Rapor 3 - Temel Prototip

"Merhaba eBPF" Prototipinin Geliştirilmesi

Raporun Amacı: Bu belge, "Bitirme Projesi Rapor-1" belgesinin 5.1. Bölümünde tanımlanan 1. Hafta görevlerinin devamı niteliğindedir. "Rapor-2: Geliştirme Ortamının Hazırlanması" belgesinde detaylandırılan CLion ve Uzak SSH geliştirme ortamı üzerinde, Rapor-1'de planlanan mimariye (libbpf ve bpftool) uygun ilk fonksiyonel eBPF prototipi geliştirilmiştir.

Bu rapor, projenin yapılandırılmasını, çekirdek ve kullanıcı alanı kodlarını, bu süreçte karşılaşılan teknik zorlukları ve doğrulama adımlarını belgelemektedir.

1. Proje Yapılandırması: CMakeLists.txt

Projenin modern libbpf ve bpftool gen skeleton akışına uygun olarak derlenebilmesi için esnek bir CMakeLists.txt dosyası oluşturulmuştur. Bu yapılandırma, projeyi iki ana bileşen olarak ele alır:

1. **eBPF Çekirdek Kodu (hello_kern.c):** clang derleyicisi ile -target bpf hedefi için derlenir.
2. **Kullanıcı Alanı Ajanı (hello_user.c):** Sistemin varsayılan C derleyicisi (GCC) ile derlenir ve libbpf kütüphanesine bağlanır.

Ana CMake Komutları:

- **Bağımlılıkların Bulunması:**
  - find_package(PkgConfig REQUIRED): Kütüphane yollarını bulmak için pkg-config modülü etkinleştirildi.
  - pkg_search_module(LIBBPF REQUIRED libbpf): Geliştirme ortamına kurulan libbpf-dev paketini bulur.
  - find_program(CLANG_EXECUTABLE clang REQUIRED): eBPF kodunu derleyecek clang derleyicisini bulur.
  - find_program(BPFTOOL_EXECUTABLE bpftool REQUIRED): İskelet (skeleton) oluşturmak için bpftool aracını bulur.
- **Özel Derleme Komutları (Custom Commands):**
  - add_custom_command(OUTPUT ${KERNEL_OBJ} ...): hello_kern.c dosyasını clang kullanarak hello_kern.o nesne dosyasına derlemek için özel bir komut tanımlandı.
  - add_custom_command(OUTPUT ${SKELETON_HEADER} ...): bpftool gen skeleton komutunu kullanarak hello_kern.o dosyasından hello_kern.skel.h başlık dosyasını (iskelet) oluşturmak için ikinci bir özel komut tanımlandı.
- **Hedeflerin Tanımlanması:**
  - add_executable(hello_ebpf ${USER_SRC}): Kullanıcı alanı ajanını (hello_user.c) derlemek için ana çalıştırılabilir hedef (hello_ebpf) tanımlandı.
  - add_dependencies(hello_ebpf ebpf_build): Ana hedefin derlenmesinin, eBPF iskelet

dosyasının oluşturulmasına bağlı olmasını zorunlu kılar. Bu, hello_user.c derlenmeden önce hello_kern.skel.h dosyasının hazır olmasını garanti eder.

2. Çekirdek Alanı (Kernel-Space) Programı (hello_kern.c)

Prototipin ilk parçası, çekirdekte çalışacak ve execve olaylarını yakalayacak olan eBPF programıdır.

- **vmlinux.h Başlık Dosyası:** Programın, çekirdek veri yapılarına (örn: struct trace_event_raw_sys_enter) erişebilmesi için vmlinux.h dosyasına ihtiyacı vardır. Bu dosya, bpftool btf dump file /sys/kernel/btf/vmlinux format c > vmlinux.h komutuyla, çalışan sanal makine çekirdeğinin BTF (BPF Type Format) bilgilerinden oluşturulmuştur.
- **Bağlanma Noktası (Hook):** Program, SEC("tracepoint/syscalls/sys_enter_execve") makrosunu kullanarak execve sistem çağrısının giriş (enter) izleme noktasına (tracepoint) bağlanmıştır.
- **Eylem:** Olay tetiklendiğinde, handle_execve_enter fonksiyonu bpf_printk("Hello eBPF! ...") yardımcısını kullanarak çekirdeğin izleme borusuna (trace_pipe) basit bir hata ayıklama (debug) mesajı yazdırmaktadır.
- **Lisans:** Çekirdeğin programı yüklemesine izin vermesi için char LICENSE[] SEC("license") = "GPL"; ile GPL lisansı zorunlu olarak belirtilmiştir.

3. Kullanıcı Alanı (User-Space) Ajanı (hello_user.c)

İkinci parça, eBPF programını çekirdeğe yükleyen, bağlayan ve çalışmasını yöneten C ajanıdır. Bu ajan, Rapor-1'de hedeflenen modern libbpf akışını kullanmaktadır.

- **İskelet (Skeleton) Kullanımı:** Ajan, bpftool tarafından oluşturulan hello_kern.skel.h dosyasını #include eder. Bu iskelet dosyası, eBPF programını yönetmek için gerekli tüm (open, load, attach, destroy) fonksiyonları otomatik olarak sağlar.
- **Yükleme Akışı:**
  1. hello_kern__open(): eBPF nesne dosyasını açar ve iskelet yapısını hazırlar.
  2. hello_kern__load(): Programı ve haritaları (varsa) doğrular ve çekirdek belleğine yükler.
  3. hello_kern__attach(): Yüklenen programı, hello_kern.c içinde tanımlanan SEC("...") bölümündeki hedefe (yani execve tracepoint'ine) bağlar.
- **Çalışma Döngüsü:** Program, signal(SIGINT, ...) ve signal(SIGTERM, ...) ile Ctrl+C gibi sonlandırma sinyallerini yakalamak için bir sinyal işleyici (signal handler) kurar. Ajan, exiting değişkeni true olana kadar bir while döngüsü içinde sleep(1) ile bekler.
- **Temizleme:** Kullanıcı Ctrl+C'ye bastığında döngü kırılır ve hello_kern__destroy(skel) fonksiyonu çağrılarak eBPF programı çekirdekten güvenli bir şekilde ayrılır (detach) ve kaldırılır (unload).

4. Doğrulama ve Test Sonuçları

Prototipin çalıştığını doğrulamak için iki terminalli bir test senaryosu uygulanmıştır (Her iki

terminal de CLion içinden açılmıştır):

1. **Terminal 1 (İzleme):** Düzeltilen yol kullanılarak çekirdek loglarını dinlemeye başlamıştır: sudo cat /sys/kernel/tracing/trace_pipe

2. **Terminal 2 (Ajan):** Derlenen ajan, eBPF programını yüklemek için sudo yetkisiyle çalıştırılmıştır:
   sudo ./cmake-build-debug-ebpf-dev-remote/hello_ebpf

   Bu komut sonrasında "eBPF program loaded successfully." mesajı alınmıştır.
3. **Tetikleme:** Ajan çalışırken, ls, pwd, whoami gibi rastgele komutlar (veya başka bir terminalden) çalıştırılarak execve sistem çağrısı tetiklenmiştir.

Sonuç: ls gibi bir komut çalıştırıldığı anda, **Terminal 1**'de (izleme terminali) aşağıdaki gibi bpf_printk çıktıları gerçek zamanlı olarak başarıyla gözlemlenmiştir:

<...>-2448 [001] .... 1435.348301: bpf_trace_printk: Hello eBPF! execve syscall detected.
<...>-2449 [000] .... 1436.728101: bpf_trace_printk: Hello eBPF! execve syscall detected.

5. Sonuç ve Sonraki Adımlar

1. Hafta planının "Merhaba eBPF" prototipi geliştirme adımı başarıyla tamamlanmıştır.
2. Rapor-2'de kurulan modern eBPF geliştirme akışı (CLion -> CMake -> bpftool -> libbpf) uçtan uca doğrulanmıştır.

Mevcut prototip, "bir execve olayının gerçekleştiğini" tespit edebilmektedir. Rapor-1'deki plana uygun olarak bir sonraki adım, bu tespiti daha anlamlı hale getirmektir: **"hangi programın çalıştırıldığını"** tespit etmek.

Bu, bpf_printk (sadece hata ayıklama içindir) kullanımını bırakıp, execve sistem çağrısının filename (dosya adı) parametresini okumayı ve bu veriyi çekirdekten kullanıcı alanına yüksek performanslı **eBPF Haritaları (BPF Maps)** (örneğin BPF_MAP_TYPE_RINGBUF) kullanarak aktarmayı gerektirecektir.