

Instagram Phishing Scanner: Kontrol Adımları Detaylı Açıklaması

Instagram Phishing Scanner, kullanıcıların Instagram bağlantılarını analiz ederek kimlik avı (phishing) risklerini tespit eden bir web uygulamasıdır. InstagramPhishingDetector sınıfı, URL'yi çeşitli açılardan inceleyen bir dizi kontrol adımı içerir. Aşağıda, her kontrol adımı **çok detaylı** bir şekilde açıklanmış, her terim (typosquatting, punycode, WHOIS, DNS, SSL, vb.) tanımlanmış ve kodun nasıl çalıştığı uzun metinsel açıklamalarla sunulmuştur.

1. İlk URL Doğrulama ve Şüpheli Alan Adı Kontrolü

Ne Yapıyor?

Bu adım, URL'nin temel geçerliliğini kontrol eder ve domainin bilinen şüpheli alan adları listesiyle eşleşip eşleşmediğini inceler. Geçersiz bir URL (ör. eksik protokol veya domain) veya şüpheli bir domain (ör. "trycloudflare.com") hemen phishing olarak işaretlenir.

Terim Açıklamaları:

- **URL (Uniform Resource Locator):** Bir web adresidir (ör. <https://instagram.com>). Protokol (https), domain (instagram.com), yol (/login), sorgu parametreleri (?id=123) gibi bileşenlerden oluşur.
- **Şüpheli Alan Adı:** Phishing siteleri tarafından sık kullanılan domainlerdir (ör. "trycloudflare.com"). Bunlar, genellikle geçici veya anonim hosting hizmetleriyle ilişkilidir.

Nasıl Çalışıyor?

- **Kod Mantığı:**
 - `analyze_url` metodunda, `urlparse(url)` (Python'un `urllib.parse` modülünden) URL'yi parçalarına ayırır: `scheme` (protokol, örn. https), `netloc` (domain, örn. instagram.com), `path`, `query`, vb.
 - Eğer `scheme` veya `netloc` eksikse (ör. `instagram.com` yerine sadece `login`), URL geçersiz kabul edilir ve sonuç:
 - `is_phishing: True`

- confidence: 1.0
- reason: "Invalid URL format"
- details: Boş bir default_details yapısı.
- Geçerli bir URL ise, netloc'tan original_domain alınır (www. çıkarılır, örn. www.instagram.com → instagram.com).
- original_domain, self.suspicious_domains listesindeki her domain ile karşılaştırılır (ör. "trycloudflare.com").
- Eşleşme varsa:
 - details["domain_analysis"]["suspicious_domain"]'a şu bilgiler eklenir:
 - detected: True
 - matched_domain: "trycloudflare.com"
 - note: "Şüpheli alan adı 'trycloudflare.com' tespit edildi"
 - Sonuç:
 - is_phishing: True
 - confidence: 0.95 (yüksek güven, çünkü şüpheli domain kesin bir göstergedir)
 - reason: "Şüpheli alan adı 'trycloudflare.com' tespit edildi"



Kullanılan Kütüphaneler:

- **urllib.parse.urlparse:** URL'yi bileşenlerine ayırır. Örneğin, <https://fake.com/login?id=123> → scheme: https, netloc: fake.com, path: /login, query: id=123.

Hata Yönetimi:

- Geçersiz URL formatı için analiz durur ve hata mesajı döner.
- Şüpheli domain eşleşmesi, diğer analizlere gerek kalmadan phishing sonucunu kesinleştirir.

Frontend Entegrasyonu:

- **Jinja2 Şablonu:** result.html'de, result["reason"] (ör. "Şüpheli alan adı 'trycloudflare.com' tespit edildi") .result-box içinde bir <p> etiketiyle gösterilir.
- **Görsel Vurgu:** is_phishing: True ise .alert sınıfıyla kırmızı bir uyarı () ve color: #ff3333, alertPulse animasyonu ile dikkat çekilir.
- **Örnek Çıktı:** .result-box'ta: “  Kimlik Avı Tespit Edildi: Şüpheli alan adı 'trycloudflare.com' tespit edildi.”

Örnek:

- **URL:** <https://trycloudflare.com/fake-login>
- **Sonuç:**
 - `is_phishing`: True
 - `confidence`: 0.95
 - `.result-box`: “Şüpheli alan adı 'trycloudflare.com' tespit edildi”
 - `.alert`: “⚠️ Kimlik Avı Tespit Edildi”

Neden Önemli?

Bu adım, analizin ilk filtresidir. Geçersiz URL’ler veya bilinen şüpheli domainler, phishing’in açık göstergeleridir ve kaynakları verimli kullanmak için hemen işaretlenir.

2. Kısaltılmış URL Kontrolü

Ne Yapıyor?

URL’nin bir URL kısaltma servisine ait olup olmadığını kontrol eder (ör. `bit.ly`, `tinyurl.com`). Kısaltılmışsa, nihai yönlendirme URL’sini alır ve bu URL’nin domaini resmi Instagram domainlerinden biri değilse phishing olarak işaretler.

Terim Açıklamaları:

- **Kısaltılmış URL:** Uzun URL’leri kısa hale getiren servislerdir (ör. `bit.ly/abc` → <https://fake.com>). Phishing’de sık kullanılır, çünkü gerçek hedefi gizler.
- **Yönlendirme:** Bir URL’nin başka bir URL’ye otomatik olarak yönlendirmesidir (HTTP 301/302). Phishing siteleri, kullanıcıyı sahte bir sayfaya yönlendirmek için bunu kullanır.

Nasıl Çalışıyor?

- **Kod Mantığı:**
 - `analyze_url`’de, `original_domain` (URL’nin netloc kısmı, örn. `bit.ly`) `self.shortening_domains` listesindeki servislerle karşılaştırılır (`is_shortened`).
 - Eğer kısaltılmışsa, `requests.get(url, allow_redirects=True, timeout=10)` ile nihai URL (`final_url`) alınır.
 - `requests` kütüphanesi, HTTP yönlendirmelerini takip eder ve son URL’yi döner.
 - Örneğin, <https://bit.ly/abc> → <https://fake.com/login>.

- `urlparse(final_url)` ile `final_domain` çıkarılır (`www.` çıkarılır, örn. www.fake.com → `fake.com`).
- Eğer `is_shortened` doğruysa ve `final_domain` `self.legitimate_domains`'te (ör. `instagram.com`) değilse:
 - Sonuç:
 - `is_phishing: True`
 - `confidence: 0.9` (yüksek güven, çünkü kısaltılmış URL'ler phishing'de yaygındır)
 - `reason: "Kısaltılmış URL Instagram dışı bir domaine yönlendiriyor"`
 - `details`'a eklenir:
 - `shortening_service: "bit.ly"`
 - `final_domain: "fake.com"`
- **Hata Yönetimi:**
 - `requests.get` başarısızsa (ör. zaman aşımı, bağlantı hatası), orijinal URL (`url`) ve domain (`original_domain`) kullanılır.
 - Hata, `except Exception as e` ile yakalanır ve analiz devam eder.

Kullanılan Kütüphaneler:

- **requests:** HTTP istekleri yapar ve yönlendirmeleri takip eder. `allow_redirects=True` ile tüm 301/302 yönlendirmeleri izlenir.
- **urllib.parse.urlparse:** Nihai URL'yi parçalara ayırır.

Hata Yönetimi:

- Bağlantı hataları için analiz orijinal URL ile devam eder, böylece kullanıcıya bir sonuç sunulur.
- Nihai domain alınamazsa, sonraki adımlara geçilir.

Frontend Entegrasyonu:

- **Jinja2 Şablonu:** `result.html`'de, `result["reason"]` (ör. "Kısaltılmış URL Instagram dışı bir domaine yönlendiriyor") `.result-box`'ta gösterilir.
- **Görsel Vurgu:** `is_phishing: True` ise `.alert` ile kırmızı uyarı, `color: #ff3333`, `alertPulse` animasyonu.
- **Detaylar:** `details["shortening_service"]` ve `details["final_domain"]`, `.result-box`'ta bir liste (``, ``) içinde: "Kısaltma Servisi: bit.ly, Nihai Domain: fake.com".
- **Örnek Çıktı:** `.result-box`'ta: "⚠ Kimlik Avı: Kısaltılmış URL (bit.ly) fake.com'a yönlendiriyor."

Örnek:

- **URL:** <https://bit.ly/fake-login>
- **Nihai URL:** <https://fake.com/login>
- **Sonuç:**
 - `is_phishing`: True
 - `confidence`: 0.9
 - `.result-box`: “Kısaltılmış URL: bit.ly, Nihai Domain: fake.com”
 - `.alert`: “⚠ Kimlik Avı Tespit Edildi”

Neden Önemli?

Kısaltılmış URL’ler, phishing’in yaygın bir aracıdır çünkü kullanıcı gerçek hedefi göremez. Bu kontrol, gizli yönlendirmeleri ortaya çıkarır ve sahte siteleri hızlıca tespit eder.

3. Resmi Instagram Domain Kontrolü

Ne Yapıyor?

URL’nin domaininin resmi Instagram domainlerinden biri olup olmadığını kontrol eder (`self.legitimate_domains`, örn. `instagram.com`, `help.instagram.com`). Eğer resmi bir domainse, site güvenilir kabul edilir.

Terim Açıklamaları:

- **Resmi Domain:** Instagram’a ait doğrulanmış alan adlarıdır. Phishing siteleri, bu domainleri taklit edemez çünkü doğrudan Instagram tarafından kontrol edilir.
- **Domain:** Bir URL’nin ana adres kısmıdır (ör. `instagram.com`). Alt alan adları (`help.instagram.com`) da resmi olabilir.

Nasıl Çalışıyor?

- **Kod Mantığı:**
 - `analyze_url`’de, `final_domain` (yönlendirmelerden sonra alınan domain) `self.legitimate_domains` listesindeki domainlerle karşılaştırılır.
 - Eşleşme varsa:
 - `_analyze_domain(final_domain)` çağrılır, domain analizi yapılır (typosquatting, domain_age, vb.), ancak bu sadece bilgi amaçlıdır.

- `_analyze_url_structure(url)` çağrılır, URL yapısı analiz edilir (şüpheli kelimeler, vb.), yine bilgi amaçlı.
- Sonuç:
 - `is_phishing: False`
 - `confidence: 0.95` (resmi domain olduğu için yüksek güven)
 - `reason: "Resmi Instagram domaini"`
- `default_details`'a `domain_analysis` ve `url_analysis` eklenir.
- Analiz burada biter, diğer kontroller (SSL, içerik, vb.) yapılmaz.

Kullanılan Kütüphaneler:

- Yok, sadece string karşılaştırma (`in` operatörü).

Hata Yönetimi:

- Hata yok, çünkü bu basit bir liste kontrolü.

Frontend Entegrasyonu:

- **Jinja2 Şablonu:** `result.html`'de, `result["reason"]` (ör. "Resmi Instagram domaini") `.result-box`'ta `<p>` ile gösterilir.
- **Görsel Vurgu:** `is_phishing: False` ise `.safe` sınıfıyla yeşil onay (✅), `color: #00ff00`, `font-family: Orbitron`.
- **Detaylar:** `details["domain_analysis"]` (ör. domain yaşı) ve `details["url_analysis"]` (ör. şüpheli kelimeler) `.result-box`'ta liste halinde (``, ``).
- **Örnek Çıktı:** `.result-box`'ta: "✅ Güvenilir Site: Resmi Instagram domaini", `.safe:` "✅ Güvenilir site".

Örnek:

- **URL:** <https://help.instagram.com/login>
- **Sonuç:**
 - `is_phishing: False`
 - `confidence: 0.95`
 - `.result-box:` "Resmi Instagram domaini"
 - `.safe:` "✅ Güvenilir site"

Neden Önemli?

Resmi Instagram domainleri, doğrudan güvenilir olduğundan, bu kontrol gereksiz analizleri önler ve kullanıcıya hızlı bir onay sunar.

4. Domain Analizi (_analyze_domain)

Ne Yapıyor?

Domainin phishing göstergelerini detaylı bir şekilde analiz eder. Kontroller: şüpheli domain (tekrar), **typosquatting**, **punycode** (homografik saldırı), **alan adı yaşı**, ve **DNS kayıtları**.

Terim Açıklamaları:

- **Typosquatting:** Gerçek bir domainin yazım hatalarıyla taklit edilmesi (ör. `instagramn.com` yerine `instagram.com`). Kullanıcıları yanıltmak için kullanılır.
- **Punycode (Homografik Saldırı):** Farklı alfabelerdeki benzer görünen karakterlerle domain oluşturma (ör. `xn--nstgram.com` → `instagram.com`). Görsel olarak orijinale benzer ama farklıdır.
- **Alan Adı Yaşı:** Domainin kayıt tarihi. Yeni domainler (ör. <30 gün) phishing için daha risklidir, çünkü geçici siteler genellikle yeni alınır.
- **DNS Kayıtları (NS):** Domain Name System kayıtları, domainin hangi isim sunucularına bağlı olduğunu gösterir. Az veya hiç NS kaydı, şüpheli bir işarettir.
- **WHOIS:** Domainin sahibi, kayıt tarihi, kayıt şirketi gibi bilgilerini içeren bir veritabanıdır.

Nasıl Çalışıyor?

- **Başlangıç:** `score = 1.0` (mükemmel güven), `details = {}` (sonuçları saklar).
- **Adımlar:**
 - **Şüpheli Alan Adı (Tekrar Kontrol):**
 - `domain`, `self.suspicious_domains`'te aranır (ör. `trycloudflare.com`).
 - Eşleşirse:
 - `suspicious_domain_score = 0.2` (düşük güven).
 - `details["suspicious_domain"]`'a:
 - `detected: True`
 - `matched_domain: "trycloudflare.com"`
 - `score: 0.2`
 - `note: "Şüpheli alan adı 'trycloudflare.com' tespit edildi"`

- Eşleşme yoksa: `detected: False, score: 1.0`.
- `score *= suspicious_domain_score`.
- **Frontend:** `.result-box`'ta "Şüpheli Alan Adı: trycloudflare.com" (``).

○ **Typosquatting:**

- **Nasıl Çalışır?:** Her `legitimate_domains` (ör. `instagram.com`) ile `domain` arasında **Levenshtein mesafesi** hesaplanır. Bu mesafe, iki string arasındaki minimum düzenleme sayısıdır (ekleme, silme, değiştirme).
- **Kod:**
 - `Levenshtein.distance(domain, legit_domain)` ile mesafe hesaplanır.
 - En küçük mesafe (`min_distance`) ve en yakın domain (`closest_domain`) bulunur.
 - `normalized_distance = min_distance / max(len(domain), len(closest_domain))` (0-1 arası normalize edilir).
 - Skorlama:
 - `0 < normalized_distance < 0.3`: `typo_score = 0.2` (yüksek typosquatting riski, örn. `instagrarn.com`).
 - `< 0.5`: `typo_score = 0.5` (orta risk).
 - Aksi halde: `typo_score = 1.0`.
 - `details["typosquatting"]`'a:
 - `score: typo_score`
 - `closest_legitimate_domain: "instagram.com"`
 - `normalized_distance: 0.1` (örneğin)
- **Frontend:** `.result-box`'ta "Benzer Alan: instagram.com, Benzerlik: 90%" (``).
- **Örnek:** `instagrarn.com` için `min_distance = 1` (bir harf farkı), `normalized_distance ≈ 0.1`, `typo_score = 0.2`.

○ **Punycode (Homografik Saldırı):**

- **Nasıl Çalışır?:** Punycode, uluslararası karakterleri (ör. Kiril, Çince) ASCII'ye çevirir (ör. `instagram.com` → `xn--nstgram.com`). Phishing'de, görsel olarak orijinale benzer domainler için kullanılır.
- **Kod:**
 - `domain.startswith("xn--")` kontrol edilir.
 - Varsa: `score *= 0.3` (yüksek risk).
 - `details["punycode"]`'a:

- detected: True
 - impact: "Severe (-70%)"
 - Yoksa: detected: False.
- **Frontend:** .result-box'ta "Punycode Tespit Edildi: Yüksek risk" ().
- **Örnek:** xn--nstgram.com → .result-box: "Punycode: Evet, Risk: -70%".
- **Alan Adı Yaşı:**
 - **Nasıl Çalışır?:** WHOIS veritabanı, domainin ne zaman kaydedildiğini (creation_date) ve diğer bilgileri (kayıt şirketi, WHOIS sunucusu) sağlar. Yeni domainler phishing için risklidir.
 - **Kod:**
 - whois_module.whois(domain) ile WHOIS sorgusu yapılır.
 - creation_date alınır (liste ise ilk eleman: creation_date[0]).
 - age_days = (datetime.now() - creation_date).days ile yaş hesaplanır.
 - Skorlama:
 - <30 gün: domain_age_score = 0.2 (çok yeni, yüksek risk).
 - <90 gün: 0.4
 - <180 gün: 0.6
 - <365 gün: 0.8
 - ≥365 gün: 1.0
 - details["domain_age"]'a:
 - creation_date: "2025-04-01"
 - age_days: 15
 - score: 0.2
 - **Hata Yönetimi:**
 - creation_date yoksa: domain_age_score = 0.4, note: "No creation date found".
 - WHOIS hatası: domain_age_score = 0.5, error: str(e).
 - **Frontend:** .result-box'ta "Alan Yaşı: 15 gün, Oluşturma: 2025-04-01" ().
 - **Örnek:** fake.com, 10 günlük → domain_age_score = 0.2.
- **DNS Kayıtları (NS):**
 - **Nasıl Çalışır?:** DNS, domainin hangi isim sunucularına (NS) bağlı olduğunu gösterir. Güvenilir domainler genellikle birden fazla NS kaydına sahiptir. Az veya hiç NS kaydı, şüpheli bir işarettir.

- **Kod:**
 - `dns.resolver.resolve(domain, 'NS')` ile NS kayıtları alınır.
 - `ns_count = len(ns_records):`
 - `0: dns_score = 0.3` (yüksek risk).
 - `<2: dns_score = 0.7` (orta risk).
 - `≥2: dns_score = 1.0`.
 - `details["dns"]`'a:
 - `ns_records: ["ns1.example.com", "ns2.example.com"]`
 - `count: 2`
 - `score: 1.0`
 - **Hata Yönetimi:** Hata olursa `dns_score = 0.5`, `error: str(e)`.
- **Frontend:** `.result-box`'ta "NS Kayıtları: 2 adet" (``).
- **Örnek:** `fake.com`, 1 NS kaydı → `dns_score = 0.7`.

Kullanılan Kütüphaneler:

- **python-whois:** WHOIS sorguları için. Domainin kayıt bilgilerini çeker.
- **Levenshtein:** Typosquatting için string mesafesi hesaplar.
- **dnspython:** DNS NS kayıtlarını sorgular.
- **datetime:** Alan adı yaşını hesaplar.

Hata Yönetimi:

- WHOIS veya DNS hataları, `score`'u düşürerek (`0.5`) ve `error` ekleyerek yönetilir.
- Eksik veri (ör. `creation_date` yok) için varsayılan düşük skorlar atanır.

Frontend Entegrasyonu:

- `details["domain_analysis"]` Jinja2 ile `result.html`'e render edilir.
- `.result-box`'ta liste (``, ``):
 - "Benzerlik: `instagram.com` (90%)"
 - "Punycod: Hayır"
 - "Alan Yaşı: 15 gün"
 - "NS Kayıtları: 2"
- Phishing tespit edilirse `.alert`, güvenilir ise `.safe`.

Örnek:

- **Domain:** `instagramn.com`

- **Sonuç:**
 - score: 0.4 (typosquatting: 0.2, yaş: 10 gün → 0.2)
 - .result-box: “Benzerlik: instagram.com (90%), Alan Yaşı: 10 gün, NS: 1”
 - .alert: “⚠ Kimlik Avı Tespit Edildi”

Neden Önemli?

Domain, bir sitenin kimliğidir. Typosquatting, punycode, yeni domainler ve zayıf DNS, phishing’in güçlü göstergeleridir. Bu analiz, sahte domainleri derinlemesine tespit eder.

5. URL Yapısı Analizi (_analyze_url_structure)

Ne Yapıyor?

URL’nin yapısını phishing göstergeleri için analiz eder: şüpheli kelimeler, alt alan adları, Instagram markası, kodlanmış karakterler, sorgu parametreleri, yol derinliği.

Terim Açıklamaları:

- **Şüpheli Kelimeler:** Phishing URL’lerinde sık kullanılan kelimeler (ör. “login”, “verify”). Kullanıcıyı kandırmak için güven veya aciliyet hissi uyandırır.
- **Alt Alan Adları:** Domainin önündeki ek bölümler (ör. fake.instagram.com’da fake). Çok sayıda alt alan, phishing’de yaygındır.
- **Instagram Markası:** URL’de “instagram” kelimesi ama resmi domain değilse (ör. instagram-login.com), sahte site işareti.
- **Kodlanmış Karakterler:** URL’de % ile kodlanmış karakterler (ör. %20 = boşluk). Phishing’de karmaşık URL’leri gizlemek için kullanılır.
- **Sorgu Parametreleri:** URL’nin ? sonrası kısmı (ör. ?id=123). Çok sayıda parametre, phishing’de manipülasyon için kullanılır.
- **Yol Derinliği:** URL’nin / ile ayrılan bölümleri (ör. /auth/login/verify). Derin yollar, sahte sitelerde yaygındır.

Nasıl Çalışıyor?

- **Başlangıç:** score = 1.0, details = {}.
- **Adımlar:**
 - **Şüpheli Kelimeler:**
 - urlparse(url) ile hostname, path, query alınır.

- `self.suspicious_keywords` (ör. "login", "verify")
hostname + path + query'de aranır.
- Her kelime için `suspicious_word_count += 1`,
`keyword_penalty = min(0.2 * count, 0.8)`.
- `score -= keyword_penalty`.
- `details["suspicious_keywords"]`'a:
 - `found: ["login", "verify"]`
 - `count: 2`
 - `penalty: 0.4`
- **Frontend:** `.result-box`'ta "Şüpheli Kelimeler: login, verify" (``).
- **Alt Alan Adları:**
 - `hostname.split('.')` ile `subdomain_count` hesaplanır (ör. `fake.sub.instagram.com` → 2 alt alan).
 - `count > 1` ise `score -= 0.4`.
 - `details["subdomains"]`'a:
 - `count: 2`
 - `parts: ["fake", "sub", "instagram", "com"]`
 - `penalty: 0.4`
 - **Frontend:** `.result-box`'ta "Alt Alan Adları: 2 (fake, sub)" (``).
- **Instagram Markası:**
 - `hostname`'de "instagram" var ama `instagram.com` değilse,
`score -= 0.5`.
 - `details["brand_in_subdomain"]`'a:
 - `detected: True`
 - `penalty: 0.5`
 - `note: "Instagram adı alt alan adında ama ana domainde değil"`
 - **Frontend:** `.result-box`'ta "Instagram Adı Alt Alanda" (``).
- **Kodlanmış Karakterler:**
 - `path.count('%')` ile `encoded_char_count`.
 - `count > 1` ise `encoded_penalty = min(0.2 * (count-1), 0.5)`, `score -= penalty`.
 - `details["encoded_characters"]`'a:
 - `count: 3`
 - `penalty: 0.4`
 - **Frontend:** `.result-box`'ta "Kodlanmış Karakterler: 3" (``).
- **Sorgu Parametreleri:**
 - `parse_qs(query)` ile parametreler alınır (ör. `id=123&token=abc` → `["id", "token"]`).

- `param_count = len(query_params):`
 - `count > 3` ise `param_penalty = min(0.2 * (count - 3), 0.5)`, `score -= penalty`.
- `details["query_parameters"]`'a:
 - `count: 4`
 - `parameters: ["id", "token"]`
- **Frontend:** `.result-box`'ta "Sorgu Parametreleri: 4 (id, token)" (``).
- **Yol Derinliği:**
 - `path.split('/')` ile `path_parts` (ör. `/auth/login/verify` → 3).
 - `len(path_parts) > 2` ise `score -= 0.3`.
 - `details["path_depth"]`'a:
 - `depth: 3`
 - `parts: ["auth", "login", "verify"]`
 - `penalty: 0.3`
 - **Frontend:** `.result-box`'ta "Yol Derinliği: 3 (/auth/login/verify)" (``).

Kullanılan Kütüphaneler:

- `urllib.parse.urlparse`: URL'yi parçalar.
- `urllib.parse.parse_qs`: Sorgu parametrelerini ayırır.

Hata Yönetimi:

- Hatalar beklenmez, çünkü string işlemleri sağlamdır.
- `score = max(0, score)` ile negatif skor engellenir.

Frontend Entegrasyonu:

- `details["url_analysis"]` `.result-box`'ta liste olarak render edilir.
- Örnek: "Şüpheli Kelimeler: login, verify; Alt Alan: 2; Sorgu: 4".
- Phishing varsa `.alert`, yoksa `.safe`.

Örnek:

- **URL:** <https://fake.instagram-login.com/auth/verify?token=abc>
- **Sonuç:**
 - `score: 0.3`
 - `.result-box`: "Şüpheli Kelimeler: login, verify; Alt Alan: 1; Sorgu: 1"
 - `.alert`: "⚠ Kimlik Avı Tespit Edildi"

Neden Önemli?

URL yapısı, phishing'in manipülatif doğasını ortaya çıkarır. Şüpheli kelimeler, karmaşık yapılar ve sahte markalama, kullanıcıyı kandırmak için tasarlanmıştır.

6. SSL Analizi (_analyze_ssl)

Ne Yapıyor?

Domainin HTTPS desteğini, SSL/TLS sertifikasının geçerliliğini, issuer'ın güvenilirliğini ve sertifikanın son kullanma tarihini kontrol eder.

Terim Açıklamaları:

- **HTTPS:** Güvenli HTTP, SSL/TLS ile şifrelenir. Güvenilir siteler HTTPS kullanır.
- **SSL/TLS Sertifikası:** Bir domainin kimliğini doğrulayan dijital bir belgedir. Kim tarafından (issuer) verildiği ve geçerlilik süresi önemlidir.
- **Issuer:** Sertifikayı sağlayan otorite (ör. Let's Encrypt, DigiCert). Güvenilir issuer'lar, sertifikanın kalitesini gösterir.
- **Son Kullanma Tarihi:** Sertifikanın geçerli olduğu süre. Yakın tarihli veya geçmiş sertifikalar risklidir.

Nasıl Çalışıyor?

- **Başlangıç:** `score = 1.0, details = {}`.
- **Adımlar:**
 - **HTTPS Desteği:**
 - `ssl.create_default_context()` ile güvenli bir SSL bağlamı oluşturulur.
 - `socket.create_connection((domain, 443), timeout=5)` ile 443 portuna bağlanılır.
 - `context.wrap_socket(sock, server_hostname=domain)` ile SSL bağlantısı kurulur.
 - Başarılysa: `details["has_https"] = True`.
 - **Sertifika Geçerliliği:**
 - `ssock.getpeercert()` ile sertifika bilgileri alınır.
 - `not_before` ve `not_after` tarihleri `datetime.strptime` ile parse edilir.
 - `now` (şimdiki zaman) ile karşılaştırılır:
 - `now < not_before` veya `now > not_after` ise:

- `score *= 0.3`
 - `details["certificate_valid"] = False`
 - `details["validity_issue"] = "Certificate not currently valid"`
- **Son Kullanma Tarihi:**
 - `not_after - now < timedelta(days=30)` ise (30 günden az kaldıysa):
 - `score *= 0.8`
 - `details["expires_soon"] = True`
 - `details["expiration_date"] = str(not_after).`
- **Issuer Güvenilirliği:**
 - `cert['issuer']`'dan `organizationName` alınır (`issuer_o`).
 - `self.trusted_issuers`'ta (ör. "Let's Encrypt") aranır.
 - Yoksa: `score *= 0.7, details["trusted_issuer"] = False.`
 - `details["issuer"] = issuer_o.`
- **Hata Yönetimi:**
 - Bağlantı hatası (`socket.gaierror, socket.timeout`): `score = 0.4, has_https: False.`
 - SSL hatası (`ssl.SSLError`): `score = 0.3, certificate_valid: False.`
 - Diğer hatalar: `score = 0.5, error: str(e).`

Kullanılan Kütüphaneler:

- **ssl:** SSL/TLS bağlantısı ve sertifika bilgileri.
- **socket:** Domainin 443 portuna bağlantı.
- **datetime:** Sertifika tarihlerini karşılaştırma.

Hata Yönetimi:

- Bağlantı veya SSL hataları, düşük skorlarla (0.3, 0.4) yönetilir.
- Hatalar `details["error"]`'a kaydedilir.

Frontend Entegrasyonu:

- `details["ssl_analysis"] .result-box`'ta render edilir:
 - "HTTPS: Evet"
 - "Sertifika Geçerli: Hayır"
 - "Issuer: Unknown"
 - "Son Kullanma: 2025-06-01"
- Phishing varsa `.alert`, yoksa `.safe`.

Örnek:

- **Domain:** fake.com
- **Sonuç:**
 - score: 0.4
 - .result-box: “HTTPS: Hayır, Sertifika: Geçersiz”
 - .alert: “⚠ Kimlik Avı Tespit Edildi”

Neden Önemli?

HTTPS ve güvenilir sertifikalar, bir sitenin güvenilirliğini gösterir. Phishing siteleri genellikle HTTPS’siz veya sahte/geçersiz sertifikalar kullanır.

7. İçerik Analizi (_analyze_content)

Ne Yapıyor?

Sayfanın HTML içeriğini analiz eder: login formları, Instagram markalaması, dış scriptler, gizli elementler, aciliyet dili.

Terim Açıklamaları:

- **Login Formları:** Kullanıcı adı/şifre girişi için formlar. Phishing siteleri, sahte login formlarıyla kimlik bilgilerini çalar.
- **Instagram Markalaması:** Resmi Instagram logoları, metinleri veya görselleri. Sahte siteler, markalamayı taklit edebilir ama eksiklikler risk göstergesidir.
- **Dış Scriptler:** Harici sunuculardan yüklenen JavaScript kodları. Çok sayıda dış script, kötü amaçlı kod riskini artırır.
- **Gizli Elementler:** display: none veya visibility: hidden ile gizlenmiş HTML elemanları. Phishing’de kullanıcıyı manipüle etmek için kullanılır.
- **Aciliyet Dili:** Kullanıcıyı panikleten ifadeler (ör. “verify now”, “account suspended”). Phishing’de yaygındır.

Nasıl Çalışıyor?

- **Başlangıç:** score = 1.0, details = {}.
- **Adımlar:**
 - **Login Formları:**
 - requests.get(url, headers={'User-Agent': ...}) ile sayfa alınır.

- BeautifulSoup(response.text, 'html.parser') ile HTML parse edilir.
- <form>'larda input[type="password"] aranır.
- Her form için:
 - inputs sayısı, action, method kaydedilir.
 - inputs <= 4 ise şüpheli (basit formlar phishing'de yaygındır).
 - Şüpheliyse: score *= 0.4, details["suspicious_forms"]["detected"] = True.
- **Frontend:** .result-box'ta "Şüpheli Login Formu Tespit Edildi" ().
- **Instagram Markalaması:**
 - soup.find_all() ile:
 - img[src|alt~=instagram]
 - string~=instagram
 - img[alt~=logo]
 - Herhangi biri varsa: has_instagram_branding = True.
 - Yoksa ve URL'de "instagram" varsa: score *= 0.5.
 - details["instagram_branding"]'a detected, reason.
 - **Frontend:** .result-box'ta "Instagram Markalaması Yok" ().
- **Dış Scriptler:**
 - script[src^=https?://] sayısı >3 ise score *= 0.6.
 - details["external_scripts"]'a count, reason.
 - **Frontend:** .result-box'ta "Dış Scriptler: 5" ().
- **Gizli Elementler:**
 - style=display:none|visibility:hidden sayısı >1 ise score *= 0.5.
 - details["hidden_elements"]'a count, reason.
 - **Frontend:** .result-box'ta "Gizli Elementler: 2" ().
- **Aciliyet Dili:**
 - soup.get_text().lower()'da self.phishing_phrases (ör. "verify now") aranır.
 - Her ifade için urgency_score += 1, urgency_penalty = min(0.2 * score, 0.7).
 - score *= (1 - penalty).
 - details["urgency_language"]'a phrases, reason.
 - **Frontend:** .result-box'ta "Acil Durum İfadeleri: verify now" ().

Kullanılan Kütüphaneler:

- **requests**: Sayfa içeriğini çeker.
- **BeautifulSoup**: HTML'i parse eder.
- **re**: Regex ile markalama ve script aramaları.

Hata Yönetimi:

- HTTP hatası (`status_code != 200`): `score = 0.5, details["error"]`.
- Genel hata: `score = 0.5, details` sıfırlanır ve error eklenir.

Frontend Entegrasyonu:

- `details["content_analysis"]` .result-box'ta liste:
 - “Şüpheli Form: Evet”
 - “Markalama: Hayır”
 - “Acil Durum: verify now”
- Phishing varsa `.alert`, yoksa `.safe`.

Örnek:

- **URL**: <https://fake.com/login>
- **Sonuç**:
 - `score`: 0.3
 - `.result-box`: “Şüpheli Form, Aciliyet: verify now”
 - `.alert`: “⚠ Kimlik Avı Tespit Edildi”

Neden Önemli?

Sayfa içeriği, phishing'in kullanıcıyı kandırma yöntemlerini ortaya çıkarır. Sahte formlar ve aciliyet dili, phishing'in temel araçlarıdır.

8. Sahte Kimlik Testi (`_test_honey_credentials`)

Ne Yapıyor?

Sahte kullanıcı adı ve şifre ile login testi yaparak sitenin phishing davranışını kontrol eder. Sahte kimlik kabul

Kullanılan kütüphaneler:

- **selenium**: Otomatik tarayıcı kontrolü.
- **webdriver_manager.chrome**: ChromeDriver yönetimi.
- **random**: Rastgele kimlik oluşturma.

Nasıl Çalışıyor?

- **Başlangıç:** `details = {}`.
- **Adımlar:**
 - **WebDriver Başlatma:**
 - `webdriver.Chrome` ile headless Chrome başlatılır (`--headless, --no-sandbox`).
 - `ChromeDriverManager().install()` ile sürücü yüklenir.
 - **Sahte Kimlik Girişi:**
 - `fake_username` ve `fake_password` rastgele oluşturulur (ör. `test_user_12345, TestP@ss67890`).
 - `driver.get(url)`, `WebDriverWait` ile sayfa yüklenir.
 - `input[name='username']` veya benzeri aranır, kimlik bilgileri girilir.
 - `button[type='submit']` tıklanır.
 - **Sonuç Kontrolü:**
 - `original_url` ve `current_url` karşılaştırılır (`details["redirected"]`).
 - Başarı (`/home, "welcome"`), hata (`"incorrect"`), 2FA (`"verification code"`) göstergeleri aranır.
 - Instagram domainine yönlendirme:
 - 2FA olmadan:
`redirected_to_legit_instagram_without_2fa: True, credentials_accepted: True.`
 - 2FA ile: `two_factor_detected: True, credentials_accepted: True.`
 - **2FA Testi:**
 - 2FA ekranı varsa, `input[name='code']`'a `fake_code` (rastgele 6 hane) girilir.
 - `twofa_submit_button` tıklanır, sonuç kontrol edilir.
 - Sahte kod kabul edilirse: `two_factor_accepted_fake_code: True.`
 - **Hata Yönetimi:**
 - Hata: `details["error"], return False, details.`
 - `driver.quit()` ile tarayıcı kapanır.

Frontend Entegrasyonu:

- `details["honey_credentials_test"].result-box'ta:`
 - "Sahte Kimlik Kabul: Evet"
 - "2FA Tespit: Evet"

- “Sahte Kod Kabul: Hayır”
- Phishing varsa .alert, yoksa .safe.

Örnek:

- URL: <https://fake.com/login>
- Sonuç:
 - is_confirmed_phishing: True
 - .result-box: “Sahte Kimlik Kabul Edildi”
 - .alert: “⚠ Kimlik Avı Tespit Edildi”

Neden Önemli?

Sahte kimlik testi, phishing sitelerinin gerçek davranışını ortaya çıkarır. Sahte kimliklerin kabul edilmesi, kesin bir phishing göstergesidir.

Sonuç

InstagramPhishingDetector, URL’yi çok yönlü analiz eder:

- **İlk Doğrulama:** Geçersiz veya şüpheli domainleri tespit eder.
- **Kısaltılmış URL:** Yönlendirmeleri kontrol eder.
- **Resmi Domain:** Güvenilir Instagram domainlerini onaylar.
- **Domain Analizi:** Typosquatting, punycode, yaş, DNS.
- **URL Yapısı:** Şüpheli kelimeler, alt alanlar, parametreler.
- **SSL:** Sertifika güvenilirliği.
- **İçerik:** Formlar, markalama, aciliyet dili.
- **Sahte Kimlik:** Gerçek phishing davranışını test eder.

Frontend Entegrasyonu:

- Jinja2 ile result.html’de .result-box, .alert, .safe, .recommendation kullanılır.
- Örnek: “⚠ Kimlik Avı: Sahte kimlik kabul edildi, öneri: Kimlik bilgilerinizi girmeyin!”