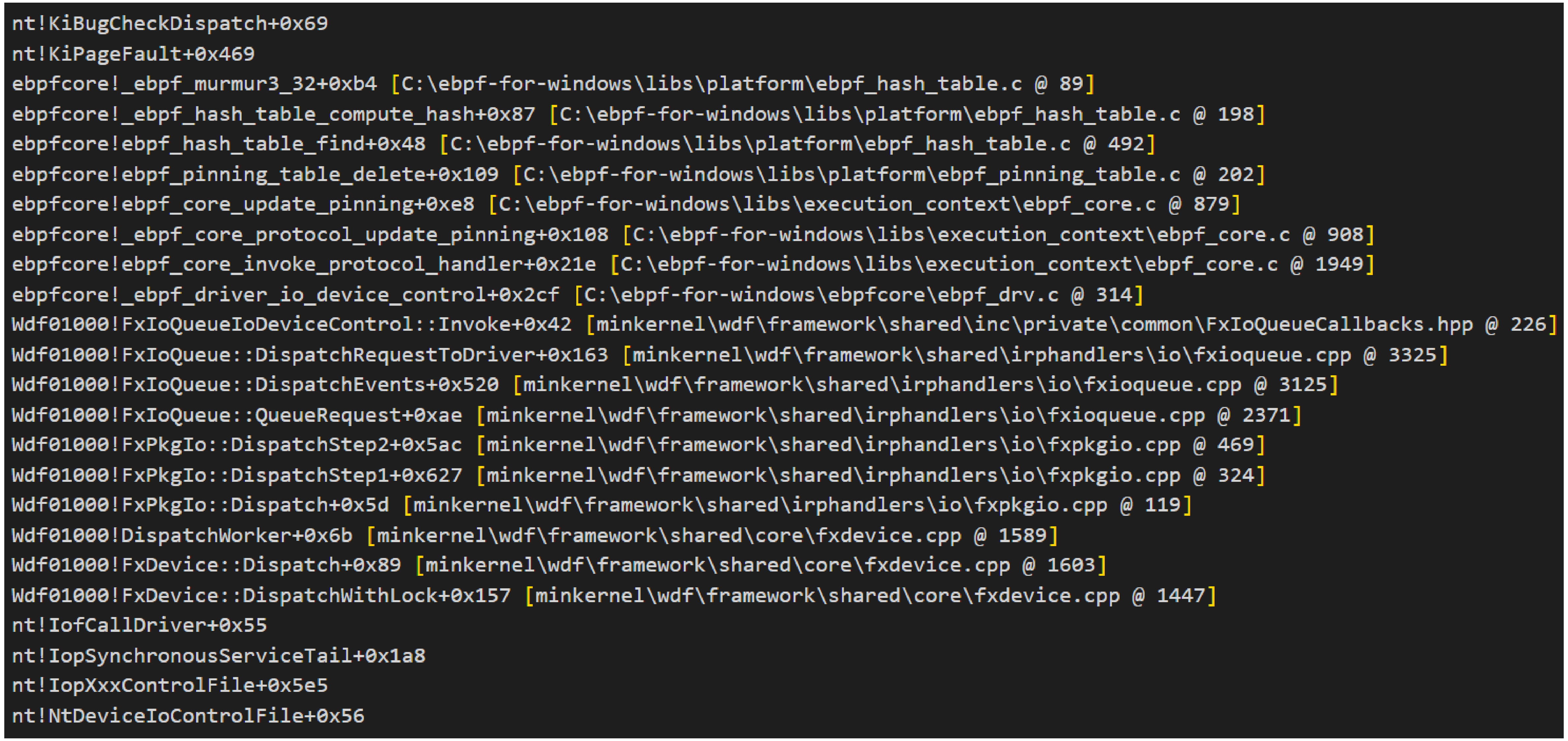
_ebpf_murmur3_32 Crash

Crash Type: Read Access Violation

Crash Cause:

- By setting the length in the packet header to a value less than the offset to the path in the packet struct you can underflow the length of the string struct created.
- The string is then passed into the ebpf_murmur function along with the length, at which point the loop inside the function will read past the end of the string and into memory it should not have access to.

_ebpf_murmur3_32 Crash



```
nt!KiBugCheckDispatch+0x69
nt!KiPageFault+0x469
ebpfcore!_ebpf_murmur3_32+0xb4 [C:\ebpf-for-windows\libs\platform\ebpf_hash_table.c @ 89]
ebpfcore!_ebpf_hash_table_compute_hash+0x87 [C:\ebpf-for-windows\libs\platform\ebpf_hash_table.c @ 198]
ebpfcore!ebpf_hash_table_find+0x48 [C:\ebpf-for-windows\libs\platform\ebpf_hash_table.c @ 492]
ebpfcore!ebpf_pinning_table_delete+0x109 [C:\ebpf-for-windows\libs\platform\ebpf_pinning_table.c @ 202]
ebpfcore!ebpf_core_update_pinning+0xe8 [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 879]
ebpfcore!_ebpf_core_protocol_update_pinning+0x108 [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 908]
ebpfcore!ebpf_core_invoke_protocol_handler+0x21e [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 1949]
ebpfcore!_ebpf_driver_io_device_control+0x2cf [C:\ebpf-for-windows\ebpfcore\ebpf_drv.c @ 314]
Wdf01000!FxIoQueueIoDeviceControl::Invoke+0x42 [minkernel\wdf\framework\shared\inc\private\common\FxIoQueueCallbacks.hpp @ 226]
Wdf01000!FxIoQueue::DispatchRequestToDriver+0x163 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3325]
Wdf01000!FxIoQueue::DispatchEvents+0x520 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3125]
Wdf01000!FxIoQueue::QueueRequest+0xae [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 2371]
Wdf01000!FxPkgIo::DispatchStep2+0x5ac [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 469]
Wdf01000!FxPkgIo::DispatchStep1+0x627 [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 324]
Wdf01000!FxPkgIo::Dispatch+0x5d [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 119]
Wdf01000!DispatchWorker+0x6b [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1589]
Wdf01000!FxDevice::Dispatch+0x89 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1603]
Wdf01000!FxDevice::DispatchWithLock+0x157 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1447]
nt!IofCallDriver+0x55
nt!IopSynchronousServiceTail+0x1a8
nt!IopXxxControlFile+0x5e5
nt!NtDeviceIoControlFile+0x56
```

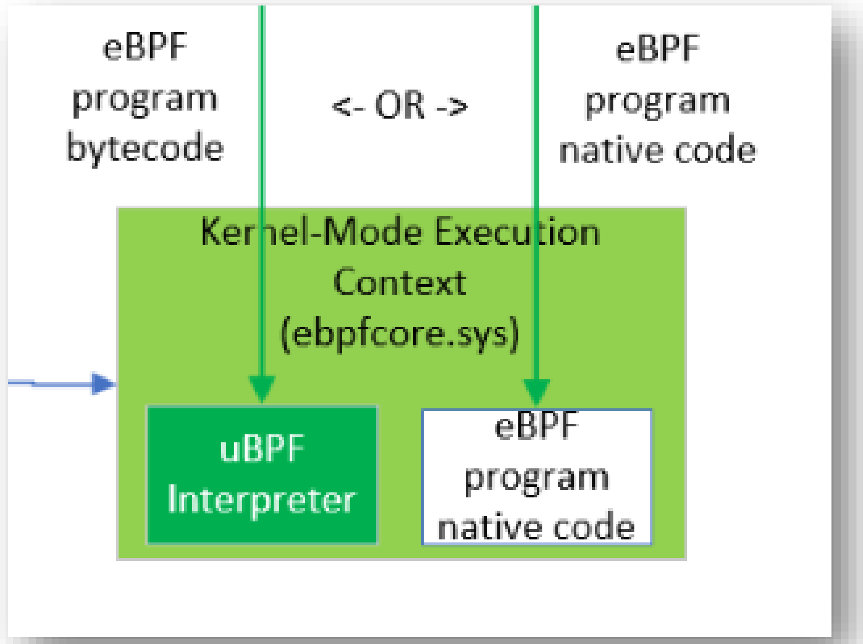
ubpf_destroy Crashes

Crash Type: Null Pointer Dereference

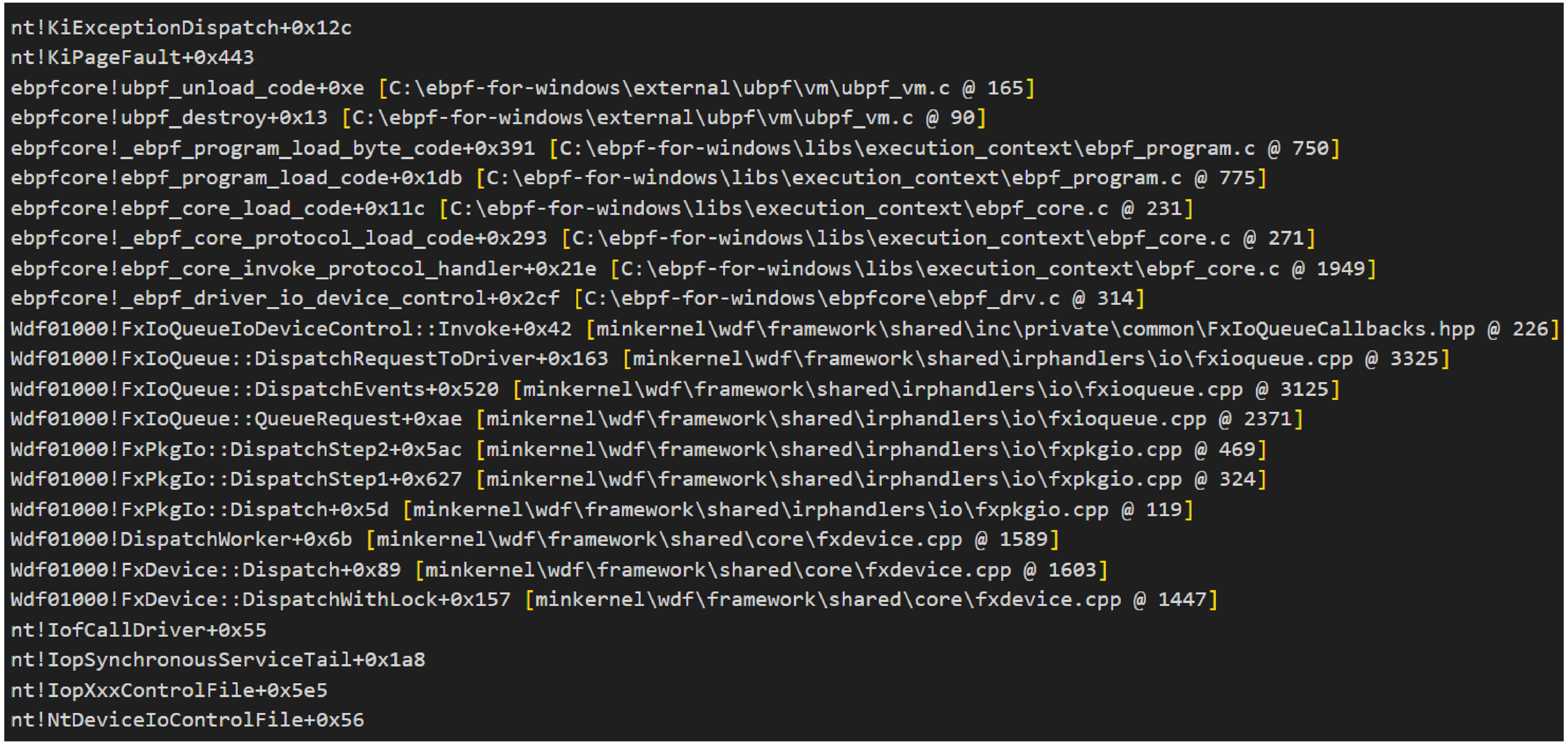
Crash Cause:

- ubpf_create runs out of memory while trying to calloc space for structs due to memory exhaustion.
- The function fails and returns a null value for the vm which is then passed into ubpf_destroy causing different null pointer dereferences depending on when the program ran out of memory.

- Note: multiple unique variations were found



ubpf_destroy Crashes



```
nt!KiExceptionDispatch+0x12c
nt!KiPageFault+0x443
ebpfcore!ubpf_unload_code+0xe [C:\ebpf-for-windows\external\ubpf\vm\ubpf_vm.c @ 165]
ebpfcore!ubpf_destroy+0x13 [C:\ebpf-for-windows\external\ubpf\vm\ubpf_vm.c @ 90]
ebpfcore!_ebpf_program_load_byte_code+0x391 [C:\ebpf-for-windows\libs\execution_context\ebpf_program.c @ 750]
ebpfcore!ebpf_program_load_code+0x1db [C:\ebpf-for-windows\libs\execution_context\ebpf_program.c @ 775]
ebpfcore!ebpf_core_load_code+0x11c [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 231]
ebpfcore!_ebpf_core_protocol_load_code+0x293 [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 271]
ebpfcore!ebpf_core_invoke_protocol_handler+0x21e [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 1949]
ebpfcore!_ebpf_driver_io_device_control+0x2cf [C:\ebpf-for-windows\ebpfcore\ebpf_drv.c @ 314]
Wdf01000!FxIoQueueIoDeviceControl::Invoke+0x42 [minkernel\wdf\framework\shared\inc\private\common\FxIoQueueCallbacks.hpp @ 226]
Wdf01000!FxIoQueue::DispatchRequestToDriver+0x163 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3325]
Wdf01000!FxIoQueue::DispatchEvents+0x520 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3125]
Wdf01000!FxIoQueue::QueueRequest+0xae [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 2371]
Wdf01000!FxPkgIo::DispatchStep2+0x5ac [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 469]
Wdf01000!FxPkgIo::DispatchStep1+0x627 [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 324]
Wdf01000!FxPkgIo::Dispatch+0x5d [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 119]
Wdf01000!DispatchWorker+0x6b [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1589]
Wdf01000!FxDevice::Dispatch+0x89 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1603]
Wdf01000!FxDevice::DispatchWithLock+0x157 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1447]
nt!IofCallDriver+0x55
nt!IopSynchronousServiceTail+0x1a8
nt!IopXxxControlFile+0x5e5
nt!NtDeviceIoControlFile+0x56
```

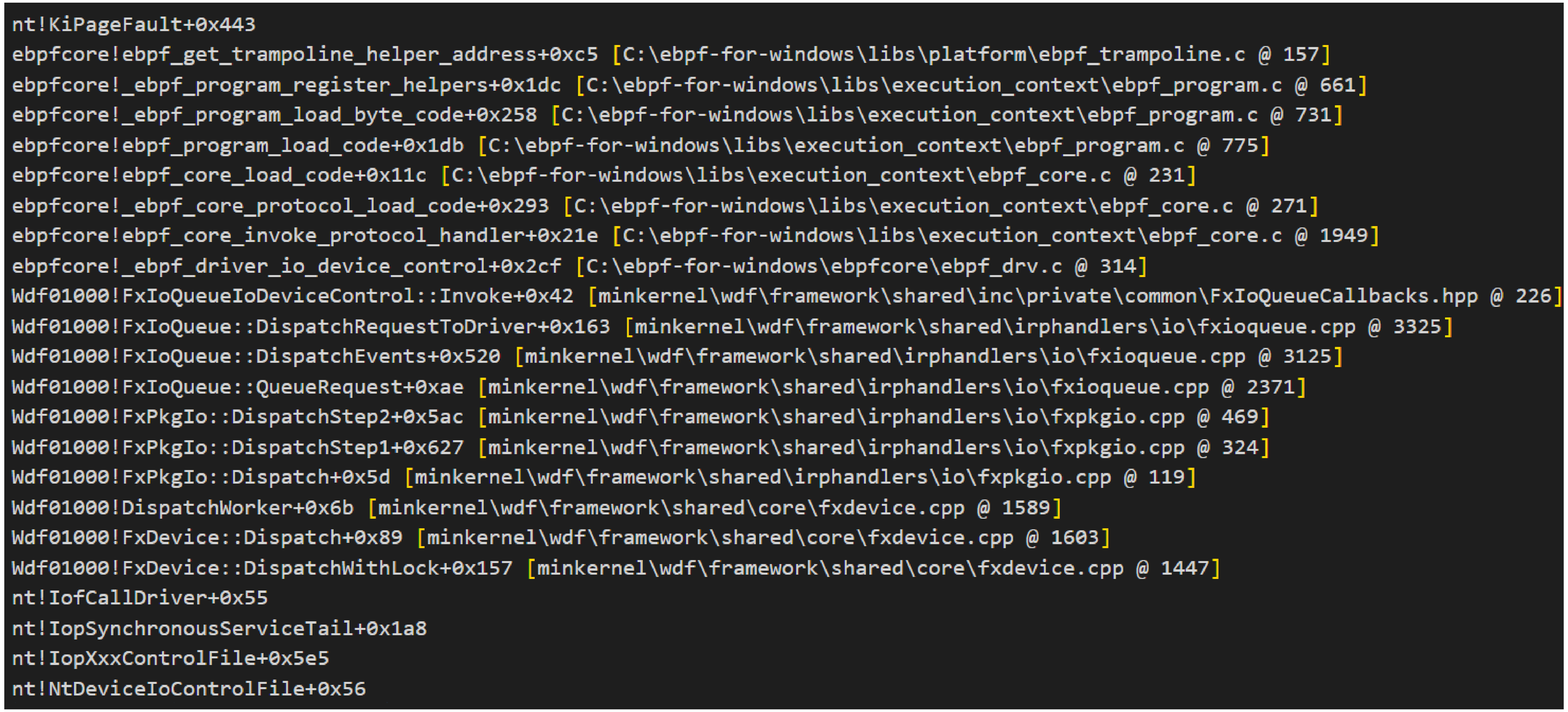
trampoline_table Crash

Crash Type: Null Pointer Dereference

Crash Cause:

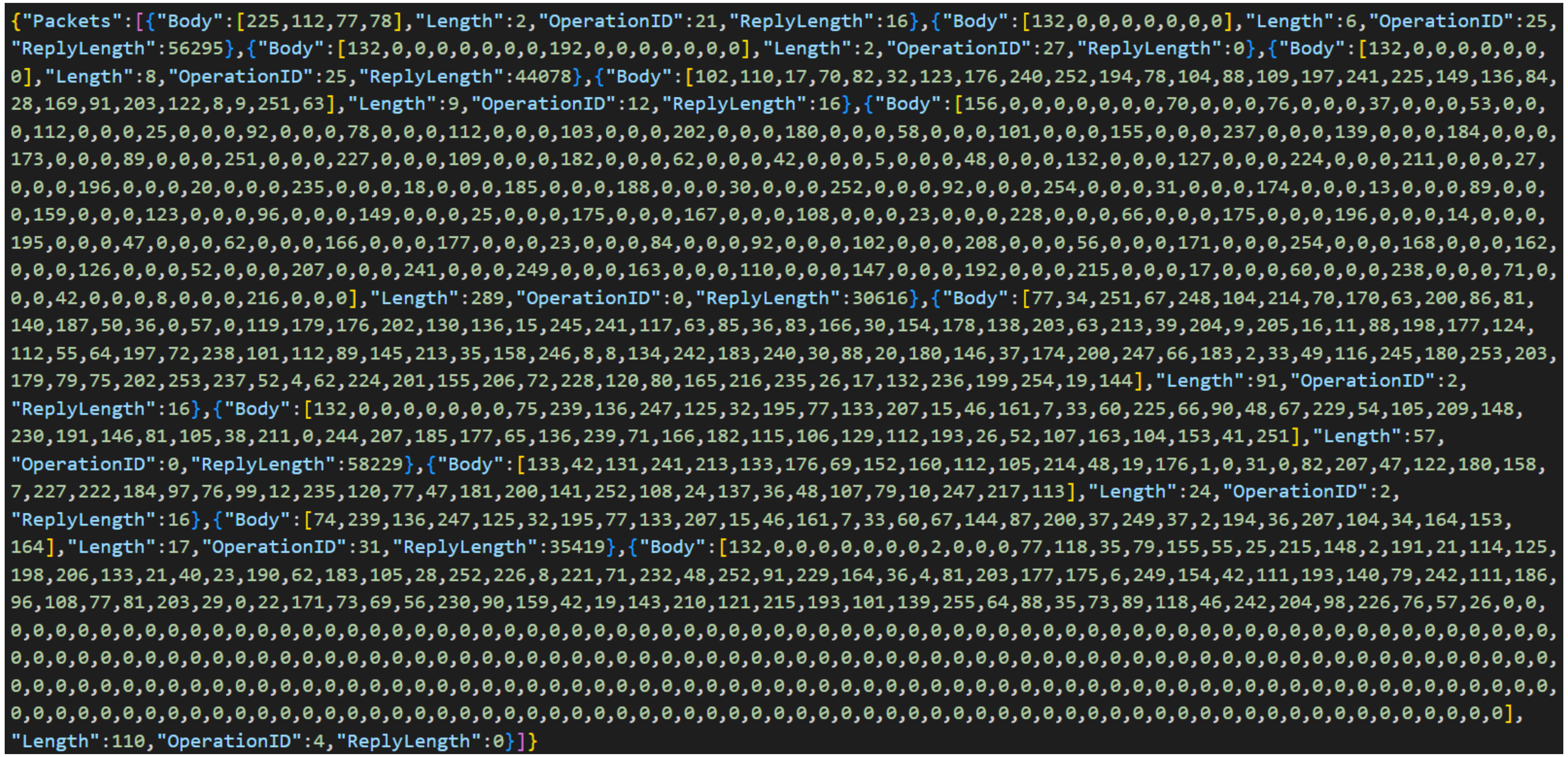
- When a program is created a callback is added to it which is trigger under certain conditions.
- If a resolve helper call is done on the program the callback is triggered, however, if the resolve helper function fails then the trampoline_table can become null.
- If the user then tries to load code the program will crash due to a null dereference.

trampoline_table Crash



```
nt!KiPageFault+0x443
ebpfcore!ebpf_get_trampoline_helper_address+0xc5 [C:\ebpf-for-windows\libs\platform\ebpf_trampoline.c @ 157]
ebpfcore!_ebpf_program_register_helpers+0x1dc [C:\ebpf-for-windows\libs\execution_context\ebpf_program.c @ 661]
ebpfcore!_ebpf_program_load_byte_code+0x258 [C:\ebpf-for-windows\libs\execution_context\ebpf_program.c @ 731]
ebpfcore!ebpf_program_load_code+0x1db [C:\ebpf-for-windows\libs\execution_context\ebpf_program.c @ 775]
ebpfcore!ebpf_core_load_code+0x11c [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 231]
ebpfcore!_ebpf_core_protocol_load_code+0x293 [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 271]
ebpfcore!ebpf_core_invoke_protocol_handler+0x21e [C:\ebpf-for-windows\libs\execution_context\ebpf_core.c @ 1949]
ebpfcore!_ebpf_driver_io_device_control+0x2cf [C:\ebpf-for-windows\ebpfcore\ebpf_drv.c @ 314]
Wdf01000!FxIoQueueIoDeviceControl::Invoke+0x42 [minkernel\wdf\framework\shared\inc\private\common\FxIoQueueCallbacks.hpp @ 226]
Wdf01000!FxIoQueue::DispatchRequestToDriver+0x163 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3325]
Wdf01000!FxIoQueue::DispatchEvents+0x520 [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 3125]
Wdf01000!FxIoQueue::QueueRequest+0xae [minkernel\wdf\framework\shared\irphandlers\io\fxioqueue.cpp @ 2371]
Wdf01000!FxPkgIo::DispatchStep2+0x5ac [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 469]
Wdf01000!FxPkgIo::DispatchStep1+0x627 [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 324]
Wdf01000!FxPkgIo::Dispatch+0x5d [minkernel\wdf\framework\shared\irphandlers\io\fxpkgio.cpp @ 119]
Wdf01000!DispatchWorker+0x6b [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1589]
Wdf01000!FxDevice::Dispatch+0x89 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1603]
Wdf01000!FxDevice::DispatchWithLock+0x157 [minkernel\wdf\framework\shared\core\fxdevice.cpp @ 1447]
nt!IofCallDriver+0x55
nt!IopSynchronousServiceTail+0x1a8
nt!IopXxxControlFile+0x5e5
nt!NtDeviceIoControlFile+0x56
```

trampoline_table Crash



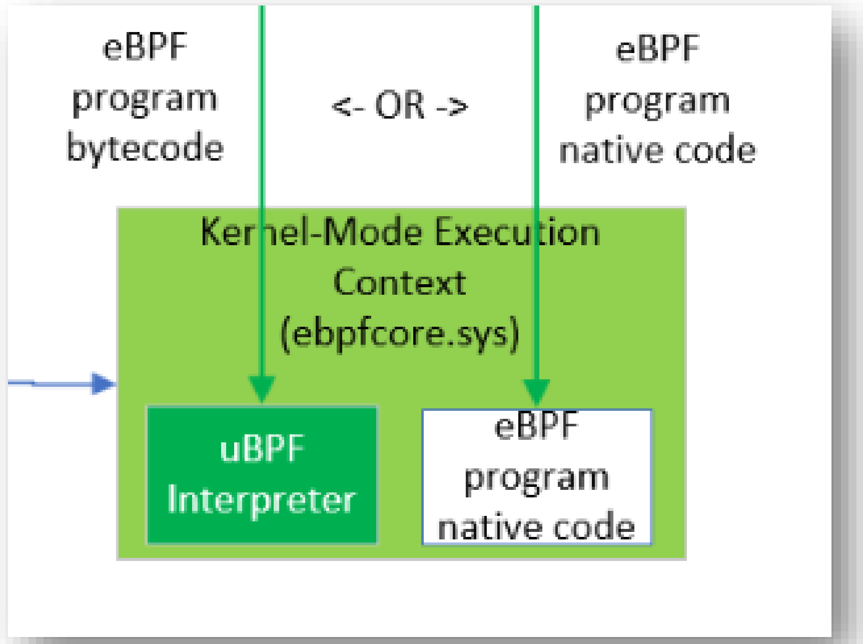
```
{"Packets":[{"Body":[225,112,77,78],"Length":2,"OperationID":21,"ReplyLength":16},{"Body":[132,0,0,0,0,0,0,0],"Length":6,"OperationID":25,
"ReplyLength":56295},{"Body":[132,0,0,0,0,0,0,0,192,0,0,0,0,0,0,0],"Length":2,"OperationID":27,"ReplyLength":0},{"Body":[132,0,0,0,0,0,0,
0],"Length":8,"OperationID":25,"ReplyLength":44078},{"Body":[102,110,17,70,82,32,123,176,240,252,194,78,104,88,109,197,241,225,149,136,84,
28,169,91,203,122,8,9,251,63],"Length":9,"OperationID":12,"ReplyLength":16},{"Body":[156,0,0,0,0,0,0,0,70,0,0,0,76,0,0,0,37,0,0,0,53,0,0,
0,112,0,0,0,25,0,0,0,92,0,0,0,78,0,0,0,112,0,0,0,103,0,0,0,202,0,0,0,180,0,0,0,58,0,0,0,101,0,0,0,155,0,0,0,237,0,0,0,139,0,0,0,184,0,0,0,
173,0,0,0,89,0,0,0,251,0,0,0,227,0,0,0,109,0,0,0,182,0,0,0,62,0,0,0,42,0,0,0,5,0,0,0,48,0,0,0,132,0,0,0,127,0,0,0,224,0,0,0,211,0,0,0,27,
0,0,0,196,0,0,0,20,0,0,0,235,0,0,0,18,0,0,0,185,0,0,0,188,0,0,0,30,0,0,0,252,0,0,0,92,0,0,0,254,0,0,0,31,0,0,0,174,0,0,0,13,0,0,0,89,0,0,
0,159,0,0,0,123,0,0,0,96,0,0,0,149,0,0,0,25,0,0,0,175,0,0,0,167,0,0,0,108,0,0,0,23,0,0,0,228,0,0,0,66,0,0,0,175,0,0,0,196,0,0,0,14,0,0,0,
195,0,0,0,47,0,0,0,62,0,0,0,166,0,0,0,177,0,0,0,23,0,0,0,84,0,0,0,92,0,0,0,102,0,0,0,208,0,0,0,56,0,0,0,171,0,0,0,254,0,0,0,168,0,0,0,162,
0,0,0,126,0,0,0,52,0,0,0,207,0,0,0,241,0,0,0,249,0,0,0,163,0,0,0,110,0,0,0,147,0,0,0,192,0,0,0,215,0,0,0,17,0,0,0,60,0,0,0,238,0,0,0,71,0,
0,0,42,0,0,0,8,0,0,0,216,0,0,0],"Length":289,"OperationID":0,"ReplyLength":30616},{"Body":[77,34,251,67,248,104,214,70,170,63,200,86,81,
140,187,50,36,0,57,0,119,179,176,202,130,136,15,245,241,117,63,85,36,83,166,30,154,178,138,203,63,213,39,204,9,205,16,11,88,198,177,124,
112,55,64,197,72,238,101,112,89,145,213,35,158,246,8,8,134,242,183,240,30,88,20,180,146,37,174,200,247,66,183,2,33,49,116,245,180,253,203,
179,79,75,202,253,237,52,4,62,224,201,155,206,72,228,120,80,165,216,235,26,17,132,236,199,254,19,144],"Length":91,"OperationID":2,
"ReplyLength":16},{"Body":[132,0,0,0,0,0,0,0,75,239,136,247,125,32,195,77,133,207,15,46,161,7,33,60,225,66,90,48,67,229,54,105,209,148,
230,191,146,81,105,38,211,0,244,207,185,177,65,136,239,71,166,182,115,106,129,112,193,26,52,107,163,104,153,41,251],"Length":57,
"OperationID":0,"ReplyLength":58229},{"Body":[133,42,131,241,213,133,176,69,152,160,112,105,214,48,19,176,1,0,31,0,82,207,47,122,180,158,
7,227,222,184,97,76,99,12,235,120,77,47,181,200,141,252,108,24,137,36,48,107,79,10,247,217,113],"Length":24,"OperationID":2,
"ReplyLength":16},{"Body":[74,239,136,247,125,32,195,77,133,207,15,46,161,7,33,60,67,144,87,200,37,249,37,2,194,36,207,104,34,164,153,
164],"Length":17,"OperationID":31,"ReplyLength":35419},{"Body":[132,0,0,0,0,0,0,0,2,0,0,0,77,118,35,79,155,55,25,215,148,2,191,21,114,125,
198,206,133,21,40,23,190,62,183,105,28,252,226,8,221,71,232,48,252,91,229,164,36,4,81,203,177,175,6,249,154,42,111,193,140,79,242,111,186,
96,108,77,81,203,29,0,22,171,73,69,56,230,90,159,42,19,143,210,121,215,193,101,139,255,64,88,35,73,89,118,46,242,204,98,226,76,57,26,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
"Length":110,"OperationID":4,"ReplyLength":0}]}
```

AFL-NYX vs ebpfcore.sys

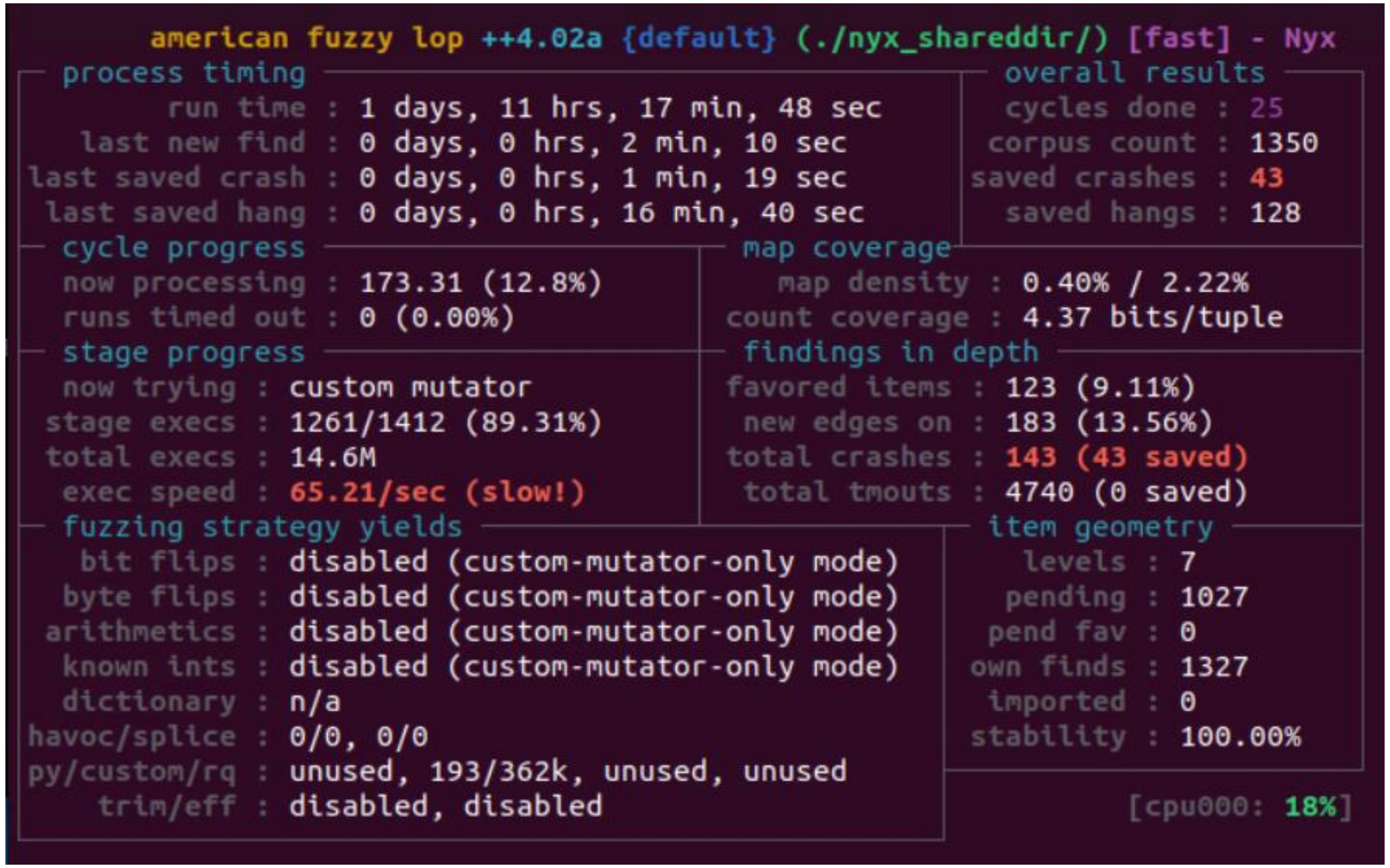
In addition to WTF, we also ported the same harness to the NYX hypervisor based snapshot fuzzer to assess capabilities and performance

NYX had significantly faster execution speed compared to WTF but did not find unique bugs due to the thoroughness of the initial fuzzer design

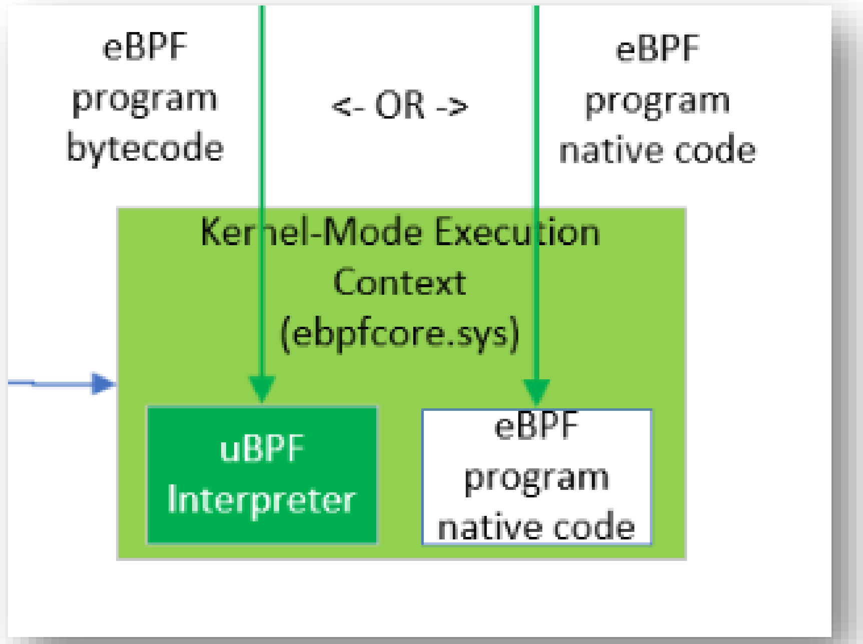
We did of course find similar bugs..



AFL-NYX vs ebpfcore.sys



```
        american fuzzy lop ++4.02a {default} (./nyx_shareddir/) [fast] - Nyx
┌─ process timing ─────────────────────────────────────┬─ overall results ───┐
│        run time : 1 days, 11 hrs, 17 min, 48 sec      │  cycles done : 25   │
│   last new find : 0 days, 0 hrs, 2 min, 10 sec        │ corpus count : 1350 │
│last saved crash : 0 days, 0 hrs, 1 min, 19 sec        │saved crashes : 43   │
│ last saved hang : 0 days, 0 hrs, 16 min, 40 sec       │  saved hangs : 128  │
├─ cycle progress ─────────────────────┬─ map coverage─┴─────────────────────┤
│  now processing : 173.31 (12.8%)      │    map density : 0.40% / 2.22%      │
│  runs timed out : 0 (0.00%)           │ count coverage : 4.37 bits/tuple    │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : custom mutator          │ favored items : 123 (9.11%)         │
│ stage execs : 1261/1412 (89.31%)      │  new edges on : 183 (13.56%)        │
│ total execs : 14.6M                   │ total crashes : 143 (43 saved)      │
│  exec speed : 65.21/sec (slow!)       │  total tmouts : 4740 (0 saved)      │
├─ fuzzing strategy yields ─────────────┴─────────────┬─ item geometry ───────┤
│   bit flips : disabled (custom-mutator-only mode)   │    levels : 7         │
│  byte flips : disabled (custom-mutator-only mode)   │   pending : 1027      │
│ arithmetics : disabled (custom-mutator-only mode)   │  pend fav : 0         │
│  known ints : disabled (custom-mutator-only mode)   │ own finds : 1327      │
│  dictionary : n/a                                   │  imported : 0         │
│havoc/splice : 0/0, 0/0                              │ stability : 100.00%   │
│py/custom/rq : unused, 193/362k, unused, unused      ├───────────────────────┘
│    trim/eff : disabled, disabled                    │          [cpu000: 18%]
└─────────────────────────────────────────────────────┘
```



eBPF4Win Kernel Extension Modules

eBPF for Windows is designed with a modular architecture on the kernel side

Instrumentation support is added to eBPF for Windows via “extension modules”

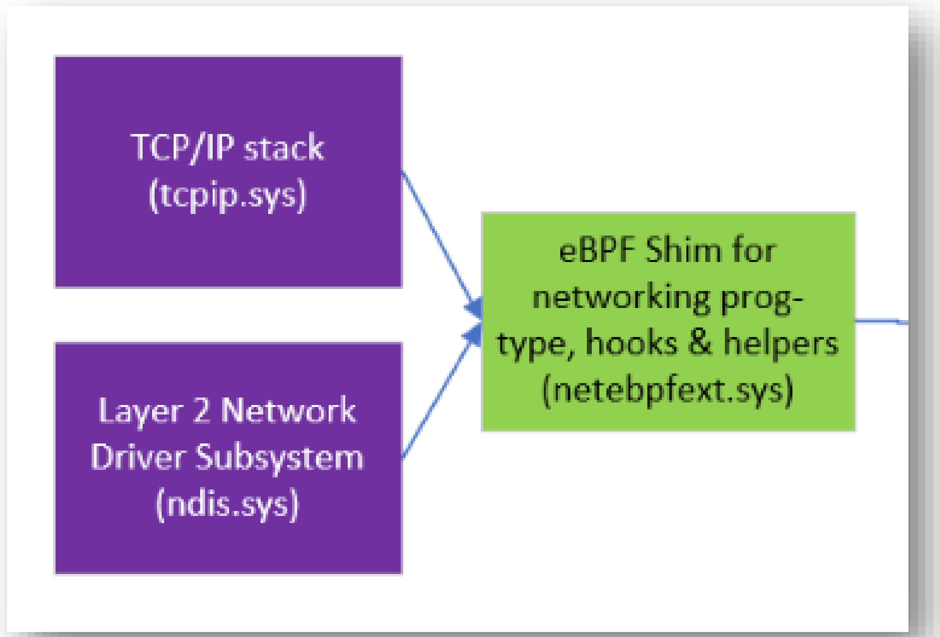
The current implementation provides a network shimming interface to allow for packet inspection and rewriting at multiple levels

eBPF4Win Network Shims (netebpfext.sys)

Microsoft is currently focused on observing and instrumenting network packets in the current eBPF implementation

Hook implementations can contain exploitable bugs that may be hard to detect

In this case we did a manual code review of the xdp, bind, and cgroup hooks and did not find any implementation errors.


TCP/IP stack (tcpip.sys)
Layer 2 Network Driver Subsystem (ndis.sys)
eBPF Shim for networking prog-type, hooks & helpers (netebpfext.sys)

eBPF4Win Code Hooks

On Linux, eBPF has strong integration with uprobe, kprobe, and tracepoint code hooking interfaces

Microsoft has libraries capable of providing similar code hooking abilities such as Detours

Currently code hooking is not supported via eBPF for Windows

An additional kernel extension module for code hooking can be added in the future to sit alongside netebpfext.sys

Concluding Thoughts

- eBPF is exciting technology for telemetry and instrumentation on modern operating systems
- Microsoft has adapted opensource projects uBPF and PREVAIL to provide the foundation for their eBPF implementation
- We found one serious ACE vulnerability and several robustness bugs during our fuzzing of the driver and userland loader code
- Microsoft has been quickly adding fuzz testing to their repo since May which has fixed many of the bugs found in the opensource projects
- With the creation of the eBPF foundation backed by several major industry players, eBPF is positioned to become a core technology for desktop, server, and cloud
- Trellix is committed to proactive vulnerability research to benefit the community

Thank you!

Richard Johnson, Trellix

@richinseattle on Twitter