Proje Yol Haritası ve Rapor Taslağı

1. GİRİŞ

1.1. Projenin Amacı ve Gerekçesi

Fidye yazılımları (ransomware), sürekli gelişen yapılarıyla geleneksel imza tabanlı güvenlik sistemleri için büyük bir tehdit oluşturmaktadır. Bu sistemlerin yetersiz kaldığı noktada, zararlı yazılımları kimliklerine göre değil, sistemdeki **davranışlarına** göre analiz eden proaktif yöntemlere ihtiyaç duyulmaktadır.

Bu projenin temel amacı, bu ihtiyaca yönelik olarak, daha önce hiç görülmemiş (zero-day) saldırıları dahi tespit etme potansiyeline sahip bir davranışsal analiz aracı geliştirmektir. Bu hedefe ulaşmak için, Linux çekirdeğindeki sistem olaylarını güvenli ve yüksek performanslı bir şekilde izlemeyi sağlayan modern **eBPF (Genişletilmiş Berkeley Paket Filtresi)** teknolojisinden faydalanılacaktır. Geliştirilecek araç, eBPF kullanarak bir fidye yazılımının tipik eylemlerini (hızlı dosya şifreleme, sistem yedeklerini silme vb.) gerçek zamanlı olarak izleyip tespit edecektir.

Akademik olarak proje, C dili, libbpf ve eBPF gibi düşük seviyeli teknolojiler aracılığıyla sistem programlama ve işletim sistemleri alanlarında pratik yetkinlik kazandırmayı hedeflemektedir.

1.2. Projenin Kapsamı

Projenin hedeflerine 3 aylık süre zarfında ulaşılabilmesi için kapsamının net bir şekilde tanımlanması esastır. Bu bağlamda, projenin sınırları aşağıdaki maddelerle belirlenmiştir:

- **Platform Bağımlılığı:** Geliştirilecek olan tespit aracı, temelindeki eBPF teknolojisinin doğal bir gereği olarak **sadece Linux işletim sistemleri** üzerinde çalışacaktır. Windows veya macOS gibi diğer platformlar için bir uyumluluk sağlanmayacaktır. Bu odaklanma, Linux çekirdeği ile daha derin bir entegrasyon kurarak daha isabetli sonuçlar elde etmeyi mümkün kılacaktır.
- **Tespit Odak Alanları:** Araç, fidye yazılımlarının en belirgin ve yüksek kesinlikli göstergeleri olan davranışlara odaklanacaktır. Kapsam dahilindeki izleme alanları öncelikli olarak **dosya sistemi işlemleri** (dosya oluşturma, okuma, yazma, yeniden adlandırma) ve **süreç yönetimi** (yeni işlem başlatma, özellikle vssadmin gibi şüpheli komutların çalıştırılması) olacaktır. Ağ aktivitesi analizi, bellek içi (in-memory) analizler veya kullanıcı alanı (user-space) API kancaları gibi daha karmaşık yöntemler bu projenin kapsamı dışındadır.
- **Proje Niteliği:** Geliştirilecek araç, ticari bir güvenlik ürünü olma iddiası taşımayan bir **"Kavram Kanıtlama" (Proof of Concept - PoC)** prototipidir. Projenin temel amacı, eBPF teknolojisinin fidye yazılımı davranış analizi için ne kadar uygun ve etkili bir yöntem

olduğunu akademik ve pratik olarak göstermektir. Dolayısıyla, grafik kullanıcı arayüzü (GUI), merkezi yönetim paneli, detaylı konfigürasyon dosyaları veya otomatik güncelleme gibi son kullanıcıya yönelik özellikler geliştirilmeyecektir.

1.3. Proje Çıktıları (Hedeflenen Sonuçlar)

Projenin tamamlanmasıyla birlikte, elde edilen bilgi birikimi ve geliştirilen yazılımın somut olarak sunulacağı üç temel çıktı hedeflenmektedir:

- **Akademik Bitirme Tezi:** Projenin teorik altyapısını, metodolojisini ve sonuçlarını içeren, üniversitenin tez yazım kurallarına uygun olarak hazırlanmış kapsamlı bir yazılı belge sunulacaktır. Bu tez; literatür taraması, kullanılan eBPF teknolojisinin detayları, geliştirilen aracın mimari tasarımı, yapılan testler, elde edilen bulgular ve gelecek çalışmalar için önerileri içerecektir.
- **Belgelendirilmiş GitHub Reposu:** Projenin tüm kaynak kodları (eBPF C kodu ve kullanıcı alanı C ajanı), derleme talimatlarını içeren Makefile dosyaları ile birlikte açık kaynaklı bir GitHub reposunda yayınlanacaktır. Repo, projenin amacını, kurulum adımlarını ve kullanımını açıklayan detaylı bir README.md dosyası ile belgelendirilecektir. Bu sayede projenin şeffaflığı, tekrarlanabilirliği ve gelecekteki akademik çalışmalara kaynak oluşturması hedeflenmektedir.
- **Çalışan Prototip ve Canlı Demo:** Projenin en kritik çıktısı, geliştirilen tespit aracının işlevselliğini kanıtlayan, çalışan bir prototiptir. Proje savunması sırasında, kontrollü bir sanal makine ortamında önceden hazırlanmış bir test senaryosu (örneğin, eğitim amaçlı bir fidye yazılımı script'inin çalıştırılması) ile aracın fidye yazılımı davranışını nasıl gerçek zamanlı olarak tespit ettiği ve alarm ürettiği canlı olarak gösterilecektir.

2. METODOLOJİ VE TEKNOLOJİ

2.1. Kullanılan Teknoloji Yığını

Projenin hedeflerine ulaşmak için seçilen teknolojiler, sistem programlama alanındaki standartları ve modern eBPF geliştirme pratiklerini yansıtmaktadır. Seçimlerde performans, kontrol ve minimal bağımlılık ilkeleri göz önünde bulundurulmuştur.

- **Çekirdek Alanı Programlama: Kısıtlı C & Clang/LLVM**
  - eBPF programları, Linux çekirdeği tarafından doğrulanabilen özel bir bytecode derlenir. Bu bytecode üretmenin standart ve tek yolu, C dilinin kısıtlı bir alt kümesini kullanarak kod yazmak ve bunu Clang/LLVM araç zinciri ile derlemektir. Bu seçim, teknolojik bir gerekliliktir.
- **Kullanıcı Alanı Programlama: C & GCC**
  - Projenin temel sistem programlama hedeflerine uygun olarak, bellek yönetimi ve sistem kaynakları üzerinde tam kontrol sağlayan C dili tercih edilmiştir. GCC, Linux platformu için standart ve olgun bir derleyicidir. Bu yaklaşım, işletim sistemiyle en alt seviyede etkileşim kurma deneyimi sunmaktadır.

- **eBPF Etkileşimi: libbpf Kütüphanesi & bpftool**
  - libbpf, çekirdek ekibi tarafından geliştirilen ve eBPF programlarını kullanıcı alanından yönetmek (yükleme, haritalara erişim vb.) için kullanılan modern C kütüphanesidir. bpftool gen skeleton aracı ise, libbpf kullanımını büyük ölçüde basitleştirerek standart kodları otomatik üreten ve geliştirme sürecini hızlandıran pratik bir yardımcıdır.
- **Durum Yönetimi: uthash**
  - Kullanıcı alanı ajanının, işlemlerin (PID) davranışlarını takip edebilmesi için verimli bir hash tablosuna ihtiyacı vardır. uthash, projeye harici bir kütüphane bağımlılığı eklemeden, sadece tek bir başlık dosyası (.h) ile bu ihtiyacı karşılayan hafif ve performanslı bir çözümdür.

Not: Proje geliştirme sürecinde, pratik ihtiyaçlar doğrultusunda bu teknoloji yığınına eklemeler veya değişiklikler yapılabileceği öngörülmektedir.

2.2. Geliştirme ve Test Ortamı Mimarisi

Proje, kontrollü ve tekrarlanabilir bir ortamda geliştirilip test edilecektir. Bu amaçla, ana geliştirme makinesinden tamamen izole edilmiş bir sanal makine mimarisi benimsenmiştir. Bu yaklaşım, hem güvenlik hem de geliştirme verimliliği açısından kritik avantajlar sunar.

- **Sanallaştırma: GNOME Boxes (QEMU/KVM)**
  - Fidye yazılımı testlerinin ana sisteme zarar vermemesi için QEMU/KVM tabanlı, yüksek performanslı bir sanallaştırma ortamı olan GNOME Boxes kullanılacaktır. Bu ortamın Anlık Görüntü (Snapshot) özelliği, her testten sonra sistemi saniyeler içinde temiz bir başlangıç noktasına döndürmeyi mümkün kılarak test döngüsünü hızlandıracaktır.
- **İşletim Sistemi: Debian 12 "Bookworm"**
  - Sanal makine üzerinde, stabil ve öngörülebilir yapısıyla bilinen Debian 12 "Stable" sürümü çalışacaktır. Sistemin sadece Komut Satırı Arayüzü (CLI) ile kurulması, arka plan gürültüsünü en aza indirerek eBPF aracının sadece hedeflenen olayları izlemesini sağlayacak ve sistem kaynaklarının verimli kullanılmasını temin edecektir.
- **Geliştirme Akışı: Uzaktan Geliştirme**
  - Geliştirme süreci, ana makinede çalışan CLion gibi tam özellikli bir IDE'nin "Remote - SSH" yetenekleri kullanılarak yürütülecektir. Kod yazma ve düzenleme işlemleri ana makinenin konforlu arayüzünde yapılırken, kodun derlenmesi ve çalıştırılması SSH üzerinden doğrudan sanal makine üzerinde gerçekleştirilecektir. Bu modern akış, minimal bir test ortamı ile güçlü bir geliştirme ortamının avantajlarını birleştirmektedir.

3. PROJE MİMARİSİ VE TASARIMI

3.1. Genel Sistem Mimarisi (Yüksek Seviye Tasarım)

Geliştirilecek aracın mimarisi, modern sistem izleme prensiplerine uygun olarak iki ana ve birbirinden mantıksal olarak ayrılmış bileşenden oluşur: **Çekirdek Alanı (Kernel Space)** ve

Kullanıcı Alanı (User Space). Bu ayrım, sistem üzerindeki performans etkisini en aza indirirken, analiz ve tespit mantığının esnek bir şekilde geliştirilmesine olanak tanır.

- **1. Çekirdek Alanı Bileşeni (Veri Toplayıcı):** Bu katman, sistemin kalbinde, yani Linux çekirdeği içinde çalışan eBPF programlarından oluşur. Bu programların tek ve net bir görevi vardır: Fidye yazılımı davranışını gösterebilecek kritik sistem çağrılarını (syscalls) ve olayları (events) yakalamak. Bu katman, olabildiğince hafif, hızlı ve "aptal" (logic-free) olacak şekilde tasarlanmıştır. Karmaşık analizler veya durum takibi burada yapılmaz; sadece ham veri toplanır ve bir sonraki katmana iletilir.
- **2. Kullanıcı Alanı Bileşeni (Analiz Motoru):** Bu katman, standart bir kullanıcı süreci olarak çalışan C ile yazılmış ajandır. Çekirdekten gelen ham veri akışını alır ve anlamlı hale getirir. Tüm karmaşık işlemler bu katmanda gerçekleşir:
  - Gelen olayları ayrıştırma.
  - uthash kullanarak işlem bazında (per-PID) durum takibi yapma.
  - Önceden tanımlanmış kuralları (heuristics) içeren Kural Motorunu çalıştırarak anomali tespiti yapma.
  - Tespit durumunda alarm üretme ve loglama.
- **Veri Akış Yönü:** Bu iki katman arasındaki iletişim, **tek yönlüdür**. Veri, daima Çekirdek Alanından Kullanıcı Alanına doğru akar. Bu akış, Ring Buffer adı verilen yüksek performanslı bir mekanizma ile sağlanır. Bu model, kullanıcı alanındaki bir hatanın veya gecikmenin çekirdeğin çalışmasını etkilemesini engeller ve sistemin genel kararlılığını korur.

Bu "görev ayrılığı" (separation of concerns) ilkesi, projenin temelini oluşturur: Çekirdek sadece *görür*, Kullanıcı Alanı ise *düşünür ve karar verir*.

3.2. Çekirdek Alanı Bileşeni (eBPF Programı)

Sistemin veri toplama mekanizmasının temelini oluşturan bu bileşen, Linux çekirdeği içinde çalışan ve C diliyle yazılmış eBPF programlarından oluşur. Bu programlar, sistemin en güvenilir ve stabil izleme noktaları olan **sistem çağrısı izleme noktalarına (syscall tracepoints)** kanca atarak (hook) çalışır. Projenin MVP (Minimum Viable Product) aşaması için izlenmesi hedeflenen temel sistem çağrıları şunlardır:

- **sys_enter_openat**: Dosya açma girişimleri.
- **sys_enter_write**: Dosyalara veri yazma girişimleri.
- **sys_enter_renameat**: Dosyaların yeniden adlandırılması (örn: dosya.txt -> dosya.txt.locked).
- **sys_enter_execve**: Yeni bir programın çalıştırılması (örn: vssadmin gibi sistem araçları).

Bu bileşenin tasarım felsefesi **"minimum mantık, maksimum performans"** ilkesine dayanır. eBPF programları, çekirdek bağlamında çalıştıkları için son derece verimli olmalıdır. Bu nedenle, programlar içinde karmaşık döngüler, koşullu ifadeler veya durum takibi yapılmaz. Görevleri, tetiklenen olayla ilgili meta veriyi (süreç ID, komut adı, dosya yolu vb.) olabildiğince

hızlı bir şekilde toplamak ve bu veriyi standart bir C struct yapısı içinde paketlemektir. Analiz ve karar verme yükü tamamen kullanıcı alanı bileşenine bırakılmıştır.

3.3. Çekirdek-Kullanıcı Alanı İletişim Modeli

Çekirdek alanındaki eBPF programlarının topladığı ham verinin, analiz edilmek üzere kullanıcı alanındaki ajana verimli ve güvenilir bir şekilde aktarılması, sistemin performansı için hayati öneme sahiptir. Bu projede, bu iletişim kanalı olarak modern eBPF harita türü olan **BPF_MAP_TYPE_RINGBUF (Ring Buffer)** seçilmiştir.

Ring Buffer, eski yöntem olan Perf Buffer'a göre önemli avantajlar sunan, paylaşımlı bellek (shared memory) tabanlı bir "Çoklu-Üretici, Tek-Tüketici" (MPMC - Multiple-Producer, Single-Consumer) kuyruk mekanizmasıdır.

- **Çalışma Prensibi:** Çekirdek ve kullanıcı alanı arasında paylaşılan bir bellek alanı oluşturulur. eBPF programları (üreticiler), topladıkları olay verilerini doğrudan bu paylaşılan alana kopyalar. Kullanıcı alanı ajanı (tüketici) ise, periyodik olarak bu alanı kontrol ederek (polling) yeni verileri okur.
- **Seçilme Nedenleri:**
    1. **Yüksek Performans:** Veri, çekirdekten kullanıcı alanına her olay için ayrı ayrı kopyalanmak yerine, paylaşılan bellek üzerinden doğrudan erişilir. Bu, CPU döngülerinden ve bellek bant genişliğinden ciddi ölçüde tasarruf sağlar.
    2. **Kayıpsız Veri Aktarımı:** Perf Buffer'ın aksine, Ring Buffer tasarımı gereği olayların üzerine yazılmasını veya kaybolmasını engeller. Eğer kullanıcı alanı ajanı veriyi yeterince hızlı okuyamazsa, çekirdek yeni olayları göndermeyi durdurur (ve olay kaybını sayar), bu da sistemin durumu hakkında net bilgi sağlar.
    3. **Verimli Bellek Kullanımı:** Tek bir paylaşımlı bellek alanı kullanması, Perf Buffer'da olduğu gibi her CPU için ayrı tampon bellekler oluşturma ihtiyacını ortadan kaldırır.

Bu model, sistemde saniyede binlerce olayın yaşandığı durumlarda bile, çeklek üzerinde minimum ek yük (overhead) oluşturarak veri boru hattının (data pipeline) tıkanmadan çalışmasını garanti altına alır.

3.4. Kullanıcı Alanı Bileşeni (C Ajanı)

Eğer çekirdek alanı bileşeni projenin "gözleri" ise, kullanıcı alanı bileşeni de projenin "beyni" ve "kontrol merkezi"dir. C dili ile geliştirilen bu ajan, standart bir kullanıcı süreci olarak çalışır ve projenin tüm mantıksal operasyonlarından sorumludur. Bileşen, modüler bir yapıda tasarlanmış olup ana sorumlulukları şunlardır:

- **eBPF Program Yükleyicisi (Loader):** Ajanın ilk görevi, libbpf kütüphanesini kullanarak derlenmiş eBPF programının (.bpf.o dosyası) bytecode'unu okumak ve Linux çekirdeğine güvenli bir şekilde yüklemektir. Bu süreç, bpftool gen skeleton aracıyla otomatik olarak üretilen iskelet kod sayesinde basitleştirilmiştir. Yükleme sırasında eBPF Doğrulayıcısı

(Verifier), programın çekirdek için güvenli olduğunu teyit eder.

- **Kanca Yöneticisi (Attacher):** eBPF programı çekirdeğe yüklendikten sonra, ajanın görevi bu programı ilgili tracepoint'lere (örn: sys_enter_renameat) bağlamaktır. Bu "kancalama" (attaching) işlemi, eBPF programının ilgili sistem çağrısı tetiklendiğinde otomatik olarak çalışmasını sağlar.
- **Olay Dinleyicisi (Event Listener):** Ajanın ana çalışma döngüsü bu modüldür. libbpf fonksiyonları aracılığıyla, çekirdekle paylaşılan Ring Buffer'ı sürekli olarak ve verimli bir şekilde dinler. Çekirdekten yeni bir olay verisi gönderildiğinde, bu modül veriyi okuyarak ham halde bir sonraki modüle aktarır.
- **Ayrıştırıcı ve Loglayıcı (Parser & Logger):** Ham olay verisi, bu modül tarafından okunabilir bir C struct yapısına dönüştürülür. Bu aşamada, temel olay bilgileri (zaman damgası, PID, komut adı, olay türü vb.) standart çıktıya veya bir log dosyasına yazılarak sistemin genel aktivitesinin izlenmesi sağlanır.

Bu temel görevler, projenin daha karmaşık olan tespit mantığı için gerekli olan sürekli ve güvenilir veri akışını sağlar.

3.5. Tespit Mantığı Mimarisi (Kural Motoru ve Durum Yönetimi)

Kullanıcı alanı ajanının en kritik ve karmaşık bileşeni, çekirdekten gelen ham veri akışını "bilgiye" dönüştüren Tespit Mantığıdır. Bu mantık, iki temel alt bileşenden oluşur:

- **Durum Yönetimi (State Management):** Davranışsal analiz yapabilmek için, her bir sürecin (process) zaman içindeki eylemlerini hatırlamak gerekir. Bu "hafıza", uthash kütüphanesi kullanılarak implemente edilen bir hash tablosu ile sağlanır. Bu tabloda anahtar (key) olarak Süreç Kimliği (PID), değer (value) olarak ise o sürece ait istatistikleri tutan bir process_stats struct'ı bulunur. Çekirdekten her yeni olay geldiğinde, ilgili PID'ye ait process_stats yapısı güncellenir (örn: dosya yazma sayacı artırılır, son işlem zamanı kaydedilir).
- **Kural Motoru (Heuristic Engine):** Bu motor, durum yönetimi tablosundaki verileri analiz eden bir C fonksiyonları setidir. Her olaydan sonra ilgili sürecin durumu güncellendiğinde, bu motor tetiklenir ve önceden tanımlanmış bir dizi kuralı (heuristic) bu güncel veri üzerinde çalıştırır. Amacı, normal sistem davranışından sapan ve fidye yazılımı aktivitesine işaret eden kalıpları aramaktır. Proje kapsamında iki temel kural türü hedeflenmektedir:
    1. **Eşik Bazlı Kurallar:** Bir sürecin belirli bir zaman aralığında belirli bir eylemi "çok fazla" yapmasını kontrol eder. Örneğin, "5 saniye içinde 10'dan fazla .txt uzantılı dosyayı yeniden adlandıran" bir süreci şüpheli olarak işaretler.
    2. **İmza Bazlı Kurallar:** Bilinen kötü niyetli komutların çalıştırılmasını anında tespit eder. Örneğin, execve olayı ile vssadmin delete shadows komutunun çalıştırıldığı tespit edildiğinde, bu durum doğrudan bir alarm tetikler.

Bu mimari, tekil ve anlamsız olayları, bir sürecin bütünsel davranışını yansıtan anlamlı bir bağlama oturtarak yüksek isabetli tespitler yapmayı hedefler.

4. RİSKLER VE GELECEK ÇALIŞMALAR

4.1. Olası Riskler ve Proje Kısıtları

Her akademik ve teknik projede olduğu gibi, bu projenin de potansiyel zorlukları ve doğal sınırları bulunmaktadır. Bu risklerin ve kısıtların farkında olmak, projenin başarısını ve elde edilen sonuçların doğru yorumlanmasını sağlamak için önemlidir.

- **Hatalı Tespit (False Positive) Riski:** Davranışsal analizin en temel zorluklarından biri, meşru bir sistem aktivitesi ile kötü niyetli bir aktiviteyi ayırt etmektir. Örneğin, bir yedekleme programı veya bir dosya indeksleme servisi de kısa sürede çok sayıda dosyaya erişebilir. Bu tür meşru programların, geliştirilen kural motoru tarafından yanlışlıkla fidye yazılımı olarak etiketlenme riski bulunmaktadır. Proje, bu riski azaltmak için kural setini olabildiğince spesifik tutmayı hedeflese de, kapsamlı bir beyaz liste (whitelist) mekanizması geliştirmek proje kapsamı dışındadır.
- **Performans Etkisi (Overhead) Kısıtı:** Sistem çağrılarının izlenmesi, doğası gereği sistem üzerinde bir miktar ek yük (overhead) oluşturur. Proje mimarisi (hafif eBPF programları, verimli Ring Buffer) bu etkiyi en aza indirmek üzere tasarlanmış olsa da, çok yüksek aktiviteye sahip sistemlerde aracın CPU ve bellek tüketimi bir kısıt haline gelebilir. Projenin amacı, bu etkinin kabul edilebilir sınırlar içinde kaldığını göstermektir, ancak detaylı performans kıyaslama (benchmarking) analizleri yapılmayacaktır.
- **Atlatma Teknikleri (Evasion) Tehdidi:** Siber güvenlik, sürekli bir kedi-fare oyunudur. Gelişmiş fidye yazılımları, izleme mekanizmalarını tespit edip bunlardan kaçınmak için karmaşık teknikler kullanabilir. Örneğin, eylemlerini çok uzun bir zamana yayarak eşik bazlı tespitten kaçınabilir veya doğrudan çekirdek seviyesinde manipülasyonlar yaparak sistem çağrısı izlemesini atlatabilirler. Bu proje, yaygın ransomware davranışlarını hedeflemektedir ve bu tür gelişmiş atlatma tekniklerine karşı bir savunma mekanizması geliştirme iddiasında değildir.

4.2. Kapsam Dışı Bırakılan ve Gelecekte Eklenebilecek Özellikler

Projenin zaman kısıtları ve "Kavram Kanıtlama" (PoC) niteliği göz önünde bulundurulduğunda, bazı ileri seviye özellikler kapsam dışında bırakılmıştır. Ancak bu özellikler, projenin temel mimarisinin ne kadar genişletilebilir olduğunu göstermekte ve gelecek akademik çalışmalar için bir yol haritası sunmaktadır:

- **Ağ Davranışı Analizi:** Mevcut proje sadece dosya ve süreç aktivitelerine odaklanmaktadır. Gelecekte, connect ve sendto gibi ağ sistem çağrıları izlenerek, fidye yazılımlarının Komuta ve Kontrol (C&C) sunucularıyla kurduğu iletişim veya şifreleme anahtarlarını dışarı sızdırma girişimleri gibi ağ tabanlı davranışları da tespit edilebilir. Bu, tespit yeteneğine yeni bir boyut ekleyecektir.
- **Çekirdek-içi Veri Agregasyonu:** Mevcut mimaride her olay çekirdekten kullanıcı alanına gönderilmektedir. Çok yoğun sistemlerde performansı daha da artırmak için, eBPF

haritaları kullanılarak bazı temel istatistikler (örn: bir saniyedeki dosya yazma sayısı) doğrudan çekirdek içinde toplanabilir. Kullanıcı alanı ajanı, bu özetlenmiş veriyi periyodik olarak okuyarak veri aktarım yükünü önemli ölçüde azaltabilir.

- **Aktif Engelleme (Prevention) Mekanizması:** Proje, doğası gereği bir *tespit* aracıdır. Gelecekte, eBPF'in Linux Güvenlik Modülleri (LSM) ile entegrasyonu kullanılarak, şüpheli olarak işaretlenen bir sürecin zararlı eylemleri (örn: dosya üzerine yazma) *gerçekleşmeden önce* aktif olarak engellenebilir. Bu, aracı pasif bir gözlemciden aktif bir savunma mekanizmasına dönüştürecektir.
- **Tuzak Dosya (Honeypot) Entegrasyonu:** Sistemin belirli konumlarına, normalde hiçbir meşru sürecin erişmemesi gereken "tuzak" dosyalar (örn: sifreler.txt) yerleştirilebilir. Bu dosyalara yapılan herhangi bir erişim, eBPF tarafından anında yakalanarak, neredeyse sıfır hatalı tespit (false positive) oranıyla çok yüksek kesinlikli bir alarm üretebilir.

5. PROJE TAKVİMİ

Bu takvim, projenin 3 aylık süre zarfında tamamlanması için tasarlanmış, hafta bazında detaylandırılmış bir yol haritasıdır. Her ay belirli bir ana temaya odaklanmakta ve her hafta somut çıktılar hedeflemektedir.

5.1. Ay 1: Altyapı Kurulumu ve Veri Boru Hattı

Bu ayın temel amacı, projenin üzerine inşa edileceği sağlam teknik temeli atmak ve çekirdekten kullanıcı alanına kadar olan veri akışını eksiksiz bir şekilde sağlamaktır.

- **Hafta 1 (20-26 Ekim):**
  - **Görevler:** Geliştirme ortamının (Debian 12 VM) kurulumu ve yapılandırılması. Gerekli tüm paketlerin (clang, libbpf-dev vb.) yüklenmesi. "Merhaba eBPF" projesi: Sadece execve olayını loglayan en basit eBPF programının ve bunu yükleyen C ajanının yazılması.
  - **Çıktı:** Çalışan geliştirme ortamı. GitHub reposu. execve olaylarını çekirdek logunda gösteren temel prototip.
- **Hafta 2 (27 Ekim - 2 Kasım):**
  - **Görevler:** BPF_MAP_TYPE_RINGBUF (Ring Buffer) mekanizmasının implementasyonu. eBPF programının olay verisini bir struct'a doldurup Ring Buffer'a göndermesi. Kullanıcı alanı ajanının Ring Buffer'ı dinleyip gelen veriyi konsola yazdırması.
  - **Çıktı:** Çekirdekten kullanıcı alanına canlı veri akışını sağlayan prototip.
- **Hafta 3 (3-9 Kasım):**
  - **Görevler:** uthash.h dosyasının projeye eklenmesi. Kullanıcı alanı ajanında, gelen olayların PID'ye göre bir hash tablosunda saklanması. process_stats yapısının oluşturulması.
  - **Çıktı:** İşlem bazında (per-PID) olayları hafızada biriktiren ve yöneten ajan.
- **Hafta 4 (10-16 Kasım):**
  - **Görevler:** openat, write, renameat sistem çağrıları için eBPF kancalarının eklenmesi.

Veri struct'ının ve kullanıcı alanı ayrıştırıcısının bu yeni olay türlerini destekleyecek şekilde güncellenmesi.
    - **Çıktı:** Dosya sistemi aktivitelerini de izleyebilen, kapsamlı veri toplayan prototip.

5.2. Ay 2: Kural Motoru Geliştirme ve MVP Prototip

Bu ayın odak noktası, toplanan ham veriyi anlamlı hale getirecek olan tespit mantığını (Kural Motoru) geliştirmek ve projenin çalışan ilk versiyonunu (MVP) ortaya çıkarmaktır.

- **Hafta 5-6 (17-30 Kasım):**
    - **Görevler:** Eşik bazlı ilk kuralın (H1: Hızlı dosya operasyonları) tasarlanması ve geliştirilmesi. process_stats yapısına zaman damgası eklenmesi. Belirli bir zaman penceresi içinde eşik aşımını kontrol eden C kodunun yazılması.
    - **Çıktı:** Hızlı dosya operasyonlarını tespit edip alarm üreten kural motoru.
- **Hafta 7 (1-7 Aralık):**
    - **Görevler:** İmza bazlı ikinci kuralın (H2: Şüpheli komut çalıştırma) geliştirilmesi. execve olaylarında çalıştırılan komutları bilinen şüpheli komut listesiyle karşılaştıran mantığın eklenmesi.
    - **Çıktı:** Hem eşik hem imza bazlı tespit yapabilen ajan.
- **Hafta 8 (8-14 Aralık):**
    - **Görevler:** Ajan'a --kill-on-detect gibi bir komut satırı argümanı eklenmesi ve tespit durumunda aksiyon (süreç sonlandırma) alan modülün yazılması. Kodun genel temizliği, yorum satırlarının eklenmesi ve README.md dosyasının güncellenmesi.
    - **Çıktı:** Sunuma hazır, belgelendirilmiş ve çalışan bir MVP (Minimum Viable Product) prototipi.

5.3. Ay 3: Test, Tez Yazımı ve Sunum Hazırlığı

Bu son ay, projenin teknik geliştirmesinden ziyade, sonuçların doğrulanması, akademik tezin yazılması ve proje savunmasına hazırlık süreçlerine ayrılmıştır.

- **Hafta 9-10 (15-28 Aralık):**
    - **Görevler:** Kontrollü test senaryolarının (basit fidye yazılımı script'leri) oluşturulması. MVP prototipinin bu senaryolarla test edilmesi ve sonuçların (ekran görüntüleri, loglar) belgelenmesi. Tezin "Giriş", "Metodoloji" ve "Mimari" (Bölüm 1, 2, 3) bölümlerinin yazılması.
    - **Çıktı:** Test sonuçları raporu. Tezin ilk yarısının taslağı.
- **Hafta 11 (29 Aralık - 4 Ocak):**
    - **Görevler:** Test sonuçlarının analiz edilerek tezin "Bulgular ve Sonuçlar" ve "Gelecek Çalışmalar" (Bölüm 4) bölümlerinin yazılması. Tezin tüm bölümlerinin birleştirilerek ilk tam taslağının oluşturulması.
    - **Çıktı:** Tezin ilk tam taslağı.
- **Hafta 12-13 (5-18 Ocak):**
    - **Görevler:** Tezin tam taslağının danışman hocaya sunulması, geri bildirimlerin alınması

ve gerekli düzeltmelerin yapılması. Proje sunum slaytlarının hazırlanması. Canlı demo senaryosunun provalarının yapılması. Projenin ve tezin son halinin teslim edilmesi.

- **Çıktı:** Teslime hazır bitirme tezi. Teslime hazır proje sunumu ve canlı demo.