Ağdaki Yazılımcıları Wireshark ile Tespit Etmek: 2025 Yılına Yönelik En Son Teknikler ve Trendler

1. Giriş

1.1. Projenin Amacı ve Kapsamı

Bu rapor, "Yazılımcı Avı" (Developer Hunter) projesi bağlamında, bir bilgisayar ağındaki yazılımcıları, ağ trafiği analiz aracı olan Wireshark kullanarak tespit etmeye yönelik 2025 yılı ve sonrası için öngörülen en son ve en etkili teknikleri ve trendleri ayrıntılı bir şekilde belirlemeyi amaçlamaktadır. Projenin temel hedefi, yazılımcıların karakteristik ağ aktivitelerini – örneğin, Git/SSH kullanımı, Entegre Geliştirme Ortamları (IDE'ler) ve kod editörlerinden kaynaklanan trafik, geliştirme/test sunucularına erişim, API test araçları trafiği, sanal makine/konteyner etkileşimleri ve özel port/protokol kullanımları – analiz ederek ayırt edici dijital izlerini saptamaktır. Bu rapor, kullanıcının belirttiği kısıtlamalara uygun olarak, spekülatif olmayan, kanıta dayalı ve güncel bilgilere odaklanmaktadır.

1.2. Ağ Trafik Analizinin ve Wireshark'ın Rolü

Ağ Trafik Analizi (NTA), ağ trafiği örüntülerini izleme ve denetleme sürecidir; bu süreç, anormallikleri veya şüpheli davranışları tanımlayarak ağ güvenliği ve operasyon ekiplerine tehditleri tespit etmede yardımcı olur.[1] NTA, aynı zamanda yüksek paket kaybı oranları veya yüksek ağ gecikmesi gibi performans sorunlarını gidermek için de faydalı olabilir.[1] Ağ trafiği örüntüleri büyük ölçüde değişebileceğinden, NTA'nın önemi daha da artmaktadır. Örneğin, paketlerin ağ segmentleri ve uç noktalar arasında hareket ederken katettiği yollar değişebilir, bu da farklı rotaların verimliliğine bağlı olarak farklı performans seviyelerine yol açabilir. Benzer şekilde, port tarama veya Hizmet Engelleme (DoS) saldırıları gibi kötü amaçlı ağ etkinlikleri genellikle olağandışı trafik örüntüleri oluşturur. Anormallikleri tespit ederek, kuruluşlar potansiyel güvenlik risklerini belirleyebilir ve bir ihlal meydana gelmeden önce bunları engelleyebilir.[1]

Wireshark, bu analiz sürecinde merkezi bir rol oynar. Ağ protokollerini mikroskobik düzeyde inceleme yeteneği sunan, yaygın olarak kullanılan güçlü bir ağ protokol analizörüdür.[2] Wireshark, ağ yöneticileri, güvenlik uzmanları, geliştiriciler ve eğitimciler tarafından ağ sorunlarını gidermek, ağ trafiğini analiz etmek ve güvenlik açıklarını tespit etmek için yaygın olarak kullanılır.[2] Projenin temel hedeflerinden biri olan, yazılımcılara özgü trafik desenlerini etkin bir şekilde saptamak ve diğer ağ trafiğinden ayrıştırmak amacıyla hem canlı veri yakalama aşamasında (yakalama filtreleri) hem de yakalanmış verilerin incelenmesi sırasında (görüntüleme filtreleri) kullanılacak kapsamlı ve özelleştirilmiş Wireshark filtreleri oluşturma hedefi, bu raporda sunulacak teknikler

ve trendler için bir temel oluşturacaktır.

1.3. Raporun Yapısı

Bu rapor, 2025 yılı ve sonrası için ağdaki yazılımcıları Wireshark kullanarak tespit etmeye yönelik belirlenen en son ve en etkili ilk 10 tekniği ve trendi sunmaktadır. Her bir teknik/trend için kısa ve öz bir başlık, ne olduğu, nasıl çalıştığı ve neden önemli olduğuna dair bir açıklama, 2025'teki potansiyel etkileri ve uygulama alanları ve mümkün olan durumlarda güvenilir bir kaynak veya referans sağlanacaktır. Bu yapı, kullanıcının "İstenen Çıktı Detayları" bölümündeki taleplerini karşılamak üzere tasarlanmıştır.

2. Ağdaki Yazılımcıları Wireshark ile Tespit Etmek İçin 2025 Yılına Yönelik En İyi 10 Teknik ve Trend

2.1. Gelişmiş Şifreli Trafik Analizi ve TLS Parmak İzi (JA3/JA4+)

- **Açıklama:**
  - **Ne Olduğu:** Gelişmiş Şifreli Trafik Analizi, ağ trafiğinin büyük bir kısmının şifreli olduğu günümüz ortamında [2], şifreyi kırmadan (veya bilinen anahtarlarla çözerek) meta verileri, el sıkışma parametrelerini ve davranışsal özellikleri analiz ederek tehditleri ve belirli aktörleri (bu durumda yazılımcıları) tespit etme yöntemidir.[6] TLS (Transport Layer Security) parmak izi, özellikle JA3 ve onun gelişmiş versiyonu olan JA4/JA4+ [8], TLS el sıkışması sırasındaki istemci (Client Hello) ve sunucu (Server Hello - JA3S, JA4S) parametrelerinden (TLS versiyonu, şifreleme takımları, uzantılar, eliptik eğriler vb.) benzersiz bir karma (hash) değer oluşturarak istemci uygulamalarını tanımlar.
  - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, TLS el sıkışma paketlerini etkili bir şekilde ayrıştırabilir. JA3 parmak izi, Wireshark'ta tls.handshake.extensions_alpn_str gibi alanlar kullanılarak veya özel Lua betikleriyle manuel olarak hesaplanabilir. JA4/JA4+ [10], tarayıcıların TLS uzantı sıralamasını rastgeleleştirmesi gibi [9] parmak izi kaçırma tekniklerine karşı daha dirençlidir ve Uygulama Katmanı Protokol Anlaşması (ALPN) gibi ek boyutlar içerir. Wireshark, bu parametreleri yakalayarak bilinen geliştirici araçlarının veya bu araçların kullandığı kütüphanelerin (örneğin, Go veya Python tabanlı araçlar [10]) JA4 parmak izleriyle karşılaştırma yapılmasına olanak tanır. SSH trafiği için, SSLKEYLOGFILE yöntemi [11] kullanılarak oturum anahtarları elde edilirse, şifreli SSH trafiği Wireshark'ta çözülebilir ve içindeki komutlar veya protokol etkileşimleri (örneğin, Git komutları) analiz edilebilir.
  - **Neden Önemli Olduğu (2025 için):** Şifreli trafik, ağ iletişiminin standart bir parçası haline gelmiştir.[6] Bu durum, yazılımcıların kullandığı araçların (IDE'ler,

API istemcileri, Git istemcileri) ve servislerin (kod depoları, CI/CD sunucuları, bulut hizmetleri) büyük çoğunluğunun iletişimlerini TLS veya SSH kullanarak şifreleyeceği anlamına gelir. JA4+'ın [10] geliştirilmesi, parmak izi tabanlı tespit yöntemlerinin sürekli bir evrim içinde olduğunu ve tespit kaçırma çabalarına karşı daha dayanıklı hale geldiğini göstermektedir. Bu, yazılımcıların kullandığı spesifik TLS kütüphanelerini veya istemci uygulamalarını, şifreli trafik içinden bile ayırt etmede kritik bir rol oynayacaktır.

- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
  - Yazılımcıların kullandığı spesifik araçların (örneğin, belirli bir programlama dili sürümüyle gelen TLS kütüphanesi, özel bir Git istemcisi, belirli bir IDE eklentisi) veya işletim sistemlerinin şifreli bağlantılarının yüksek doğrulukla tespiti.
  - Geliştirme ortamlarına yönelik hedefli saldırıların veya yetkisiz erişim girişimlerinin, anormal TLS/SSH parmak izleri aracılığıyla erken tespiti.
  - Kurumsal ağlarda kullanılan, onaylanmamış veya güncel olmayan geliştirici araçlarının ya da kütüphanelerinin, parmak izleri aracılığıyla belirlenmesi ve potansiyel uyumluluk veya güvenlik risklerinin saptanması.
  - SSH oturum anahtarları elde edilebildiğinde, Git operasyonlarının (commit mesajları, push/pull edilen dosyaların meta verileri) veya SSH üzerinden yapılan diğer geliştirici aktivitelerinin (uzak sunucularda komut çalıştırma, dosya transferleri) ayrıntılı analizi.
- **Kaynak/Referans:**
  - JA4/JA4+ için: FoxIO-LLC [9], Cloudflare Blog.[10]
  - SSH Şifre Çözme için: Wireshark Wiki.[11]
  - Genel Şifreli Trafik Analizi: Progress Flowmon [7], Group-IB.[6]
- **Derinlemesine Analiz ve Öngörüler:**
  - Şifrelemenin yaygınlaşması, ağ trafiği analizinde geleneksel içerik tabanlı yöntemlerin etkinliğini azaltmıştır. Bu durum, şifreleme sürecinin kendisinden – özellikle el sıkışma aşamasındaki meta verilerden ve davranışsal özelliklerden – yeni sinyaller ve parmak izleri çıkarma çabalarını hızlandırmıştır. JA3'ten JA4'e [9] olan evrim, bu alandaki tespit ve kaçınma teknikleri arasındaki sürekli "kedi-fare" oyununu ve adaptasyon sürecini açıkça göstermektedir. Yazılımcıların kullandığı araçlar ve kütüphaneler güncellendikçe veya değiştikçe, bu araçların TLS/SSH parmak izleri de değişebilir. Örneğin, bir IDE'nin yeni bir sürümü farklı bir TLS kütüphanesi kullanmaya başlarsa veya varsayılan şifreleme takımlarını değiştirirse, bu durum mevcut JA3/JA4 parmak izlerini geçersiz kılabilir. Bu, "Developer Hunter" gibi projelerin, sürekli güncellenen bir parmak izi veritabanına sahip olması veya bu değişikliklere dinamik olarak uyum sağlayabilen makine öğrenimi tabanlı yaklaşımlar kullanması gerektiğini ima eder. TLS parmak izi tekniklerinin daha sofistike hale

gelmesi, yazılım geliştiricilerini ve araç üreticilerini, araçlarının ağ parmak izlerini daha jenerik hale getirmeye veya kasıtlı olarak sık sık değiştirmeye teşvik edebilir. Bu durum, parmak izi tabanlı tespitin zorluğunu artırarak, daha karmaşık davranışsal analizlere ve çoklu göstergelerin birleştirilmesine olan ihtiyacı pekiştirecektir.

- **Wireshark Filtre Önerileri:**
  - TLS İstemci Merhaba (Client Hello) paketlerini yakalamak için: tls.handshake.type == 1
  - Belirli JA3 karmalarını aramak için (eğer biliniyorsa ve bir araç tarafından hesaplanıyorsa veya özel bir sütun olarak eklenmişse): tls.ja3_hash == "abcdef123..." (Not: tls.ja3_hash standart bir Wireshark alanı değildir; özel bir Lua eklentisi veya harici bir araçla entegrasyon gerektirebilir. Ancak, JA3'ü oluşturan temel parametreler filtrelenebilir: örn. tls.handshake.extensions_server_name, tls.handshake.ciphersuite, tls.handshake.extension.type).
  - SSH trafiği için: tcp.port == 22.[11] Şifre çözme etkinleştirilmişse, protokol ssh olarak görünecektir.
  - Belirli ALPN (Application-Layer Protocol Negotiation) değerlerini aramak (JA4'ün önemli bir bileşeni [10]): tls.handshake.extensions_alpn_str == "h2" (HTTP/2 için) veya tls.handshake.extensions_alpn_str == "http/1.1". Geliştirici araçlarına özgü olabilecek diğer ALPN dizeleri de hedeflenebilir.

2.2. Geliştirici Araç Zinciri Trafiği için Makine Öğrenimi Destekli Davranışsal Analiz

- **Açıklama:**
  - **Ne Olduğu:** Bu teknik, yazılımcıların kullandığı araçların (IDE'ler, derleyiciler, test araçları, sürüm kontrol sistemleri, CI/CD araçları) ve tipik iş akışlarının (kod yazma, derleme, test etme, hata ayıklama, dağıtma) oluşturduğu karmaşık ağ trafiği desenlerini öğrenmek ve modellemek için makine öğrenimi (ML) algoritmalarını kullanır. Temel amaç, "normal" yazılımcı davranışının bir temel çizgisini (baseline) oluşturmak ve bu temel çizgiden önemli ölçüde sapan anormal veya şüpheli aktiviteleri tespit etmektir.[1]
  - **Nasıl Çalıştığı (Wireshark ile):** Wireshark ile yakalanan ham ağ trafiği verileri (paket başlıkları, akış meta verileri, protokol etkileşim sıralamaları, bağlantı süreleri, veri hacimleri vb.) ML modelleri için özellik (feature) vektörlerine dönüştürülür. Kullanılabilecek özellikler arasında paket boyut dağılımları, paketler arası zaman aralıkları, belirli sunucularla (örn. kod depoları, geliştirme sunucuları, API uç noktaları) iletişim sıklığı, kullanılan port numaraları ve hatta şifreli trafik meta verileri (örneğin, JA3/JA4 karmaları, TLS sertifika bilgileri) bulunabilir.[14] Bu özellikler kullanılarak, bilinen yazılımcı ve yazılımcı olmayan

trafikle etiketlenmiş veri kümeleriyle denetimli öğrenme (supervised learning) algoritmaları (örneğin, Random Forest, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Gradient Boosting [14]) eğitilebilir veya etiketlenmemiş veriler üzerinde normal davranış profilleri oluşturup aykırılıkları tespit etmek için denetimsiz öğrenme (unsupervised learning) teknikleri (örneğin, kümeleme, anomali tespiti [15]) uygulanabilir.
    - **Neden Önemli Olduğu (2025 için):** Yazılımcı iş akışları ve kullandıkları araç zincirleri giderek daha karmaşık, çeşitli ve dağıtık hale gelmektedir. Bu durum, statik kurallara veya basit filtrelere dayalı geleneksel tespit yöntemlerinin etkinliğini azaltmaktadır. Makine öğrenimi, bu dinamik ve karmaşık ortamlara uyum sağlayabilen, yeni ve daha önce görülmemiş yazılımcı davranışlarını bile potansiyel olarak tespit edebilen daha esnek ve güçlü bir yaklaşım sunar.[6] Özellikle "sıfır gün" (zero-day) tehditlerinin, sofistike iç tehditlerin veya geliştiricilerin normalden sapan ancak kötü amaçlı olmayan ancak yine de riskli olabilecek (örneğin, hassas verilerin yanlışlıkla herkese açık bir depoya gönderilmesi) aktivitelerinin tespitinde kritik bir öneme sahip olacaktır.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Yazılımcıların normal çalışma saatleri, tipik proje sorumlulukları veya coğrafi konumları dışındaki şüpheli ağ aktivitelerinin (örneğin, gece geç saatlerde beklenmedik bir kod deposuna büyük miktarda veri yüklenmesi) tespiti.
    - Farklı yazılımcı rollerinin (örneğin, frontend geliştirici, backend geliştirici, DevOps mühendisi, veritabanı yöneticisi) veya farklı ekiplerin kendilerine özgü ağ davranış profillerinin çıkarılması ve bu profillere göre özelleştirilmiş güvenlik politikalarının veya uyarı mekanizmalarının uygulanması.
    - Geliştirme ortamlarından veya test sunucularından hassas verilerin (kaynak kodu, API anahtarları, müşteri verileri) sızdırılma girişimlerinin veya yetkisiz dış kaynaklara (örneğin, kişisel bulut depolama) erişimin, normal veri akış desenlerinden sapmalar yoluyla tespiti.
    - Otomatikleştirilmiş tehdit avcılığı (threat hunting) ve olay müdahale (incident response) süreçlerinin, ML tarafından işaret edilen şüpheli yazılımcı aktivitelerine öncelik verilerek iyileştirilmesi.
- **Kaynak/Referans:**
    - "Network Intrusion Detection Using Wireshark and Machine Learning" [14] - Makine öğrenimi modellerinin Wireshark verileriyle kullanımı ve çeşitli algoritmaların değerlendirilmesi.
    - USENIX NSDI'20 tpprof makalesi [13] - Ağ trafiği desenlerini profilleme ve "network states" ile "traffic pattern subsequences" kavramları.
    - Genel Ağ Trafik Analizi ve Makine Öğrenimi: Group-IB [6], SOCWISE.[15]
- **Derinlemesine Analiz ve Öngörüler:**

- Güvenlik analizinde, reaktif (olay sonrası müdahale) yaklaşımlardan, proaktif (tehditleri önceden tahmin etme ve önleme) ve öngörücü (gelecekteki potansiyel riskleri modelleme) yaklaşımlara doğru belirgin bir kayış gözlemlenmektedir. Makine öğrenimi, bu dönüşümün merkezinde yer almakta ve büyük, karmaşık ağ trafiği veri kümelerinden anlamlı içgörüler ve desenler çıkararak insan analistlerin yeteneklerini ve verimliliğini artırmaktadır. Geleneksel imza tabanlı tespit sistemleri, daha önce bilinmeyen veya hızla değişen tehditlere ve davranışlara karşı genellikle yetersiz kalmaktadır.[14] Yazılımcı davranışları da doğası gereği dinamik, çeşitli ve sürekli evrilen bir yapıya sahiptir. ML modelleri, bu dinamik ve çeşitli davranışlardan "normal" kabul edilen kalıpları öğrenebilir ve bu kalıplardan önemli sapmaları anomali olarak işaretleyebilir.[1] Bu yetenek, sadece bilinen kötü amaçlı yazılımları veya saldırı tekniklerini değil, aynı zamanda anormal ancak meşru olmayan iç aktiviteleri (örneğin, bir geliştiricinin yetkisi dışındaki sistemlere erişmeye çalışması) veya daha önce hiç karşılaşılmamış yeni saldırı vektörlerini de tespit etme potansiyeli sunar.
- Etkili ve güvenilir ML modelleri oluşturmanın önündeki en büyük engellerden biri, yüksek kaliteli, doğru bir şekilde etiketlenmiş ve temsili veri kümelerine olan ihtiyaçtır. "Developer Hunter" projesi özelinde, hem yazılımcıların hem de yazılımcı olmayan personelin ağ trafiğini içeren, ayrıca "normal" ve "anormal" yazılımcı aktivitelerini ayırt edebilen etiketlere sahip veri kümelerinin oluşturulması ve sürekli olarak güncel tutulması, önemli bir zorluk ve devam eden bir çaba gerektirecektir. Veri toplama, temizleme, özellik mühendisliği ve doğru etiketleme süreçleri, zaman alıcı ve alan uzmanlığı gerektiren adımlardır.[14] Veri setindeki olası yanlılıklar (bias) veya eksiklikler, eğitilen modelin gerçek dünya performansını olumsuz etkileyerek yanlış pozitif (false positive) veya yanlış negatif (false negative) oranlarının artmasına neden olabilir.
- ML tabanlı tespit sistemlerinin ağ güvenliği alanında daha yaygın bir şekilde benimsenmesi, "Açıklanabilir Yapay Zeka (XAI)" tekniklerine olan talebi de beraberinde getirecektir. Güvenlik analistleri ve olay müdahale ekipleri, bir ML modelinin neden belirli bir ağ trafiği akışını veya kullanıcı davranışını "şüpheli yazılımcı aktivitesi" olarak sınıflandırdığını veya işaretlediğini anlamak ve doğrulamak isteyecektir. Özellikle derin öğrenme modelleri gibi bazı ML yaklaşımları, doğaları gereği "kara kutu" (black box) olarak çalışabilir ve karar verme süreçleri insanlar için kolayca yorumlanabilir olmayabilir. Bir güvenlik olayı durumunda, analistlerin sadece bir alarm almak yerine, bu alarmın arkasındaki nedenleri, modelin hangi özelliklere veya desenlere dayanarak bu kararı verdiğini anlaması kritik öneme sahiptir. XAI teknikleri, bu şeffaflığı

sağlayarak yanlış pozitiflerin azaltılmasına, modelin zayıf yönlerinin belirlenip iyileştirilmesine ve en önemlisi analistlerin sisteme olan güveninin artmasına katkıda bulunabilir.

- **Wireshark Filtre Önerileri:**
  - ML modelleri doğrudan Wireshark arayüzü içinde çalışmaz; bunun yerine, Wireshark ile yakalanan veriler (genellikle.pcap veya.pcapng formatında) bu modellere girdi olarak sağlanır. Özellik çıkarımı (feature extraction) aşaması için önemli olabilecek belirli trafik türlerini veya segmentlerini yakalamak amacıyla kullanılabilecek bazı temel Wireshark filtreleri şunlardır:
    - Belirli geliştirme sunucularına, kod depolarına veya CI/CD sistemlerine giden/gelen trafik: ip.addr == <dev_server_ip_1> | | ip.addr == <git_repo_ip> | | ip.addr == <ci_cd_tool_ip>
    - Yazılımcıların sıkça kullandığı bilinen protokoller: ssh | | http | | https | | dns | | git | | rdp | | smb | | nfs (Projenin özelinde belirlenen diğer protokoller de eklenebilir).
    - Geliştirme veya test amacıyla kullanılan ve genellikle standart olmayan yüksek port numaralarını içeren trafik: (tcp.port >= 10000 && tcp.port <= 12000) | | (udp.port >= 10000 && udp.port <= 12000) (Bu port aralığı örnek olup, organizasyonun veya geliştiricilerin kullandığı spesifik portlara göre ayarlanmalıdır).
    - Belirli bir zaman dilimindeki tüm trafik (davranışsal analiz için zaman serisi verisi oluşturmak amacıyla): Bu, Wireshark'ın yakalama seçenekleri veya editcap gibi komut satırı araçlarıyla zaman damgalarına göre filtreleme ile yapılabilir.

2.3. Entegre Geliştirme Ortamı (IDE) ve Kod Editörü Ağ İmzalarının Derinlemesine Analizi

- **Açıklama:**
  - **Ne Olduğu:** Bu teknik, Visual Studio Code (VS Code), IntelliJ IDEA, Eclipse, Sublime Text gibi popüler Entegre Geliştirme Ortamlarının (IDE'ler) ve gelişmiş kod editörlerinin ağa bağlanırken oluşturduğu benzersiz veya karakteristik trafik desenlerini belirlemeye ve analiz etmeye odaklanır. Bu desenler; yazılım güncellemeleri, eklenti (plugin/extension) marketiyle etkileşimler, telemetri verilerinin gönderilmesi, uzaktan geliştirme (remote development) özelliklerinin kullanılması, lisanslama sunucularıyla iletişim ve bulut tabanlı senkronizasyon hizmetleri gibi çeşitli aktivitelerden kaynaklanabilir.
  - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, IDE'lerin veya kod editörlerinin başlattığı ağ bağlantılarını yakalar. Analiz edilecek temel unsurlar arasında

hedef IP adresleri ve alan adları (örneğin, VS Code için vscode-update.azurewebsites.net, az764295.vo.msecnd.net [17] veya eklenti marketi için marketplace.visualstudio.com [18]; IntelliJ IDEA için JetBrains sunucuları olan plugins.jetbrains.com, account.jetbrains.com [19]), kullanılan ağ protokolleri (genellikle HTTPS, bazen düz HTTP veya özel TCP/UDP protokolleri), HTTP isteklerindeki User-Agent başlıkları ve periyodik olarak gönderilen telemetri verilerinin [21] oluşturduğu trafik örüntüleri (bağlantı sıklığı, veri yükü boyutları) bulunur.
    - **Neden Önemli Olduğu (2025 için):** IDE'ler ve kod editörleri, yazılımcıların günlük iş akışlarının merkezinde yer alan temel araçlardır ve modern sürümleri kaçınılmaz olarak ağ bağlantılarına dayanır. 2025 yılına doğru, IDE'lerin bulut tabanlı özelliklerinin (örneğin, GitHub Codespaces, AWS Cloud9 gibi platformlarla entegrasyonlar veya JetBrains'in uzaktan geliştirme çözümleri [22]) ve yapay zeka destekli kodlama asistanlarının (örneğin, GitHub Copilot, Amazon CodeWhisperer) daha da yaygınlaşması beklenmektedir. Bu gelişmeler, bu araçların ağ üzerindeki imzalarını daha belirgin, çeşitli ve potansiyel olarak daha kolay tespit edilebilir hale getirecektir. Bu durum, IDE ve editör trafiğini, yazılımcı varlığını ve aktivitesini saptamak için zengin bir bilgi kaynağına dönüştürmektedir.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Ağ üzerinde belirli IDE'lerin veya kod editörlerinin kullanımının (ve hatta bazen sürümlerinin) tespiti, bu sayede potansiyel yazılımcı varlığının ve kullanılan geliştirme araçlarının belirlenmesi.
    - Kurumsal güvenlik politikalarına veya lisans anlaşmalarına aykırı IDE, kod editörü veya eklenti kullanımının tespiti ve raporlanması.
    - IDE'ler veya eklentileri üzerinden gerçekleşebilecek potansiyel veri sızıntısı (örneğin, kaynak kodunun veya hassas konfigürasyon dosyalarının yetkisiz bir hedefe gönderilmesi) veya zararlı eklenti indirme gibi güvenlik olaylarının izlenmesi ve analizi.
    - Uzaktan geliştirme ortamlarına (örneğin, bir sunucuda çalışan IDE backend'ine) bağlanan yazılımcıların ağ aktivitesinin analizi ve bu bağlantıların güvenliğinin değerlendirilmesi.
- **Kaynak/Referans:**
    - VS Code Telemetri/Güncellemeler: VS Code Belgeleri [21], Stack Overflow [17], VS Code Marketplace Belgeleri.[18]
    - IntelliJ IDEA Telemetri/Güncellemeler: JetBrains Belgeleri [22], JetBrains Destek Forumları.[19]
    - Genel HTTP Trafik Analizi: LabEx.[23]
- **Derinlemesine Analiz ve Öngörüler:**

- Modern IDE'ler, artık sadece yerel makinelerde çalışan bağımsız araçlar olmaktan çıkarak, bulut servisleriyle (eklenti marketleri, ayar senkronizasyonu, uzaktan geliştirme sunucuları, yapay zeka tabanlı kod asistanları vb.) derinlemesine entegre olan karmaşık platformlara dönüşmektedir. Bu evrim, IDE'lerin ağa olan bağımlılığını ve dolayısıyla ağ üzerindeki görünür ayak izlerini önemli ölçüde artırmaktadır. Bir IDE'nin başlatılması, kullanılması ve kapatılması sırasında, güncelleme kontrolleri, eklenti bildirimleri, telemetri gönderimi, lisans doğrulaması ve bulut tabanlı hizmetlerle senkronizasyon gibi birçok farklı ağ etkileşimi gerçekleşebilir.[18] Bu etkileşimlerin her biri, zamanlama, hedef sunucular, veri boyutları ve protokol özellikleri açısından analiz edilebilecek potansiyel imzalar taşır.
- Bir IDE'nin telemetri ayarlarının kullanıcı tarafından değiştirilmesi, örneğin gizlilik endişeleriyle tamamen kapatılması [21], bu spesifik ağ imzasının kaybolmasına neden olabilir. Ancak, yazılım güncellemeleri, eklenti marketi erişimi ve lisanslama gibi IDE'nin temel işlevselliği için gerekli olan diğer ağ iletişimleri genellikle devam edecektir. Bu durum, tek bir ağ imzasına (örneğin, sadece telemetri trafiğine) güvenmek yerine, birden fazla farklı göstergeyi (örneğin, belirli güncelleme sunucusu IP adresleri, eklenti marketi alan adlarına yapılan DNS sorguları, bu sunucularla kurulan TLS bağlantılarının parmak izleri) birleştiren daha kapsamlı ve katmanlı bir tespit stratejisinin gerekliliğini ortaya koymaktadır.
- IDE üreticileri, altyapılarını iyileştirmek, hizmetlerini coğrafi olarak dağıtmak veya potansiyel olarak tespit edilmeyi zorlaştırmak amacıyla sunucu IP adreslerini veya kullandıkları alan adlarını periyodik olarak değiştirebilirler. Bu durum, statik IP adresi veya alan adı listelerine dayalı olarak oluşturulmuş Wireshark filtrelerinin zamanla etkinliğini yitirmesine neden olabilir. Bu zorluğun üstesinden gelmek için, belirli IP'ler yerine daha genel üst düzey alan adlarına (örneğin, *.jetbrains.com, *.visualstudio.com) yapılan DNS sorgularının (dns.qry.name contains "jetbrains") veya bu alan adlarıyla ilişkili TLS sertifikalarındaki Subject Alternative Name (SAN) veya Common Name (CN) alanlarının (tls.handshake.certificate) izlenmesi daha dayanıklı ve uzun ömürlü bir tespit yaklaşımı sunabilir. Ayrıca, IDE'lerin kullandığı bilinen User-Agent dizgelerindeki değişiklikler de takip edilmelidir.
- **Wireshark Filtre Önerileri:**
  - VS Code güncelleme ve eklenti marketi trafiği için (HTTPS trafiği varsayılarak) [17]:
    - tls.handshake.extensions_server_name contains "vscode-update.azurewebsites.net" || tls.handshake.extensions_server_name contains

"az764295.vo.msecnd.net" | | tls.handshake.extensions_server_name contains "marketplace.visualstudio.com"
    - IntelliJ IDEA (ve diğer JetBrains ürünleri) güncelleme, eklenti marketi ve hesap senkronizasyon trafiği için [19]:
        - tls.handshake.extensions_server_name contains "jetbrains.com" | | tls.handshake.extensions_server_name contains "jetbrains.ai" | | tls.handshake.extensions_server_name contains "intellij.net" | | tls.handshake.extensions_server_name contains "grazie.aws.intellij.net"
    - Genel IDE telemetri desenleri (genellikle periyodik, nispeten küçük boyutlu HTTPS POST istekleri şeklinde olabilir, ancak bu çok genel bir filtredir ve diğer uygulamalarla karışabilir; belirli hedef IP'ler veya User-Agent kalıplarıyla birleştirilmesi önerilir):
        - http.request.method == "POST" && ssl.record.content_type == 23 && frame.len < 1500 (Bu filtre, HTTPS üzerinden yapılan ve toplam çerçeve boyutu 1500 bayttan küçük olan POST isteklerini hedefler. Daha spesifik hale getirmek için ip.dst veya http.user_agent ile AND'lenmelidir.)
    - VS Code telemetri log dosyasının incelenmesi için (Wireshark doğrudan bunu yapmaz, ancak VS Code içinden etkinleştirilebilir [21]): VS Code'da Developer: Show Telemetry komutu telemetri olaylarını bir çıktı kanalında gösterir ve telemetry.log dosyasına yazar. Bu loglar, VS Code'un hangi tür bilgileri gönderdiğini anlamak için kullanılabilir ve bu bilgiler ağ trafiğiyle ilişkilendirilebilir.

2.4. API Test Araçlarının (Postman, Insomnia) Ağ Davranışlarının ve User-Agent Parmak İzlerinin Gelişmiş Analizi

- **Açıklama:**
    - **Ne Olduğu:** Bu teknik, yazılımcılar tarafından API geliştirme, test etme ve dokümantasyon süreçlerinde yaygın olarak kullanılan Postman ve Insomnia gibi API test araçlarının ağ trafiği özelliklerini derinlemesine incelemeyi içerir. Analiz, özellikle bu araçların HTTP/HTTPS isteklerinde kullandığı User-Agent başlıklarına, bağlantı kurdukları tipik uç noktalara (genellikle yerel geliştirme ortamları, staging sunucuları veya dahili API'ler) ve istek/cevap döngülerinin yapısal ve davranışsal özelliklerine odaklanır.
    - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, Postman veya Insomnia gibi araçlardan kaynaklanan HTTP ve HTTPS isteklerini yakalar. Bu araçların varsayılan User-Agent başlıkları (örneğin, Postman için genellikle PostmanRuntime/x.y.z formatında bir dize [24]) önemli bir ilk göstergedir. Ancak,

bu User-Agent başlıkları kullanıcılar tarafından kolayca özelleştirilebilir [26], bu nedenle tespit sadece bu başlığa dayanmamalıdır. Daha kapsamlı bir analiz, User-Agent'ın yanı sıra, isteklerin yapıldığı hedef IP adresleri (genellikle özel ağ aralıkları olan 127.0.0.1, 192.168.x.x, 10.x.x.x veya geliştirme ortamlarına ait diğer IP'ler), isteklerin sıklığı ve zamanlaması, kullanılan HTTP metodları (GET, POST, PUT, DELETE, PATCH, OPTIONS vb. [28]) ve potansiyel olarak API anahtarları, yetkilendirme token'ları (örneğin, Authorization: Bearer...) veya özel başlıklar (X-Custom-Header) içeren başlıkların varlığı gibi davranışsal özelliklere de odaklanmalıdır.
    - **Neden Önemli Olduğu (2025 için):** API'ler, modern yazılım mimarilerinin ve dijital dönüşümün temel yapı taşlarıdır ve API test araçları, yazılımcılar ve QA mühendisleri tarafından geliştirme yaşam döngüsünün ayrılmaz bir parçası olarak yoğun bir şekilde kullanılmaktadır.[29] 2025 yılına doğru, mikroservis mimarilerinin, başsız (headless) CMS'lerin, IoT uygulamalarının ve API odaklı geliştirme pratiklerinin daha da yaygınlaşmasıyla bu araçların kullanımı katlanarak artacaktır. Bu araçların ağ imzalarını doğru bir şekilde belirlemek ve analiz etmek, ağ üzerindeki yazılımcı aktivitesini ve API geliştirme süreçlerini tespit etmede önemli bir rol oynayacaktır. User-Agent başlıklarının özelleştirilebilir olması [26], bu araçları tespit etmek için daha sofistike, çok faktörlü ve davranışsal analiz yöntemlerinin geliştirilmesini zorunlu kılmaktadır.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Ağ üzerinde aktif API geliştirme, test etme veya entegrasyon çalışmalarının tespiti ve bu faaliyetlerin hangi sistemler veya kullanıcılar tarafından gerçekleştirildiğinin belirlenmesi.
    - Geliştirme veya test aşamasında olan, henüz kamuya açık olmayan veya zayıf güvenlik önlemlerine sahip olabilecek API'lere yapılan yetkisiz erişim denemelerinin veya aşırı yükleme testlerinin saptanması.
    - Kurumsal ağlarda kullanılan API test araçlarının (ve potansiyel olarak sürümlerinin) envanterinin çıkarılması, lisans uyumluluğunun takibi ve onaylanmamış araçların kullanımının engellenmesi.
    - API test araçları aracılığıyla, test senaryolarında veya ortam değişkenlerinde saklanan hassas verilerin (API anahtarları, veritabanı bağlantı dizeleri, kullanıcı kimlik bilgileri) yanlışlıkla veya kasıtlı olarak ağ üzerinden sızdırılma potansiyelinin analizi.
- **Kaynak/Referans:**
    - Postman User-Agent: ZenRows Blog [26], BrightData Blog [24], Stack Overflow.[25]
    - Insomnia User-Agent: Insomnia Eklenti Belgeleri.[27] (Not: Insomnia'nın varsayılan User-Agent'ı hakkında net bilgi snippet'lerde az, ancak insomnia-plugin-user-agents [27] gibi eklentilerle değiştirilebildiği belirtiliyor.)

    - Genel API Test Aracı Karşılaştırmaları ve Özellikleri: Abstracta [29], Postman.[30]
- **Derinlemesine Analiz ve Öngörüler:**
    - API test araçları, basit HTTP istek göndericilerinden, karmaşık test senaryoları oluşturma, otomatik testler çalıştırma, sahte (mock) sunucular kurma, ekip içinde işbirliği yapma ve CI/CD süreçleriyle entegre olma gibi yeteneklere sahip tam teşekküllü geliştirme ve test platformlarına doğru evrilmektedir.[30] Bu evrim, bu araçların ağ üzerindeki etkileşimlerinin de daha karmaşık ve çeşitli hale gelmesi anlamına gelir. Örneğin, Postman'in bulut senkronizasyon özellikleri, koleksiyonların ve ortam değişkenlerinin Postman sunucularıyla periyodik olarak eşitlenmesini gerektirir; ekip çalışma alanları ise birden fazla kullanıcının aynı kaynaklar üzerinde eş zamanlı çalışmasına olanak tanır ve bu da ek ağ trafiği üretir. Bu ek ağ etkileşimleri, tespit için daha fazla yüzey alanı sunarken, aynı zamanda analizin karmaşıklığını da artırır.
    - Bir geliştiricinin, kullandığı API test aracındaki (örneğin Postman) User-Agent başlığını, yaygın bir web tarayıcısının User-Agent dizesiyle (örneğin, Chrome veya Firefox) değiştirmesi [26], sadece User-Agent'a dayalı basit filtreleme mekanizmalarını kolayca atlatabilir. Ancak, bu durum tespitin imkansız olduğu anlamına gelmez. İsteklerin yapıldığı kaynak ve hedef IP adresleri (genellikle geliştirme ortamlarına özgü özel IP aralıkları veya localhost), isteklerin frekansı ve zamanlaması (örneğin, bir API'nin farklı uç noktalarına kısa aralıklarla çok sayıda istek gönderilmesi), kullanılan HTTP metodlarının çeşitliliği (özellikle PUT, PATCH, DELETE gibi veri değiştiren metodların yoğun kullanımı) ve hedef API'lerin doğası (örneğin, henüz kamuya açık olmayan, DNS kaydı bulunmayan veya sadece belirli IP'lerden erişilebilen API'ler) gibi diğer davranışsal özellikler, bu aktivitenin büyük olasılıkla bir API test aracından kaynaklandığına dair güçlü ipuçları sunabilir. Bu tür çok faktörlü analiz, User-Agent aldatmacalarına karşı daha dirençlidir.
    - API güvenliğinin ve API ağ geçitlerinin (API Gateways) kurumsal mimarilerde giderek daha merkezi bir rol oynamasıyla birlikte, API test araçlarının bu güvenlik mekanizmalarıyla (örneğin, OAuth 2.0 yetkilendirme akışları, API anahtarı tabanlı kimlik doğrulama, JWT token'ları) nasıl etkileşim kurduğuna dair ağ trafiği desenleri de analiz için önemli bir veri kaynağı haline gelecektir. Yanlış yapılandırılmış test senaryoları, süresi dolmuş veya sızdırılmış kimlik bilgilerinin kullanımı, veya zayıf şifreleme yöntemleriyle yapılan bağlantılar bu trafik akışlarından tespit edilebilir. Örneğin, bir API test aracının bir OAuth 2.0 token uç noktasına sık sık istek yapması veya HTTP istek başlıklarında Authorization alanını taşıması [32], API testi aktivitesine işaret eder. Bu etkileşimlerin detaylı analizi, sadece aracın kullanımını değil, aynı zamanda potansiyel güvenlik yapılandırma hatalarını veya güvenlik politikalarının

ihlallerini de ortaya çıkarabilir.
    - **Wireshark Filtre Önerileri:**
        - Varsayılan Postman User-Agent için (sürüm numarası değişebilir): http.user_agent contains "PostmanRuntime" [24]
        - Insomnia için (varsayılan User-Agent'ı insomnia/ ile başlayabilir veya bir eklenti ile değiştirilmiş olabilir [27]; bu nedenle daha genel bir yaklaşım veya bilinen eklenti imzaları gerekebilir): http.user_agent contains "insomnia/" (Bu, teyit edilmesi gereken bir varsayımdır, çünkü snippet'ler net bir varsayılan UA belirtmemektedir.)
        - API testlerinde sıkça kullanılan HTTP metodları (özellikle yerel veya özel IP adreslerine yönelik): (ip.dst == 127.0.0.1 | | ip.dst_host matches "^192\\.168\\..*" | | ip.dst_host matches "^10\\..*" | | ip.dst_host matches "^172\\.(1[6-9]|2[0-9]|3[0-1])\\..*") && (http.request.method == "POST" | | http.request.method == "PUT" | | http.request.method == "DELETE" | | http.request.method == "PATCH")
        - API iletişimlerinde yaygın olarak kullanılan içerik türleri (Content-Type) için filtreler: http.content_type contains "application/json" | | http.content_type contains "application/xml" | | http.content_type contains "application/x-www-form-urlencoded"
        - Authorization başlığını içeren HTTP istekleri (API anahtarları veya token taşıyabilir): http.authorization

2.5. Sürüm Kontrol Sistemi (Git/SSH) Etkileşimlerinin Ayrıntılı Protokol Analizi

- **Açıklama:**
    - **Ne Olduğu:** Bu teknik, yazılımcıların Git gibi dağıtık sürüm kontrol sistemleriyle (Version Control Systems - VCS) etkileşimde bulunurken oluşturduğu ağ trafiğini derinlemesine analiz etmeye odaklanır. Temel amaç, özellikle SSH (Secure Shell, varsayılan olarak TCP port 22) veya HTTPS (HTTP Secure, varsayılan olarak TCP port 443) üzerinden gerçekleştirilen Git operasyonlarını (örneğin, git clone, git fetch, git pull, git push) ve bu operasyonların altında yatan protokol etkileşimlerini tespit etmektir.
    - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, TCP port 22 (SSH için) veya TCP port 443 (HTTPS üzerinden Git için) üzerindeki ağ trafiğini yakalar. SSH üzerinden yapılan Git trafiği için, eğer SSLKEYLOGFILE benzeri bir mekanizma veya önceden paylaşılan anahtarlar (pre-shared keys) kullanılarak oturum anahtarları elde edilebilirse [11], Wireshark şifreli SSH yükünü çözebilir. Bu durumda, Git protokolüne özgü komutlar veya SSH alt sistem istekleri (örneğin, git-upload-pack veya git-receive-pack gibi servis istekleri) paket ayrıntılarında okunabilir hale gelir. Şifre çözme mümkün olmasa bile, SSH bağlantılarının

sıklığı, transfer edilen veri miktarları, SSH el sıkışması sırasındaki parametreler (kullanılan anahtar değişim algoritmaları, sunucu anahtar türleri, şifreleme ve MAC algoritmaları) ve bağlantı kurulan hedef sunucular (örneğin, GitHub, GitLab, Bitbucket gibi genel hizmetlerin IP adresleri veya kuruma özel Git sunucularının IP'leri) analiz edilebilir. HTTPS üzerinden Git trafiği için ise, standart TLS el sıkışma analizi (JA3/JA4 parmak izleri dahil) ve özellikle TLS SNI (Server Name Indication) uzantısında belirtilen hedef alan adı (örneğin, github.com, gitlab.mycompany.com) değerli bilgiler sunar.
    - **Neden Önemli Olduğu (2025 için):** Git, modern yazılım geliştirme süreçlerinin temel ve vazgeçilmez bir parçası haline gelmiştir. 2025 yılında, dağıtık geliştirme ekiplerinin, bulut tabanlı kod depolarının (cloud-hosted repositories) ve GitOps gibi otomasyon pratiklerinin daha da yaygınlaşmasıyla birlikte, ağ üzerindeki Git trafiğinin hacmi ve önemi artmaya devam edecektir. SSH anahtar yönetimi ve güvenli bağlantıların sürdürülmesi, kurumsal güvenlik politikalarının önemli bir unsuru olmaya devam edeceğinden, özellikle şifre çözme yetenekleri geliştikçe veya uygun anahtar materyallerine erişim sağlandıkça, SSH üzerinden Git trafiğinin ayrıntılı analizi, yazılımcı aktivitesini ve potansiyel güvenlik risklerini tespit etmede kritik bir rol oynayacaktır.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Ağ üzerinde Git protokolünün (hem SSH hem de HTTPS üzerinden) kullanımının ve dolayısıyla potansiyel yazılımcı aktivitesinin tespiti.
    - Yazılımcıların hangi kod depolarına (genel/halka açık veya özel/kurumsal) ne sıklıkta ve ne tür operasyonlarla (clone, push, pull vb.) eriştiğinin belirlenmesi.
    - Büyük boyutlu veya sık aralıklarla yapılan git push veya git pull operasyonları aracılığıyla, hassas kaynak kodlarının veya diğer önemli verilerin yetkisiz bir şekilde dışarı sızdırılması veya yetkisiz kodun içeri enjekte edilmesi gibi potansiyel veri sızıntısı veya güvenlik ihlali girişimlerinin izlenmesi ve tespiti.
    - Kurumsal güvenlik standartlarına uymayan zayıf SSH şifreleme algoritmalarının, eski protokol versiyonlarının veya güvensiz anahtar türlerinin kullanımının Git etkileşimleri sırasında tespit edilmesi.
- **Kaynak/Referans:**
    - Wireshark SSH Dissector ve Şifre Çözme: Wireshark Wiki.[11]
    - Genel Ağ Trafiği Analizi ve Filtreleme: LabEx.[33]
- **Derinlemesine Analiz ve Öngörüler:**
    - "Her şey kod olarak" (Everything as Code - XaC) yaklaşımının ve GitOps gibi modern dağıtım ve operasyon pratiklerinin yazılım geliştirme ekosistemlerinde giderek daha fazla benimsenmesi, Git protokolünün kullanım alanını önemli ölçüde genişletmektedir. Git artık sadece uygulama kaynak kodunu yönetmek için değil, aynı zamanda altyapı yapılandırmalarını (Infrastructure as Code -

IaC), güvenlik politikalarını (Policy as Code), dağıtım manifestolarını (örneğin, Kubernetes YAML dosyaları), veritabanı şemalarını (Schema as Code) ve hatta dokümantasyonu (Docs as Code) sürümlemek ve yönetmek için de merkezi bir araç olarak kullanılmaktadır. Bu durum, ağ üzerindeki Git trafiğinin sadece yazılımcıların manuel kodlama aktivitelerinden değil, aynı zamanda otomatikleştirilmiş CI/CD (Sürekli Entegrasyon/Sürekli Dağıtım) pipeline'larından, yapılandırma yönetimi araçlarından ve çeşitli otomasyon sistemlerinden de kaynaklanabileceği anlamına gelir. Bu, "yazılımcı aktivitesi" tanımını genişletir ve tespit edilen Git trafiğinin bağlamını (örneğin, bir insan kullanıcı mı yoksa bir otomasyon aracı mı tarafından başlatıldığı) anlamayı daha da önemli hale getirir.

- Birçok kuruluş, güvenlik ve erişim kontrolü amacıyla, iç ağdaki kritik sunuculara (Git depolarını barındıran sunucular dahil) doğrudan SSH erişimini kısıtlar ve bunun yerine tüm SSH bağlantılarının merkezi bir atlama sunucusu (jump server veya bastion host) üzerinden yapılmasını zorunlu kılar. Bu tür bir mimaride, ağın kenarında (perimeter) Wireshark ile yakalanan SSH trafiği, genellikle sadece geliştiricinin makinesi ile atlama sunucusu arasındaki bağlantıyı gösterecektir; Git deposu sunucusuna olan nihai bağlantı bu ilk SSH tüneli içinde kapsüllenmiş olabilir. Bu durumda, bir geliştiricinin hangi spesifik Git deposuna eriştiğini anlamak için, atlama sunucusundaki bağlantı logları (kimin, ne zaman, hangi iç hedefe bağlandığına dair kayıtlar) ile Wireshark ile yakalanan ağ trafiği verilerinin birleştirilmesi ve korelasyonu gerekebilir. Bu, tespit sürecine ek bir karmaşıklık katmanı ekler.
- Geleneksel uzun ömürlü SSH anahtar çiftlerinin yönetimi (özellikle authorized_keys dosyalarının bakımı) operasyonel zorluklar ve potansiyel güvenlik riskleri (örneğin, özel anahtarların sızması, eski çalışanların erişiminin iptal edilmemesi) taşıyabilir. Bu nedenle, SSH sertifikalarının veya HashiCorp Vault gibi araçlarla yönetilen kısa ömürlü, dinamik olarak oluşturulan SSH anahtarlarının kullanımı giderek daha popüler hale gelmektedir. Bu modern yaklaşımlar, SSH erişim güvenliğini artırırken, belirli bir SSH bağlantısını statik bir anahtar veya kullanıcıyla uzun süreli olarak ilişkilendirmeyi zorlaştırabilir. Ancak, bu durum genel SSH protokolü kullanımının tespitini engellemez. Tespit stratejileri, bireysel anahtarlardan ziyade, SSH protokolünün kendisinin (el sıkışma desenleri, trafik hacmi, bağlantı süreleri) ve bağlantı kurulan hedef sunucuların (bilinen Git sunucuları veya kurumsal atlama sunucuları) analizine odaklanabilir.
- **Wireshark Filtre Önerileri:**
  - Standart SSH trafiği için: tcp.port == 22 [11]
  - HTTPS üzerinden Git trafiği için (GitHub, GitLab, Bitbucket gibi genel

hizmetlere yapılan bağlantılarda SNI alanını kontrol ederek): tls.handshake.extensions_server_name contains "github.com" | | tls.handshake.extensions_server_name contains "gitlab.com" | | tls.handshake.extensions_server_name contains "bitbucket.org"
    - SSH el sıkışması sırasında kullanılan anahtar değişim (kex) algoritmalarını veya sunucu anahtar türlerini incelemek (potansiyel olarak istemci veya sunucu yazılımını parmak izi almak için): ssh.protocol contains "kex" veya daha spesifik olarak ssh.message_code == 20 (SSH_MSG_KEXINIT paketlerini gösterir, bu paketler algoritma listelerini içerir). Bu paketlerin yükü daha sonra manuel olarak veya bir Lua betiği ile analiz edilebilir.
    - Büyük hacimli veri transferlerini (potansiyel olarak büyük git clone veya git push işlemleri) SSH üzerinden tespit etmek için (eşik değeri ağın normal kullanımına göre ayarlanmalıdır): tcp.port == 22 && tcp.len > 10000 (Bu, tek bir TCP segmentinin boyutunu gösterir; daha uzun süreli yüksek hacimli akışlar için Wireshark'ın konuşma istatistikleri veya IO grafikleri daha faydalı olabilir.)
    - HTTPS üzerinden Git trafiğinde büyük veri transferleri için benzer bir yaklaşım: tcp.port == 443 && tls.handshake.extensions_server_name contains "git" && tcp.len > 10000 (Bu, SNI'da "git" içeren ve büyük segmentlere sahip TLS trafiğini hedefler.)

2.6. Sanal Makine (VM) ve Konteyner (Docker, Kubernetes) Etkileşimlerinin Ağ Düzeyinde İzlenmesi

- **Açıklama:**
    - **Ne Olduğu:** Bu teknik, yazılımcıların sanal makineler (VM'ler) ve özellikle Docker gibi konteyner teknolojileri ile etkileşimde bulunurken veya Kubernetes gibi konteyner orkestrasyon platformlarında uygulamalar geliştirirken, test ederken ve dağıtırken oluşturduğu ağ trafiğini analiz etmeyi amaçlar. Bu analiz, VM veya konteyner imajlarının (images) çeşitli depolardan (registries) indirilmesi, konteynerler arası (inter-container) iletişimi, ana makine (host) ile konteynerler arasındaki iletişimi, Kubernetes API sunucusuyla yapılan kontrol düzlemi (control plane) etkileşimlerini ve bu sanallaştırılmış ortamlarda kullanılan sanal ağ arayüzlerindeki (virtual network interfaces) trafiği kapsar.
    - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, sanal makinelerin veya konteynerlerin çalıştığı ana makine (host) üzerinde veya sanal ağ anahtarlarında (virtual switches) ya da ağ geçitlerinde çalıştırılarak VM/konteyner trafiğini yakalayabilir. Docker ortamları için, Docker daemon API'sine yapılan istekler (genellikle yerel olarak bir Unix soketi üzerinden gerçekleşir ancak TCP portu üzerinden de açığa çıkarılabilir – varsayılan olarak

2375 veya TLS ile 2376) veya konteynerlerin kullandığı sanal köprü (bridge) ağlarındaki (örneğin, docker0 arayüzü) trafik izlenebilir. Kubernetes ortamlarında ise, Kubeshark [35] gibi bu ortamlara özel olarak tasarlanmış araçlar, pod'lar arası, servisler arası ve Kubernetes API sunucusuyla olan tüm iletişimi (şifreli trafik dahil) protokol düzeyinde derinlemesine görünürlük sağlayarak Wireshark benzeri bir analiz deneyimi sunar. Kubeshark, zengin bir sorgu dili ve kimlik farkındalığına sahip bir servis haritası gibi özelliklerle analizi kolaylaştırır.[36] Standart Wireshark ile, geliştiricilerin kubectl komutları aracılığıyla Kubernetes API sunucusuna (genellikle HTTPS üzerinden TCP port 6443 veya bulut sağlayıcısına özel portlar üzerinden) yaptığı REST tabanlı API çağrıları veya Kubernetes'in temel bileşenleri olan etcd gibi sistemlerle olan iletişim (eğer bu iletişim ağ üzerinden şifresiz veya şifresi çözülebilir bir şekilde yapılıyorsa) yakalanabilir.
    - **Neden Önemli Olduğu (2025 için):** Konteynerizasyon ve mikroservis mimarileri, modern yazılım geliştirme ve dağıtım süreçlerinin temel standartları haline gelmiştir.[35] 2025 yılına gelindiğinde, Kubernetes'in ekosistemdeki baskınlığının devam etmesi ve serverless container platformları (örneğin, AWS Fargate, Google Cloud Run) ile WebAssembly (Wasm) gibi yeni nesil konteyner teknolojilerinin yükselişiyle, yazılımcıların bu tür dinamik ve soyutlanmış ortamlarla etkileşimi daha da artacak ve çeşitlenecektir. Bu etkileşimlerin ağ düzeyindeki izlerini doğru bir şekilde analiz edebilmek, yazılımcı aktivitesini, geliştirme süreçlerini ve potansiyel güvenlik açıklarını tespit etmek için kritik öneme sahip olacaktır. Kubeshark gibi özel araçların, Agentic-AI (otonom, adaptif ve kendi kendini iyileştiren ağ ajanları) [36] gibi gelişmiş özelliklerle donatılması, bu alandaki tespit ve analiz yeteneklerinde önemli bir otomasyon ve zeka artışı sağlayacaktır.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Geliştirme, test ve hazırlık (staging) amacıyla kullanılan Docker ve Kubernetes ortamlarındaki spesifik yazılımcı aktivitelerinin (örneğin, yeni bir uygulamanın dağıtılması, bir servisin ölçeklendirilmesi, logların çekilmesi) tespiti.
    - Genel veya özel konteyner imaj depolarından (örneğin, Docker Hub, Google Container Registry (GCR), Amazon Elastic Container Registry (ECR), Quay.io) konteyner imajlarının çekilme (pull) veya itilme (push) işlemlerinin izlenmesi ve bu işlemlerin yetkili kullanıcılar tarafından yapılıp yapılmadığının kontrolü.
    - Kubernetes API sunucusuna yapılan yetkisiz, anormal veya şüpheli API çağrılarının (örneğin, beklenmedik bir kaynaktan gizli (secret) bilgilere erişim denemesi) tespiti.
    - Mikroservisler arasındaki anormal iletişim desenlerinin (örneğin, normalde iletişim kurmaması gereken iki servis arasında trafik gözlemlenmesi veya

beklenenden çok daha yüksek hacimli veri transferi) belirlenmesi; bu durum potansiyel güvenlik açıklarını, yanlış yapılandırmaları veya uygulama hatalarını işaret edebilir.
    - Geliştirici araçlarının kendilerinin konteynerize edilmiş versiyonlarının (örneğin, bir Docker konteyneri içinde çalışan bir IDE veya bir test aracı) ağ davranışlarının analizi.
- **Kaynak/Referans:**
    - Kubeshark: Kubeshark Web Sitesi [36], Dev.to Makalesi.[35]
    - Docker Ağ Trafiği Yakalama: Docker Forumları (Edgeshark eklentisi) [37], Hacker News (Subtrace aracı).[38]
- **Derinlemesine Analiz ve Öngörüler:**
    - Geliştirme ortamları, geleneksel yerel makine kurulumlarından giderek daha fazla soyutlanmakta ve dağıtık bir yapıya kavuşmaktadır. Yazılımcılar artık sadece kendi fiziksel makinelerinde değil, aynı zamanda bulutta barındırılan sanal ortamlarda, yerel veya uzak Kubernetes kümelerinde çalışan dinamik ve geçici (ephemeral) geliştirme ortamlarında kod yazmakta, test etmekte ve hata ayıklamaktadır. Bu paradigma kayması, ağ trafiği analizinin odağını, bu sanal, orkestre edilmiş ve genellikle çok katmanlı ağ yapılarına kaydırmaktadır. Bu ortamlardaki trafik, sadece uygulamanın kendisinden değil, aynı zamanda altta yatan sanallaştırma katmanlarından, konteyner çalışma zamanlarından ve orkestrasyon platformlarının kontrol düzlemlerinden de kaynaklanabilir.
    - Kubernetes gibi platformlarda, Ağ Politikalarının (Network Policies) etkin bir şekilde kullanılması, pod'lar ve servisler arasındaki izin verilen iletişim yollarını tanımlayarak bir "normal" trafik davranış temeli oluşturabilir. Bu politikalar, hangi pod'un hangi diğer pod'larla veya ağ uç noktalarıyla hangi portlar ve protokoller üzerinden iletişim kurabileceğini belirler. Bu tanımlanmış politikalara aykırı herhangi bir ağ trafiği (örneğin, bir frontend pod'unun normalde erişmemesi gereken bir veritabanı pod'una doğrudan bağlanmaya çalışması), bir anormallik, potansiyel bir güvenlik ihlali veya bir geliştiricinin yetkisiz bir denemesi olarak değerlendirilebilir. Kubeshark gibi araçlar, bu ağ politikalarının uygulanıp uygulanmadığını izlemeye ve olası ihlalleri tespit etmeye yardımcı olabilir [36]'da belirtilen "ağ politikalarını uygulama" yeteneği bu bağlamda önemlidir. Bu, Wireshark ile toplanan trafik verilerinin, Kubernetes API'sinden alınan ağ politikası tanımlarıyla karşılaştırılarak analiz edilmesi anlamına gelebilir.
    - Servis ağı (Service Mesh) teknolojilerinin (örneğin, Istio, Linkerd, Consul Connect) Kubernetes ortamlarında ve genel olarak mikroservis mimarilerinde yaygınlaşması, ağ trafiği analizine yeni bir boyut ve karmaşıklık katmanı eklemektedir. Service mesh'ler, uygulamalar arasındaki iletişimi yönetmek,

güvenliğini sağlamak ve gözlemlenebilirliğini artırmak için her bir uygulama pod'una genellikle bir yan sepet (sidecar) proxy (örneğin, Envoy) enjekte eder. Tüm uygulama trafiği bu proxy'ler üzerinden akar ve proxy'ler kendi aralarında ve service mesh kontrol düzlemiyle iletişim kurar. Bu durum, ağda gözlemlenen trafiğin büyük bir kısmının artık doğrudan uygulama bileşenleri arasında değil, bu proxy'ler arasında olduğu anlamına gelir. Bu proxy trafiği, istek yönlendirme kararları, güvenlik politikası uygulamaları (örneğin, mTLS şifrelemesi), yeniden deneme mekanizmaları ve dağıtık izleme (distributed tracing) verileri gibi service mesh işlevleri hakkında zengin bilgiler içerir. Kubeshark'ın "kimlik farkındalığına sahip servis haritası" (identity-aware service map) [36] gibi özellikleri, bu karmaşık ve dolaylı iletişim yollarını anlamlandırmaya ve gerçek uygulama etkileşimlerini ortaya çıkarmaya yardımcı olabilir. Yazılımcıların service mesh yapılandırmalarıyla (örneğin, trafik kuralları, sanal servisler, yetkilendirme politikaları) etkileşimi de API sunucusu üzerinden ağ izleri bırakacaktır.

- **Wireshark Filtre Önerileri:**
  - Kubernetes API sunucusu trafiği (genellikle HTTPS üzerinden, varsayılan port 6443 veya bulut sağlayıcısına göre değişebilir): tcp.port == 6443 && tls (Eğer API sunucusu farklı bir port kullanıyorsa, o port numarası belirtilmelidir.)
  - Docker daemon API'sine yapılan istekler (eğer TCP üzerinden erişime açılmışsa, varsayılan olarak şifresiz 2375 veya TLS ile şifreli 2376): tcp.port == 2375 | | tcp.port == 2376
  - Bilinen genel konteyner imaj depolarına (Docker Hub, GCR, ECR, Quay.io vb.) yapılan trafik (genellikle HTTPS): tls.handshake.extensions_server_name contains "docker.io" | | tls.handshake.extensions_server_name contains "gcr.io" | | tls.handshake.extensions_server_name contains "quay.io" | | tls.handshake.extensions_server_name contains "amazonaws.com" (Sonuncusu ECR için daha geneldir ve daraltılması gerekebilir.)
  - Kubeshark [35] gibi araçlar kendi gelişmiş filtreleme mekanizmalarını sunar. Ancak, Kubeshark'ın kendi ön yüzüne (dashboard) erişim trafiği de bir gösterge olabilir (genellikle localhost:8899 gibi bir adreste çalışır, eğer Kubeshark bir pod içinde çalışıyorsa ve port yönlendirme yapılıyorsa bu port değişebilir): tcp.port == 8899 (veya Kubeshark tarafından kullanılan spesifik port).
  - Pod'lar arası iletişim için, eğer pod IP adresleri biliniyorsa: ip.addr == <pod_ip_1> && ip.addr == <pod_ip_2> (Bu, belirli iki pod arasındaki trafiği gösterir.) Kubernetes ortamında pod IP'leri dinamik olabileceğinden, bu

filtre genellikle canlı analizden ziyade belirli bir senaryoyu incelerken kullanışlıdır.

2.7. Geliştiricilere Özgü Port ve Protokol Kullanım Desenlerinin Tanımlanması

- **Açıklama:**
  - **Ne Olduğu:** Bu teknik, yazılımcıların geliştirme, test ve hata ayıklama süreçlerinde standart dışı (non-standard), uygulamaya özel veya geçici (ephemeral) ağ portlarını ve protokollerini kullanma eğilimlerini analiz etmeye odaklanır. Bu, geliştirme sunucuları için kullanılan ve genellikle iyi bilinen portlar (well-known ports) dışında kalan dinamik veya kayıtlı port aralıklarındaki [39] portları, mesaj kuyrukları (örneğin, RabbitMQ için AMQP, Kafka), çeşitli NoSQL veya SQL veritabanları (örneğin, MongoDB, Redis, PostgreSQL, MySQL), özel olarak geliştirilmiş API'ler veya daha az yaygın olan açık kaynak protokollerini içerebilir.
  - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, ağdaki tüm trafiği yakalayarak kullanılan TCP ve UDP port numaralarını [39] ve uygulama katmanı protokollerini [40] listeleme ve analiz etme yeteneğine sahiptir. Wireshark'ın "İstatistikler -> Protokol Hiyerarşisi" [33] özelliği, yakalanan trafikteki tüm protokollerin bir dökümünü ve bunların toplam trafikteki yüzdelerini gösterir; bu, ağda beklenmedik veya olağandışı protokollerin varlığını hızla belirlemeye yardımcı olabilir. Benzer şekilde, "İstatistikler -> Konuşmalar" özelliği, aktif olan tüm ağ iletişimlerini (endpoint çiftleri, portlar, protokoller ve transfer edilen veri miktarları) gösterir. Yazılımcılar, özellikle yerel geliştirme ortamlarında veya test sunucularında, standart olmayan (dinamik aralıktaki 49152-65535 veya kayıtlı aralıktaki 1024-49151 portlar [39]) portlarda çeşitli servisler çalıştırabilirler. Wireshark'ta, tcp.port == <port_num> veya udp.port == <port_num> gibi görüntüleme filtreleri [28], bu özel portlardaki trafiği izole etmek ve analiz etmek için kullanılır. Ayrıca, AMQP (RabbitMQ için [43]), MQTT [44], MongoDB gibi bazı geliştirici odaklı protokollere özgü Wireshark ayrıştırıcıları (dissectors) mevcuttur ve bu ayrıştırıcılar, ilgili protokollerin içindeki mesajları, komutları ve veri yapılarını daha anlamlı bir şekilde analiz etmeye olanak tanır.
  - **Neden Önemli Olduğu (2025 için):** Mikroservis mimarilerinin, olay güdümlü (event-driven) sistemlerin, çeşitli NoSQL ve NewSQL veritabanlarının ve gerçek zamanlı mesajlaşma platformlarının yazılım geliştirme pratiklerinde giderek daha fazla benimsenmesiyle, yazılımcılar artık çok daha fazla sayıda ve türde arka uç servisiyle (backend services) etkileşime girecektir. Bu servislerin her biri genellikle kendilerine özgü port numaraları ve bazen de özel uygulama katmanı protokolleri kullanır. Bu karmaşık ve çeşitli port/protokol kullanım desenlerinin tespiti ve analizi, ağ üzerindeki geliştirme aktivitelerinin doğasını,

kullanılan spesifik teknolojileri ve potansiyel olarak hangi projeler üzerinde çalışıldığını belirlemede 2025 yılında kilit bir rol oynayacaktır.

- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
  - Geliştirme, test veya prototipleme amacıyla kullanılan geçici veya özel olarak yapılandırılmış servislerin (örneğin, yerel bir makinede veya bir VM/konteyner içinde çalışan veritabanları, mesaj kuyrukları, özel API sunucuları) ağdaki varlığının ve kullanımının tespiti.
  - Ağ üzerinde hangi geliştirme teknolojilerinin, programlama dili ekosistemlerinin (örneğin, Node.js uygulamaları için tipik portlar, Python/Django geliştirme sunucusu portları) ve framework'lerinin aktif olarak kullanıldığına dair dolaylı çıkarımlar yapılması.
  - Standart dışı portlarda çalışan ve potansiyel olarak güncel olmayan, yamasız veya zayıf güvenlik yapılandırmalarına sahip olabilecek servislerin belirlenmesi, bu da bir güvenlik risk değerlendirmesi için önemli bir girdi sağlar.
  - Belirli bir projeye, teknoloji yığınına (technology stack) veya geliştirme metodolojisine aşina olan yazılımcıların veya ekiplerin ağ üzerindeki karakteristik aktivite profillerinin çıkarılması.
- **Kaynak/Referans:**
  - Wireshark Port Referansı ve Protokol Referansı: Wireshark Wiki.[39]
  - AMQP Trafik İncelemesi (RabbitMQ): RabbitMQ Belgeleri.[43]
  - MQTT Trafik Analizi: EMQX Blog.[44]
  - Genel Wireshark Filtreleri: Wireshark Wiki [42], LabEx.[28]
- **Derinlemesine Analiz ve Öngörüler:**
  - Modern yazılım geliştirmede "her iş için en uygun aracı seçme" (polyglot persistence, polyglot programming) felsefesinin yaygınlaşması, geliştiricilerin tek bir monolitik teknoloji yığınına bağlı kalmak yerine, projelerinin farklı bileşenleri veya gereksinimleri için birden fazla ve çeşitli veritabanı türü, mesajlaşma sistemi, programlama dili ve framework kullanmasına yol açmaktadır. Bu durum, bir geliştirme ortamında veya bir yazılımcının tipik bir iş gününde ağ üzerinde çok daha çeşitli port ve protokol desenlerinin gözlemlenmesi anlamına gelir. Örneğin, bir web uygulaması geliştiren bir yazılımcı aynı anda bir PostgreSQL veritabanına (port 5432), bir Redis önbelleğine (port 6379), bir RabbitMQ mesaj kuyruğuna (port 5672) ve kendi yerel geliştirme sunucusuna (örneğin, Node.js için port 3000 veya Python/Flask için port 5000) bağlanıyor olabilir. Bu çeşitlilik, tespit için daha fazla potansiyel sinyal sunarken, aynı zamanda "normal" ağ davranışının ne olduğunu tanımlamayı ve yanlış pozitifleri azaltmayı da karmaşıklaştırır.
  - Bir geliştiricinin, projesinin bir parçası olarak yerel makinesinde veya bir Docker konteyneri içinde geçici bir veritabanı (örneğin, test verileriyle dolu bir

PostgreSQL veritabanı port 5432'de) başlatması ve uygulamasıyla bu veritabanına bağlanması, ağ üzerinde bu porta yönelik kısa süreli ancak belirgin bir ağ aktivitesi yaratacaktır. Bu tür bir aktivite, özellikle geliştiricinin genel ağ profiliyle (örneğin, aynı zaman diliminde IDE trafiği, sürüm kontrol sistemi (Git) sunucularıyla iletişim, API test aracı kullanımı gibi diğer göstergelerle) birleştirildiğinde, bu port kullanımının büyük olasılıkla bir geliştirme faaliyeti olduğunu teyit edebilir. Tek başına bir porta yapılan bağlantı her zaman kesin bir kanıt olmasa da (örneğin, bir sistem yöneticisi de bir veritabanına bakım amacıyla bağlanabilir), diğer yazılımcı odaklı ağ göstergeleriyle birlikte değerlendirildiğinde anlamlı bir örüntü oluşturur.

- Geliştirme ortamlarında, özellikle işbirlikçi projelerde, geliştiricilerin birbirlerinin yerel olarak çalıştırdığı servisleri (örneğin, bir ekip üyesinin geliştirdiği bir API'yi test etmek) veya paylaşılan geliştirme kaynaklarını (örneğin, ortak bir test veritabanı) kolayca bulmasını ve bunlara erişmesini sağlamak için sıfır yapılandırmalı ağ (zero-configuration networking) protokollerinin (örneğin, mDNS/Bonjour, SSDP) veya diğer servis keşif mekanizmalarının kullanılması yaygınlaşabilir. Bu tür protokoller, ağ üzerinde belirli multicast adreslerine periyodik duyurular veya sorgular göndererek çalışır (örneğin, mDNS için 224.0.0.251 adresine UDP port 5353 üzerinden). Bu "konuşkan" protokollerin trafiği, özellikle geliştirme odaklı VLAN'larda veya alt ağlarda yoğun bir şekilde gözlemleniyorsa, aktif bir geliştirme ortamına ve işbirlikçi geliştirme pratiklerine işaret edebilir. Bu trafik, aynı zamanda ağda ek bir "gürültü" katmanı oluşturabilir, ancak belirli desenleri tanınabilirse tespit için kullanılabilir.
- **Wireshark Filtre Önerileri:**
  - Yaygın olarak kullanılan geliştirme odaklı veritabanı portları: tcp.port == 5432 | | udp.port == 5432 (PostgreSQL) || tcp.port == 27017 (MongoDB) || tcp.port == 3306 (MySQL/MariaDB) || tcp.port == 6379 (Redis) || tcp.port == 1433 (Microsoft SQL Server)
  - Mesaj kuyruğu sistemleri için bilinen portlar: tcp.port == 5672 | | tcp.port == 5671 (AMQP/RabbitMQ [43]) || tcp.port == 9092 (Apache Kafka) || tcp.port == 1883 | | tcp.port == 8883 (MQTT [44])
  - Wireshark'ın "İstatistikler -> Protokol Hiyerarşisi" [33] özelliği, ağdaki tüm protokollerin ve bunların göreceli kullanım oranlarının bir özetini sunar. Bu görünüm, beklenmedik, standart dışı veya az bilinen protokollerin (örneğin, özel bir RPC mekanizması veya geliştirme aşamasındaki bir protokol) varlığını hızla tespit etmeye yardımcı olabilir.
  - Belirli bir iç ağ segmentinde (örneğin, 192.168.1.0/24) bilinmeyen veya genellikle sunucu tarafında kullanılmayan yüksek numaralı portlara

(dinamik/özel portlar) yapılan TCP bağlantılarını bulmak için (bilinen yaygın portlar hariç tutularak): (ip.src_host matches "^192\\.168\\.1\\..*" | | ip.dst_host matches "^192\\.168\\.1\\..*") && (tcp.port > 1023 &&!(tcp.port == 443 | | tcp.port == 80 | | tcp.port == 22 | | tcp.port == 53 | | tcp.port == <diğer_bilinen_servis_portları>))
    - mDNS (Bonjour) trafiğini filtrelemek için: udp.port == 5353

2.8. Bulut Tabanlı IDE'ler ve Geliştirme Platformlarının Ağ İzlerinin Analizi

- **Açıklama:**
    - **Ne Olduğu:** Bu teknik, AWS Cloud9, GitHub Codespaces, Gitpod, Google Cloud Workstations, Microsoft Dev Box gibi bulut tabanlı Entegre Geliştirme Ortamlarının (IDE'ler) ve uzak geliştirme platformlarının kullanımından kaynaklanan ağ trafiği özelliklerini ve dijital izlerini analiz etmeye odaklanır. Analiz; kullanıcının bu platformlara ilk bağlantı kurma sürecini, platform arayüzüyle etkileşimini, kaynak kodu ve proje dosyalarının yerel makine ile bulut ortamı arasında senkronizasyonunu, bulut ortamındaki terminal etkileşimlerini (genellikle WebSockets üzerinden) ve platformun kendisinin çeşitli hizmetler (kimlik doğrulama, yetkilendirme, depolama, işlem kaynakları) için yaptığı arka plan ağ çağrılarını içerir.
    - **Nasıl Çalıştığı (Wireshark ile):** Bir kullanıcı bu tür bir bulut tabanlı geliştirme platformuna bağlandığında, genellikle standart bir web tarayıcısı üzerinden HTTPS ile güvenli hale getirilmiş bir web arayüzü kullanır veya bazen platforma özgü ince bir istemci (thin client) yazılımı aracılığıyla bağlantı kurar. Wireshark, bu bağlantıların hedef alan adlarını (örneğin, GitHub Codespaces için *.github.dev, *.online.visualstudio.com; Gitpod için *.gitpod.io; AWS Cloud9 için *.aws.amazon.com/cloud9 veya ilgili bölgesel uç noktalar), bu bağlantılar sırasında gerçekleşen TLS el sıkışma parametrelerini (JA3/JA4 parmak izleri dahil), terminal erişimi ve gerçek zamanlı işbirliği özellikleri için sıkça kullanılan WebSocket (WSS) iletişimlerini ve platformun çeşitli arka uç servisleriyle etkileşimde bulunmak için yaptığı RESTful API çağrılarını (genellikle JSON formatında veri taşıyan) yakalayabilir. Bu platformlar, genellikle iyi bilinen genel IP adresi aralıklarından veya büyük bulut sağlayıcılarının (AWS, Azure, GCP) ve İçerik Dağıtım Ağlarının (CDN'ler) altyapıları üzerinden hizmet verir.
    - **Neden Önemli Olduğu (2025 için):** Bulut tabanlı geliştirme ortamları, sundukları esneklik (herhangi bir cihazdan erişim), işbirliği kolaylığı (paylaşılan ortamlar, eş zamanlı düzenleme), güçlü ve ölçeklenebilir altyapıya anında erişim (önceden yapılandırılmış geliştirme yığınları, yüksek işlem gücü) gibi önemli avantajlar nedeniyle yazılım geliştiriciler arasında giderek daha popüler hale gelmektedir.[45] 2025 yılına gelindiğinde, bu platformların kullanımı, özellikle

büyük ve dağıtık ekipler için veya karmaşık projelere hızlı bir başlangıç yapmak isteyen geliştiriciler için standart bir uygulama haline gelebilir. Bu platformların ağ üzerindeki karakteristik imzalarını ve trafik desenlerini anlamak, yazılımcıların nerede, ne zaman ve nasıl çalıştığını tespit etmek, kurumsal veri güvenliğini sağlamak ve potansiyel politika ihlallerini belirlemek açısından giderek daha önemli olacaktır.

- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
  - Kurumsal ağlardan veya uzaktan çalışan kullanıcıların cihazlarından, onaylanmış veya onaylanmamış bulut tabanlı geliştirme platformlarına yapılan bağlantıların tespiti.
  - Bu platformlar üzerinden hassas kurumsal verilere (örneğin, özel kaynak kodu, API anahtarları, müşteri veritabanı şemaları) erişilip erişilmediğinin veya bu tür verilerin bu platformlara yetkisiz bir şekilde yüklenip yüklenmediğinin izlenmesi ve denetlenmesi.
  - Kurumsal güvenlik politikaları veya lisans anlaşmalarıyla çelişen, lisanssız veya onaylanmamış bulut geliştirme araçlarının veya hizmetlerinin kullanımının belirlenmesi.
  - Bu platformlarla ilişkili anormal veri transferlerinin (örneğin, beklenmedik zamanlarda veya beklenmedik hacimlerde büyük dosya yüklemeleri/indirmeleri, şüpheli dış IP adresleriyle sürekli veri alışverişi) tespiti, bu da potansiyel bir veri sızıntısı veya güvenlik ihlali göstergesi olabilir.
- **Kaynak/Referans:**
  - AWS Cloud9 ve GitHub Codespaces Karşılaştırması: PeerSpot [45] (Bu kaynak, platformların pazar payı, kullanım senaryoları ve kullanıcı geri bildirimleri hakkında bilgi sunarak popülerliklerini ve yaygınlıklarını desteklemektedir.)
  - Genel Ağ Analizi Prensipleri ve TCP/UDP Trafik İncelemesi: TryHackMe Wireshark Analiz Rehberi [46] (Bu kaynak, genel ağ trafiği analizi ve Wireshark kullanımı hakkında temel bilgiler sunar, bu da bulut platform trafiğini anlamak için bir temel oluşturabilir.)
- **Derinlemesine Analiz ve Öngörüler:**
  - Geliştirme ortamlarının "Hizmet Olarak Geliştirme Ortamı" (Development Environment as a Service - DEaaS) modeline doğru kayması, yazılımcıların herhangi bir coğrafi konumdan, herhangi bir cihazdan (hatta düşük güçlü tabletler veya Chromebook'lar gibi) tutarlı, güçlü ve önceden yapılandırılmış bir geliştirme ortamına anında erişebilmesini sağlamaktadır. Bu paradigma, geleneksel yerel makine tabanlı geliştirmeden önemli bir kopuşu temsil eder. Ağ trafiği analizi açısından bu durum, "yazılımcı trafiğinin" artık sadece belirli kurumsal ağlardan, bilinen ofis IP adreslerinden veya VPN bağlantı noktalarından kaynaklanmayabileceği anlamına gelir. Bunun yerine, yazılımcı

aktivitesi, bu küresel bulut geliştirme platformlarının iyi tanımlanmış (ancak potansiyel olarak dinamik) IP adreslerine ve alan adlarına yönelik şifreli HTTPS ve WebSocket (WSS) trafiği olarak kendini gösterecektir. Bu, tespit stratejilerinin coğrafi olarak dağıtık ve çeşitli kaynaklardan gelen trafiği analiz edebilecek şekilde uyarlanmasını gerektirir.

- Bir kuruluşun, veri egemenliği, güvenlik endişeleri veya maliyet yönetimi gibi nedenlerle belirli bulut tabanlı IDE'lerin veya geliştirme platformlarının kurumsal ağ içinde veya kurumsal verilerle kullanılmasını yasakladığını varsayalım. Ancak, bir geliştirici üretkenliğini artırmak, yeni teknolojileri denemek veya kişisel projeleri üzerinde çalışmak için bu platformlara kişisel hesabıyla ve potansiyel olarak kurumsal bir cihazdan erişmeye çalışabilir. Bu tür bir politika ihlali, Wireshark ile ağ trafiği izlenerek tespit edilebilir. Analiz, şirketin normalde izin vermediği veya engellediği belirli alan adlarına (örneğin, *.gitpod.io) veya bu platformlarla ilişkili bilinen IP adresi aralıklarına yapılan bağlantıları, TLS el sıkışmalarındaki SNI bilgilerini veya bu platformlara özgü JA4 parmak izlerini arayarak bu tür bir aktiviteyi ortaya çıkarabilir. Bu, sadece bir politika ihlalinin değil, aynı zamanda potansiyel bir "gölge BT" (shadow IT) durumunun ve olası veri sızıntısı risklerinin de bir göstergesi olabilir.
- Bulut tabanlı IDE'lerin ve geliştirme platformlarının artan kullanımı, ağ güvenliği mimarilerinde Sıfır Güven (Zero Trust) prensiplerinin [47] benimsenmesinin ve uygulanmasının önemini daha da artırmaktadır. Geleneksel çevre tabanlı güvenlik modelleri (yani, "içerisi güvenli, dışarısı güvensiz" varsayımı), geliştirme ortamı ve potansiyel olarak hassas kaynak kodu gibi varlıklar kurumsal ağın fiziksel sınırlarının dışına taştığında yetersiz kalır. Sıfır Güven yaklaşımı, ağın içinden veya dışından geldiğine bakılmaksızın hiçbir kullanıcıya, cihaza veya uygulamaya varsayılan olarak güvenmez ve her erişim isteğini ve veri akışını sürekli olarak doğrular, yetkilendirir ve izler. Bu bağlamda, kurumsal ağlardan bulut tabanlı geliştirme platformlarına yapılan bağlantıların Wireshark veya benzeri araçlarla izlenmesi ve analiz edilmesi, Sıfır Güven stratejisinin kritik bir bileşeni olarak, kimin hangi bulut kaynaklarına eriştiğini, bu erişimin meşru ve politikalarla uyumlu olup olmadığını ve herhangi bir anormal davranış sergileyip sergilemediğini doğrulamak için hayati öneme sahiptir.
- **Wireshark Filtre Önerileri:**
  - GitHub Codespaces trafiği için (alan adları değişebilir, güncel tutulmalıdır): tls.handshake.extensions_server_name contains "github.dev" || tls.handshake.extensions_server_name contains "codespaces.github.com" || tls.handshake.extensions_server_name contains "online.visualstudio.com"
  - AWS Cloud9 trafiği için (AWS'nin geniş IP aralıkları ve çeşitli servis uç

noktaları nedeniyle daha karmaşık olabilir; SNI daha güvenilirdir): tls.handshake.extensions_server_name contains "cloud9.aws.amazon.com" | | tls.handshake.extensions_server_name ends with ".c9users.io" (Cloud9 ortamlarının kullandığı tipik bir alan adı son eki) (Spesifik AWS IP aralıkları için: ip.addr == <bilinen_Cloud9_IP_aralığı_1> | | ip.addr == <bilinen_Cloud9_IP_aralığı_2>) (Bu IP aralıkları AWS tarafından yayınlanan güncel listelerden alınmalıdır ve dinamik olabilir.)
    - Gitpod trafiği için: tls.handshake.extensions_server_name contains "gitpod.io"
    - Bu platformlar tarafından terminal, düzenleyici senkronizasyonu ve işbirliği için sıkça kullanılan genel WebSocket (WSS) trafiğini yakalamak (daha sonra hedef IP/alan adlarına veya diğer göstergelere göre daraltılmalıdır): websocket | | tls.handshake.extensions_alpn_str == "http/1.1" (WebSocket genellikle HTTP/1.1 üzerinden yükseltilir) && tcp.port == 443
    - Bu platformlara yapılan bağlantılarda belirli JA4 parmak izlerini aramak (eğer bu platformlara özgü parmak izleri biliniyorsa veya öğrenilebiliyorsa).

2.9. Ağ Üzerinden Çalışan Hata Ayıklama (Debugging) Protokollerinin İzlenmesi

- **Açıklama:**
    - **Ne Olduğu:** Bu teknik, yazılımcıların uygulamalardaki hataları bulmak, analiz etmek ve düzeltmek (hata ayıklama) için kullandığı ve ağ üzerinden iletişim kurabilen çeşitli hata ayıklama (debugging) protokollerinin trafiğini tespit etmeye ve analiz etmeye odaklanır. Örnek protokoller arasında PHP uygulamaları için Xdebug (özellikle DBGp protokolü), Java uygulamaları için Java Debug Wire Protocol (JDWP), Node.js uygulamaları veya Chromium tabanlı tarayıcılar için Chrome DevTools Protocol ve.NET uygulamaları için Visual Studio Debugger Protocol (MSVSMON) gibi protokoller bulunmaktadır.
    - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, bu özel hata ayıklama protokollerinin kullandığı bilinen varsayılan TCP veya UDP portlarını dinleyerek bu protokollerle ilgili ağ bağlantılarını ve veri paketlerini yakalayabilir. Örneğin, Xdebug genellikle TCP port 9000 veya 9003'ü kullanır [48]; JDWP için standart bir port olmamakla birlikte, geliştiriciler tarafından genellikle 8000, 8787 gibi portlar veya dinamik olarak atanan portlar kullanılır [50]; Chrome DevTools Protocol ise genellikle TCP port 9222 veya 9229 üzerinden WebSocket bağlantıları kullanır.[51] Bu protokollerin çoğu, ya düz metin (plain text) tabanlı komutlar ve yanıtlar ya da basit, iyi tanımlanmış ikili (binary) formatlarda veri paketleri gönderir. Örneğin, JDWP bağlantısı bir JDWP-Handshake dizesiyle başlar.[48] Chrome DevTools Protocol, WebSocket üzerinden JSON formatında mesajlar kullanır.[51] Wireshark, bu el sıkışmalarını, komut dizilerini ve veri

yapılarını (eğer protokol için yerleşik bir ayrıştırıcısı varsa veya özel bir Lua tabanlı ayrıştırıcı yazılırsa) paket ayrıntıları bölmesinde gösterebilir ve filtrelenebilir hale getirebilir.
    - **Neden Önemli Olduğu (2025 için):** Dağıtık sistemlerin, mikroservis mimarilerinin, konteynerize edilmiş uygulamaların ve bulut tabanlı platformların yazılım geliştirme süreçlerinde giderek daha fazla kullanılmasıyla birlikte, uzaktan hata ayıklama (remote debugging) senaryoları daha yaygın ve gerekli hale gelmektedir. 2025 yılında, geliştiricilerin kendi yerel makinelerinden farklı ortamlarda (örneğin, bir test sunucusunda, bir Docker konteynerinde, bir Kubernetes pod'unda veya bir bulut sanal makinesinde) çalışan uygulamalarda hata ayıklaması standart bir pratik olacaktır. Bu nedenle, bu ağ tabanlı hata ayıklama protokollerinin ağdaki varlığı ve kullanımı, aktif bir yazılımcı faaliyeti veya devam eden bir geliştirme/sorun giderme süreci için önemli bir gösterge olacaktır. Ayrıca, bu hata ayıklama portlarının ve protokollerinin yanlışlıkla veya dikkatsizce ağa açık bırakılması, yetkisiz erişim ve kod çalıştırma gibi ciddi güvenlik riskleri de oluşturabilir [50], bu da tespitlerini daha da önemli kılar.
- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
    - Ağ üzerinde aktif olarak devam eden hata ayıklama oturumlarının ve dolayısıyla bu oturumları başlatan veya katılan yazılımcıların varlığının tespiti.
    - Yanlışlıkla veya güvensiz bir şekilde internete veya yetkisiz iç ağ segmentlerine açık bırakılmış hata ayıklama portlarının (örneğin, bir üretim sunucusunda açık kalmış bir JDWP portu) proaktif olarak belirlenmesi; bu durum, istismar edilmeden önce kapatılması gereken kritik bir güvenlik açığıdır.
    - Uzaktan hata ayıklama protokolleri üzerinden, yetkisiz bir kişi tarafından hedef sistemde rastgele kod çalıştırma, bellek içeriğini okuma/değiştirme veya hassas bilgilere (örneğin, veritabanı şifreleri, özel anahtarlar) erişme girişimlerinin potansiyel tespiti.
    - Bir kuruluş içindeki hangi uygulamaların, servislerin veya sistem bileşenlerinin aktif olarak geliştirilmekte, test edilmekte veya sorun giderilmekte olduğunun, bu sistemlere yönelik hata ayıklama trafiği aracılığıyla anlaşılması.
- **Kaynak/Referans:**
    - Xdebug DBGp Protokolü: Xdebug Belgeleri [48], JetBrains YouTrack Tartışması.[49]
    - Java Debug Wire Protocol (JDWP): HackTricks (Pentesting JDWP) [50], GraalVM Belgeleri (JDWP Desteği).[53]
    - Chrome DevTools Protocol: Resmi Protokol Belgeleri [52], Jorian Woltjer'in Kitabından Bölüm [51], Chrome Geliştirici Belgeleri [54], Meraki Dokümantasyonu.[55]
- **Derinlemesine Analiz ve Öngörüler:**
    - Yazılım geliştirme yaşam döngüsünde "güvenliği sola kaydırma" (Shift Left Security) ve "testi sola kaydırma" (Shift Left Testing) gibi modern pratikler,

güvenlik kontrollerinin ve test süreçlerinin geliştirme döngüsünün mümkün olduğunca erken aşamalarına entegre edilmesini teşvik etmektedir. Bu durum, hata ayıklama araçlarının ve dolayısıyla bu araçların kullandığı ağ tabanlı hata ayıklama protokollerinin daha sık, daha çeşitli ortamlarda (sadece yerel makineler değil, aynı zamanda CI/CD ortamları, paylaşılan test sunucuları vb.) ve geliştirme sürecinin daha erken evrelerinde kullanılmasına yol açabilir. Bu artan kullanım, bu protokollerin ağdaki görünürlüğünü ve tespit edilme olasılığını da artıracaktır.

- Bir geliştirme ortamında, özellikle aceleyle veya yetersiz bilgiyle yapılandırılmış güvenlik duvarı kurallarının veya ağ segmentasyon politikalarının bulunması, normalde sadece geliştiricinin yerel makinesinden (localhost) veya güvenli bir geliştirme ağı içinden erişilmesi gereken hata ayıklama portlarının (örneğin, JDWP veya Xdebug portları) yanlışlıkla tüm iç ağa veya daha da kötüsü internete açık kalmasına neden olabilir. Bu durum, hem ağda bir yazılımcı aktivitesinin (meşru bir hata ayıklama oturumu ise) bir göstergesi olabilir hem de eğer bu portlar kimlik doğrulama gerektirmiyorsa veya zayıf kimlik bilgilerine sahipse, saldırganlar için sisteme sızmak, bilgi toplamak veya rastgele kod çalıştırmak üzere istismar edilebilecek kritik bir güvenlik açığı anlamına gelir.[50] Wireshark ile bu portlara yapılan beklenmedik veya yetkisiz bağlantı denemeleri veya bu portlardan kaynaklanan anormal trafik, bu tür riskli durumları ortaya çıkarabilir.
- Hata ayıklama protokollerinin kendileri de, herhangi bir yazılım veya ağ protokolü gibi, tasarım veya implementasyonlarında güvenlik açıklarına sahip olabilir. Bu protokollerin ağdaki varlığının tespiti, bu protokolleri kullanan sistemlerin, ilgili protokollerin bilinen zafiyetlerine (örneğin, CVE veri tabanlarında listelenmiş olanlar) karşı taranması ve korunması için önemli bir başlangıç noktası olabilir. Örneğin, belirli bir JDWP sürümünün uzaktan kod çalıştırmaya (RCE) izin veren bilinen bir zafiyeti varsa ve ağda bu JDWP sürümünün kullanıldığı tespit edilirse, bu durum acil bir yama veya azaltma önlemi gerektirebilir. Bu nedenle, hata ayıklama trafiğinin tespiti sadece yazılımcı aktivitesini değil, aynı zamanda potansiyel saldırı yüzeylerini de ortaya çıkarır.
- **Wireshark Filtre Önerileri:**
  - Xdebug için (varsayılan DBGp portları): tcp.port == 9000 | | tcp.port == 9003 [48]
  - JDWP Handshake için (port numarası değişebilir, ancak handshake dizesi sabittir): tcp contains "JDWP-Handshake".[48] Eğer JDWP'nin kullandığı spesifik bir port biliniyorsa (örneğin 8787): tcp.port == 8787 && tcp contains "JDWP-Handshake". Wireshark'ın bazı sürümleri jdwp protokol

filtresini destekleyebilir, bu durumda doğrudan jdwp kullanılabilir.
    - Chrome DevTools Protocol için (genellikle WebSocket üzerinden JSON mesajları, varsayılan portlar 9222 veya 9229): tcp.port == 9222 | | tcp.port == 9229. Bu filtreleme yapıldıktan sonra, ilgili TCP akışları "Follow TCP Stream" ile incelenerek WebSocket el sıkışması ve ardından gelen JSON formatındaki protokol mesajları görülebilir. Alternatif olarak, eğer trafik WSS (WebSocket Secure) ise tls ve ardından websocket filtreleri birleştirilebilir.
    - Node.js hata ayıklama portu için (eski sürümlerde 5858, modern sürümlerde genellikle Chrome DevTools Protocol ile aynı portlar kullanılır veya --inspect ile belirtilen port): tcp.port == 5858 (eski Node.js için) veya yukarıdaki Chrome DevTools portları.

2.10. Wireshark için Lua ile Özelleştirilmiş Ayrıştırıcılar (Dissectors) ve Gelişmiş Otomasyon Betikleri Geliştirme

- **Açıklama:**
    - **Ne Olduğu:** Bu ileri düzey teknik, Wireshark'ın standart protokol analiz yeteneklerinin ötesine geçerek, yazılımcıların kullandığı özel (proprietary), standart dışı (non-standard) veya daha az bilinen, niş ağ protokollerini analiz etmek amacıyla Lua programlama dilini kullanarak özel protokol ayrıştırıcıları (custom dissectors) geliştirmeyi içerir. Buna ek olarak, tekrarlayan veya karmaşık analiz görevlerini otomatikleştirmek, belirli geliştirici davranış kalıplarını veya anormal aktiviteleri tespit etmek için özel tespit mantıkları uygulamak ve Wireshark'ın çıktılarını zenginleştirmek amacıyla Lua tabanlı otomasyon betikleri (automation scripts), "tap"ler ve "post-dissector"lar oluşturmayı kapsar.
    - **Nasıl Çalıştığı (Wireshark ile):** Wireshark, Lua programlama dili [56] aracılığıyla son derece esnek ve güçlü bir genişletilebilirlik mimarisine sahiptir. Geliştiriciler ve analistler, belirli bir ağ protokolünün yapısını (örneğin, başlık alanları, veri türleri, mesaj sıralamaları, alanların anlamları) tanımlayan Lua betikleri yazarak Wireshark'ın bu özel protokolü "anlamasını" ve paket ayrıntıları bölmesinde (packet detail pane) insan tarafından okunabilir ve filtrelenebilir bir şekilde düzgün bir şekilde göstermesini sağlayabilirler.[57] Lua ayrıca, yakalanan paketler üzerinde daha karmaşık analizler yapmak, özel istatistikler çıkarmak, belirli olayları veya kalıpları aramak, hatta harici sistemlerle entegrasyon sağlamak için "tap"ler (belirli noktalarda veri toplamak için) ve "post-dissector"lar (diğer tüm ayrıştırıcılar çalıştıktan sonra ek analizler yapmak için) oluşturmak amacıyla da kullanılabilir.[56]
    - **Neden Önemli Olduğu (2025 için):** Yazılım geliştirme ekosistemi sürekli

olarak evrilmekte, yeni araçlar, framework'ler, iletişim kütüphaneleri ve hatta tamamen yeni ağ protokolleri ortaya çıkmaktadır. Bu yeni teknolojilerin hepsi hemen standartlaşmış veya Wireshark gibi genel amaçlı analiz araçları tarafından varsayılan olarak destekleniyor olmayabilir. Özellikle kurum içi geliştirilmiş özel araçlar, niş endüstriyel protokoller veya hızla gelişen açık kaynak projeleri için, bu özel protokollerin ağ trafiğini etkin bir şekilde analiz edebilmek amacıyla özel ayrıştırıcılara ve otomasyon betiklerine olan ihtiyaç 2025 yılında daha da artacaktır. Lua, bu esnekliği, özelleştirmeyi ve otomasyonu sağlayarak "Developer Hunter" gibi projelerin tespit doğruluğunu, kapsamını ve genel etkinliğini önemli ölçüde artırma potansiyeline sahiptir.

- **2025'teki Potansiyel Etkileri ve Uygulama Alanları:**
  - Kurum içi geliştirilmiş veya ticari olmayan, kapalı kaynaklı geliştirici araçlarının, test framework'lerinin veya özel iletişim kütüphanelerinin kullandığı özel (proprietary) ağ protokollerinin ayrıntılı analizi ve bu araçların kullanımının tespiti.
  - Oyun geliştirme (özel oyun motoru ağ protokolleri), gömülü sistemler (cihazlar arası özel iletişim), Endüstriyel Kontrol Sistemleri (ICS) veya IoT (Nesnelerin İnterneti) gibi özel alanlarda kullanılan ve standart Wireshark ayrıştırıcıları bulunmayan iletişim protokollerindeki geliştirici aktivitelerinin (örneğin, firmware yükleme, uzaktan komut gönderme, telemetri toplama) tespiti.
  - Karmaşık, çok adımlı ve bağlama duyarlı geliştirici iş akışlarını (örneğin, belirli bir API çağrı dizisi, bir derleme sunucusuna bağlanıp ardından test sonuçlarını belirli bir formatta raporlama gibi) tespit etmek için özel mantık içeren otomatikleştirilmiş analiz betikleri oluşturma.
  - Wireshark'ın tespit yeteneklerini, projenin veya organizasyonun özel ihtiyaçlarına, değişen tehdit ortamına veya yeni ortaya çıkan geliştirici araçlarına göre sürekli olarak uyarlama, iyileştirme ve genişletme.
  - Yanlış pozitif alarmları azaltmak ve tespit doğruluğunu artırmak için, birden fazla protokol katmanından veya farklı trafik akışlarından gelen bilgileri birleştirerek daha zengin ve bağlamsal analizler yapabilen Lua tabanlı post-dissector'lar geliştirme.
- **Kaynak/Referans:**
  - Wireshark Lua Scripting: Wireshark Wiki [56] (Lua'nın Wireshark'ta nasıl kullanıldığı, dissector, post-dissector ve tap yazımı hakkında temel bilgiler içerir), Wireshark Geliştirici Rehberi (WSDG) Lua Desteği Bölümü [56]'de referans verilmiştir.
  - Özel Wireshark Dissector Geliştirme: Sewio RTLS Belgeleri (bir dissector yazma süreci hakkında genel bir örnek sunar).[57]
- **Derinlemesine Analiz ve Öngörüler:**

- Ağ analizi araçları ve genel olarak siber güvenlik alanında, "tek bir çözüm herkese uymaz" (one-size-fits-all) yaklaşımının sınırları giderek daha fazla kabul görmektedir. Wireshark gibi standart ve güçlü araçlar mükemmel bir temel sunarken, her kuruluşun veya her projenin karşılaştığı özel zorluklar, hızla değişen teknolojik ortamlar ve benzersiz tehdit vektörleri, bu araçların kullanıcı tarafından programlanabilir, genişletilebilir ve özelleştirilebilir olmasını giderek daha fazla gerektirmektedir. Lua, Wireshark'a bu esnekliği kazandıran anahtar teknolojilerden biridir. Bu, analistlerin ve geliştiricilerin, genel amaçlı bir aracı kendi özel ihtiyaçlarına göre şekillendirmelerine olanak tanır.
- Bir şirketin, geliştiricilerinin kullandığı ve standart olmayan, kurum içi geliştirilmiş bir yapılandırma yönetim protokolü veya özel bir RPC (Remote Procedure Call) mekanizması olduğunu varsayalım. Bu protokol için özel bir Lua ayrıştırıcısı yazmak, bu protokol üzerinden yapılan yetkisiz değişiklikleri, hatalı yapılandırmaları, performans sorunlarını veya güvenlik açıklarını tespit etmeyi ve analiz etmeyi büyük ölçüde kolaylaştırabilir. Özel bir ayrıştırıcı olmadan, bu tür bir trafiği analiz etmek, genellikle karmaşık frame contains "belirli_bir_bayt_dizisi" gibi filtrelere, paketlerin ham bayt içeriğinin manuel olarak incelenmesine veya harici betiklerle pcap dosyalarının işlenmesine dayanır ki bu yöntemler hem verimsiz hem de hataya açıktır. Özel bir Lua ayrıştırıcısı ile, bu protokolün alanları Wireshark arayüzünde anlamlı isimlerle görünür hale gelir (örneğin, my_custom_protocol.command_type == "SET_CONFIG") ve bu da hem interaktif analizi hem de otomatik filtrelemeyi ve uyarı oluşturmayı basitleştirir.
- Lua betiklerinin Wireshark içinde kullanılması, sadece bilinmeyen protokollerin ayrıştırılmasının ötesine geçer. Örneğin, bir "Developer Hunter" projesi, belirli bir geliştirici davranışını (örneğin, bir Git push işleminin ardından hemen bir CI/CD sunucusuna bildirim gönderilmesi ve ardından bir test ortamına dağıtım yapılması gibi sıralı olaylar) tespit etmek isteyebilir. Bu tür karmaşık, durum bilgisi gerektiren (stateful) ve birden fazla trafik akışını ilişkilendiren bir mantığı standart Wireshark görüntüleme filtreleriyle uygulamak çok zor veya imkansız olabilir. Ancak, bir Lua "tap" betiği, ilgili protokollerden (SSH, HTTP, özel CI/CD protokolü) veri toplayabilir, bu verileri bir oturum veya kullanıcı bağlamında ilişkilendirebilir ve tanımlanmış bir davranış kalıbı tetiklendiğinde bir uyarı üretebilir veya log kaydı oluşturabilir. Bu, tespit yeteneklerini basit imza tabanlı yaklaşımlardan daha sofistike, davranışsal ve bağlamsal analizlere doğru taşır. 2025'e doğru, bu tür özelleştirilmiş otomasyon ve zeka katmanı, ağ trafiği analizinin etkinliğini artırmada giderek daha önemli hale gelecektir.
- **Wireshark Filtre Önerileri:**
  - Lua ile yazılmış özel bir ayrıştırıcı (örneğin, myproto adında) yüklendikten

sonra, bu protokolün paketlerini filtrelemek için: myproto
- Özel ayrıştırıcınızın tanımladığı belirli alanlara göre filtreleme (örneğin, myproto içinde command adında bir alan varsa ve bu alanın değeri login ise): myproto.command == "login"
- Bir Lua "tap" betiği, analiz sonuçlarını Wireshark'a yeni, özel alanlar (pseudo-fields) olarak ekleyebilir. Örneğin, bir tap betiği bir HTTP akışının "geliştiriciye ait olma olasılığını" hesaplayıp bunu devhunter.likelihood adında bir alan olarak ekleyebilir. Bu durumda filtre şöyle olabilir: devhunter.likelihood > 0.8
- Lua betikleri genellikle doğrudan filtre olarak çağrılmaz; bunun yerine, ya protokolleri ayrıştırarak filtrelenebilir alanlar sunarlar ya da analiz sonuçlarını (örneğin, "Uzman Bilgileri" bölümüne notlar ekleyerek veya özel sütunlar oluşturarak) Wireshark arayüzüne yansıtırlar. Bu nedenle, Lua kullanımı genellikle doğrudan "filtre" yazmaktan ziyade, Wireshark'ın analiz ve görüntüleme yeteneklerini genişletmekle ilgilidir.

3. Sonuç ve Değerlendirme

Bu rapor, 2025 yılı ve sonrası için bir bilgisayar ağındaki yazılımcıları Wireshark kullanarak tespit etmeye yönelik en son ve en etkili olduğu öngörülen 10 tekniği ve trendi ayrıntılı bir şekilde incelemiştir. Analiz edilen her bir yöntem, yazılımcıların karakteristik ağ aktivitelerini (Git/SSH kullanımı, IDE trafiği, API test araçları, sanal makine/konteyner etkileşimleri, özel port/protokol kullanımları vb.) farklı açılardan ele alarak, "Developer Hunter" projesinin hedeflerine ulaşmasına katkıda bulunabilecek potansiyel yaklaşımlar sunmaktadır.

Öne çıkan temel bulgular şunlardır:

1. **Şifreli Trafik Analizinin Kaçınılmazlığı:** Ağ trafiğinin büyük çoğunluğunun şifreli olması [2], TLS parmak izi (özellikle JA4/JA4+ gibi gelişmiş yöntemler [10]) ve SSH trafiğinin (mümkünse SSLKEYLOGFILE ile [11]) analizini zorunlu kılmaktadır. Bu teknikler, şifrelemeye rağmen istemci uygulamalarını ve potansiyel olarak geliştirici araçlarını tanımlamak için kritik öneme sahiptir.
2. **Makine Öğreniminin Yükselişi:** Yazılımcı davranışlarının karmaşıklığı ve çeşitliliği, statik kuralların ötesine geçen, öğrenen ve adapte olan sistemlere olan ihtiyacı artırmaktadır. Makine öğrenimi destekli davranışsal analiz [14], normalden sapan yazılımcı aktivitelerini tespit etmede ve yeni tehditlere karşı proaktif bir duruş sergilemede önemli bir potansiyel sunmaktadır.
3. **Araç Zinciri İmzalarının Önemi:** IDE'ler [21], API test araçları [24] ve sürüm kontrol sistemleri [11] gibi yazılımcıların temel araçları, kendilerine özgü ağ imzaları

bırakmaktadır. Bu imzaların (User-Agent'lar, hedef sunucular, protokol davranışları) detaylı analizi, yazılımcı varlığını ve faaliyetlerini belirlemede değerli ipuçları sunar.

4. **Sanallaştırma ve Orkestrasyon Katmanlarının İzlenmesi:** Konteynerler (Docker) ve orkestrasyon platformları (Kubernetes) [35], modern geliştirme ortamlarının merkezindedir. Bu katmanlardaki ağ etkileşimlerinin (API sunucusu çağrıları, pod'lar arası iletişim) Kubeshark gibi özel araçlarla veya Wireshark ile izlenmesi, bu ortamlardaki geliştirici aktivitelerini anlamak için gereklidir.
5. **Özelleştirme ve Otomasyon İhtiyacı:** Standart dışı protokollerin veya karmaşık davranış kalıplarının tespiti, Wireshark'ın Lua [56] gibi betikleme dilleriyle genişletilmesini ve özel ayrıştırıcılar veya otomasyon betikleri geliştirilmesini gerektirebilir. Bu, projenin özel ihtiyaçlara göre uyarlanabilirliğini ve etkinliğini artıracaktır.
6. **Port ve Protokol Çeşitliliği:** Yazılımcıların kullandığı çok çeşitli veritabanları, mesaj kuyrukları ve özel servisler [39], ağ üzerinde zengin bir port ve protokol çeşitliliği yaratır. Bu desenlerin tespiti, kullanılan teknolojiler ve dolayısıyla geliştirici profilleri hakkında çıkarımlar yapılmasına olanak tanır.
7. **Bulut Tabanlı Geliştirme Ortamlarının Etkisi:** AWS Cloud9, GitHub Codespaces gibi platformların [45] artan kullanımı, yazılımcı trafiğinin kaynaklarını ve hedeflerini değiştirmekte, bu da tespit stratejilerinin bu yeni paradigmaya uyum sağlamasını gerektirmektedir.
8. **Ağ Üzerinden Hata Ayıklama Protokolleri:** Xdebug, JDWP gibi protokollerin [48] ağdaki varlığı, aktif bir geliştirme veya sorun giderme sürecine işaret eder ve aynı zamanda potansiyel güvenlik risklerini de beraberinde getirebilir.

Bu tekniklerin ve trendlerin her biri, "Developer Hunter" projesi kapsamında özel Wireshark filtreleri (hem yakalama hem de görüntüleme için) oluşturulurken dikkate alınmalıdır. Ancak, hiçbir tekniğin tek başına yeterli olmayacağı unutulmamalıdır. En etkili yaklaşım, bu tekniklerin bir kombinasyonunu kullanarak, birden fazla göstergeyi ilişkilendiren ve bağlamsal analiz yapabilen katmanlı bir tespit stratejisi geliştirmek olacaktır.

2025 ve sonrası için öngörülen bu gelişmeler, ağ güvenliği ve yazılımcı aktivitesi izleme alanında sürekli bir adaptasyon ve öğrenme gerektirecektir. Geliştirici araçları, protokoller ve çalışma yöntemleri değiştikçe, tespit mekanizmalarının da bu değişimlere ayak uydurması, projenin uzun vadeli başarısı için hayati önem taşımaktadır. Yanlış pozitiflerin en aza indirilmesi ve tespit doğruluğunun en üst düzeye çıkarılması, bu tekniklerin pratik uygulamasında temel zorluklar olmaya devam edecektir. Bu nedenle, dinamik temel çizgiler oluşturma, anomali tespit eşiklerini dikkatlice ayarlama ve mümkün olduğunca çok bağlamsal bilgiyi (örneğin, zaman,

kullanıcı rolü, proje bilgisi) analiz sürecine dahil etme çabaları kritik olacaktır.

Alıntılanan çalışmalar

1. What is Network Traffic Analysis? | VMware, erişim tarihi Mayıs 29, 2025, https://www.vmware.com/topics/network-traffic-analysis
2. Wireshark Essentials: Mastering Network Traffic Analysis | Lenovo US, erişim tarihi Mayıs 29, 2025, https://www.lenovo.com/us/en/glossary/wireshark/
3. How to Use Wireshark: Comprehensive Tutorial + Tips, erişim tarihi Mayıs 29, 2025, https://www.varonis.com/blog/how-to-use-wireshark
4. What Is Wireshark and How to Use It | Cybersecurity - CompTIA, erişim tarihi Mayıs 29, 2025, https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it
5. Beginner's Guide to Wireshark - zenarmor.com, erişim tarihi Mayıs 29, 2025, https://www.zenarmor.com/docs/network-basics/what-is-wireshark
6. What is Network Traffic Analysis (NTA): Importance and Methods - Group-IB, erişim tarihi Mayıs 29, 2025, https://www.group-ib.com/resources/knowledge-hub/network-traffic-analysis/
7. Encrypted Traffic Analysis - Progress Flowmon, erişim tarihi Mayıs 29, 2025, https://www.progress.com/flowmon/solutions/security-operations/encrypted-traffic-analysis
8. TLS Fingerprinting using JA3 for Android Application - Theseus, erişim tarihi Mayıs 29, 2025, https://www.theseus.fi/bitstream/10024/866474/2/Agarwal_Ashrika.pdf
9. Adapting to Change: JA3 Fingerprints Fade as Browsers Embrace TLS Extension Randomization - Stamus Networks, erişim tarihi Mayıs 29, 2025, https://www.stamus-networks.com/blog/ja3-fingerprints-fade-browsers-embrace-tls-extension-randomization
10. Advancing Threat Intelligence: JA4 fingerprints and inter-request ..., erişim tarihi Mayıs 29, 2025, https://blog.cloudflare.com/ja4-signals/
11. SSH - Wireshark Wiki, erişim tarihi Mayıs 29, 2025, https://wiki.wireshark.org/SSH
12. How to capture HTTP traffic using Wireshark, Fiddler, or tcpdump - Atlassian Support, erişim tarihi Mayıs 29, 2025, https://support.atlassian.com/atlassian-knowledge-base/kb/how-to-capture-http-traffic-using-wireshark-fiddler-or-tcpdump/
13. www.usenix.org, erişim tarihi Mayıs 29, 2025, https://www.usenix.org/system/files/nsdi20-paper-yaseen.pdf
14. journalspub.com, erişim tarihi Mayıs 29, 2025, https://journalspub.com/wp-content/uploads/2024/09/23-31-Network-Intrusion-Detection-Using-Wireshark-And-Machine-Learning.pdf
15. The power of AI and ML in network traffic analysis: next generation NDR solutions, erişim tarihi Mayıs 29, 2025, https://socwise.eu/the-power-of-ai-and-ml-in-network-traffic-analysis-next-generation-ndr-solutions/
16. Flow Feature-Based Network Traffic Classification Using Machine Learning - ResearchGate, erişim tarihi Mayıs 29, 2025,

https://www.researchgate.net/publication/358641088_Flow_Feature-Based_Network_Traffic_Classification_Using_Machine_Learning
17. Visual Studio Code Update IP's - Stack Overflow, erişim tarihi Mayıs 29, 2025, https://stackoverflow.com/questions/35900758/visual-studio-code-update-ip-s
18. Extension Marketplace - Visual Studio Code, erişim tarihi Mayıs 29, 2025, https://code.visualstudio.com/docs/configure/extensions/extension-marketplace
19. What's the IP whitelist of IntelliJ IDE in case of firewall policy or restricted network, erişim tarihi Mayıs 29, 2025, https://youtrack.jetbrains.com/articles/SUPPORT-A-288/Whats-the-IP-whitelist-of-IntelliJ-IDE-in-case-of-firewall-policy-or-restricted-network
20. Intellij Updates/Marketplace Port - IDEs Support (IntelliJ Platform) | JetBrains, erişim tarihi Mayıs 29, 2025, https://intellij-support.jetbrains.com/hc/en-us/community/posts/12122215925906-Intellij-Updates-Marketplace-Port
21. Telemetry - Visual Studio Code, erişim tarihi Mayıs 29, 2025, https://code.visualstudio.com/docs/configure/telemetry
22. OpenTelemetry tracing and metrics | IntelliJ IDEA Documentation - JetBrains, erişim tarihi Mayıs 29, 2025, https://www.jetbrains.com/help/idea/open-telemetry-tracing-and-metrics.html
23. How to analyze HTTP traffic in Wireshark for Cybersecurity purposes - LabEx, erişim tarihi Mayıs 29, 2025, https://labex.io/tutorials/wireshark-how-to-analyze-http-traffic-in-wireshark-for-cybersecurity-purposes-415495
24. Postman User Agent Guide: Setting and Changing - Bright Data, erişim tarihi Mayıs 29, 2025, https://brightdata.com/blog/web-data/postman-user-agent
25. Postman - Override User-Agent for each request - Stack Overflow, erişim tarihi Mayıs 29, 2025, https://stackoverflow.com/questions/71938214/postman-override-user-agent-for-each-request
26. How to Set Postman User Agent - ZenRows, erişim tarihi Mayıs 29, 2025, https://www.zenrows.com/blog/postman-user-agent
27. insomnia-plugin-user-agents, erişim tarihi Mayıs 29, 2025, https://insomnia.rest/plugins/insomnia-plugin-user-agents
28. How to filter network traffic based on protocol, port, and HTTP method in Wireshark for Cybersecurity | LabEx, erişim tarihi Mayıs 29, 2025, https://labex.io/tutorials/wireshark-how-to-filter-network-traffic-based-on-protocol-port-and-http-method-in-wireshark-for-cybersecurity-415199
29. Insomnia vs Postman vs SoapUI: Which API Tool is Best? - Abstracta US, erişim tarihi Mayıs 29, 2025, https://abstracta.us/blog/testing-tools/insomnia-vs-postman/
30. Postman vs. Insomnia, erişim tarihi Mayıs 29, 2025, https://www.postman.com/platform/postman-insomnia/
31. Top 10 Debugging APIs You Must Know: Essential Tools for Efficient Development - Apidog, erişim tarihi Mayıs 29, 2025, https://apidog.com/blog/best-debugging-apis/

32. How to chain requests using Insomnia (get token from login api to use as header for another api) - Stack Overflow, erişim tarihi Mayıs 29, 2025, https://stackoverflow.com/questions/63520613/how-to-chain-requests-using-insomnia-get-token-from-login-api-to-use-as-header
33. How to identify suspicious network activities using Wireshark in Cybersecurity - LabEx, erişim tarihi Mayıs 29, 2025, https://labex.io/tutorials/wireshark-how-to-identify-suspicious-network-activities-using-wireshark-in-cybersecurity-415497
34. How to use Wireshark for monitoring network activity in Cybersecurity | LabEx, erişim tarihi Mayıs 29, 2025, https://labex.io/tutorials/wireshark-how-to-use-wireshark-for-monitoring-network-activity-in-cybersecurity-415120
35. Kubeshark: The Wireshark For Kubernetes - DEV Community, erişim tarihi Mayıs 29, 2025, https://dev.to/thenjdevopsguy/kubeshark-the-wireshark-for-kubernetes-3a72
36. Kubeshark — Deep Network Observability for Kubernetes, erişim tarihi Mayıs 29, 2025, https://www.kubeshark.co/
37. [HOWTO] Capture the communication of and inside a Docker container using Wireshark with Edgeshark plugin, erişim tarihi Mayıs 29, 2025, https://forums.docker.com/t/howto-capture-the-communication-of-and-inside-a-docker-container-using-wireshark-with-edgeshark-plugin/139440
38. Show HN: Subtrace – Wireshark for Docker Containers | Hacker News, erişim tarihi Mayıs 29, 2025, https://news.ycombinator.com/item?id=43096477
39. PortReference - Wireshark Wiki, erişim tarihi Mayıs 29, 2025, https://wiki.wireshark.org/PortReference
40. ProtocolReference - Wireshark Wiki, erişim tarihi Mayıs 29, 2025, https://wiki.wireshark.org/ProtocolReference
41. Inspect and analyze packet capture files - Azure Network Watcher | Microsoft Learn, erişim tarihi Mayıs 29, 2025, https://learn.microsoft.com/en-us/azure/network-watcher/packet-capture-inspect
42. DisplayFilters - Wireshark Wiki, erişim tarihi Mayıs 29, 2025, https://wiki.wireshark.org/DisplayFilters
43. Inspecting AMQP 0-9-1 Traffic using Wireshark - RabbitMQ, erişim tarihi Mayıs 29, 2025, https://www.rabbitmq.com/amqp-wireshark
44. Mastering MQTT Analysis with Wireshark: A Beginner's Guide - EMQX, erişim tarihi Mayıs 29, 2025, https://www.emqx.com/en/blog/mastering-mqtt-analysis-with-wireshark
45. AWS Cloud9 vs Codespaces comparison - PeerSpot, erişim tarihi Mayıs 29, 2025, https://www.peerspot.com/products/comparisons/aws-cloud9_vs_codespaces
46. TryHackMe_and_HackTheBox/Wireshark Traffic Analysis.md at master - GitHub, erişim tarihi Mayıs 29, 2025, https://github.com/jesusgavancho/TryHackMe_and_HackTheBox/blob/master/Wireshark%20Traffic%20Analysis.md
47. The Evolution of Networking: 2024 Review and 2025 Outlook - Zones Blog, erişim

tarihi Mayıs 29, 2025, https://blog.zones.com/the-evolution-of-networking-2024-review-and-2025-outlook
48. Documentation » DBGP - A common debugger protocol ... - Xdebug, erişim tarihi Mayıs 29, 2025, https://xdebug.org/docs/dbgp
49. Use port 9003 by default when sending xdebug.client_port and not xdebug.remote_port : WI-64944 - JetBrains YouTrack, erişim tarihi Mayıs 29, 2025, https://youtrack.jetbrains.com/issue/WI-64944/Use-port-9003-by-default-when-sending-xdebug.clientport-and-not-xdebug.remoteport
50. Pentesting JDWP - Java Debug Wire Protocol - HackTricks, erişim tarihi Mayıs 29, 2025, https://angelica.gitbook.io/hacktricks/network-services-pentesting/pentesting-jdwp-java-debug-wire-protocol
51. Chrome Remote DevTools | Practical CTF, erişim tarihi Mayıs 29, 2025, https://book.jorianwoltjer.com/web/chrome-remote-devtools
52. Chrome DevTools Protocol - GitHub Pages, erişim tarihi Mayıs 29, 2025, https://chromedevtools.github.io/devtools-protocol/
53. Java Debug Wire Protocol (JDWP) with Native Image - GraalVM, erişim tarihi Mayıs 29, 2025, https://www.graalvm.org/jdk25/reference-manual/native-image/debugging-and-diagnostics/JDWP/
54. Inspect network activity | Chrome DevTools, erişim tarihi Mayıs 29, 2025, https://developer.chrome.com/docs/devtools/network
55. Using Chrome Devtools with Wireshark - Cisco Meraki Documentation, erişim tarihi Mayıs 29, 2025, https://documentation.meraki.com/General_Administration/Other_Topics/Using_Chrome_Devtools_with_Wireshark
56. Lua - Wireshark Wiki, erişim tarihi Mayıs 29, 2025, https://wiki.wireshark.org/lua
57. How to Write Wireshark Dissector - Sewio RTLS, erişim tarihi Mayıs 29, 2025, https://www.sewio.net/open-sniffer/develop/how-to-write-wireshark-dissector/