Bölüm 13: Güvenlik

İşletim Sistemleri

Güvenlik hedefleri ve tehditler

Saldırganlar ve derinlemesine savunma

Şifreleme, özet, imza, PKI, TLS

Koruma etki alanları ve erişim matrisi

ACL ve yetenekler, en az ayrıcalık

Bell-LaPadula, Biba, gizli kanallar

Kimlik doğrulama: parola, MFA, geçiş anahtarı

Yazılım açıkları ve savunmalar

İçeriden saldırılar

Kötü amaçlı yazılımlar

Korumalı alan, konteyner, SELinux

Güvenli önyükleme ve yan kanallar

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/os/slides/Bolum_13_Guvenlik.html

GİRİŞ

Öğrenme Çıktıları

Hedefler ve tehditler
Gizlilik, bütünlük ve erişilebilirlik hedeflerini; saldırı türlerini ve saldırgan profillerini ayırt etmek.

Kriptografi
Simetrik ve asimetrik şifrelemeyi, özet fonksiyonlarını, dijital imzayı, sertifikaları ve TLS el sıkışmasını açıklamak.

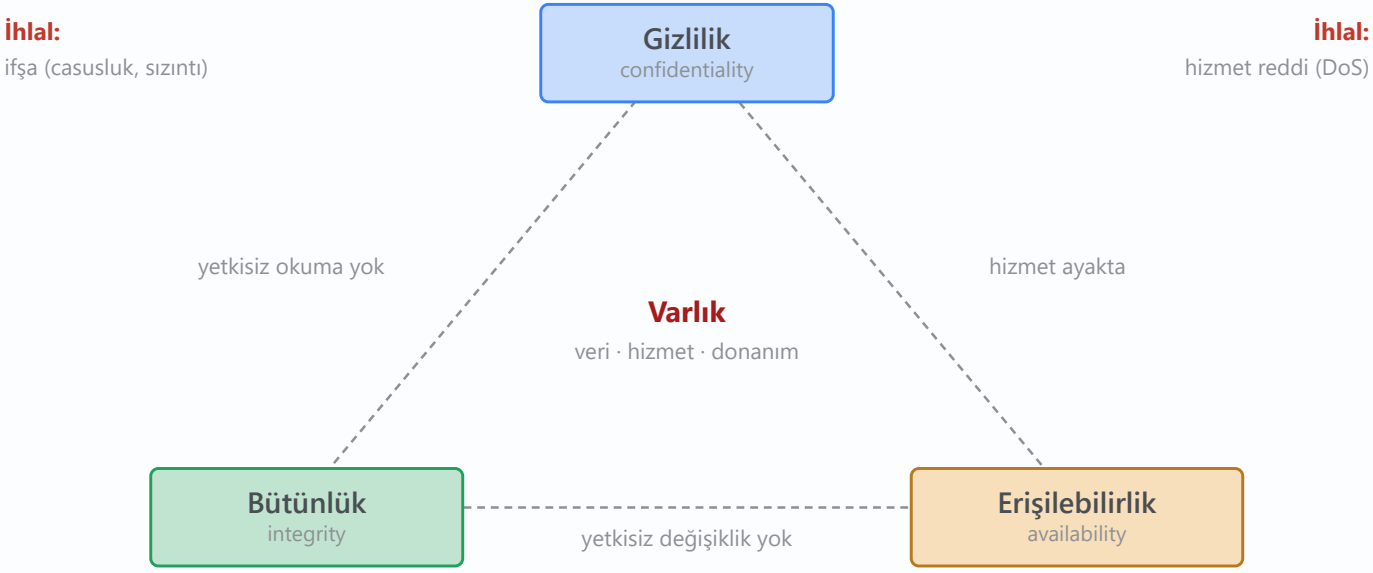
Koruma
Erişim matrisini ACL ve yetenek biçiminde okumak; en az ayrıcalığı, setuid ve Linux yeteneklerini uygulamak.

Biçimsel modeller
Bell-LaPadula ve Biba kurallarını uygulamak; gizli kanalların neden kapatılamadığını açıklamak.

Kimlik doğrulama
Tuzlu yavaş parola özetlerini, MFA, TOTP ve FIDO2 geçiş anahtarlarını karşılaştırmak.

Saldırılar ve savunmalar
Bellek güvenliği hatalarını, TOCTOU'yu ve kötü amaçlı yazılımları; kanarya, NX, ASLR, CFI, korumalı alan ve güvenli önyüklemeyi tanımak.

Güvenlik Hedefleri: CIA Üçlüsü

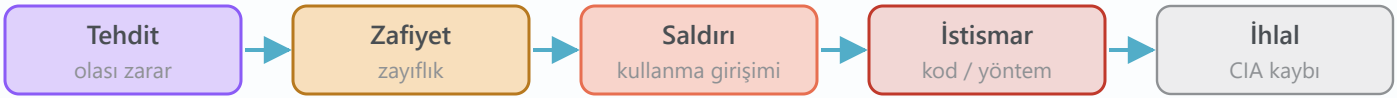


İngilizce karşılıklar: **confidentiality** (gizlilik), **integrity** (bütünlük), **availability** (erişilebilirlik/kullanılabilirlik). Bunlara sık eklenenler: **özgünlük** (authenticity) ve **hesap verebilirlik** (inkâr edilemezlik, denetim kaydı).

GİRİŞ

Tehditler ve İhlal Kategorileri

Kategori	Tanım	Örnek
Gizlilik ihlali	Verinin izinsiz okunması	Kredi kartı veritabanının sızdırılması
Bütünlük ihlali	Verinin izinsiz değiştirilmesi	Not sisteminde kayıt tahrifatı
Erişilebilirlik ihlali	Verinin/hizmetin yok edilmesi	Fidye yazılımının diski şifrelemesi
Hizmet hırsızlığı	Kaynakların izinsiz kullanımı	Ele geçirilen sunucuda kripto para kazma
Hizmet reddi (DoS/DDoS)	Meşru kullanımın engellenmesi	Botnet ile SYN seli
Kimlik değiştirme	Başkası gibi davranma (masquerading)	Oltalama ile çalınan parola
Ayrıcalık yükseltme	Hak edilenden fazla yetki kazanma	setuid programdaki hatadan root olma



yama zafiyeti, kimlik doğrulama saldırıyı, kanarya/NX istismarı, yedek ihlalin sonucunu sınırlar

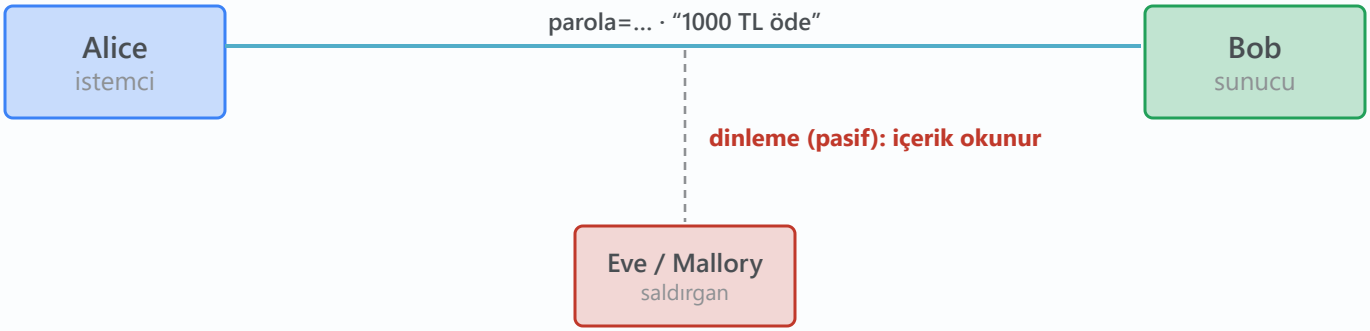
Tehdit (threat) olası bir zarar, **zafiyet** (vulnerability) onu mümkün kılan zayıflık, **saldırı** (attack) zafiyeti kullanma girişimi, **istismar** (exploit) de bunu yapan kod ya da yöntemdir.

Ağ Saldırıları: Dinleme, Tekrar, Ortadaki Adam



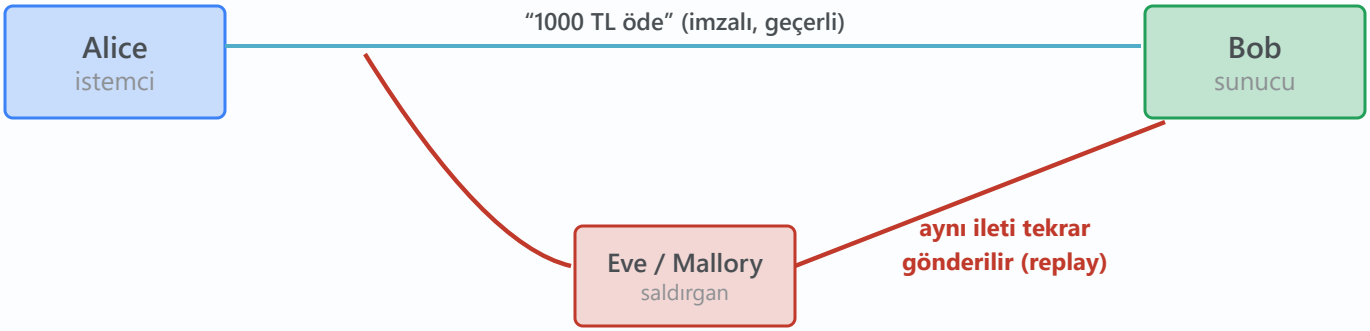
Alice ile Bob arasında normal iletişim. Kanal şifresiz ve kimlik doğrulamasızsa aşağıdaki saldırıların hepsi mümkündür.

Ağ Saldırıları: Dinleme, Tekrar, Ortadaki Adam



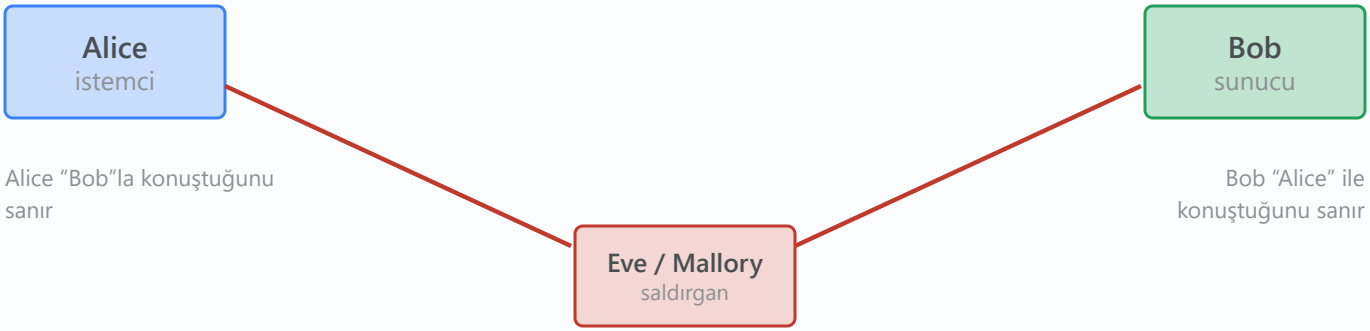
Dinleme (eavesdropping): pasif saldırgan trafiği kopyalar → gizlilik ihlali. Çözüm: şifreleme (TLS).

Ağ Saldırıları: Dinleme, Tekrar, Ortadaki Adam



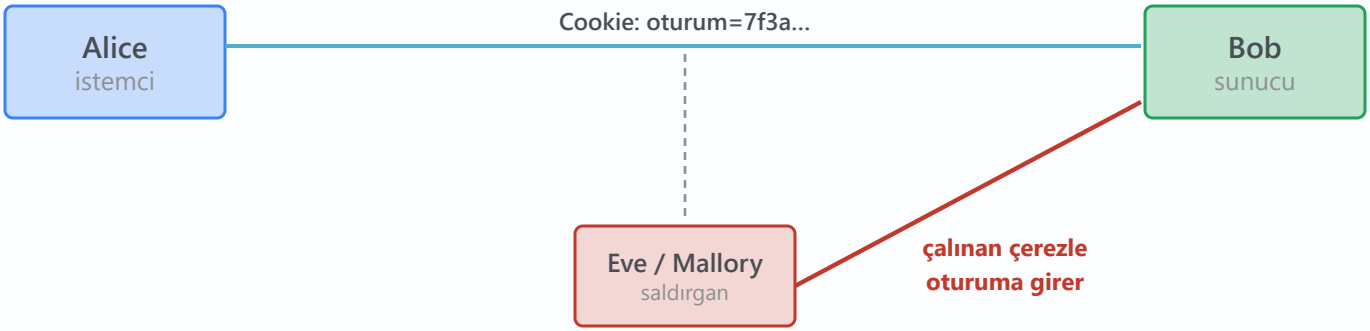
Tekrar saldırısı (replay): şifreli ve imzalı ileti bile yeniden gönderilebilir. Çözüm: tek kullanımlık sayı (nonce), zaman damgası, sıra numarası.

Ağ Saldırıları: Dinleme, Tekrar, Ortadaki Adam



Ortadaki adam (MITM): saldırgan iki tarafa da karşı taraf gibi görünür. Çözüm: karşı tarafın kimliğini sertifika ile doğrulamak.

Ağ Saldırıları: Dinleme, Tekrar, Ortadaki Adam



Oturum kaçırma (session hijacking): kimlik doğrulama sonrası verilen oturum belirteci çalınır. Çözüm: yalnız HTTPS, `Secure` / `HttpOnly` çerez, kısa ömür.

GİRİŞ

Saldırganlar ve Kazayla Veri Kaybı

Meraklı içeriden kişi
Teknik bilgisi az; başkasının e-postasına, dosyasına “bakmak”. Etik dışı, çoğu zaman yasa dışı.

Kötü niyetli çalışan
Yetkisini kötüye kullanır: veri sızdırma, sabotaj, mantık bombası. En zor fark edilen grup.

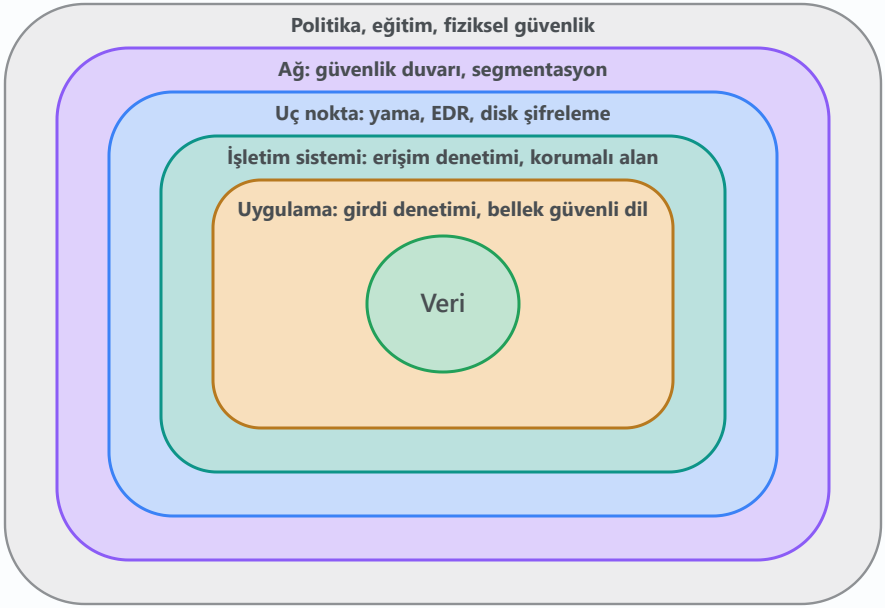
Siber suçlu
Kâr amaçlı: fidye yazılımı, kimlik ve kart hırsızlığı, kiralık botnet. Bugün en yaygın tehdit.

Devlet destekli gruplar (APT)
Askerî ve ticari casusluk, kritik altyapı sabotajı; sıfırıncı gün açıkları ve tedarik zinciri saldırıları.

Kazayla veri kaybı	Örnek	Önlem
Doğal olaylar	yangın, sel, deprem (ve bantları kemiren fareler)	coğrafi olarak ayrı yedek
Donanım/yazılım hatası	disk arızası, CPU hatası, yazılım hatası	RAID, sağlama toplamı, sürüm
İnsan hatası	yanlış komut, yanlış diski biçimleme, kayıp aygıt	3-2-1 yedekleme, şifreli disk

Kazayla kayıp, saldırılardan daha sık veri kaybettirir. 3-2-1: 3 kopya, 2 farklı ortam, 1 kopya başka yerde ve tercihen değiştirilemez (immutable) — fidye yazılımına karşı da işe yarar.

Derinlemesine Savunma ve İyi Uygulamalar

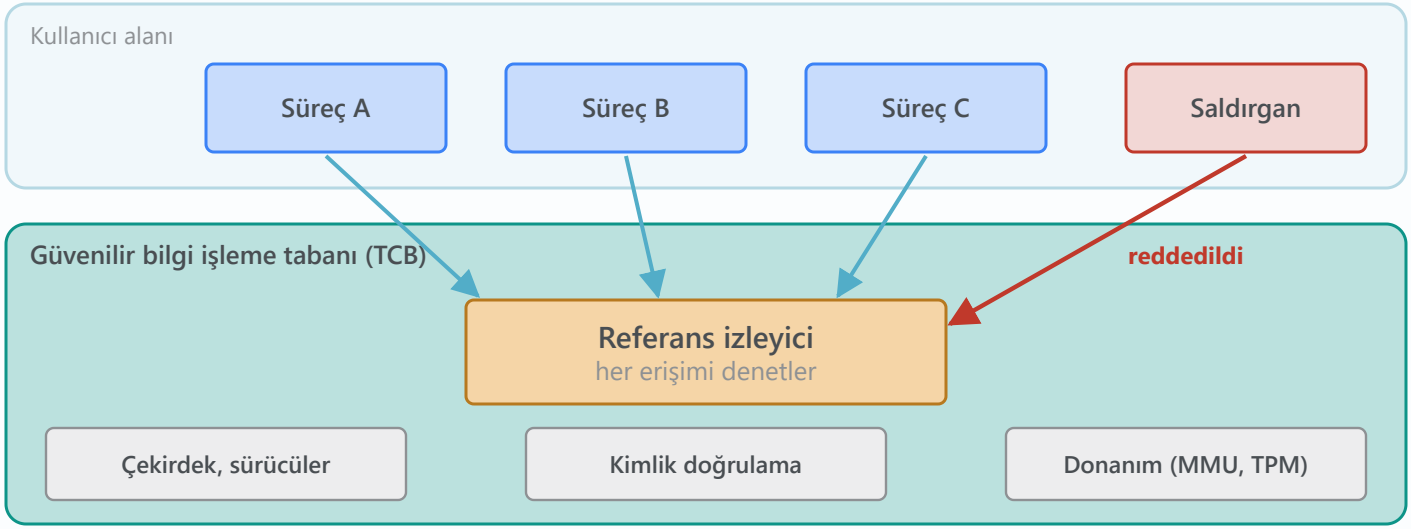




- **Yama**: güvenlik güncellemeleri hızla uygulanır (çoğu saldırı bilinen açıkları kullanır).
- **Güçlü kimlik doğrulama** ve en az ayrıcalıkla erişim denetimi; **günlük kaydı** ve izleme.
- Düzenli **sızma testi**; kullanıcıların oltalamaya karşı **eğitimi**; sıfır güven (zero trust): ağın içi de güvenilir sayılmaz.

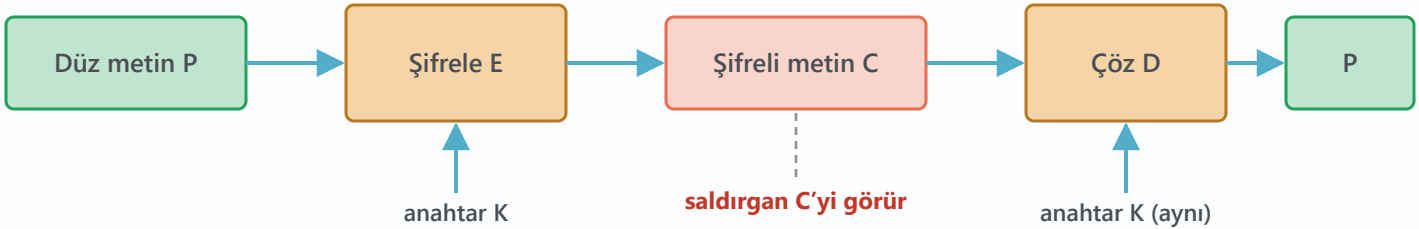
GİRİŞ

Güvenilir Bilgi İşleme Tabanı (TCB)



Referans izleyici üç koşulu sağlamalı: **her** erişimde çağrılır (atlanamaz), **kurcalanamaz**, doğrulanabilecek kadar **küçüktür**. TCB ne kadar büyükse hata olasılığı o kadar yüksektir: mikro çekirdek ve seL4 gibi biçimsel doğrulanmış çekirdekler bu yüzden ilgi görür.

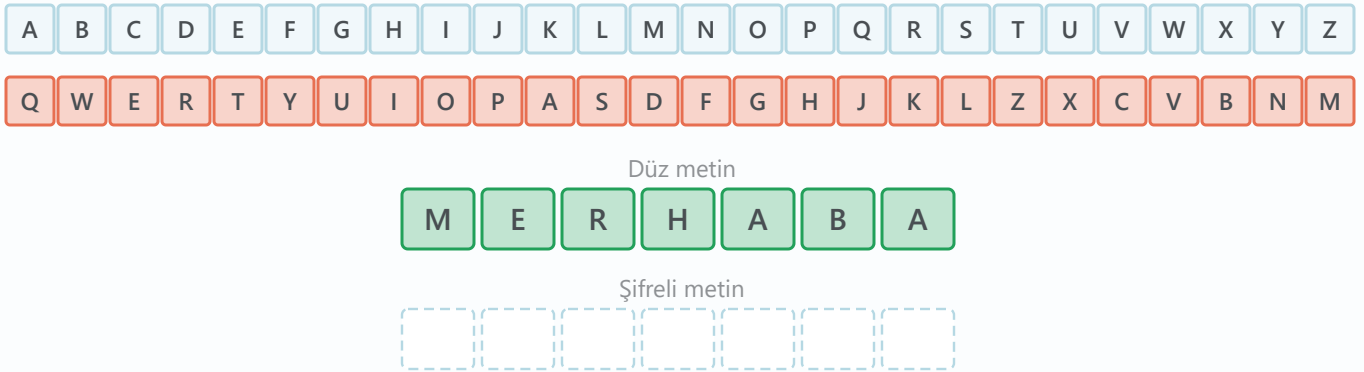
Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



$$C = E(P, K) \cdot P = D(C, K)$$

Simetrik (gizli anahtarlı) şifreleme: aynı K ile şifrele ve çöz. **Kerckhoffs ilkesi**: algoritma açık olabilir, güvenlik yalnızca anahtarın gizliliğine dayanmalı.

Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



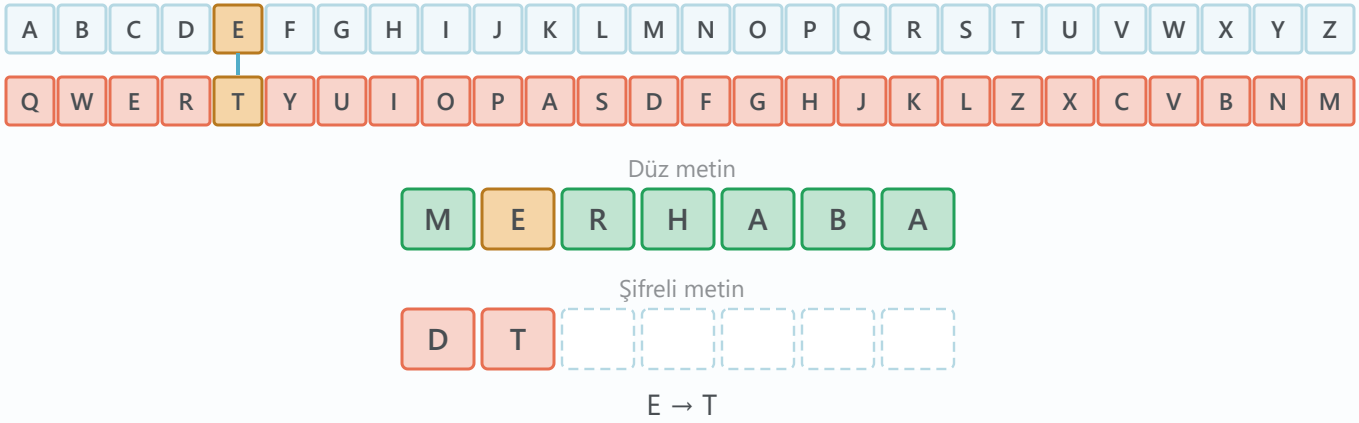
Tek alfabeli yerine koyma: her harf sabit bir harfe eşlenir; anahtar bu 26 harflik permütasyondur ($26! \approx 4\times10^{26}$ olası anahtar).

Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme

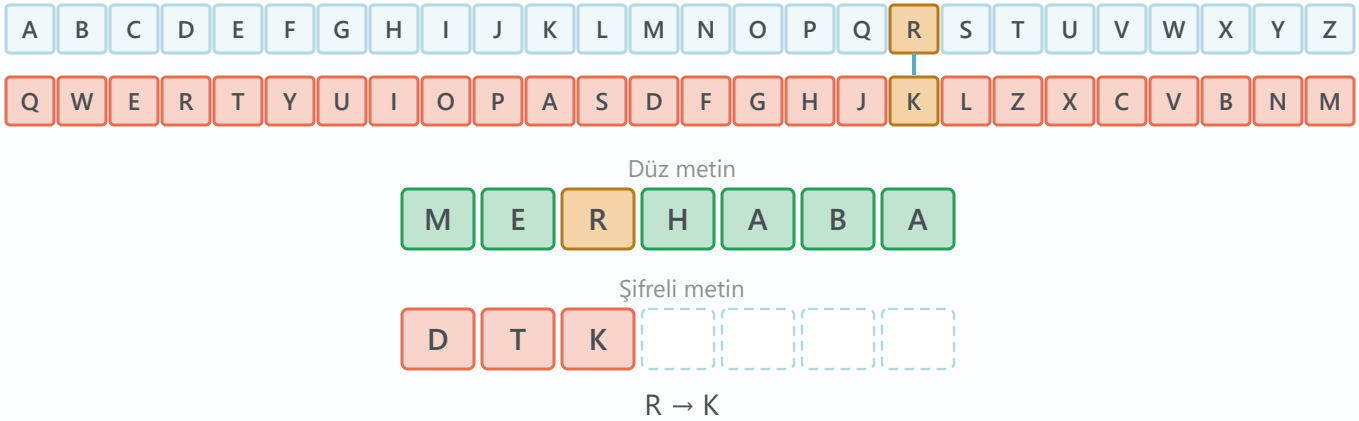


M → D: tabloda M’nin altındaki harf D.

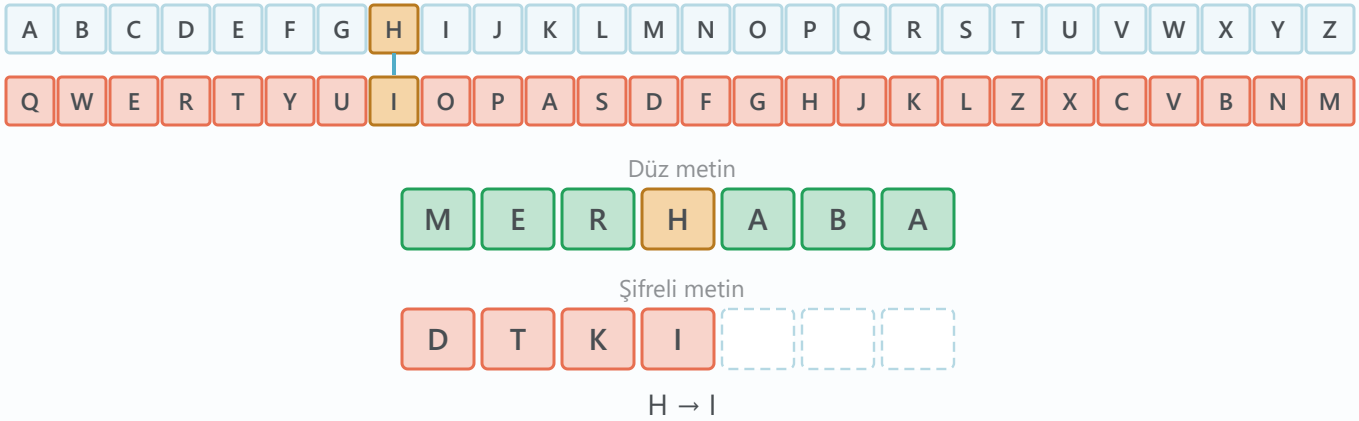
Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



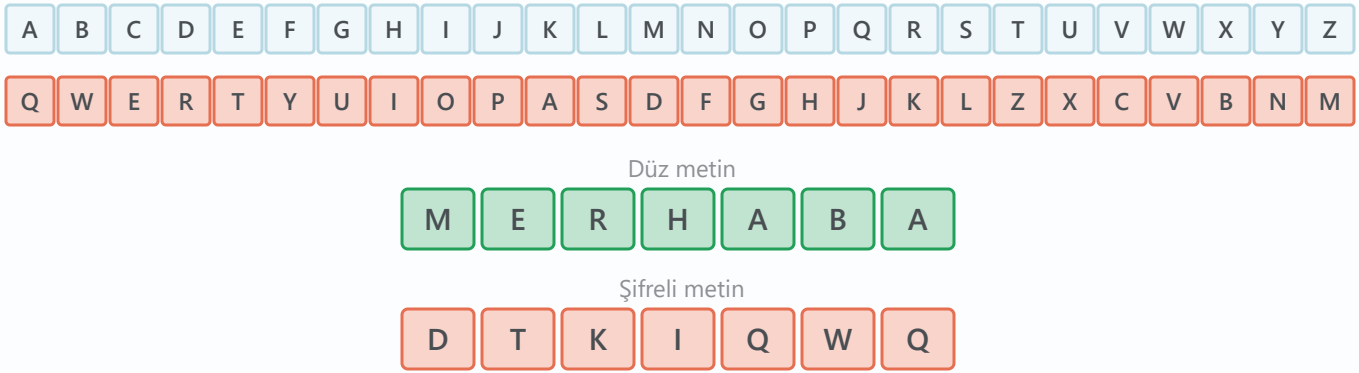
Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme

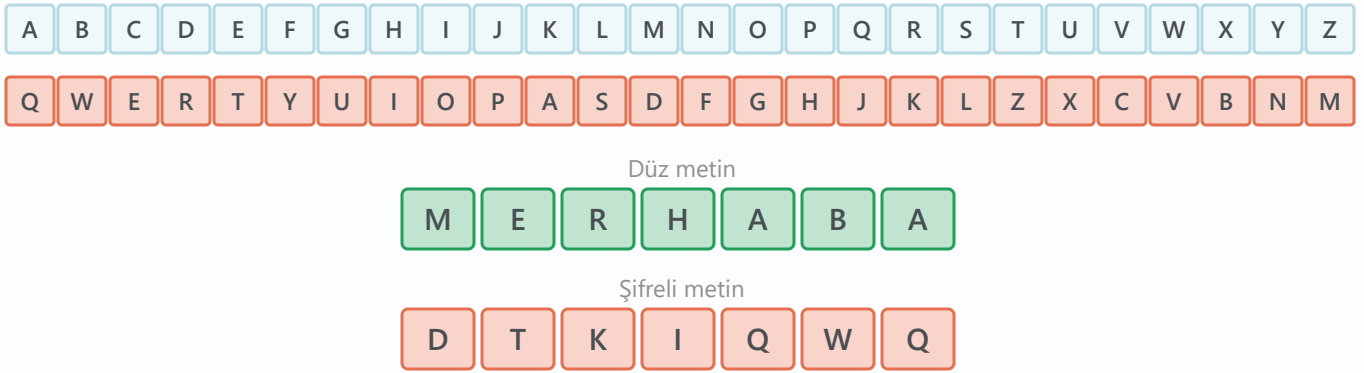


Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



Sonuç: MERHABA → **DTKIQWQ**. İki A da Q oldu: harf sıklıkları korunur.

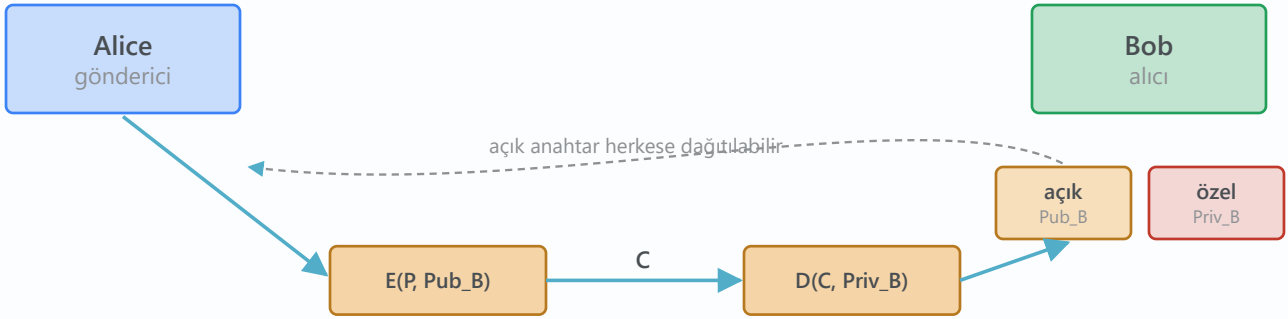
Şifrelemenin Temelleri ve Gizli Anahtarlı Şifreleme



Anahtar uzayı büyük olsa da **sıklık analizi** (Türkçede A, E, İ en sık) birkaç satırlık metinle şifreyi kırar. Bugün: **AES-256-GCM**, **ChaCha20-Poly1305** (şifreleme + bütünlük birlikte).

KRİPTOGRAFİ

Açık Anahtarlı Şifreleme

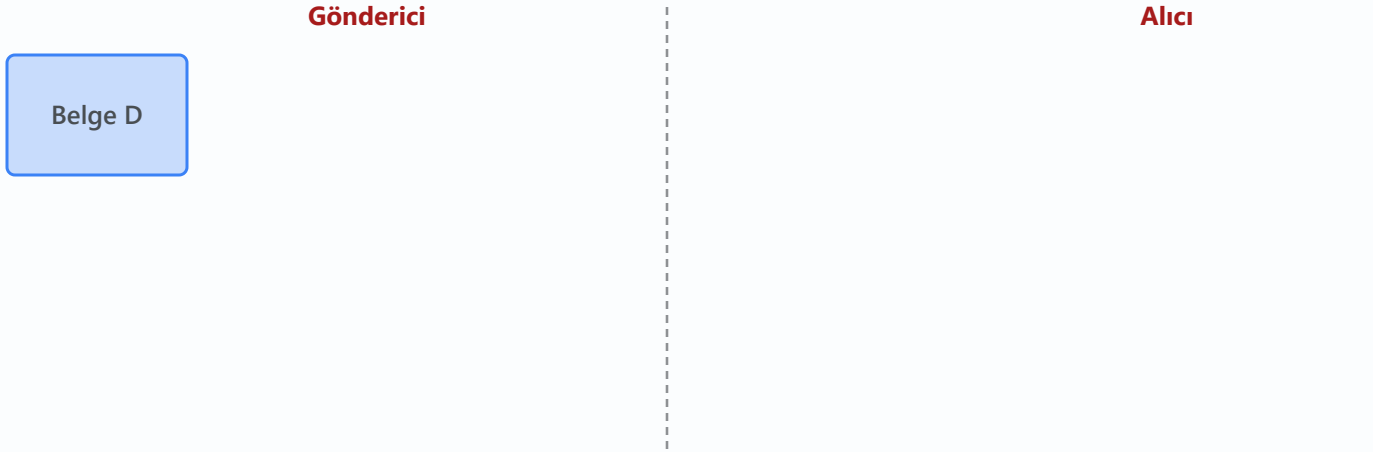


Pub_B ile şifrelenen yalnız Priv_B ile çözülür.

	Simetrik	Asimetrik (açık anahtarlı)
Anahtar	Tek, paylaşılan	Açık + özel çift
Hız	Çok hızlı (AES-NI)	100–1000 kat yavaş
Kullanım	Toplu veri	Anahtar değişimi, imza
Örnek	AES, ChaCha20	RSA, ECDH/X25519, Ed25519, ML-KEM

Asimetrik şifrelemenin temeli **tek yönlü tuzak kapılı fonksiyondur**: iki büyük asalı çarpmak kolay, çarpımı çarpanlarına ayırmak (RSA) ya da ayrık logaritma (ECC) zor. Kuantuma dayanıklı ML-KEM 2024'te standartlaştı (FIPS 203).

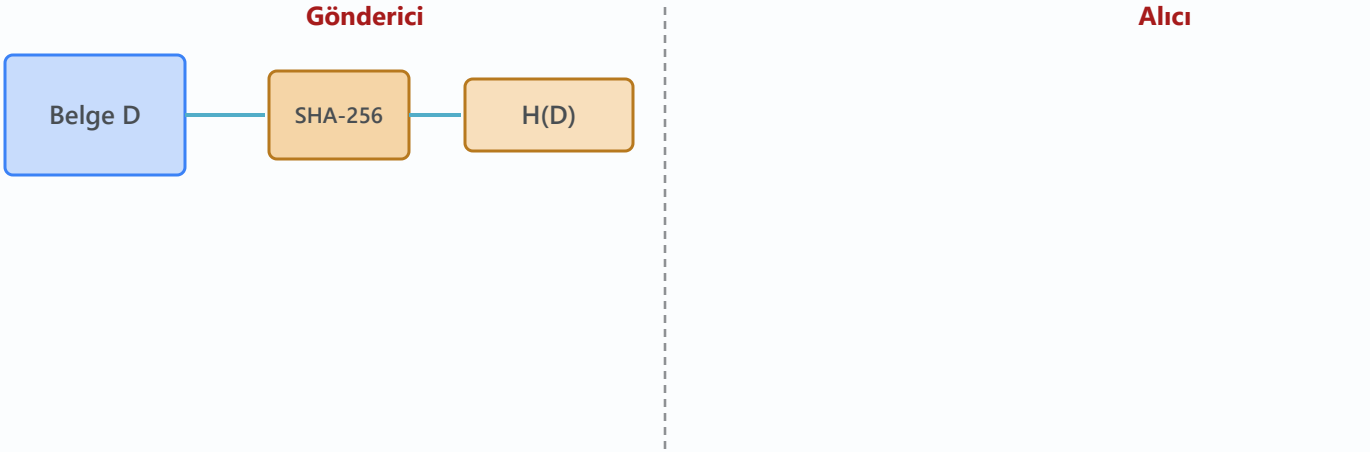
Özet Fonksiyonları ve Dijital İmza



Gönderici D belgesini değiştirilmeden ve kendisinden geldiği kanıtlanarak iletmek istiyor (gizlilik şart değil).

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

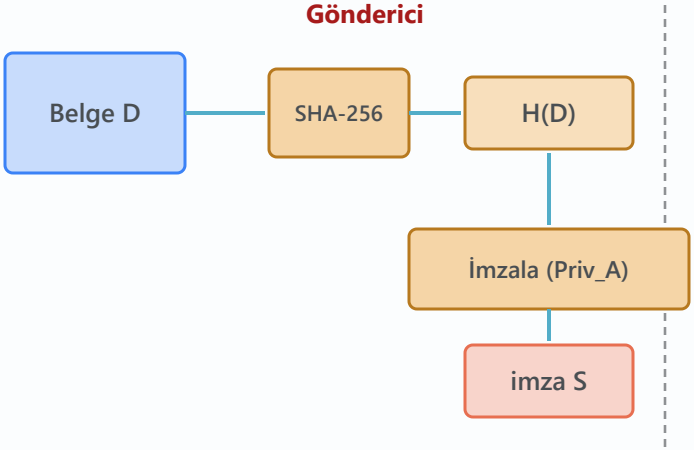
Özet Fonksiyonları ve Dijital İmza



Özet fonksiyonu (SHA-256, SHA-3) her boydaki girdiden sabit 256 bitlik özet üretir: tek yönlü, çakışmaya dayanıklı; tek bit değişince özetin yarısı değişir. MD5 ve SHA-1 kırıktır.

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

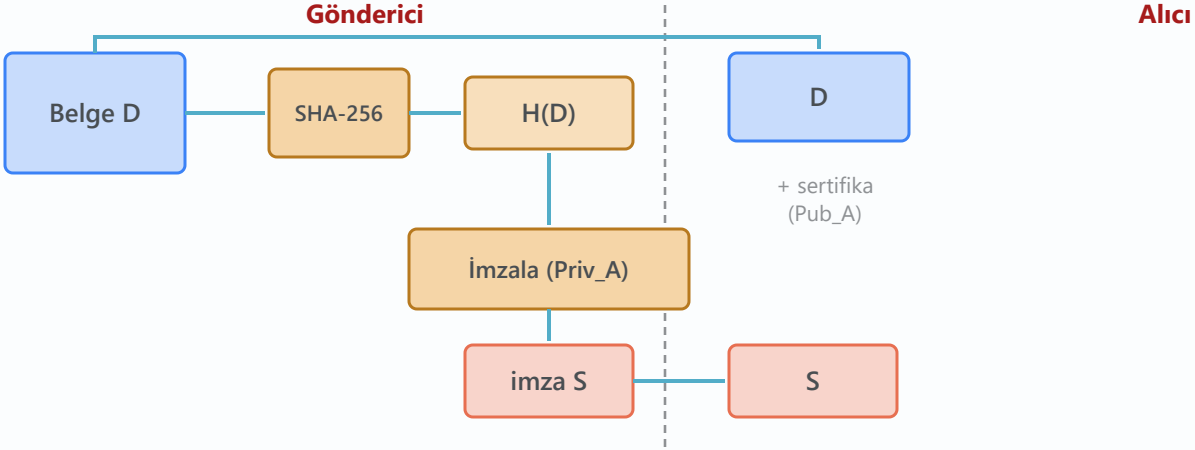
Özet Fonksiyonları ve Dijital İmza



Özet **gönderenin özel anahtarıyla** imzalanır (RSA-PSS, ECDSA, Ed25519). Bu adımı yalnız Priv_A sahibi yapabilir.

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

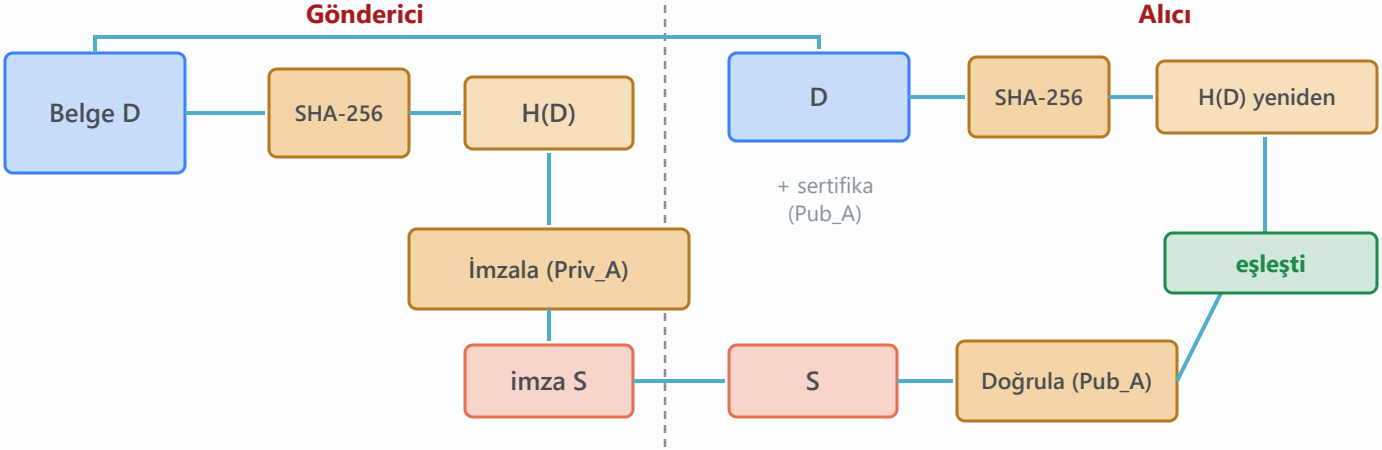
Özet Fonksiyonları ve Dijital İmza



Belge, imza ve göndericinin açık anahtarını taşıyan sertifika birlikte gönderilir.

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

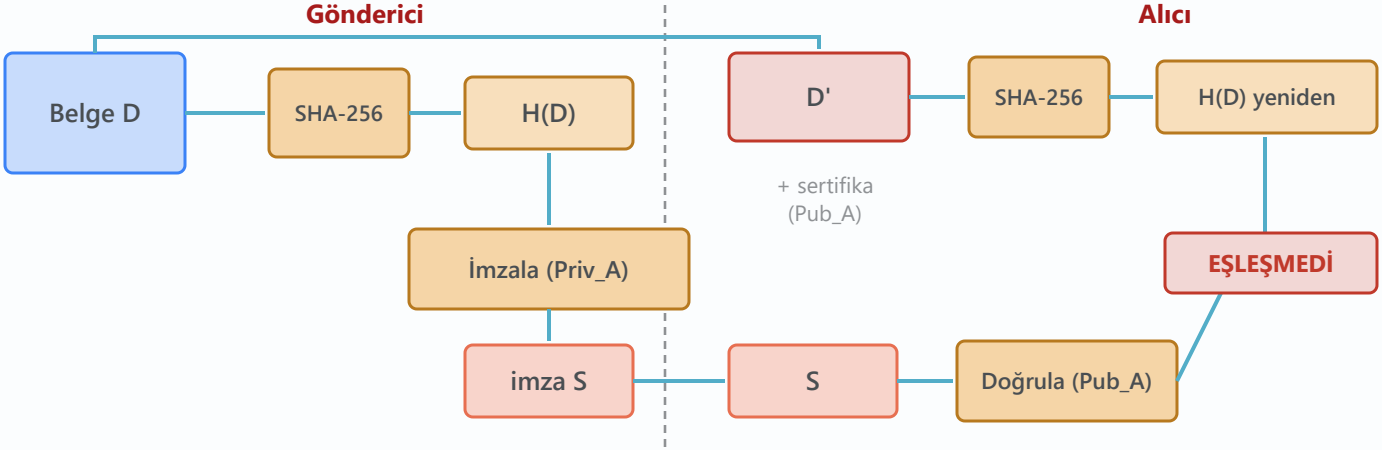
Özet Fonksiyonları ve Dijital İmza



Alıcı özeti kendisi hesaplar ve imzayı Pub_A ile doğrular: eşleşirse belge **bozulmamış** ve **Alice'ten** gelmiştir (inkâr edilemez).

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

Özet Fonksiyonları ve Dijital İmza



Yolda tek bayt değişirse (D′) özet tutmaz ve doğrulama başarısız olur. Saldırgan yeni geçerli imza üretemez, çünkü Priv_A onda yok.

Özet bir sıkıştırma değildir; **imzasız** özet de tek başına hiçbir şey kanıtlamaz (saldırgan belgeyle birlikte özeti de değiştirebilir).

KRİPTOGRAFİ

Sertifikalar ve Açık Anahtar Altyapısı (PKI)



Tarayıcının doğrulaması

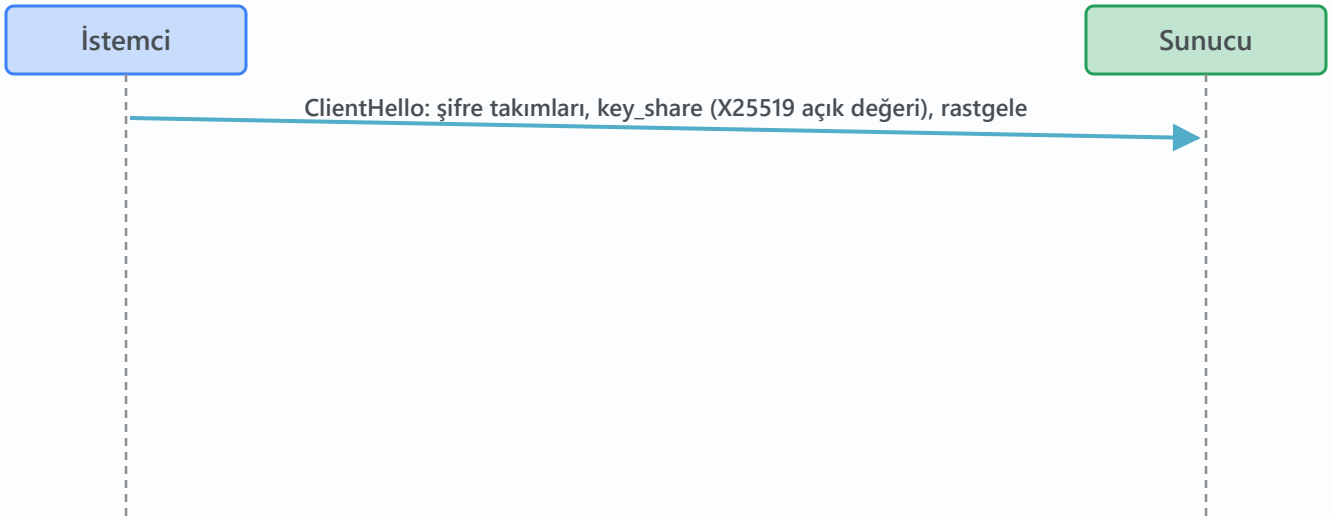
1. Sunucu sertifikası + ara CA gelir
2. İmzalar zincir boyunca doğrulanır
3. Kök, güven deposunda mı?
4. Alan adı eşleşiyor mu? Süresi?
5. İptal (OCSP/CRL) ve CT kaydı
6. Hepsi tamam → bağlantı güvenilir

Sahte sertifika (MITM)

güvenilir bir CA imzalamadığı için
zincir kurulamaz → uyarı

Sertifika (X.509) = açık anahtar + kimlik (alan adı) + geçerlilik süresi + CA imzası. Açık anahtarın gerçekten o kişiye ait olduğu sorununu çözer. Let’s Encrypt (ACME) sayesinde sertifikalar ücretsiz ve 90 günlük; ömürlerin kademeli olarak 47 güne inmesi kararlaştırıldı.

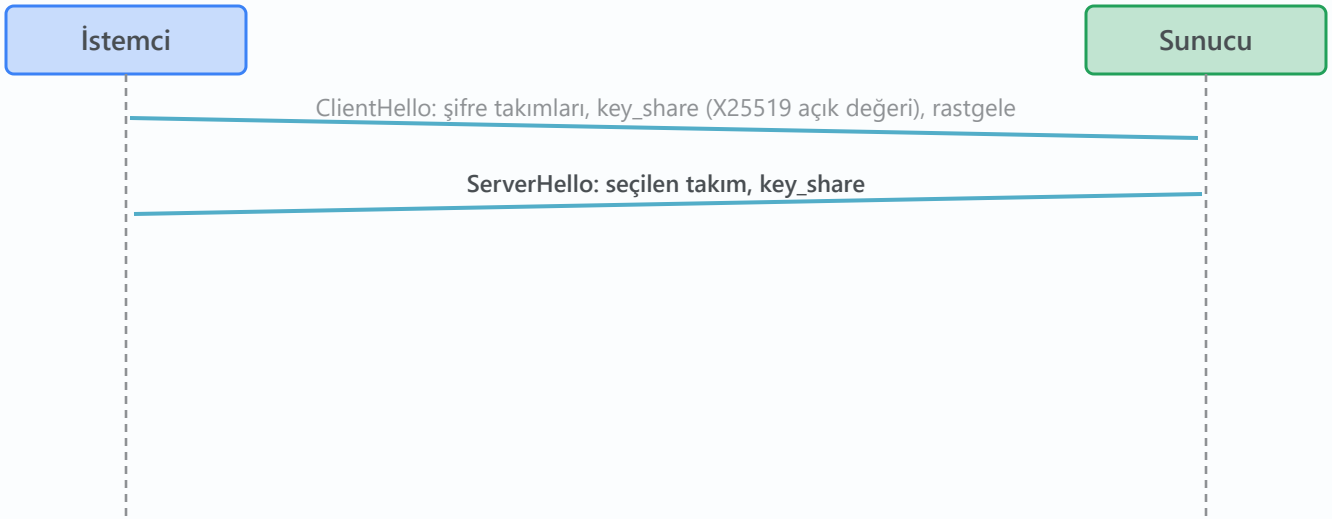
TLS 1.3 El Sıkışması (Basitleştirilmiş)



{ } = şifreli gönderilir (yeşil)

İstemci desteklediği algoritmaları ve **geçici** (ephemeral) Diffie-Hellman açık değerini ilk iletide gönderir.

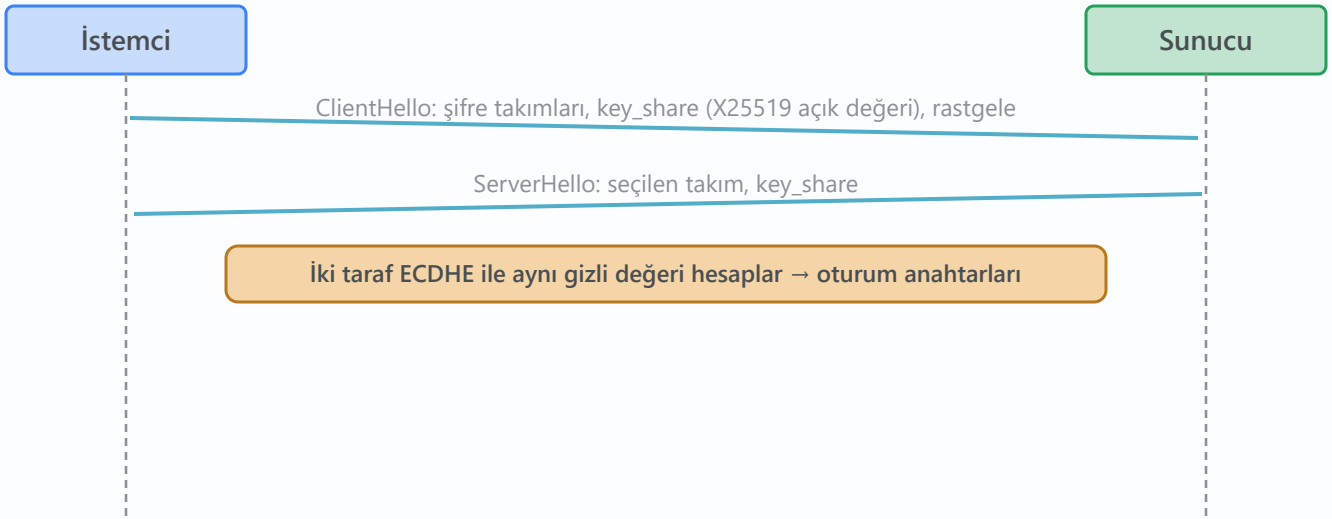
TLS 1.3 El Sıkışması (Basitleştirilmiş)



{ } = şifreli gönderilir (yeşil)

Sunucu birini seçer ve kendi DH açık değerini gönderir. Şimdiye kadarki her şey açıktır.

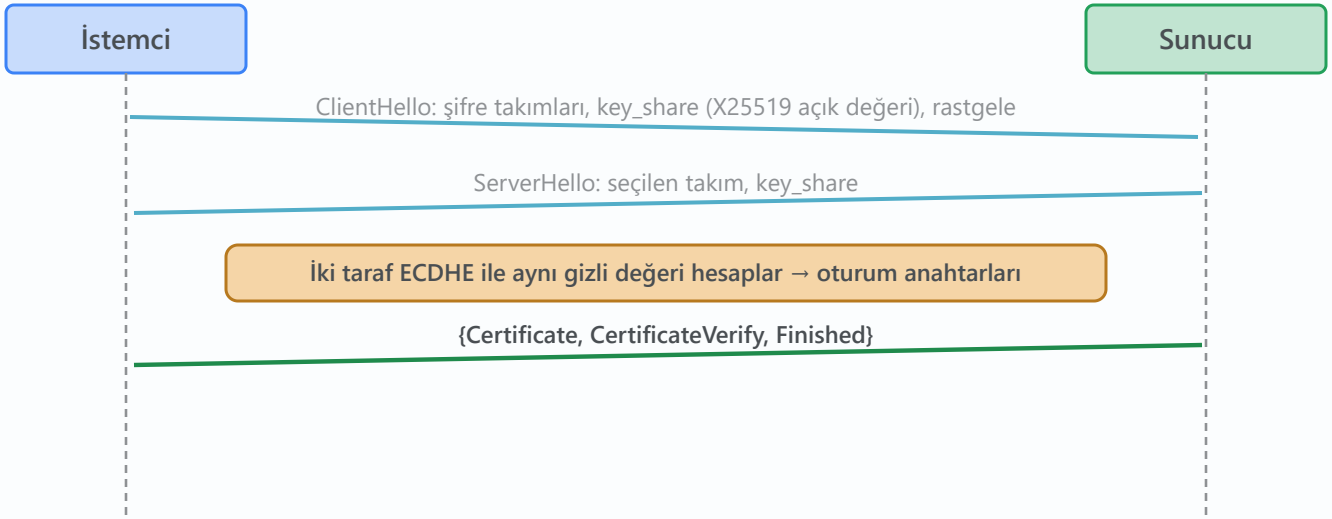
TLS 1.3 El Sıkışması (Basitleştirilmiş)



{ } = şifreli gönderilir (yeşil)

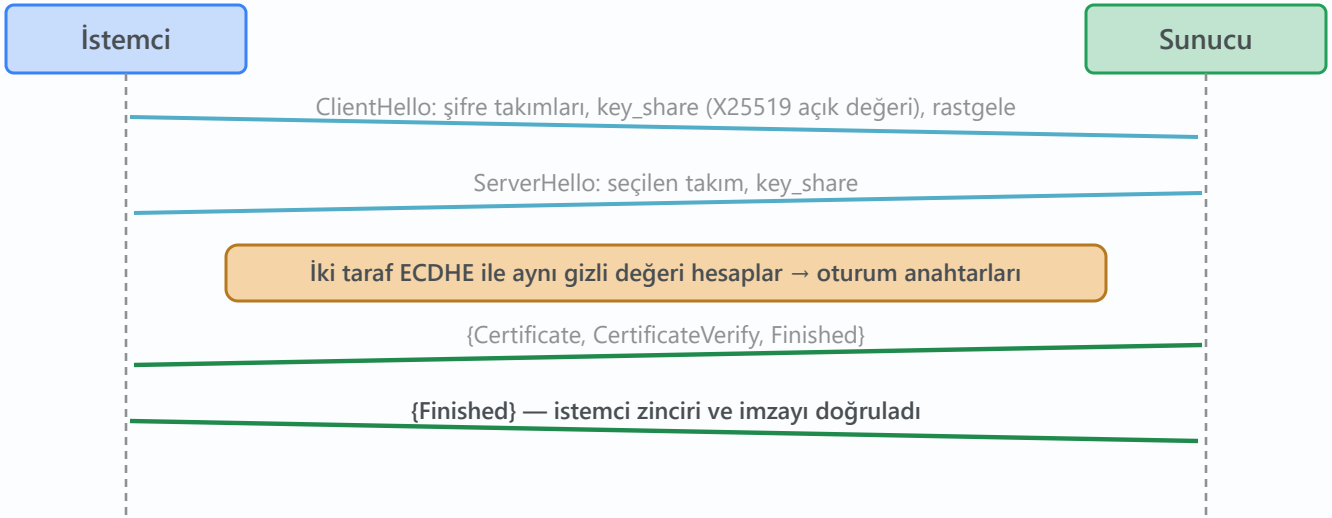
Her iki taraf da karşı tarafın açık değeri ile kendi gizli değerinden **aynı ortak sırrı** hesaplar; sır ağda hiç dolaşmaz.

TLS 1.3 El Sıkışması (Basitleştirilmiş)



Sunucu sertifikasını gönderir ve el sıkışmanın özetini **özel anahtarıyla imzalar** (CertificateVerify): MITM bu imzayı üretemez.

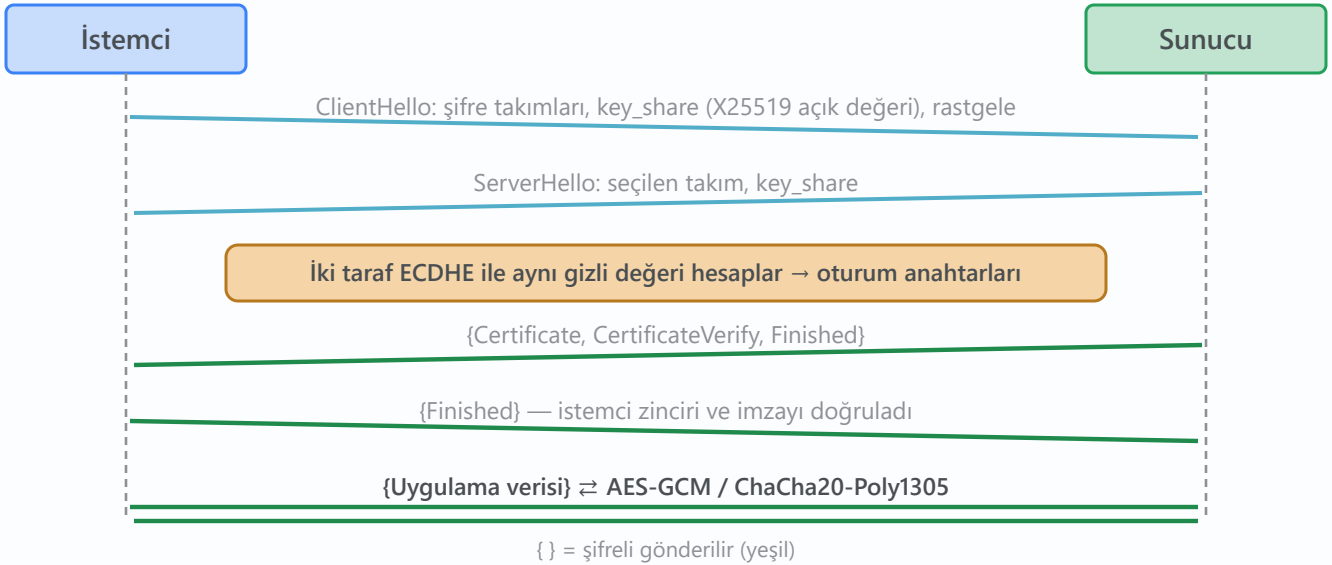
TLS 1.3 El Sıkışması (Basitleştirilmiş)



{ } = şifreli gönderilir (yeşil)

İstemci zinciri ve imzayı doğrular, Finished ile el sıkışmanın değiştirilmediğini onaylar. Yalnız **1 gidiş-dönüş** (1-RTT).

TLS 1.3 El Sıkışması (Basitleştirilmiş)



Uygulama verisi simetrik, kimlik doğrulamalı şifreleme ile akar. Geçici anahtarlar sayesinde **ileriye dönük gizlilik**: özel anahtar sonradan çalınsa bile eski trafik çözülemez.

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
 1 long modpow(long g, long e, long p) {
 2     long r = 1;
 3     while (e > 0) {
 4         if (e & 1) r = r * g % p;
 5         g = g * g % p;
 6         e >>= 1;
 7     }
 8     return r;
 9 }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = ?	A = ?	s = ?
	B = ?	

modpow(5, 6, 23)

e: 1 1 0 r = 1 g = 5

e = 6 = 110_2; bitler sağdan sola işlenir

Alice gizli **a = 6**'yı seçmiş; açık değeri A = 5^6 mod 23 hesaplanıyor. Kare al ve çarp yöntemi üssün bitlerini sağdan sola tarar; r = 1.

Adım 1 / 8

DEĞİŞKENLER

g = 5 e = 6 p = 23 r = 1

ÇIKTI

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
▶      while (e > 0) {
▶          if (e & 1) r = r * g % p;
▶          g = g * g % p;
▶          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = ?	A = ?	s = ?
	B = ?	

modpow(5, 6, 23)

e: 1 1 0 r = 1 g = 2

bit 0 = 0 → r aynı kalır

e = 6 çift: r değişmez. Taban kare alınır: g = 5·5 mod 23 = **2**; e = 3.

Adım 2 / 8

DEĞİŞKENLER

g = 2 | e = 3 | p = 23 | r = 1

ÇIKTI

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
▶      while (e > 0) {
▶          if (e & 1) r = r * g % p;
▶          g = g * g % p;
▶          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = ?	A = ?	s = ?
	B = ?	

modpow(5, 6, 23)

e: 1 **1** 0 r = 2 g = 4

bit 1 = 1 → r = r·g mod p

e = 3 tek: r = 1·2 = **2**. g = 2·2 = 4; e = 1.

Adım 3 / 8

DEĞİŞKENLER

g = 4 | e = 1 | p = 23 | r = 2

ÇIKTI

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
▶      while (e > 0) {
▶          if (e & 1) r = r * g % p;
▶          g = g * g % p;
▶          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = ?	A = ?	s = ?
	B = ?	

modpow(5, 6, 23)

e: 1 1 0 r = 8 g = 16

bit 2 = 1 → r = r·g mod p

e = 1 tek: r = 2·4 = **8**. g = 16; e = 0, döngü biter. 6 çarpma yerine 3 tur: üs n bitse yalnız n tur.

Adım 4 / 8

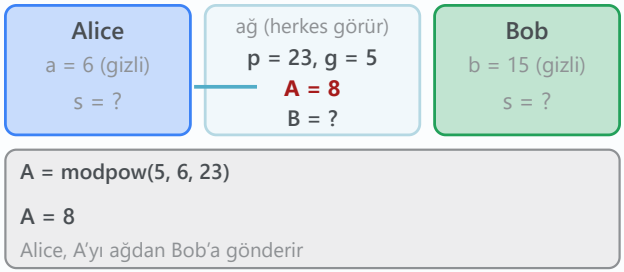
DEĞİŞKENLER

g = 16 e = 0 p = 23 r = 8

ÇIKTI

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
3      while (e > 0) {
4          if (e & 1) r = r * g % p;
5          g = g * g % p;
6          e >>= 1;
7      }
▶      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
▶      long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```



Fonksiyon **8** döndürür: A = 8. Bu değer açıktır; a = 6 ise Alice'ten hiç çıkmaz.

Adım 5 / 8

DEĞİŞKENLER

A = 8

ÇIKTI

```
A = 8
```

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
3      while (e > 0) {
4          if (e & 1) r = r * g % p;
5          g = g * g % p;
6          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
▶      long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = ?	A = 8	s = ?
	B = 19	

B = modpow(5, 15, 23)

e = 15 = 1111_2: 4 tur → B = 19

Bob, B'yi ağdan Alice'e gönderir

Bob da gizli b = 15 ile aynı fonksiyonu çalıştırır: 5^{15} mod 23 = **19**. B = 19 ağdan Alice'e gider.

Adım 6 / 8

DEĞİŞKENLER

A = **8** B = **19**

ÇIKTI

```
A = 8
B = 19
```

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
3      while (e > 0) {
4          if (e & 1) r = r * g % p;
5          g = g * g % p;
6          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
▶      long sA = modpow(B, 6, 23);
15     long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = 2	A = 8	s = ?
	B = 19	

sA = modpow(19, 6, 23)

Alice: B^a mod p = 19^6 mod 23 = 2

ortak sır ağda dolaşmadı

Alice, Bob'un açık değerini kendi gizli üssüyle yükseltir: 19^6 mod 23 = **2**.

Adım 7 / 8

DEĞİŞKENLER

A = **8** | B = **19** | sA = **2**

ÇIKTI

```
A = 8
B = 19
sA = 2
```

Diffie-Hellman Anahtar Değişimi: Kod İzleme

```
1  long modpow(long g, long e, long p) {
2      long r = 1;
3      while (e > 0) {
4          if (e & 1) r = r * g % p;
5          g = g * g % p;
6          e >>= 1;
7      }
8      return r;
9  }
10 int main(void) {
11     /* açık: p = 23, g = 5 */
12     long A  = modpow(5, 6, 23);
13     long B  = modpow(5, 15, 23);
14     long sA = modpow(B, 6, 23);
▶      long sB = modpow(A, 15, 23);
16 }
```

Alice	ağ (herkes görür)	Bob
a = 6 (gizli)	p = 23, g = 5	b = 15 (gizli)
s = 2	A = 8	**s = 2**
	B = 19	

sB = modpow(8, 15, 23)

Bob: A^b mod p = 8^{15} mod 23 = 2

her iki taraf da 5^(6·15) mod 23 = 2 buldu

Bob da 8^{15} mod 23 = **2** bulur: ikisi de 5^(a·b) mod p'yi hesapladı. Ağı dinleyen yalnız p, g, A, B'yi görür; a'yı bulmak ayrık logaritma problemidir (gerçekte 2048 bit p ya da X25519).

Adım 8 / 8

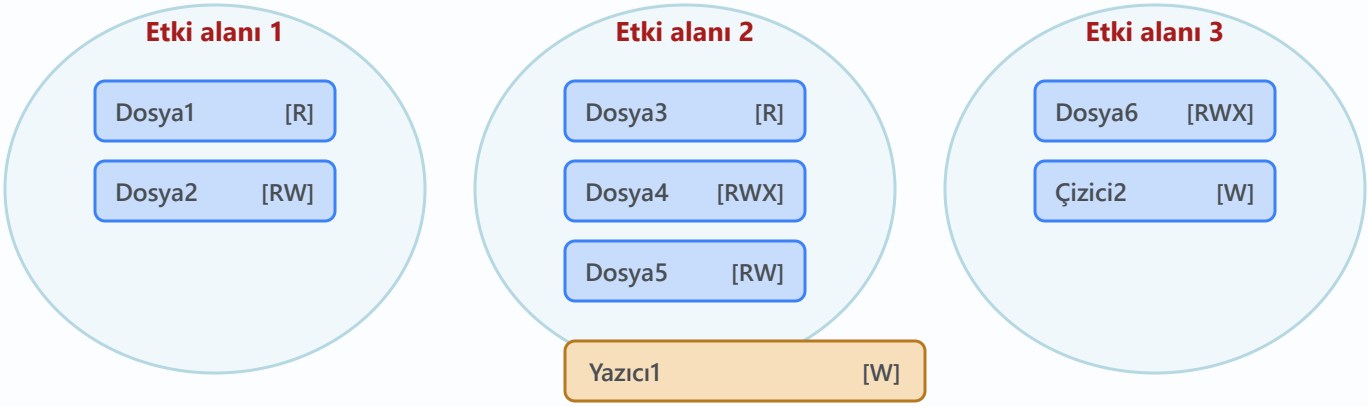
DEĞİŞKENLER

`A = 8` `B = 19` `sA = 2` `sB = 2`

ÇIKTI

```
A = 8
B = 19
sA = 2
sB = 2
```

Koruma Etki Alanları



Yazıcı1 hem D2 hem D3'te: nesneler etki alanları arasında paylaşılabilir

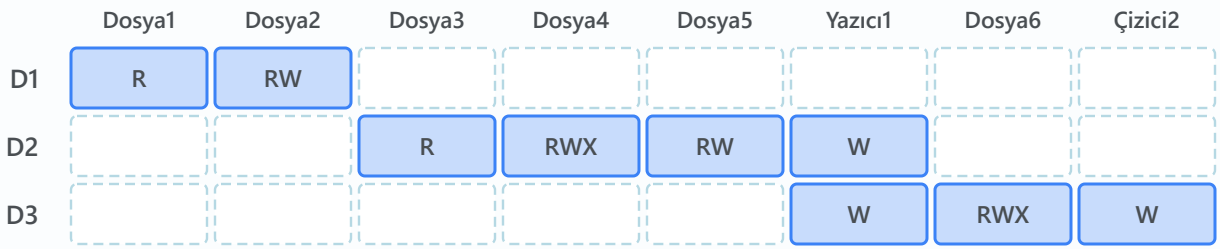
- **Etki alanı** (domain): (nesne, haklar) çiftlerinden oluşan küme. Süreç her an bir etki alanında çalışır.
- UNIX/Linux: etki alanı = (UID, GID kümesi); ayrıca çekirdek ve kullanıcı modu iki ayrı etki alanıdır.
- Etki alanı değiştirme: `setuid` program çalıştırma, sistem çağrısıyla çekirdeğe geçiş.

Koruma Matrisi

	Dosya1	Dosya2	Dosya3	Dosya4	Dosya5	Yazıcı1	Dosya6	Çizici2	D1	D2	D3
D1	R	RW								**Gir**	
D2			R	RWX	RW	W					
D3						W	RWX	W			

- Satır = etki alanı, sütun = nesne, hücre = haklar. Etki alanları da nesne olabilir: D1 → D2 hücresindeki **Gir** (enter), D1'deki sürecin D2'ye geçebileceğini söyler.
- Matris çok **seyrek**: binlerce kullanıcı × milyonlarca dosya. Tümü saklanmaz; ya sütunlar (ACL) ya satırlar (yetenekler) saklanır.
- **Yetkili ve yetkisiz durum**: matrisin her durumu güvenlik politikasıyla karşılaştırılır. Örneğin politikaya göre Eve'in bir dosyayı okuyamaması gerekirken bir komut dizisinden sonra hücrede R belirirse sistem **yetkisiz** duruma geçmiştir.
- Harrison-Ruzzo-Ullman (HRU, 1976): genel durumda "bir hak sızabilir mi?" sorusu **karar verilemez**; bu yüzden kısıtlı modeller (BLP, Biba) kullanılır.

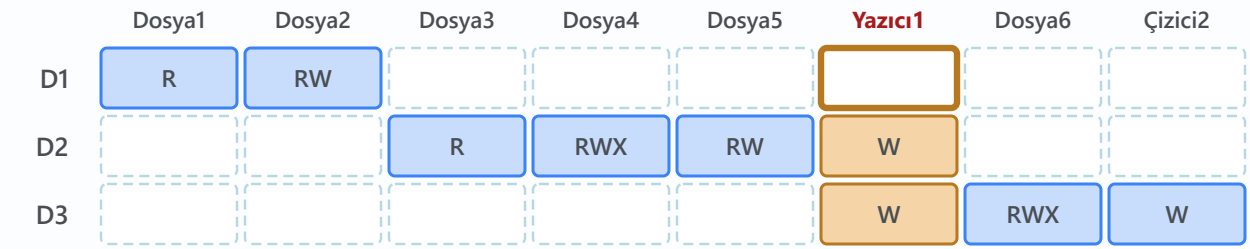
Aynı Matris İki Biçimde: ACL ve Yetenekler



24 hücreden 15'i boş: matris seyrek

Üç etki alanı ve sekiz nesnenin koruma matrisi.

Aynı Matris İki Biçimde: ACL ve Yetenekler



ACL = sütun (nesneyle saklanır)

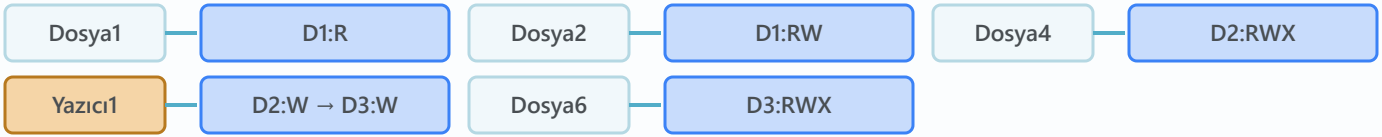


Erişim denetim listesi (ACL): matrisin bir **sütunu**; nesnenin yanında saklanır. Yazıcı1'in ACL'si: D2 yazabilir, D3 yazabilir.

Aynı Matris İki Biçimde: ACL ve Yetenekler

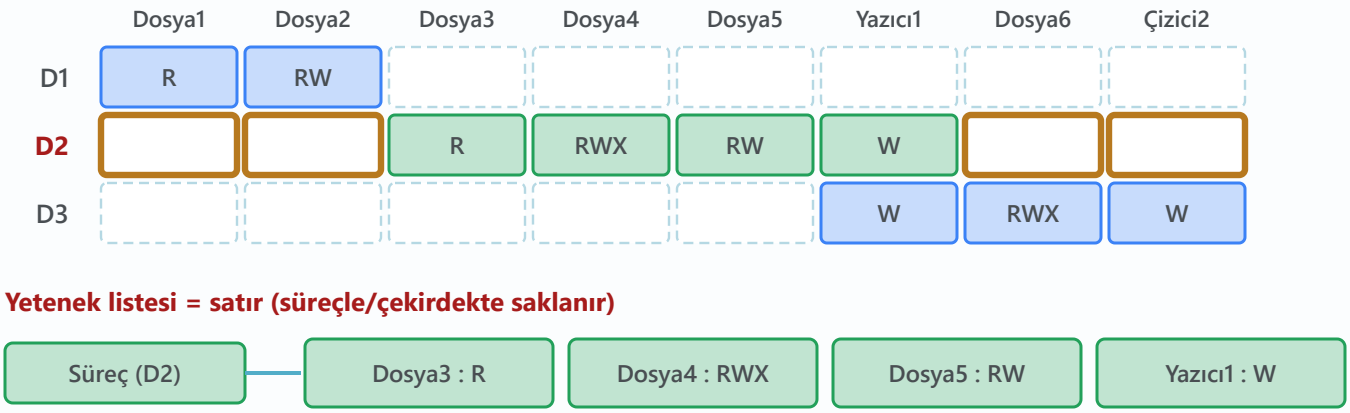
	Dosya1	Dosya2	Dosya3	Dosya4	Dosya5	Yazıcı1	Dosya6	Çizici2
D1	R	RW						
D2			R	RWX	RW	W		
D3						W	RWX	W

ACL = sütun (nesneyle saklanır)



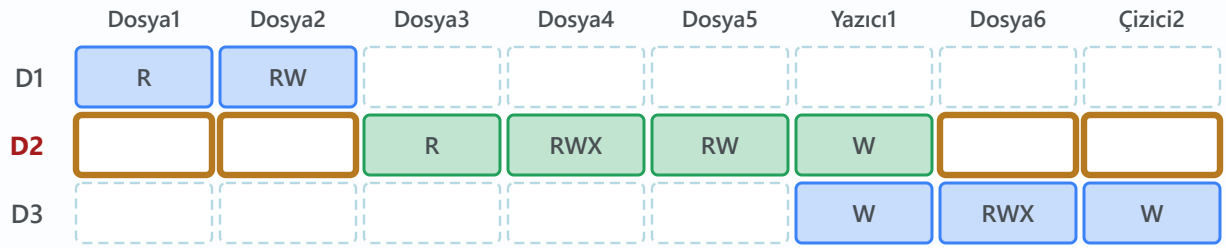
Her nesnenin boş olmayan girdileri birer ACL olur. Erişimde çekirdek, nesnenin ACL'sinde sürecin kimliğini arar.

Aynı Matris İki Biçimde: ACL ve Yetenekler



Yetenek listesi (capability, C-list): matrisin bir **satırı**; etki alanına (sürece) aittir. D2'nin yetenekleri: Dosya3:R, Dosya4:RWX, Dosya5:RW, Yazıcı1:W.

Aynı Matris İki Biçimde: ACL ve Yetenekler



Yetenek listesi = satır (süreçle/çekirdekte saklanır)



Süreç nesneye erişirken listedeki yeteneği gösterir; isim çözümleme ve ACL taraması gerekmez

Karşılaştırma: ACL'de "bu nesneye kim erişebilir?" ve **geri alma** kolaydır; yetenekte **devretme** ve hızlı denetim kolaydır ama tüm kopyaları geri almak zordur.

KORUMA

Erişim Denetim Listeleri Uygulamada

```
$ ls -l rapor.txt
-rw-r----- 1 ali muhasebe 4096 rapor.txt
 rw-  sahip   r--  grup   ---  diğer

$ setfacl -m u:ayse:rw rapor.txt
$ getfacl rapor.txt
user::rw-
user:ayse:rw-
group::r--
mask::rw-
other::---
```

Grup	A	B	C	D
Grup 1	R	RWX	—	—
Grup 2	W	—	—	—
Grup 3	—	—	X	R
Grup 4	—	—	—	—
Yönetici	RWX	RWX	RWX	RWX

- rwx 9 bitle hızlı ama kaba tanelidir; ACL adlı kullanıcı/grup ekler, `mask` bunların en fazla hakkını sınırlar (`ls` 'te `+`).
- Artı: anlaşılır, nesne başına geri alma kolay.
- Eksi: büyük sistemlerde yönetim zor; her erişimde liste taraması (önbellekle hafifletilir).
- NTFS DACL: sıralı izin/ret girdileri (SID + hak maskesi), açık ret önce, klasörden kalıtım (`icacls`). POSIX, NFSv4 ve NTFS ACL'leri tam eşlenmez.

Etkileşimli simülatör: İşletim Sistemi Koruma ve İzinler Laboratuvarı: rwx bitleri, sekizli gösterim, chmod ve erişim kararları.

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
6      for (int i = 0; i < 3; i++)
7          if (acl[i].grup == grup) {
8              int h = acl[i].hak;
9              return (h & istek) == istek;
10         }
11     return 0;   /* listede yok: yasak */
12 }
```

istek: —

acl[] (A dosyasıyla birlikte saklanır)



A dosyasının ACL'si koruma matrisinin A sütunudur: G1 okur, G2 yazar, yönetici her şeyi yapar. G3 ve G4 listede yok; boş hücreler saklanmaz.

Adım 1 / 7

DEĞİŞKENLER

`girdi sayısı = 3`

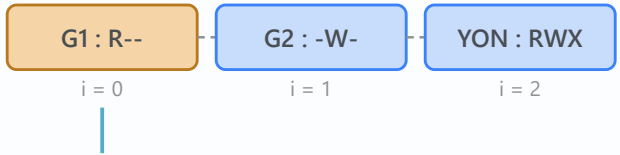
ÇIKTI

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
▶  int erisim(int grup, int istek) {
▶      for (int i = 0; i < 3; i++)
▶          if (acl[i].grup == grup) {
8              int h = acl[i].hak;
9              return (h & istek) == istek;
10         }
11     return 0;   /* listede yok: yasak */
12 }
```

istek: G2 · W (2)

acl[] (A dosyasıyla birlikte saklanır)



erisim(G2, W): i = 0'daki girdi G1'e ait, grup eşleşmez; tarama sürer.

Adım 2 / 7

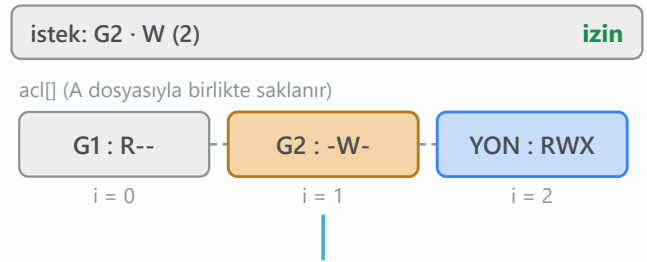
DEĞİŞKENLER

`grup = G2` `istek = W` `i = 0`

ÇIKTI

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
6      for (int i = 0; i < 3; i++)
▶          if (acl[i].grup == grup) {
▶              int h = acl[i].hak;
▶              return (h & istek) == istek;
10         }
11     return 0;   /* listede yok: yasak */
12 }
```



i = 1'de G2 bulundu: h = 2. (2 & 2) == 2 olduğu için **izin** verilir.

Adım 3 / 7

DEĞİŞKENLER

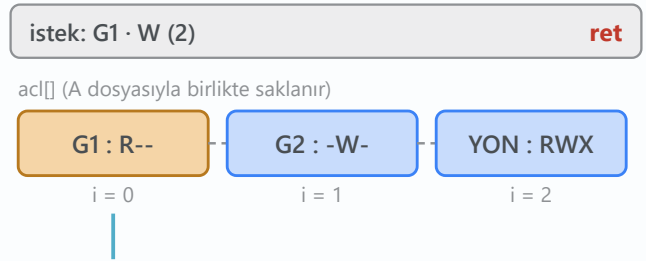
`grup = G2` `istek = W` `i = 1` `h = -W-`

ÇIKTI

```
G2 W: izin
```

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
▶      for (int i = 0; i < 3; i++)
▶          if (acl[i].grup == grup) {
▶              int h = acl[i].hak;
▶              return (h & istek) == istek;
10         }
11     return 0;   /* listede yok: yasak */
12 }
```



erisim(G1, W): ilk girdi eşleşir, h = R. 4 & 2 = 0 ≠ 2: **ret**. Karar ilk eşleşen girdide verilir, liste daha fazla taranmaz.

Adım 4 / 7

DEĞİŞKENLER

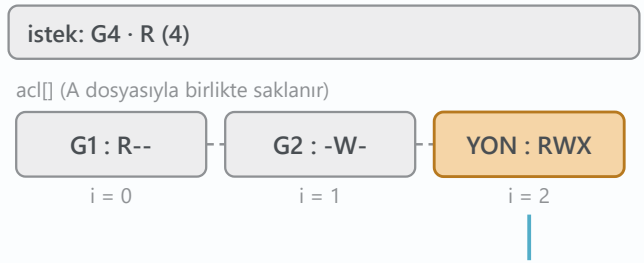
`grup = G1` `istek = W` `i = 0` `h = R--`

ÇIKTI

```
G2 W: izin
G1 W: ret
```

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
▶      for (int i = 0; i < 3; i++)
▶          if (acl[i].grup == grup) {
8              int h = acl[i].hak;
9              return (h & istek) == istek;
10         }
11     return 0;   /* listede yok: yasak */
12 }
```



erisim(G4, R): G4 hiçbir girdide yok; çekirdek listenin tamamını tarar. Uzun ACL'lerde her erişimin maliyeti budur.

Adım 5 / 7

DEĞİŞKENLER

`grup = G4` `istek = R` `i = 2`

ÇIKTI

```
G2 W: izin
G1 W: ret
```

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
6      for (int i = 0; i < 3; i++)
7          if (acl[i].grup == grup) {
8              int h = acl[i].hak;
9              return (h & istek) == istek;
10         }
▶      return 0;   /* listede yok: yasak */
12 }
```

istek: G4 · R (4) **ret**

acl[] (A dosyasıyla birlikte saklanır)

G1 : R--	G2 : -W-	YON : RWX
i = 0	i = 1	i = 2

Döngü bitti, eşleşme yok: 0 döner. **Güvenli varsayılan**: açıkça izin verilmeyen her şey yasaktır.

Adım 6 / 7

DEĞİŞKENLER

`grup = G4` `istek = R` `i = 3`

ÇIKTI

```
G2 W: izin
G1 W: ret
G4 R: ret
```

Erişim Denetim Listesi: Kod İzleme

```
1  /* A dosyasının ACL'si: R=4 W=2 X=1 */
2  struct girdi { int grup, hak; } acl[] =
3      {{G1, 4}, {G2, 2}, {YON, 7}};
4
5  int erisim(int grup, int istek) {
▶      for (int i = 0; i < 3; i++)
▶          if (acl[i].grup == grup) {
▶              int h = acl[i].hak;
▶              return (h & istek) == istek;
10         }
11     return 0;    /* listede yok: yasak */
12 }
```

istek: YON · R|X (5) — **izin**

acl[] (A dosyasıyla birlikte saklanır)

G1 : R--	G2 : -W-	YON : RWX
i = 0	i = 1	i = 2

erisim(YON, R|X): i = 2'de eşleşir, h = 7. 7 & 5 = 5: **izin**. Birden çok hak tek istekte denetlenir; hepsi varsa izin verilir.

Adım 7 / 7

DEĞİŞKENLER

`grup = YON` `istek = RX` `i = 2` `h = RWX`

ÇIKTI

```
G2 W: izin
G1 W: ret
G4 R: ret
YON RX: izin
```

KORUMA

Yetenek Listeleri



Yeteneği kopyala
Aynı nesne için yeni yetenek (devretme).

Nesneyi kopyala
Yeni nesne ve ona ait yeni yetenek.

Yeteneği kaldır
Listeden girdi silinir; nesne yerinde kalır.

Nesneyi yok et
Nesne ve yetenek kalıcı olarak silinir.

Yetenekler sahte üretilemez olmalı: ya çekirdekte tutulur (indisle erişilir), ya etiketli donanımla korunur (CHERI), ya da şifreli imzalanır. Gündelik örnekler: UNIX **dosya tanımlayıcıları** (açıldıktan sonra yeniden izin denetimi yok), FreeBSD Capsicum, seL4, Fuchsia.

En Az Ayrıcalık ve setuid

```
▶ /* /usr/bin/passwd  -rwsr-xr-x root */
2 int main(void) {
3     uid_t me = getuid();
4     int fd = open("/etc/shadow", O_RDWR);
5     update_own_entry(fd, me);
6     close(fd);
7     setresuid(me, me, me);
8     /* ... kalan iş ... */
9     fd = open("/etc/shadow", O_RDWR);
10    if (fd < 0) perror("open");
11    else close(fd);
12    return 0;
13 }
```



kabuk: ali, ayrıcalıksız

Ali (uid 1000) `passwd` çalıştırır. Dosyanın sahibi root ve **setuid** biti (s) açık.

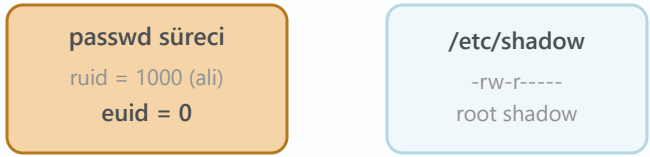
Adım 1 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 1000` `fd = –`

En Az Ayrıcalık ve setuid

```
1  /* /usr/bin/passwd  -rwsr-xr-x root */
2  int main(void) {
3      uid_t me = getuid();
4      int fd = open("/etc/shadow", O_RDWR);
5      update_own_entry(fd, me);
6      close(fd);
7      setresuid(me, me, me);
8      /* ... kalan iş ... */
9      fd = open("/etc/shadow", O_RDWR);
10     if (fd < 0) perror("open");
11     else close(fd);
12     return 0;
13 }
```



execve: euid ← dosya sahibi

`execve` sırasında etkin kimlik dosya sahibine döner: **euid = 0**. Gerçek kimlik (ruid) hâlâ Ali.

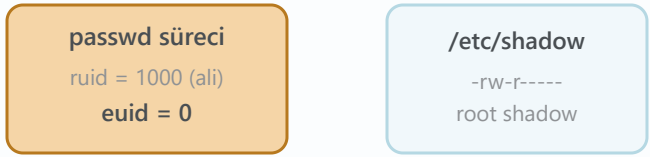
Adım 2 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 0` `fd = –`

En Az Ayrıcalık ve setuid

```
1  /* /usr/bin/passwd  -rwsr-xr-x root */
2  int main(void) {
▶      uid_t me = getuid();
4      int fd = open("/etc/shadow", O_RDWR);
5      update_own_entry(fd, me);
6      close(fd);
7      setresuid(me, me, me);
8      /* ... kalan iş ... */
9      fd = open("/etc/shadow", O_RDWR);
10     if (fd < 0) perror("open");
11     else close(fd);
12     return 0;
13 }
```



getuid() → 1000

`getuid()` gerçek kimliği döndürür: program kimin adına çalıştığını bilir ve yalnız onun satırını değiştirir.

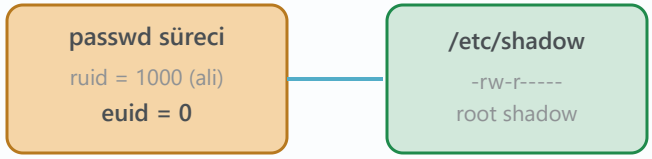
Adım 3 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 0` `me = 1000` `fd = –`

En Az Ayrıcalık ve setuid

```
1  /* /usr/bin/passwd  -rwsr-xr-x root */
2  int main(void) {
3      uid_t me = getuid();
▶      int fd = open("/etc/shadow", O_RDWR);
▶      update_own_entry(fd, me);
6      close(fd);
7      setresuid(me, me, me);
8      /* ... kalan iş ... */
9      fd = open("/etc/shadow", O_RDWR);
10     if (fd < 0) perror("open");
11     else close(fd);
12     return 0;
13 }
```



izin verildi (euid 0)

Çekirdek erişimi **etkin** kimliğe göre denetler: euid 0, dosya açılır.

Adım 4 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 0` `me = 1000` `fd = 3`

En Az Ayrıcalık ve setuid

```
1  /* /usr/bin/passwd  -rwsr-xr-x root */
2  int main(void) {
3      uid_t me = getuid();
4      int fd = open("/etc/shadow", O_RDWR);
5      update_own_entry(fd, me);
▶      close(fd);
▶      setresuid(me, me, me);
8      /* ... kalan iş ... */
9      fd = open("/etc/shadow", O_RDWR);
10     if (fd < 0) perror("open");
11     else close(fd);
12     return 0;
13 }
```

passwd süreci
ruid = 1000 (ali)
euid = 1000

/etc/shadow
-rw-r-----
root shadow

ayrıcalık kalıcı bırakıldı

İş biter bitmez `setresuid` ile gerçek, etkin ve saklı kimliğin üçü de 1000 yapılır: **en az ayrıcalık**, geri dönüş yok.

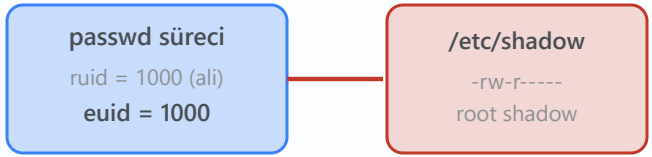
Adım 5 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 1000` `me = 1000` `fd = –`

En Az Ayrıcalık ve setuid

```
1  /* /usr/bin/passwd  -rwsr-xr-x root */
2  int main(void) {
3      uid_t me = getuid();
4      int fd = open("/etc/shadow", O_RDWR);
5      update_own_entry(fd, me);
6      close(fd);
7      setresuid(me, me, me);
8      /* ... kalan iş ... */
▶      fd = open("/etc/shadow", O_RDWR);
▶      if (fd < 0) perror("open");
11     else close(fd);
12     return 0;
13 }
```



EACCES: izin reddedildi

Programın geri kalanında bir hata istismar edilse bile saldırgan artık root değildir. setuid root programlar, ayrıcalık yükseltme saldırılarının klasik hedefidir.

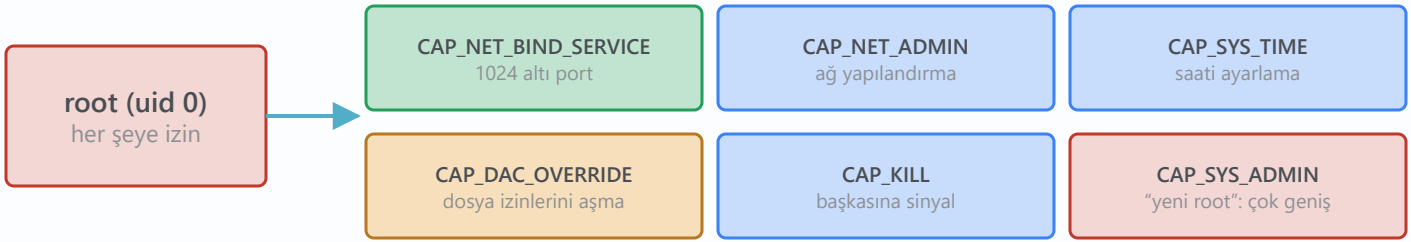
Adım 6 / 6

DEĞİŞKENLER

`ruid = 1000` `euid = 1000` `me = 1000` `fd = -1`

KORUMA

Linux Yetenekleri ve Tasarım İlkeleri

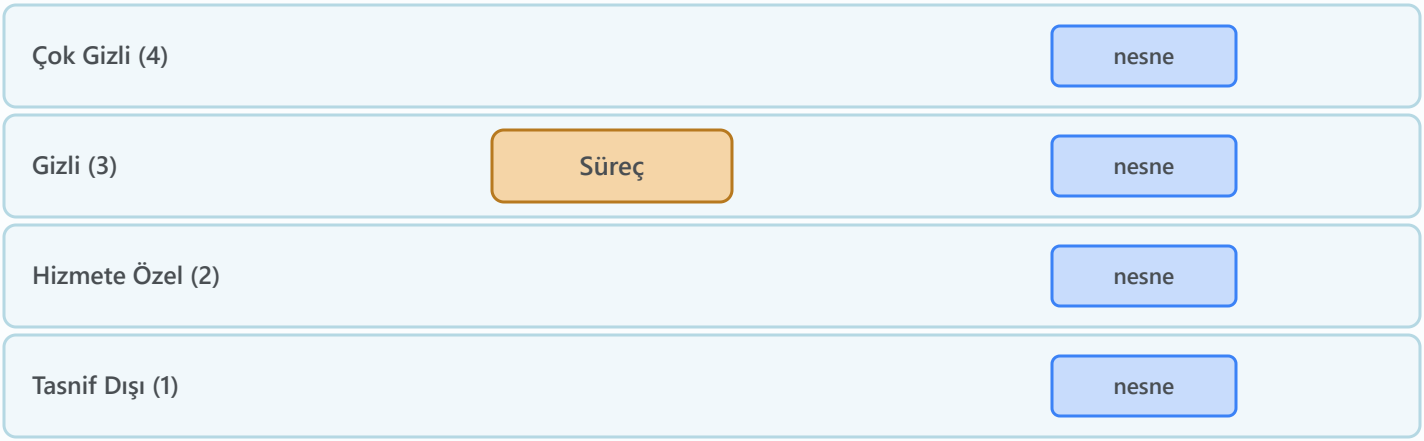


≈ 41 yetenek; her iş parçacığının izinli, etkin, kalıtılabilir ve sınırlayıcı kümeleri var

```
$ sudo setcap cap_net_bind_service=+ep /usr/local/bin/websrv
$ getcap /usr/local/bin/websrv
/usr/local/bin/websrv cap_net_bind_service=ep     # 80. portu açabilir, başka hiçbir root hakkı yok
```

- **En az ayrıcalık** (Saltzer & Schroeder, 1975): her bileşen işini yapmaya yetecek en küçük hak kümesiyle çalışmalı; ele geçirilirse zarar sınırlı kalır.
- Diğer ilkeler: güvenli varsayılan (açıkça izin verilmeyen yasak), **tam aracılık** (her erişim denetlenir), açık tasarım, ayrıcalıkların ayrılması, psikolojik kabul edilebilirlik.
- Uygulama: ayrıcalık ayrımı (OpenSSH: küçük root süreç + ayrıcalıksız çocuk), `systemd` `CapabilityBoundingSet=`, `NoNewPrivileges=`.

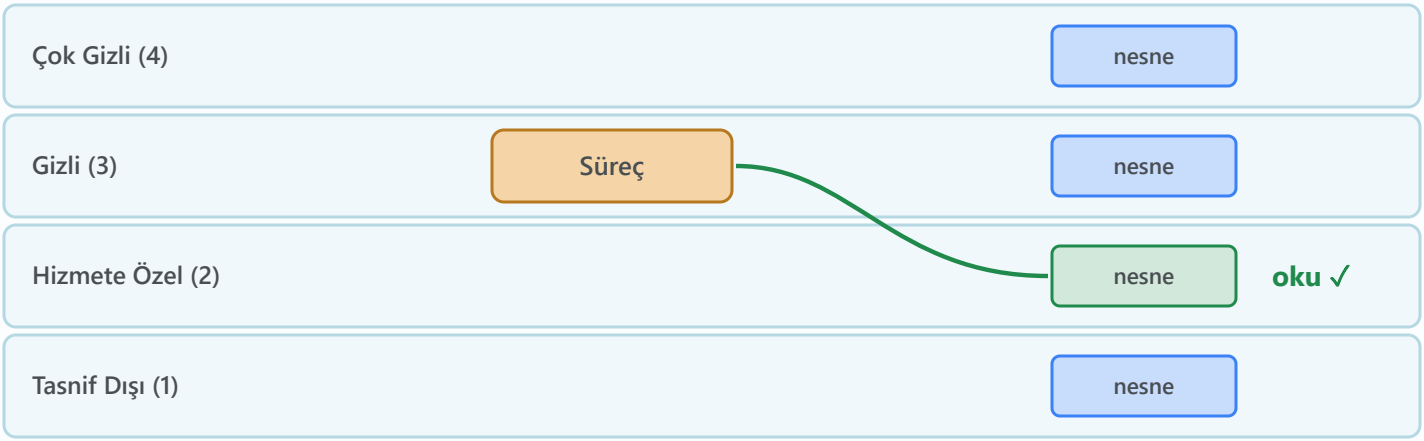
Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



Süreç “Gizli” yetkisiyle çalışıyor

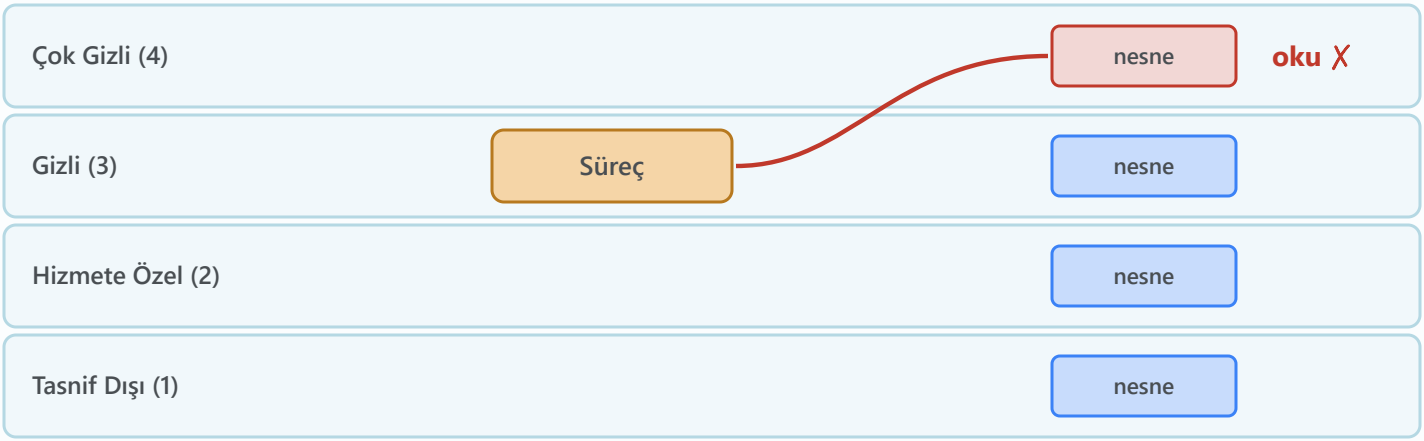
Askerî gizlilik düzeyleri; her süreç ve nesne bir düzeye sahip. Amaç: bilginin **aşağı sızmaması** (gizlilik).

Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



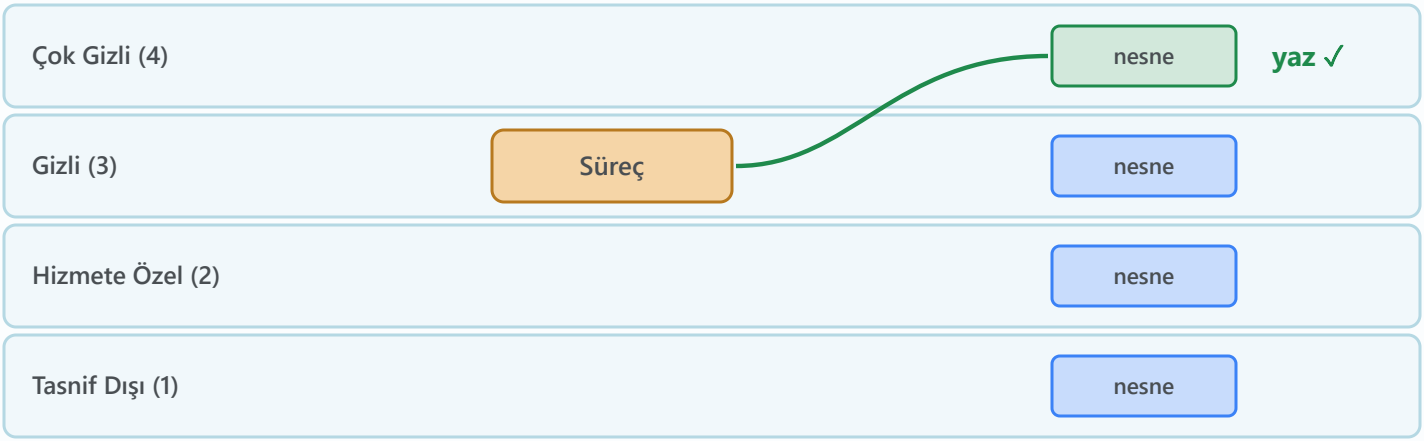
Basit güvenlik özelliği: k düzeyindeki süreç yalnız ≤ k düzeyindeki nesneleri **okuyabilir**. Aşağı okuma serbest.

Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



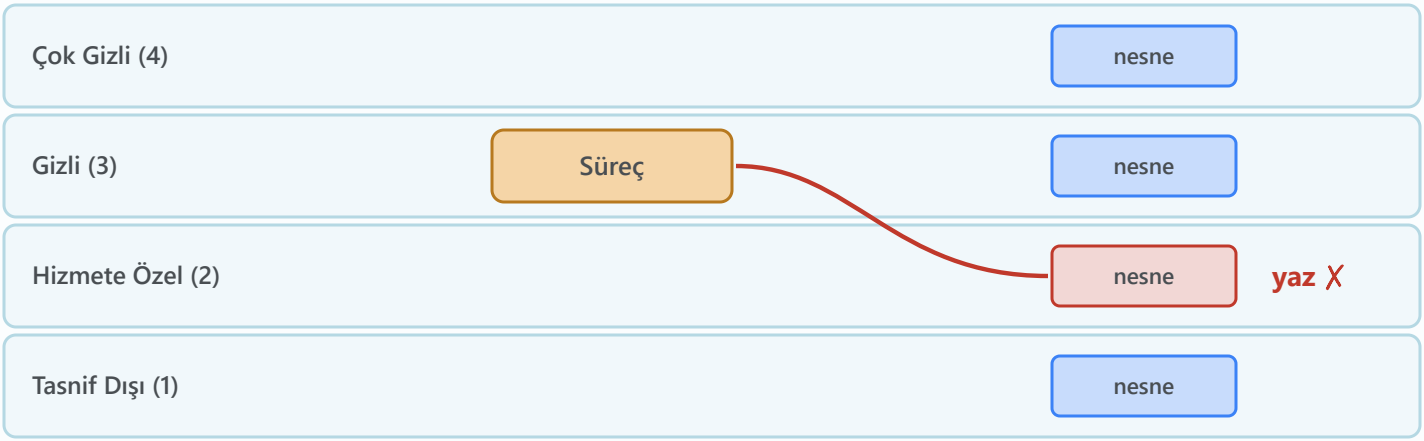
Yukarı okuma yok (no read up): Gizli yetkili süreç Çok Gizli belgeyi okuyamaz.

Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



*** özelliği**: k düzeyindeki süreç yalnız ≥ k düzeyine **yazabilir**. Yukarı yazmak gizliliği bozmaz (genelde "kör" ekleme).

Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



Aşağı yazma yok (no write down): aksi hâlde Gizli bilgi Hizmete Özel bir dosyaya kopyalanarak sızardı.

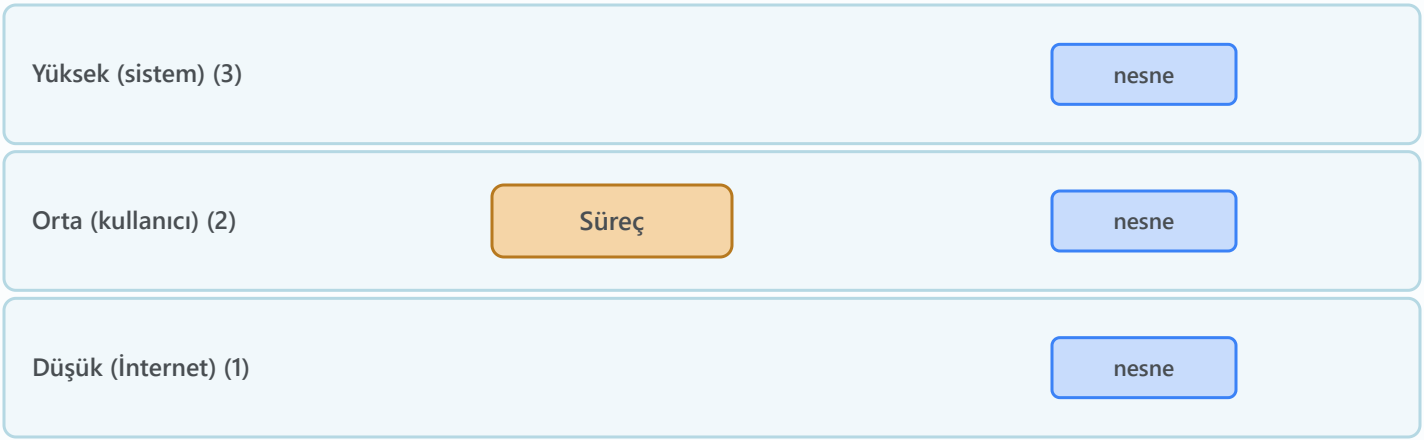
Bell-LaPadula: Yukarı Okuma Yok, Aşağı Yazma Yok



Truva atı Gizli süreç içinde çalışsa bile bilgiyi aşağı yazamaz

Kazanç: kullanıcının çalıştırdığı bir **Truva atı** bile bilgiyi aşağı aktaramaz, çünkü kural kullanıcının iyi niyetine değil sisteme bağlıdır (zorunlu erişim denetimi, MAC).

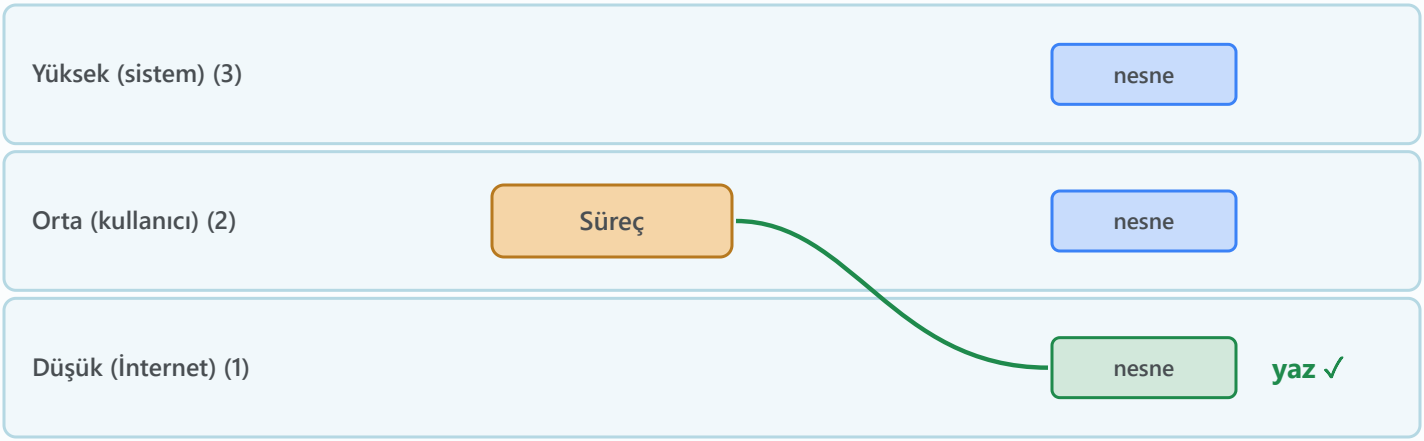
Biba: Aşağı Okuma Yok, Yukarı Yazma Yok



Süreç orta bütünlük düzeyinde

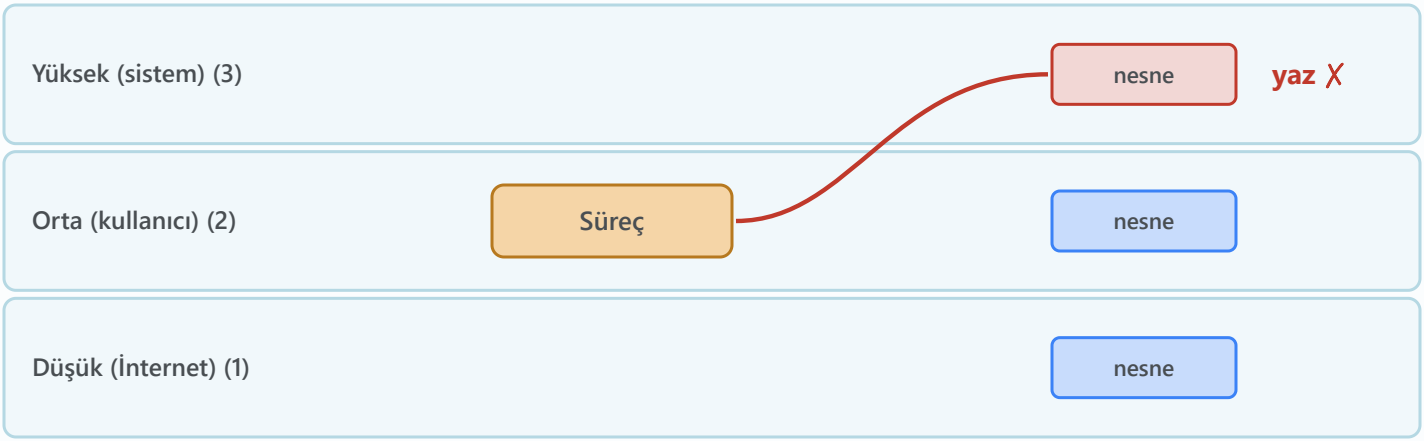
Biba (1977) BLP'nin **bütünlük** için aynasıdır: düzeyler güvenilirliği gösterir. Amaç: güvenilmeyen verinin **yukarı bulaşmaması**.

Biba: Aşağı Okuma Yok, Yukarı Yazma Yok



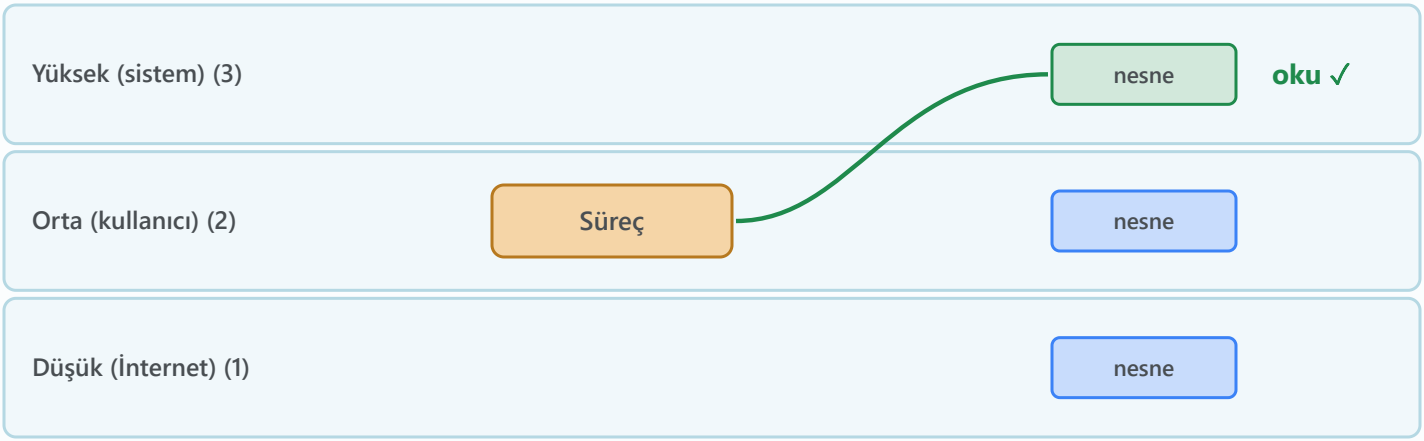
* **bütünlük özelliği**: k düzeyindeki süreç yalnız ≤ k düzeyine **yazabilir**.

Biba: Aşağı Okuma Yok, Yukarı Yazma Yok



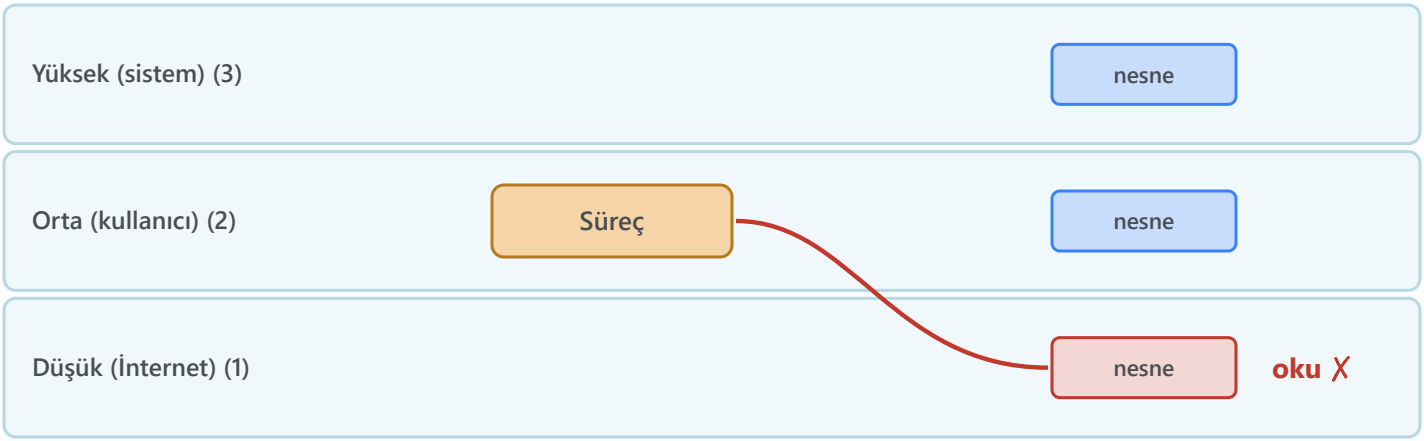
Yukarı yazma yok: kullanıcı süreci sistem dosyalarını değiştiremez.

Biba: Aşağı Okuma Yok, Yukarı Yazma Yok



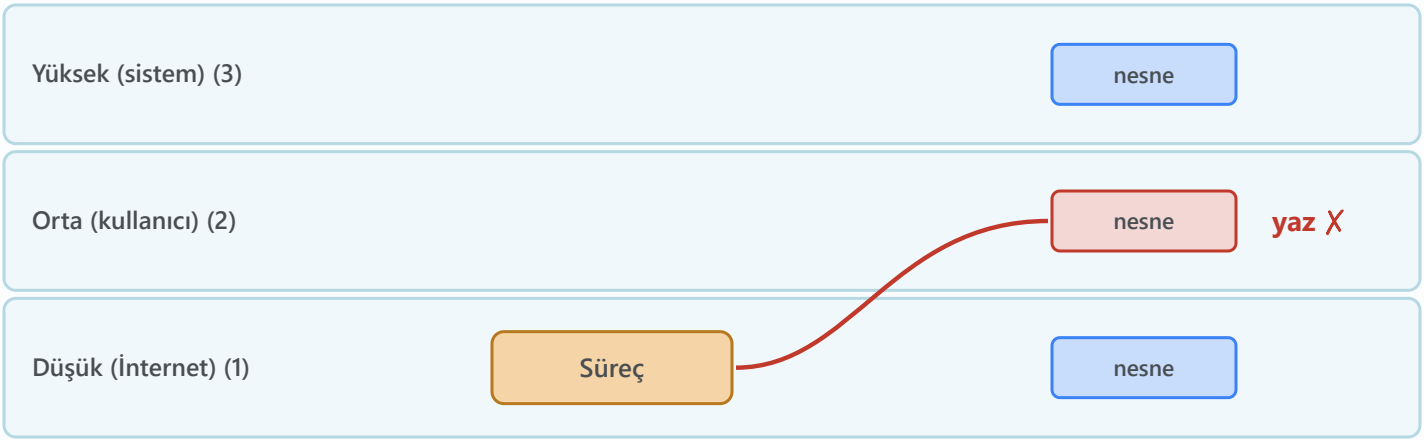
Basit bütünlük özelliği: süreç yalnız ≥ k düzeyinden **okuyabilir**; güvenilir veriyi okumak serbest.

Biba: Aşağı Okuma Yok, Yukarı Yazma Yok



Aşağı okuma yok: İnternetten gelen güvenilmeyen veriyi okuyup ona göre davranan süreç “kirlenirdi”.

Biba: Aşağı Okuma Yok, Yukarı Yazma Yok

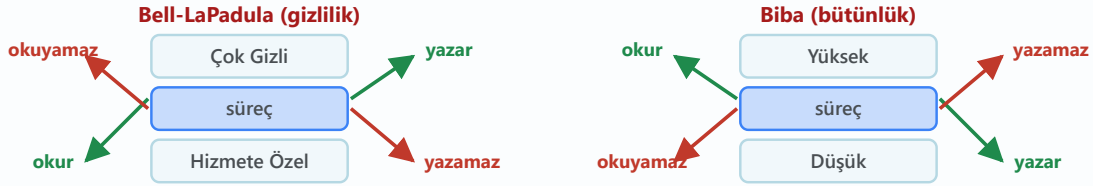


Düşük bütünlüklü tarayıcı süreci kullanıcı dosyalarına yazamaz

Gerçek örnek: Windows **zorunlu bütünlük denetimi** (MIC). Tarayıcı ve korumalı mod süreçleri Düşük düzeyde çalışır, Orta düzeydeki belgelere yazamaz (yukarı yazma yok).

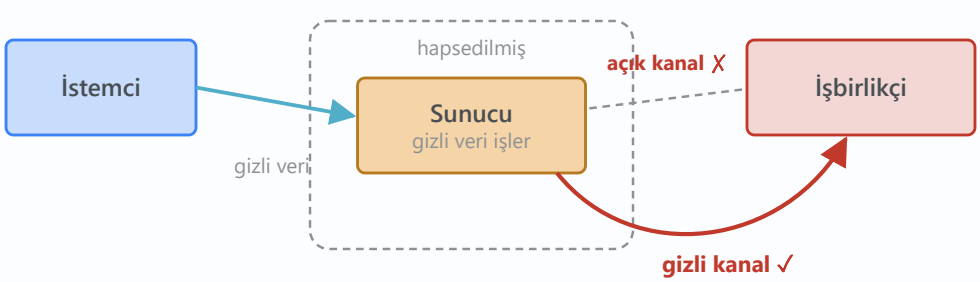
Bell-LaPadula ve Biba Karşılaştırması

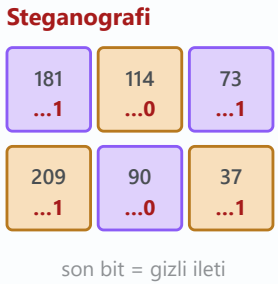
	Bell-LaPadula (1973)	Biba (1977)
Korur	Gizlilik	Bütünlük
Okuma	Aşağı serbest, **yukarı yok**	Yukarı serbest, **aşağı yok**
Yazma	Yukarı serbest, **aşağı yok**	Aşağı serbest, **yukarı yok**
Engellenen	Bilginin aşağı sızması	Kirli verinin yukarı bulaşması
Bugün	SELinux MLS, askerî sistemler	Windows MIC, FreeBSD/Linux bütünlük politikaları



- İki model birlikte kullanılırsa süreç neredeyse yalnızca kendi düzeyiyle konuşabilir: biri gizliliği, öteki bütünlüğü korur ve çatışırlar.
- Adlandırma: BLP'de **basit güvenlik özelliği** yukarı okumayı, *** özelliği** aşağı yazmayı; Biba'da (1977) **basit bütünlük özelliği** aşağı okumayı, *** bütünlük özelliği** yukarı yazmayı yasaklar.
- Her iki model de **gizli kanalları** engellemez: kurallar yalnız açık okuma/yazma işlemlerini denetler.

Gizli Kanallar ve Steganografi





Depolama kanalı

Paylaşılan bir durumun değişmesi: dosya var/yok, kilitli/açık, disk doluluğu.

Zamanlama kanalı

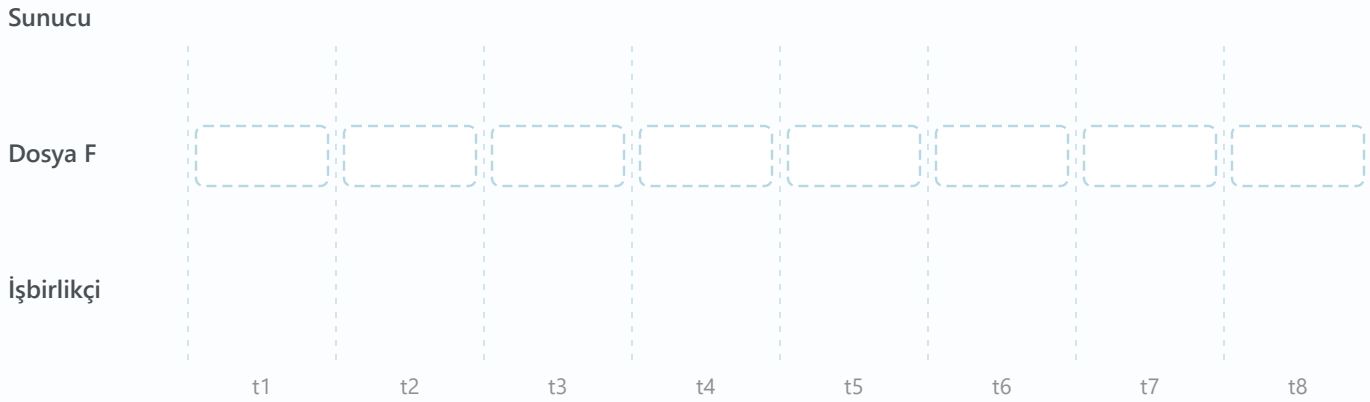
Olayların zamanlaması: CPU'yu yoğun kullanma = 1, boşta kalma = 0; önbellek gecikmesi.

Kaynak kanalı

Paylaşılan kaynağın kullanımı: sayfa hatası oranı, ağ trafiği hacmi.

Gizli kanal: tasarımcının iletişim için öngörmediği, güvenlik politikasını atlayan bilgi yolu. Lampson'ın hapsetme problemi (1973): hepsini kapatmak pratikte imkânsızdır, ancak bant genişliği düşürülebilir. Steganografi: üç zebralı fotoğrafın piksellerinin son bitlerine bir metin (ör. Shakespeare oyunları) gizlenebilir; görüntü gözle değişmez.

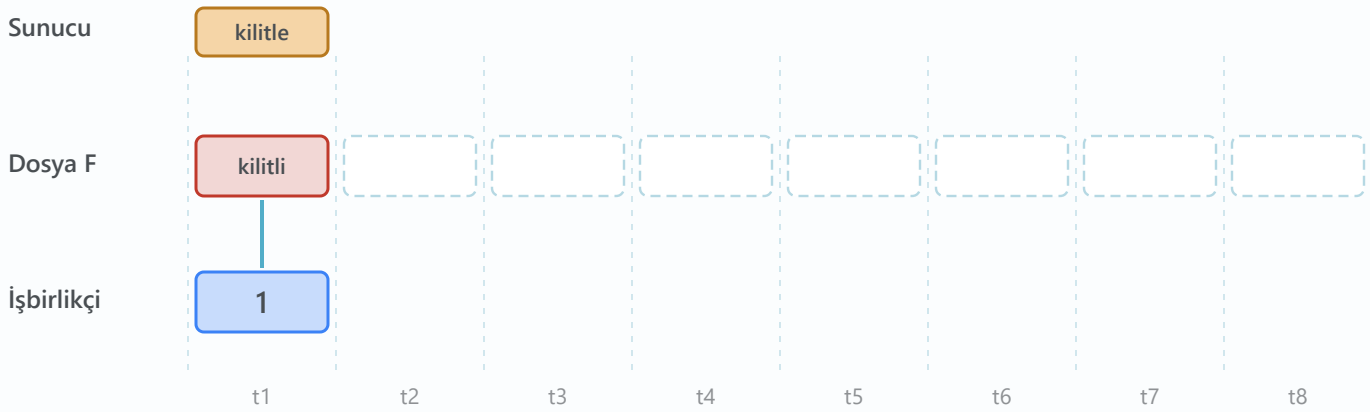
Dosya Kilidi ile Gizli Kanal



Alınan bitler: —

Sunucu ve işbirlikçi önceden anlaşır: her saat aralığında F dosyası **kilitliyse 1**, **serbestse 0**. Ortada ne okuma ne yazma var; F'nin içeriği hiç değişmez.

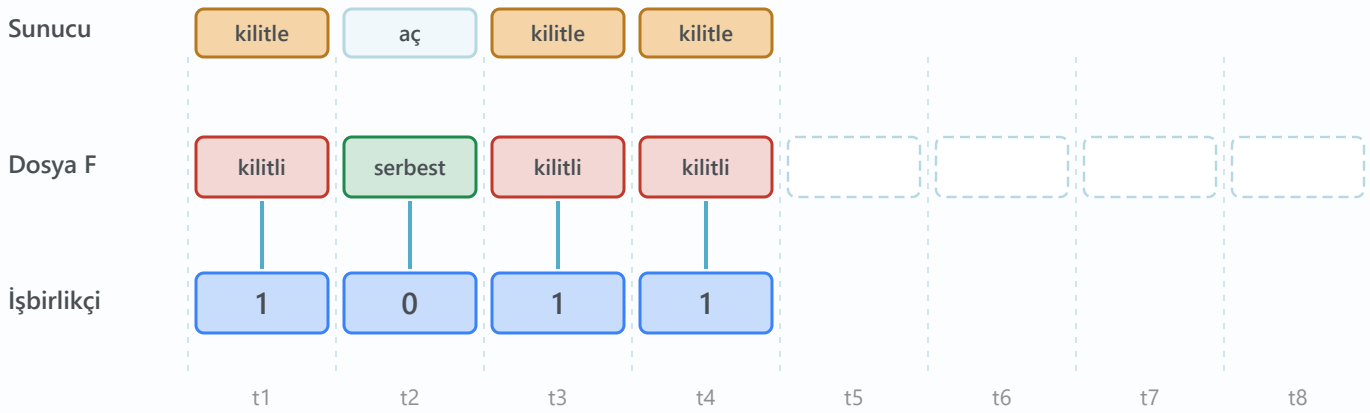
Dosya Kilidi ile Gizli Kanal



Alınan bitler: 1

Sunucu t1'de F'yi kilitler; işbirlikçi kilitlemeyi dener, başarısız olur → 1.

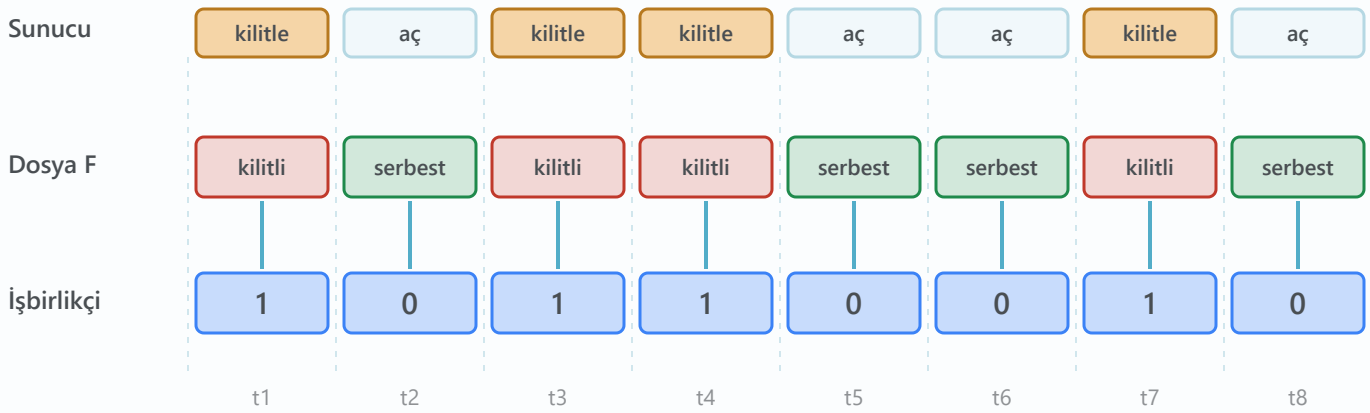
Dosya Kilidi ile Gizli Kanal



Alınan bitler: 1011

Aynı yöntemle ilk dört bit: 1 0 1 1.

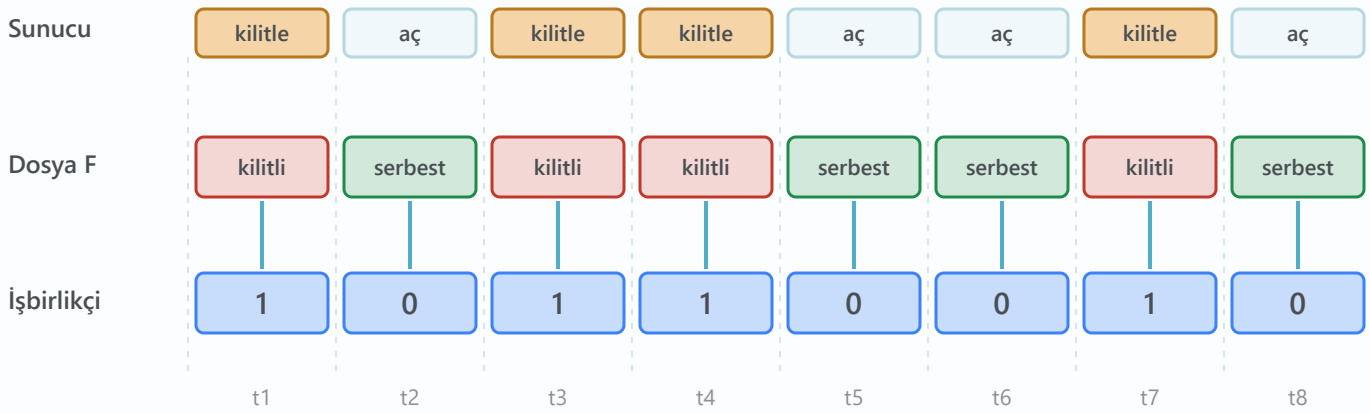
Dosya Kilidi ile Gizli Kanal



Alınan bitler: 10110010

Sekiz aralıkta bir bayt: 10110010 (0xB2). Saniyede 100 aralıkla ≈ 100 bit/s; bir parola ya da anahtar birkaç saniyede sızar.

Dosya Kilidi ile Gizli Kanal



Alınan bitler: 10110010

Önlemler: kilit durumunu başkasına göstermemek, gürültü ve gecikme eklemek, kaynakları düzeylere ayırmak. Bant genişliği azalır ama sıfırlanmaz.

KİMLİK DOĞRULAMA

Kimlik Doğrulama Etkenleri

Bildiğin bir şey

Parola, PIN, güvenlik sorusu. Ucuz ama çalınabilir, tahmin edilebilir, yeniden kullanılır.

Sahip olduğun bir şey

Telefon, akıllı kart, FIDO2 anahtarı, TOTP üreteci. Çalınabilir ama uzaktan kopyalanamaz.

Olduğun bir şey

Parmak izi, yüz, iris. Kolay ama değiştirilemez; sızarsa kalıcı olarak yanar.

Bulunduğun yer / davranış

Konum, IP, yazım ritmi, aygıt parmak izi. Genelde ek risk sinyali olarak kullanılır.

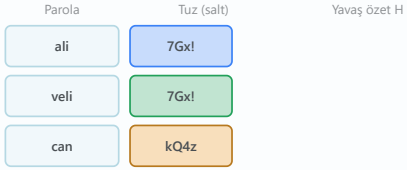
Çok etkenli kimlik doğrulama (MFA): farklı kategorilerden **en az iki** etken. "Parola + güvenlik sorusu" MFA değildir (ikisi de bilgidir). MFA, çalınan parolaya karşı en etkili tek önlemdir.

Soru-yanıt (challenge-response) doğrulamada kullanıcıya önceden kayıtlı sorular sorulur; ama "annenizin kızlık soyadı", "ilk evcil hayvanınız" gibi yanıtlar sosyal medyadan bulunabildiği için bugün zayıf kabul edilir.

Parola Saklama: Tuz ve Yavaş Özet

```
# Kayıt / parola değiştirme
salt = os.urandom(16)
h = argon2(password, salt)
store(user, salt, h)

# Giriş denemesi
salt, h = lookup(user)
g = argon2(attempt, salt)
if constant_eq(g, h): ok()
```



Parolalar **asla açık** saklanmaz. Üç kullanıcıdan ikisi aynı parolayı seçmiş (insanlar bunu sık yapar).

Adım 1 / 4

DEĞİŞKENLER

—

Düz SHA-256/MD5 parola için uygun değildir: bunlar hızlıdır, tam da parola için istenmeyen şey. 1970'lerin UNIX'i 2 karakterlik tuz + yavaşlatılmış DES (crypt) kullanırdı; fikir aynı, parametreler bugün çok daha güçlü. Sızıntı olursa yine de parolalar sıfırlanmalı.

Parola Saklama: Tuz ve Yavaş Özet

```
1 # Kayıt / parola değiştirme
▶ salt = os.urandom(16)
3 h = argon2(password, salt)
4 store(user, salt, h)
5
6 # Giriş denemesi
7 salt, h = lookup(user)
8 g = argon2(attempt, salt)
9 if constant_eq(g, h): ok()
```



Düz (tuzsuz) özet yeterli değil: aynı parola aynı özeti verir; saldırgan tekrarları görür ve **gökkuşağı tablosu** ile hepsini birden kırar.

Adım 2 / 4

DEĞİŞKENLER

—

Düz SHA-256/MD5 parola için uygun değildir: bunlar hızlıdır, tam da parola için istenmeyen şey. 1970'lerin UNIX'i 2 karakterlik tuz + yavaşlatılmış DES (crypt) kullanırdı; fikir aynı, parametreler bugün çok daha güçlü. Sızıntı olursa yine de parolalar sıfırlanmalı.

Parola Saklama: Tuz ve Yavaş Özet

```
1 # Kayıt / parola değiştirme
▶ salt = os.urandom(16)
▶ h = argon2(password, salt)
4 store(user, salt, h)
5
6 # Giriş denemesi
7 salt, h = lookup(user)
8 g = argon2(attempt, salt)
9 if constant_eq(g, h): ok()
```



Tuz (salt): her kullanıcı için rastgele, gizli olmayan bir değer parolaya eklenir. Aynı parola artık farklı girdi; önceden hesaplama işe yaramaz.

Adım 3 / 4

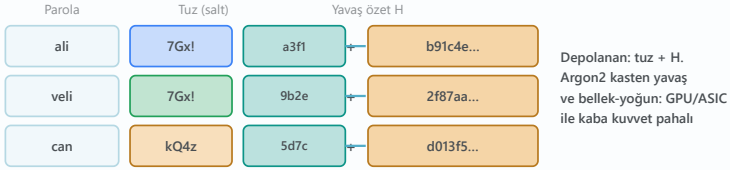
DEĞİŞKENLER

—

Düz SHA-256/MD5 parola için uygun değildir: bunlar hızlıdır, tam da parola için istenmeyen şey. 1970'lerin UNIX'i 2 karakterlik tuz + yavaşlatılmış DES (crypt) kullanırdı; fikir aynı, parametreler bugün çok daha güçlü. Sızıntı olursa yine de parolalar sıfırlanmalı.

Parola Saklama: Tuz ve Yavaş Özet

```
1 # Kayıt / parola değiştirme
2 salt = os.urandom(16)
3 h = argon2(password, salt)
4 store(user, salt, h)
5
6 # Giriş denemesi
7 salt, h = lookup(user)
▶ g = argon2(attempt, salt)
▶ if constant_eq(g, h): ok()
```



Yavaş, bellek-yoğun özet: Argon2id (ya da scrypt, bcrypt). Tek doğrulama milisaniyeler; milyarlarca denemeyse yıllar sürer. Doğrulamada **sabit zamanlı** karşılaştırma zamanlama sızıntısını önler.

Adım 4 / 4

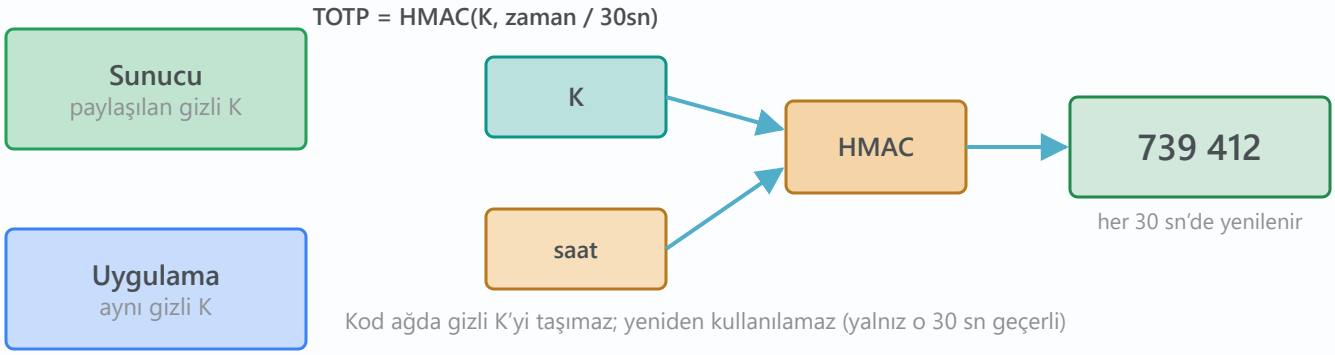
DEĞİŞKENLER

—

Düz SHA-256/MD5 parola için uygun değildir: bunlar hızlıdır, tam da parola için istenmeyen şey. 1970'lerin UNIX'i 2 karakterlik tuz + yavaşlatılmış DES (crypt) kullanırdı; fikir aynı, parametreler bugün çok daha güçlü. Sızıntı olursa yine de parolalar sıfırlanmalı.

KİMLİK DOĞRULAMA

MFA ve Zaman Tabanlı Tek Kullanımlık Parola (TOTP)



SMS kodu

Yaygın ama zayıf: SIM takası ve SS7 saldırılarıyla ele geçirilebilir. Hiç yoktan iyidir.

TOTP (üreteç uygulaması)

RFC 6238; çevrim dışı çalışır. Oltalamaya açıktır: kullanıcı sahte siteye kodu girebilir.

Anlık bildirim onayı

Kolay; ama "onay yorgunluğu" saldırısına açık. Numara eşleştirme ekler.

TOTP Kodu Üretimi: Kod İzleme

```
▶ uint32_t totp(uint8_t *K, uint64_t t) {
2     uint64_t T = t / 30;     /* 30 s */
3     uint8_t h[20];
4     hmac_sha1(K, T, h);
5     int o = h[19] & 0x0F;    /* bağıl konum
  */
6     uint32_t v = (h[o] & 0x7F) << 24
7                | h[o+1] << 16
8                | h[o+2] << 8
9                | h[o+3];
10    return v % 1000000;      /* 6 hane */
11 }
```

h[0..19]: henüz hesaplanmadı

K: kayıtta QR kodla paylaşılan 20 baytlık sır
t = 59: Unix zamanı (saniye)

Telefon ve sunucu aynı gizli **K**'yı ve aynı saati bilir. RFC 6238'in test anahtarı ("12345678901234567890") ile t = 59 s anı.

Adım 1 / 7

DEĞİŞKENLER

`t = 59` `K = 20 bayt`

ÇIKTI

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
▶      uint64_t T = t / 30;     /* 30 s */
3      uint8_t h[20];
4      hmac_sha1(K, T, h);
5      int o = h[19] & 0x0F;    /* bağıl konum
   */
6      uint32_t v = (h[o] & 0x7F) << 24
7                 | h[o+1] << 16
8                 | h[o+2] << 8
9                 | h[o+3];
10     return v % 1000000;      /* 6 hane */
11 }
```

h[0..19]: henüz hesaplanmadı

T = 59 / 30 = 1

0–29 s → 0, 30–59 s → 1, 60–89 s → 2

Zaman 30 saniyelik adımlara bölünür: T = **1**. Aynı adımdaki her an aynı kodu verir.

Adım 2 / 7

DEĞİŞKENLER

`t = 59` `T = 1`

ÇIKTI

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
2      uint64_t T = t / 30;     /* 30 s */
▶      uint8_t h[20];
▶      hmac_sha1(K, T, h);
5      int o = h[19] & 0x0F;    /* bağıl konum
   */
6      uint32_t v = (h[o] & 0x7F) << 24
7                 | h[o+1] << 16
8                 | h[o+2] << 8
9                 | h[o+3];
10     return v % 1000000;      /* 6 hane */
11 }
```

0	1	2	3	4	5	6	7	8	9
75	a4	8a	19	d4	cb	e1	00	64	4e
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
8a	c1	39	7e	ea	74	7a	2d	33	ab

h = HMAC-SHA1(K, T)

20 bayt; K'yı bilmeyen bunu üretemez

T, 8 baytlık sayı olarak K ile HMAC-SHA1'den geçer: 20 baytlık özet. Yalnız bir artan T bile tamamen farklı özet verir.

Adım 3 / 7

DEĞİŞKENLER

`T = 1` `h[19] = 0xab`

ÇIKTI

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
2      uint64_t T = t / 30;     /* 30 s */
3      uint8_t h[20];
4      hmac_sha1(K, T, h);
▶      int o = h[19] & 0x0F;    /* bağıl konum
   */
6      uint32_t v = (h[o] & 0x7F) << 24
7                 | h[o+1] << 16
8                 | h[o+2] << 8
9                 | h[o+3];
10     return v % 1000000;      /* 6 hane */
11 }
```

0	1	2	3	4	5	6	7	8	9
75	a4	8a	19	d4	cb	e1	00	64	4e
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
8a	c1	39	7e	ea	74	7a	2d	33	**ab**

h[19] = 0xab → alt 4 bit = 11

bağıl konum o = 11

Dinamik kesme: son baytın alt yarısı (0xab & 0x0F) bağıl konumu seçer: **o = 11**.

Adım 4 / 7

DEĞİŞKENLER

`T = 1` `h[19] = 0xab` `o = 11`

ÇIKTI

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
2      uint64_t T = t / 30;     /* 30 s */
3      uint8_t h[20];
4      hmac_sha1(K, T, h);
5      int o = h[19] & 0x0F;    /* bağıl konum
   */
▶      uint32_t v = (h[o] & 0x7F) << 24
▶                 | h[o+1] << 16
▶                 | h[o+2] << 8
▶                 | h[o+3];
10     return v % 1000000;      /* 6 hane */
11 }
```

0	1	2	3	4	5	6	7	8	9
75	a4	8a	19	d4	cb	e1	00	64	4e
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
8a	c1	39	7e	ea	74	7a	2d	33	ab

h[11..14] = c1 39 7e ea

en üst bit silinir → v = 1094287082

h[11]…h[14] dört baytı 32 bitlik sayı olur; en üst bit (& 0x7F) silinerek işaret sorunu önlenir: v = **1094287082**.

Adım 5 / 7

DEĞİŞKENLER

T = **1** | o = **11** | v = **1094287082**

ÇIKTI

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
2      uint64_t T = t / 30;     /* 30 s */
3      uint8_t h[20];
4      hmac_sha1(K, T, h);
5      int o = h[19] & 0x0F;    /* bağıl konum
   */
6      uint32_t v = (h[o] & 0x7F) << 24
7                 | h[o+1] << 16
8                 | h[o+2] << 8
9                 | h[o+3];
▶      return v % 1000000;      /* 6 hane */
11 }
```

0	1	2	3	4	5	6	7	8	9
75	a4	8a	19	d4	cb	e1	00	64	4e
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
8a	c1	39	7e	ea	74	7a	2d	33	ab

1094287082 mod 1000000

kod = 287082

Son 6 hane: **287082**. Telefon bunu gösterir; sunucu aynı hesabı yapıp karşılaştırır.

Adım 6 / 7

DEĞİŞKENLER

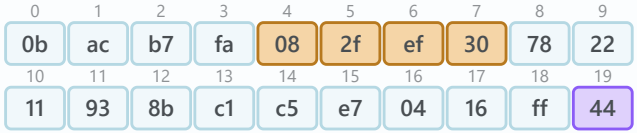
`T = 1` `v = 1094287082` `kod = 287082`

ÇIKTI

```
t=59: 287082
```

TOTP Kodu Üretimi: Kod İzleme

```
1  uint32_t totp(uint8_t *K, uint64_t t) {
▶      uint64_t T = t / 30;     /* 30 s */
3      uint8_t h[20];
▶      hmac_sha1(K, T, h);
▶      int o = h[19] & 0x0F;    /* bağıl konum
   */
6      uint32_t v = (h[o] & 0x7F) << 24
7                 | h[o+1] << 16
8                 | h[o+2] << 8
9                 | h[o+3];
▶      return v % 1000000;      /* 6 hane */
11 }
```



t = 75 → T = 2, o = 4

kod = 359152

30 saniye sonra T = 2: özet ve bağıl konum değişir, kod **359152** olur. Çalınan eski kod birkaç saniye içinde geçersizleşir; sunucu saat kayması için genelde T − 1 … T + 1'i kabul eder.

Adım 7 / 7

DEĞİŞKENLER

`t = 75` `T = 2` `o = 4` `kod = 359152`

ÇIKTI

```
t=59: 287082
t=75: 359152
```

KİMLİK DOĞRULAMA

FIDO2, WebAuthn ve Geçiş Anahtarları



İmza sitenin kökenini içerir: sahte siteye verilen yanıt gerçek sitede geçersiz → oltalanamaz

- **FIDO2/WebAuthn** ve **geçiş anahtarları** (passkeys): her site için bir açık/özel anahtar çifti. Özel anahtar aygıttan (TPM, Secure Enclave, güvenlik anahtarı) **hiç çıkmaz**.
- Sunucu yalnız **açık anahtarı** saklar: sızsa bile işe yaramaz. Paylaşılan sır yok, yeniden kullanılacak parola yok.
- İmza sayfanın **kökenine** (origin) bağlıdır → **oltalamaya dayanıklı**. Geçiş anahtarları platformlar arası senkronlanabilir; parolanın yerini almak üzere tasarlandı.

Biyometrik Kimlik Doğrulama

Ölçüt	Açıklama
Yanlış kabul (FAR)	Yabancıyı kabul etme oranı — güvenlik riski
Yanlış ret (FRR)	Gerçek kullanıcıyı reddetme oranı — kullanılabilirlik sorunu
Canlılık testi	Fotoğraf, maske, parmak kalıbına karşı (liveness)
İptal edilemezlik	Parmak izi sızarsa değiştirilemez; parola değiştirilebilir

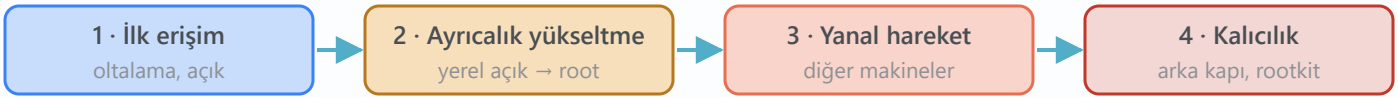
- Biyometri bir **kimlik (identity)** için iyidir, **sır (secret)** için kötüdür: yüz ve parmak izi gizli değildir.
- Doğru kullanım: aygıtta yerel olarak, güvenli donanıma erişimi açan etken (Face ID → Secure Enclave). Ham şablon aygıttan çıkmamalı.
- Tek başına uzaktan kimlik doğrulama için uygun değildir; genelde “sahip olduğun şey” ile birleştirilir.

Parmak uzunluğunu ölçen ilk aygıtlar tarihsel örneklerdir; bugünkü sistemler parmak izi çizgilerinin ayrıntı noktalarını (minutiae), 3B yüz ve iris deseni kullanır ve donanım destekli canlılık testi şarttır.

KİMLİK DOĞRULAMA

Saldırganlar Sisteme Nasıl Girer?

Yol	Açıklama	Önlem
Oltalama (phishing)	Sahte giriş sayfasıyla parola toplama	FIDO2, kullanıcı eğitimi
Kimlik bilgisi doldurma	Sızan parolaları başka sitede deneme	Benzersiz parola, MFA
Kaba kuvvet / sözlük	Zayıf parolayı tahmin	Yavaş özet, hız sınırı, kilitleme
Yama yapılmamış açık	Bilinen zafiyeti kullanma	Hızlı yama yönetimi
Sosyal mühendislik	Yardım masasını kandırma	Süreç ve doğrulama
Varsayılan/zayıf kimlik	admin/admin, açık port	Sağlamlaştırma (hardening)

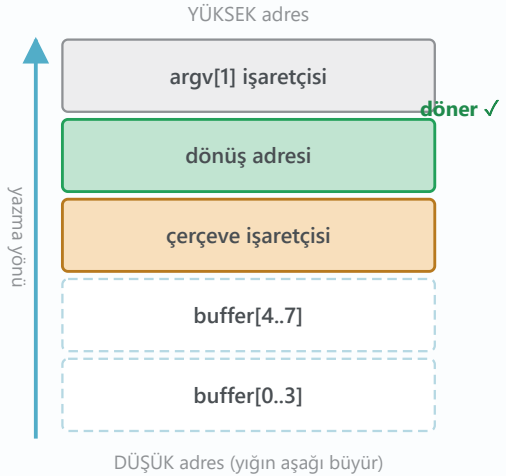


MFA ve FIDO2 · yama ve en az ayrıcalık · ağ bölütleme · EDR ve güvenli önyükleme: her halkada bir savunma

Saldırı genelde zincirdir: oltalama ile ilk erişim → yerel bir açıkla **ayrıcalık yükseltme** → yanal hareket → kalıcılık (arka kapı). Savunma da katmanlı olmalı; tek bir halkanın kopması sistemi ele geçirmemeli.

Yığın Tampon Taşması: Çerçeve Adım Adım

```
1 /* küçük, kasıtlı olarak hatalı örnek */
▶ void A(char *in) {
▶     char buffer[8];
4     strcpy(buffer, in);   // sınır denetimi YOK
5     return;
6 }
7 A(argv[1]);               // in: kullanıcı girdisi
```



Her fonksiyon çağrısı yığında bir **çerçeve** açar: yerel değişkenler, kayıtlı çerçeve işaretçisi ve **dönüş adresi** (fonksiyon bitince CPU'nun gideceği yer).

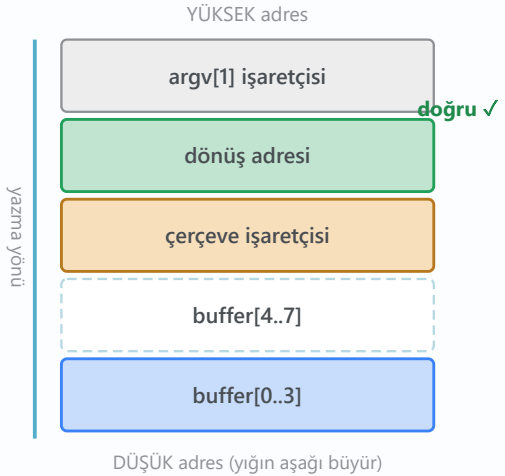
Adım 1 / 4

DEĞİŞKENLER

—

Yığın Tampon Taşması: Çerçeve Adım Adım

```
1 /* küçük, kasıtlı olarak hatalı örnek */
2 void A(char *in) {
3     char buffer[8];
▶     strcpy(buffer, in);   // sınır denetimi YOK
5     return;
6 }
7 A(argv[1]);              // in: kullanıcı girdisi
```



Girdi tampona sığdığı sürece sorun yok: `strcpy` sonlandırıcı `\0` dahil en çok sekiz baytı `buffer`'a yazar.

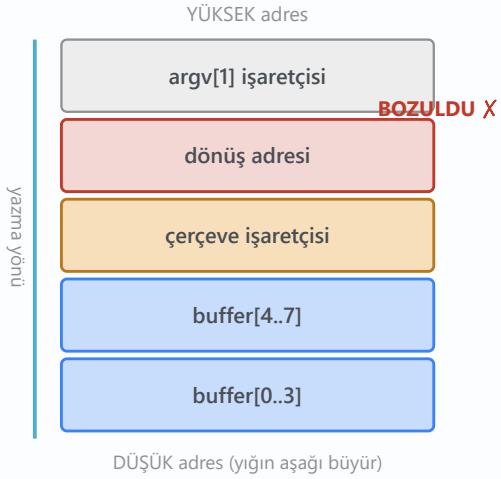
Adım 2 / 4

DEĞİŞKENLER

—

Yığın Tampon Taşması: Çerçeve Adım Adım

```
1 /* küçük, kasıtlı olarak hatalı örnek */
2 void A(char *in) {
3     char buffer[8];
▶     strcpy(buffer, in);   // sınır denetimi YOK
5     return;
6 }
7 A(argv[1]);              // in: kullanıcı girdisi
```



Taşma: `strcpy` uzunluğu denetlemediği için fazla baytlar komşu hücrelere, en tehlikelisi **dönüş adresinin** üzerine taşar.

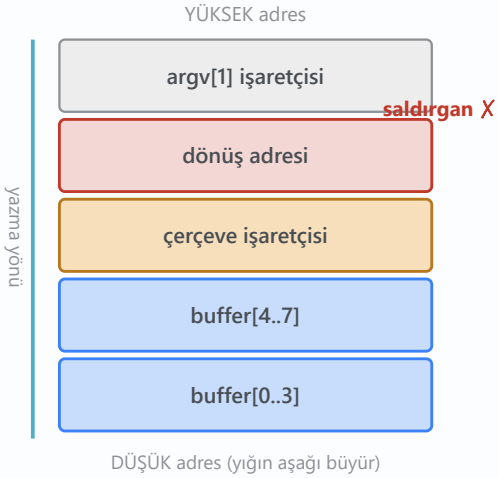
Adım 3 / 4

DEĞİŞKENLER

—

Yığın Tampon Taşması: Çerçeve Adım Adım

```
1 /* küçük, kasıtlı olarak hatalı örnek */
2 void A(char *in) {
3     char buffer[8];
4     strcpy(buffer, in);   // sınır denetimi YOK
▶     return;
6 }
7 A(argv[1]);               // in: kullanıcı girdisi
```



Fonksiyon `return` ettiğinde CPU bozulmuş adrese atlar. Saldırgan bunu seçebilirse **yürütme akışını ele geçirir**. Bu yüzden asla güvenilmeyen girdi sınır denetimi olmadan kopyalanmaz.

Adım 4 / 4

DEĞİŞKENLER

—

YAZILIM AÇIKLARI

Tampon Taşmasına Karşı Savunmalar

Yığın kanaryası
Dönüş adresinden önce rastgele bir değer; dönüşten önce bozulduysa program durdurulur (SSP).

DEP / NX
Veri sayfaları yürütülemez işaretlenir; yığına enjekte edilen kod çalışamaz.

ASLR
Yığın, heap ve kütüphaneler rastgele adreslere; saldırgan hedef adresi bilemez. PIE ile birlikte.

CFI
Denetim akışı bütünlüğü: dolaylı atlamalar yalnız geçerli hedeflere izin verir.

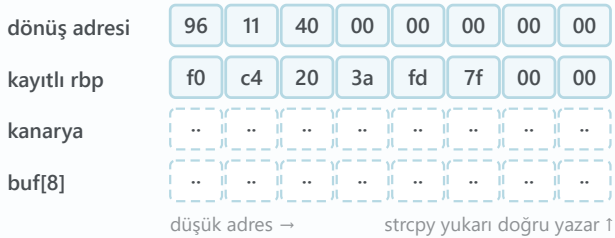
Gölge yığın / CET
Dönüş adresinin donanım korumalı kopyası: Intel CET, ARM GCS. (ARM PAC/BTI ayrı korumalardır.)

Bellek-güvenli dil
Rust, Go, Java, C#: sınır ve ömür denetimi derleyici/ çalışma zamanında. En köklü çözüm.

ROP ve return-to-libc: NX kodu enjekte etmeyi engelleyince saldırganlar **var olan** kod parçalarını (libc'deki `system()` ya da küçük "gadget" dizileri) dönüş adreslerini zincirleyerek çalıştırır. CFI, gölge yığın ve ASLR bu saldırıları zorlaştırmak için birlikte kullanılır — tek bir savunma yeterli değildir.

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
▶ void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
5     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



çağrı: dönüş adresi ve rbp yığına kondu

kopyala çağrıldı: dönüş adresi (0x401196) ve çağıranın çerçeve işaretçisi yığında. Girdi 16 karakter, buf ise 8 bayt.

Adım 1 / 7

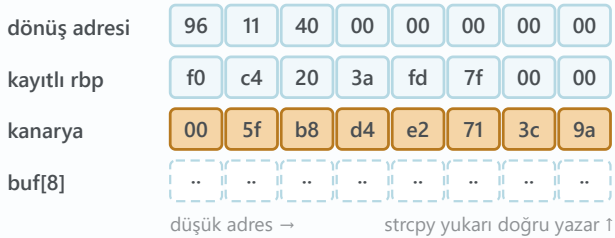
DEĞİŞKENLER

`girdi = 16 karakter`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
▶     long kanarya = __stack_chk_guard;
▶     char buf[8];
5     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



kanarya = rastgele; ilk baytı 00

Derleyicinin eklediği giriş kodu, süreç başında rastgele seçilen değeri buf ile kayıtlı rbp arasına koyar. İlk baytı 0x00: dizgi işlevleri kanaryayı ne okuyup sızdırabilir ne de aynen yazabilir.

Adım 2 / 7

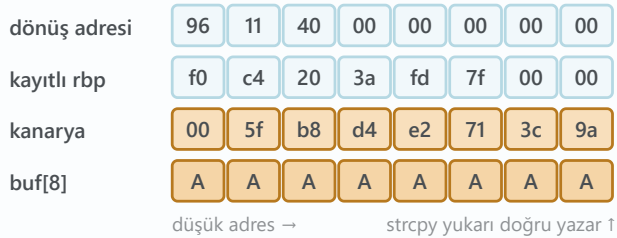
DEĞİŞKENLER

`kanarya = 0x9a3c71e2d4b85f00`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
▶     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



strcpy: 8 bayt buf'a sığdı

strcpy uzunluk bilmez, kaynaktaki \0'a kadar kopyalar. İlk 8 bayt (A) buf'u doldurur.

Adım 3 / 7

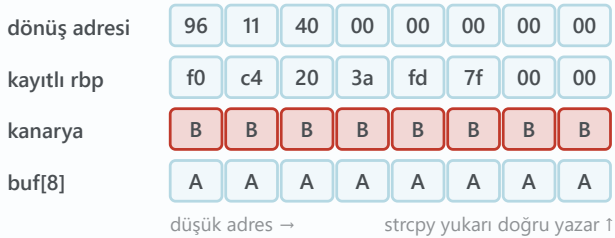
DEĞİŞKENLER

`kanarya = 0x9a3c71e2d4b85f00` `yazılan = 8`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
▶     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



taşma: kanarya ezildi

Sonraki 8 bayt (B) buf'un dışına, kanaryanın üzerine yazılır: kanarya artık 0x4242424242424242.

Adım 4 / 7

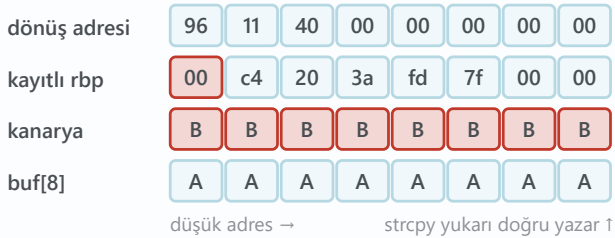
DEĞİŞKENLER

`kanarya = 0x4242424242424242` `yazılan = 16`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
▶     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



sonlandırıcı \0 kayıtlı rbp'ye taştı

Sonlandırıcı \0 kayıtlı rbp'nin ilk baytını bozar. Girdi biraz daha uzun olsaydı sıra dönüş adresine gelirdi; ama dönüş adresine ulaşmanın tek yolu kanaryanın üzerinden geçmektir.

Adım 5 / 7

DEĞİŞKENLER

`kanarya = 0x4242424242424242` `yazılan = 17`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
5     strcpy(buf, girdi);
▶     if (kanarya != __stack_chk_guard)
7         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



0x4242424242424242 ≠ 0x9a3c71e2d4b85f00

Dönmeden önce çıkış kodu yığındaki kanaryayı süreç başındaki değerle karşılaştırır: farklı.

Adım 6 / 7

DEĞİŞKENLER

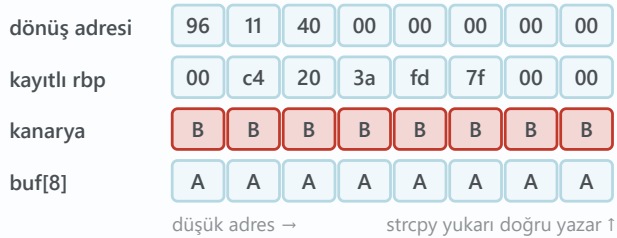
`kanarya = 0x4242424242424242`

`guard = 0x9a3c71e2d4b85f00`

ÇIKTI

Yığın Kanaryası: Kod İzleme

```
1 /* kanarya satırlarını derleyici ekler */
2 void kopyala(const char *girdi) {
3     long kanarya = __stack_chk_guard;
4     char buf[8];
5     strcpy(buf, girdi);
6     if (kanarya != __stack_chk_guard)
▶         __stack_chk_fail();
8 }
9 /* girdi: "AAAAAAAABBBBBBBB" */
```



return hiç çalışmaz: süreç durduruldu

__stack_chk_fail iletiyi yazar ve süreci SIGABRT ile sonlandırır. Bozulmuş çerçeveye hiç dönülmez; saldırı bir çökmeye dönüşür (erişilebilirlik kaybı, ama kod yürütme yok).

Adım 7 / 7

DEĞİŞKENLER

`kanarya = 0x4242424242424242` `sonuç = SIGABRT`

ÇIKTI

```
*** stack smashing detected ***: terminated
```

YAZILIM AÇIKLARI

Diğer Bellek ve Enjeksiyon Hataları

Açık	Neden tehlikeli	Önlem
Biçim dizgisi	`printf(user)` — %n/%x ile belleği okur/yazar	`printf("%s", user)`
Tam sayı taşması	`len = a + b` sarması küçük ayırmaya yol açar	Denetimli aritmetik, boyut tipleri
Serbest sonrası kullanım	Serbest bırakılan belleğe erişim (UAF)	Sahiplik (Rust), smart pointer, karantina
Çifte serbest bırakma	Aynı blok iki kez free; heap bozulur	free sonrası NULL; güçlendirilmiş ayırıcı
Sınır dışı okuma	Heartbleed gibi bellek sızıntısı	Sınır denetimi, fuzzing
Komut/SQL enjeksiyonu	Girdi koda/sorguya karışır	Parametreli sorgu, kaçış, allowlist

Ortak kök neden: **veri ile kodun/denetimin karışması** ve güvenilmeyen girdinin denetimsiz kullanımı. Savunma: girdi doğrulanır, veri koddan ayrılır, bellek-güvenli diller ve **fuzzing** hataları erken yakalar.

TOCTOU Yarış Durumu

```
1 /* TOCTOU: setuid root yardımcı */
▶ if (access(path, W_OK) == 0) {   // KONTROL
3     fd = open(path, O_WRONLY);     // KULLANIM
4     write(fd, data, n);
5 }
```



Program önce `access()` ile “ali bu dosyaya yazabilir mi?” diye **kontrol** eder: `/tmp/f` sıradan, izinli bir dosya. Sonuç olumlu.

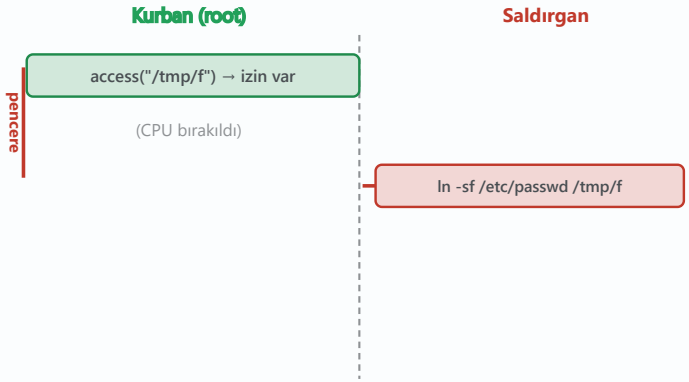
Adım 1 / 5

DEĞİŞKENLER

—

TOCTOU Yarış Durumu

```
1 /* TOCTOU: setuid root yardımcı */
▶ if (access(path, W_OK) == 0) {    // KONTROL
3     fd = open(path, O_WRONLY);      // KULLANIM
4     write(fd, data, n);
5 }
```



Kontrol ile kullanım arasında küçük bir **zaman penceresi** var. Program tam burada CPU'yu bırakırsa...

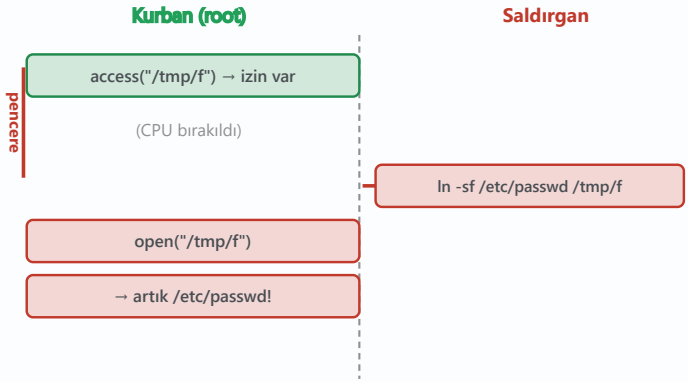
Adım 2 / 5

DEĞİŞKENLER

—

TOCTOU Yarış Durumu

```
1 /* TOCTOU: setuid root yardımcı */
▶ if (access(path, W_OK) == 0) {   // KONTROL
▶     fd = open(path, O_WRONLY);     // KULLANIM
4     write(fd, data, n);
5 }
```



Saldırgan `/tmp/f`'yi silip yerine `/etc/passwd`'e **sembolik bağ** koyar. Dosya adı aynı, hedef değişti.

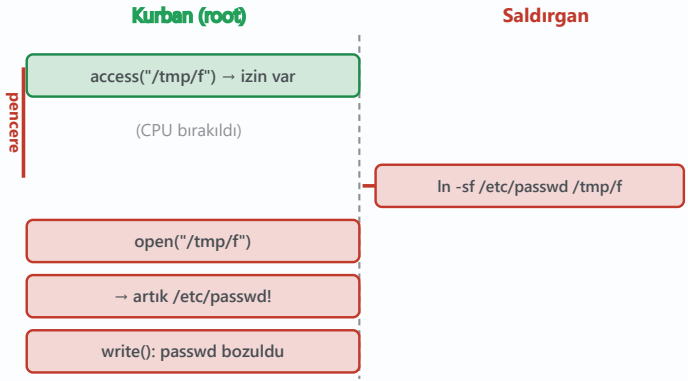
Adım 3 / 5

DEĞİŞKENLER

—

TOCTOU Yarış Durumu

```
1 /* TOCTOU: setuid root yardımcı */
2 if (access(path, W_OK) == 0) {    // KONTROL
▶     fd = open(path, O_WRONLY);      // KULLANIM
▶     write(fd, data, n);
5 }
```



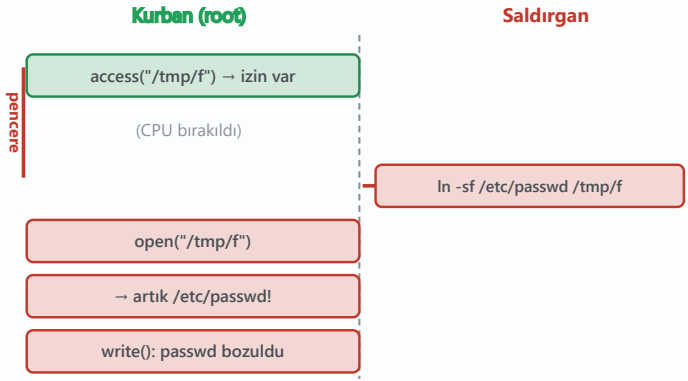
`open()` artık adı değil **yeni hedefi** açar: root yetkisiyle `/etc/passwd`. **Kontrol zamanı ≠ kullanım zamanı** (TOCTOU).

DEĞİŞKENLER

—

TOCTOU Yarış Durumu

```
1 /* TOCTOU: setuid root yardımcı */
▶ if (access(path, W_OK) == 0) {   // KONTROL
▶     fd = open(path, O_WRONLY);     // KULLANIM
4     write(fd, data, n);
5 }
```



Çözüm: ad iki kez çözülmez. Dosya bir kez açılır (`open` + `O_NOFOLLOW`), sonra **aynı tanımlayıcı** üzerinde `fstat` / `faccessat` ile denetlenir; ya da `openat` ve ayrıcalık düşürme kullanılır.

DEĞİŞKENLER

—

İçeriden Saldırılar: Mantık Bombası, Arka Kapı

(a) Normal kod

```
if (user_ok(name, pw))
grant_access();
else
deny();
```

(b) Arka kapılı kod

```
if (user_ok(name, pw)
|| eq(name, "zzz"))
grant_access();
else
deny();
```

"zzz" her zaman girer

Mantık bombası

Belirli koşulda (tarih, işten çıkarma) tetiklenen zararlı kod. Çoğu kez hoşnutsuz çalışan koyar.

Arka kapı (back door)

Kimlik doğrulamayı atlayan gizli giriş. Tedarik zincirine de sokulabilir.

Giriş sahtekârlığı

Sahte oturum açma ekranı kullanıcı adı/parolayı toplar, sonra gerçek ekrana devreder.

Giriş sahtekârlığına (login spoofing) karşı **güvenli dikkat dizisi** (secure attention key): Windows'ta Ctrl+Alt+Del yalnız gerçek çekirdek giriş ekranını çağırır; hiçbir kullanıcı programı onu taklit edemez. Arka kapılara karşı: kod incelemesi, en az iki kişi onayı, yeniden üretilebilir derleme (reproducible builds).

KÖTÜ AMAÇLI YAZILIM

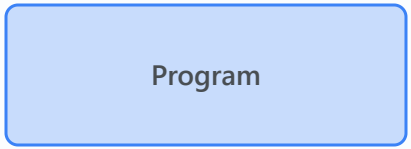
Kötü Amaçlı Yazılım Türleri

Tür	Yayılma	Amaç / Davranış
Virüs	Başka programa kendini ekler; kullanıcı çalıştırınca bulaşır	Dosyaları bozar, yayılır
Solucan (worm)	Ağ üzerinden kendi kendine; kullanıcı gerekmez	Hızlı yayılır, kaynak tüketir
Truva atı	Yararlı görünen programın içinde	Arka kapı açar, veri çalar
Fidye yazılımı	Oltalama, açık, RDP	Dosyaları şifreler, fidye ister
Rootkit	Ayrıcalık kazandıktan sonra kurulur	Varlığını ve diğer zararlıları gizler
Casus yazılım	Truva atı, paket yazılım, sahte eklenti	Veri toplar, uzağa gönderir
Bot / botnet	Solucan + arka kapı	Uzaktan komutla DDoS, spam, kazım

Sınırlar bulanıktır: modern zararlı genelde birkaç türü birleştirir (ör. solucan gibi yayılan, rootkit ile gizlenen, fidye yazılımı taşıyan bir bot). Casus yazılım tarayıcı ana sayfasını/arama motorunu değiştirebilir, araç çubuğu ekleyebilir ve durdurulamayan reklam akışı üretebilir.

Parazitik Virüsler

(a) temiz program



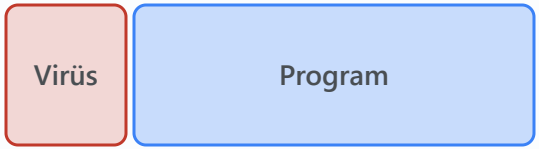
Giriş noktası virüse yönlendirilir; virüs çalışır, sonra asıl programa atlar

Temiz çalıştırılabilir dosya: baştan sona asıl program kodu.

Diğer türler: eşlik eden virüs (aynı adlı .exe'den önce çalışan bir .com dosyası), bellekte yerleşik, önyükleme sektörü, aygıt sürücüsü, makro (belge içi) ve kaynak kodu virüsleri. Çoğu bugün tarihseldir; modern zararlı betik, bellek-içi (fileless) ve tedarik zinciri yollarını kullanır.

Parazitik Virüsler

(b) başta yerleşik



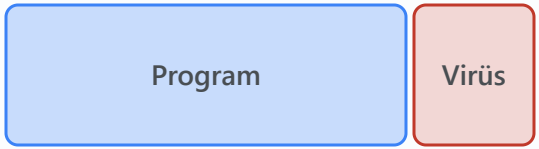
Giriş noktası virüse yönlendirilir; virüs çalışır, sonra asıl programa atlar

Başta yerleşik virüs: virüs kodu başa eklenir, giriş noktası virüse çevrilir.

Diğer türler: eşlik eden virüs (aynı adlı .exe'den önce çalışan bir .com dosyası), bellekte yerleşik, önyükleme sektörü, aygıt sürücüsü, makro (belge içi) ve kaynak kodu virüsleri. Çoğu bugün tarihseldir; modern zararlı betik, bellek-içi (fileless) ve tedarik zinciri yollarını kullanır.

Parazitik Virüsler

(c) sonda yerleşik



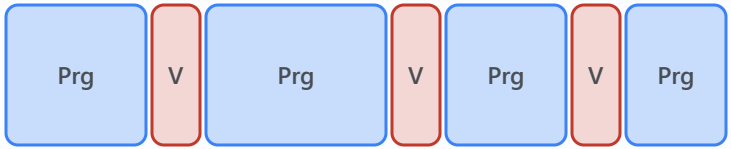
Giriş noktası virüse yönlendirilir; virüs çalışır, sonra asıl programa atlar

Sonda yerleşik virüs: virüs sona eklenir, başa ona atlayan bir komut konur (en yaygın).

Diğer türler: eşlik eden virüs (aynı adlı .exe'den önce çalışan bir .com dosyası), bellekte yerleşik, önyükleme sektörü, aygıt sürücüsü, makro (belge içi) ve kaynak kodu virüsleri. Çoğu bugün tarihseldir; modern zararlı betik, bellek-içi (fileless) ve tedarik zinciri yollarını kullanır.

Parazitik Virüsler

(d) boşluklara yayılmış

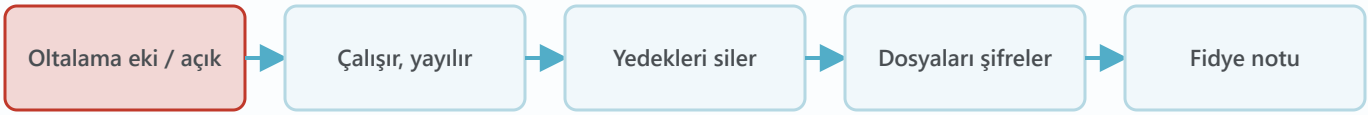


Giriş noktası virüse yönlendirilir; virüs çalışır, sonra asıl programa atlar

Boşluklara yayılmış (cavity) virüs: programdaki kullanılmayan boşluklara dağılır; dosya boyutu değişmez, tespiti zorlaşır.

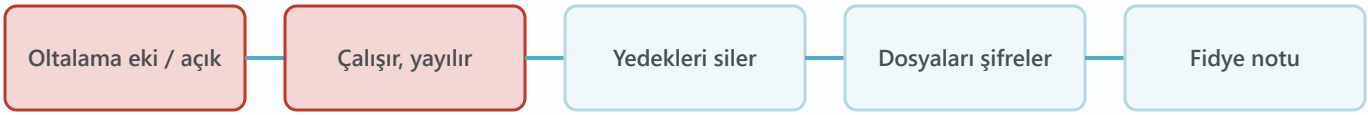
Diğer türler: eşlik eden virüs (aynı adlı .exe'den önce çalışan bir .com dosyası), bellekte yerleşik, önyükleme sektörü, aygıt sürücüsü, makro (belge içi) ve kaynak kodu virüsleri. Çoğu bugün tarihseldir; modern zararlı betik, bellek-içi (fileless) ve tedarik zinciri yollarını kullanır.

Fidye Yazılımı



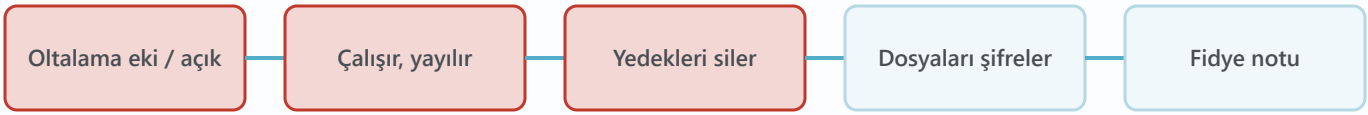
İlk erişim genelde oltalama eki, açık bir hizmet ya da çalınan RDP kimliğiyle olur.

Fidye Yazılımı



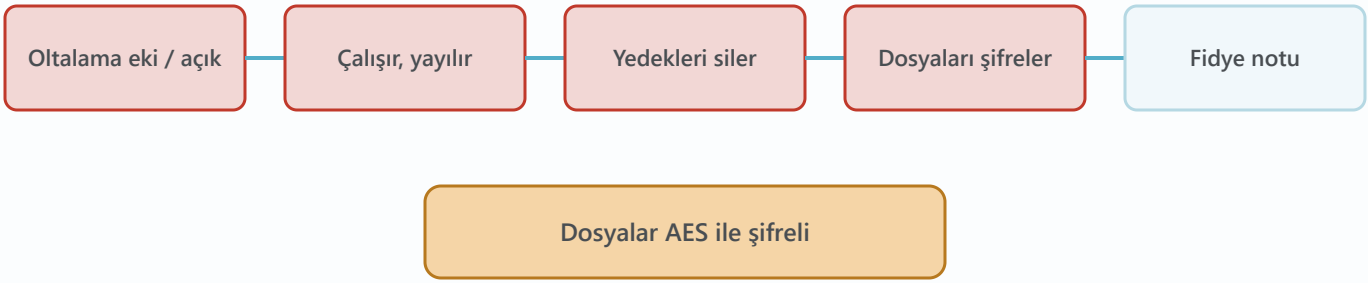
Zararlı çalışır ve ağda yanal hareket ederek olabildiğince çok makineye ulaşır.

Fidye Yazılımı



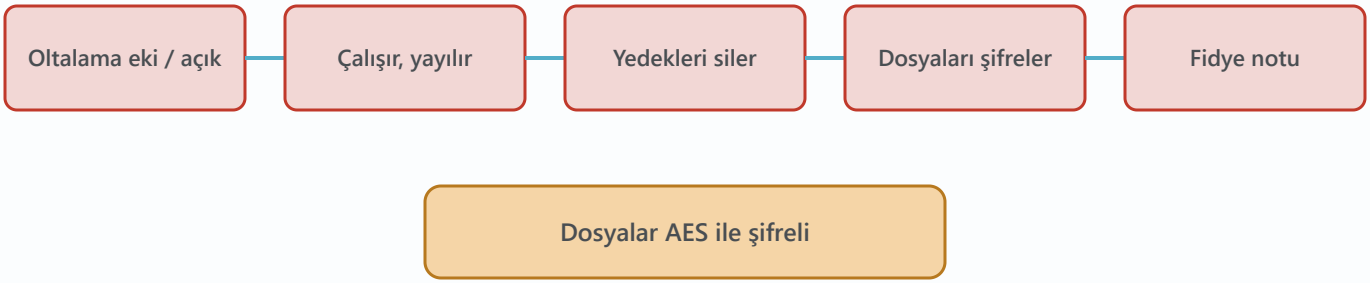
Kurtarmayı engellemek için **gölge kopyaları ve yedekleri siler** (bu yüzden çevrim dışı/değiştirilemez yedek şart).

Fidye Yazılımı



Dosyalar güçlü simetrik şifreyle kilitlenir; anahtar saldırganın sunucusuna gönderilir.

Fidye Yazılımı



Ödeme = anahtar garantisi değil; yedek + yama tek güvenilir savunma

Fidye notu görüntülenir. Ödeme anahtarı garanti etmez ve yeni saldırıları teşvik eder. Savunma: 3-2-1 yedek, yama, MFA, segmentasyon, EDR.

KÖTÜ AMAÇLI YAZILIM

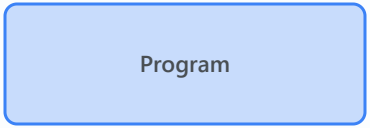
Rootkit ve Gizlenme



Rootkit varlığını gizler: dosyaları, süreçleri, ağ bağlantılarını işletim sisteminden saklar. Ne kadar derin yerleşirse o kadar güçlü ama kurması da zordur. Çekirdek ve altı için savunma: imzalı sürücüler, **güvenli önyükleme**, ölçülü önyükleme (TPM) ve dışarıdan (çevrim dışı) tarama.

Antivirüs ve Kaçırma Teknikleri

(a) temiz



İmza: bilinen bayt dizisi aranır

Temiz program referans alınır.

Adım 1 / 5

İmza tarama
Bilinen desenler; hızlı, düşük yanlış alarm, yeniyi kaçırır.

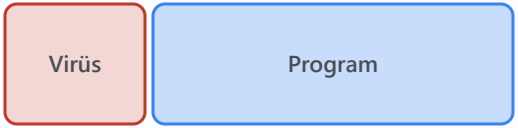
Bütünlük denetimi
Dosya özetlerini saklar, değişimi fark eder.

Davranışsal / sezgisel
Şüpheli eylemleri (şifreleme, enjeksiyon) izler.

Korumalı alan + bulut
Şüpheliyi yalıtılmış ortamda çalıştırıp gözler.

Antivirüs ve Kaçırma Teknikleri

(b) virüslü



İmza virüsü yakalar

İmza tarama: bilinen virüslerin bayt dizileri (imza) aranır. Hızlı ama yalnız bilineni yakalar.

İmza tarama
Bilinen desenler; hızlı, düşük yanlış alarm, yeniyi kaçırır.

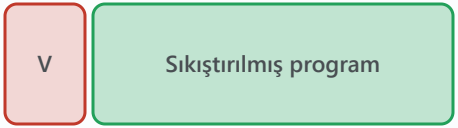
Bütünlük denetimi
Dosya özetlerini saklar, değişimi fark eder.

Davranışsal / sezgisel
Şüpheli eylemleri (şifreleme, enjeksiyon) izler.

Korumalı alan + bulut
Şüpheliyi yalıtılmış ortamda çalıştırıp gözler.

Antivirüs ve Kaçırma Teknikleri

(c) sıkıştırılmış



Sıkıştırma imzayı gizler; tarayıcı açıp bakmalı

Virüs kendini **sıkıştırırsa** imza görünmez; tarayıcı dosyayı açıp içini taramak zorundadır.

İmza tarama
Bilinen desenler; hızlı, düşük yanlış alarm, yeniyi kaçırır.

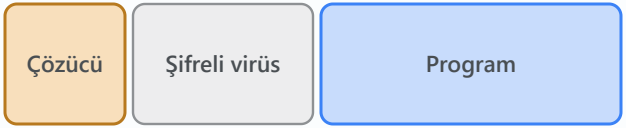
Bütünlük denetimi
Dosya özetlerini saklar, değişimi fark eder.

Davranışsal / sezgisel
Şüpheli eylemleri (şifreleme, enjeksiyon) izler.

Korumalı alan + bulut
Şüpheliyi yalıtılmış ortamda çalıştırıp gözler.

Antivirüs ve Kaçırma Teknikleri

(d) şifreli virüs



Gövde şifreli: yalnız çözücü sabit kalır, onun imzası aranır

Şifreli virüs: gövde şifreli, yalnız küçük çözücü açıkta; tarayıcı çözücünün imzasını arar.

İmza tarama
Bilinen desenler; hızlı, düşük yanlış alarm, yeniyi kaçırır.

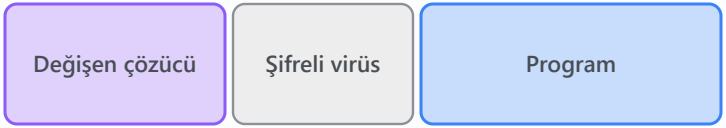
Bütünlük denetimi
Dosya özetlerini saklar, değişimi fark eder.

Davranışsal / sezgisel
Şüpheli eylemleri (şifreleme, enjeksiyon) izler.

Korumalı alan + bulut
Şüpheliyi yalıtılmış ortamda çalıştırıp gözler.

Antivirüs ve Kaçırma Teknikleri

(e) çok şekilli



Çok şekilli (polymorphic): çözücü her kopyada farklı → imza işe yaramaz, davranış analizi gerekir

Çok şekilli virüs: çözücü de her kopyada değişir. İmza yetmez → **sezgisel/davranışsal** analiz, **korumalı alanda** çalıştırıp gözlemleme gerekir.

Adım 5 / 5

İmza tarama
Bilinen desenler; hızlı, düşük yanlış alarm, yeniyi kaçırır.

Bütünlük denetimi
Dosya özetlerini saklar, değişimi fark eder.

Davranışsal / sezgisel
Şüpheli eylemleri (şifreleme, enjeksiyon) izler.

Korumalı alan + bulut
Şüpheliyi yalıtılmış ortamda çalıştırıp gözler.

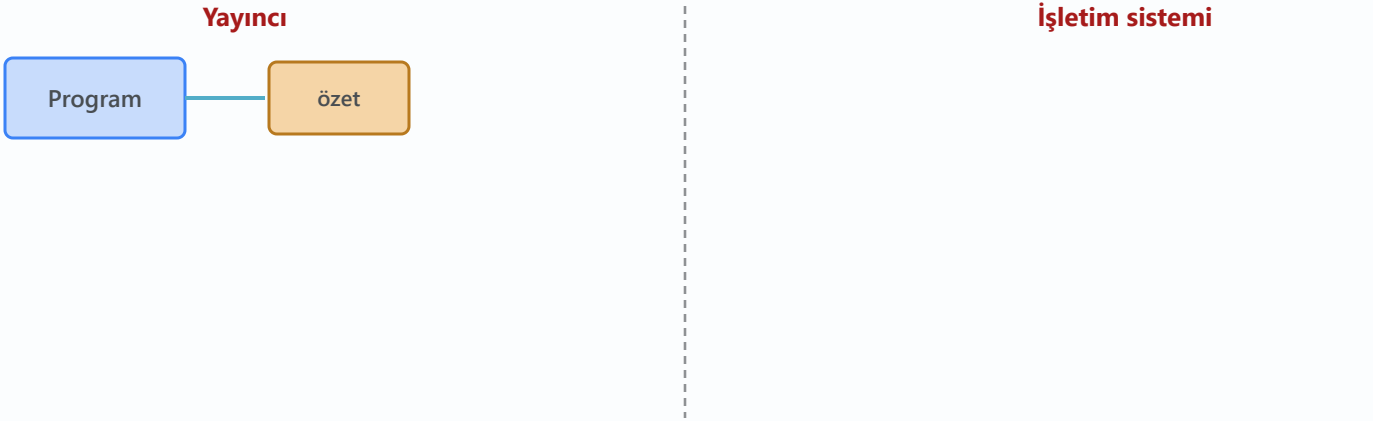
Kod İmzalama

Yayıncı

İşletim sistemi

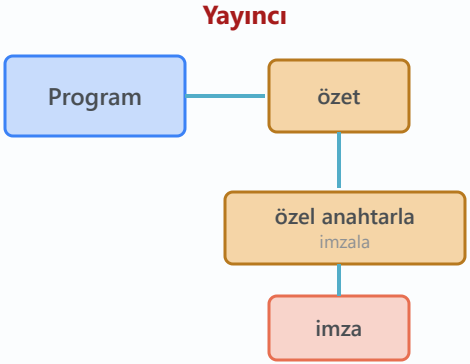
Yayıncı programın **özetini** hesaplar (dijital imzanın aynısı, bu kez koda uygulanır).

Kod İmzalama



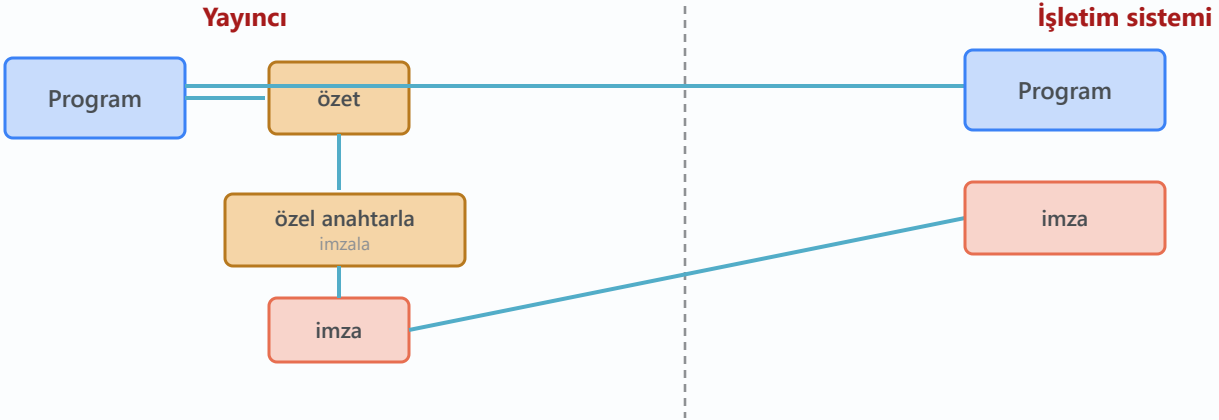
Özet yayıncının **özel anahtarıyla imzalanır**; imza programla birlikte dağıtılır.

Kod İmzalama



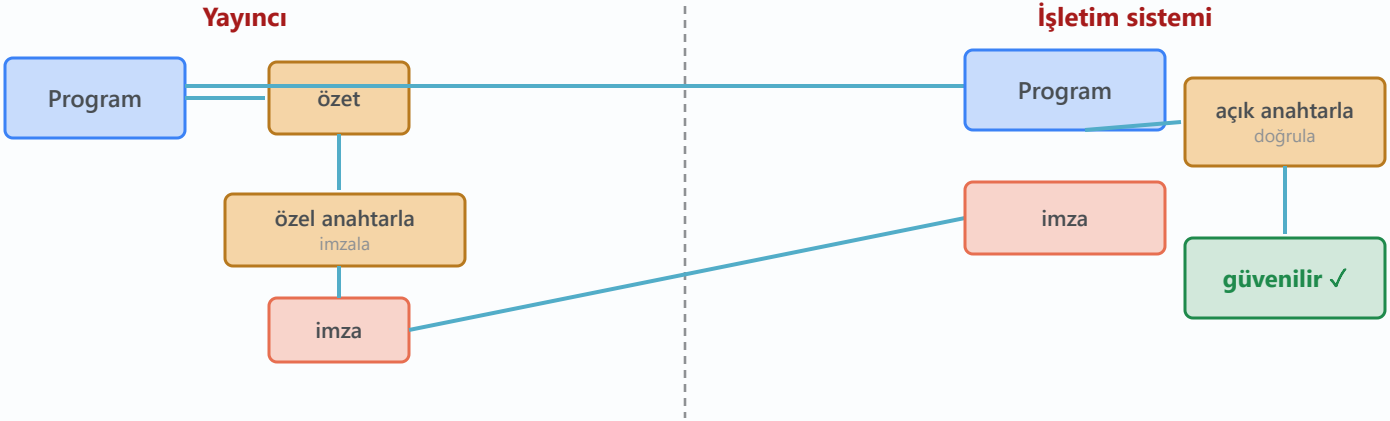
Kullanıcı programı ve imzayı alır; yayıncının sertifikası açık anahtarını taşır.

Kod İmzalama



İşletim sistemi özeti yeniden hesaplar ve imzayı açık anahtarla doğrular; eşleşirse program **değiştirilmemiş** ve o yayıncıdandır.

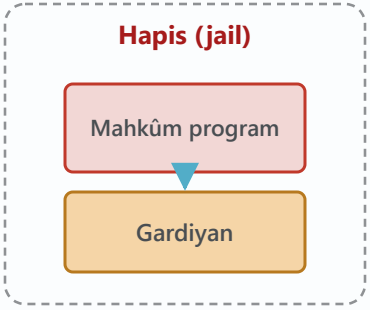
Kod İmzalama



App Store'lar, sürücü imzalama, `apt` / `dnf` paket imzaları ve platform "yalnız imzalı kod" ilkeleri bu temele dayanır.

SAVUNMA VE YALITIM

Korumalı Alan ve Hapsetme



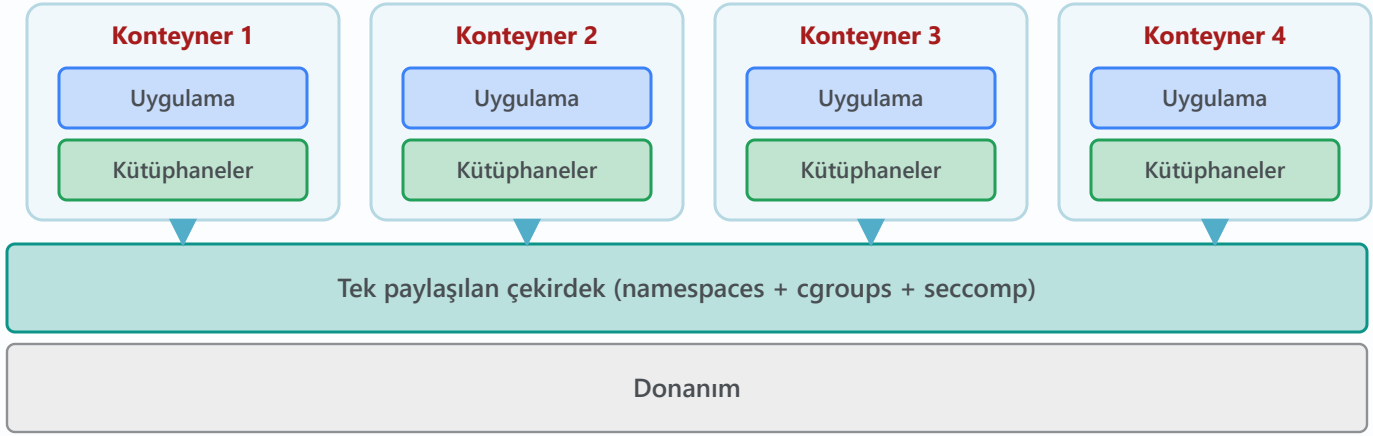
Her sistem çağrısı gardiyan tarafından denetlenir, izinsizler **engellenir**

Modern korumalı alan

- seccomp-bpf: izinli sistem çağrıları
- namespaces: ayrı PID/ağ/dosya görünümü
- cgroups: CPU/bellek sınırı
- yetenekler düşürülür
- SELinux/AppArmor profili

Korumalı alan (sandboxing): güvenilmeyen kodu kısıtlı bir ortamda çalıştırıp verebileceği zararı sınırlar. Tarayıcı sekmeleri, mobil uygulamalar ve sunucu hizmetleri bugün korumalı alanda koşar. Linux'ta temel taşlar: `seccomp`, ad alanları (namespaces), cgroups ve zorunlu erişim denetimi.

Konteynerler ve Yalıtım



	Konteyner	**Sanal makine**
Yalıtım	Çekirdek özellikleri	Donanım + hipervizör
Ek yük	Düşük, saniyede başlar	Yüksek, ayrı çekirdek
Saldırı yüzeyi	Çekirdek açığı = kaçış riski	Daha güçlü sınır

SAVUNMA VE YALITIM

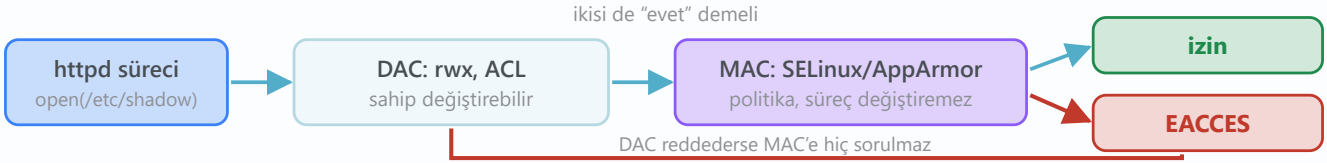
Zorunlu Erişim Denetimi: SELinux ve AppArmor

DAC (isteğe bağlı)
Sahibin verdiği izinler (rwx, ACL). Kullanıcı kendi dosyasının iznini değiştirebilir; zararlı da değiştirebilir.

MAC (zorunlu)
Politikayı sistem/yönetici belirler; süreç kendi kısıtını gevşetemez. BLP/Biba, SELinux, AppArmor.

SELinux
Her özne ve nesneye etiket; tip zorlaması (type enforcement) ile ayrıntılı kurallar. Güçlü ama karmaşık.

AppArmor
Yol tabanlı profiller; uygulama başına izin listesi. Yazması ve anlaması daha kolay.



Zorunlu erişim denetimi (MAC), ele geçirilen bir süreci bile profilinin izin verdiği işlemlerle sınırlar: web sunucusu açığı kullanılsa bile saldırgan yalnız o profilin tanımladığı dosyalara erişebilir. Android bütünüyle SELinux zorunlu modda çalışır.

Güvenli Önyükleme ve TPM



Güvenli önyükleme: her katman bir sonrakinin imzasını doğrular; donanımdaki kök güvenine dayanır. **TPM** (Güvenilir Platform Modülü) önyükleme bileşenlerinin özetlerini (PCR) saklar; **ölçülü önyükleme** ile disk şifreleme anahtarı yalnız sistem beklenen durumdaysa açılır (uzaktan kanıtlama, attestation). Rootkit ve önyükleme kiti (bootkit) bu zinciri kırmayı dener.

Spectre, Meltdown ve Yan Kanallar



Yan kanal (side channel): veriyi doğrudan değil, dolaylı fiziksel etkiden (zaman, güç, önbellek, ses) sızdırır. **Meltdown** ve **Spectre** (2018) CPU'ların **spekülatif yürütmesini** kötüye kullanır: iptal edilen işlemin önbellekte bıraktığı iz zamanlama ile okunur ve yalıtım aşılır. Önlemler: çekirdek sayfa tablosu yalıtımı (KPTI), mikrokod, derleyici bariyerleri — çoğu performans bedeliyle gelir.

Özet

Konu	Ana fikir	Bugün
Hedefler	Gizlilik, bütünlük, erişilebilirlik	Sıfır güven, +özgünlük/hesap verebilirlik
Kriptografi	Simetrik + asimetrik + özet + imza	AES-GCM, X25519, Ed25519, TLS 1.3, ML-KEM
Erişim denetimi	Matris → ACL veya yetenek	En az ayrıcalık, Linux yetenekleri, MAC
Biçimsel modeller	BLP gizlilik, Biba bütünlük	SELinux MLS, Windows MIC
Kimlik doğrulama	Tuzlu yavaş özet + MFA	Argon2, FIDO2/geçiş anahtarı
Bellek güvenliği	Taşma dönüş adresini bozar	Kanarya+NX+ASLR+CFI, Rust
Yalıtım	Korumalı alan zararı sınırlar	seccomp, namespaces, konteyner
Donanım	Güven zinciri ve ölçüm	Güvenli önyükleme, TPM; yan kanallar

ÖZET

Kontrol Soruları

1. CIA üçlüsünün üç bileşenini, her birine birer ihlal örneğiyle açıklayın.
2. Aynı erişim matrisinin ACL ve yetenek gösterimlerini karşılaştırın: geri alma ve devretme hangisinde kolaydır, neden?
3. Bell-LaPadula ve Biba kurallarını (okuma/yazma yönleri) yazın. Hangisi gizliliği, hangisi bütünlüğü korur?
4. Parolalar neden düz SHA-256 ile değil, tuzlu Argon2/bcrypt ile saklanmalı? Tuz ve yavaşlığın ayrı ayrı rolleri nedir?
5. FIDO2/geçiş anahtarları parola + TOTP'ye göre oltalamaya neden dayanıklıdır?
6. Yığın tampon taşmasında dönüş adresi nasıl ele geçirilir? ASLR, NX ve yığın kanaryası bunu hangi aşamada zorlaştırır?
7. TOCTOU yarış durumunu `access()` / `open()` örneğiyle anlatın ve bir çözüm verin.
8. Gizli depolama kanalı ile zamanlama kanalını örnekle ayırın. Neden tümüyle kapatılamazlar?
9. Güvenli önyükleme güven zincirini ve TPM'in rolünü açıklayın. Rootkit'i nerede durdurur?
10. Konteyner ile sanal makine yalıtımı arasındaki temel fark nedir? Hangisinde çekirdek açığı daha tehlikelidir?

▶ **Cevaplar**

Kaynaklar

- A. S. Tanenbaum, H. Bos, *Modern Operating Systems*, 4. baskı (2014) / 5. baskı (2023), Pearson — Bölüm 9: Security.
- A. Silberschatz, P. B. Galvin, G. Gagne, *Operating System Concepts*, 10. baskı, Wiley, 2018 — Bölüm 16: Security; Bölüm 17: Protection.
- R. H. Arpaci-Dusseau, A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces* (OSTEP), ostep.org — güvenlik bölümleri.
- R. Anderson, *Security Engineering*, 3. baskı, Wiley, 2020.
- J. H. Saltzer, M. D. Schroeder, "The Protection of Information in Computer Systems", 1975 (en az ayrıcalık ve tasarım ilkeleri).
- D. E. Bell, L. J. LaPadula (1973); K. J. Biba (1977) — biçimsel güvenlik modelleri. B. W. Lampson, "A Note on the Confinement Problem", 1973.
- RFC 8446 (TLS 1.3), RFC 6238 (TOTP), RFC 9106 (Argon2); FIDO2/WebAuthn (W3C); NIST FIPS 203 (ML-KEM).
- Aleph One, "Smashing the Stack for Fun and Profit", Phrack 49, 1996; Kocher vd., "Spectre"; Lipp vd., "Meltdown", 2018.

Son

Bölüm 13: Güvenlik

Sonraki bölüm: İşletim Sistemi Tasarımı · Terim sözlüğü