Rapor 4 - Veri Boru Hattı Entegrasyonu

Ring Buffer (BPF_MAP_TYPE_RINGBUF) Mekanizmasının Implementasyonu

Raporun Amacı

Bu belge, "Bitirme Projesi Rapor-3: Temel Prototip" belgesinde elde edilen "Merhaba eBPF" prototipinin bir sonraki aşamasını belgelemektedir. Rapor-3'teki prototip, bpf_printk kullanarak bir execve olayının gerçekleştiğini yalnızca çekirdeğin izleme borusuna (trace_pipe) yazabilmekteydi.

Bu raporun amacı, "Rapor-1: Kritik Tasarım ve Planlama" belgesinin 5.1. Bölümünde tanımlanan 2. Hafta görevini (27 Ekim - 2 Kasım) detaylandırmaktır. Bu görev, bpf_printk hata ayıklama yöntemini terk ederek, "Rapor-1" Bölüm 3.3'te tasarlanan modern ve yüksek performanslı BPF_MAP_TYPE_RINGBUF (Ring Buffer) iletişim modelini projeye entegre etmektir.

Bu çalışma, çekirdek alanından (hello_kern.c) toplanan ham verinin (PID, komut adı, dosya yolu), analiz edilmek üzere kullanıcı alanı ajanına (hello_user.c) başarılı bir şekilde aktarılmasını ve Rapor-2'de kurulan geliştirme ortamının tüm yeteneklerinin doğrulamasını kapsar.

1. Mimari Geçiş: bpf_printk'ten BPF_MAP_TYPE_RINGBUF'a

Rapor-3 prototipi, veri aktarımı için tasarlanmamıştı. Projenin "davranışsal analiz" hedefine ulaşması için, çekirdek alanında toplanan verinin, analiz motorunun çalışacağı kullanıcı alanına aktarılması zorunludur.

Bu entegrasyon, Rapor-1'de tasarlanan iki bileşenli mimariyi (Çekirdek Veri Toplayıcı, Kullanıcı Analiz Motoru) fonksiyonel hale getiren kritik adımdır.

2. İmplementasyon Adımları

Proje mimarisinin her iki bileşeni ve bu bileşenler arasındaki "sözleşme" (contract), planlandığı gibi güncellenmiştir.

2.1. Adım 1: Paylaşılan Veri Sözleşmesinin Oluşturulması (common.h)

Çekirdek ve kullanıcı alanı arasında tutarlı veri aktarımı sağlamak için, her iki bileşenin de dahil edeceği common.h adında bir başlık dosyası oluşturulmuştur.

Bu dosya, Rapor-1, Bölüm 3.2'de hedeflenen "standart bir C struct yapısı içinde paketleme" görevini yerine getiren struct event yapısını tanımlar.

```
/* common.h */
#define TASK_COMM_LEN 16
#define MAX_FILENAME_LEN 256

/*
 * struct event
 * Çekirdekten kullanıcı alanına gönderilen veri kaydı.
 */
struct event {
    __u32 pid;                  // Süreç ID
    char comm[TASK_COMM_LEN];       // Süreç adı (örn: "bash")
    char filename[MAX_FILENAME_LEN]; // çalıştırılan dosya yolu
};
```

2.2. Adım 2: Çekirdek Alanı Güncellemesi (hello_kern.c)

hello_kern.c, Rapor-3'teki bpf_printk kullanan basit yapısından, Rapor-1'de tasarlanan "Veri Toplayıcı" rolüne terfi ettirilmiştir.

1. **Ring Buffer Haritası Tanımlandı:** BPF_MAP_TYPE_RINGBUF tipinde ve 256KB boyutunda rb adında bir eBPF haritası SEC(".maps") bölümünde tanımlandı.
2. **handle_execve_enter Fonksiyonu Güncellendi:**
    - bpf_printk çağrısı tamamen kaldırıldı.
    - bpf_ringbuf_reserve kullanılarak Ring Buffer'dan struct event boyutu kadar yer ayrıldı.
    - bpf_get_current_pid_tgid ve bpf_get_current_comm yardımcıları ile PID ve süreç adı alındı.
    - execve sistem çağrısının ilk argümanı (ctx->args[0]), bpf_probe_read_user_str ile güvenli bir şekilde okunarak çalıştırılan filename (dosya adı) elde edildi.
    - Doldurulan struct event yapısı, bpf_ringbuf_submit ile kullanıcı alanına gönderildi.

2.3. Adım 3: Kullanıcı Alanı Güncellemesi (hello_user.c)

hello_user.c ajanı, Rapor-3'teki sleep(1) ile pasif bekleme durumundan, Rapor-1, Bölüm 3.4'te tasarlanan "Olay Dinleyicisi" ve "Ayrıştırıcı/Loglayıcı" rollerini üstlenen aktif bir "Veri Tüketicisi"ne dönüştürülmüştür.

1. **Ring Buffer Yöneticisi Kurulumu:** hello_kern__attach başarıyla tamamlandıktan sonra, ring_buffer__new fonksiyonu çağrıldı. Bu fonksiyon, iskelet (skeleton) üzerinden erişilen skel->maps.rb haritasını, gelen verileri işleyecek olan handle_event adlı bir *callback* fonksiyonuna bağladı.

2. **handle_event Callback Fonksiyonu:** Gelen ham veriyi (void *data) alıp const struct event * yapısına dönüştüren (cast) bu fonksiyon, Rapor-1'deki "Ayrıştırıcı/Loglayıcı" rolünün ilk adımını gerçekleştirerek olay detaylarını printf ile standart çıktıya (stdout) yazdırdı.
3. **Ana Döngü Güncellemesi:** while (!exiting) döngüsü, sleep(1) yerine ring_buffer__poll(100) fonksiyonunu çağıracak şekilde güncellendi. Bu, ajanın her 100 milisaniyede bir Ring Buffer'ı aktif olarak kontrol etmesini (polling) ve yeni veri varsa handle_event fonksiyonunun tetiklenmesini sağladı.

4. Doğrulama ve Sonuç

sudo ile çalıştırılan ajan terminali (hello_ebpf) izlemeye alınmış ve Rapor-3'teki gibi ikinci bir terminalden ls gibi rastgele komutlar çalıştırılarak execve sistem çağrısı tetiklenmiştir.

Rapor-3'ün aksine, trace_pipe'ta bir çıktı gözlemlenmemiş; bunun yerine, hello_ebpf ajanının çalıştığı terminalde aşağıdaki gibi handle_event fonksiyonundan gelen printf çıktıları başarıyla gözlemlenmiştir:

```
Event received: PID=1759    COMM=bash          FILENAME=/usr/bin/ls
Event received: PID=1760    COMM=whoami        FILENAME=/usr/bin/whoami
```

Bu sonuç, "Rapor-1: Kritik Tasarım ve Planlama" belgesinin 5.1. Bölümündeki 2. Hafta (Veri Boru Hattı) hedefinin başarıyla tamamlandığını doğrulamaktadır. Prototip artık "olay oldu" demek yerine, "hangi olay (PID, komut, dosya adı) oldu" bilgisini kullanıcı alanına anlık olarak aktarabilmektedir.

5. Sonraki Adımlar

Veri boru hattı başarıyla kurulduğuna göre, proje Rapor-1, Bölüm 5.1'deki 3. Hafta (3-9 Kasım) görevine geçmeye hazırdır. Bu görev, gelen verileri sadece ekrana basmak yerine, Rapor-1, Bölüm 3.5'te tasarlanan "Durum Yönetimi" (State Management) mimarisini (uthash kütüphanesi ve process_stats yapısı) implemente etmeyi içermektedir.