Rapor 5 - Durum Yönetimi ve Kapsamlı Dosya Sistemi İzleme Entegrasyonu

1. Özet

Bu rapor, fidye yazılımlarının gerçek zamanlı tespiti amacıyla geliştirilen eBPF (Extended Berkeley Packet Filter) tabanlı güvenlik aracının geliştirme sürecindeki kritik bir aşamayı belgelemektedir. Projenin önceki fazlarında kurulan temel veri boru hattı, bu dönemde gerçekleştirilen çalışmalarla **hafızalı** ve **bağlama duyarlı** bir yapıya evrilmiştir.

Raporlanan dönemde iki ana hedef başarıyla gerçekleştirilmiştir:

1. **Durum Yönetimi:** Kullanıcı alanı ajanına uthash kütüphanesi entegre edilerek, sistemdeki süreçlerin tarihsel aktivitelerinin izlenmesi ve istatistiksel verilerinin (dosya yazma hızı vb.) bellekte tutulması sağlanmıştır.
2. **Dosya Sistemi İzleme Genişletmesi:** Fidye yazılımlarının saldırı zincirini oluşturan şifreleme (write) ve dosya uzantısı değiştirme (rename) eylemleri, modern Linux çekirdek yapılarına (renameat2) uyumlu olacak şekilde izleme kapsamına alınmıştır.

2. Giriş ve Problem Tanımı

2.1. Mevcut Durum Analizi

Projenin Rapor-3 ve Rapor-4 aşamalarında, çekirdek alanından kullanıcı alanına (User Space) Ring Buffer aracılığıyla veri aktaran yüksek performanslı bir iletişim kanalı kurulmuştu. Ancak sistem, yakaladığı olayları anlık olarak işleyip unutan **"Hafızasız"** bir yapıydaydı.

2.2. Davranışsal Analiz Gereksinimi

Geleneksel imza tabanlı sistemlerin aksine, davranışsal analiz "tekil olaylara" değil, "olaylar zincirine" odaklanmak zorundadır. Örneğin:

- Meşru bir metin editörünün bir dosyaya yazması (**Normal Davranış**).
- Bir sürecin 1 saniye içinde 500 farklı dosyaya yazması (**Anomali / Fidye Yazılımı Şüphesi**).

Bu ayrımı yapabilmek için sistemin, *"Bu süreç (PID) daha önce ne yaptı?"* sorusuna cevap verebilecek bir hafıza mekanizmasına ve dosya sistemi üzerindeki tüm kritik operasyonları (open, write, rename) görebilen geniş bir vizyona ihtiyacı vardır. Bu çalışma dönemi, bu eksiklikleri gidermek üzerine kurgulanmıştır.

3. Mimari Tasarım ve Teknik İmplementasyon

Bu bölümde, sisteme entegre edilen modüllerin mühendislik kararları ve kod seviyesindeki detayları sunulmuştur.

3.1. Durum Yönetimi Modülü (User Space State Management)

Süreçlerin takibi için bir veri yapısı ve arama algoritması gereklidir. Bu kapsamda, projenin performans gereksinimleri (Düşük gecikme, Düşük CPU kullanımı) gözetilerek mimari seçimler yapılmıştır.

- Teknoloji Seçimi: uthash
  Harici bir veritabanı (Redis, SQL vb.) kullanmak, eBPF'in sağladığı performans avantajını darboğaza sokacağından, C dili için geliştirilmiş makro tabanlı bir hash tablosu olan uthash tercih edilmiştir. Bu yapı, verileri doğrudan RAM üzerinde tutar ve O(1) karmaşıklığı ile erişim sağlar.
- Veri Yapısı Tasarımı (struct process_stats):
  Her bir süreç için aşağıdaki metrikleri tutan dinamik bir struct tasarlanmıştır:

```
struct process_stats {
    int pid;                // ANAHTAR (Key): Süreci tanımlayan benzersiz ID
    char comm[16];          // Süreç Adı (Örn: "bash", "python")
    unsigned long exec_count;  // Sayaç: Çalıştırılan alt program sayısı
    unsigned long write_count; // Sayaç: Dosya yazma (şifreleme potansiyeli) sayısı
    UT_hash_handle hh;      // Uthash yönetim kancası
};
```

  Bu yapı, ilerleyen aşamalarda "Zaman Damgası" (Timestamp) ve "Eşik Değerleri" (Thresholds) eklenmesine uygun şekilde esnek tasarlanmıştır.

3.2. Çekirdek Alanı İzleme Genişletmesi

Fidye yazılımlarının yaşam döngüsünü kapsayabilmek için hello_kern.c içerisindeki eBPF programı, aşağıdaki sistem çağrılarını izleyecek şekilde genişletilmiştir:

1. **sys_enter_write (Yazma Operasyonu):**
   - **Amaç:** Dosya içeriğinin değiştirilmesini (şifreleme) yakalamak.
   - **Teknik Detay:** write çağrısı dosya yolu yerine dosya tanımlayıcısı (File Descriptor - FD) kullandığı için, eBPF tarafında dosya ismini çözümlemek maliyetlidir. Bu aşamada, dosya isminden ziyade **"yazma eyleminin sıklığı"** istatistiği hedeflendiği için dosya ismi ihmal edilmiştir.
2. **sys_enter_renameat ve sys_enter_renameat2 (Yeniden Adlandırma):**
   - **Amaç:** Şifreleme sonrası dosya uzantısının değişmesini (Örn: .docx -> .docx.locked) tespit etmek.
   - **Kritik Keşif:** Modern Linux dağıtımlarında, mv komutunun standart renameat yerine, atomik işlem garantisi sunan **renameat2** çağrısını kullandığı tespit edilmiştir. Bu

nedenle projeye renameat2 kancası eklenerek tam uyumluluk sağlanmıştır.

3.3. Veri Protokolü (common.h)

Çekirdek ve kullanıcı alanı arasındaki veri bütünlüğünü sağlamak adına, olay türlerini ayırt eden bir enum yapısı protokole eklenmiştir:

```
enum event_type {
    EVENT_EXEC = 1,
    EVENT_WRITE = 2,
    EVENT_OPEN = 3,
    EVENT_RENAME = 4
};
```

4. Karşılaşılan Mühendislik Zorlukları ve Çözümler

Projenin bu safhasında karşılaşılan zorluklar, sistem programlamanın doğasından kaynaklanan karmaşık problemlerdir.

4.1. Geri Besleme Döngüsü (Feedback Loop) Problemi

- **Problem:** Dosya yazma (write) olaylarının izlenmeye başlanmasıyla birlikte, sistemde kontrolsüz bir yük artışı gözlemlenmiştir. Yapılan kök neden analizinde; ajanın yakaladığı bir write olayını terminale basmak için printf() fonksiyonunu kullandığı, printf()'in arka planda tekrar write sistem çağrısı yaptığı ve bunun eBPF tarafından tekrar yakalanarak sonsuz bir döngü oluşturduğu tespit edilmiştir.
- **Çözüm (Self-Filtering):** Kullanıcı alanı ajanına **"Kendini Yoksay"** mantığı eklenmiştir. Ajan başlatıldığında getpid() ile kendi kimliğini öğrenmekte ve eBPF'ten gelen paketin kaynak PID'si kendisine aitse, bu paketi işlemeden düşürmektedir.

4.2. Gürültü Filtreleme (Noise Reduction)

- **Problem:** İşletim sisteminin arka plan servisleri (systemd, journald, ssh) saniyede yüzlerce dosya operasyonu gerçekleştirmektedir. Bu durum, test çıktılarını analiz etmeyi zorlaştırmıştır.
- **Çözüm:** Geliştirme aşamasında odaklanmayı kolaylaştırmak adına, sshd ve sudo gibi bilinen altyapı süreçlerini loglamadan hariç tutan geçici bir beyaz liste mekanizması hello_user.c içerisine kodlanmıştır.

5. Doğrulama, Test ve Bulgular

Geliştirilen modüllerin doğruluğu, izole sanal makine ortamında aşağıdaki test senaryoları ile kanıtlanmıştır:

Test 1: Süreç Hafızası Doğrulaması

- **Senaryo:** Aynı terminalde ardışık komutlar (ls, id) çalıştırılması.
- **Beklenen:** Hash tablosunda ilgili shell PID'si için exec_count değerinin artması.
- **Sonuç:** [EXEC] PID: 3422 | Count: 5 şeklinde artan sayaçlar gözlemlenmiş, uthash entegrasyonunun hafıza tuttuğu doğrulanmıştır.

Test 2: Fidye Yazılımı Simülasyonu (Dosya Şifreleme)

- **Senaryo:** echo "veri" > test.txt komutu ile dosya yazma işlemi.
- **Sonuç:** [WRITE] olayı yakalanmış ve ilgili PID'nin write_count değeri artmıştır. Bu veri, bir sonraki aşamada "Eşik Bazlı Tespit" (saniyede >100 yazma) kuralı için temel teşkil edecektir.

Test 3: Fidye Yazılımı Simülasyonu (Uzantı Değiştirme)

- **Senaryo:** mv test.txt test.txt.locked komutu.
- **Sonuç:** Sistem, modern renameat2 çağrısını başarıyla yakalamış ve [RENAME] logunu üretmiştir. Bu, fidye yazılımı tespitinde en güçlü imzalardan biri olan uzantı değişimini yakalayabildiğimizi kanıtlamaktadır.

6. Sonuç ve Gelecek Dönem Planlaması

Bu rapor dönemi sonunda, projenin **"Veri Toplama"** ve **"Bağlam Oluşturma"** katmanları tamamlanmıştır. Sistem artık kör bir gözlemci değil; olayları türlerine ayıran, süreçleri tanıyan ve kendi yarattığı gürültüyü filtreleyebilen akıllı bir veri toplayıcıdır.

Gelecek Çalışmalar (Hafta 5 ve Sonrası):
Proje takvimine uygun olarak, toplanan bu zengin veri seti üzerinde çalışacak olan Kural Motoru (Heuristic Engine) geliştirilecektir. Odaklanılacak konular:

1. **Hız Analizi (Rate Limiting):** struct process_stats yapısına zaman damgası eklenerek, *"1 saniyede X dosya işlemi"* kuralının kodlanması.
2. **Anomali Tespiti:** Normal kullanıcı davranışından sapan yüksek hacimli dosya aktivitelerinin (Bulk File Operations) tespiti.