Hafta 12 — Güvenlik Sertifikasyonları ve Sızma Testi Planlaması

Tarih	04.12.2026
Öğrenme çıktıları	ÖÇ.5, 6, 7
Süre	3 saat
Ön bilgi	Hafta 2'den CVSS ve zafiyet sınıflandırma; Hafta 9'dan gizleme kurallarının maliyet/kazanç çerçevesi; birim testi kavramı
Uygulamalar	`code/week-12` — 2 demo; `code` klasöründe bir kez derleyin, sonra `bin/linux` (Windows'ta `bin\windows`) altından çalıştırın

Bu haftanın çalışan demosu

`code/week-12/01-birim-test` — Birim test kosucusu: guvenlik fonksiyonlarini test kartlariyla dogrular (S16 sonuclari). · `code/week-12/02-saldiri-potansiyeli` — Saldırı potansiyeli hesaplayıcı: beş faktör → dirençlik düzeyi (düşük = ciddi bulgu).

Çalıştırma: `code` klasöründe bir kez `./build.sh` (Windows'ta `.\build.ps1`), sonra demo klasöründeki `bin/linux` (Windows'ta `bin\windows`) altından. Adım adım komutlar aşağıdaki kutuda. Tümüyle sentetik ve güvenlidir; öğrenci bilgisayarına zarar vermez.

Demoyu kendiniz çalıştırın — adım adım (kopyala-yapıştır)

İlk derleme `code/README.md`'de anlatılır. `code` klasöründen:

```
# Windows (PowerShell)
.\build.ps1
cd week-12\01-birim-test
.\bin\windows\birim_test.exe
cd ..\02-saldiri-potansiyeli
.\bin\windows\saldiri_potansiyeli.exe
```

```
# WSL / Linux
./build.sh
cd week-12/01-birim-test && ./bin/linux/birim_test
cd ../02-saldiri-potansiyeli && ./bin/linux/saldiri_potansiyeli
```

Beklenen çıktı: Birim test koşucusu iki güvenlik fonksiyonunu **test kartlarıyla** sınar; her test için amaç/gözlenen/karar basar ve **çıkış kodu = başarısız test sayısıdır** (hepsi geçerse 0 — S16'nın "plan değil **sonuç**" mantığı). İkinci demo beş faktörden (zaman · uzmanlık · bilgi · fırsat · ekipman) **toplam puanı ve dereceyi** hesaplar.

Bu haftanın sonunda şunları yapabileceksiniz

1. Bir ürünün **bağımsız bir laboratuvarda** güvenlik değerlendirmesinden geçişini uçtan uca, adım adım anlatmak.
2. ETSI, EMV, PCI DSS ve ISO/IEC 27001'in **ne tür güvenlik testleri** istediğini ayırt etmek.
3. Kod incelemesi, statik ve dinamik analiz, fuzzing ve sızma testini bir **zafiyet değerlendirmesi** içinde doğru sırada kullanmak.
4. Bir **sızma testi planı** yazmak: kapsam, kurallar (angajman kuralları), yöntem (OWASP WSTG/MASTG, PTES, NIST SP 800-115) ve raporlama.
5. Bir bulguyu **saldırı potansiyeli** ve CVSS ile derecelendirmek; bulgu–öneri–aksiyon döngüsünü ve **güvenlik etki analizi / delta değerlendirmeyi** açıklamak.

Ders akışı (öğretim üyesi için zaman planı)

Zaman	Bölüm	Ne yapılıyor
0:00–0:20	1	Neden bağımsız değerlendirme? Standartlar ve sertifikasyon manzarası
0:20–0:50	2	Değerlendirme süreci: hedef tanımından delta değerlendirmeye 13 adım
0:50–1:00	Ara	
1:00–1:30	3–4	Zafiyet değerlendirmesinin araçları; standartların istediği testler
1:30–1:50	5	Saldırı potansiyeli ve bulgu derecelendirme
1:50–2:00	Ara	
2:00–2:35	6	Sızma testi planı: kapsam, kurallar, yöntem, test şablonu — **takım etkinliği**
2:35–2:50	7–8	Raporlama; eleştirel okuma alıştırması
2:50–3:00	9+	Proje adımı (S16 test planı), kendini sınama

Etik ve hukuki çerçeve

Sızma testi, **yazılı izin** ve kapsamı açıkça tanımlanmış bir sözleşme olmadan yapılamaz. İzinsiz test, amacı ne olursa olsun suçtur. Bu haftanın bütün etkinlikleri kendi dönem projeleriniz ve ders demoları üzerinde, kağıt üstünde **planlama** olarak yapılır; başka bir takımın projesini test etmek ancak o takımın ve öğretim üyesinin onayıyla, ders ortamında olur.

0. Temel kavramlar (sıfırdan)

Bu bölüm **hiçbir ön bilgi varsaymaz**. Haftanın geri kalanında kullanacağımız terimleri sıfırdan tanımlıyoruz. Bir terimi bilmiyorsanız önce burayı okuyun; sonraki bölümler bunların üzerine kurulur.

Neden bu bölüm?

Bu hafta "değerlendirme", "sertifikasyon", "sızma testi" gibi terimler geçecek.

Önce hepsini **tek tek** tanımlayalım ki konu havada kalmasın.

Güvenlik değerlendirmesi nedir?

- **Güvenlik değerlendirmesi:** bir ürünün güvenlik iddialarının **bağımsız** biri tarafından **sınanması**.
- "Ben güvenliyim" demek yetmez; **kanıt** ve **test** gerekir.

Sertifikasyon nedir?

- **Sertifikasyon:** bir ürünün/kurumun belirli bir **standarda** uyduğunun, yetkili bir tarafça **belgelenmesi**.
- Değerlendirme başarılıysa bir **sertifika** verilir.

Laboratuvar (değerlendirici) kim?

- **Değerlendirme laboratuvarı:** ürünü sınayan bağımsız, akredite kuruluş.
- Ürünü geliştiren **değildir** (tarafsızlık için).
- Bütün kaynak koda ve belgeye erişir.

Standart nedir?

- **Standart:** neyin nasıl yapılacağını belirleyen ortak kurallar bütünü.
- Örnek: ISO/IEC 27001, Ortak Kriterler, FIPS 140-3, PCI, OWASP MASVS.
- Her biri farklı şeyi ölçer (birazdan).

Zafiyet (vulnerability) nedir?

- **Zafiyet:** bir sistemde kötüye kullanılabilecek **zayıf nokta** (ör. sınır denetimsiz tampon).
- Zafiyet + kötüye kullanım = güvenlik olayı.

Bulgu (finding) nedir?

- **Bulgu:** değerlendirmede tespit edilen bir sorun ya da iyileştirme noktası.
- Her bulgu için: kanıt, ciddiyet, **öneri**.

Beyaz kutu vs kara kutu test

- **Beyaz kutu test:** test eden **kaynak koda + belgeye** sahip.
- **Kara kutu test:** test eden yalnız dışarıdan (kullanıcı gibi) erişir.
- Değerlendirici genelde **beyaz kutu** çalışır (her şeyi görür).

SAST nedir?

- **SAST (Static Application Security Testing):** kaynağı **çalıştırmadan** analiz eden araç.
- Tehlikeli kalıpları, olası bellek hatalarını bulur.
- Hızlı ve geniş; ama **yanlış pozitif** üretir.

(4. haftada "statik analiz" olarak gördük.)

DAST nedir?

- **DAST (Dynamic Application Security Testing):** programı **çalıştırırken** sınar.
- Bellek erişim hataları, tanımsız davranışları yakalar (ör. sanitizer'lar).

(4. haftada ASan/UBSan.)

Fuzzing nedir?

- **Fuzzing:** programa **beklenmeyen/rastgele girdiler** verip çökme/bozulma aramak.
- İnsanın aklına gelmeyen girdileri bulur.

(4. haftada libFuzzer/AFL kavramı.)

Sızma testi (pentest) nedir?

- **Sızma testi (penetration test):** bir saldırganın bakışıyla, **izinli** ve **planlı** olarak sistemi aşmayı denemek.
- Yukarıdaki yöntemleri **birleştirir**; en son ve en pahalı adımdır.

TOE nedir?

- **TOE (Target of Evaluation):** değerlendirilen **tam olarak ne**?
- Ortak Kriterler terimidir.
- Benzersiz tanımlanır: sürüm + ikili + kaynak + özet değeri.

CVSS nedir?

- **CVSS (Common Vulnerability Scoring System):** bir zafiyetin **etkisini** standart bir puanla (0–10) ifade eder.
- Yüksek puan = daha ciddi etki.
- Önceliklendirmede kullanılır.

Saldırı potansiyeli nedir?

- **Saldırı potansiyeli:** bir saldırıyı gerçekleştirmenin **ne kadar zor** olduğu.
- Süre, uzmanlık, ekipman gibi faktörlerle puanlanır.
- Düşük potansiyel (kolay saldırı) = ciddi bulgu.

Saldırı potansiyeli — beş faktör

Puan beş faktörün toplamından çıkar; her faktör "saldırganın işini ne kadar zorlaştırıyor?" sorusunu ölçer:

Faktör	Neyi ölçer?
Geçen zaman	Saldırı ne kadar sürüyor (saatler mi, aylar mı)?
Uzmanlık	Sıradan bir kullanıcı mı, alan uzmanı mı gerekiyor?
Hedef bilgisi	Kamuya açık bilgi mi, iç tasarım belgesi mi gerekiyor?
Fırsat (erişim)	Cihaza ne kadar süre, ne kadar yakından erişmek gerekiyor?
Ekipman	Sıradan bilgisayar mı, özel laboratuvar donanımı mı?

Toplam puan bir **direnç düzeyine** çevrilir. Dikkat: **düşük** puan, saldırının **kolay** olduğu anlamına gelir; yani düşük puan ciddi bir bulgudur. Beş faktörün tam tablosunu ve iki işlenmiş hesabı 5. bölümde göreceğiz.

Etki analizi ve delta değerlendirme

- **Güvenlik etki analizi:** bir değişikliğin güvenlik etkisini **belgeleyen** rapor.
- **Delta değerlendirme:** yalnız **değişen kısmın** yeniden değerlendirilmesi.

Bunları 5. bölümde ayrıntılı göreceğiz.

Risk nedir? (zafiyet, bulgu, risk aynı şey değildir)

Yukarıdaki tanımlarda geçen üç kelime — zafiyet, bulgu, risk — günlük dilde birbirinin yerine kullanılır ama değerlendirmede **üçü ayrı şeydir**:

- **Risk:** bir zafiyetin **sömürülme olasılığı** ile sömürüldüğünde oluşacak **etkinin** birlikte değerlendirilmesi — kabaca "ne kadar olası × ne kadar kötü".
- Her zafiyet aynı riski taşımaz: sömürülmesi neredeyse imkânsızsa (örn. kilitli bir odadaki, ağa hiç bağlı olmayan bir sunucu) risk düşüktür; aynı zafiyet internete açık bir sunucuda ise risk yüksektir — **zafiyet aynı, risk farklı.**
- **Kalan risk (residual risk):** bütün önlemlerden sonra hâlâ kapatılamayan risktir; iyi bir rapor bunu **açıkça** yazar, saklamaz (bölüm 6 ve 12'de örneğini göreceğiz).

*(Kısaca: **zafiyet** teknik bir zayıflıktır, **bulgu** bu zayıflığın değerlendirmede kanıtla belgelenmesidir, **risk** ise "bu zayıflık gerçekten ne kadar tehlikeli" sorusuna verilen bağlamsal yanıttır.)*

Terimleri birbirinden ayırmak: uçtan uca küçük bir örnek

Yukarıdaki tanımlar tek tek anlamlı ama aralarındaki fark, tek bir örnek üzerinde görülmeden soyut kalır. Aşağıdaki sentetik fonksiyonu düşünün:

oturum_ac.c (sentetik, hatalı)

```c
int oturum_ac(const char *kullanici, const char *sifre)
{
    char sorgu[256];
    /* HATA: kullanici ve sifre dogrudan sorguya ekleniyor */
    sprintf(sorgu, "SELECT * FROM kullanicilar WHERE ad='%s' AND sifre='%s'",
            kullanici, sifre);
    return veritabani_calistir(sorgu);   /* 1 = giris basarili */
}
```

Bu fonksiyonda `kullanici` alanına `' OR '1'='1` yazan biri, şifre bilmeden giriş yapabilir (klasik **SQL enjeksiyonu**, CWE-89). Şimdi her yöntemin bu tek hatayı nasıl (ve ne kadar) yakaladığını satır satır izleyelim — bu, bölüm 3'teki "hangi yöntem ne bulur" sorusunun ilk somut örneğidir:

Yöntem	Bu hatayı bulur mu?	Neden
Kod incelemesi (el ile)	**Evet, güvenilir biçimde**	İnceleyen kişi `sprintf` ile kullanıcı girdisinin doğrudan SQL'e eklendiğini **bağlamıyla** görür; "bu girdi güvenilmez, parametreli sorgu gerekir" sonucuna varır.
SAST	Genelde evet	Çoğu SAST aracı "kullanıcı girdisi → sorgu dizesi" veri akışını (taint analizi) tanır ve işaretler; ama aracın kural kümesine bağlıdır, bazı SAST'lar kaçırabilir.
DAST	Yalnız SQLi'ye özel test yapılırsa	Genel bir DAST taraması rastgele girdilerle çalışırsa hatayı **kaçırabilir**; SQLi'ye özel bir tarayıcı (`' OR '1'='1` gibi yükler dener) yakalar.
Fuzzing	Düşük olasılıkla	Rastgele bayt dizileri, geçerli SQL sözdizimi (`' OR '1'='1`) üretme olasılığı düşüktür; fuzzing bu tür **mantık** hatalarında zayıftır, bellek hatalarında güçlüdür.
Sızma testi	**Evet, kanıtla**	Saldırgan bakışıyla doğrudan `' OR '1'='1` denenir, gerçek bir bypass **gözlenir** ve kanıtlanır (ekran görüntüsü/günlük).

Terimleri şimdi bu örnek üzerinden yerleştirelim:

- **Zafiyet:** `sprintf` ile parametresiz SQL sorgusu kurmak — teknik zayıf nokta.
- **Bulgu:** değerlendiricinin bunu yazılı olarak belgelemesi: "T-14: `oturum_ac` fonksiyonu SQL enjeksiyonuna açık, kanıt: ekran görüntüsü, CWE-89."
- **Risk:** bu uygulama internete açıksa ve arkasında gerçek kullanıcı verisi varsa risk **yüksektir** (olasılık yüksek × etki yüksek); yalnız kapalı bir test ağında çalışan bir prototipse aynı zafiyetin riski daha düşük değerlendirilebilir.
- **Öneri:** parametreli sorgu (prepared statement) kullanımı.

Sık yapılan hata: 'SAST/DAST temiz çıktı verdi, öyleyse güvenliyiz' demek

Hiçbir tek araç bütün hata sınıflarını yakalamaz: SAST mantık hatalarını bazen kaçırır, DAST test edilmeyen bir yolu hiç görmez, fuzzing SQL gibi yapılandırılmış girdilerde zayıftır. **Sonuç:** tek bir aracın "temiz" raporu, "hiç zafiyet yok" anlamına gelmez. **Kural:** değerlendirme bu yöntemleri **art arda ve tamamlayıcı** kullanır (bölüm 3); hiçbiri tek başına yeterli kanıt sayılmaz.

Standart mı, sertifika mı? Sık karıştırılan bir çift

- **Standart**, bir kuraldır: "anahtar türetme böyle yapılmalı" der. Standardın kendisi hiçbir ürünü **onaylamaz**; yalnızca ölçütü tanımlar.
- **Sertifika**, bir ürünün ya da kurumun o kuralı **karşıladığının belgesidir**. Standart olmadan sertifika olmaz; ama bir standardı okumak (ya da ona "uygun tasarlamak") kendi başına sertifika **almak** anlamına gelmez — bağımsız bir doğrulama (değerlendirme) gerekir.

> ⚠ **Sık yapılan hata: 'standarda uygun tasarladık' ile 'sertifikalıyız' cümlelerini eşitlemek**
>
> Bir ekip, bir standardın gereksinimlerini okuyup ona göre tasarım yapabilir — bu **iyi bir başlangıçtır** ama sertifika değildir. Sertifika, bağımsız bir laboratuvarın bu iddiayı **test ederek doğrulamasından** sonra verilir. **Kural:** "standarda uygun tasarlandı" ile "standarda göre sertifikalandı" cümlelerini net ayırın.

Kerckhoffs ilkesi: değerlendiricinin varsayımı nereden geliyor?

İlerideki bölümlerde "değerlendirici, saldırganın sistemi tam bildiğini varsayar" cümlesini sık göreceğiz. Bu varsayımın bir adı var:

- **Kerckhoffs ilkesi:** bir sistemin güvenliği, **tasarımın gizli kalmasına değil**, yalnızca **anahtarın** gizli kalmasına dayanmalıdır. Yani "algoritmamızı kimse bilmiyor, o yüzden güvenliyiz" savunması **geçersizdir** — saldırganın kaynak kodu, tasarımı, hatta kılavuzu bildiğini varsayarak tasarlamak gerekir.
- Değerlendiricinin **beyaz kutu** çalışması (aşağıda) tam olarak bu ilkenin doğal sonucudur: madem gerçek bir saldırgan er ya da geç tasarımı öğrenebilir, değerlendirici de baştan her şeyi görsün.

Beyaz kutu neden tercih edilir? Kısa bir karşılaştırma

	Beyaz kutu	Kara kutu
Erişim	Kaynak kod + belge + iç tasarım	Yalnız dışarıdan, kullanıcı gibi
Bulduğu hata türü	Derin mantık hataları, tasarım zaafları	Yalnız dışarıdan gözlenebilir davranışlar
Süre/maliyet	Daha hızlı (doğrudan bakar)	Daha yavaş (önce keşfetmesi gerekir)
Gerçek saldırganı temsil eder mi?	Kısmen — çoğu saldırgan kaynağı görmez	Evet — ilk günkü saldırgan deneyimine yakın

> ℹ **İkisi neden birlikte kullanılır?**
>
> Değerlendirme çoğunlukla **beyaz kutu** çalışır (Kerckhoffs gereği, "saldırgan öğrenebilir" varsayımıyla), ama sızma testi adımında (bölüm 7) genellikle bir **kara kutu bileşeni** de eklenir: "gerçek bir saldırgan, hiçbir iç bilgi olmadan, ilk elde ne kadar ilerleyebilir?" sorusu da ayrıca yanıtlanır. İkisi birlikte, hem "en kötü durumu" (beyaz kutu) hem "gerçekçi ilk günü" (kara kutu) gösterir.

Bütün terimleri tek bir cümlede birleştirelim

Şimdiye kadar tek tek tanımladığımız terimleri, gerçek bir sürecin tek bir akışında görelim — bu, haftanın geri kalanının **tek cümlelik özetidir**:

> Bir **standart** hangi **gereksinimlerin** karşılanması gerektiğini söyler; bir **değerlendirme laboratuvarı**, **beyaz kutu** erişimle (Kerckhoffs ilkesi gereği, saldırganın her şeyi bildiğini varsayarak) ürünü inceler; kod incelemesi, **SAST**, **DAST**, **fuzzing** ve **sızma testi** ile **zafiyetleri** bulur; her zafiyeti kanıtla bir **bulguya** dönüştürür; her bulguyu **saldırı potansiyeli** ve **CVSS** ile derecelendirir; geliştirici düzeltme yapınca laboratuvar yalnız değişeni **delta değerlendirmeyle** yeniden inceler; sonunda bir **sertifikasyon** otoritesi, belirli bir **TOE** kimliği için sertifikayı verir.

Bu tek paragrafı okuyup her terimi tanıyabiliyorsanız, bölüm 1'den 9'a kadar göreceğimiz her şeyin **iskeletini** zaten kurmuşsunuz demektir; geri kalan bölümler bu iskeletin **etini** dolduracak.

Şimdi hazırız

Bildiğimiz terimler:

değerlendirme · sertifikasyon · laboratuvar · standart · zafiyet · bulgu · risk · beyaz/kara kutu · Kerckhoffs ilkesi · SAST · DAST · fuzzing · sızma testi · TOE · CVSS · saldırı potansiyeli · etki analizi/delta

Şimdi: **neden bağımsız değerlendirme?**

1. Neden bağımsız değerlendirme?

Bir yazılımı yazan ekip, onun en iyi test edicisi değildir: tasarım varsayımlarını paylaşır, "bu nasılsa olmaz" dediği yolları hiç denemez. Bu yüzden ödeme, kimlik, sağlık ve kamu gibi alanlarda ürünler, **bağımsız bir laboratuvarın** değerlendirmesinden geçer. Değerlendirmenin sonunda bir sertifika (ya da onay) verilir; sertifika "bu ürün kırılamaz" demez, "bu ürün şu standardın şu gereksinimlerini, şu saldırgan modeline karşı, şu tarihte karşılıyordu" der.

Kısa tarihçe: güvenlik değerlendirmesi ve sertifikasyon

- **1985** — ABD **TCSEC** ("Orange Book"): ilk resmi güvenlik değerlendirme ölçütleri.
- **1991–1993** — Avrupa **ITSEC** ve Kanada **CTCPEC**.
- **1999** — bunlar **Ortak Kriterler (ISO/IEC 15408)** altında birleşir; **EAL** güvence ölçeği buradan gelir (13. hafta).
- **2001** — **OWASP** kurulur (uygulama güvenliği testi kültürü); sonra **PTES** ve **NIST SP 800-115** sızma testi metodolojilerini standartlaştırır.
- **2005 → 2023** — **CVSS** zafiyet ciddiyet puanı (v2 → v3.1 → v4.0).
- **2010'lar** — mobil için **OWASP MASVS/MASTG**, tüketici IoT için **ETSI EN 303 645**.

Ortak fikir tek cümlede: **üretici kendi ürününü onaylayamaz** — bağımsız, kanıta dayalı değerlendirme gerekir.

Bu dersin projesini baştan beri "bir sertifikasyondan geçecekmiş gibi" tasarladık. Bu hafta o sürecin kendisini, bir değerlendiricinin gözünden, adım adım işliyoruz.

Standartlar ve sertifikasyon manzarası

Standart / şema	Alan	Ne değerlendirilir?	Bu dersteki yeri
Ortak Kriterler (ISO/IEC 15408) + **CEM** (ISO/IEC 18045)	Genel BT ürünleri	Güvenlik hedefi (ST), koruma profili (PP), EAL düzeyleri; zafiyet analizi ve saldırı potansiyeli	13. hafta
FIPS 140-3	Kriptografik modüller	Modülün kripto algoritmaları, anahtar yönetimi, fiziksel güvenlik, öz testler	13. hafta
EMVCo belgeleri	Ödeme kartları, terminaller, mobil ödeme	Çip ve yazılım tabanlı ödeme çözümlerinin güvenlik değerlendirmesi	3, 6, 11, 13. haftalar
PCI standartları (DSS, PIN, MPoC, SSF)	Kart verisi işleyen kurum ve yazılımlar	Kart verisinin korunması; yazılımın güvenli geliştirilmesi; test ve tarama gereksinimleri	Bu hafta
ISO/IEC 27001	Kurumun bilgi güvenliği yönetim sistemi	Süreçler, risk yönetimi, kontroller (Ek A); ürün değil **kurum** sertifikalanır	Bu hafta
ETSI (EN 303 645, TS 103 732 vb.)	Tüketici IoT, telekom, mobil	Cihaz güvenliği temel gereksinimleri, test belirtimleri	13. hafta
OWASP ASVS / MASVS	Web ve mobil uygulamalar	Uygulama güvenliği doğrulama düzeyleri (sertifika değil, test çerçevesi)	5, 6. haftalar

Ürün mü, süreç mi?

ISO/IEC 27001 bir **kurumu** (onun bilgi güvenliği yönetim sistemini) sertifikalar; Ortak Kriterler, FIPS 140-3 ve EMVCo değerlendirmeleri ise bir **ürünü**. PCI DSS, kart verisi işleyen bir ortamın uyumunu denetler. Bir proje kılavuzunda bu farkı karıştırmamak gerekir: "ürünümüz ISO 27001 sertifikalı" cümlesi yanlıştır; "ürünümüz ISO 27001 sertifikalı bir kurumda geliştirilmektedir" doğrudur.

Sertifikasyon ne söyler, ne söylemez?

Bir sertifika okuyucusunun en sık yaptığı hata, "sertifikalı" sözcüğünü "kırılamaz" ile eş tutmaktır. Sertifika aslında dört şeyi **birlikte** söyler; dördünden biri eksikse cümle anlamsızlaşır:

1. **Hangi ürün** — TOE'nin tam kimliği: sürüm + ikili dosya + kaynak kod + özet değeri (bölüm 2'de göreceğiz).
2. **Hangi standarda göre** — ör. Ortak Kriterler, EMVCo, PCI.
3. **Hangi saldırgan modeline karşı** — ör. "fiziksel erişimi olmayan, ağ üzerinden saldıran orta düzey saldırgan".
4. **Hangi tarihte** — yazılım güncellenince sertifika o **eski** sürüm için geçerliliğini korur; yeni sürüm ya delta değerlendirmeden geçmeli ya da sertifikasız sayılmalıdır.

Sık yapılan hata: 'X sertifikalı' demek, ürünü/sürümü/tarihi belirtmemek

Bir proje sunumunda "kütüphanemiz Ortak Kriterler sertifikalı" demek eksik ve yanıltıcıdır: **hangi sürüm**, **hangi güvence düzeyi**, **hangi saldırgan modeline karşı** belirtilmeden bu cümle bir pazarlama iddiasından farksızdır. **Kural:** sertifika iddiası her zaman sürüm numarası + standart adı + (varsa) güvence düzeyiyle birlikte yazılır; 1. haftadaki sürüm kimliği disiplini tam olarak bu yüzden gerekir.

Neden geliştirici kendi ürününü onaylayamaz?

Bu bir yetenek sorunu değil, bir **bakış açısı** sorunudur. Bir ürünü yazan ekip üç önyargıyı aynı anda taşır:

- Tasarım **varsayımlarını paylaşır** — "kullanıcı bu alana asla 300 karakter girmez" diyen geliştirici, testini de bu varsayıma göre yazar; sınır dışı girdiyi hiç denemez.
- "Buraya kimse dokunmaz" dediği yollara **hiç bakmaz** — çünkü kendi zihin modelinde o yol zaten kapalıdır.
- Kendi kodunu **eleştirmekte isteksizdir** — zaman baskısı altında "büyük ihtimalle güvenlidir" cümlesi bir kanıt değildir ama insan doğası buna eğilimlidir.

Bağımsız bir laboratuvar bu üç önyargıdan **hiçbirini paylaşmaz**: geliştiricinin belgesini okur ama sınırlarını sorgular, "kimse dokunmaz" dediği yolu ilk kontrol ettiği yer yapar ve ürünün kendi çıkarına bakmadığı için "muhtemelen güvenli" cümlesini kabul etmez — **kanıt** ister.

Kural: iddia değil, kanıt

Bu dersin projesinde de aynı disiplin geçerlidir: "güvenlidir" cümlesi tek başına hiçbir şey kanıtlamaz. Her güvenlik iddiası bir **test sonucuyla** (S16), bir **kanıtla** (ekran çıktısı, günlük, test kartı) desteklenir. Vize/final değerlendirmesinde öğretim üyesi tam olarak bağımsız laboratuvarın rolündedir.

Aynı ürün, farklı şemalarda farklı testler: kısa bir örnek

Aynı mobil ödeme uygulamasının farklı hedef pazarlara göre **farklı** değerlendirmelerden geçmesi gerekebilir:

- Uygulama bir **bankaya** satılacaksa: bankanın bilgi güvenliği yönetim sistemi muhtemelen **ISO/IEC 27001** kapsamındadır — ama bu, uygulamanın kendisini değil, bankanın **süreçlerini** sertifikalar.
- Uygulama **kart verisi işliyorsa**: **PCI DSS** (kurum) ve/veya **PCI MPoC/SSF** (yazılım) kapsamına girer — yazılımın kart verisini nasıl işlediği, sakladığı, ilettiği test edilir.
- Uygulama bir **ödeme şemasının** (ör. bir kart ağının) markasını taşıyacaksa: **EMVCo** değerlendirmesinden geçmesi gerekir — çip/yazılım tabanlı ödeme akışlarının güvenliği, laboratuvar sızma testiyle sınanır.
- Uygulama genel bir **mobil güvenlik** iddiası taşıyorsa: **OWASP MASVS** düzeyine göre test edilebilir — bu bir sertifika değil ama tanınmış bir **test çerçevesidir**.

Aynı kod tabanı, dört farklı mercekten dört farklı soruyla sınanır. Projenizde "hangi standarda yakınız" sorusunun cevabı, S17'deki gereksinim şablonunun **hangi sütunlarla** doldurulacağını belirler.

Sertifikasyonun maliyeti: neden her ürün sertifikalanmaz?

Bağımsız değerlendirme **ücretsiz değildir** — laboratuvar zamanı, uzman mühendis saatleri, bazen özel ekipman gerektirir; süreç haftalar-aylar sürebilir. Bu yüzden her yazılım her standarda göre sertifikalanmaz; bir ekip şu soruyu sorar: "bu sertifikanın **maliyeti**, sağladığı **güven**e değer mi?"

- Bir bankaya satılacak bir ödeme kütüphanesi için cevap genelde **evet**tir: sertifika olmadan banka o kütüphaneyi hiç almayabilir; sertifikanın maliyeti, kaybedilecek satıştan çok daha küçüktür.
- Bir iç şirket aracı (yalnız kendi çalışanlarının kullandığı, dışarı satılmayan bir uygulama) için tam kapsamlı bir Ortak Kriterler değerlendirmesi genelde **orantısızdır**; bunun yerine daha hafif bir iç güvenlik incelemesi (kod incelemesi + SAST + temel sızma testi) tercih edilir.

> **Ödünleşim burada da geçerli**
>
> 1. haftadaki ödünleşim (trade-off) disiplini sertifikasyon kararında da işler: "tam kapsamlı sertifika" her zaman doğru cevap değildir. Doğru soru "hedef pazarımız, varlığımızın değeri ve saldırgan modelimiz göz önüne alındığında, hangi **derinlikte** bir doğrulama yeterli?" sorusudur — bu, bölüm 4'teki "test derinliği" tablosunun kararı nasıl etkilediğini gösterir.

Sertifikasyonun tarihsel itici gücü: neden ortaya çıktı?

Bölümün başındaki kısa tarihçeyi bir de "neden" sorusuyla okuyalım — her adım aslında gerçek bir **güven krizine** cevaptır:

- TCSEC (1985) öncesinde, ABD hükümeti güvenlik iddialarını değerlendirecek **ortak bir ölçüt** bulamıyordu; her tedarikçi kendi "güvenlidir" tanımını kullanıyordu — karşılaştırma imkânsızdı.
- Ortak Kriterler (1999), ABD/Avrupa/Kanada'nın **ayrı ayrı** ölçütlerinin (TCSEC/ITSEC/CTCPEC) birbiriyle **karşılaştırılamaz** olmasının doğurduğu bir sorunu çözmek için birleştirildi — uluslararası ticarette aynı ürünün her ülkede yeniden değerlendirilmesi gerekmesin diye.
- CVE/CWE/CVSS (2. haftada gördüğümüz gibi) 1999→2006 arasında, güvenlik açıklarının **isimsiz** ve **karşılaştırılamaz** biçimde raporlanmasının yarattığı kargaşaya cevap olarak doğdu.

Kural: her standart bir gerçek soruna cevaptır

Bir standardı ezberlemek yerine "bu hangi soruna cevap veriyor?" diye sormak, hangi standardın projenize uygun olduğunu seçerken daha sağlam bir pusuladır.

Sık sorulan soru: bir sertifika süresiz midir?

Zayıf bir öğrencinin çoğu zaman atladığı ama önemli bir nokta: sertifika, "bu ürün sonsuza kadar güvenlidir" demez. Üç ayrı neden yüzünden bir sertifika zamanla geçerliliğini yitirebilir ya da yeniden gözden geçirilmesi gerekebilir:

1. **Ürün değişir** — yeni bir sürüm çıkar; bölüm 2'deki delta değerlendirme devreye girer, sertifika **yeni TOE kimliğine** taşınmalıdır.
2. **Saldırı teknikleri gelişir** — bugün "çoklu uzman, aylar" gerektiren bir saldırı, birkaç yıl sonra ucuz araçlarla "sıradan kullanıcı, bir gün"e düşebilir (bölüm 5'teki puanlama zamanla **değişebilir**).
3. **Standart güncellenir** — bir standardın yeni sürümü, eskiden yeterli sayılan bir kontrolü artık yetersiz bulabilir.

Sık yapılan hata: 'bir kere sertifikalandık, tamamdır' demek

Bir ürünü yıllar önce alınmış bir sertifikayla pazarlamak, o sertifikanın **hangi tarihte, hangi tehdit modeline karşı** verildiğini görmezden gelmek olur (bölüm 1'in başındaki dört bilgi). **Kural:** uzun ömürlü ürünler için sertifikalar genellikle periyodik olarak **yeniden gözden geçirilir** ya da bir süre sonunda yenilenir; "sertifikalıyız" cümlesi her zaman bir **tarihle** okunmalıdır.

---

2. Değerlendirme süreci: 13 adım

Aşağıdaki akış, bir yazılım bileşeninin (ör. bir mobil ödeme kütüphanesi) üçüncü taraf bir laboratuvarda değerlendirilmesinin genelleştirilmiş halidir. Ürün, kurum ve şema adları çıkarılmıştır; adımlar birçok şemada benzerdir.

TOE kimliği: neden "sürüm 3.2.0" yetmez?

Az sonra göreceğimiz 13 adımın ilki, bir sürüm numarasından **çok daha kesin** bir tanım ister. Nedenini bir örnekle görelim: bir ekip "sürüm 3.2.0" der, ama aynı sürüm numarasıyla üç farklı ikili dosya var olabilir — biri hata ayıklama sembolleriyle derlenmiş (debug build), biri optimize edilmiş sürüm derlemesi (release build), biri de bir geliştiricinin yerel makinesinde farklı derleyici bayraklarıyla ürettiği bir kopya. Bunların **davranışı** (hata ayıklama çıktısı, zamanlama, hatta bazı güvenlik kontrollerinin varlığı) farklı olabilir. Bu yüzden TOE kimliği dört parçadan oluşur (yukarıdaki görsel):

1. **Sürüm etiketi** — insan okur için (ör. "3.2.0").
2. **İkili dosya** — hangi derleme (release/debug, hangi mimari: ARM64/x86_64).
3. **Kaynak kod** — hangi commit/etiket (git tag/commit özeti).
4. **Özet değeri (hash)** — ikili dosyanın SHA-256'sı; **tek bir baytın** bile değişip değişmediğini kanıtlar.

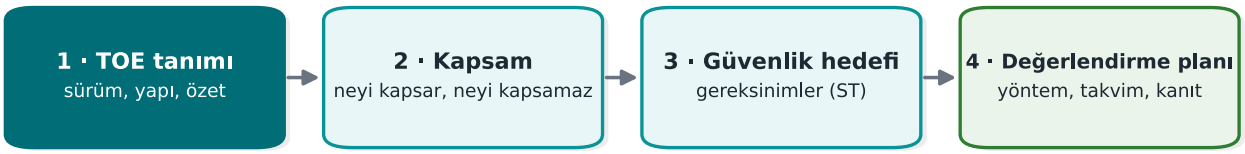
Sık yapılan hata: 'aynı sürüm numarası, aynı ürün' varsaymak

Bir geliştirici "değişen bir şey yok, hâlâ 3.2.0" diyebilir ama derleme ortamı (derleyici sürümü, optimizasyon bayrağı) değişmişse **ikili dosya farklıdır** ve önceki sertifika artık o ikiliyi kapsamaz. **Kural:** TOE kimliği her zaman özet değeriyle (hash) doğrulanır, sürüm numarasına güvenilmez — 1. haftadaki "sürüm kimliği" pratiği tam olarak bunun içindir.

Bu görseldeki üç aşamayı adlandıralım: **Hazırlık** (adım 1–4: TOE, belgeler, şablon, atölye), **Değerlendirme** (adım 5–9: kod inceleme → zafiyet analizi → sızma testi → işlevsel uygunluk → bulgular), **Süreklilik** (adım 10–13: etki analizi → delta değerlendirme → ödünleşim/kalan risk → değişiklik yönetimi). Sertifika, Değerlendirme aşamasının sonunda verilir; ama Süreklilik aşaması ürünün ömrü boyunca **hiç bitmez**.

Hazırlık aşaması: adım 1–4

yanlış tanımlanmış TOE, tüm değerlendirmeyi geçersiz kılar



TOE benzersiz tanımlanmazsa 'hangi sürüm sertifikalı?' sorusu yanıtsız kalır.

Adım	Ne yapılır?	Çıktı	Projenizde
1. Değerlendirme hedefi	Değerlendirilecek şey **benzersiz** tanımlanır: sürüm numarası yetmez; ikili dosya + kaynak kod + özet değeri ya da sürüm etiketi. Kapsam dışı bileşenler yazılır. Varsayım: **platform güvenilmez**	TOE tanımı	S0, S1 (1. hafta sürüm kimliği)
2. Belgelerin teslimi	Kaynak kod, API belgesi, **güvenlik kılavuzu**, hata ayıklama ve sürüm derlemeleri güvenli bir kanalla teslim edilir	Teslim listesi	Deponuz + kılavuz
3. Gereksinim şablonu	Standardın numaralı gereksinimleri; her biri için gereksinim metni, **test kapsamı**, geliştiricinin **uyum gerekçesi ve belge referansı**; durum: karşılandı / devredildi / karşılanmadı	Doldurulmuş şablon	S17 uyum matrisi (13. hafta)
4. Atölye	Hangi önlemin hangi gereksinimi karşıladığı, eksiklerin nasıl kapatılacağı planlanır	Belge güncelleme planı	—
5. Kaynak kod incelemesi	Bütün kaynak, laboratuvarda **beyaz kutu** bağlamında incelenir: tehlikeli kalıplar, yanlış kripto, sızıntı noktaları	Kod inceleme bulguları	4. hafta CERT, statik analiz
6. Zafiyet analizi	Varlıklar ve anahtar hiyerarşisi çıkarılır; her varlık için "nerede, hangi halde açığa çıkıyor?" sorulur	Bulgu listesi + **sızma testi planı**	S4, S5, S16
7. Sızma testi	Plandaki her test, sekiz başlıklı şablonla yürütülür ve **saldırı potansiyeliyle** derecelendirilir	Test sonuç tabloları	S16
8. İşlevsel uygunluk	Ürünün standart işlevleri (ör. ödeme akışları) şemanın test paketiyle sınanır	Yürütme raporu	Birim testleri
9. Bulgular ve düzeltmeler	Her bulgu için laboratuvar öneri, geliştirici aksiyon üretir; bazı bulgular "güvenlik sorunu değil, iyi uygulama önerisi" olarak kapanır	Bulgu–aksiyon tablosu	7. hafta bulgu listesi
10. Güvenlik etki analizi	Düzeltmeler yeni sürüm doğurunca: değişiklik dizini (yeni özellik / iyileştirme / hata düzeltme), etkilenen dosyalar, güvenlik etkisi	Etki analizi belgesi	S13, 1. hafta değişiklik yönetimi
11. Delta değerlendirme	Yalnız değişen kısım yeniden değerlendirilir; laboratuvar işaretlenmiş belgeler, etki analizi, dosya listesi ve yeni TOE kimliğini ister	Delta raporu	—
12. Ödünleşim ve kalan risk	Her korumanın maliyeti ölçülür, kararlar gerekçelendirilir, kalan risk açıkça yazılır	Ödünleşim kaydı	1. hafta
13. Değişiklik yönetimi	Temel çizgi → talep → sınıflandırma → onay → geliştirme ve test → yayın → doğrulama	Süreç kayıtları	S13

Sahada nasıl uygulanır?

Bu dersin her haftasında gördüğümüz "Sahada nasıl uygulanır?" notlarının kaynağı olan kılavuz, tam olarak 2. adımda teslim edilen **güvenlik kılavuzudur**: birincil okuru değerlendirme laboratuvarıdır. Kılavuzun her bölümü bir gereksinim bloğuyla açılır ("gereksinim — durum"), ardından ürünün o gereksinimi nasıl karşıladığı anlatılır. Dönem projenizde yazdığınız kılavuz, bu belgenin küçültülmüş halidir.

Adım 2'deki "güvenli kanal" ne demektir?

Tablodaki "güvenli bir kanalla teslim edilir" ifadesi boşuna değildir: kaynak kod ve güvenlik kılavuzu, henüz sertifikalanmamış bir ürünün **en hassas** bilgileridir. Bunları e-posta ekiyle ya da genel bir dosya paylaşım bağlantısıyla göndermek, değerlendirme başlamadan önce ürünün gizliliğini zedeler.

- **Şifreli aktarım:** dosyalar AEAD (hafta 3) ile şifrelenip, anahtar ayrı bir kanaldan paylaşılır; ya da uçtan uca şifreli bir kurumsal dosya paylaşım sistemi kullanılır.
- **Bütünlük kontrolü:** teslim edilen paketin bir özet değeri (hash) ayrıca iletilir; laboratuvar dosyayı aldığında hash'i **doğrular** — aktarımda bozulma ya da kurcalama olup olmadığını anlamak için (hafta 3'teki AEAD etiket doğrulamasının kurumsal karşılığı).
- **Erişim kaydı:** kim, ne zaman, hangi dosyayı indirdi kaydı tutulur; bu, kaynak kodun yetkisiz kopyalanmasını sonradan tespit etmeyi sağlar.

Sahada nasıl uygulanır?

Ders projenizde bu kadar ağır bir güvenli kanal kurmanız beklenmez; ama depo erişiminizi **yalnız** takım üyeleri ve öğretim üyesiyle sınırlı tutmak, gerçek bir sır (API anahtarı, şifre) asla commit etmemek (hafta 1, 10'daki sır yönetimi kuralları) aynı ilkenin ders ölçeğindeki karşılığıdır.

Uçtan uca örnek: "GüvenPay Cüzdan SDK v3.2.0" bir laboratuvardan geçiyor

Yukarıdaki tablo 13 adımı özetliyor; ama bir tabloyu okumak ile bir süreci **yaşamak** aynı şey değildir. Aşağıda, tamamen kurgusal bir ürünü ("GüvenPay Cüzdan SDK", sürüm 3.2.0 — bir mobil cüzdan uygulamasının kart saklama ve ödeme kütüphanesi) bu 13 adımdan **atlamadan** geçiriyoruz. Amaç, her adımda kimin ne yaptığını, hangi belgenin elden ele geçtiğini somut biçimde görmek.

Adım 1 — Değerlendirme hedefi (TOE). GüvenPay ekibi laboratuvara şunu yazar: "Değerlendirilecek ürün: GüvenPay Cüzdan SDK, sürüm 3.2.0, Android ARM64 derlemesi, SHA-256 özet değeri `a1b2c3...` (1. haftadaki sürüm kimliği disiplini burada geri döner). Kapsam dışı: uygulamayı barındıran mobil işletim sistemi, kullanıcının cihazı." Değerlendirici bu satırı okur okumaz ilk sorusunu sorar: "peki platform güvenilir mi?" Cevap hayırdır — **varsayım budur** (aşağıda göreceğiz).

Adım 2 — Belgelerin teslimi. GüvenPay, kaynak kodu, API belgesini ve en önemlisi **güvenlik kılavuzunu** şifreli bir kanalla laboratuvara gönderir. Kılavuzda "kart numarası yalnız AES-256-GCM ile şifreli saklanır" gibi cümleler, her biri bir **gereksinim referansıyla** yazılmıştır (bkz. hafta 3 AEAD, hafta 10 sürüm imzalama).

Adım 3 — Gereksinim şablonu. Laboratuvar, seçilen standardın (burada varsayım: EMVCo + PCI MPoC karışımı) numaralı gereksinimlerini bir şablona döker: "Gereksinim 4.2 — Kart verisi saklanırken şifrelenmelidir. Durum: GüvenPay 'karşılandı' diyor, kanıt: kılavuz bölüm 3.1, kaynak kod `kart_deposu.c` satır 40–85." Bu, projenizdeki S17 uyum matrisinin küçük bir örneğidir.

Adım 4 — Atölye. Laboratuvar ve GüvenPay ekibi bir toplantıda (fiziksel ya da çevrimiçi) şablonu birlikte gözden geçirir: "Gereksinim 6.1 (anahtar rotasyonu) için henüz kanıt yok — bunu nasıl kapatacaksınız?" GüvenPay bir belge güncelleme planı sunar.

Adım 5 — Kaynak kod incelemesi. Laboratuvarın iki mühendisi, `kart_deposu.c`, `anahtar_yonetimi.c` ve ağ katmanını **satır satır** okur (beyaz kutu). Bir mühendis, anahtar türetmede sabit bir tuz kullanıldığını fark eder — bu, zafiyet analizi adımına geçmeden **ilk bulgu adayı** olarak not edilir.

Adım 6 — Zafiyet analizi. Laboratuvar, GüvenPay'in varlık listesini (kart numarası, CVV, oturum anahtarı; bu varlık kavramını 1. haftada tanımlamıştık) çıkarır ve her biri için "bu varlık nerede, hangi durumda (bellekte/diskte/ağda) açığa çıkabilir?" sorusunu sorar. Sonuç: bir **sızma testi planı** (bölüm 7) hazırlanır; sabit tuz bulgusu da plana bir test maddesi olarak eklenir.

Adım 7 — Sızma testi. Plandaki her madde, sekiz başlıklı test kartıyla (bölüm 7) yürütülür. Sabit tuz maddesi için: aynı anahtar türetme fonksiyonu iki farklı cihazda çalıştırılır, **aynı** anahtarın çıktığı gözlenir (kanıt). Bu bulgu saldırı potansiyeliyle derecelendirilir (bölüm 5) — hedef bilgisi kamuya açık (kaynak koddaki sabit), ekipman standart, süre kısa → **düşük saldırı potansiyeli** → **ciddi bulgu**.

Adım 8 — İşlevsel uygunluk. Laboratuvar, standart test paketiyle GüvenPay'in **beklenen işlevlerini** de sınar: ödeme akışı doğru tutarı mı gönderiyor, hatalı PIN girişinde doğru şekilde mi reddediyor? Bu, güvenlik testinden ayrı ama tamamlayıcı bir adımdır — güvenli ama **çalışmayan** bir ürün de sertifikalanamaz.

Adım 9 — Bulgular ve düzeltmeler. Laboratuvar raporu: "Bulgu F-03: anahtar türetmede sabit tuz kullanılıyor, kanıt: iki cihazdan aynı anahtar. Öneri: cihaz başına rastgele tuz (hafta 3'teki CSPRNG kuralı) + KDF." GüvenPay yanıt verir: "Kabul, v3.2.1'de düzeltilecek." Bazı küçük bulgular (ör. "günlük mesajları İngilizce değil") ise "güvenlik sorunu değil, iyi uygulama önerisi" diye kapanır.

Adım 10 — Güvenlik etki analizi. GüvenPay, v3.2.1'i hazırlarken bir belge yazar: "Değişiklik türü: hata düzeltmesi (güvenlik). Etkilenen dosya: `anahtar_yonetimi.c`. Güvenlik etkisi: anahtar türetme artık cihaz başına rastgele tuz kullanıyor; önceki sürümdeki zayıflık kapatıldı, başka bir varlığı etkilemiyor."

Adım 11 — Delta değerlendirme. Laboratuvar, tüm ürünü baştan incelemez; yalnız `anahtar_yonetimi.c` dosyasını, etki analizini ve **yeni** TOE kimliğini (v3.2.1, yeni özet değeri) inceler. Adım 7'deki test tekrarlanır: bu kez iki cihazdan **farklı** anahtar çıktığı doğrulanır → bulgu kapanır.

Adım 12 — Ödünleşim ve kalan risk. Rapor kalan bir riski açıkça yazar: "Anahtar rotasyon sıklığı yılda birdir; daha sık rotasyon önerilir ama performans maliyeti nedeniyle GüvenPay bu riski **kabul ettiğini** gerekçeleriyle belgelemiştir." Bu, 1. haftadaki ödünleşim (trade-off) disiplininin sertifikasyondaki karşılığıdır.

Adım 13 — Değişiklik yönetimi. GüvenPay'in v3.2.1'i yayınlaması, kendi temel çizgisine (baseline) göre sınıflandırılır, onaylanır, test edilir, yayınlanır ve laboratuvarca doğrulanır; bu döngü ürünün **hayatı boyunca** her sürümde tekrarlanır — sertifika bir kere alınıp unutulan bir belge değil, **sürdürülen** bir durumdur.

13 adımdan çıkan tek cümle

Bir sertifika, bir **anlık fotoğraf değil**, tanımlı bir sürüm + tanımlı bir standart + tekrarlanabilir bir süreçtir. Ürün değiştiği an, o fotoğraf **eskir**; bu yüzden 10–11. adımlar (etki analizi + delta değerlendirme) sertifikasyonun "bir kerelik" değil **sürekli** olduğunu gösterir.

Değerlendiricinin temel tutumu

Değerlendirici iki varsayımla çalışır: (1) bütün kaynak koda ve belgelere erişimi vardır (**beyaz kutu** bağlam) ve (2) ürünün çalıştığı platform güvenilmezdir. Bu yüzden bir korumanın **var olması** yetmez; **ne kadar kolay aşıldığı** ölçülür. Tek bir korumayı değil, korumaların **birlikte** aşılmasının ne kadar sürdüğünü raporlar; iyi tasarlanmış bir üründe saldırgan birkaç korumayı **zincirlemek** zorunda kalır.

Süreçteki roller: kim kimdir?

GüvenPay örneğinde "laboratuvar" ve "GüvenPay ekibi" dedik ama gerçek bir değerlendirmede en az dört ayrı taraf vardır; bunları karıştırmak, özellikle "kim neye karar veriyor" sorusunda kafa karışıklığına yol açar:

Rol	Kim?	Görevi
Geliştirici (TOE sahibi)	Ürünü yazan ekip (GüvenPay)	Belgeleri, kaynak kodu teslim eder; bulgulara aksiyon üretir
Sponsor	Genellikle geliştiricinin kendisi (bazen bir müşteri/banka)	Değerlendirmeyi **talep eder** ve finanse eder
Değerlendirme laboratuvarı	Bağımsız, akredite kuruluş	Kod inceler, testleri yürütür, bulguları raporlar — **karar vermez**, kanıt üretir
Sertifikasyon otoritesi (şema sahibi)	Ulusal/uluslararası şema yürüten kurum	Laboratuvarın raporunu inceler, **sertifikayı resmen verir**

Sahada nasıl uygulanır?

Değerlendirme laboratuvarlarının kendisi de genellikle bağımsız bir kuruluş tarafından **akredite edilir** (test ve kalibrasyon laboratuvarlarının yeterliliğini tanımlayan ISO/IEC 17025 gibi genel bir çerçeveye göre) — yani "laboratuvar güvenilir mi?" sorusunun da kendi bağımsız cevaplama mekanizması vardır. Bu, "üretici kendi ürününü onaylayamaz" ilkesinin (bölüm 1) bir kademe yukarıya taşınmış hâlidir: laboratuvar da kendi kendini akredite edemez.

Sık yapılan hata: laboratuvarın 'karar verdiğini' sanmak

Laboratuvar sertifika **vermez**; yalnızca test eder ve bulguları **kanıtla** raporlar. Sertifikayı resmen veren taraf **sertifikasyon otoritesidir** (şema sahibi kurum). Bir proje raporunda "laboratuvar bizi sertifikaladı" demek yaygın ama teknik olarak yanlış bir cümledir.

Neden bu üç aşama bu sırada? (Hazırlık → Değerlendirme → Süreklilik)

13 adımı üç aşamaya ayırmıştık; sıranın **neden** böyle olduğunu düşünelim — her aşama bir öncekinin üzerine kurulur, atlanamaz:

- **Hazırlık önce gelir, çünkü** neyin test edileceği (TOE) belirsizse, hiçbir test **tekrarlanabilir** olmaz. Bir laboratuvar önce "tam olarak neyi inceliyorum?" sorusuna kesin bir cevap ister; bu cevap olmadan kod incelemesine başlamak, hangi dosyanın kapsamda olduğunu bile bilmeden çalışmak demektir.

- **Değerlendirme ortada gelir, çünkü** hazırlıktaki gereksinim şablonu (adım 3) olmadan "bu bir bulgu mu, yoksa normal mi?" sorusuna cevap verilemez — bulgu, bir **gereksinime göre** tanımlanır, boşlukta değil.
- **Süreklilik en son gelir, çünkü** ancak bir sertifika/karar **varsa** onun üzerine bir "değişiklik" kavramından söz edilebilir; henüz sertifikalanmamış bir ürün için "delta değerlendirme" anlamsızdır — değişecek bir temel çizgi (baseline) yoktur.

Projenizdeki karşılığı

Kendi döneminizde de aynı sıra işler: önce proje kapsamınızı ve gereksinimlerinizi netleştirdiniz (1–7. haftalar, S0-S13), sonra test/değerlendirme aşamasındasınız (bu hafta, S16), final sonrası bir düzeltme yaparsanız o da kendi küçük "süreklilik" döngünüz olur.

3. Zafiyet değerlendirmesinin yöntemleri

Değerlendirme sürecinin 5. ve 6. adımları (kod incelemesi ve zafiyet analizi), bir dizi yöntemi **doğru sırada** kullanmayı gerektirir. Bunları savunmacı gözüyle, "kendi ürünümü teslim etmeden önce nasıl sınarım?" sorusuyla ele alıyoruz. Sıralama önemlidir: ucuz ve geniş kapsamlı yöntemlerle başlanır, pahalı ve derin yöntemlere doğru gidilir.

Geliştirici önce kendi test ederse değerlendirmeye 'temiz' gider — maliyet düşer.

Sıra	Yöntem	Ne bulur?	Bu dersteki yeri
1	**Kaynak kod incelemesi** (el ile)	Tasarım ve mantık hataları, yanlış kripto kullanımı, sızıntı noktaları	4. hafta CERT/CWE
2	**Statik analiz (SAST)**	Kaynağı çalıştırmadan: bellek hataları, tehlikeli çağrılar, kalıplar	4. hafta (statik analiz)
3	**Dinamik analiz (DAST) + sanitizer'lar**	Çalışırken: bellek erişim hataları, UB, sızıntılar	4. hafta (ASan/UBSan)
4	**Fuzzing**	Beklenmeyen girdilerle çökme/bozulma yolları	4. hafta (libFuzzer/AFL kavramı)
5	**Sızma testi**	Yukarıdakilerin birleştirilmesiyle, saldırgan bakışıyla aşılabilirlik	Bu hafta bölüm 6

Neden bu sıra?

Statik analiz ucuzdur ve geniş tarar ama **yanlış pozitif** üretir; el ile inceleme pahalıdır ama bağlamı görür; fuzzing çalıştırma gerektirir ama insanın aklına gelmeyen girdileri bulur; sızma testi en pahalısıdır ve en son gelir. Önce ucuz yöntemlerle "kolay" bulguları temizlemek, sızma testinin zamanını gerçek, birleşik sorunlara ayırır. Bu, 4. haftadaki "güvenli derleme hattı"nın (CI'da SAST + sanitizer + fuzz) değerlendirmedeki karşılığıdır.

Kod incelemesi: değerlendiricinin baktığı yerler

Kod incelemesi, otomatik araçların kaçırdığı **mantık** hatalarını bulur. Değerlendiricinin öncelikli baktığı yerler (hepsi önceki haftalardan):

- **Kripto kullanımı:** doğru algoritma/kip/dolgu mu? Anahtar nereden geliyor, nereye yazılıyor, ne zaman siliniyor? (3, 10, 11. haftalar)
- **Girdi doğrulama ve bellek:** sınır denetimi, tamsayı taşması, biçim dizisi, UAF (4. hafta CERT).
- **Hata ve sır sızıntısı:** günlükte hassas veri, hata iletilerinde iç durum (4, 6. haftalar).
- **Denetim atlama yolları:** bir güvenlik denetiminin tek bir dalla ya da tek bir dönüş değeriyle atlanabilmesi (9. hafta opak boolean, rastgele çıkış).

Her madde neden bu listede? Çünkü hepsi **otomatik araçların gözden kaçırdığı** hatalardır: bir SAST aracı "burada `AES_encrypt` çağrılıyor" diyebilir ama "doğru **kip** kullanılmış mı" (bölüm 3'ün haftası olan 3. haftadaki AES-GCM/CBC/CTR ayrımı) sorusuna genellikle cevap veremez — bu, bağlamı anlayan bir **insanın** işidir.

Sahada nasıl uygulanır?

Gerçek bir değerlendirmede kod incelemesi, SAST ve sanitizer'lar genellikle geliştiricinin **CI/CD hattına** (4. haftadaki güvenli derleme hattı) zaten gömülüdür; laboratuvar önce bu hattın **çıktılarını** (geçmiş çalıştırma kayıtları) ister, sonra kendi bağımsız taramasını yapar. İki sonucun **tutarlı** olması (laboratuvarın bulduğu ile geliştiricinin kendi CI'ının bulduğu) ek bir güven işaretidir; tutarsızlık varsa (laboratuvar bir şey buluyor ama CI hiç bulmuyor) CI yapılandırması sorgulanır.

Statik ve dinamik analiz, fuzzing

Bu üç yöntemi 4. haftada güvenli derleme hattının parçası olarak gördük. Değerlendirmede aynı araçlar, bu kez **bağımsız** bir tarafça çalıştırılır. Önemli olan, geliştiricinin bu araçları **kendisinin** çalıştırmış ve sonuçlarını (S16) sunmuş olmasıdır: değerlendirici, "bu ürünü daha önce kimse sınamamış" ile "geliştirici şu araçları çalıştırmış, şu bulguları kapatmış" arasında büyük fark görür.

Bir bellek hatasını beş yöntemle izlemek

Bölüm 0'da bir **mantık** hatasını (SQL enjeksiyonu) beş yöntemle karşılaştırmıştık; şimdi tersini yapıp bir **bellek** hatasını izleyelim — çünkü yöntemlerin güçlü olduğu alanlar farklıdır.

tampon_kopyala.c (sentetik, hatalı)

```
void tampon_kopyala(char *hedef, const char *kaynak)
{
    strcpy(hedef, kaynak);   /* HATA: hedefin boyutu kontrol edilmiyor */
}

void isle(const char *disaridan_gelen)
{
    char arabellek[64];
    tampon_kopyala(arabellek, disaridan_gelen);   /* disaridan_gelen 64 bayttan uzunsa tasma */
}
```

Sıra	Yöntem	Bu hatayı bulur mu?	Neden
1	Kod incelemesi	Deneyimli bir inceleyici **evet** bulur	`strcpy` + sabit boyutlu tampon deseni, CERT kurallarını (hafta 4) bilen biri için tanıdıktır
2	SAST	Çoğunlukla **evet**	`strcpy`, `sprintf`, `gets` gibi "tehlikeli fonksiyon" kalıpları SAST kural kümelerinde neredeyse her zaman vardır
3	DAST + sanitizer	**Evet, kanıtla**	Program 65+ baytlık bir girdiyle **çalıştırılırsa** ASan taşmayı anında yakalar ve tam olarak hangi satırda olduğunu gösterir (hafta 4)
4	Fuzzing	**Evet, çoğunlukla en güçlü yöntem burada**	Fuzzing rastgele uzunlukta girdiler dener; bir bellek hatası genellikle **birkaç dakika–saat** içinde tetiklenir — bu tür hatalarda fuzzing SQLi'den çok daha etkilidir
5	Sızma testi	**Evet ama en pahalı yol**	Saldırgan uzun bir girdiyle programı çökertip devamında **çalıştırılabilir kod enjeksiyonuna** çevirip çeviremediğini araştırır

Karşılaştırma: SQLi gibi mantık hataları kod incelemesi ve SAST'ta güçlüdür, fuzzing'de zayıftır; bellek hataları ise fuzzing ve sanitizer'larda çok güçlüdür. **Bu yüzden bölüm 3'ün sırası (kod inceleme → SAST → DAST/sanitizer → fuzzing → sızma testi) her hata sınıfını en az bir yöntemin güçlü olduğu bir noktadan geçirir.**

Sık yapılan hata: fuzzing'i birkaç saniye çalıştırıp 'bulgu yok' demek

Fuzzing **kapsam (coverage) biriktirerek** çalışır; ilk birkaç saniyede yalnızca yüzeysel yollar denenir. Kısa bir çalıştırmadan sonra "temiz" demek, aslında "henüz derine inemedi" demektir. **Kural:** fuzzing süresi ve ulaşılan kod kapsamı (`%` olarak) rapora birlikte yazılır; "X saat çalıştı, kod kapsamı %Y'ye ulaştı, N çökme bulundu" biçiminde. Süre yazılmayan bir fuzzing sonucu değerlendirici için **anlamsızdır**.

Kural: yöntemi hatanın türüne göre seçin

"Tek bir araç her şeyi bulur" varsayımı yanlıştır. Mantık hataları için el ile inceleme ve SAST; bellek/UB hataları için sanitizer ve fuzzing; birleşik/gerçek dünya senaryoları için sızma testi güvenilir sonuç verir. Kendi projenizde S16'yı doldururken, her test maddesi için "bu hatayı bulmaya en uygun yöntem hangisi?" sorusunu sorun.

Yanlış pozitif ve yanlış negatif: otomatik araçların iki riski

Bölüm başındaki "SAST hızlı ama yanlış pozitif üretir" cümlesini açalım — bu iki terim değerlendirmede sürekli geçer:

- **Yanlış pozitif (false positive):** aracın "burada sorun var" dediği ama aslında **sorun olmayan** durum. Örnek: SAST, `strcpy` çağrısını görür görmez işaretler; ama hedef tampon zaten güvenli biçimde büyük ayrılmışsa bu bir yanlış pozitiftir. Çok sayıda yanlış pozitif, geliştiricinin aracı **kapatmasına** (alarm yorgunluğu) yol açar.
- **Yanlış negatif (false negative):** aracın **kaçırdığı gerçek** sorun. Örnek: bir SAST aracı yalnız bilinen tehlikeli fonksiyon listesine bakıyorsa, aynı hatayı yapan ama listede olmayan özel bir sarmalayıcı (wrapper) fonksiyonu kaçırır. Yanlış negatif, yanlış pozitiften **daha tehlikelidir**: "araç temiz dedi" güveni sahte olur.

Hangisi daha kötü?

Yanlış pozitif zaman kaybettirir; yanlış negatif **güvenlik açığını gizler**. Bir değerlendirici bu yüzden hiçbir otomatik aracın çıktısına körü körüne güvenmez — el ile inceleme (bölüm 3, yöntem 1) her zaman **bağımsız** bir kontrol katmanı olarak kalır.

Neden "ucuzdan pahalıya" sıra bir maliyet meselesidir

Bölüm başındaki sıralamayı (kod inceleme → SAST → DAST/sanitizer → fuzzing → sızma testi) bir de **göreli maliyet** açısından okuyalım. Aşağıdaki tablo **gerçek bir ölçüm değil**, yalnız fikri somutlaştırmak için verilen **gösterim amaçlı** bir sıralamadır:

Yöntem	Bir bulgu başına göreli çaba (gösterim amaçlı)	Ne kadar geniş tarar?
SAST	Çok düşük (otomatik, dakikalar)	Çok geniş (bütün kod tabanı)
Kod incelemesi	Orta (insan saatleri)	Orta (inceleyenin baktığı kadar)
DAST + sanitizer	Düşük-orta (test çalıştırma + otomatik tespit)	Yalnız çalıştırılan yollar
Fuzzing	Orta (makine saatleri, insan az)	Zamanla genişler (coverage'a bağlı)
Sızma testi	Yüksek (uzman insan günleri)	Dar ama **gerçekçi** (gerçek saldırgan senaryosu)

Bu tablo neden 'gösterim amaçlı'?

Gerçek maliyetler; kod tabanının büyüklüğüne, ekibin deneyimine, araçların olgunluğuna göre çok değişir. Burada önemli olan **sıra**dır: ucuz/geniş yöntemler önce, pahalı/derin yöntemler sonra gelir — bu sıralamanın **kendisi** evrensel bir prensiptir, tablodaki sayılar değil.

Yöntemler birbirini nasıl doğrular? Çapraz kontrol

Bir yöntemin bulduğu bir şeyi başka bir yöntemle **doğrulamak**, değerlendirmede güveni artırır. Örnek: SAST bir tamsayı taşması **adayı** işaretler (yanlış pozitif olabilir); DAST/sanitizer aynı yolu gerçek bir girdiyle **çalıştırıp** taşmanın **gerçekten** oluştuğunu gösterirse, bu artık bir yanlış pozitif değil, **doğrulanmış** bir bulgudur. Sızma testi, genellikle bu doğrulanmış bulguları **birleştirerek** (chaining) gerçek bir saldırı senaryosu kurar — tek başına düşük etkili görünen iki bulgu, birlikte kullanıldığında ciddi bir zincir oluşturabilir.

Sık yapılan hata: 'SAST işaretledi' demeyi 'doğrulandı' ile karıştırmak

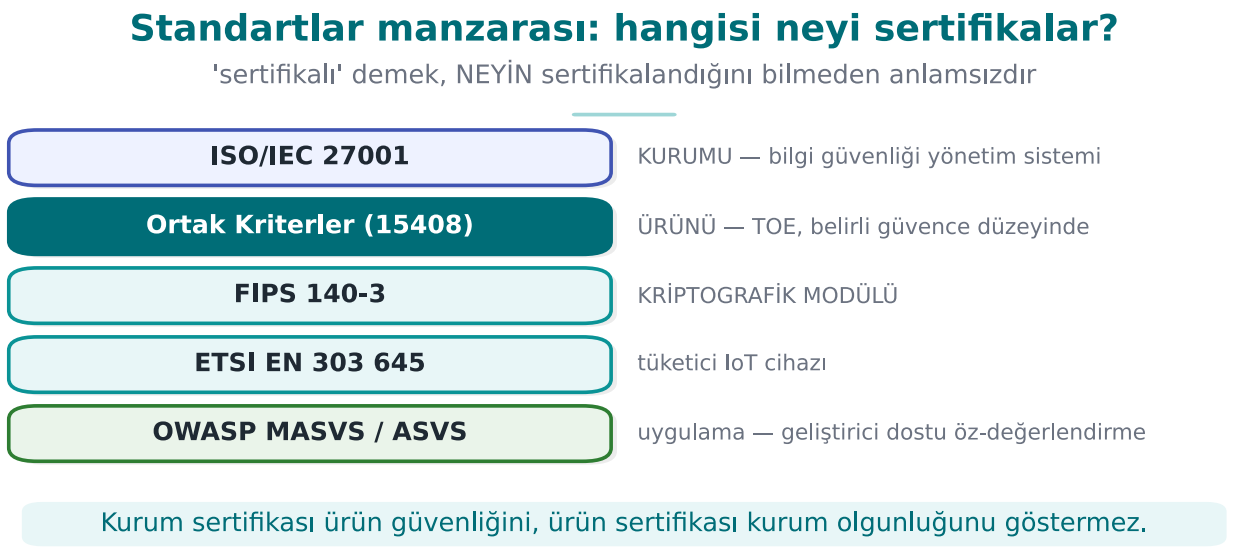
Bir SAST uyarısını doğrulamadan doğrudan rapora "bulgu" olarak yazmak, raporu yanlış pozitiflerle şişirir. **Kural:** otomatik bir aracın işaretlediği her aday, mümkünse başka bir yöntemle (kod incelemesi ya da çalıştırarak) **doğrulanmadan** kesin bulgu sayılmaz.

Kendi projenizde bölüm 3'ü uygularken kontrol listesi

- Kod incelemesi yapıldı mı — özellikle kripto kullanımı, girdi doğrulama, hata/sır sızıntısı, denetim atlama yolları (yukarıdaki dört madde)?
- SAST ve DAST/sanitizer (4. haftadaki güvenli derleme hattı) çalıştırıldı mı, sonuçlar S16'ya eklendi mi?
- Fuzzing çalıştırıldıysa **süresi** ve ulaşılan kapsam yazıldı mı (yoksa sonuç anlamsızdır)?
- Her SAST/fuzzing bulgusu, başka bir yöntemle (kod incelemesi ya da çalıştırarak) **doğrulandı** mı?
- Yöntemler **sırayla** mi (ucuzdan pahalıya) uygulandı, yoksa doğrudan sızma testine mi atlandı?

4. Standartların istediği testler

Farklı standartlar farklı test türlerini vurgular. Projenizin hangi gereksinim ailesine yakın olduğunu bilmek, hangi testleri önceliklendireceğinizi belirler.



Standart / çerçeve	Ağırlıklı beklediği test	Kısa not
ISO/IEC 27001	Süreç ve yönetim denetimi	Ürünü değil, **kurumu** sertifikalar; test yerine kontrol kanıtı
Ortak Kriterler (ISO/IEC 15408)	Kaynak inceleme + zafiyet analizi + sızma testi; **saldırı potansiyeli** derecelendirme	Derinlik EAL ile artar (13. hafta)
FIPS 140-3	Kriptografik modül testleri (algoritma doğrulama, kendini test)	Yalnız modülü kapsar (13. hafta)
ETSI EN 303 645	Temel IoT güvenlik gereksinimlerinin doğrulanması	Hafif, geniş kapsamlı
EMVCo / PCI	İşlevsel uygunluk + laboratuvar sızma testi + saldırı potansiyeli	Ödeme alanı; sıkı
OWASP MASVS + MASTG	Mobil uygulama güvenlik testi (statik + dinamik)	Projeniz için en uygulanabilir rehber

Test yöntemi rehberleri

Sızma testinin **nasıl** yürütüleceği için olgunlaşmış açık rehberler vardır: web için **OWASP WSTG**, mobil için **OWASP MASTG**, genel süreç için **PTES** ve **NIST SP 800-115**. Bunlar bir "saldırı tarifi" değil, kapsamı ve yöntemi standartlaştıran **test metodolojileridir**; bir bulgunun tekrarlanabilir ve karşılaştırılabilir olmasını sağlarlar.

Her standardın somut bir test örneği

Tablodaki "ağırlıklı beklediği test" satırını somutlaştıralım — her standart için **bir örnek test sorusu**:

- **ISO/IEC 27001:** "Kart verisine erişimi olan çalışanların listesi güncel mi, erişim üç ayda bir gözden geçiriliyor mu?" — bu bir **kod** sorusu değil, bir **süreç** sorusudur; kanıt bir erişim kayıt defteridir, bir test kartı değil.
- **Ortak Kriterler:** "Anahtar türetme fonksiyonunun tersine mühendislikle kaç saatte kırıldığı, hangi uzmanlık düzeyinde?" — doğrudan bölüm 5'teki saldırı potansiyeli sorusudur.
- **FIPS 140-3:** "Modül açılışta kendi kendini test ediyor mu (power-up self-test), kriptografik algoritma onaylı bir uygulamayla mı çalışıyor?" — algoritma **doğrulama** sorusudur, saldırı senaryosu değil.
- **ETSI EN 303 645:** "Cihazda varsayılan/tahmin edilebilir bir parola var mı?" — temel, geniş kapsamlı bir kontrol sorusudur; derinlemesine sızma testi beklemez.
- **EMVCo / PCI:** "Kart verisi bellekte ne kadar süre açık kalıyor, işlem bitince siliniyor mu?" — hem kod incelemesi hem laboratuvar sızma testi gerektirir.
- **OWASP MASVS:** "Uygulama, kök/jailbreak yapılmış bir cihazda çalışırken hassas veriyi koruyor mu?" — hem statik hem dinamik test ister.

Projeniz için hangi standart 'en yakın'?

Kendi projenizin gereksinim ailesini seçerken şu soruyu sorun: "biz bir **kurumu mu**, bir **kriptografik modülü mü**, bir **genel ürünü mü**, yoksa bir **mobil uygulamayı mı** değerlendiriyoruz?" Bu dersin projesi çoğunlukla üçüncü ve dördüncü kategoriye (genel ürün / mobil uygulama) girer; bu yüzden **OWASP MASVS/MASTG** en doğrudan uygulanabilir rehberdir, ama saldırı potansiyeli kavramını **Ortak Kriterler** geleneğinden ödünç alıyoruz.

GüvenPay'in altı sorusu tek tek yanıtlansa nasıl görünür?

Yukarıdaki altı örnek soruyu GüvenPay'e uygulayıp, aynı ürünün **farklı mercekler**den nasıl göründüğünü görelim — bu, "aynı ürün dört farklı sınavdan geçebilir" fikrinin (bölüm 1) somutlaşmış hâlidir:

Soru	GüvenPay'in cevabı (kurgusal)
Erişim kaydı güncel mi? (ISO 27001)	GüvenPay'in kendisi değil, GüvenPay'i barındıran şirketin BT departmanı bu soruya cevap verir
Anahtar türetme kaç saatte kırılır? (Ortak Kriterler)	F-03 düzeltmesi öncesi: saatler (TEMEL). Sonrası: aylar (YÜKSEK)
Modül kendi kendini test ediyor mu? (FIPS 140-3)	GüvenPay bir kriptografik modül **değil**, bir uygulama SDK'sıdır — bu soru kapsam dışıdır
Varsayılan parola var mı? (ETSI)	GüvenPay bir IoT cihazı değil; bu soru da kapsam dışıdır
Kart verisi bellekte ne kadar açık kalıyor? (EMVCo/PCI)	Kullanım sonrası bellek sıfırlanıyor (kanıt: PT-05)
Root'lu cihazda veri korunuyor mu? (MASVS)	Evet — PT-07 test kartı bunu kanıtlıyor (bölüm 7)

Bu tablodan çıkan ders

GüvenPay'in **her** standarda aynı ölçüde "yakın" olmadığını görüyoruz: FIPS 140-3 ve ETSI soruları kapsam dışı çıktı, ama Ortak Kriterler, EMVCo/PCI ve MASVS soruları doğrudan uygulanabilir oldu. Kendi projeniz için de aynı egzersizi yapmanız, hangi standarda **gerçekten** yakın olduğunuzu netleştirir.

Ödeme alanında iki ayrı kapsam: yazılım mı, kurum mu?

Ödeme güvenliği alanında "PCI" tek bir standart değil, bir **standartlar ailesidir**; hangisinin geçerli olduğu "neyin" değerlendirildiğine bağlıdır:

- **PCI DSS (Data Security Standard):** kart verisi işleyen bir **kurumun/ortamın** uyumunu denetler — sunucular, ağ segmentasyonu, erişim kayıtları. Bir yazılımın kendisi değil, o yazılımın **çalıştığı ortam** kapsamdadır. ISO/IEC 27001'e benzer şekilde (bölüm 1) **süreç ve ortam** odaklıdır.
- **PCI yazılım standartları (ör. SSF — Software Security Framework, MPoC — Mobile Payments on COTS):** bir **yazılımın kendisinin** (kart verisini nasıl işlediği, sakladığı, sildiği) güvenli tasarlanıp tasarlanmadığını değerlendirir — bu, GüvenPay gibi bir SDK'nın kapsamına daha yakındır.
- **EMVCo:** çip ve yazılım tabanlı ödeme **akışlarının** (işlem doğrulama, kriptografik protokol adımları) uygun şekilde uygulanıp uygulanmadığını, laboratuvar sızma testiyle değerlendirir.

Sık yapılan hata: 'PCI uyumluyuz' derken hangi PCI'ı kastettiğini belirtmemek

"PCI uyumluyuz" cümlesi eksiktir: PCI **DSS** uyumu bir işletmenin ortamıyla ilgilidir, PCI **SSF/MPoC** uyumu bir yazılımın kendisiyle. Bir mobil ödeme SDK'sı geliştiren bir ekip için doğru cümle "yazılımımız PCI MPoC gereksinimlerini hedefliyor" olabilir; "PCI DSS uyumluyuz" demek — DSS bir kurumsal ortam standardı olduğu için — SDK'nın kendisi hakkında hiçbir şey söylemez.

Test derinliği her standartta aynı değildir

Aynı "sızma testi" sözcüğü, standarttan standarda **çok farklı derinlikte** bir çalışmayı ifade edebilir:

Derinlik	Tipik süre	Örnek bağlam
Yüzeysel tarama	Saatler	Temel IoT kontrolü
Odaklı fonksiyonel test	Günler	OWASP MASVS L1
Derin, uzman ekip	Haftalar	Ortak Kriterler yüksek güvence düzeyleri (13. hafta), EMVCo
Sürekli/bağımsız üçüncü parti	Aylar boyunca tekrarlanan	PCI DSS yıllık değerlendirme döngüsü

Derinlik neden değişir?

Bir standardın hedef aldığı **varlığın değeri** ve **saldırgan modeli** ne kadar yüksekse, beklenen derinlik o kadar artar. Bir IoT ampulün temel güvenliği ile bir ödeme kartının anahtar hiyerarşisi aynı titizlikte test edilmez — çünkü kırılmalarının sonucu **aynı ölçekte değildir**. Bu, 13. haftadaki güvence düzeyleri tartışmasının ön hazırlığıdır.

Aynı gereksinim, altı farklı sözle: bir karşılaştırma

"Kart verisi şifrelenmelidir" gibi tek bir güvenlik gereksinimi, farklı standartlarda **farklı sözcüklerle**, ama aynı temel fikirle karşımıza çıkar. Bu benzerliği görmek, 13. haftadaki uyum matrisini doldururken çok işinize yarayacak:

Standart	Aynı fikri nasıl ifade eder?
ISO/IEC 27001	"Kriptografik kontroller politikası" (Ek A maddesi) — kurumun şifreleme **politikası** olmalı
Ortak Kriterler	"Gizlilik" güvenlik işlevi gereksinimi (FCS ailesi) — TOE'nin hangi kriptografik işlevi **nasıl** sağladığı tanımlanır
FIPS 140-3	Onaylı bir algoritmanın, onaylı bir kip ve anahtar yönetimiyle **kullanılması**
ETSI EN 303 645	"Hassas güvenlik parametreleri güvenli şekilde saklanmalı" (temel gereksinim)
EMVCo / PCI	Kart verisinin saklanırken ve iletilirken **şifrelenmesi**, anahtar yönetimi süreçleri
OWASP MASVS	"MASVS-CRYPTO" kontrol ailesi — uygulamanın doğru kriptografik ilkeleri kullanması

Kural: gereksinimin 'ruhunu' önce anlayın, sonra standarda çevirin

Bu altı satırın **hepsi aynı şeyi** söylüyor: "hassas veriyi güçlü, doğru kullanılan kriptografiyle koru." Bir gereksinim şablonunu doldururken önce bu ortak fikri anlamak, sonra hangi standardın hangi kelimeleri kullandığını öğrenmek, ezberden çok daha kalıcıdır.

Standartları kim yazıyor, kim güncelliyor?

Zayıf bir öğrencinin sık sorduğu, ama nadiren yanıtlanan bir soru: "bu standartları kim uyduruyor?" Kısa cevap: hiçbiri tek bir kişinin ya da şirketin ürünü değildir; her biri, o alandaki **birçok kurumun ortak çalışmasıyla** oluşturulur ve düzenli olarak (genelde birkaç yılda bir) güncellenir — tıpkı 2. haftada gördüğümüz CVSS'in v2'den v3.1'e, v4.0'a evrilmesi gibi. Bu güncellemelerin nedeni genelde ya (a) yeni bir saldırı tekniğinin ortaya çıkması ya da (b) önceki sürümün belirsiz/tutarsız bulunan bir maddesinin netleştirilmesidir.

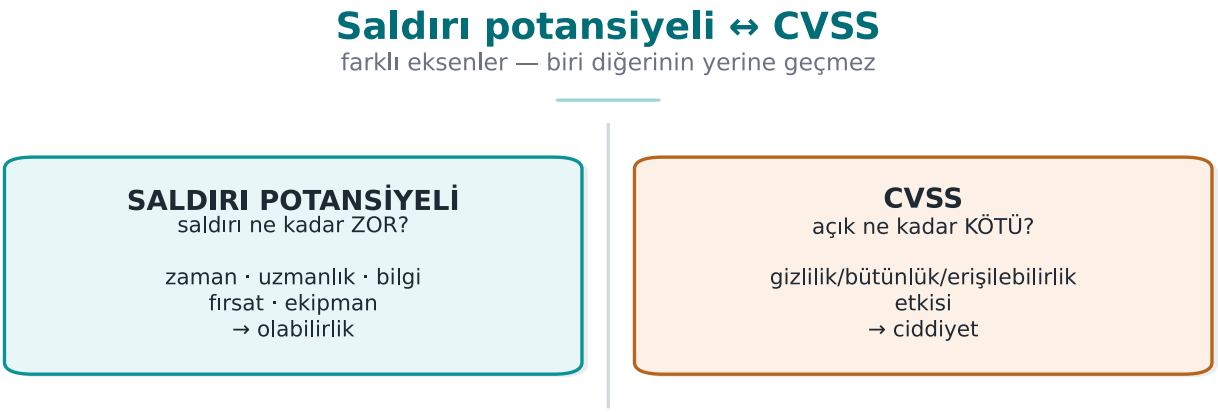
Bu neden önemlidir?

Bir projede "standart X'e uyuyoruz" derken **hangi sürümü** kastettiğinizi belirtmek gerekir — tıpkı TOE kimliğinde sürüm + hash gerektiği gibi (bölüm 2). Eski bir sürüme göre tasarlanmış bir gereksinim şablonu, güncel tehdit modelini yansıtmayabilir.

5. Saldırı potansiyeli ve bulgu derecelendirme

Bir korumanın "var olması" yetmediğini, "ne kadar kolay aşıldığının" ölçülmesi gerektiğini söylemiştik. Ortak Kriterler ve ödeme şemaları bu ölçüyü **saldırı potansiyeli** (attack potential) puanlamasıyla yapar. Bir bulgunun ciddiyeti, "kırmak için ne gerekti?" sorusuna verilen yanıtla belirlenir. Bu, 9. haftadaki "gizlemenin ölçülmesi" (güç, dayanıklılık) çerçevesinin değerlendirmedeki resmî halidir.

Saldırı potansiyeli saldırının ZORLUĞUNU, CVSS ise açığın ETKİSİNİ ölçer.



Bulgu raporunda ikisi birlikte verilir; önceliklendirme ikisine göre yapılır.

Saldırı potansiyeli faktörleri

Bir saldırıyı gerçekleştirmek için gereken çabayı beş–altı faktörle puanlarız; her faktörün puanları toplanır ve toplam bir dirençlik düzeyine (ör. "temel / orta / yüksek / çok yüksek") eşlenir.

Faktör	Soru	Düşük puan	Yüksek puan
Geçen süre (elapsed time)	Saldırı ne kadar sürdü?	Bir gün	Aylar
Uzmanlık (expertise)	Ne düzeyde beceri gerekti?	Sıradan kullanıcı	Konu uzmanı
Hedef bilgisi (knowledge of TOE)	Ürün hakkında ne bilmek gerekti?	Kamuya açık	İç/gizli belgeler
Fırsat penceresi (window of opportunity)	Ne kadar erişim/deneme gerekti?	Sınırsız/uzaktan	Kısıtlı fiziksel erişim
Ekipman (equipment)	Hangi araçlar gerekti?	Standart, ücretsiz	Özel, pahalı

Yorum: Toplam puan **yüksekse**, saldırı zor demektir; koruma iyidir. Puan **düşükse** (ör. sıradan kullanıcı, bir saatte, ücretsiz araçla), bu ciddi bir bulgudur. Buradaki mantık 9. haftadaki dört ölçütle aynıdır: bir savunmanın gücü, onu aşmanın maliyetiyle ölçülür.

Her faktör neden orada? Tek tek gerekçesi

Beş faktör rastgele seçilmemiştir; her biri gerçek bir saldırganın karşılaştığı **farklı bir maliyet türünü** temsil eder. Zayıf bir öğrencinin en çok karıştırdığı nokta budur — "bunlar neden bu beşi, başka değil?" sorusunun cevabı:

- **Geçen süre:** saldırganın **zamanı** sonsuz değildir; bir kurumsal saldırgan bile bir açığı bulmak için ayları değil genelde günleri göze alır. Süre arttıkça, gerçek dünyada bu saldırıyı **deneyecek kişi sayısı azalır**.
- **Uzmanlık:** herkes tersine mühendislik ya da kriptanaliz yapamaz. Uzmanlık düzeyi arttıkça, o saldırıyı gerçekleştirebilecek **insan havuzu daralır** — "sıradan kullanıcı" milyonlarca kişiyi kapsarken "çoklu uzman" belki birkaç yüz kişiyi kapsar.
- **Hedef bilgisi:** kamuya açık bir bilgiyle (ör. genel API belgesi) yapılabilen bir saldırı, iç/gizli belge gerektiren bir saldırıdan **çok daha erişilebilirdir** — çünkü saldırganın önce o bilgiyi **sızdırması ya da çalması** gerekmez.
- **Fırsat (erişim penceresi):** bir saldırı sınırsız deneme hakkıyla (ör. kendi cihazında, offline) mı yapılıyor, yoksa kısıtlı bir pencerede (ör. bir ATM'ye üç dakikalığına erişim) mi? Deneme sayısı sınırlıysa, saldırganın **ilk seferde başarılı olması** gerekir — bu çok daha zordur.
- **Ekipman:** standart, ücretsiz bir araçla (ör. bir hata ayıklayıcı) yapılabilen bir saldırı; özel-uyarlanmış, pahalı bir laboratuvar ekipmanı (ör. odaklı iyon ışını) gerektiren bir saldırıdan **erişilebilirlik** açısından bambaşka bir kategoridedir.

Kural: her faktör 'kaç kişi bunu yapabilir' sorusunu farklı bir açıdan daraltır

Beş faktörün toplamı aslında dolaylı olarak şu soruyu yanıtlar: "gerçek dünyada, bu saldırıyı gerçekleştirebilecek kaç kişi/kurum var?" Düşük puan = çok kişi yapabilir = yaygın risk. Yüksek puan = neredeyse kimse yapamaz = dar, uzman bir tehdit kümesi.

Örnek (sentetik)

"Sürüm derlemesinde bir lisans denetimi, ücretsiz bir tersine derleyiciyle, orta düzey bir kullanıcının bir saatte bulup tek bayt yamasıyla atlayabildiği bir dala bağlıydı." → süre: düşük, uzmanlık: orta, bilgi: kamuya açık, ekipman: ücretsiz → **düşük saldırı potansiyeli → ciddi bulgu.** Öneri: opak boolean + rastgele çıkış (9. hafta) ve sunucu tarafı denetim.

Puan tablosu (derste kullanacağımız sadeleştirilmiş ölçek)

Yukarıdaki tablo "düşük/yüksek" diyor; puanlama yapabilmek için **sayılar** gerekir. Derste (ve demoda) şu sadeleştirilmiş Ortak Kriterler/JIL ölçeğini kullanıyoruz:

Faktör	Seviye 0	Seviye 1	Seviye 2	Seviye 3
Geçen süre	< 1 gün → **0**	< 1 hafta → **4**	< 1 ay → **10**	aylar → **19**
Uzmanlık	sıradan → **0**	yetkin → **3**	uzman → **6**	çoklu uzman → **8**
Hedef bilgisi	kamuya açık → **0**	kısıtlı → **3**	hassas → **7**	kritik → **11**
Fırsat	sınırsız/uzak → **0**	kısıtlı → **4**	çok kısıtlı → **10**	—
Ekipman	standart/ücretsiz → **0**	özel → **4**	özel-uyarlanmış → **7**	—

Toplam puanın karşılığı:

Toplam	Düzey
0–9	**TEMEL** — düşük direnç → **ciddi bulgu**
10–13	ORTA
14–19	YÜKSEK
20–24	ÇOK YÜKSEK
25+	ÖTESİ (yalnız çok yetenekli saldırgan)

Uçtan uca örnek: bir bulguyu baştan sona puanlayalım

Senaryo. "Sürüm derlemesindeki lisans denetimi, ücretsiz bir tersine derleyiciyle, **yetkin** (uzman değil) bir kullanıcının **bir saatte** bulup **tek bayt yamasıyla** atlayabildiği bir dala bağlı."

Beş faktörü tek tek okuyup puanlıyoruz:

#	Faktör	Senaryodaki karşılığı	Seviye	Puan
1	Geçen süre	bir saat	< 1 gün	**0**
2	Uzmanlık	yetkin kullanıcı	yetkin	**3**
3	Hedef bilgisi	mağazadan inen ikili, belge gerekmedi	kamuya açık	**0**
4	Fırsat	kendi cihazında sınırsız deneme	sınırsız/uzak	**0**
5	Ekipman	ücretsiz tersine derleyici	standart/ücretsiz	**0**

```
Toplam = 0 + 3 + 0 + 0 + 0 = 3        →  3 ≤ 9  →  TEMEL
```

Sonuç: TEMEL düzey = düşük direnç = ciddi bulgu. Rapora "önce bu kapatılmalı" diye yazılır.

Şimdi koruma ekleyelim ve yeniden puanlayalım. 9. haftadan opak boolean + rastgele çıkış, üstüne sunucu tarafı denetim koyduk. Aynı saldırı artık: bir haftada (4), **uzman** gerektiriyor (6), iç davranışı öğrenmek **kısıtlı bilgi** istiyor (3), sunucu denetimi yüzünden deneme **kısıtlı** (4), ekipman hâlâ ücretsiz (0):

```
Toplam = 4 + 6 + 3 + 4 + 0 = 17       →  14 ≤ 17 ≤ 19  →  YÜKSEK
```

Yani koruma, bulguyu **TEMEL (3)** düzeyinden **YÜKSEK (17)** düzeyine taşıdı. "Koruma ekledim" demek yerine **ölçüyle** konuşmuş olduk; S16'ya yazılacak cümle budur.

Aynı sayıları demoda üretin

`code/week-12/02-saldiri-potansiyeli` demosu bu ölçeği uygular. Faktör seviyelerini argüman olarak verin (sıra: süre uzmanlık bilgi fırsat ekipman):

```
./bin/linux/saldiri_potansiyeli 0 1 0 0 0     # korumasız hâl  -> puan 3  (TEMEL)
./bin/linux/saldiri_potansiyeli 1 2 1 1 0     # korumalı hâl   -> puan 17 (YÜKSEK)
```

Windows'ta: `.\bin\windows\saldiri_potansiyeli.exe 0 1 0 0 0`. Argümansız çalıştırırsanız gömülü üç örnek bulguyu (lisans yaması · WBC anahtar çıkarma · güvenli öğe kırma) puanlar.

İkinci uçtan uca örnek: tam tersi uçta bir senaryo ("güvenli öğe kırma")

İlk örnekte (lisans denetimi) saldırı **çok kolaydı**. Şimdi ölçeğin öbür ucuna, demodaki üçüncü gömülü örneğe ("Güvenli öğe (SE) kırma") bakalım — bu kez bir saldırının **ne kadar zor olabileceğini** görmek için.

Senaryo B. "Bir ödeme kartındaki güvenli öğeden (secure element) simetrik ana anahtarın, donanım seviyesinde bir yan kanal/hata enjeksiyonu saldırısıyla çıkarılması. Saldırgan, çip üreticisinin iç belgelerine ve özel olarak uyarlanmış laboratuvar ekipmanına (ör. hassas ölçüm probu, odaklı iyon ışını) erişen, birden fazla alanda (donanım + kriptanaliz) uzman bir **ekip** gerektiriyor; saldırı aylar sürüyor ve cihaza **çok kısıtlı** fiziksel erişim istiyor."

Beş faktörü aynı sadeleştirilmiş ölçekle (yukarıdaki puan tablosu) tek tek puanlayalım:

#	Faktör	Senaryodaki karşılığı	Seviye	Puan
1	Geçen süre	saldırı aylar sürüyor	aylar (Seviye 3)	**19**
2	Uzmanlık	donanım + kriptanaliz, birden fazla uzman	çoklu uzman (Seviye 3)	**8**
3	Hedef bilgisi	çip üreticisinin iç belgeleri gerekiyor	kritik (Seviye 3)	**11**
4	Fırsat	cihaza çok kısıtlı fiziksel erişim	çok kısıtlı (Seviye 2)	**10**
5	Ekipman	odaklı iyon ışını, özel ölçüm probu	özel-uyarlanmış (Seviye 2)	**7**

```
Toplam = 19 + 8 + 11 + 10 + 7 = 55        →  55 >= 25  →  OTESI (yalniz cok yetenekli saldirgan)
```

Sonuç: ÖTESİ düzey = çok yüksek direnç = düşük öncelikli bulgu (yine de sıfır risk demek değildir — çok kaynaklı bir saldırganın erişimi olabilir; rapor bunu "kalan risk" olarak yazar, bölüm 6 ve 12'de göreceğiz).

Demoda doğrulayın

`code/week-12/02-saldiri-potansiyeli` demosunu **argümansız** çalıştırın; üçüncü gömülü örnek ("Guvenli oge (SE) kirma") tam olarak `SURE[3] + UZMAN[3] + BILGI[3] + FIRSAT[2] + EKIPMAN[2] = 19+8+11+10+7 = 55` hesabını yapar ve ekrana `puan=55 -> OTESI` yazar — yukarıda elle yapılan hesapla **birebir** aynı sonucu üretir.

Üç sonucun karşılaştırması:

	Senaryo A (lisans, korumasız)	Senaryo A (lisans, korumalı)	Senaryo B (SE kırma)
Toplam puan	3	17	55
Düzey	TEMEL	YÜKSEK	ÖTESİ
Anlamı	Ciddi bulgu, hemen kapatılmalı	Kabul edilebilir, sürekli izlenmeli	Gerçekçi tehdit modelinde göz ardı edilebilir

Ek alıştırma: demodaki üçüncü gömülü örneği elle doğrulayalım

İki uç örneği (3 ve 55) gördük; demonun ikinci gömülü bulgusunu ("WBC anahtar çıkarma") da elle hesaplayıp aracın çıktısıyla karşılaştıralım:

"Beyaz kutu kriptografi (white-box crypto) uygulamasından anahtar çıkarma; saldırgan **çoklu uzman** düzeyinde bilgi gerektiren farklı analiz (DCA — differential computation analysis) tekniğini uyguluyor, ürünün iç yapısı hakkında **hassas** (genel kullanıcıya kapalı) belgeye ihtiyaç duyuyor, saldırı **bir ay** kadar sürüyor, cihaza erişimi **kısıtlı** ve **özel** (hazır satılan bir analiz aracı) kullanıyor."

#	Faktör	Seviye	Puan
1	Geçen süre	< 1 ay (Seviye 2)	**10**
2	Uzmanlık	çoklu uzman (Seviye 3)	**8**
3	Hedef bilgisi	hassas (Seviye 2)	**7**
4	Fırsat	kısıtlı (Seviye 1)	**4**
5	Ekipman	özel (Seviye 1)	**4**

```
Toplam = 10 + 8 + 7 + 4 + 4 = 33        ->  33 >= 25  ->  OTESI
```

Demoda `./bin/linux/saldiri_potansiyeli 2 3 2 1 1` komutuyla (sıra: süre uzmanlık bilgi fırsat ekipman) aynı sonucu (`puan=33 -> OTESI`) doğrulayabilirsiniz — kod içindeki gömülü "WBC anahtar cikarma (DCA)" örneğiyle birebir aynı girdilerdir. Bu, ilk bakışta "lisans yamasından çok daha zor ama SE kırma kadar da uç olmayan" bir senaryonun bile, beş faktör birlikte toplandığında hızla **ÖTESİ** bandına çıkabileceğini gösterir; sayılar sezgisel "orta zorlukta" hissinden **daha yüksek** çıkabilir — bu yüzden tahmin değil, hesap gerekir.

CVSS ile derecelendirme

Saldırı potansiyeli "kırmak ne kadar zor?" derken, **CVSS** (Common Vulnerability Scoring System) "bu zafiyetin etkisi ne kadar büyük?" sorusuna standart bir puan verir. İki ölçü birbirini tamamlar: biri **olabilirliği** (saldırının zorluğu), öteki **etkiyi** (gizlilik/bütünlük/erişilebilirlik kaybı) tartar. Bir bulgu raporunda çoğunlukla ikisi birden verilir; önceliklendirme buna göre yapılır.

Aynı bulgu, iki farklı puan: CVSS neden saldırı potansiyelinden farklı çıkar?

Şimdi **aynı** bulguyu (Senaryo B — güvenli öğeden anahtar çıkarma) hem saldırı potansiyeliyle (yukarıda: **55, ÖTESİ**) hem de CVSS ile puanlayalım; ikisinin **neden aynı sayıyı vermediğini** adım adım görelim.

CVSS v3.1 taban puanı, sekiz metrikten hesaplanır (2. haftada tanıtıldı); burada ilgili dördünü kullanıyoruz:

- **Saldırı vektörü (AV):** saldırgan nereden erişiyor? Ağ (N) · Bitişik (A) · Yerel (L) · **Fiziksel (P)**. Senaryo B fiziksel erişim istiyor → **AV:P**.
- **Saldırı karmaşıklığı (AC):** koşullar saldırganın kontrolü dışında mı zorlaşıyor? Düşük (L) · **Yüksek (H)**. Özel ekipman + uzman ekip gerektiren bir donanım saldırısı → **AC:H**.
- **Gerekli yetki (PR) / kullanıcı etkileşimi (UI):** saldırgan önceden bir hesaba ya da kullanıcı eylemine mi ihtiyaç duyuyor? Burada hiçbiri gerekmiyor → **PR:N, UI:N**.
- **Etki (C/I/A):** başarılı olursa sonuç ne kadar kötü? Ana anahtar ele geçerse **bütün** kartların gizliliği, bütünlüğü ve işlem onayının güvenilirliği çöker → **C:H, I:H, A:H** (tam kayıp).

Vektör: **AV:P/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H**

CVSS v3.1 formülü iki alt puanı birleştirir: **Etki** (impact) alt puanı yalnız C/I/A'dan, **Sömürülebilirlik** (exploitability) alt puanı AV/AC/PR/UI'den hesaplanır. Sayıları kendimiz toplayalım:

```
Etki alt puani (Scope: Degismedi, C=H, I=H, A=H -> her biri 0.56):
  ISS    = 1 - [(1-0.56) x (1-0.56) x (1-0.56)]
         = 1 - (0.44 x 0.44 x 0.44)
         = 1 - 0.085184
         = 0.914816
  Impact = 6.42 x ISS = 6.42 x 0.914816 = 5.873   (ara deger)

Somurulebilirlik alt puani (AV:P=0.20, AC:H=0.44, PR:N=0.85, UI:N=0.85):
  Exploitability = 8.22 x 0.20 x 0.44 x 0.85 x 0.85 = 0.523   (ara deger)

Taban puan = yukari_yuvarla( min(5.873 + 0.523, 10) )
           = yukari_yuvarla(6.396)
           = 6.4   ->  "Orta" bandi (NVD: 4.0-6.9)
```

Şimdi aynı etkiyi (C:H/I:H/A:H) ama farklı bir erişim biçimiyle karşılaştıralım — bu yalnız öğretim amaçlı bir karşılaştırma, GüvenPay ürününde böyle bir açık **iddia edilmiyor**. Aynı sonucu ağ üzerinden, düşük karmaşıklıkla elde edebilen **varsayımsal** bir açık olsaydı (**AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H**):

```
Exploitability = 8.22 x 0.85 x 0.77 x 0.85 x 0.85 = 3.887
Taban puan     = yukari_yuvarla( min(5.873 + 3.887, 10) ) = yukari_yuvarla(9.760) = 9.8  -> "Kritik"
```

	Senaryo B (gerçek, fiziksel)	Varsayımsal (yalnız karşılaştırma)
CVSS taban puanı	**6.4** (Orta)	**9.8** (Kritik)
Saldırı potansiyeli	**55** (ÖTESİ)	—

Neden farklı çıkıyor? Çünkü iki ölçü **farklı soruları** farklı **çözünürlüklerle** yanıtlıyor:

- CVSS'in AV/AC metrikleri **kaba, kategorik** seçeneklerdir (N/A/L/P; L/H) — "fiziksel erişim ve yüksek karmaşıklık" demek, saldırının bir hafta mı yoksa altı ay mı süreceğini, bir kişi mi yoksa beş uzmanlık dalından bir ekip mi gerektiğini **ayırt etmez**.
- Saldırı potansiyeli ise süreyi, uzmanlığı, bilgiyi, fırsatı ve ekipmanı **ayrı ayrı, ince taneli** puanlar (0'dan 19'a kadar) toplar — tam olarak "altı ay + beş uzman + özel iyon ışını" ile "bir gün + tek kullanıcı + ücretsiz araç" arasındaki farkı yakalamak **için tasarlanmıştır**.
- CVSS **etkiyi** (bu açık gerçekleşirse ne olur) ölçer; saldırı potansiyeli **maliyeti** (bu açığı gerçekleştirmek ne kadara mal olur) ölçer. İkisi de "yüksek" çıkabilir (Senaryo B'de olduğu gibi: etki felaket ama maliyet çok yüksek), ya da ikisi de "düşük" çıkabilir (lisans yaması örneğinde olduğu gibi).

✓ Kural: ikisini birlikte okuyun, birini diğerinin yerine koymayın

Bir raporda yalnız CVSS 9.8 görüp "acil, hemen kapatılmalı" demek, saldırı potansiyelinin **55/ÖTESİ** olduğunu (gerçekte çok yüksek maliyetli bir saldırı gerektirdiğini) gözden kaçırabilir — kaynaklar yanlış önceliklendirilir. Tersi de olur: düşük CVSS'li ama saldırı potansiyeli TEMEL olan (kolay, düşük etkili ama **çok sık** sömürülebilir) bir bulgu, toplamda daha çok zarar verebilir. **Kural:** önceliklendirme ikisine **birlikte** bakar; hiçbir sertifikasyon şeması tek bir sayıya indirgemez.

Sık yapılan hata: CVSS ile saldırı potansiyelini karıştırmak

"CVSS 9.8, çok tehlikeli" ile "saldırı potansiyeli TEMEL, çok kolay" cümleleri **aynı şey değildir**. Bir öğrenci S16'da yalnız CVSS yazıp saldırı potansiyelini atlarsa (ya da tam tersi), raporun okuyucusu **maliyet** ya da **etki** boyutlarından birini hiç göremez. **Kural:** her ciddi bulgu için ikisi de yazılır; farklı çıkmaları hata değil, **beklenen** bir durumdur (farklı soruları yanıtlıyorlar).

Zincirleme saldırılar: birden fazla zafiyetin birleşimi

Bölüm 2'nin sonunda "iyi tasarlanmış bir üründe saldırgan birkaç korumayı **zincirlemek** zorunda kalır" demiştik; bunu saldırı potansiyeli diliyle açalım. Tek başına **çok yüksek** dirençli (ör. ÖTESİ) bir koruma bile, **başka** bir zafiyetle birleşince toplam direnç düşebilir. Örnek: Senaryo B'deki güvenli öğe (ÖTESİ, puan 55) tek başına neredeyse sömürülemez; ama eğer ayrıca cihazın işletim sistemi eski bir sürümdeyse ve bilinen bir kök (root) açığı varsa, saldırgan önce o **daha kolay** açıkla cihaza tam erişim kazanıp, sonra donanım saldırısı için gereken "çok kısıtlı" fırsatı "sınırsız"a çevirebilir — bu da fırsat faktörünü 10'dan 0'a düşürür ve toplam puanı **55'ten 45'e** indirir (hâlâ ÖTESİ, ama daha kolay bir ÖTESİ).

Sık yapılan hata: her bulguyu tek başına, izole değerlendirmek

Bir zafiyeti yalnız kendi başına puanlamak, başka bir zafiyetle **birleştiğinde** ne olacağını gözden kaçırır. **Kural:** zafiyet analizi adımında (adım 6) değerlendirici yalnız tek tek zafiyetlere değil, olası **zincirlere** de bakar; bir bulgu raporu bazen "F-04 ile F-09 birlikte kullanılırsa..." gibi bir **birleşik senaryo** da içerir.

6. Bulgu → öneri → aksiyon ve etki analizi

Değerlendirme bir "geçti/kaldı" damgası değil, bir **iyileştirme döngüsüdür**. Her bulgu için:

1. **Bulgu:** laboratuvar ne buldu (kanıtla, tekrarlanabilir biçimde).
2. **Öneri:** laboratuvarın önerdiği düzeltme yönü.
3. **Aksiyon:** geliştiricinin ne yaptığı (ya da neden yapmadığı — gerekçeli kabul de bir aksiyondur).
4. **Kapanış:** bazı bulgular "güvenlik açığı değil, iyi uygulama önerisi" olarak da kapanabilir.

Bu döngü, vize sonrası projenizde beklediğimiz **bulgu–aksiyon listesinin** (7. hafta) kaynağıdır.

Güvenlik etki analizi ve delta değerlendirme

Düzeltmeler yeni bir sürüm doğurduğunda, her şeyi baştan değerlendirmek pahalıdır. Bu yüzden iki belge devreye girer (süreçteki 10. ve 11. adımlar):

- **Güvenlik etki analizi (security impact analysis):** Yeni sürümdeki değişiklikleri sınıflandırır (yeni özellik / iyileştirme / hata düzeltmesi), etkilenen dosyaları listeler ve her değişikliğin **güvenlik etkisini** yazar.
- **Delta değerlendirme:** Laboratuvar, bu belgeye dayanarak **yalnız değişen kısmı** yeniden değerlendirir; işaretli belgeleri, değişiklik dizinini, dosya listesini ve **yeni TOE kimliğini** ister.

> ⚠ **Karıştırılan iki kavram**
>
> **Etki analizi** değişikliğin güvenlik etkisini *belgeler*; **delta değerlendirme** bu belgeye dayanıp *yalnız değişeni yeniden değerlendirir*. Biri geliştiricinin çıktısı, öteki laboratuvarın işidir. Sürüm kimliği ve özet değerleri tutarlı olmazsa delta değerlendirme yapılamaz (değerlendirilen ürün belirsizleşir).

Uçtan uca örnek: bir bulgunun doğuşundan kapanışına

Bölüm 2'deki GüvenPay örneğinde geçen "F-03: sabit tuzla anahtar türetme" bulgusunu, dört adımın **her birinde** somut olarak izleyelim — bu, projenizin bulgu–aksiyon tablosunun nasıl doldurulacağının tam örneğidir.

1. **Bulgu (laboratuvar, 12 Kasım):** "`anahtar_yonetimi.c` satır 22'de anahtar türetme fonksiyonu sabit bir dize (`\"guvenpay_tuz_2024\"`) kullanıyor. Kanıt: iki farklı test cihazında aynı girdiyle çalıştırıldığında **aynı** 64 baytlık anahtar üretildi (ekran çıktısı ekte). Saldırı potansiyeli: süre < 1 gün (kaynak kodda görülüyor), uzmanlık sıradan, bilgi kamuya açık, fırsat sınırsız, ekipman ücretsiz → **toplam 0, TEMEL** (bölüm 5'teki puan tablosuyla)."

2. **Öneri (laboratuvar, aynı raporda):** "Her cihaz için benzersiz, CSPRNG'den (hafta 3) üretilmiş bir tuz kullanın; tuzu cihaza özgü güvenli depoda saklayın; HKDF ile anahtar türetin."
3. **Aksiyon (GüvenPay, 15 Kasım):** "Kabul edildi. `anahtar_yonetimi.c` içinde işletim sisteminin CSPRNG'i (hafta 3) ile 32 baytlık rastgele tuz üretiliyor, cihazın güvenli deposuna yazılıyor. Değişiklik v3.2.1'e girdi."
4. **Kapanış (laboratuvar, delta değerlendirme sonrası, 22 Kasım):** "Doğrulandı: iki cihazdan artık **farklı** anahtar üretiliyor. Bulgu **kapandı**."

Karşılaştırma için, "güvenlik açığı değil, iyi uygulama önerisi" olarak kapanan bir örnek de görelim: "Bulgu F-07: günlük mesajları yalnız İngilizce, Türkçe hata mesajı yok." GüvenPay yanıtı: "Bu bir güvenlik açığı değil, günlük mesajları hiçbir hassas veri içermiyor (kanıt: F-03 incelemesinde günlük çıktısı da kontrol edildi) — kullanılabilirlik önerisi olarak not edildi, güvenlik bulgusu listesinden çıkarıldı." Laboratuvar bu gerekçeyi kabul eder ve bulguyu **"etkilenmez"** olarak kapatır.

Sık yapılan hata: 'kabul ediyoruz' deyip hiçbir şey değiştirmemek

Bir bulguyu "kabul edildi" diye işaretleyip kaynak kodda **hiçbir değişiklik yapmamak**, delta değerlendirmede hemen ortaya çıkar (adım 11) — laboratuvar aynı testi tekrar çalıştırır ve bulgunun hâlâ **açık** olduğunu görür. **Kural:** "kabul edildi" bir aksiyon taahhüdüdür; aksiyon sütunu her zaman **ne yapıldığını**, "yapılacak" değil **yapılmış** zamanla yazılır (S16 finalde "sonuç" ister, bölüm 9).

Uçtan uca örnek: bir güvenlik etki analizi belgesi

Adım 10'daki etki analizini tam olarak GüvenPay'in v3.2.0 → v3.2.1 geçişi için yazalım (bu, projenizde vize sonrası bir düzeltme yaptığınızda dolduracağınız belgenin küçültülmüş hâlidir):

Alan	İçerik
Önceki sürüm / yeni sürüm	3.2.0 (özet: `a1b2...`) → 3.2.1 (özet: `f9e8...`)
Değişiklik türü	Hata düzeltmesi (güvenlik)
Etkilenen dosyalar	`anahtar_yonetimi.c` (değişti); `kart_deposu.c`, ağ katmanı (değişmedi)
Değişikliğin özeti	Sabit tuz yerine cihaz başına CSPRNG tuzu + güvenli depoda saklama
Güvenlik etkisi	F-03 bulgusunu kapatır; başka bir varlığı **olumsuz etkilemez** (geriye dönük uyumluluk ayrı bir geçiş betiğiyle test edildi)
Delta değerlendirme gerekli mi?	Evet — yalnız `anahtar_yonetimi.c` ve F-03 testinin tekrarı

Kural: etki analizi 'ne değişti' değil, 'güvenliği nasıl etkiledi' sorusuna cevap verir

Yalnız "dosya X değişti" yazmak yetmez; **hangi varlığı, hangi yönde** etkilediği (iyileştirdi mi, yeni risk mi getirdi mi, nötr mü) açıkça yazılmalıdır. Bir değişiklik bazen **yeni bir zafiyet** de getirebilir — etki analizinin görevi tam olarak bunu erken yakalamaktır.

Kalan risk: her şey kapanmaz, kapanmayan açıkça yazılır

Bölüm 0'da tanımladığımız **kalan risk** kavramını burada somutlaştıralım. GüvenPay'in F-08 bulgusunu (anahtar rotasyonu yılda bir kez) düşünelim: laboratuvar bunun daha sık yapılmasını önerir, ama GüvenPay performans ve kullanıcı deneyimi maliyetini gerekçe göstererek riski **kabul eder**. Rapor bunu şöyle yazar: "F-08: anahtar rotasyon sıklığı yılda bir. Saldırı potansiyeli: YÜKSEK (17) — kolay değil ama imkânsız da değil. GüvenPay bu riski, performans maliyeti gerekçesiyle **kabul etmiştir**; kalan risk laboratuvar tarafından **kayıt altına alınmıştır**, kapatılmamıştır."

Sık yapılan hata: kalan riski raporda hiç yazmamak

Kapatılmayan her bulguyu sessizce atlamak, okuyucuya "her şey kapandı" izlenimi verir — bu **yanıltıcıdır**. **Kural:** kapatılmayan her bulgu, gerekçesiyle birlikte **kalan risk** listesine yazılır; final projenizde (bölüm 9, madde 5) bu liste açıkça istenir.

F-03'ün tüm yaşam döngüsü: bir zaman çizelgesinde

Bu haftanın birçok bölümünde F-03 bulgusunu farklı açılardan gördük — bölüm 2'de nasıl bulunduğunu, bu bölümde bulgu-aksiyon döngüsünü, bölüm 7'de test kartlarını (PT-03/PT-07). Hepsini tek bir zaman çizelgesinde toplayalım; bu, bir bulgunun **baştan sona** nasıl izlendiğinin tam resmidir:

Tarih	Adım	Ne oldu	Belge/kanıt
~5 Kasım	Adım 5 (kod incelemesi)	Sabit tuz fark edildi, bulgu adayı olarak not edildi	Değerlendirici notu
~8 Kasım	Adım 6 (zafiyet analizi)	Sabit tuz, sızma testi planına PT-03 olarak eklendi	Sızma testi planı
10–14 Kasım	Adım 7 (sızma testi)	PT-03 çalıştırıldı, iki cihazdan aynı anahtar gözlendi	PT-03 test kartı (KALDI)
12 Kasım	Adım 9 (bulgular)	Bulgu resmen raporlandı: F-03	Bulgu raporu
15 Kasım	Bulgu-aksiyon (bölüm 6)	GüvenPay düzeltmeyi v3.2.1'e ekledi	Kaynak kod commit'i
~18 Kasım	Adım 10 (etki analizi)	GüvenPay etki analizi belgesini yazdı	Etki analizi belgesi
22 Kasım	Adım 11 (delta değerlendirme)	PT-07 çalıştırıldı, farklı anahtarlar doğrulandı	PT-07 test kartı (GEÇTİ)
22 Kasım	Kapanış (bölüm 6)	F-03 resmen kapandı	Bulgu raporu güncellemesi

Kural: bir bulgunun izlenebilirliği, tek bir belgede değil zincirde yaşar

Dikkat edin: F-03'ün hikâyesi tek bir dosyada değil, **birbirine referans veren** birden fazla belgede (sızma testi planı, test kartları, bulgu raporu, etki analizi belgesi, kaynak kod geçmişi) yaşıyor. Bu izlenebilirlik zinciri kopmadan tutulursa, bir yıl sonra bile "bu bulgu neden, nasıl kapandı?" sorusu saniyeler içinde yanıtlanabilir — 13. haftadaki izlenebilirlik/uyum matrisi tam olarak bu disiplinin genelleştirilmiş hâlidir.

Delta değerlendirmede kapsam kayması (scope creep)

Bir düzeltme yaparken sık karşılaşılan bir tuzak: geliştirici "zaten oradaydım" diyerek F-03'ü düzeltirken ilgisiz başka küçük değişiklikler de yapar (ör. bir arayüz metnini günceller, kullanılmayan bir fonksiyonu siler). Etki analizi belgesi yalnız F-03'ü anlatırsa, laboratuvar delta değerlendirmede **beklenmedik** bir dosya değişikliği görür.

Sık yapılan hata: 'küçük, ilgisiz' değişiklikleri etki analizine yazmamak

"Zaten o dosyadaydım, küçük bir şey daha değiştirdim" diye düşünülen her satır, etki analizi belgesinde **yer almalıdır** — küçüklüğüne siz karar veremezsiniz, laboratuvar karar verir. **Kural:** etki analizi, gerçekte **değişen her dosyayı** listeler; "önemsiz" diye atlanan bir satır, delta değerlendirmenin kapsamını daraltıp gözden kaçan bir zafiyete kapı açabilir.

Ne zaman bir düzeltme yeni bir zafiyet getirir?

Bölüm 10'daki (adım) etki analizi belgesi hep "iyileştirdi" örnekleriyle anlatılır; ama gerçekte bir düzeltme **yeni bir sorun** da getirebilir. Kurgusal bir örnek: GüvenPay, F-03'ü kapatırken tuzu güvenli depoya yazarken bir hata ayıklama derlemesinde yanlışlıkla tuzu da **günlüğe** yazdıran bir satır bırakır. Delta değerlendirme sırasında laboratuvar yalnız "tuz artık benzersiz mi?" sorusuna değil, `anahtar_yonetimi.c` dosyasının **tamamına** bakar ve bu yeni günlük satırını fark eder — yeni bir bulgu (F-11) olarak kaydeder.

Sık yapılan hata: bir düzeltmenin yalnız 'iyi' olduğunu varsaymak

Bir değişikliğin **yalnızca** eski bir bulguyu kapattığını varsaymak tehlikelidir; her değişiklik, değiştirdiği kodun **çevresinde** yeni bir davranışa da yol açabilir. **Kural:** delta değerlendirme, yalnız "eski bulgu kapandı mı?" değil, "değişen dosyada **yeni** bir sorun var mı?" sorusunu da sorar — bu yüzden delta değerlendirme yalnızca eski testi tekrarlamaz, değişen dosyanın **tamamını** yeniden inceler.

7. Sızma testi planı

Sızma testi, yukarıdaki yöntemleri saldırgan bakışıyla birleştiren, **planlı** ve **izinli** bir etkinliktir. "Planlı" ve "izinli" sözcükleri isteğe bağlı değildir: kapsamı ve kuralları yazılı olmayan bir test, meşru bir güvenlik çalışması değildir. Bir plan en az şu dört başlığı içerir.

Durdurma koşulu, gerçek zarar/veri kaybı riskini sınırlar — etik ve güvenli yürütme.

1. Kapsam (scope)

Neyin test edileceği ve neyin **edilmeyeceği** açıkça yazılır: hangi ikili dosya/sürüm, hangi bileşenler, hangi ortam (test ortamı mı, üretim mi), hangi veriler (yalnız sentetik). Kapsam dışı bırakılanlar da yazılır (ör. üçüncü taraf sunucular, gerçek kullanıcı verisi).

GüvenPay kapsam maddesi (örnek metin)

"Kapsam: GüvenPay Cüzdan SDK v3.2.1, Android ARM64 derlemesi, test ortamı (üretim değil), yalnız sentetik kart verisi. **Kapsam dışı:** işletim sistemi çekirdeği, üçüncü taraf ödeme ağ geçidi, gerçek kullanıcı hesapları."

2. Angajman kuralları (rules of engagement)

- **Yazılı izin** ve yetkili imzası; test penceresi (tarih/saat).
- İzin verilen ve **yasak** yöntemler (ör. hizmet kesintisine yol açacak testler hariç).
- Veri işleme kuralı: gerçek kişisel veri kullanılmaz; bulgular gizli tutulur.
- Durdurma koşulu (kill switch): kritik bir etki görülürse test durur ve derhal bildirilir.
- İletişim ve yükseltme (escalation) noktaları.

GüvenPay angajman kuralı maddesi (örnek metin)

"Test penceresi: 10–14 Kasım, 09:00–18:00. Yasak yöntem: üretim sunucusuna karşı hizmet reddi (DoS) denemesi. Durdurma koşulu: gerçek kullanıcı verisine rastlanırsa test **anında** durur ve GüvenPay güvenlik ekibine bildirilir. İletişim noktası: güvenlik ekibi sorumlusu."

3. Yöntem (methodology)

Tanınmış bir metodoloji seçilir ve ona uyulur: mobil için **OWASP MASTG/MASVS**, web için **OWASP WSTG**, genel süreç için **PTES** ya da **NIST SP 800-115**. Yöntem seçimi, sonucun **tekrarlanabilir** ve **karşılaştırılabilir** olmasını sağlar.

GüvenPay yöntem maddesi (örnek metin)

"Yöntem: OWASP MASTG'nin depolama (storage) ve kriptografi (cryptography) test kategorileri temel alınmıştır."

4. Test şablonu (sekiz başlık)

Değerlendirmedeki her test aynı şablonla yazılır; bu, projenizin S16 bölümünün de iskeletidir:

Alan	İçerik
Test kimliği	Benzersiz numara
İlgili gereksinim	Hangi gereksinimi/varlığı sınıyor (S17 ile bağ)
Amaç	Ne doğrulanıyor
Ön koşullar	Ortam, sürüm, veriler
Yöntem/adımlar	Tekrarlanabilir biçimde
Beklenen sonuç	Güvenli davranış ne olmalı
Gözlenen sonuç	Ne oldu (kanıtla)
Saldırı potansiyeli / karar	Puanlama + geçti/kaldı + öneri

Sekiz alan da rastgele seçilmemiştir; her biri, testin **tekrarlanabilir** ve **tartışılabilir** olması için gerekli bir soruyu yanıtlar:

- **Test kimliği:** bir bulguyu (bölüm 6) ya da rapor satırını (bölüm 8) hangi teste bağlayacağınızı bilmeniz gerekir — kimliksiz bir test, kanıtı olmayan bir iddiadır.
- **İlgili gereksinim:** bir test, hiçbir gereksinime bağlı değilse "neden bu testi yapıyoruz?" sorusuna cevap veremez; bu alan S17 uyum matrisine giden **köprüdür**.
- **Amaç:** "ne doğrulanıyor" cümlesi olmadan, bir okuyucu testin **neyi kanıtlamaya çalıştığını** anlayamaz.
- **Ön koşullar:** hangi sürüm, hangi ortam, hangi veriyle test edildiği yazılmazsa, aynı testi bir başkası **tekrar edemez** (delta değerlendirmede bu hayati önemdedir).
- **Yöntem/adımlar:** adım adım yazılmamış bir test, "nasıl yaptım hatırlamıyorum" riskini taşır; tekrarlanabilirlik burada da kilit kelimedir.
- **Beklenen sonuç:** önceden yazılmazsa, testi çalıştıran kişi sonucu **istediği gibi yorumlama** eğilimine girebilir (bilişsel önyargı, bölüm 1'deki "geliştirici kendi ürününü onaylayamaz" mantığının küçük ölçekli hâli).
- **Gözlenen sonuç:** kanıtsız bir "çalıştı" cümlesi, bölüm 0'da tanımladığımız anlamda bir **bulgu bile değildir**.
- **Saldırı potansiyeli/karar:** testin sonucunu bir **önceliğe** (bölüm 8'deki öncelik matrisi) çevirir.

Takım etkinliği (sınıf içi)

Her takım, kendi projesinden bir varlık seçer (ör. yerel veritabanındaki hassas kayıt) ve onun için **bir** test kartı doldurur: amaç, ön koşul, adımlar, beklenen/gözlenen sonuç, saldırı potansiyeli. Amaç bir açık bulmak değil, **doğru test kartını yazmayı** öğrenmektir.

Doldurulmuş bir test kartı örneği

Aşağıda, sekiz başlıklı şablonun **gerçekten doldurulmuş** hâlini görüyoruz — hafta 3'teki AES-GCM/AEAD demosuyla doğrudan bağlantılı bir test:

Alan	İçerik
Test kimliği	**PT-07**
İlgili gereksinim	S5 / R-12: "Yerel veritabanındaki kart alanları AEAD ile şifrelenir" (hafta 3, S17'ye bağlı)
Amaç	Root edilmiş bir cihazda, yerel veritabanı dosyasının doğrudan okunmasıyla kart verisinin açık metin olarak ele geçirilip geçirilemediğini; kurcalanmış bir kaydın **reddedilip reddedilmediğini** doğrulamak
Ön koşullar	Test cihazı root edilmiş; GüvenPay v3.2.1 kurulu; veritabanında yalnız **sentetik** test kartı verisi (gerçek PAN yok, bu haftanın etik kuralı)
Yöntem/adımlar	(1) Uygulamanın veri dizinindeki `.db` dosyası cihazdan çekilir. (2) Dosya bir hex düzenleyicide incelenir, 16 haneli PAN deseni aranır. (3) Bulunamazsa, şifreli bir kayıttaki etiketin (tag) son baytı değiştirilir. (4) Uygulama yeniden başlatılır, davranış gözlenir.
Beklenen sonuç	Dosyada hiçbir alan açık metin görünmemeli (AEAD şifreli); kurcalanan kayıt açılışta **reddedilmeli** (fail-closed, hafta 3 kuralı)
Gözlenen sonuç	Dosya incelendi: tüm hassas alanlar rastgele görünen bayt dizileri (şifreli), desen eşleşmesi **yok**. Kurcalanan kaydın etiketi değiştirildiğinde uygulama "bütünlük doğrulama başarısız" günlüğü yazdı ve kaydı **sildi**; çökmedi, sessizce yanlış veri de döndürmedi.
Saldırı potansiyeli / karar	Doğrudan dosya okuması başarısız olduğundan saldırgan anahtar çıkarmaya yönelmek zorunda kalır — bu, bölüm 5'teki **Senaryo B** ile aynı sınıfa girer (fiziksel + özel ekipman, puan 55, ÖTESİ). **Karar: GEÇTİ.**

Bu tablo neden S16'nın iskeleti?

Dikkat edin: "gözlenen sonuç" satırı **kanıtla** (günlük çıktısı, dosya inceleme sonucu) yazılmış, "muhtemelen çalışıyor" gibi bir varsayım içermiyor. Finalde S16'dan beklenen tam olarak budur (bölüm 9) — **plan değil, sonuç.**

Bir de KALDI (başarısız) örneği görelim

PT-07 **geçti**; ama bir test kartının **kaldığı** durumu da görmeden bölüm eksik kalır — çünkü bir öğrencinin en sık yaptığı hata, yalnız geçen testleri göstermektir. Aynı GüvenPay'in ilk sürümünde (v3.2.0, F-03 düzeltilmeden önce) aynı test şöyle sonuçlanırdı:

Alan	İçerik
Test kimliği	**PT-03** (F-03'ün kaynağı)
İlgili gereksinim	R-11: "Her cihazda anahtar türetme benzersiz olmalı"
Amaç	İki farklı cihazda aynı girdiyle anahtar türetmenin **farklı** anahtarlar üretip üretmediğini doğrulamak
Ön koşullar	İki test cihazı, GüvenPay v3.2.0, aynı sentetik kullanıcı verisi
Yöntem/adımlar	(1) Her iki cihazda da uygulama sıfırdan kurulur. (2) Aynı kullanıcı verisiyle anahtar türetme tetiklenir. (3) Üretilen anahtarlar dışa aktarılıp karşılaştırılır.
Beklenen sonuç	İki cihazdan **farklı** anahtar üretilmeli (cihaza özgü rastgele tuz sayesinde)
Gözlenen sonuç	İki cihazdan da **aynı** 64 baytlık anahtar üretildi (ekran çıktısı ekte, bayt bayt eşleşme). Kaynak kod incelendiğinde tuzun sabit bir dize olduğu görüldü.
Saldırı potansiyeli / karar	Süre < 1 gün, uzmanlık sıradan, bilgi kamuya açık, fırsat sınırsız, ekipman ücretsiz → **toplam 0, TEMEL**. **Karar: KALDI.** Bu, F-03 bulgusunun doğduğu testtir (bölüm 6).

Kural: KALDI bir başarısızlık değil, sürecin çalıştığının kanıtıdır

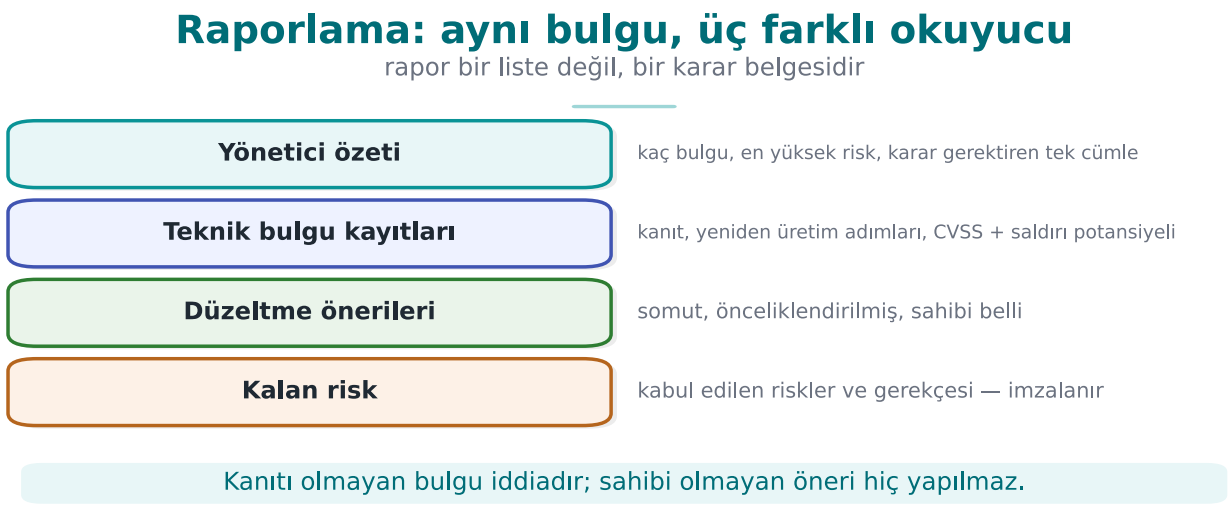
Bir test planında **hiç** KALDI görmemek, genelde "hiçbir zafiyet yok" değil, "yeterince derin bakılmadı" anlamına gelir. İyi bir S16, en az birkaç KALDI içerip, her birinin nasıl **kapandığını** (bölüm 6'daki bulgu-aksiyon döngüsüyle) göstermelidir — bu tam olarak PT-03 → F-03 → PT-07 zincirinde gördüğümüz şeydir.

Sahada nasıl uygulanır? Yazılı izin neden bu kadar katı?

Angajman kurallarındaki "yazılı izin" maddesi süs değildir: yetkisiz bir sisteme erişim denemesi, niyet ne olursa olsun (ders projesi, "iyi niyetli" araştırma), birçok ülkede **suç** sayılır. Bu yüzden gerçek laboratuvarlar teste başlamadan önce imzalı bir sözleşme olmadan **tek bir komut bile** çalıştırmaz. Ders projenizde bu riski tamamen ortadan kaldırmak için sızma testi çalışmaları yalnız **kendi** projeniz üzerinde, kağıt üstünde planlanır (yukarıdaki etik ve hukuki çerçeve kutusu).

8. Raporlama

İyi bir güvenlik raporu, bir "kaldınız" listesi değil, **karar verilebilir** bir belgedir:



- **Yönetici özeti:** teknik olmayan okur için genel durum ve en kritik üç bulgu. Okuyucu genellikle bir yönetici ya da müşteridir; "kaç bulgu var" değil, "üretime hazır mıyız" sorusuna **doğrudan** cevap verir.
- **Yöntem ve kapsam:** ne test edildi, ne edilmedi, hangi metodolojiyle. Kapsam dışı bırakılanların **neden** dışarıda kaldığı da yazılır — aksi hâlde okuyucu "test edilmedi mi, yoksa unutuldu mu" ayrımını yapamaz.
- **Bulgular:** her biri kanıt, saldırı potansiyeli ve CVSS, öneri ile. Bulgular önem sırasına göre (en ciddi önce) listelenir; her biri tek başına okunduğunda bile anlaşılır olmalıdır.
- **Bulgu–aksiyon tablosu:** geliştiricinin yanıtı ve kapanış durumu. Bu, bölüm 6'daki dört adımlı döngünün (bulgu → öneri → aksiyon → kapanış) rapordaki karşılığıdır.
- **Kalan risk:** kapatılmayan ya da devredilen riskler, gerekçesiyle. Bu bölüm **atlanamaz**: bir raporun en çok güven kazandıran kısmı, aslında "her şeyi çözdük" değil, "şunu çözemedik, işte nedeni" cümlesidir.

Eleştirel okuma alıştırması

Size verilen kısa bir bulgu metnini birlikte okuyun: bu bulgunun saldırı potansiyeli doğru mu derecelendirilmiş? Öneri uygulanabilir mi? "Karşılandı" diyen bir satırın kanıtı var mı? Bu okuma, 13. haftadaki uyum matrisi okumasının provasıdır.

Bir rapor, birden çok okuyucuya hizmet eder

Yukarıdaki beş bölümün neden hepsinin gerekli olduğunu anlamak için, aynı raporu **kim okuyacak** sorusunu soralım — her bölüm aslında farklı bir okuyucu için yazılır:

Bölüm	Kim okur?	Ne arıyor?
Yönetici özeti	Yönetici, müşteri, banka	"Üretime hazır mı, evet/hayır?"
Yöntem ve kapsam	Bir sonraki değerlendirici, denetçi	"Bu test **tekrarlanabilir** mi, neyi kapsamamış?"
Bulgular	Geliştirme ekibi lideri	"Hangi sırayla düzeltmeliyiz?"
Bulgu–aksiyon tablosu	Proje yöneticisi, gelecekteki bakım ekibi	"Bu sorun daha önce görülmüş mü, ne yapılmış?"
Kalan risk	Üst yönetim, sigorta/hukuk	"Kabul ettiğimiz risk tam olarak ne, kim onayladı?"

Sahada nasıl uygulanır?

Bu yüzden gerçek raporlar genellikle **tek bir üslupla** yazılmaz: yönetici özeti sade, jargonsuz bir dille; bulgular bölümü teknik terimlerle (CWE kimliği, saldırı potansiyeli puanı) yazılır. Bir raporu tek bir okuyucuya göre (yalnız yöneticiye ya da yalnız mühendise) yazmak, diğer okuyucuların ihtiyacını karşılamaz.

Örnek: bir yönetici özeti paragrafı

GüvenPay değerlendirme raporu — yönetici özeti (örnek)

"GüvenPay Cüzdan SDK v3.2.1, OWASP MASVS L2 hedefiyle değerlendirilmiştir. Toplam 9 bulgudan 7'si kapatılmış, 1'i düşük öncelikli olarak kabul edilmiş, 1'i (F-03, sabit tuz) düzeltilip **doğrulanmıştır**. En kritik üç bulgu: (1) F-03 — anahtar türetmede sabit tuz [kapandı], (2) F-05 — hata ayıklama günlüklerinin sürüm derlemesinde aktif kalması [kapandı], (3) F-08 — anahtar rotasyon sıklığının düşük olması [kalan risk, gerekçeli kabul]. Ürün, üretime **hazır** değerlendirilmiştir."

Örnek: bir bulgu satırının rapor biçimi

Bulgular tablosunda her satır aynı disiplinle yazılır — kanıt, saldırı potansiyeli/CVSS, öneri birlikte:

Bulgu no	Açıklama	Kanıt	Saldırı pot.	CVSS*	Öneri	Durum
F-03	Sabit tuzla anahtar türetme	2 cihazdan aynı anahtar (ek A)	0 (TEMEL)	5.9 (Orta)	CSPRNG tuzu + HKDF	Kapandı (PT-03 ile doğrulandı)
F-08	Anahtar rotasyonu yıllık	Kılavuz bölüm 6.1	17 (YÜKSEK)	4.2 (Orta)	Rotasyonu sıklaştırın	Kalan risk (kabul)

*(*Bu örnekteki CVSS değerleri gösterim amaçlıdır; gerçek puanlama bölüm 5'teki AV/AC/PR/UI/C/I/A seçimleriyle ve standart CVSS hesaplayıcısıyla yapılır.)*

> ⚠ **Sık yapılan hata: kanıtsız 'kapandı' yazmak**
>
> Bir bulguyu "kapandı" diye işaretleyip **hangi testin bunu doğruladığını** yazmamak, okuyucunun (ve bir sonraki değerlendiricinin) bu iddiaya güvenmesini imkânsız kılar. **Kural:** her "kapandı" satırı, onu doğrulayan test kimliğine (ör. "PT-07 ile doğrulandı") işaret eder.

Bulguları önceliklendirmek: basit bir öncelik matrisi

Bölüm 5'te CVSS'in **etkiyi**, saldırı potansiyelinin **maliyeti** ölçtüğünü görmüştük. Bir raporda düzinelerce bulgu varsa, hangisine önce bakılacağı bu ikisinin **birleşiminden** çıkar. Aşağıdaki matris, resmî bir standart değil, **bu ders için kullandığımız basit bir öğretim aracıdır** — gerçek şemalarda öncelik kuralları daha ayrıntılıdır, ama fikir aynıdır:

	Saldırı pot. TEMEL/ORTA (kolay)	Saldırı pot. YÜKSEK/ÇOK YÜKSEK	Saldırı pot. ÖTESİ (çok zor)
CVSS Kritik/Yüksek (etki büyük)	🔴 Acil — hemen kapatılmalı	🟠 Yüksek öncelik	🟡 İzlenmeli, acil değil
CVSS Orta (etki sınırlı)	🟠 Yüksek öncelik	🟡 Orta öncelik	🟢 Düşük öncelik
CVSS Düşük (etki küçük)	🟡 Orta öncelik	🟢 Düşük öncelik	🟢 Düşük öncelik

GüvenPay örneğine uygulayalım: **F-03** (CVSS Orta, saldırı potansiyeli TEMEL) tabloda "yüksek öncelik" hücresine düşer — tam olarak bu yüzden ilk kapatılan bulgu odur. **F-08** (CVSS Orta, saldırı potansiyeli YÜKSEK) "orta öncelik" hücresine düşer — kalan risk olarak kabul edilebilir olması bu yüzden makuldür.

> ✓ **Kural: tek eksenli önceliklendirme yanıltır**
>
> Yalnız CVSS'e (ya da yalnız saldırı potansiyeline) bakarak sıralama yapmak, matrisin yalnız bir satırını ya da sütununu okumak demektir. **Kural:** öncelik kararı her zaman **iki eksenin kesişiminden** verilir.

Rapor ne kadar uzun olmalı?

Zayıf bir öğrencinin sık düştüğü iki uç vardır: ya tek sayfalık, kanıtsız bir "her şey güvenli" raporu yazmak, ya da her ayrıntıyı (her SAST uyarısını, doğrulanmamış her aday bulguyu) art arda dökmek. İkisi de yanlıştır:

- **Çok kısa rapor:** okuyucu hangi kararın **neye dayandığını** göremez; "güvenli" kelimesi bir kanıt değildir (bölüm 1'deki "iddia değil kanıt" kuralı).
- **Çok uzun, filtrelenmemiş rapor:** doğrulanmamış onlarca SAST uyarısını olduğu gibi dökmek, okuyucunun gerçek üç-beş kritik bulguyu **gürültü içinde kaybetmesine** yol açar (yanlış pozitif tartışmasının, bölüm 3, raporlama düzeyindeki karşılığı).

> ✓ **Kural: uzunluk değil, izlenebilirlik önemlidir**
>
> Doğru ölçü sayfa sayısı değildir: her iddianın bir kanıta, her kanıtın bir test kimliğine bağlı olmasıdır. Bir yönetici özeti yarım sayfa, bulgular bölümü bulgu sayısı kadar uzun olabilir — önemli olan, **hiçbir cümlenin kanıtsız kalmamasıdır**.

9. Dönem projesi: bu hafta (S16 — test planı ve sonuçları)

1. **Test planı:** En kritik iki–üç varlık/gereksinim için sekiz başlıklı test kartları yazın (S17 ile bağlı).
2. **Yöntem:** Hangi metodolojiyi (MASTG/WSTG/PTES/NIST SP 800-115) temel aldığınızı belirtin.
3. **Sonuçlar:** Testleri çalıştırıp **gözlenen sonuçları** yazın — finalde plan değil, **sonuç** beklenir.
4. **Bulgu–aksiyon:** Vize geri bildirimlerini bir bulgu–aksiyon tablosuna dökün.
5. **Kalan risk:** Kapatmadığınız riskleri ve gerekçesini yazın.

Bu madde 4 neden özellikle önemli: vize geri bildirimleriniz zaten birer "bulgu"

Öğretim üyesinin vize sunumunuzda verdiği her geri bildirim, aslında bu haftanın diliyle bir **bulgu**dur: bir zayıflık, kanıtla (sunumdaki gözlem) işaretlenmiş bir gözlem. Bu haftanın 4 adımlı döngüsünü (bölüm 6) doğrudan kendi projenize uygulayın:

1. **Bulgu:** öğretim üyesinin vizede söylediği somut cümle (ör. "girdi doğrulaması eksik").
2. **Öneri:** bu geri bildirimden çıkan somut düzeltme yönü.
3. **Aksiyon:** finale kadar **ne yaptığınız** (kod değişikliği, commit referansı).
4. **Kapanış:** düzeltmeyi kanıtlayan bir test (yeni ya da güncellenmiş bir test kartı).

Bu, GüvenPay örneğindeki F-03 zincirinin sizin projenizdeki karşılığıdır

Bölüm 2, 6 ve 7'de izlediğimiz "PT-03 (KALDI) → F-03 → düzeltme → PT-07 (GEÇTİ)" zinciri kurgusal bir örnekti; finalde beklenen, vize geri bildiriminiz için **gerçek** bir zincir kurmanızdır: hangi geri bildirim, hangi değişiklik, hangi testle doğrulandı?

Etik ve hukuki çerçeve (tekrar)

Sızma testi yalnız **kendi** projeniz ve yetkili olduğunuz sistemler üzerinde yapılır. Başkasının sistemine izinsiz test, amacı ne olursa olsun suçtur. Bütün veriler sentetik olmalı; depoda gerçek sır ya da kişisel veri bulunmamalıdır.

Kendi projenizde S16'yı doldururken kontrol listesi

- [ ] Her test kartında sekiz alanın **hepsi** dolu mu (bölüm 7)?
- [ ] "Gözlenen sonuç" alanı bir **kanıt** (ekran çıktısı, günlük, dosya) içeriyor mu — yoksa yalnız "çalıştı" mı yazıyor?
- [ ] Her ciddi bulgu için saldırı potansiyeli **beş faktörle** hesaplanmış mı (bölüm 5)?
- [ ] Bulgu–aksiyon tablosunda her satırın bir **durumu** (kapandı / kalan risk / etkilenmez) var mı?
- [ ] Kalan riskler **gerekçeleriyle** yazılmış mı, yoksa sessizce mi atlanmış?
- [ ] Kullanılan metodoloji (MASTG/WSTG/PTES/NIST SP 800-115) açıkça belirtilmiş mi?

10. Kendini sınama

1. Bir ürünün bağımsız değerlendirmesindeki 13 adımı ana hatlarıyla sıralayın. Hazırlık, değerlendirme ve süreklilik aşamaları neleri kapsar?

Hazırlık (1–4): değerlendirme hedefini (TOE) tanımlama, kapsam, güvenlik hedefi/gereksinimler, plan. **Değerlendirme:** yöntemleri uygulama (kod inceleme → SAST → DAST → fuzzing → sızma testi), bulguları saldırı potansiyeliyle derecelendirme, raporlama. **Süreklilik:** karar/sertifika, sonra her değişiklikte etki analizi + delta değerlendirme ve sürüm bakımı.

2. Değerlendirici hangi iki varsayımla çalışır? Bu neden 'korumanın var olması yetmez' sonucunu doğurur?

(1) Saldırgan sistemi **tam bilir** (Kerckhoffs), (2) saldırgan **yetenekli ve kaynaklıdır**. Bu yüzden bir korumanın sadece bulunması değil; belirli bir **saldırı potansiyeline** dayanması ve bunun **kanıtla** gösterilmesi gerekir.

3. Zafiyet değerlendirmesinin yöntemlerini (kod incelemesi, SAST, DAST, fuzzing, sızma testi) neden bu sırada uygularız?

Ucuz/otomatik/geniş kapsamlı yöntemlerden pahalı/el emeği/derin yöntemlere doğru. Erken ucuz taramalar kolay bulguları eler; pahalı el emeği (fuzzing, sızma testi) yalnız kalan derin sorunlara harcanır → bütçe ve zaman verimli kullanılır.

4. ISO/IEC 27001 ile Ortak Kriterler'in test beklentisi nasıl ayrılır? Biri neyi, öteki neyi sertifikalar?

ISO/IEC 27001 **kurumun bilgi güvenliği yönetim sistemini** (süreç/organizasyon) denetler ve sertifikalar. Ortak Kriterler (ISO/IEC 15408) **belirli bir ürünü (TOE)** tanımlı bir güvence düzeyinde teknik olarak değerlendirir ve sertifikalar.

5. Saldırı potansiyelinin beş faktörünü sayın. Düşük saldırı potansiyeli neden ciddi bir bulgudur?

Geçen zaman, uzmanlık, ürün hakkında bilgi, fırsat (erişim/pencere), ekipman. Düşük saldırı potansiyeli, az zaman/beceri/araçla sömürülebilir demektir → çok sayıda saldırgan başarabilir → geniş ve olası tehdit, yani yüksek risk.

6. Saldırı potansiyeli ile CVSS neyi ölçer? İkisi neden birlikte verilir?

Saldırı potansiyeli saldırıyı gerçekleştirmenin **zorluğunu/maliyetini**; **CVSS** açığın **ciddiyet/etkisini** ölçer. Birlikte verilince hem 'ne kadar kolay' hem 'ne kadar kötü' görülür ve önceliklendirme sağlıklı yapılır.

7. Bulgu–öneri–aksiyon döngüsünü açıklayın. Bir bulgu 'güvenlik açığı değil' diye nasıl kapanır?

Her bulguya bir **düzeltme önerisi** ve sorumlu/**aksiyon** atanır, sonra düzeltme **doğrulanır**. Bir bulgu, bağlam veya telafi kontrolleri nedeniyle gerçekten **sömürülemez** olduğu kanıtlanırsa 'etkilenmez / risk kabul' gerekçesiyle (VEX benzeri) kapatılır.

8. Güvenlik etki analizi ile delta değerlendirme arasındaki fark nedir? Hangisi geliştiricinin, hangisi laboratuvarın çıktısıdır?

Etki analizi geliştiricinin çıktısıdır: yapılan değişiklik güvenliği/TOE'yi etkiliyor mu? **Delta değerlendirme laboratuvarın** işidir: etkilenen kısmı bağımsız olarak yeniden değerlendirir (tüm ürünü baştan değil).

9. Bir sızma testi planının dört başlığını (kapsam, angajman kuralları, yöntem, test şablonu) ve her birinin amacını yazın.

Kapsam: neyin test edilip edilmeyeceği. **Angajman kuralları:** izinler, sınırlar, zaman, durdurma koşulu. **Yöntem:** metodoloji ve araçlar (WSTG/PTES vb.). **Test şablonu:** tekrarlanabilir test kartları. Amaç: güvenli, etik, tekrarlanabilir ve kapsamı net bir test.

10. Angajman kurallarında 'durdurma koşulu' neden bulunur?

Gerçek zarar, veri kaybı veya hizmet kesintisi riskini sınırlamak için. Belirlenen bir eşikte (örn. üretim etkisi ya da kritik bir açık) test **hemen durdurulur** → güvenli ve etik yürütme sağlanır.

11. Sekiz başlıklı test kartının alanlarını sayın. Hangi alan S17 uyum matrisine bağlanır?

Örn: **kimlik/no, ilgili gereksinim-hedef, ön koşul, adımlar, beklenen sonuç, gözlenen sonuç, karar (geçti/kaldı), kanıt/ek.** 'İlgili gereksinim / karar' alanı **S17 uyum matrisine** kanıt olarak bağlanır.

12. Finalde S16'dan 'plan' değil 'sonuç' beklenmesi ne demektir?

Testin yalnızca nasıl yapılacağının yazılması yetmez; testlerin gerçekten **çalıştırılıp** gözlenen sonuçlarının (geçti/kaldı, kanıt, ekran çıktısı) raporlanması beklenir. Yani plan değil, **gözlenen sonuçlar** teslim edilir.

13. Risk nedir ve zafiyet, bulgu, risk terimleri nasıl birbirinden ayrılır?

Risk, bir zafiyetin sömürülme **olasılığı** ile sömürüldüğünde oluşacak **etkinin** birlikte değerlendirilmesidir. **Zafiyet** teknik bir zayıflıktır (ör. sabit tuz kullanımı); **bulgu** bu zayıflığın değerlendirmede kanıtla belgelenmesidir; **risk** ise "bu zayıflık gerçekte ne kadar tehlikeli" sorusunun yanıtıdır (erişim, ortam, veri değeri gibi bağlamsal etkenlere göre değişir).

14. Yanlış pozitif ile yanlış negatif arasındaki fark nedir? Hangisi daha tehlikelidir?

Yanlış pozitif, aracın "sorun var" dediği ama aslında sorun olmayan durumdur (zaman kaybettirir, alarm yorgunluğuna yol açar). **Yanlış negatif**, aracın **kaçırdığı gerçek** sorundur ve daha tehlikelidir; çünkü "araç temiz dedi" güveni sahte bir güvenlik hissi yaratır ve gerçek açık fark edilmeden kalır.

15. TOE kimliği neden yalnızca bir sürüm numarasından ibaret olamaz?

Aynı sürüm numarasıyla farklı derlemeler (debug/release, farklı derleyici bayrakları) var olabilir ve bunların davranışı farklı olabilir. Bu yüzden TOE kimliği sürüm etiketi + ikili dosya (derleme türü/mimari) + kaynak kod (commit/etiket) + **özet değeri (hash)** olmak üzere dört parçadan oluşur; hash tek bir baytlık farkı bile ortaya çıkarır.

16. CVSS'in AV (saldırı vektörü) ve AC (saldırı karmaşıklığı) metrikleri hangi değerleri alabilir?

AV (Attack Vector): Ağ (Network) · Bitişik (Adjacent) · Yerel (Local) · Fiziksel (Physical). **AC (Attack Complexity):** Düşük (Low) · Yüksek (High). Bunlar kaba, kategorik seçeneklerdir; saldırı potansiyelinin sayısal, ince taneli puanlamasından farklı bir çözünürlükte çalışırlar.

17. Bölüm 5'teki 'güvenli öğe kırma' (Senaryo B) örneğinde beş faktörün puanlarını ve toplamını yazın. Hangi düzeye denk gelir?

Geçen süre (aylar) = 19, uzmanlık (çoklu uzman) = 8, hedef bilgisi (kritik) = 11, fırsat (çok kısıtlı) = 10, ekipman (özel-uyarlanmış) = 7. **Toplam = 19+8+11+10+7 = 55** → 55 ≥ 25 olduğundan **ÖTESİ** düzeyi (yalnız çok yetenekli/kaynaklı bir saldırgan).

18. Aynı bulgu için CVSS taban puanı 6,4 (fiziksel/yüksek karmaşıklık), saldırı potansiyeli 55/ÖTESİ çıkıyor. Bu farkı nasıl açıklarsınız?

CVSS **etkiyi** (bu açık gerçekleşirse ne olur) kaba kategorik metriklerle (AV/AC gibi 2–4 seçenekli) ölçer; saldırı potansiyeli ise **maliyeti** (süre, uzmanlık, bilgi, fırsat, ekipman) ince taneli, toplamsal bir puanlamayla ölçer. İkisi farklı soruları farklı çözünürlükte yanıtladığı için sayılar örtüşmek zorunda değildir; bu bir hata değil, beklenen bir durumdur.

19. Bir sızma testi planında 'kapsam' ile 'angajman kuralları' arasındaki fark nedir?

Kapsam, NEYİN test edileceğini ve edilmeyeceğini (hangi sürüm, ortam, veri) tanımlar. **Angajman kuralları** ise testin NASIL yürütüleceğini sınırlar: izin, zaman penceresi, yasak yöntemler, durdurma koşulu, iletişim noktaları. Biri "neyi" biri "nasıl" sorusuna cevap verir.

20. Bulgu-aksiyon döngüsünde 'kabul edildi' yazıp kaynak kodda hiçbir değişiklik yapmamak neden risklidir?

Delta değerlendirme aşamasında (adım 11) laboratuvar aynı testi tekrar çalıştırır; değişiklik yapılmamışsa bulgu hâlâ **açık** çıkar ve güven kaybına yol açar. Kural: "kabul edildi" bir aksiyon **taahhüdüdür**, aksiyon sütunu yapılacak değil **yapılmış** olanı, geçmiş zamanla ve kanıtla yazılır.

21. Fuzzing sonucunda 'kod kapsamı' (coverage) neden önemlidir?

Fuzzing, zamanla daha fazla kod yoluna ulaşarak (kapsam biriktirerek) çalışır; kısa bir çalıştırma yalnızca yüzeysel yolları dener. Kod kapsamı yazılmadan verilen bir "temiz" sonuç, aracın henüz derin yollara **hiç inmediği** anlamına gelebilir; bu yüzden süre ve ulaşılan kapsam yüzdesi birlikte raporlanır.

22. Delta değerlendirme ne zaman gerekir ve neden tüm ürünü baştan değerlendirmekten farklıdır?

Bir düzeltme/güncelleme yeni bir sürüm doğurduğunda gerekir. Tüm ürünü baştan değerlendirmek yerine, laboratuvar yalnız **değişen kısmı** (etki analizi belgesine, değişen dosyalara ve yeni TOE kimliğine dayanarak) yeniden değerlendirir; bu hem hızlı hem de her sürümde tam bir sertifikasyon döngüsü gerektirmeden **sürekliliği** sağlar.

23. Bir standardın (ör. Ortak Kriterler) test derinliği neye göre değişir?

Hedeflenen varlığın **değerine** ve varsayılan **saldırgan modeline** göre değişir: bir IoT ampulün temel güvenliği ile bir ödeme kartının anahtar hiyerarşisi aynı titizlikte test edilmez, çünkü kırılmalarının sonucu aynı ölçekte değildir. Bu derinlik farkı Ortak Kriterler'de güvence düzeyleriyle (13. hafta) resmîleştirilir.

24. Bir rapor satırında 'F-03 kapandı' yazmak neden tek başına yetersizdir?

Hangi testin bu kapanışı **doğruladığı** (ör. "PT-07 ile doğrulandı") belirtilmezse okuyucu iddiaya güvenemez. Kural: her 'kapandı' kaydı, onu doğrulayan test kimliğine veya kanıta açıkça bağlanmalıdır — aksi hâlde bir sonraki değerlendirici (ya da öğretim üyesi) aynı testi baştan yapmak zorunda kalır.

25. Kod incelemesi ile sızma testi, bir SQL enjeksiyonu hatasını bulma gücü açısından nasıl karşılaştırılır? Fuzzing bu hatada neden zayıftır?

Kod incelemesi ve sızma testi bu tür **mantık** hatalarında güçlüdür (biri bağlamı okuyarak, öteki doğrudan deneyerek bulur). Fuzzing rastgele bayt dizileriyle çalışır; geçerli SQL sözdizimi (`' OR '1'='1` gibi) üretme olasılığı düşük olduğundan bu tür yapılandırılmış mantık hatalarında **zayıftır** — fuzzing bellek/UB hatalarında çok daha güçlüdür.

26. 'Sertifikalı' bir ürün hakkında eksiksiz bir cümle hangi dört bilgiyi içermelidir?

(1) **Hangi ürün/sürüm** (TOE kimliği, hash dahil), (2) **hangi standarda göre** (ör. Ortak Kriterler, EMVCo), (3) **hangi saldırgan modeline karşı** (ör. fiziksel erişimi olmayan orta düzey saldırgan), (4) **hangi tarihte** — çünkü ürün güncellenince sertifika o eski sürüm için geçerliliğini korur, yeni sürüm delta değerlendirmeden geçmelidir.

Bu haftanın terimleri: yan yana karşılaştırma

Bu haftanın en sık karıştırılan beş terim çiftini, sınava girmeden önce son bir kez yan yana koyalım:

Çift	Birincisi	İkincisi	Ayırt edici soru
Zafiyet / Bulgu / Risk	Zafiyet: teknik zayıflık	Bulgu: kanıtla belgelenmiş hâli	Risk: "gerçekte ne kadar tehlikeli?" (olasılık × etki)
SAST / DAST	Kaynağı **çalıştırmadan** analiz	Programı **çalıştırarak** analiz	"Kod mu okunuyor, davranış mı gözleniyor?"
Saldırı potansiyeli / CVSS	Saldırının **maliyeti/zorluğu**	Açığın **etkisi/ciddiyeti**	"Ne kadar zor?" mu, "gerçekleşirse ne kadar kötü?" mü?
Etki analizi / Delta değerlendirme	Geliştiricinin **belgesi**	Laboratuvarın **yeniden testi**	"Kim yazdı, kim doğruladı?"
Kapsam / Angajman kuralları	**Neyin** test edildiği	**Nasıl** test edildiği	"Ne test ediliyor?" mu, "hangi sınırlar içinde?" mü?

Sınav öncesi son kontrol

Bu tablodaki beş satırı kapatıp yalnız "Ayırt edici soru" sütununa bakarak her çiftin iki tarafını hatırlayabiliyorsanız, bu haftanın kavramsal iskeletini kurmuşsunuz demektir.

11. Kaynaklar ve ileri okuma

- **Güvenli programlama teknik kılavuzu** (ders kaynağı) — teslim edilen güvenlik kılavuzunun ve değişiklik/etki analizi belgelerinin yapısı; gereksinim bloğu ve durum ("karşılandı / devredildi / karşılanmadı").
- **OWASP MASVS** ve **MASTG** — mobil uygulama güvenlik gereksinimleri ve test rehberi.
- **OWASP WSTG** — web uygulama güvenlik testi rehberi.
- **PTES** (Penetration Testing Execution Standard) ve **NIST SP 800-115** — sızma testi süreç metodolojileri.
- **Ortak Kriterler (ISO/IEC 15408)** ve ilgili saldırı potansiyeli kılavuzları — 13. haftaya köprü.
- **CVSS** — zafiyet ciddiyet puanlaması; taban puan formülünün ayrıntılı hesaplama adımları CVSS v3.1 Belirtimi'nde (FIRST.org) yayımlanır.

Bu haftanın tek paragrafta özeti

Bir ürün, bağımsız bir laboratuvarda **beyaz kutu** erişimle, kod incelemesi → SAST → DAST → fuzzing → sızma testi sırasıyla incelenir; her zafiyet kanıtla bir **bulguya**, her bulgu **saldırı potansiyeli** (beş faktör: süre, uzmanlık, bilgi, fırsat, ekipman) ve **CVSS** (etki) ile bir **önceliğe** dönüşür; geliştirici düzeltme yapınca laboratuvar yalnız değişeni **delta değerlendirmeyle** yeniden inceler; kapatılmayan her risk **açıkça** yazılır; ve tüm bu zincir — bulgu, test kartı, etki analizi, aksiyon — birbirine referans vererek **izlenebilir** kalır. GüvenPay örneğinde baştan sona izlediğimiz F-03 zinciri (Adım 5 → PT-03 → bulgu → düzeltme → PT-07 → kapanış), bu paragrafın tek bir gerçek zamanlı örneğidir.

Bir sonraki hafta

13. hafta — Güvenlik gereksinimleri. Bu hafta "gereksinim şablonu" ve "uyum matrisi"ni süreç içinde gördük; 13. haftada iyi bir gereksinimin nasıl yazıldığını, izlenebilirlik/uyum matrisini ve Ortak Kriterler, FIPS 140-3, ETSI, EMVCo, PCI, MASVS gereksinim setlerini ayrıntılı işleyeceğiz.