

LEGACY

Siber Güvenlikte Düello:
Legacy Zihniyeti vs. Zero Trust Mimarisi
ZTDALD Projesi Üzerinden Somut Bir Karşılaştırma



ZERO TRUST



"Kodun kendisi konuşacak. Hangi mimarinin ayakta kalacağını görmek için ringe çıkıyoruz."

İki Kale, İki Felsefe



Geleneksel Güvenlik (Kale ve Hendek Modeli)

- Zorlu bir dış çevre, "güvenilir" bir iç ağ varsayımlına dayanır.
- Bir saldırgan surları aştığından, içerisinde serbestçe hareket edebilir.



Sıfır Güven Modeli (Zero Trust)

- Güven diye bir şey yoktur. Ağın içi veya dışı fark etmeksiz her istek doğrulanmalıdır.
- Erişim, "bilinmesi gereken" ve "en az ayrıcalık" ilkelerine göre katı bir şekilde kontrol edilir.

Tanışın: Rakiplerimiz

Meydan Okuyan - Legacy Sistem (L/)

Yaygın güvenlik açıklarıyla dolu, 'böyle yapılmamalı' dedirten savunmasız bir parola kasası. Eski alışkanlıkların ve ihmallerin bir ürünü.

```
└── L/  
    ├── index.php  
    └── config.php
```



Tüm mantık ve zayıflıklar tek bir dosyada

Şampiyon - Zero Trust Sistemi (ZT/)

Modern güvenlik prensiplerini uygulayan, her isteği sorulayan ve her veriyi koruyan güçlendirilmiş bir parola kasası. Güvenlik, sonradan eklenen bir özellik değil, mimarinin temelidir.

```
└── ZT/  
    ├── index.php  
    └── config.php  
    └── database.php  
    └── security_config.php  
    └── security_helper.php
```



Güvenlik fonksiyonları modüller ve merkezi

Raund 1: SQL Injection - Veritabanına Açılan Arka Kapı

'Kullanıcıdan Gelen Veriye Asla Güvenme' İlkesinin İhlali

LEGACY Saldırı Altında

L/index.php - Savunmasız Sorgu

```
<?php  
$username = $_POST['username'];  
$password = $_POST['password'];  
  
$sql = "SELECT * FROM l_users WHERE  
username = '$username' AND password =  
'$password"';
```

// ... sorgu çalıştırılır ...



Saldırganın Gözünden

Bir saldırgan bu koda baktığında ne görür? Bir davetiye.

Kullanıcı Adı:

' OR '1'='1' --

Sonuç

Oluşan SQL sorgusu:

SELECT * FROM l_users WHERE username = '' OR
'1'='1' --' AND password = '....'

Bu sorgu, '1'='1' koşulunun her zaman doğru olması nedeniyle parola kontrolünü atlar ve veritabanındaki tüm kullanıcıları (genellikle ilk kullanıcı olan 'admin'i) geri döndürür.

Raund 1: SQL Injection - Zero Trust Savunması

Parametreli Sorgular (Prepared Statements) ile Saldırıyı Etkisiz Hale Getirme

ZERO TRUST Karşı Koyuyor

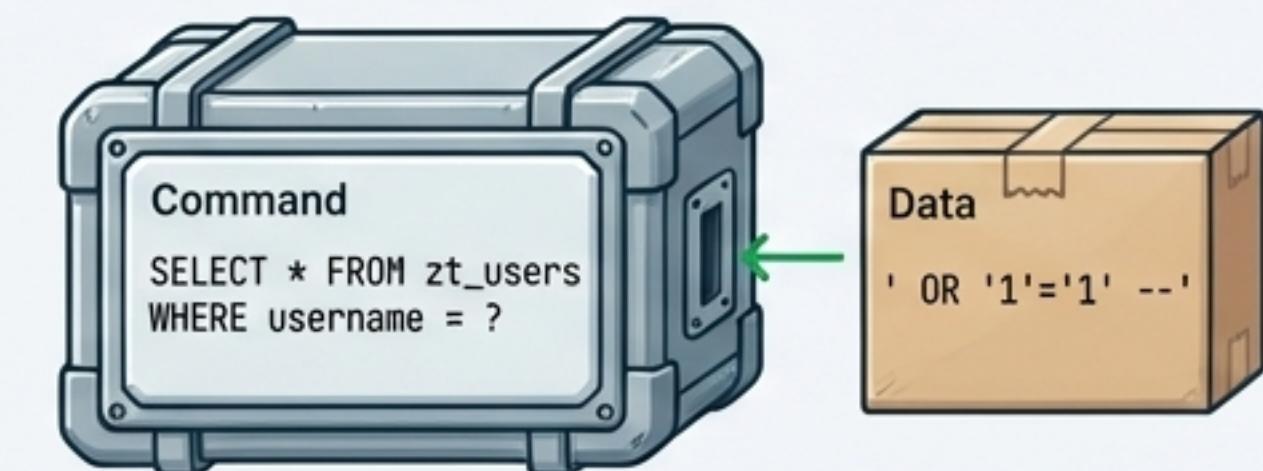
'ZT/index.php' & 'database.php' - Güçlendirilmiş Soru

```
// database.php içinde güvenli bir fonksiyon
function secure_query($sql, $params) {
    $stmt = $this->pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt;
}
```

```
// ZT/index.php kullanımı
$sql = "SELECT * FROM zt_users WHERE username = ?";
$stmt = secure_query($sql, [$username]);
```

Savunmacının Aklından

Gelen veri (' OR '1'='1' --') asla kod olarak yorumlanmaz. Veritabanı bunu sadece anlamsız bir metin dizisi olarak görür ve username sütununda eşleşme arar.



Sonuç

Saldırı başarısız. Veritabanı, bu isimde bir kullanıcı bulamaz. Kapı kapalı kalır.

KAZANAN: ZERO TRUST

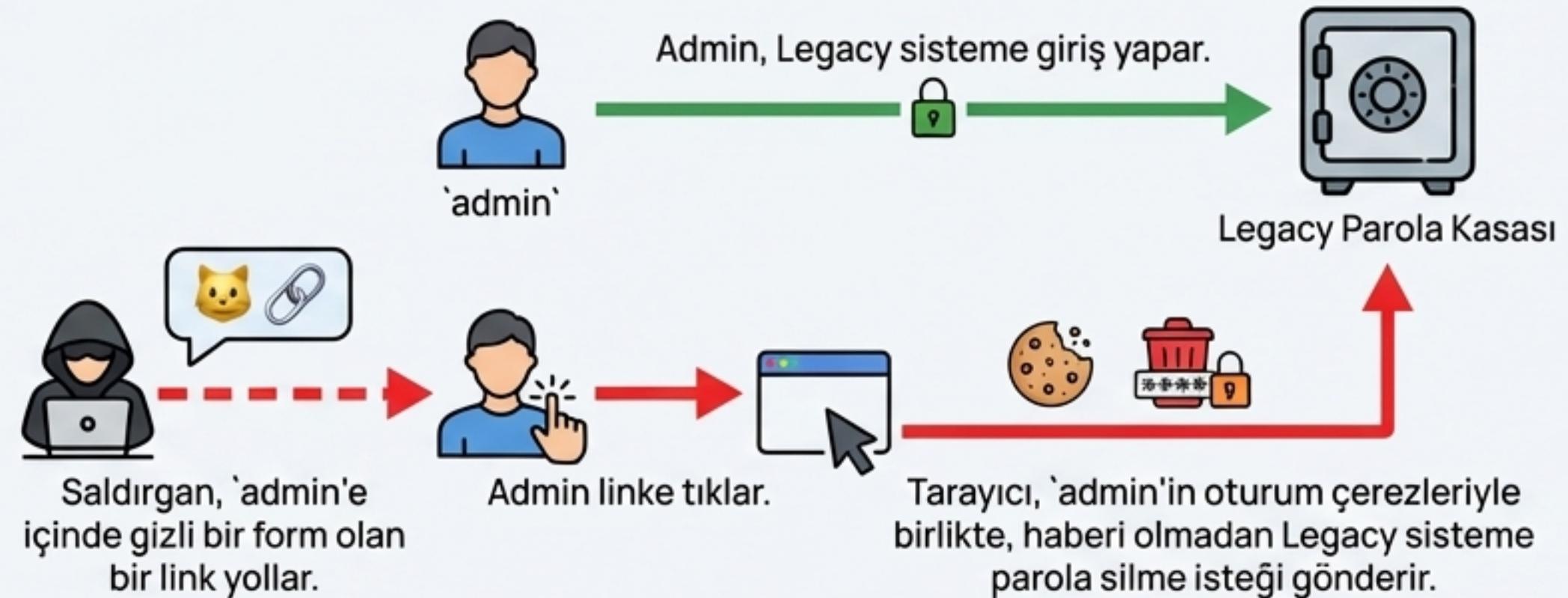
Raund 2: CSRF - Kullanıcının Oturumunu Gasp Etme Sanatı

'Her İstek Gerçekten Kullanıcıdan mı Geliyor?'

LEGACY Tuzağa Düşüyor

Senaryo

Kullanıcı 'admin' olarak Legacy parola kasasına giriş yapmış durumda. Saldırıgan, 'admin'e sahte bir "Kedi Videosu İzle" linki gönderiyor.



Kod Açığı

Legacy sistemde, parola silme isteğinin gerçekten kullanıcının kendi arayüzünden mi, yoksa sahte bir siteden mi geldiğini kontrol eden hiçbir mekanizma yok.

L/index.php

```
// Parola silme işlemi  
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    // Token kontrolü yok!  
    // ... parola silme ...  
}
```

Raund 2: CSRF - Zero Trust'ın 'Sihirli Bilet' Savunması

Anti-CSRF Tokenları ile İsteklerin Kimliğini Doğrulama

ZERO TRUST Tetikte

Savunma Mekanizması

Zero Trust mimarisi, her oturum için benzersiz ve tek kullanımı bir 'bilet' (CSRF token) üretir. Bu bilet olmadan gönderilen hiçbir hassas işlem (parola ekleme/silme) kabul edilmez.



1. Token Üretimi (`security_helper.php`)

```
// Oturuma özel, rastgele bir token üretilir.  
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));  
...
```



```
// Oturuma özel, rastgele bir token üretilir.  
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

2. Forma Ekleme (`ZT/index.php`)

```
<form action="index.php"  
method="post"><input  
type="hidden" name="csrf_token"  
echo $_SESSION['csrf_token']; ?>>  
...</form>
```

3. Doğrulama (`ZT/index.php`)

```
if ($_POST['csrf_token'] !==  
$_SESSION['csrf_token']) {  
die("Geçersiz CSRF token!");  
}
```

Sonuç

Saldırganın sitesi, bu 'sihirli bilete' sahip olmadığı için gönderdiği istek anında reddedilir. Kullanıcının oturumu güvendedir.

KAZANAN: ZERO TRUST

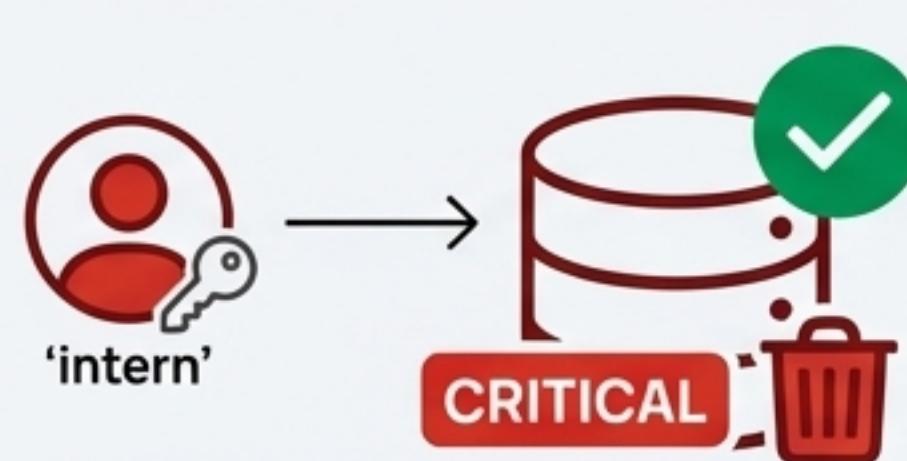
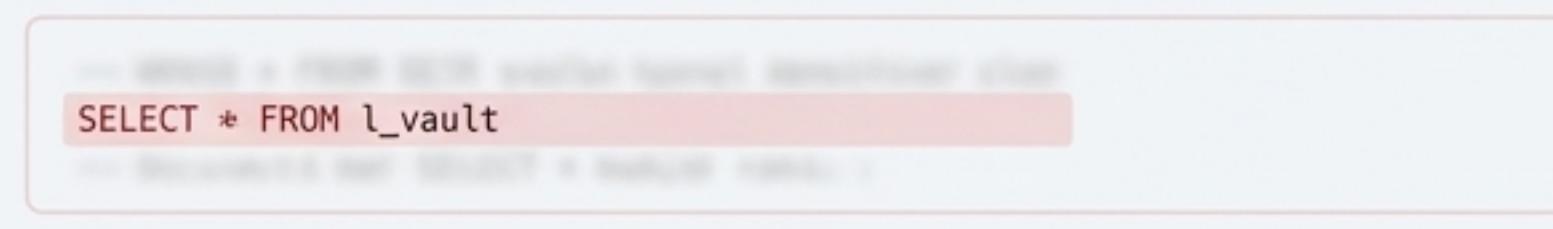
Raund 3: Yetkisiz Erişim - 'Anahtarın Var Ama Bu Odaya Giremezsin'

Rol Tabanlı Erişim Kontrolü (RBAC) vs. Kontrolsüz Güç

LEGACY'de Kapılar Ardına Kadar Açık

Problem

Legacy sistemde, bir kullanıcı giriş yaptığında `admin`, `tech` veya `intern` olması fark etmez. Herkes tüm parolaları görebilir ve silebilir, **'CRITICAL'** seviyedekiler dahil.

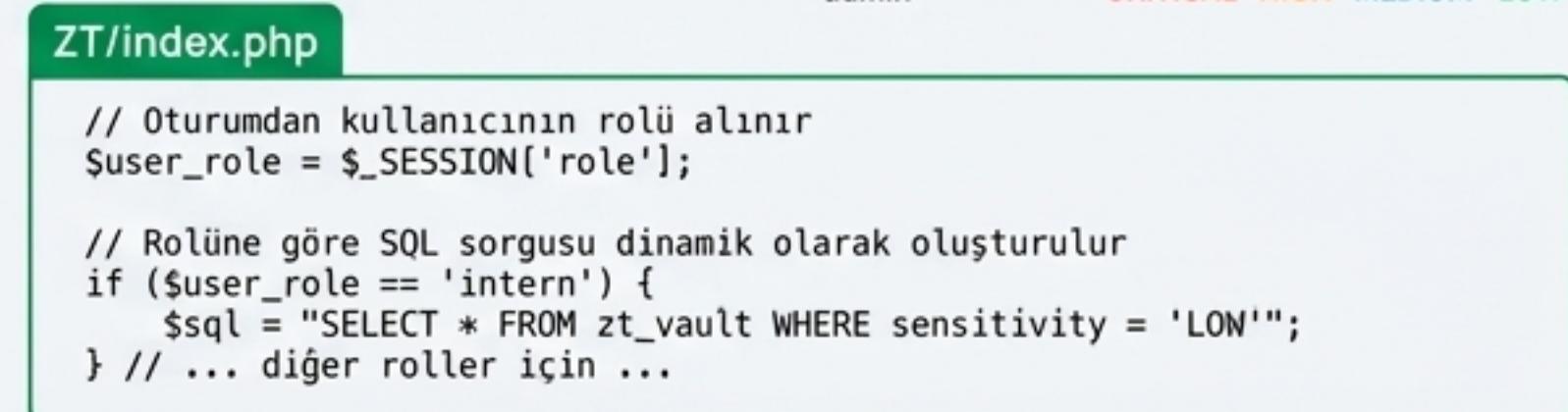
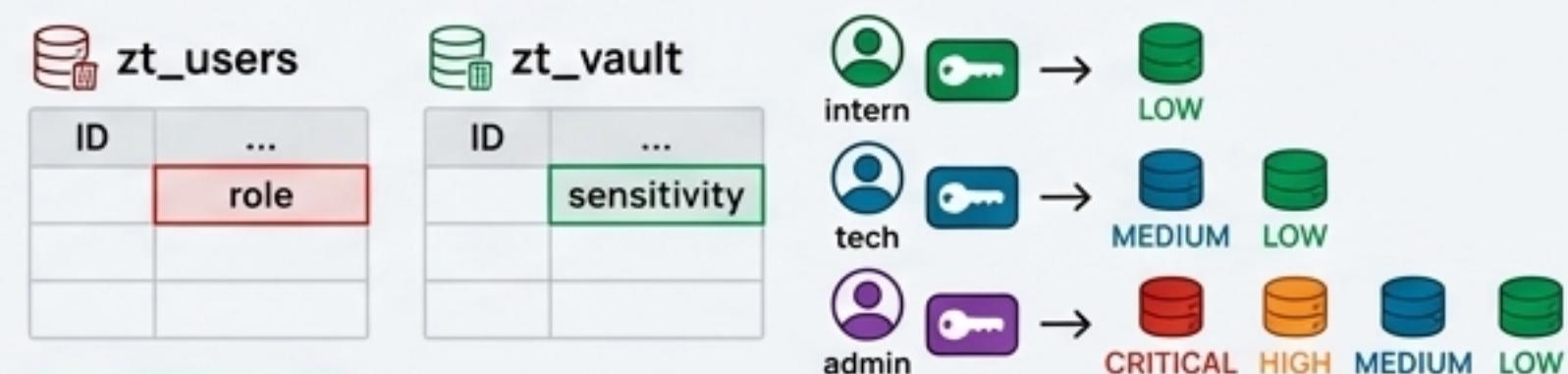


Bir stajyer ('intern'), yanlışlıkla veya kasıtlı olarak şirketin en kritik sunucu parolasını (**CRITICAL**) silebilir. Sistem buna izin verir.

ZERO TRUST'ta Herkes Yerini Bilir

Prensip

Zero Trust, "en az ayrıcalık" ilkesiyle çalışır. Her kullanıcı, sadece işini yapmak için kesinlikle gerekli olan verilere erişebilir.



KAZANAN: ZERO TRUST

Düellonun Sonucu: Nakavt!

`comprehensive_test.py` Otomatik Güvenlik Testi Sonuçları

Test Senaryosu	Legacy Sistem ('L/')	Zero Trust Sistemi ('ZT/')
SQL Injection (Kimlik Doğrulama Atlama)	FAIL	PASS
SQL Injection (Veri Sızdırma)	FAIL	PASS
CSRF (Parola Silme)	FAIL	PASS
Oturum Ele Geçirme (Session Hijacking)	FAIL	PASS
Yetki Yükseltme (Stajyer > Admin)	FAIL	PASS
Hassas Veriye Yetkisiz Erişim	FAIL	PASS
Güvenlik Başlıklar (CSP, HSTS) Kontrolü	FAIL	PASS

Rakamlar ve testler yalan söylemez. Legacy mimari, modern tehditler karşısında her rauntta başarısız olurken, Zero Trust mimarisi tüm saldırı vektörlerine karşı sağlam bir savunma hattı oluşturuyor.

Şampiyonun Cephaneliği: Zero Trust'ı Güçlü Kılan Ek Ek Savunma Katmanları



1. Güvenlik Başlıklarları (HTTP Security Headers)

`security_config.php` dosyası aracılığıyla tarayıcıya gönderilen direktiflerdir. Tarayıcıyı ek bir savunma hattına dönüştürür.

- ✓ **CSP (Content Security Policy):** Sadece güvenilir kaynaklardan script yüklenmesine izin vererek XSS riskini azaltır.
- ✓ **HSTS (Strict-Transport-Security):** Tarayıcıyı sadece HTTPS üzerinden iletişim kurmaya zorlar.
- ✓ **X-Frame-Options:** Sitenin başka siteler içine gömülmesini (Clickjacking) engeller.



2. Güvenli Oturum Yönetimi

Oturumlar sadece bir başlangıçtır. Güvenlikleri sürekli sağlanmalıdır.

- ✓ Güvenli çerez ayarları (HttpOnly, Secure), düzenli oturum ID'si yenileme, aktivite takibi ve otomatik zaman aşımı.



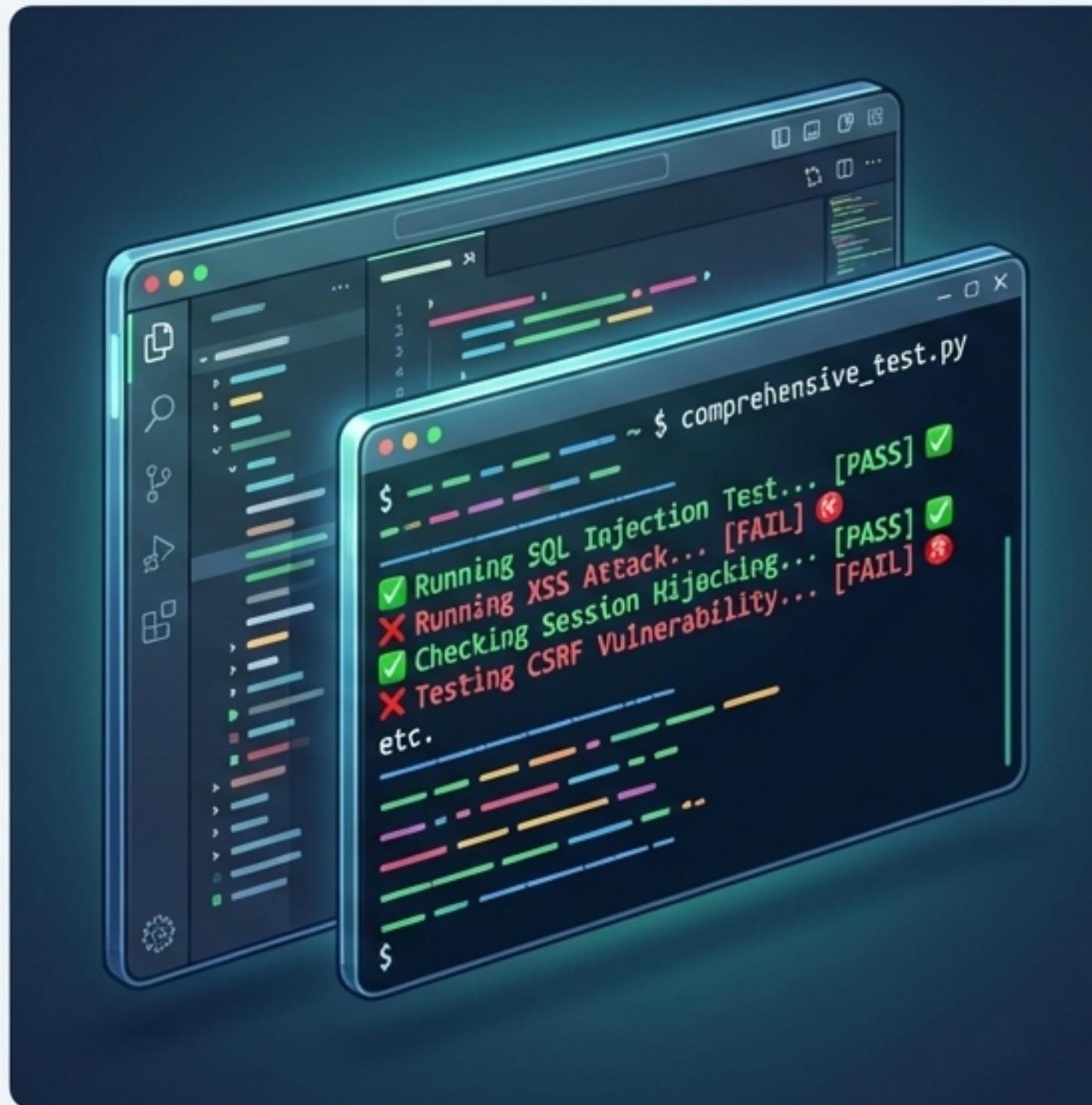
3. Kapsamlı Veri Doğrulama ve Sanitizasyon

Sisteme giren her veri potansiyel bir tehdittir. Önce temizlenir, sonra doğrulanır.

- ✓ Kullanıcıdan gelen tüm girdiler, beklenmedik karakterleri ve potansiyel saldırı kodlarını temizlemek için filtrelenir (sanitization) ve beklenen formata (örn. e-posta, sayı) uygun olup olmadığı kontrol edilir (validation).

Şimdi Sıra Sizde: Ringe Çıkın ve Kendiniz Test Edin!

Kendi başınıza bir güvenlik düello-su düzenleyin ve sistemlerinizi test edin.



1. Projeyi Klonlayın

```
bash\  
git clone [repository_url]  
cd ZTDALD
```

2. Veritabanını Oluşturun

- sql.md dosyasındaki SQL komutlarını kullanarak MySQL veritabanınızı ve tablolarınızı (`l_users`, `l_vault`, `zt_users`, `zt_vault`) oluşturun.

3. Bağlantıyı Yapılandırın

- config.php dosyasını açın ve kendi veritabanı kullanıcı adı ve şifrenizle güncelleyin.

4. Uygulamaları Çalıştırın

- PHP'nin dahili sunucusunu kullanarak projeyi başlatın veya bir web sunucusu (Apache/Nginx) üzerinde kurun. Tarayıcınızda L/ ve ZT/ dizinlerine gidin.

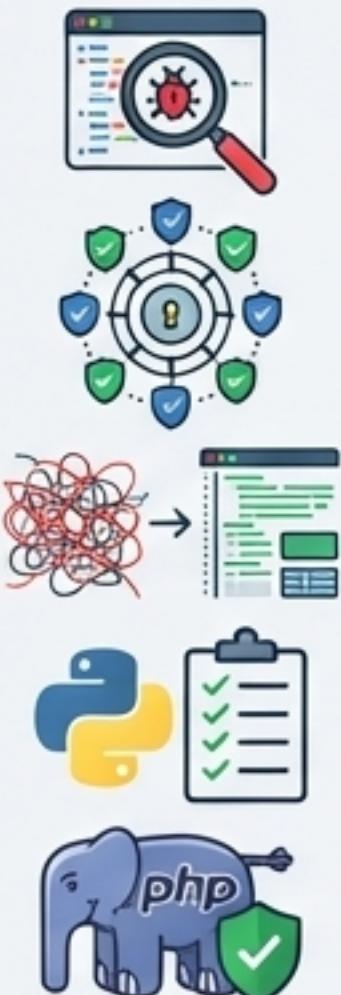
5. Testleri Ateşleyin! 🚀

- Python ortamınızı hazırladıktan sonra, `comprehensive_test.py` betığını çalıştırın ve düellonun sonuçlarını kendi gözlerinize görün.

```
bash\  
python comprehensive_test.py
```

Bu Düellodan Öğrenecekleriniz

ZTDALD Projesinin Eğitimsel Çıktıları



- Somut Zayıflıkları Anlama:** SQL Injection, XSS, CSRF gibi yaygın web açıklarının pratikte nasıl çalıştığını ve sömürüldüğünü ilk elden görün.
- Zero Trust Prensiplerini Kavrama:** Soyut bir konsept olan 'Sıfır Güven'i, kod seviyesinde uygulanmış somut örneklerle ('RBAC', 'Güvenlik Başlıkları', 'Parametreli Sorgular') dönüştürün.
- Güvenli Kodlama Alışkanlıklarını Geliştirme:** 'Kötü' ve 'İyi' kodları yan yana karşılaştırarak, güvenli yazılım geliştirmenin 'neden' ve 'nasıl'ını öğrenin.
- Güvenlik Test Metodolojileri:** Otomatik test betiklerinin ('comprehensive_test.py') bir mimarinin güvenlik durusunu doğrulamak için nasıl kullanıldığını deneyimleyin.
- PHP Güvenlik Pratikleri:** PHP'nin modern güvenlik özelliklerini (PDO, güvenli oturum yönetimi) gerçek dünya senaryolarında uygulayın.

Düello Devam Ediyor: Projenin Geleceği

ZTDALD, yaşayan bir projedir. Güvenlik dünyası sürekli gelişiyor ve bu projenin de gelişmesi gerekiyor. İşte üzerinde düşündüğümüz bazı gelecek adımları:



Fikirleriniz ve katkılarınız bu projeyi daha da değerli kılacaktır.
GitHub reposunu inceleyin, issue açın veya bir pull request gönderin!