# CEN429 GÃ¼venli Programlama Hafta-4

Kod GÃ¼Ã§lendirme Teknikleri

Yazar: Dr. Ã–Ÿr. Ãœyesi UÄŸur CORUH

## İçindekiler

# Şekil Listesi

# Tablo Listesi

# 1 CEN429 GÃ¼venli Programlama

## 1.1 Hafta-4

#### 1.1.0.1 Kod GÃ¼Ã§lendirme Teknikleri Ä°ndir

- PDF[1]
- DOC[2]
- SLIDE[3]
- PPTX[4]

### 1.1.1 Outline

- Kod GÃ¼Ã§lendirme Teknikleri

[1] pandoc_cen429-week-4.pdf
[2] pandoc_cen429-week-4.docx
[3] cen429-week-4.pdf
[4] cen429-week-4.pptx

- Native C/C++ Ä°Ã§in Kod GÃ¼Ã§lendirme
- Java ve Yorumlanan Diller Ä°Ã§in Kod GÃ¼Ã§lendirme

## 1.2 Hafta-4: Kod GÃ¼Ã§lendirme Teknikleri

#### 1.2.0.1 1. Native C/C++ Ä°Ã§in Kod GÃ¼Ã§lendirme Teknikleri

C ve C++ gibi dÃ¼ÅŸÃ¼k seviye dillerde gÃ¼venli kod yazmak ve saldÄ±rÄ±lara karÅŸÄ± dayanÄ±klÄ± hale getirmek iÃ§in Ã§eÅŸitli teknikler kullanÄ±lÄ±r. Bu teknikler, kodun analiz edilmesini ve geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rmayÄ± amaÃ§lar.

#### 1.2.0.2 1.1 Opaque Loops (Opak DÃ¶ngÃ¼ler)

**Teorik AÃ§Ä±klama:** Opak dÃ¶ngÃ¼ler, dÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda amacÄ± belli olmayan dÃ¶ngÃ¼lerdir. Bu dÃ¶ngÃ¼ler sayesinde kodun analizi zorlaÅŸÄ±r. SaldÄ±rgan, dÃ¶ngÃ¼nÃ¼n iÅŸlevini anlamakta zorlanÄ±r ve kodun Ã§Ã¶zÃ¼lmesi daha karmaÅŸÄ±k hale gelir.

**Uygulama Ã–rnekleri:**

1. Rastgele bir koÅŸul ile oluÅŸturulmuÅŸ dÃ¶ngÃ¼ler ekleyerek kodun analizini zorlaÅŸtÄ±rma.
2. DÄ±ÅŸarÄ±dan anlaÅŸÄ±lmayan ancak programÄ±n iÅŸleyiÅŸine zarar vermeyen dÃ¶ngÃ¼ler ekleme.
3. Opak dÃ¶ngÃ¼ler ile programÄ±n Ã§alÄ±ÅŸma sÃ¼resini arttÄ±rarak saldÄ±rganÄ± yanÄ±ltma.

### 1.2.1 1.1.1. Rastgele Bir KoÅŸul ile OluÅŸturulmuÅŸ DÃ¶ngÃ¼ (Opaque Loop with Random Condition)

Bu Ã¶rnekte, rastgele bir koÅŸul ile dÃ¶ngÃ¼ oluÅŸturularak kodun anlaÅŸÄ±lmasÄ± zorlaÅŸtÄ±rÄ±lÄ±yor.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

void opak_rastgele_dongu() {
    srand(time(0)); // Rastgele sayÄ± Ã¼reticisini baÅŸlatÄ±r
    int x = rand() % 50; // DÃ¶ngÃ¼ koÅŸulu iÃ§in rastgele bir deÄŸer
    for (int i = 0; i < x; i++) {
        if (i % 2 == 0) {
            std::cout << "Ä°terasyon: " << i << std::endl;
        }
    }
}

int main() {
    opak_rastgele_dongu();
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, dÃ¶ngÃ¼ koÅŸulu rastgele bir sayÄ±dan oluÅŸur. Bu durum, dÃ¶ngÃ¼nÃ¼n amacÄ±nÄ± dÄ±ÅŸarÄ±dan anlamayÄ± zorlaÅŸtÄ±rÄ±r ve saldÄ±rganlar iÃ§in analizi karmaÅŸÄ±k hale getirir.

### 1.2.2 1.1.2. DÄ±ÅŸarÄ±dan AnlaÅŸÄ±lmayan Ancak ProgramÄ±n Ä°ÅŸleyiÅŸine Zarar Vermeyen DÃ¶ngÃ¼ (Opaque Loop with No External Effect)

Bu Ã¶rnekte, dÃ¶ngÃ¼ kodun iÅŸleyiÅŸine zarar vermeyen ancak dÄ±ÅŸarÄ±dan anlaÅŸÄ±lmayan bir iÅŸlev iÃ§erir.

```
#include <iostream>
```

```
void opak_islevsiz_dongu() {
    for (int i = 0; i < 100; i++) {
        if (i % 3 == 0) {
            int gizli_islem = i * 42; // Ä°ÅŸleyiÅŸe etkisi olmayan gereksiz iÅŸlem
        }
    }
    std::cout << "Opak dÃ¶ngÃ¼ tamamlandÄ±." << std::endl;
}

int main() {
    opak_islevsiz_dongu();
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, dÃ¶ngÃ¼de yapÄ±lan iÅŸlem programÄ±n ana iÅŸleyiÅŸine katkÄ± saÄŸlamaz. Bu tÃ¼r dÃ¶ngÃ¼ler, saldÄ±rganlarÄ± yanÄ±ltmak ve kodun analizini zorlaÅŸtÄ±rmak iÃ§in kullanÄ±lÄ±r.

### 1.2.3 1.1.3. Opak DÃ¶ngÃ¼ler ile ProgramÄ±n Ã‡alÄ±ÅŸma SÃ¼resini ArttÄ±rma (Opaque Loops to Delay Program Execution)

Bu Ã¶rnek, programÄ±n Ã§alÄ±ÅŸma sÃ¼resini uzatarak saldÄ±rganlarÄ± yanÄ±ltmak amacÄ±yla kullanÄ±labilir.

```
#include <iostream>
#include <thread>
#include <chrono>

void gecikmeli_opak_dongu() {
    for (int i = 0; i < 5; i++) {
        std::this_thread::sleep_for(std::chrono::seconds(1)); // Her dÃ¶ngÃ¼de 1 saniye bekleme
        std::cout << "Gecikmeli dÃ¶ngÃ¼ iterasyonu: " << i << std::endl;
    }
}

int main() {
    gecikmeli_opak_dongu();
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, dÃ¶ngÃ¼ her iterasyonda programÄ±n Ã§alÄ±ÅŸmasÄ±nÄ± yavaÅŸlatan bir gecikme ekler. Bu tÃ¼r teknikler, saldÄ±rganlarÄ±n programÄ±n tam olarak ne yaptÄ±ÄŸÄ± konusunda kafa karÄ±ÅŸÄ±klÄ±ÄŸÄ± yaratmak iÃ§in kullanÄ±lÄ±r.

#### 1.2.3.1 1.2 Shared Object Sembollerini Gizleme (Configure Shared Object Symbol Invisible)

**Teorik AÃ§Ä±klama:** PaylaÅŸÄ±lan nesneler (shared object) iÃ§inde kullanÄ±lan sembollerin gizlenmesi, bu nesnelere dÄ±ÅŸarÄ±dan eriÅŸimi zorlaÅŸtÄ±rÄ±r. Bu iÅŸlem, analiz ve geri mÃ¼hendislik iÅŸlemlerini engellemek iÃ§in kullanÄ±lÄ±r.

**Uygulama Ã–rnekleri:**

1. Derleyici seÃ§enekleriyle sembollerin gÃ¶rÃ¼nÃ¼rlÃ¼ÄŸÃ¼nÃ¼ sÄ±nÄ±rlama.
2. Sadece gerekli sembolleri dÄ±ÅŸa aÃ§arak diÄŸer sembollerin eriÅŸilemez olmasÄ±nÄ± saÄŸlama.
3. PaylaÅŸÄ±lan kÃ¼tÃ¼phanelerdeki kritik fonksiyonlarÄ± gizleyerek gÃ¼venliÄŸi artÄ±rma.

### 1.2.4 1.2.1. Derleyici SeÃ§enekleriyle Sembollerin GÃ¶rÃ¼nÃ¼rlÃ¼ÄŸÃ¼nÃ¼ SÄ±nÄ±rlama (Limiting Symbol Visibility with Compiler Options)

**AÃ§Ä±klama:**
PaylaÅŸÄ±lan nesne dosyalarÄ±nda, sembollerin varsayÄ±lan olarak dÄ±ÅŸa aÃ§Ä±k (public) olmasÄ±,

gÃ¼venlik aÃ§Ä±klarÄ±na yol aÃ§abilir. Bu durumu Ã¶nlemek iÃ§in derleyici seÃ§enekleriyle sembollerin gÃ¶rÃ¼nÃ¼rlÃ¼ÄŸÃ¼ sÄ±nÄ±rlandÄ±rÄ±labilir. Ã–rneÄŸin, GCC ve Clang derleyicilerinde `-fvisibility=hidden` seÃ§eneÄŸi kullanÄ±larak, sadece dÄ±ÅŸa aÃ§Ä±lmasÄ±na izin verilen semboller gÃ¶rÃ¼nÃ¼r olur.

**Ã–rnek:**

```
// foo.cpp
#include <iostream>

void gizli_fonksiyon() __attribute__((visibility("hidden"))); // Fonksiyon gizli

void gizli_fonksiyon() {
    std::cout << "Bu fonksiyon dÄ±ÅŸa aÃ§Ä±k deÄŸildir." << std::endl;
}

void acik_fonksiyon() {
    std::cout << "Bu fonksiyon dÄ±ÅŸa aÃ§Ä±ktÄ±r." << std::endl;
}
```

**Derleme Komutu:**

```
g++ -fvisibility=hidden -shared -o libfoo.so foo.cpp
```

**AÃ§Ä±klama:**
Bu komut, tÃ¼m sembolleri varsayÄ±lan olarak gizler (`-fvisibility=hidden`), ancak `acik_fonksiyon` dÄ±ÅŸa aÃ§Ä±labilir durumda kalÄ±r. `gizli_fonksiyon` ise dÄ±ÅŸarÄ±dan eriÅŸilemez olur.

### 1.2.5 1.2.2. Sadece Gerekli Sembolleri DÄ±ÅŸa AÃ§ma (Only Exporting Necessary Symbols)

**AÃ§Ä±klama:**
PaylaÅŸÄ±lan kÃ¼tÃ¼phanelerde yalnÄ±zca gerekli semboller dÄ±ÅŸa aÃ§Ä±lÄ±r. Bu sayede, sadece belirli fonksiyonlar dÄ±ÅŸarÄ±dan Ã§aÄŸrÄ±labilirken diÄŸer semboller gizli kalÄ±r.

**Ã–rnek:**

```
// gizli_kutuphane.cpp
#include <iostream>

void gizli_fonksiyon() {
    std::cout << "Bu fonksiyon gizli kalacak." << std::endl;
}

extern "C" void acik_fonksiyon() {
    std::cout << "Bu fonksiyon dÄ±ÅŸa aÃ§Ä±k." << std::endl;
}
```

**Derleme Komutu:**

```
g++ -fvisibility=hidden -shared -o libgizli.so gizli_kutuphane.cpp
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `acik_fonksiyon` dÄ±ÅŸa aÃ§Ä±labilirken, `gizli_fonksiyon` dÄ±ÅŸarÄ±dan eriÅŸilemez durumda kalÄ±r. Bu teknik, sadece dÄ±ÅŸarÄ±dan kullanÄ±lmasÄ±na izin verilen fonksiyonlarÄ±n eriÅŸime aÃ§Ä±lmasÄ±nÄ± saÄŸlar.

### 1.2.6 1.2.3. PaylaÅŸÄ±lan KÃ¼tÃ¼phanelerde Kritik FonksiyonlarÄ± Gizleme (Hiding Critical Functions in Shared Libraries)

**AÃ§Ä±klama:**
Kritik Ã¶neme sahip fonksiyonlar gizlenerek, geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rmak ve

paylaÅŸÄ±lan kÃ¼tÃ¼phanelerin gÃ¼venliÄŸini artÄ±rmak mÃ¼mkÃ¼ndÃ¼r. Bu yaklaÅŸÄ±m, saldÄ±rganlarÄ±n Ã¶nemli fonksiyonlarÄ± analiz etmesini ve manipÃ¼le etmesini Ã¶nler.

**Ã–rnek:**

```
// kritik_kutuphane.cpp
#include <iostream>

__attribute__((visibility("hidden"))) void kritik_fonksiyon() {
    std::cout << "Bu kritik fonksiyon gizlenmiÅŸtir." << std::endl;
}

extern "C" void genel_fonksiyon() {
    std::cout << "Bu genel fonksiyon dÄ±ÅŸa aÃ§Ä±ktÄ±r." << std::endl;
    kritik_fonksiyon(); // Gizli fonksiyon burada iÃ§sel olarak Ã§aÄŸrÄ±lÄ±r
}
```

**Derleme Komutu:**

```
g++ -fvisibility=hidden -shared -o libkritik.so kritik_kutuphane.cpp
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `kritik_fonksiyon` dÄ±ÅŸarÄ±dan eriÅŸilemez ve yalnÄ±zca `genel_fonksiyon` Ã¼zerinden Ã§aÄŸrÄ±labilir. Bu, kritik fonksiyonlarÄ±n dÄ±ÅŸa kapalÄ± kalmasÄ±nÄ± saÄŸlayarak gÃ¼venliÄŸi artÄ±rÄ±r.

#### 1.2.6.1 1.3. Aritmetik Ä°ÅŸlemlerin Obfuske Edilmesi (Obfuscation of Arithmetic Instructions) Teorik AÃ§Ä±klama:

Aritmetik iÅŸlemler, programÄ±n en temel yapÄ± taÅŸlarÄ±dÄ±r. Bu iÅŸlemleri karmaÅŸÄ±k hale getirmek, kodun analizini ve anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

**Uygulama Ã–rnekleri:**

1. Basit toplama iÅŸlemlerini daha karmaÅŸÄ±k matematiksel ifadeler ile deÄŸiÅŸtirme.
2. Aritmetik iÅŸlemlerine gereksiz adÄ±mlar ekleyerek iÅŸlevselliÄŸi korurken kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rma.
3. Aritmetik iÅŸlemler Ã¼zerinde bit manipÃ¼lasyonu yaparak daha karmaÅŸÄ±k hale getirme.

### 1.2.7 1.3.1. Basit Toplama Ä°ÅŸlemlerini Daha KarmaÅŸÄ±k Matematiksel Ä°fadeler ile DeÄŸiÅŸtirme (Replacing Simple Additions with Complex Mathematical Expressions)

**AÃ§Ä±klama:**
Basit aritmetik iÅŸlemlerini karmaÅŸÄ±k hale getirerek, kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rabiliriz. Ã–rneÄŸin, bir toplama iÅŸlemini daha uzun ve karmaÅŸÄ±k matematiksel iÅŸlemlerle deÄŸiÅŸtirmek, saldÄ±rganlarÄ±n kodu analiz etmesini zorlaÅŸtÄ±rÄ±r.

**Ã–rnek:**

```
#include <iostream>

int karmaÅŸÄ±k_toplama(int a, int b) {
    // Basit toplama iÅŸlemi: a + b
    // KarmaÅŸÄ±k hale getirilmiÅŸ hali
    return ((a * 2) + (b * 2) - a - b); // a + b ile aynÄ± sonucu verir
}

int main() {
    int a = 5, b = 10;
    int sonuc = karmaÅŸÄ±k_toplama(a, b);
    std::cout << "KarmaÅŸÄ±k toplama sonucu: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda basit bir toplama iÅŸlemi (a + b) daha karmaÅŸÄ±k bir matematiksel ifadeye dÃ¶nÃ¼ÅŸtÃ¼rÃ¼lmÃ¼ÅŸtÃ¼r. Her ne kadar sonuÃ§ aynÄ± olsa da kodun analizi zorlaÅŸÄ±r.

### 1.2.8 1.3.2. Aritmetik Ä°ÅŸlemlerine Gereksiz AdÄ±mlar Ekleyerek Kodun AnlaÅŸÄ±lmasÄ±nÄ± ZorlaÅŸtÄ±rma (Adding Redundant Steps to Obfuscate Arithmetic Operations)

**AÃ§Ä±klama:**
Aritmetik iÅŸlemlerine gereksiz adÄ±mlar eklemek, iÅŸlevi deÄŸiÅŸtirmeden kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. Ã–rneÄŸin, ekleme ve Ã§Ä±karma iÅŸlemleri eklenerek kod daha karmaÅŸÄ±k hale getirilebilir.

**Ã–rnek:**

```
#include <iostream>

int gereksiz_adimlarla_toplama(int a, int b) {
    // Toplama iÅŸlemini gereksiz adÄ±mlarla karmaÅŸÄ±klaÅŸtÄ±rma
    int sonuc = (a * 2 - a) + (b * 2 - b); // a + b iÅŸlemini yapar
    return sonuc;
}

int main() {
    int a = 7, b = 3;
    int sonuc = gereksiz_adimlarla_toplama(a, b);
    std::cout << "Gereksiz adÄ±mlarla toplama sonucu: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, toplama iÅŸlemi gereksiz adÄ±mlarla karmaÅŸÄ±k hale getirilmiÅŸtir. SonuÃ§ yine `a + b` olsa da iÅŸlem, dÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda anlaÅŸÄ±lmasÄ± zor hale gelmiÅŸtir.

### 1.2.9 1.3.3. Aritmetik Ä°ÅŸlemler Ãœzerinde Bit ManipÃ¼lasyonu Yaparak KarmaÅŸÄ±k Hale Getirme (Adding Bit Manipulation to Obfuscate Arithmetic Operations)

**AÃ§Ä±klama:**
Aritmetik iÅŸlemlere bit manipÃ¼lasyonu ekleyerek kodu daha da karmaÅŸÄ±k hale getirebiliriz. Bu yÃ¶ntem, Ã¶zellikle dÃ¼ÅŸÃ¼k seviyeli dillerde kodun geri mÃ¼hendislik iÅŸlemine karÅŸÄ± dayanÄ±klÄ±lÄ±ÄŸÄ±nÄ± artÄ±rÄ±r.

**Ã–rnek:**

```
#include <iostream>

int bit_manipulasyonu_ile_toplama(int a, int b) {
    // Aritmetik iÅŸlemi bit manipÃ¼lasyonu ile karmaÅŸÄ±k hale getirme
    int sonuc = ((a << 1) >> 1) + ((b << 1) >> 1); // a + b iÅŸlemi yapar
    return sonuc;
}

int main() {
    int a = 4, b = 8;
    int sonuc = bit_manipulasyonu_ile_toplama(a, b);
    std::cout << "Bit manipÃ¼lasyonu ile toplama sonucu: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, toplama iÅŸlemi bit kaydÄ±rma iÅŸlemleri (left shift, right shift) ile karmaÅŸÄ±k hale getirilmiÅŸtir. Bit manipÃ¼lasyonu, aritmetik iÅŸlemi gizler ve kodun analiz edilmesini zorlaÅŸtÄ±rÄ±r.

#### 1.2.9.1 1.4. Fonksiyon Ä°simlerinin Obfuske Edilmesi (Obfuscation of Function Names) Teorik AÃ§Ä±klama:

Fonksiyon isimlerinin rastgele karakter dizileri ile deÄŸiÅŸtirilmesi, kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. Bu teknik, Ã¶zellikle tersine mÃ¼hendislik (reverse engineering) iÅŸlemlerini engellemek iÃ§in kullanÄ±lÄ±r.

**Uygulama Ã–rnekleri:**

1. Fonksiyon isimlerini anlamsÄ±z karakter dizileri ile deÄŸiÅŸtirme.
2. Her derlemede farklÄ± fonksiyon isimleri oluÅŸturarak statik analiz araÃ§larÄ±nÄ± yanÄ±ltma.
3. Kritik fonksiyonlarÄ±n isimlerini rastgele hale getirerek saldÄ±rganlarÄ±n bu fonksiyonlarÄ± anlamasÄ±nÄ± zorlaÅŸtÄ±rma.

### 1.2.10 1.4.1. Fonksiyon Ä°simlerini AnlamsÄ±z Karakter Dizileri ile DeÄŸiÅŸtirme (Replacing Function Names with Nonsense Character Strings)

**AÃ§Ä±klama:**
Fonksiyon isimleri, anlaÅŸÄ±lmayÄ± zorlaÅŸtÄ±rmak amacÄ±yla anlamsÄ±z karakter dizileri ile deÄŸiÅŸtirilir. Bu yÃ¶ntem, saldÄ±rganlarÄ±n hangi fonksiyonun ne iÅŸe yaradÄ±ÄŸÄ±nÄ± anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

**Ã–rnek:**

```
#include <iostream>

// Fonksiyon isimleri anlamsÄ±z karakter dizileriyle deÄŸiÅŸtirilmiÅŸ
void abcdef123() {
    std::cout << "Kritik bir iÅŸlem gerÃ§ekleÅŸtirildi." << std::endl;
}

int main() {
    abcdef123(); // Fonksiyon Ã§aÄŸrÄ±sÄ±
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `abcdef123` gibi anlamsÄ±z bir karakter dizisi kullanÄ±larak fonksiyon ismi gizlenmiÅŸtir. Bu durum, kodun anlaÅŸÄ±lmasÄ±nÄ± ve analiz edilmesini zorlaÅŸtÄ±rÄ±r.

### 1.2.11 1.4.2. Her Derlemede FarklÄ± Fonksiyon Ä°simleri OluÅŸturarak Statik Analiz AraÃ§larÄ±nÄ± YanÄ±ltma (Generating Different Function Names on Every Compile)

**AÃ§Ä±klama:**
Her derlemede farklÄ± fonksiyon isimleri oluÅŸturmak, statik analiz araÃ§larÄ±nÄ± ve geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rabilir. Derleme sÄ±rasÄ±nda fonksiyon isimleri rastgele oluÅŸturularak kod analizi karmaÅŸÄ±k hale getirilir.

**Ã–rnek:**

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// Rastgele fonksiyon ismi oluÅŸturma
#define OBFUSCATED_FUNC_NAME random_function_name

void OBFUSCATED_FUNC_NAME() {
    std::cout << "Bu fonksiyonun ismi her derlemede deÄŸiÅŸebilir." << std::endl;
}

int main() {
    OBFUSCATED_FUNC_NAME(); // Fonksiyon Ã§aÄŸrÄ±sÄ±
```

```
    return 0;
}
```

**AÃ§Ä±klama:**
Makro tanÄ±mÄ± ve derleme sÃ¼recinde rastgele fonksiyon isimleri oluÅŸturmak, her derlemede farklÄ± bir isim kullanarak kodun statik analiz araÃ§larÄ± tarafÄ±ndan analiz edilmesini zorlaÅŸtÄ±rÄ±r.

### 1.2.12 1.4.3. Kritik FonksiyonlarÄ±n Ä°simlerini Rastgele Hale Getirerek SaldÄ±rganlarÄ±n Bu FonksiyonlarÄ± AnlamasÄ±nÄ± ZorlaÅŸtÄ±rma (Randomizing Critical Function Names to Obfuscate Purpose)

**AÃ§Ä±klama:**
Kritik fonksiyonlarÄ±n isimlerini rastgele hale getirmek, saldÄ±rganlarÄ±n bu fonksiyonlarÄ±n amacÄ±nÄ± anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. Bu teknik, Ã¶zellikle geri mÃ¼hendislik iÅŸlemlerini engellemek iÃ§in kullanÄ±lÄ±r.

**Ã–rnek:**

```
#include <iostream>

// Kritik fonksiyonlarÄ±n isimleri rastgele belirlenmiÅŸ
void xj239rf84() {
    std::cout << "Kritik bir gÃ¼venlik iÅŸlemi gerÃ§ekleÅŸtiriliyor." << std::endl;
}

void z8kd93p2() {
    std::cout << "Kritik bir doÄŸrulama iÅŸlemi gerÃ§ekleÅŸtiriliyor." << std::endl;
}

int main() {
    xj239rf84(); // Kritik gÃ¼venlik fonksiyonu Ã§aÄŸrÄ±sÄ±
    z8kd93p2();  // Kritik doÄŸrulama fonksiyonu Ã§aÄŸrÄ±sÄ±
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte kritik fonksiyonlar rastgele isimler almÄ±ÅŸtÄ±r (`xj239rf84`, `z8kd93p2`). Bu, tersine mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rÄ±r, Ã§Ã¼nkÃ¼ isimler fonksiyonun iÅŸlevi hakkÄ±nda bilgi vermez.

#### 1.2.12.1 1.5. Kaynak Dosya Ä°simlerinin Obfuske Edilmesi (Obfuscation of Source File Names) Teorik AÃ§Ä±klama:

Kaynak dosyalarÄ±n isimlerini anlamsÄ±z hale getirerek kodun hangi fonksiyona veya sÄ±nÄ±fa ait olduÄŸunu gizleme.

**Uygulama Ã–rnekleri:**

1. Kaynak dosyalarÄ±n isimlerini rastgele karakterler ile deÄŸiÅŸtirme.
2. Kaynak dosyalar arasÄ±ndaki iliÅŸkiyi gizleyerek kod yapÄ±sÄ±nÄ± anlaÅŸÄ±lmaz hale getirme.
3. Dosya isimlerini obfuske ederken kaynak kodu etkilemeyecek Åekilde yapÄ±larÄ± deÄŸiÅŸtirme.

### 1.2.13 1.5.1. Kaynak DosyalarÄ±n Ä°simlerini Rastgele Karakterler ile DeÄŸiÅŸtirme (Randomizing Source File Names)

**AÃ§Ä±klama:**
Kaynak dosya isimlerini rastgele karakter dizileri ile deÄŸiÅŸtirerek, bu dosyalarÄ±n hangi iÅŸlevleri barÄ±ndÄ±rdÄ±ÄŸÄ±nÄ± gizleyebiliriz. Bu teknik, saldÄ±rganlarÄ±n hangi dosyanÄ±n hangi iÅŸlemi gerÃ§ekleÅŸtirdiÄŸini anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

**Ã–rnek:**

1. **Orijinal Dosya AdÄ±:**

hesaplama.cpp

```
// hesaplama.cpp
#include <iostream>

void hesapla() {
    std::cout << "Hesaplama iÅŸlemi" << std::endl;
}
```

2. **Obfuske EdilmiÅŸ Dosya AdÄ±:**

x7z23f.cpp

```
// x7z23f.cpp

void x7z23f() {
    std::cout << "Hesaplama iÅŸlemi" << std::endl;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `hesaplama.cpp` adlÄ± dosyanÄ±n adÄ± `x7z23f.cpp` olarak deÄŸiÅŸtirilmiÅŸtir. AyrÄ±ca, fonksiyon adÄ± da aynÄ± rastgele karakterlerle deÄŸiÅŸtirilmiÅŸtir. Bu, kod yapÄ±sÄ±nÄ± gizlemek iÃ§in etkili bir yÃ¶ntemdir.

### 1.2.14 1.5.2. Kaynak Dosyalar ArasÄ±ndaki Ä°liÅŸkiyi Gizleyerek Kod YapÄ±sÄ±nÄ± AnlaÅŸÄ±lmaz Hale Getirme (Hiding Relationships Between Source Files)

**AÃ§Ä±klama:**
Kod yapÄ±sÄ±ndaki dosyalar arasÄ±ndaki iliÅŸkiyi gizlemek, saldÄ±rganlarÄ±n kaynak dosyalarÄ±n nasÄ±l etkileÅŸimde bulunduÄŸunu anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. Bu teknik, kodun genel yapÄ±sÄ±nÄ± daha gizli hale getirir.

**Ã–rnek:**

1. **Orijinal Dosyalar:**
    - `hesaplama.cpp`
    - `utils.cpp`

```
// hesaplama.cpp
#include "utils.h"

void hesapla() {
    int sonuc = toplama(5, 10); // utils.cpp dosyasÄ±ndaki fonksiyon Ã§aÄŸrÄ±sÄ±
    std::cout << "SonuÃ§: " << sonuc << std::endl;
}
```

```
// utils.cpp
int toplama(int a, int b) {
    return a + b;
}
```

**Obfuske EdilmiÅŸ Dosyalar:**

- `a9s8d.cpp`
- `p2f6k.cpp`

```
// a9s8d.cpp
#include "p2f6k.h"

void a9s8d() {
    int sonuc = p2f6k(5, 10); // Fonksiyon iliÅŸkisi gizlenmiÅŸ
    std::cout << "SonuÃ§: " << sonuc << std::endl;
}
```

```cpp
// p2f6k.cpp
int p2f6k(int a, int b) {
    return a + b;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, iki dosya arasÄ±ndaki iliÅŸki dosya isimlerinin ve fonksiyon isimlerinin deÄŸiÅŸtirilmesiyle gizlenmiÅŸtir. Dosyalar arasÄ±ndaki iliÅŸki, dÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda anlaÅŸÄ±lamaz hale getirilmiÅŸtir.

### 1.2.15 1.5.3. Dosya Ä°simlerini Obfuske Ederken Kaynak Kodu Etkilemeyecek Åžekilde YapÄ±larÄ± DeÄŸiÅŸtirme (Obfuscating File Names Without Affecting Source Code Structure)

**AÃ§Ä±klama:**
Dosya isimleri obfuske edilse bile kaynak kodun Ã§alÄ±ÅŸma mantÄ±ÄŸÄ± deÄŸiÅŸtirilmez. Derleme sÄ±rasÄ±nda dosya isimleri ve kod yapÄ±larÄ± arasÄ±nda doÄŸru baÄŸlantÄ± kurularak kodun iÅŸlevselliÄŸi korunur.

**Ã–rnek:**

1. **Orijinal Dosya YapÄ±sÄ±:**

```
kaynak/
â"œâ"€â"€ hesaplama.cpp
â""â"€â"€ utils.cpp
```

```cpp
// hesaplama.cpp
#include "utils.h"

void hesapla() {
    std::cout << "Hesaplama iÅŸlemi baÅŸladÄ±." << std::endl;
}
```

2. **Obfuske EdilmiÅŸ Dosya YapÄ±sÄ±:**

```
kaynak/
â"œâ"€â"€ q1w2e.cpp
â""â"€â"€ z3x4c.cpp
```

```cpp
// q1w2e.cpp
#include "z3x4c.h"

void q1w2e() {
    std::cout << "Hesaplama iÅŸlemi baÅŸladÄ±." << std::endl;
}
```

**AÃ§Ä±klama:**
Dosya isimleri ve fonksiyon isimleri deÄŸiÅŸtirilmiÅŸ olsa da, dosyalar arasÄ±ndaki iliÅŸki doÄŸru referanslarla korunmuÅŸ ve kaynak kodun iÅŸlevselliÄŸi etkilenmemiÅŸtir.

#### 1.2.15.1 1.6. Statik Dizelerin Obfuske Edilmesi (Obfuscation of Static Strings)

**Teorik AÃ§Ä±klama:** Statik dizeler, saldÄ±rganlar iÃ§in Ã¶nemli bilgi kaynaklarÄ±dÄ±r. Bu dizelerin ÅŸifrelenmesi ve gizlenmesi, kod gÃ¼venliÄŸini artÄ±rÄ±r.

**Uygulama Ã–rnekleri:**

1. Statik dizeleri ÅŸifreleyerek Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lmesini saÄŸlama.
2. Rastgele dize maskeleri uygulayarak dizelerin anlamÄ±nÄ± gizleme.
3. Dize sabitlerini kaldÄ±rarak sabit dize kullanÄ±mÄ±nÄ± azaltma.

### 1.2.16 1.6.1. Statik Dizeleri Åžifreleyerek Ã‡alÄ±ÅŸma AnÄ±nda Ã‡Ã¶zÃ¼lmesini SaÄŸlama (Encrypting Static Strings and Decrypting at Runtime)

**AÃ§Ä±klama:**
Statik dizeler saldÄ±rganlar iÃ§in Ã¶nemli bilgi kaynaklarÄ± olabilir. Bu yÃ¼zden, dizeler Ã§alÄ±ÅŸma zamanÄ±nda ÅŸifrelenip, yalnÄ±zca gerekli olduÄŸunda Ã§Ã¶zÃ¼lerek kullanÄ±labilir.

**Ã–rnek:**

```
#include <iostream>
#include <string>

// Basit bir XOR ÅŸifreleme ve Ã§Ã¶zme fonksiyonu
std::string xor_sifrele(const std::string &input, char key) {
    std::string output = input;
    for (size_t i = 0; i < input.size(); i++) {
        output[i] ^= key; // XOR iÅŸlemi
    }
    return output;
}

int main() {
    std::string sifreli_dize = xor_sifrele("GizliMesaj", 0xAA); // Åžifreleme
    std::cout << "ÅžifrelenmiÅŸ Dize: " << sifreli_dize << std::endl;

    std::string cozulmus_dize = xor_sifrele(sifreli_dize, 0xAA); // Ã‡Ã¶zme
    std::cout << "Ã‡Ã¶zÃ¼lmÃ¼ÅŸ Dize: " << cozulmus_dize << std::endl;

    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, statik bir dize (`"GizliMesaj"`) XOR iÅŸlemi kullanÄ±larak ÅŸifrelenmiÅŸtir. Dizeler Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lerek anlamlÄ± hale getirilir ve saldÄ±rganlarÄ±n dizeleri statik analiz araÃ§larÄ±yla doÄŸrudan gÃ¶rmesi engellenir.

### 1.2.17 1.6.2. Rastgele Dize Maskeleri Uygulayarak Dizelerin AnlamÄ±nÄ± Gizleme (Applying Random String Masks to Obfuscate String Meaning)

**AÃ§Ä±klama:**
Statik dizeler Ã¼zerine rastgele maskeler uygulanarak dizelerin anlamÄ± gizlenir. Bu teknik, saldÄ±rganlarÄ±n ÅŸifrelenmiÅŸ dizeleri Ã§Ã¶zmesini zorlaÅŸtÄ±rÄ±r.

**Ã–rnek:**

```
#include <iostream>
#include <string>

std::string dize_maskele(const std::string &input) {
    std::string output = input;
    for (size_t i = 0; i < input.size(); i++) {
        output[i] ^= (i % 255); // Rastgele bir maskeleme iÅŸlemi
    }
    return output;
}

int main() {
    std::string orijinal_dize = "Ã–nemliBilgi";
    std::string maske_dize = dize_maskele(orijinal_dize); // Maskeleme
    std::cout << "Masked Dize: " << maske_dize << std::endl;
```

```
    std::string cozulmus_dize = dize_maskele(maske_dize); // Maskeleme ters iÅŸlemi
    std::cout << "Ã‡Ã¶zÃ¼lmÃ¼Å Dize: " << cozulmus_dize << std::endl;

    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, dizenin her karakterine maskeleme uygulanarak dize karmaÅŸÄ±k hale getirilmiÅŸtir. Ã‡alÄ±Åma anÄ±nda maskeler ters Ã§evrilerek dizenin anlamÄ± tekrar ortaya Ã§Ä±kar. Bu yÃ¶ntem, dizelerin doÄŸrudan okunmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

### 1.2.18 1.6.3. Dize Sabitlerini KaldÄ±rarak Sabit Dize KullanÄ±mÄ±nÄ± Azaltma (Reducing the Use of Static String Constants)

**AÃ§Ä±klama:**
Kodda sabit dizeler kullanmak, saldÄ±rganlar iÃ§in ipuÃ§larÄ± saÄŸlayabilir. Bu yÃ¼zden sabit dize kullanÄ±mÄ±nÄ± en aza indirerek ve Ã§alÄ±Åma zamanÄ±nda dizeleri oluÅŸturarak gÃ¼venliÄŸi artÄ±rmak mÃ¼mkÃ¼ndÃ¼r.

**Ã–rnek:**

```
#include <iostream>
#include <sstream>

std::string dinamik_dize_olustur() {
    std::ostringstream ss;
    ss << "Parola" << "2024"; // Sabit dizeleri dinamik olarak birleÅŸtiriyoruz
    return ss.str();
}

int main() {
    std::string parola = dinamik_dize_olustur(); // Parola dinamik olarak oluÅŸturuluyor
    std::cout << "Dinamik Dize: " << parola << std::endl;

    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, sabit dize yerine, dizeler Ã§alÄ±Åma zamanÄ±nda dinamik olarak oluÅŸturulmuÅŸtur. Bu yaklaÅŸÄ±m, kod iÃ§inde sabit dizelerin bulunmasÄ±nÄ± ve bu dizelere saldÄ±rÄ± yapÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

### 1.2.19 1.7. Opak Boolean DeÄŸiÅŸkenler (Opaque Boolean Variables)

**Teorik AÃ§Ä±klama:**
Opak boolean deÄŸiÅŸkenler, koÅŸullu ifadelerin anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rmak iÃ§in kullanÄ±lÄ±r. Bu teknik, koÅŸullarÄ±n karmaÅŸÄ±k hale getirilmesiyle kodun analizini ve geri mÃ¼hendislik iÅŸlemlerini gÃ¼Ã§leÅŸtirir.

**Ã–rnek Ã–nerisi:**

- Rastgele boolean deÄŸerleri dÃ¶ndÃ¼ren bir fonksiyonun koÅŸullu ifadelerde kullanÄ±lmasÄ±.
- Åžartların karmaÅŸÄ±klaÅŸtÄ±rÄ±lmasÄ±yla kodun Ã¶ngÃ¶rÃ¼lemez hale getirilmesi.

### 1.2.20 1.7.1. Rastgele Boolean DeÄŸerleri DÃ¶ndÃ¼ren Bir Fonksiyonun KullanÄ±lmasÄ±

**AÃ§Ä±klama:**
Bu Ã¶rnekte, rastgele boolean deÄŸer dÃ¶ndÃ¼ren bir fonksiyon, koÅŸullu ifadelerde kullanÄ±larak kodun Ã¶ngÃ¶rÃ¼lemez hale getirilmesi saÄŸlanmÄ±ÅŸtÄ±r. Bu durum, kodun anlaÅŸÄ±lmasÄ±nÄ± ve analiz edilmesini zorlaÅŸtÄ±rÄ±r.

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

// Rastgele opak boolean deÄŸer dÃ¶ndÃ¼ren fonksiyon
bool opak_boolean() {
    srand(time(0)); // Rastgele sayÄ± Ã¼retici baÅŸlatÄ±lÄ±yor
    return rand() % 2; // Rastgele true veya false dÃ¶ner
}

void gizli_islem(int a) {
    bool durum = opak_boolean(); // Rastgele boolean koÅŸul
    if (a > 10 && durum) {
        std::cout << "Gizli iÅŸlem Ã§alÄ±ÅŸtÄ±rÄ±lÄ±yor, durum: true" << std::endl;
    } else {
        std::cout << "KoÅŸul saÄŸlanmadÄ±, durum: false" << std::endl;
    }
}

int main() {
    gizli_islem(12); // Girdi deÄŸeriyle fonksiyon Ã§aÄŸrÄ±lÄ±yor
    gizli_islem(7);  // FarklÄ± girdi deÄŸeriyle tekrar Ã§aÄŸrÄ±lÄ±yor
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, `opak_boolean` fonksiyonu rastgele olarak `true` ya da `false` dÃ¶ndÃ¼rÃ¼r. KoÅŸullu ifade, bu rastgele deÄŸerle birleÅŸtiÄŸinde saldÄ±rganlar iÃ§in kodun analizi zorlaÅŸÄ±r.

### 1.2.21 1.7.2. Åžartların KarmaÅŸÄ±klaÅŸtırılmasÄ± ile Kodun Öngörülemez Hale Getirilmesi

**AÃ§Ä±klama:**
Bu örnek, boolean değerleriyle karmaşık koşullar oluşturularak kodun öngörülemez hale getirilmesini sağlar. Bu tür karmaşık koşullar, kodun geri mühendislik sürecini zorlaştırır.

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

// KarmaÅŸÄ±k boolean dÃ¶ndÃ¼ren fonksiyon
bool karmaÅŸÄ±k_boolean(int a) {
    srand(time(0));
    int b = rand() % 10;
    return ((a + b) % 3 == 0) && (a % 2 == 0);
}

void kontrol_et(int a) {
    if (karmaÅŸÄ±k_boolean(a)) {
        std::cout << "KoÅŸul saÄŸlandÄ±, karmaÅŸÄ±k boolean: true" << std::endl;
    } else {
        std::cout << "KoÅŸul saÄŸlanmadÄ±, karmaÅŸÄ±k boolean: false" << std::endl;
    }
}

int main() {
    kontrol_et(12); // Girdi deÄŸeriyle fonksiyon Ã§aÄŸrÄ±lÄ±yor
    kontrol_et(7);  // FarklÄ± girdi deÄŸeriyle tekrar Ã§aÄŸrÄ±lÄ±yor
```

```
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, boolean deÄŸeri hesaplamak iÃ§in kullanÄ±lan karmaÅŸÄ±k bir koÅŸul seti bulunur. Bu ÅŸartlar, kodun anlaÅŸÄ±lmasÄ±nÄ± ve analiz edilmesini zorlaÅŸtÄ±rÄ±r, Ã§Ã¼nkÃ¼ her defasÄ±nda farklÄ± sonuÃ§lar Ã¼retilebilir ve koÅŸullarÄ±n ne zaman saÄŸlandÄ±ÄŸÄ±nÄ± anlamak gÃ¼Ã§leÅŸir.

### 1.2.22 1.8. Fonksiyon Boolean Return KodlarÄ±nÄ± KarmaÅŸÄ±klaÅŸtÄ±rma (Function Boolean Return Codes)

**Teorik AÃ§Ä±klama:**
FonksiyonlarÄ±n dÃ¶nÃ¼Å deÄŸerlerini karmaÅŸÄ±klaÅŸtÄ±rmak, kodun akÄ±ÅŸÄ±nÄ± ve iÅŸlevselliÄŸini anlamayÄ± zorlaÅŸtÄ±rÄ±r. Bu teknik, fonksiyonlarÄ±n hangi koÅŸullar altÄ±nda hangi deÄŸerleri dÃ¶ndÃ¼rdÃ¼ÄŸÃ¼nÃ¼ belirsiz hale getirir.

**Ã–rnek Ã–nerisi:**

- KarmaÅŸÄ±k matematiksel iÅŸlemlerle koÅŸullu dÃ¶nÃ¼Å deÄŸerleri Ã¼reten bir fonksiyon.
- Geri mÃ¼hendislik iÅŸlemine karÅŸÄ± fonksiyonlarÄ±n dÃ¶nÃ¼Å deÄŸerlerini tahmin edilemez hale getirme.

### 1.2.23 1.8.1. KarmaÅŸÄ±k Matematiksel Ä°Ålemlerle KoÅŸullu DÃ¶nÃ¼Å DeÄŸerleri Ãœreten Bir Fonksiyon

**AÃ§Ä±klama:**
Bu Ã¶rnekte, fonksiyonun dÃ¶nÃ¼Å deÄŸeri karmaÅŸÄ±k matematiksel iÅŸlemlerle belirlenir. Bu, fonksiyonun ne zaman `true` veya `false` dÃ¶ndÃ¼rdÃ¼ÄŸÃ¼nÃ¼ anlamayÄ± zorlaÅŸtÄ±rarak kodun analiz edilmesini gÃ¼Ã§leÅŸtirir.

```
#include <iostream>
#include <cmath> // Matematiksel fonksiyonlar iÃ§in

// KarmaÅŸÄ±k matematiksel iÅŸlemle boolean dÃ¶ndÃ¼ren fonksiyon
bool karmaÅŸÄ±k_donus_degeri(int a, int b) {
    int sonuc = static_cast<int>(pow(a, 3) - pow(b, 2)); // KarmaÅŸÄ±k hesaplama
    return (sonuc % 7 == 0); // KoÅŸula gÃ¶re boolean dÃ¶ner
}

void kontrol_et(int a, int b) {
    if (karmaÅŸÄ±k_donus_degeri(a, b)) {
        std::cout << "KarmaÅŸÄ±k dÃ¶nÃ¼Å deÄŸeri: true" << std::endl;
    } else {
        std::cout << "KarmaÅŸÄ±k dÃ¶nÃ¼Å deÄŸeri: false" << std::endl;
    }
}

int main() {
    kontrol_et(4, 2);  // Girdi deÄŸerleriyle fonksiyon Ã§aÄŸrÄ±lÄ±yor
    kontrol_et(7, 3);  // FarklÄ± girdi deÄŸerleriyle tekrar Ã§aÄŸrÄ±lÄ±yor
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, `karmaÅŸÄ±k_donus_degeri` fonksiyonu `a` ve `b` deÄŸerlerine gÃ¶re karmaÅŸÄ±k matematiksel iÅŸlemler gerÃ§ekleÅŸtirir. Hesaplamalar sonucunda `sonuc % 7 == 0` koÅŸuluna gÃ¶re `true` veya `false` dÃ¶ner. Bu, fonksiyonun ne zaman hangi deÄŸeri dÃ¶ndÃ¼rdÃ¼ÄŸÃ¼nÃ¼ anlamayÄ± zorlaÅŸtÄ±rÄ±r ve geri mÃ¼hendislik iÅŸlemlerini karmaÅŸÄ±k hale getirir.

### 1.2.24 1.8.2. Geri MÃ¼hendislik Ä°ÅŸlemine KarÅŸÄ± FonksiyonlarÄ±n DÃ¶nÃ¼Ş DeÄŸerlerini Tahmin Edilemez Hale Getirme

**AÃ§Ä±klama:**
Bu Ã¶rnekte, fonksiyonlarÄ±n dÃ¶nÃ¼Ş deÄŸerleri rastgele iÅŸlemlerle belirsiz hale getirilir. BÃ¶ylece, fonksiyonlarÄ±n ne zaman hangi deÄŸeri dÃ¶ndÃ¼rdÃ¼ÄŸÃ¼ dÄ±ÅŸarÄ±dan anlaÅŸÄ±lmaz.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// Geri mÃ¼hendisliÄŸi zorlaÅŸtÄ±rmak iÃ§in rastgele iÅŸlemlerle boolean dÃ¶ndÃ¼ren fonksiyon
bool tahmin_edilemez_donus(int a, int b) {
    srand(time(0));
    int rastgele = rand() % 100;
    int sonuc = (a * b) + rastgele; // Rastgele sayÄ± ile iÅŸlem
    return (sonuc % 5 == 0); // KoÅŸula gÃ¶re true veya false dÃ¶ner
}

void sonuc_kontrol(int a, int b) {
    if (tahmin_edilemez_donus(a, b)) {
        std::cout << "Tahmin edilemez dÃ¶nÃ¼Ş deÄŸeri: true" << std::endl;
    } else {
        std::cout << "Tahmin edilemez dÃ¶nÃ¼Ş deÄŸeri: false" << std::endl;
    }
}

int main() {
    sonuc_kontrol(6, 4); // FarklÄ± girdi deÄŸerleriyle fonksiyon Ã§aÄŸrÄ±lÄ±yor
    sonuc_kontrol(7, 5);
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `tahmin_edilemez_donus` fonksiyonu rastgele bir sayÄ± Ã¼reterek hesaplamalarÄ±na dahil eder. Bu rastgelelik, fonksiyonun ne zaman `true` veya `false` dÃ¶ndÃ¼receÄŸini tahmin etmeyi imkansÄ±z hale getirir. Bu yaklaÅŸÄ±m, geri mÃ¼hendislik ve statik analiz araÃ§larÄ±na karÅŸÄ± fonksiyonlarÄ± daha korunaklÄ± hale getirir.

### 1.2.25 1.9. Fonksiyon Parametrelerinin Gizlenmesi (Obfuscation of Function Parameters)

**Teorik AÃ§Ä±klama:**
Fonksiyon parametrelerini gizlemek, fonksiyonlarÄ±n aldÄ±ÄŸÄ± verilerin ne olduÄŸunu ve nasÄ±l iÅŸlendiÄŸini anlamayÄ± zorlaÅŸtÄ±rÄ±r. Bu teknik, parametrelerin anlamÄ±nÄ± ve kullanÄ±mÄ±nÄ± belirsiz hale getirir.

**Ã–rnek Ã–nerisi:**

- Parametre isimlerinin anlamsÄ±z hale getirildiÄŸi ve fonksiyonlarÄ±n iÅŸlevinin dÄ±ÅŸarÄ±dan anlaÅŸÄ±lmaz olduÄŸu bir Ã¶rnek.
- Parametrelerin maskeleme yÃ¶ntemleriyle gizlenmesi.

### 1.2.26 1.9.1. Parametre Ä°simlerinin AnlamsÄ±z Hale Getirilmesi

**AÃ§Ä±klama:**
Bu teknik, fonksiyon parametrelerinin isimlerini anlamsÄ±z hale getirerek kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. DÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda, fonksiyonun ne yaptÄ±ÄŸÄ± veya parametrelerin ne amaÃ§la kullanÄ±ldÄ±ÄŸÄ± anlaÅŸÄ±lmaz hale gelir.

```
#include <iostream>
```

```
// AnlamsÄ±z parametre isimleri ile tanÄ±mlanan fonksiyon
int z4m1nq0(int p1, int p2) {
    return p1 * p2 + (p1 - p2); // KarmaÅŸÄ±k bir iÅŸlem
}

int main() {
    int sonuc = z4m1nq0(10, 5); // Fonksiyon Ã§aÄŸrÄ±sÄ±
    std::cout << "SonuÃ§: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `z4m1nq0` gibi anlamsÄ±z bir fonksiyon ismi ve `p1`, `p2` gibi parametre isimleri kullanÄ±lmÄ±ÅŸtÄ±r. Parametrelerin ne olduÄŸu ve nasÄ±l kullanÄ±ldÄ±ÄŸÄ± anlaÅŸÄ±lmaz hale getirilmiÅŸtir. Bu, kodun geri mÃ¼hendislik ile analiz edilmesini zorlaÅŸtÄ±rÄ±r.

### 1.2.27 1.9.2. Parametrelerin Maskeleme YÃ¶ntemleri ile Gizlenmesi

**AÃ§Ä±klama:**
Bu teknikte, parametreler maskeleme iÅŸlemi ile gizlenir. Parametreler Ã§alÄ±ÅŸma zamanÄ±nda aÃ§Ä±ÄŸa Ã§Ä±karÄ±lÄ±r ve kodun ne yaptÄ±ÄŸÄ± dÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda anlaÅŸÄ±lamaz.

```
#include <iostream>

// Maskeleme fonksiyonu
int parametre_maskele(int param) {
    return param ^ 0x5A; // XOR ile basit bir maskeleme
}

// GizlenmiÅŸ parametre ile iÅŸlem yapan fonksiyon
int gizli_fonksiyon(int a, int b) {
    int gercek_a = parametre_maskele(a); // Parametre maskelemesini Ã§Ã¶z
    int gercek_b = parametre_maskele(b);
    return gercek_a + gercek_b; // GerÃ§ek deÄŸerlerle iÅŸlem yap
}

int main() {
    int a = parametre_maskele(10); // Parametreler maskelenmiÅŸ
    int b = parametre_maskele(20);
    int sonuc = gizli_fonksiyon(a, b); // Gizli fonksiyon Ã§aÄŸrÄ±sÄ±
    std::cout << "SonuÃ§: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, `parametre_maskele` fonksiyonu parametreleri maskeler ve daha sonra maskeleme Ã§Ã¶zÃ¼lerek fonksiyon iÃ§inde gerÃ§ek deÄŸerler kullanÄ±lÄ±r. Bu teknik, fonksiyonun aldÄ±ÄŸÄ± parametrelerin ne olduÄŸunu gizler ve saldÄ±rganlarÄ±n analiz etmesini zorlaÅŸtÄ±rÄ±r.

### 1.2.28 1.10. AnlamsÄ±z Parametreler ve Ä°ÅŸlemler Ekleyerek Kodun Analizini ZorlaÅŸtÄ±rma (Bogus Function Parameters & Operations)

**Teorik AÃ§Ä±klama:**
Kodun analizini zorlaÅŸtÄ±rmak iÃ§in fonksiyonlara anlamsÄ±z parametreler ve gereksiz iÅŸlemler eklemek kullanÄ±lÄ±r. Bu teknik, saldÄ±rganlarÄ±n fonksiyonlarÄ±n gerÃ§ek amacÄ±nÄ± belirlemesini engeller.

**Ã–rnek Ã–nerisi:**

- Fonksiyonlara gereksiz parametreler ekleyerek ve anlamsÄ±z iÅŸlemler yaparak kodun karmaÅŸÄ±k hale getirildiÄŸi bir Ã¶rnek.

- SaldÄ±rganlarÄ± yanÄ±ltacak anlamsÄ±z hesaplama ve koÅŸullarÄ±n eklendiÄŸi bir fonksiyon.

### 1.2.29 1.10.1. Fonksiyonlara Gereksiz Parametreler Ekleyerek Kodun KarmaÅŸÄ±k Hale Getirilmesi

**AÃ§Ä±klama:**
Bu teknik, fonksiyonlara gereksiz parametreler ekleyerek kodun anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rÄ±r. Parametrelerin iÅŸlevi olmadÄ±ÄŸÄ±ndan, fonksiyonun gerÃ§ek amacÄ± belirsiz hale gelir.

```
#include <iostream>

// Gereksiz parametreler iÃ§eren fonksiyon
int anlamsiz_fonksiyon(int a, int b, int gereksiz1, int gereksiz2) {
    // GerÃ§ek iÅŸlem sadece a ve b ile yapÄ±lÄ±r
    return (a * b) + 10;
}

int main() {
    // Gereksiz parametrelerle fonksiyon Ã§aÄŸrÄ±sÄ±
    int sonuc = anlamsiz_fonksiyon(5, 3, 100, 200);
    std::cout << "SonuÃ§: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, `anlamsiz_fonksiyon` adlÄ± fonksiyona gereksiz parametreler (`gereksiz1`, `gereksiz2`) eklenmiÅŸtir. Bu parametrelerin gerÃ§ek bir iÅŸlevi olmadÄ±ÄŸÄ± iÃ§in, dÄ±ÅŸarÄ±dan bakÄ±ldÄ±ÄŸÄ±nda fonksiyonun ne yaptÄ±ÄŸÄ± anlaÅŸÄ±lmaz. Bu teknik, geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rÄ±r.

### 1.2.30 1.10.2. AnlamsÄ±z Hesaplama ve KoÅŸullarÄ±n EklendiÄŸi Bir Fonksiyon

**AÃ§Ä±klama:**
Bu Ã¶rnekte, fonksiyona anlamsÄ±z iÅŸlemler ve gereksiz koÅŸullar eklenerek kod karmaÅŸÄ±k hale getirilir. Bu yaklaÅŸÄ±m, fonksiyonun gerÃ§ek amacÄ±nÄ± gizleyerek analiz edilmesini zorlaÅŸtÄ±rÄ±r.

```
#include <iostream>

// AnlamsÄ±z iÅŸlemler ve koÅŸullar iÃ§eren fonksiyon
int karmasik_fonksiyon(int a, int b, int c) {
    int temp = a * b; // GerÃ§ek iÅŸlem
    if (c > 100) { // AnlamsÄ±z koÅŸul
        temp += c; // AnlamsÄ±z iÅŸlem
    }
    for (int i = 0; i < c; i++) { // Gereksiz dÃ¶ngÃ¼
        temp -= i; // AnlamsÄ±z hesaplama
    }
    return temp;
}

int main() {
    int sonuc = karmasik_fonksiyon(5, 3, 50); // Fonksiyon Ã§aÄŸrÄ±sÄ±
    std::cout << "SonuÃ§: " << sonuc << std::endl;
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, fonksiyonun ana iÅŸlemi `a * b` ile yapÄ±lÄ±r. Ancak gereksiz koÅŸullar (`if (c > 100)`) ve dÃ¶ngÃ¼ler eklenerek kod karmaÅŸÄ±k hale getirilmiÅŸtir. AnlamsÄ±z iÅŸlemler ve hesaplamalar, saldÄ±rganlarÄ±n fonksiyonun gerÃ§ek iÅŸlevini anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

### 1.2.31 1.11. Kontrol AkÄ±ÅŸÄ±nÄ± DÃ¼zleÅŸtirerek Tahmin Edilemez Hale Getirme (Control Flow Flattening)

**Teorik AÃ§Ä±klama:**
Kontrol akÄ±ÅŸÄ±nÄ± dÃ¼zleÅŸtirmek, kodun normal akÄ±ÅŸÄ±nÄ± bozar ve programÄ±n akÄ±ÅŸÄ±nÄ± tahmin etmeyi zorlaÅŸtÄ±rÄ±r. Bu teknik, kontrol yapÄ±larÄ±nÄ± karmaÅŸÄ±klaÅŸtÄ±rarak kodun analizini zorlaÅŸtÄ±rÄ±r.

**Ã–rnek Ã–nerisi:**

- DÃ¼zleÅŸtirilmiÅŸ kontrol akÄ±ÅŸÄ±yla koÅŸullu ifadeler yerine durum tabanlÄ± geÃ§iÅŸlerin kullanÄ±ldÄ±ÄŸÄ± bir Ã¶rnek.
- Kod akÄ±ÅŸÄ±nÄ±n tahmin edilemez hale getirilmesi.

### 1.2.32 1.11.1. Durum TabanlÄ± GeÃ§iÅŸlerin KullanÄ±ldÄ±ÄŸÄ± DÃ¼zleÅŸtirilmiÅŸ Kontrol AkÄ±ÅŸÄ±

**AÃ§Ä±klama:**
Bu Ã¶rnekte, kontrol akÄ±ÅŸÄ± dÃ¼zleÅŸtirilmiÅŸ ve durum tabanlÄ± geÃ§iÅŸler kullanÄ±larak kodun akÄ±ÅŸÄ± karmaÅŸÄ±k hale getirilmiÅŸtir. Bu yÃ¶ntem, koÅŸullu ifadeler yerine durumlar Ã¼zerinden ilerler ve kodun normal akÄ±ÅŸÄ± bozulur.

```
#include <iostream>

void kontrol_akisi_duzlestir(int a) {
    int state = 0; // BaÅŸlangÄ±Ã§ durumu
    while (true) {
        switch (state) {
            case 0:
                if (a > 10) {
                    state = 1; // Durum 1'e geÃ§
                } else {
                    state = 2; // Durum 2'ye geÃ§
                }
                break;
            case 1:
                std::cout << "Durum 1: a > 10" << std::endl;
                state = 3; // Son duruma geÃ§
                break;
            case 2:
                std::cout << "Durum 2: a <= 10" << std::endl;
                state = 3; // Son duruma geÃ§
                break;
            case 3:
                return; // Program sona erer
        }
    }
}

int main() {
    kontrol_akisi_duzlestir(12); // Girdi deÄŸerine gÃ¶re durum tabanlÄ± akÄ±ÅŸ
    kontrol_akisi_duzlestir(8);  // FarklÄ± girdi ile Ã§aÄŸrÄ±lÄ±yor
    return 0;
}
```

**AÃ§Ä±klama:**
Bu Ã¶rnekte, kontrol akÄ±ÅŸÄ± dÃ¼zleÅŸtirilmiÅŸtir ve `state` deÄŸiÅŸkeni ile duruma baÄŸlÄ± olarak hareket edilir. KoÅŸullu ifadeler yerine, durum geÃ§iÅŸleri yapÄ±lÄ±r. Bu, programÄ±n akÄ±ÅŸÄ±nÄ± tahmin etmeyi zorlaÅŸtÄ±rÄ±r, Ã§Ã¼nkÃ¼ durum tabanlÄ± bir yapÄ± kullanÄ±ldÄ±ÄŸÄ±nda hangi durumda hangi iÅŸlem yapÄ±lacaÄŸÄ± dÄ±ÅŸarÄ±dan gÃ¶rÃ¼nmez hale

gelir.

### 1.2.33 1.11.2. Kod AkÄ±ÅŸÄ±nÄ± Tahmin Edilemez Hale Getirme

**AÃ§Ä±klama:**
Bu Ã¶rnekte, kontrol akÄ±ÅŸÄ±nÄ± dÃ¼zleÅŸtirmek iÃ§in tahmin edilemez durumlar eklenmiÅŸtir. Bu durum, kodun analizini zorlaÅŸtÄ±rÄ±r ve saldÄ±rganlarÄ±n kod akÄ±ÅŸÄ±nÄ± anlamasÄ±nÄ± engeller.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// Rastgele durumlar ile kontrol akÄ±ÅŸÄ±
void tahmin_edilemez_kontrol_akisi(int a) {
    srand(time(0)); // Rastgele sayÄ± Ã¼reteci
    int state = rand() % 3; // Rastgele baÅŸlangÄ±Ã§ durumu

    while (true) {
        switch (state) {
            case 0:
                std::cout << "BaÅŸlangÄ±Ã§ durumu, a: " << a << std::endl;
                if (a > 5) {
                    state = 1; // Durum 1'e geÃ§
                } else {
                    state = 2; // Durum 2'ye geÃ§
                }
                break;
            case 1:
                std::cout << "Durum 1: a > 5" << std::endl;
                state = rand() % 3; // Rastgele durum deÄŸiÅŸtir
                break;
            case 2:
                std::cout << "Durum 2: a <= 5" << std::endl;
                state = rand() % 3; // Rastgele durum deÄŸiÅŸtir
                break;
            case 3:
                std::cout << "Program sona eriyor." << std::endl;
                return; // Program biter
        }
    }
}

int main() {
    tahmin_edilemez_kontrol_akisi(7); // Fonksiyon Ã§aÄŸrÄ±sÄ±
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, rastgele durumlarla kontrol akÄ±ÅŸÄ± yÃ¶netilir. Program her Ã§alÄ±ÅŸtÄ±ÄŸÄ±nda farklÄ± bir baÅŸlangÄ±Ã§ durumu ve farklÄ± bir akÄ±ÅŸ oluÅŸabilir. Bu durum, kodun akÄ±ÅŸÄ±nÄ± anlamayÄ± ve analiz etmeyi zorlaÅŸtÄ±rÄ±r. Kod, tahmin edilemez hale gelerek saldÄ±rganlar iÃ§in daha gÃ¼venli olur.

### 1.2.34 1.12. Ã‡Ä±kÄ±ÅŸ NoktalarÄ±nÄ± Rastgele Hale Getirerek Kodun Ã–ngÃ¶rÃ¼lebilirliÄŸini Azaltma (Randomized Exit Points)

**Teorik AÃ§Ä±klama:**
Ã‡Ä±kÄ±ÅŸ noktalarÄ±nÄ± rastgele hale getirmek, kodun ne zaman sona ereceÄŸini belirsizleÅŸtirir. Bu

teknik, programÄ±n kontrol akÄ±ÅŸÄ±nÄ± tahmin etmeyi zorlaÅŸtÄ±rÄ±r ve analiz araÃ§larÄ±nÄ±n iÅŸini karmaÅŸÄ±klaÅŸtÄ±rÄ±r.

**Ã–rnek Ã–nerisi:**

- Rastgele belirlenen Ã§Ä±kÄ±Åž noktalarÄ±yla programÄ±n beklenmedik yerlerde sona erdiÄŸi bir Ã¶rnek.
- ProgramÄ±n farklÄ± koÅŸullarda farklÄ± Ã§Ä±kÄ±Åž noktalarÄ±na sahip olmasÄ±.

### 1.2.35 .12.1. Rastgele Belirlenen Ã‡Ä±kÄ±Åž NoktalarÄ±yla ProgramÄ±n Beklenmedik Yerlerde Sona Ermesi

**AÃ§Ä±klama:**
Bu Ã¶rnekte, programÄ±n rastgele belirlenen koÅŸullara baÄŸlÄ± olarak farklÄ± Ã§Ä±kÄ±Åž noktalarÄ±na sahip olduÄŸu bir yapÄ± oluÅŸturulmuÅŸtur. Bu yaklaÅŸÄ±m, programÄ±n ne zaman ve nerede sona ereceÄŸini belirsiz hale getirir.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// Rastgele Ã§Ä±kÄ±Åž noktasÄ± belirleyen fonksiyon
void rastgele_cikis(int a) {
    srand(time(0)); // Rastgele sayÄ± Ã¼reteci baÅŸlatÄ±lÄ±yor
    int random_exit = rand() % 3; // 0, 1 veya 2 arasÄ±nda rastgele deÄŸer

    std::cout << "Ä°Åžlem baÅŸlatÄ±ldÄ±..." << std::endl;

    if (a > 10 && random_exit == 0) {
        std::cout << "Ã‡Ä±kÄ±Åž NoktasÄ± 1" << std::endl;
        return; // Program burada sona erer
    }

    if (a < 5 && random_exit == 1) {
        std::cout << "Ã‡Ä±kÄ±Åž NoktasÄ± 2" << std::endl;
        return; // Program burada sona erer
    }

    std::cout << "Normal iÅŸleyiÅŸ devam ediyor." << std::endl;
    // Program buraya kadar devam ederse sona ermez
}

int main() {
    rastgele_cikis(12); // FarklÄ± giriÅŸler ile test ediliyor
    rastgele_cikis(3);
    rastgele_cikis(7);
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, `rastgele_cikis` fonksiyonu rastgele bir Ã§Ä±kÄ±Åž noktasÄ± seÃ§er. EÄŸer belirlenen rastgele koÅŸullar saÄŸlanÄ±rsa, program beklenmedik bir noktada sona erer. Bu yaklaÅŸÄ±m, kodun Ã¶ngÃ¶rÃ¼lebilirliÄŸini azaltÄ±r ve analiz araÃ§larÄ± iÃ§in programÄ±n akÄ±ÅŸÄ±nÄ± takip etmeyi zorlaÅŸtÄ±rÄ±r.

### 1.2.36 1.12.2. ProgramÄ±n FarklÄ± KoÅŸullarda FarklÄ± Ã‡Ä±kÄ±Åž NoktalarÄ±na Sahip OlmasÄ±

**AÃ§Ä±klama:**
Bu Ã¶rnekte, programÄ±n farklÄ± koÅŸullarda rastgele Ã§Ä±kÄ±Åž noktalarÄ±na sahip olmasÄ±

saÄŸlanmÄ±ÅŸtÄ±r. Bu, kodun akÄ±ÅŸÄ±nÄ± tahmin etmeyi zorlaÅŸtÄ±rÄ±r ve kodun analizi sÄ±rasÄ±nda programÄ±n tam olarak ne zaman sona ereceÄŸini belirsiz hale getirir.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// FarklÄ± koÅŸullara gÃ¶re Ã§Ä±kÄ±ÅŸ yapan fonksiyon
void tahmin_edilemez_cikis(int a, int b) {
    srand(time(0)); // Rastgele sayÄ± Ã¼retici
    int random_exit = rand() % 4; // 0, 1, 2 veya 3

    if (a > b && random_exit == 0) {
        std::cout << "Ã‡Ä±kÄ±ÅŸ NoktasÄ± 1: a > b" << std::endl;
        return; // Program sona erer
    }

    if (b > a && random_exit == 1) {
        std::cout << "Ã‡Ä±kÄ±ÅŸ NoktasÄ± 2: b > a" << std::endl;
        return; // Program burada sona erer
    }

    if (a == b && random_exit == 2) {
        std::cout << "Ã‡Ä±kÄ±ÅŸ NoktasÄ± 3: a == b" << std::endl;
        return; // Program burada sona erer
    }

    std::cout << "Program normal ÅŸekilde sona erdi." << std::endl;
}

int main() {
    tahmin_edilemez_cikis(5, 10); // FarklÄ± giriÅŸler ile test ediliyor
    tahmin_edilemez_cikis(10, 5);
    tahmin_edilemez_cikis(7, 7);
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, program farklÄ± giriÅŸ deÄŸerlerine ve rastgele seÃ§ilen Ã§Ä±kÄ±ÅŸ noktalarÄ±na gÃ¶re sona erer. `tahmin_edilemez_cikis` fonksiyonu, hem giriÅŸ koÅŸullarÄ±na hem de rastgele sayÄ±lara gÃ¶re Ã§Ä±kÄ±ÅŸ yapar. Bu yapÄ±, programÄ±n ne zaman sona ereceÄŸini tahmin etmeyi zorlaÅŸtÄ±rarak kodun analiz edilmesini engeller.

### 1.2.37 1.13. Son SÃ¼rÃ¼mde LoglamalarÄ±n Devre DÄ±ÅŸÄ± BÄ±rakÄ±lmasÄ± (Logging Disabled on Release)

**Teorik AÃ§Ä±klama:**
Loglama, geliÅŸtirme sÃ¼recinde faydalÄ± olsa da, son sÃ¼rÃ¼mlerde devre dÄ±ÅŸÄ± bÄ±rakÄ±lmasÄ± gÃ¼venlik aÃ§Ä±sÄ±ndan Ã¶nemlidir. Loglar, hassas bilgileri aÃ§Ä±ÄŸa Ã§Ä±karabilir ve saldÄ±rganlarÄ±n sistem hakkÄ±nda bilgi edinmesini kolaylaÅŸtÄ±rabilir. Bu nedenle, loglamanÄ±n yalnÄ±zca geliÅŸtirme aÅŸamasÄ±nda aktif olmasÄ± saÄŸlanmalÄ± ve son sÃ¼rÃ¼mlerde kapatÄ±lmalÄ±dÄ±r.

**Ã–rnek Ã–nerisi:**

- Derleme aÅŸamasÄ±nda `DEBUG` veya `RELEASE` modlarÄ±na gÃ¶re loglamayÄ± devre dÄ±ÅŸÄ± bÄ±rakan bir makro Ã¶neÄŸi.
- Son sÃ¼rÃ¼mde loglamalarÄ±n tamamen kaldÄ±rÄ±ldÄ±ÄŸÄ± bir uygulama.

### 1.2.38 1.13.1. Derleme AÅŸamasÄ±nda DEBUG veya RELEASE ModlarÄ±na GÃ¶re Loglamayı Devre DÄ±ÅŸÄ± BÄ±rakan Bir Makro Ã–rneÄŸi

**AÃ§Ä±klama:**
Bu Ã¶rnekte, loglama iÅŸlemleri `DEBUG` veya `RELEASE` modlarÄ±na baÄŸlÄ± olarak kontrol edilir. GeliÅŸtirme sÄ±rasÄ±nda (`DEBUG`) loglamalar aktif, son sÃ¼rÃ¼mde (`RELEASE`) devre dÄ±ÅŸÄ± bÄ±rakÄ±lÄ±r.

```
#include <iostream>

// DEBUG veya RELEASE modlarÄ±na gÃ¶re loglama kontrolÃ¼
#ifdef DEBUG
    #define LOG(x) std::cout << "LOG: " << x << std::endl;
#else
    #define LOG(x) // BoÅŸ tanÄ±m, loglama yapÄ±lmaz
#endif

void sistem_bilgisi() {
    LOG("Sistem bilgileri alÄ±nÄ±yor...");
    // DiÄŸer iÅŸlemler...
    std::cout << "Sistem iÅŸlemleri tamamlandÄ±." << std::endl;
}

int main() {
    sistem_bilgisi();
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, `DEBUG` modunda loglamalar aktifken, `RELEASE` modunda loglama makrosu boÅŸ tanÄ±mlanarak loglamalar devre dÄ±ÅŸÄ± bÄ±rakÄ±lÄ±r. Bu, son sÃ¼rÃ¼mde loglarÄ±n Ã§Ä±kmasÄ±nÄ± Ã¶nler ve hassas bilgilerin ifÅŸa olmasÄ±nÄ± engeller.

### 1.2.39 1.13.2. Son SÃ¼rÃ¼mde LoglamalarÄ±n Tamamen KaldÄ±rÄ±ldÄ±ÄŸÄ± Bir Uygulama

**AÃ§Ä±klama:**
Bu Ã¶rnek, son sÃ¼rÃ¼mde tÃ¼m loglama iÅŸlemlerinin tamamen kaldÄ±rÄ±ldÄ±ÄŸÄ± bir yapÄ± iÃ§erir. GeliÅŸtirme aÅŸamasÄ±nda aktif olan loglar, son sÃ¼rÃ¼me geÃ§ildiÄŸinde derleme sÄ±rasÄ±nda tamamen devre dÄ±ÅŸÄ± bÄ±rakÄ±lÄ±r.

```
#include <iostream>

// DEBUG modunda loglama aktif, RELEASE modunda devre dÄ±ÅŸÄ±
void kritik_islem() {
#ifdef DEBUG
    std::cout << "DEBUG: Kritik iÅŸlem baÅŸlatÄ±ldÄ±." << std::endl;
#endif
    // Kritik iÅŸlemler burada gerÃ§ekleÅŸtirilir
    std::cout << "Kritik iÅŸlem tamamlandÄ±." << std::endl;
}

int main() {
    kritik_islem();
    return 0;
}
```

**AÃ§Ä±klama:**
Bu kodda, kritik iÅŸlemler sÄ±rasÄ±nda yalnÄ±zca `DEBUG` modunda loglamalar gÃ¶rÃ¼nÃ¼r. `RELEASE` modunda loglama kodu derleme aÅŸamasÄ±nda tamamen kaldÄ±rÄ±lÄ±r, bu sayede son sÃ¼rÃ¼mde loglama iÅŸlemi gerÃ§ekleÅŸmez ve hassas verilerin sÄ±zdÄ±rÄ±lmasÄ± Ã¶nlenir.

#### 1.2.40 2. Java ve Yorumlanan Diller Ä°Ã§in Kod GÃ¼Ã§lendirme Teknikleri

Java ve diÄŸer yorumlanan dillerde kod gÃ¼Ã§lendirme, gÃ¼venlik aÃ§Ä±klarÄ±nÄ± azaltmak ve geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rmak iÃ§in kullanÄ±lÄ±r.

##### 1.2.40.1 2.1. Proguard ile Kod Obfuske ve Koruma (Proguard Code Obfuscation and Code Shrink Protection)

**Teorik AÃ§Ä±klama:** Proguard, Java kodlarÄ±nÄ± kÃ¼Ã§Ã¼ltme, optimize etme ve obfuske ederek kodun analiz edilmesini zorlaÅŸtÄ±rÄ±r.

**Uygulama Ã–rnekleri:**

1. Proguard yapÄ±landÄ±rma dosyasÄ± ile kodun kÃ¼Ã§Ã¼ltÃ¼lmesi ve optimize edilmesi.
2. Obfuske edilmiÅŸ kodun test edilmesi ve hatalarÄ±n Ã§Ã¶zÃ¼lmesi.
3. Proguard raporlarÄ±nÄ±n analizi ile hangi Ã¶ÄŸelerin obfuske edildiÄŸinin tespiti.

#### 1.2.41 2.1.1 Proguard yapÄ±landÄ±rma dosyasÄ± ile kodun kÃ¼Ã§Ã¼ltÃ¼lmesi ve optimize edilmesi

## 1.3 1. Mobil Android Projesi (Gradle)

**Dosya YapÄ±sÄ±:**

```
/MyAndroidApp
â",   â"œâ"€â"€ app
â",       â"œâ"€â"€ src
â",           â"œâ"€â"€ main
â",               â"œâ"€â"€ java/com/example/myandroidapp/MainActivity.java
â",               â""â"€â"€ res/layout/activity_main.xml
â",       â"œâ"€â"€ build.gradle
â",       â"œâ"€â"€ proguard-rules.pro
â"œâ"€â"€ build.gradle
â"œâ"€â"€ settings.gradle
```

#### 1.3.1 Proje DosyalarÄ±:

**app/build.gradle**

```
android {
    compileSdkVersion 30
    defaultConfig {
        applicationId "com.example.myandroidapp"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```

**proguard-rules.pro**

```
# Keep MainActivity class and its methods
-keep class com.example.myandroidapp.MainActivity { *; }
```

**MainActivity.java**

```java
package com.example.myandroidapp;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

**activity_main.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Proguard!"
        android:textSize="20sp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

#### 1.3.2 NasÄ±l Ã‡alÄ±ÅŸtÄ±rÄ±lÄ±r:

- Android Studio ile projeyi aÃ§Ä±n.
- `Build > Generate Signed Bundle / APK` ile `release` APK oluÅŸturun.
- Proguard otomatik olarak kodu obfuske eder.

### 1.4 2. MasaÃ¼stÃ¼ Java Projesi (Gradle)

**Dosya YapÄ±sÄ±:**

```
/MyDesktopApp
â"œâ"€â"€ src
â",   â""â"€â"€ main
â",       â""â"€â"€ java/com/example/mydesktopapp/Main.java
â"œâ"€â"€ build.gradle
â"œâ"€â"€ proguard-rules.pro
```

#### 1.4.1 Proje DosyalarÄ±:

**build.gradle**

```groovy
plugins {
    id 'application'
}

application {
    mainClass = 'com.example.mydesktopapp.Main'
}

task proguard(type: JavaExec) {
    main = 'proguard.ProGuard'
```

```
    classpath = configurations.proguard
    args = '-injars', "$buildDir/classes/java/main", '-outjars', "$buildDir/classes/obfuscated.jar", '-l
}

configurations {
    proguard
}

dependencies {
    proguard 'net.sf.proguard:proguard-base:6.0.3'
}
```

**proguard-rules.pro**

```
-keep class com.example.mydesktopapp.Main { *; }
```

**Main.java**

```
package com.example.mydesktopapp;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, Proguard!");
    }
}
```

#### 1.4.2 NasÄ±l Ã‡alÄ±ÅŸtÄ±rÄ±lÄ±r:

- Terminalde projenin bulunduÄŸu dizinde ÅŸu komutu Ã§alÄ±ÅŸtÄ±rÄ±n

```
gradlew proguard
```

Obfuske edilmiÅŸ kodu `build/classes/obfuscated.jar` iÃ§inde bulabilirsiniz.

### 1.5 3. JavaCard Projesi (Maven)

**Dosya YapÄ±sÄ±:**

```
/MyJavaCardApp
â"œâ"€â"€ src
â",   â""â"€â"€ main
â",       â""â"€â"€ java/com/example/myjavacardapp/MyJavaCardApplet.java
â"œâ"€â"€ pom.xml
â"œâ"€â"€ proguard-javacard-rules.pro
```

#### 1.5.1 Proje DosyalarÄ±:

**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>myjavacardapp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
```

```
                <groupId>com.github.wvengen</groupId>
                <artifactId>proguard-maven-plugin</artifactId>
                <version>2.0.15</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>proguard</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

**proguard-javacard-rules.pro**

```
-keep class javacard.framework.Applet { *; }
-keep class com.example.myjavacardapp.MyJavaCardApplet { *; }
```

**MyJavaCardApplet.java**

```
package com.example.myjavacardapp;

import javacard.framework.*;

public class MyJavaCardApplet extends Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new MyJavaCardApplet().register(bArray, (short) (bOffset + 1), bArray[bOffset]);
    }
}
```

#### 1.5.2 NasÄ±l Ã‡alÄ±ÅŸtÄ±rÄ±lÄ±r:

- Terminalde projenin bulunduÄŸu dizinde Åžu komutu Ã§alÄ±ÅŸtÄ±rÄ±n

```
mvn clean package
```

Maven, Proguard ile kodu kÃ¼Ã§Ã¼lterek `target` dizininde obfuske edilmiÅŸ bir jar dosyasÄ± oluÅŸturur.

### 1.6 2. MasaÃ¼stÃ¼ Java Projesi (Gradle)

#### 1.6.1 Proguard YapÄ±landÄ±rma DosyasÄ± ile Kodun KÃ¼Ã§Ã¼ltÃ¼lmesi ve Optimize Edilmesi

**1.6.1.1 AÃ§Ä±klama:** Proguard, masaÃ¼stÃ¼ Java uygulamalarÄ±nda kullanÄ±lmayan kodlarÄ± kaldÄ±rarak, kodu kÃ¼Ã§Ã¼ltmek ve optimize etmek iÃ§in kullanÄ±lÄ±r. Bu iÅŸlem, uygulamanÄ±n boyutunu azaltÄ±r ve sÄ±nÄ±flarÄ±, metotlarÄ±, ve deÄŸiÅŸkenleri anlamsÄ±z isimlerle deÄŸiÅŸtirerek kodun analiz edilmesini zorlaÅŸtÄ±rÄ±r.

**YapÄ±landÄ±rma dosyasÄ±:** `proguard-rules.pro`

```
-keep class com.example.mydesktopapp.Main { *; }
-keepclassmembers class * {
    public <init>(...);
}
```

- **AÃ§Ä±klama:**
  Bu yapÄ±landÄ±rmada `Main` sÄ±nÄ±fÄ± korunuyor ve obfuske edilmiyor. Proguardâ€™Ä±n diÄŸer sÄ±nÄ±flarÄ± kÃ¼Ã§Ã¼ltmesine ve optimize etmesine izin veriliyor. Kurallar, belirli sÄ±nÄ±flarÄ± korurken geri kalanlarÄ± obfuske etmeyi saÄŸlar.

### 1.6.2 Obfuske EdilmiÅŸ Kodun Test Edilmesi ve HatalarÄ±n Ã‡Ã¶zÃ¼lmesi

1. **Kodun Derlenmesi ve Obfuske Edilmesi:**
   - Terminalde projenizin bulunduÄŸu dizinde ÅŸu komutu Ã§alÄ±ÅŸtÄ±rarak Proguard ile derleme iÅŸlemini baÅŸlatÄ±n:

     ```
     gradlew proguard
     ```

2. **Test AdÄ±mlarÄ±:**
   - `build/classes/obfuscated.jar` dosyasÄ±na gÃ¶z atÄ±n ve programÄ±n Ã§alÄ±ÅŸÄ±p Ã§alÄ±ÅŸmadÄ±ÄŸÄ±nÄ± doÄŸrulamak iÃ§in test edin.
   - EÄŸer uygulama beklendiÄŸi gibi Ã§alÄ±ÅŸmÄ±yorsa ve bazÄ± kritik sÄ±nÄ±flar veya metotlar obfuske edilmiÅŸse, `proguard-rules.pro` dosyasÄ±na o sÄ±nÄ±f ve metotlarÄ± koruyacak ek kurallar ekleyin.
   - Ã–rnek: EÄŸer `myMethod()` metodu Ã§alÄ±ÅŸmÄ±yorsa ÅŸu kuralÄ± ekleyebilirsiniz:

     ```
     -keep class com.example.mydesktopapp.MyClass {
         public void myMethod();
     }
     ```

### 1.6.3 Proguard RaporlarÄ±nÄ±n Analizi ile Hangi Ã–ÄŸelerin Obfuske EdildiÄŸinin Tespiti

1. **Mapping DosyasÄ±:** Proguard, `mapping.txt` adÄ±nda bir rapor oluÅŸturur. Bu dosya, hangi sÄ±nÄ±flarÄ±n, metodlarÄ±n ve deÄŸiÅŸkenlerin obfuske edildiÄŸini ve hangi isimlerle deÄŸiÅŸtirildiÄŸini gÃ¶sterir. `mapping.txt` dosyasÄ±nÄ± analiz ederek, hangi Ã¶ÄŸelerin korunduÄŸunu ve hangilerinin obfuske edildiÄŸini Ã¶ÄŸrenebilirsiniz.

**Ã–rnek Mapping DosyasÄ±:**

```
com.example.mydesktopapp.Main -> a:
    void main(java.lang.String[]) -> a
    int myVariable -> b
```

- **AÃ§Ä±klama:**
  Bu Ã¶rnekte `com.example.mydesktopapp.Main` sÄ±nÄ±fÄ± `a` olarak, `myVariable` ise `b` olarak deÄŸiÅŸtirilmiÅŸtir. Mapping dosyasÄ±, hata ayÄ±klamak veya korunan sÄ±nÄ±flarÄ±n durumunu kontrol etmek iÃ§in kullanÄ±lÄ±r.

## 1.7 3. JavaCard Projesi (Maven)

### 1.7.1 Proguard YapÄ±landÄ±rma DosyasÄ± ile Kodun KÃ¼Ã§Ã¼ltÃ¼lmesi ve Optimize Edilmesi

**1.7.1.1 AÃ§Ä±klama:** JavaCard projelerinde, Proguard kullanÄ±larak sÄ±nÄ±flar ve metotlar kÃ¼Ã§Ã¼ltÃ¼lÃ¼r ve optimize edilir. JavaCardâ€™Ä±n sÄ±nÄ±rlÄ± kaynaklarÄ± nedeniyle kod kÃ¼Ã§Ã¼ltme ve optimize iÅŸlemleri bÃ¼yÃ¼k Ã¶nem taÅŸÄ±r.

**YapÄ±landÄ±rma dosyasÄ±:** `proguard-javacard-rules.pro`

```
-keep class javacard.framework.Applet { *; }
-keep class com.example.myjavacardapp.MyJavaCardApplet { *; }
```

- **AÃ§Ä±klama:**
  Bu yapÄ±landÄ±rmada JavaCard frameworkâ€™Ã¼ iÃ§inde yer alan `Applet` sÄ±nÄ±fÄ± ve uygulamamÄ±zdaki `MyJavaCardApplet` sÄ±nÄ±fÄ± korunuyor. Geri kalan sÄ±nÄ±flar kÃ¼Ã§Ã¼ltÃ¼lecek ve obfuske edilecektir.

### 1.7.2 Obfuske EdilmiÅŸ Kodun Test Edilmesi ve HatalarÄ±n Ã‡Ã¶zÃ¼lmesi

1. **Kodun Derlenmesi ve Obfuske Edilmesi:**
   - Terminalde şu komutu Ã§alÄ±ÅŸtÄ±rarak Maven ile Proguard iÅŸlemini baÅŸlatÄ±n:

```
mvn clean package
```

2. **Test AdÄ±mlarÄ±:**
   - `target/myjavacardapp-obfuscated.jar` dosyasÄ±na gÃ¶z atÄ±n ve uygulamanÄ±n doÄŸru Ã§alÄ±ÅŸtÄ±ÄŸÄ±nÄ± test edin.
   - EÄŸer bazÄ± sÄ±nÄ±flar ya da metotlar gerektiÄŸi ÅŸekilde Ã§alÄ±ÅŸmÄ±yorsa, bu sÄ±nÄ±flarÄ± `proguard-javacard-rules.pro` dosyasÄ±na ekleyerek koruyabilirsiniz.
   - Ã–rnek: EÄŸer `javacard.framework.ISO7816` sÄ±nÄ±fÄ± obfuske edilmiÅŸse ve hataya neden oluyorsa, bu sÄ±nÄ±fÄ± koruma kuralÄ± ekleyin:

     ```
     -keep class javacard.framework.ISO7816 { *; }
     ```

#### 1.7.3 Proguard RaporlarÄ±nÄ±n Analizi ile Hangi Ã–ÄŸelerin Obfuske EdildiÄŸinin Tespiti

1. **Mapping DosyasÄ±:** Maven ile oluÅŸturulan Proguard raporu, `mapping.txt` dosyasÄ± altÄ±nda bulunur. Bu dosya, hangi sÄ±nÄ±flarÄ±n, metodlarÄ±n ve deÄŸiÅŸkenlerin isimlerinin obfuske edildiÄŸini gÃ¶sterir.

**Ã–rnek Mapping DosyasÄ±:**

```
com.example.myjavacardapp.MyJavaCardApplet -> a:
    void install(byte[], short, byte) -> a
```

- **AÃ§Ä±klama:**
  `com.example.myjavacardapp.MyJavaCardApplet` sÄ±nÄ±fÄ± `a` olarak deÄŸiÅŸtirilmiÅŸ, `install()` metodu ise `a` olarak adlandÄ±rÄ±lmÄ±ÅŸtÄ±r. Bu rapor sayesinde kodun nasÄ±l obfuske edildiÄŸini gÃ¶rebilir ve hangi Ã¶ÄŸelerin deÄŸiÅŸtirilip deÄŸiÅŸtirilmediÄŸini kontrol edebilirsiniz.

##### 1.7.3.1 2.2. Cihaz BaÄŸlama Ã°Ã§in AyrÄ± Parmak Ã°zi Depolama (Separated Fingerprint Storage for Device Binding)

**Teorik AÃ§Ä±klama:** CihazÄ±n benzersiz Ã¶zelliklerini kullanarak, uygulamanÄ±n yalnÄ±zca belirli bir cihazda Ã§alÄ±ÅŸmasÄ±nÄ± saÄŸlamak iÃ§in kullanÄ±lan bir tekniktir.

**Uygulama Ã–rnekleri:**

1. Cihaz parmak izinin ÅŸifrelenerek gÃ¼venli bir ÅŸekilde depolanmasÄ±.
2. Parmak izi doÄŸrulamasÄ± ile uygulamanÄ±n cihaz Ã¼zerinde Ã§alÄ±ÅŸmasÄ±nÄ± saÄŸlama.
3. Parmak izi verilerinin gizlenmesi ve saldÄ±rÄ±lara karÅŸÄ± korunmasÄ±.

#### 1.7.4 2.2.1. Cihaz Parmak Ã°zinin Åžifrelenerek GÃ¼venli Bir Åžekilde DepolanmasÄ±

**AÃ§Ä±klama:**
CihazÄ±n benzersiz bir parmak izi, genellikle donanÄ±m veya sistem Ã¶zelliklerinden alÄ±nan bilgilerden oluÅŸturulur. Bu parmak izi ÅŸifrelenerek gÃ¼venli bir ÅŸekilde cihazda saklanÄ±r.

**Java Ã–rneÄŸi:**

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class DeviceFingerprint {

    // Benzersiz bir cihaz parmak izi oluÅŸturma
    public static String generateFingerprint(String deviceID, String hardwareSerial) throws NoSuchAlgori
        String rawFingerprint = deviceID + hardwareSerial;

        // SHA-256 ile cihaz parmak izini ÅŸifreleme
```

```
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] encodedHash = digest.digest(rawFingerprint.getBytes(StandardCharsets.UTF_8));

        // ÅžifrelenmiÅŸ parmak izini Base64 ile encode etme
        return Base64.getEncoder().encodeToString(encodedHash);
    }

    public static void main(String[] args) {
        try {
            // Benzersiz cihaz bilgileri
            String deviceID = "device12345";
            String hardwareSerial = "hwserial67890";

            // Parmak izi oluÅŸturulup ÅŸifreleniyor
            String fingerprint = generateFingerprint(deviceID, hardwareSerial);
            System.out.println("ÅžifrelenmiÅŸ Parmak Ä°zi: " + fingerprint);

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

**AÃ§Ä±klama:**

- `generateFingerprint` metodu, cihazdan elde edilen `deviceID` ve `hardwareSerial` bilgilerini kullanarak bir cihaz parmak izi oluÅŸturur.
- Parmak izi SHA-256 algoritmasÄ±yla ÅŸifrelenir ve Base64 formatÄ±nda saklanabilir

### 1.7.5 2.2.2. Parmak Ä°zi DoÄŸrulamasÄ± ile UygulamanÄ±n Cihaz Ãœzerinde Ã‡alÄ±ÅŸmasÄ±nÄ± SaÄŸlama

**AÃ§Ä±klama:**
Cihaz parmak izi, yalnÄ±zca belirli bir cihazda uygulamanÄ±n Ã§alÄ±ÅŸmasÄ±nÄ± saÄŸlamak iÃ§in doÄŸrulanÄ±r. Parmak izi eÅŸleÅŸmediÄŸinde uygulama Ã§alÄ±ÅŸtÄ±rÄ±lmaz.

**Java Ã–rneÄŸi:**

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class DeviceFingerprintCheck {

    // Daha Ã¶nce saklanan ÅŸifrelenmiÅŸ parmak izi
    private static final String STORED_FINGERPRINT = "storedFingerprintValue"; // Bu deÄŸeri parmak izi

    public static String generateFingerprint(String deviceID, String hardwareSerial) throws NoSuchAlgori
        String rawFingerprint = deviceID + hardwareSerial;

        // Parmak izini tekrar oluÅŸturma
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] encodedHash = digest.digest(rawFingerprint.getBytes(StandardCharsets.UTF_8));

        return Base64.getEncoder().encodeToString(encodedHash);
    }

    public static boolean verifyFingerprint(String deviceID, String hardwareSerial) throws NoSuchAlgori
        // Åžu anki cihazÄ±n parmak izi
        String currentFingerprint = generateFingerprint(deviceID, hardwareSerial);
```

```
        // Daha Ã¶nce saklanan parmak izi ile karÅŸÄ±laÅŸtÄ±rma
        return STORED_FINGERPRINT.equals(currentFingerprint);
    }

    public static void main(String[] args) {
        try {
            // GerÃ§ek cihaz bilgileri (Ã¶rnek olarak)
            String deviceID = "device12345";
            String hardwareSerial = "hwserial67890";

            // Parmak izi doÄŸrulama
            if (verifyFingerprint(deviceID, hardwareSerial)) {
                System.out.println("Parmak izi doÄŸrulandÄ±, uygulama bu cihazda Ã§alÄ±ÅŸabilir.");
            } else {
                System.out.println("Parmak izi doÄŸrulanamadÄ±, uygulama bu cihazda Ã§alÄ±ÅŸamaz.");
            }

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

**AÃ§Ä±klama:**

- `verifyFingerprint` metodu, mevcut cihazÄ±n parmak izini daha Ã¶nce depolanan parmak izi ile karÅŸÄ±laÅŸtÄ±rÄ±r.
- EÄŸer parmak izleri eÅŸleÅŸirse, uygulama cihazda Ã§alÄ±ÅŸabilir.

### 1.7.6 2.2.3. Parmak Ä°zi Verilerinin Gizlenmesi ve SaldÄ±rÄ±lara KarÅŸÄ± KorunmasÄ±

**AÃ§Ä±klama:**
Parmak izi verileri gÃ¼venli bir ÅŸekilde saklanmalÄ±dÄ±r. ÅžifrelenmiÅŸ parmak izi verisi bir dosyada veya gÃ¼venli bir veri deposunda saklanabilir.

**Java Ã–rneÄŸi:**

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class SecureFingerprintStorage {

    // Åžifreleme anahtarÄ± (GÃ¼venli bir ÅŸekilde saklanmalÄ±)
    private static final String KEY = "1234567890123456"; // 16-byte key


    // Cihaz parmak izini AES ile ÅŸifreleme
    public static String encryptFingerprint(String fingerprint) throws Exception {
        SecretKey secretKey = new SecretKeySpec(KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedData = cipher.doFinal(fingerprint.getBytes());
        return Base64.getEncoder().encodeToString(encryptedData);
    }

    // ÅžifrelenmiÅŸ cihaz parmak izini Ã§Ã¶zme
    public static String decryptFingerprint(String encryptedFingerprint) throws Exception {
```

```
        SecretKey secretKey = new SecretKeySpec(KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedData = cipher.doFinal(Base64.getDecoder().decode(encryptedFingerprint));
        return new String(decryptedData);
    }

    public static void main(String[] args) {
        try {
            String fingerprint = "uniqueDeviceFingerprint"; // Ã-rnek parmak izi

            // Parmak izini ÅŸifreleme
            String encryptedFingerprint = encryptFingerprint(fingerprint);
            System.out.println("ÅžifrelenmiÅŸ Parmak Ä°zi: " + encryptedFingerprint);

            // Parmak izini Ã§Ã¶zme
            String decryptedFingerprint = decryptFingerprint(encryptedFingerprint);
            System.out.println("Ã‡Ã¶zÃ¼len Parmak Ä°zi: " + decryptedFingerprint);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**AÃ§Ä±klama:**

- `encryptFingerprint` metodu, cihaz parmak izini AES ile ÅŸifreler.
- `decryptFingerprint` metodu, ÅŸifrelenmiÅŸ parmak izini Ã§Ã¶zer. Bu sayede parmak izi gÃ¼venli bir ÅŸekilde saklanÄ±r ve yalnÄ±zca yetkili kiÅŸiler tarafÄ±ndan eriÅŸilebilir.

### 1.7.7 Ã–zet:

1. **Parmak Ä°zinin Åžifrelenerek DepolanmasÄ±:** CihazÄ±n benzersiz Ã¶zelliklerinden parmak izi oluÅŸturulup ÅŸifrelenir ve gÃ¼venli bir ÅŸekilde saklanÄ±r.
2. **Parmak Ä°zi DoÄŸrulamasÄ±:** UygulamanÄ±n belirli bir cihazda Ã§alÄ±ÅŸÄ±p Ã§alÄ±ÅŸmadÄ±ÄŸÄ± doÄŸrulanÄ±r.
3. **Verilerin Gizlenmesi ve KorunmasÄ±:** Parmak izi verileri AES gibi gÃ¼venli algoritmalarla ÅŸifrelenir ve saldÄ±rÄ±lara karÅŸÄ± korunur.

#### 1.7.7.1 2.3. Yerel KÃ¼tÃ¼phane JNI API Obfuske Etme (Native Library JNI API Obfuscation) Teorik AÃ§Ä±klama:

Java Native Interface (JNI) kullanÄ±larak Ã§aÄŸrÄ±lan yerel kÃ¼tÃ¼phanelerin obfuske edilmesi, geri mÃ¼hendislik iÅŸlemlerini zorlaÅŸtÄ±rÄ±r.

**Uygulama Ã–rnekleri:**

1. JNI fonksiyon isimlerinin rastgele karakterlerle deÄŸiÅŸtirilmesi.
2. JNI parametrelerinin gizlenmesi ve anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rma.
3. JNI hata yÃ¶netimi ile saldÄ±rganlarÄ±n hatalarÄ± analiz etmesini engelleme.

---

### 1.7.8 2.3.1. JNI Fonksiyon Ä°simlerinin Rastgele Karakterlerle DeÄŸiÅŸtirilmesi

**AÃ§Ä±klama:**
JNI fonksiyonlarÄ±nÄ±n isimleri rastgele karakterlerle deÄŸiÅŸtirilerek, yerel kÃ¼tÃ¼phaneye yapÄ±lan Ã§aÄŸrÄ±larÄ±n anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±rabilirsiniz. Bu, kodun geri mÃ¼hendislik iÅŸlemine karÅŸÄ± korunmasÄ±nÄ± saÄŸlar.

**Java Kodu:**

```
public class MyJNIExample {
    // Rastgele isimlendirilmiÅŸ JNI fonksiyonu
    public native void nJx57F(); // Rastgele isimli JNI fonksiyonu

    static {
        System.loadLibrary("myNativeLib");
    }

    public static void main(String[] args) {
        MyJNIExample example = new MyJNIExample();
        example.nJx57F(); // Rastgele isimli fonksiyonu Ã§aÄŸÄ±rma
    }
}
```

**C Kodu (Yerel KÃ¼tÃ¼phane - myNativeLib.c):**

```
#include <jni.h>
#include <stdio.h>
#include "MyJNIExample.h"

// Rastgele isimlendirilmiÅŸ JNI fonksiyonu
JNIEXPORT void JNICALL Java_MyJNIExample_nJx57F(JNIEnv *env, jobject obj) {
    printf("Yerel kÃ¼tÃ¼phane fonksiyonu Ã§alÄ±ÅŸtÄ±!\n");
}
```

**AÃ§Ä±klama:**

- JNI fonksiyonu `nJx57F()` gibi rastgele bir isimle tanÄ±mlanmÄ±ÅŸtÄ±r. Bu, geri mÃ¼hendislik yapan bir kiÅŸinin fonksiyonun ne yaptÄ±ÄŸÄ±nÄ± anlamasÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

### 1.7.9 2.3.2. JNI Parametrelerinin Gizlenmesi ve AnlaÅŸÄ±lmasÄ±nÄ± ZorlaÅŸtÄ±rma

**AÃ§Ä±klama:**
JNI fonksiyonlarÄ±na gÃ¶nderilen parametreler anlaÅŸÄ±lmasÄ±nÄ± zorlaÅŸtÄ±racak ÅŸekilde gizlenebilir. Ã–rneÄŸin, parametreler bir diziyle veya ÅŸifreli veriyle gÃ¶nderilebilir.

**Java Kodu:**

```
public class MyJNIExample {
    // Rastgele parametre isimlendirilmiÅŸ JNI fonksiyonu
    public native void obfuscatedJNI(byte[] encryptedData);

    static {
        System.loadLibrary("myNativeLib");
    }

    public static void main(String[] args) {
        MyJNIExample example = new MyJNIExample();

        // Parametre olarak ÅŸifrelenmiÅŸ veri gÃ¶nderme
        byte[] encryptedData = { 0x12, 0x34, 0x56 };
        example.obfuscatedJNI(encryptedData);
    }
}
```

**C Kodu (Yerel KÃ¼tÃ¼phane - myNativeLib.c):**

```
#include <jni.h>
#include <stdio.h>
#include "MyJNIExample.h"
```

```
// Rastgele parametreli JNI fonksiyonu
JNIEXPORT void JNICALL Java_MyJNIExample_obfuscatedJNI(JNIEnv *env, jobject obj, jbyteArray encryptedDa
    jsize length = (*env)->GetArrayLength(env, encryptedData);
    jbyte *data = (*env)->GetByteArrayElements(env, encryptedData, 0);

    printf("Veri uzunluÄŸu: %d\n", length);

    for (int i = 0; i < length; i++) {
        printf("Veri %d: %x\n", i, data[i]);
    }

    (*env)->ReleaseByteArrayElements(env, encryptedData, data, 0);
}
```

**AÃ§Ä±klama:**

- JNI fonksiyonu bir byte dizisi alÄ±r ve veriler ÅŸifrelenmiÅŸ veya anlamsÄ±z ÅŸekilde saklanarak fonksiyonun ne yaptÄ±ÄŸÄ± daha zor anlaÅŸÄ±lÄ±r hale getirilir.

### 1.7.10 2.3.3. JNI Hata YÃ¶netimi ile SaldÄ±rganlarÄ±n HatalarÄ± Analiz Etmesini Engelleme

**AÃ§Ä±klama:**
JNI fonksiyonlarÄ±nda meydana gelen hatalar saldÄ±rganlar tarafÄ±ndan analiz edilmemelidir. Bunun iÃ§in Ã¶zel hata yÃ¶netimi uygulanÄ±r ve detaylÄ± hata mesajlarÄ± saklanÄ±r.

**Java Kodu:**

```
public class MyJNIExample {
    public native void performTask();

    static {
        System.loadLibrary("myNativeLib");
    }

    public static void main(String[] args) {
        MyJNIExample example = new MyJNIExample();
        try {
            example.performTask();
        } catch (Exception e) {
            System.out.println("Bir hata oluÅŸtu.");
        }
    }
}
```

**C Kodu (Yerel KÃ¼tÃ¼phane - myNativeLib.c):**

```
#include <jni.h>
#include <stdio.h>
#include "MyJNIExample.h"

// Ã-zel hata yÃ¶netimi
JNIEXPORT void JNICALL Java_MyJNIExample_performTask(JNIEnv *env, jobject obj) {
    // Hata mesajÄ± detaysÄ±z
    if (/* Bir hata meydana gelirse */) {
        printf("Bir hata meydana geldi\n");
        return;
    }

    // Normal iÅŸlem
```

```
    printf("GÃ¶rev baÅŸarÄ±yla tamamlandÄ±.\n");
}
```

**AÃ§Ä±klama:**

- Hata meydana geldiÄŸinde yalnÄ±zca genel bir mesaj (“Bir hata meydana geldi”) gÃ¶sterilir. Bu, saldÄ±rganlarÄ±n hatalardan fazla bilgi edinmesini engeller.

### 1.7.11 Ã–zet:

1. **JNI Fonksiyon Ä°simlerinin RastgeleleÅŸtirilmesi:** Fonksiyon isimlerini rastgele karakterlerle deÄŸiÅŸtirerek analiz edilmeyi zorlaÅŸtÄ±rabilirsiniz.
2. **JNI Parametrelerinin Gizlenmesi:** Parametreler ÅŸifrelenmiÅŸ veya gizlenmiÅŸ formatta gÃ¶nderilerek ne yapÄ±ldÄ±ÄŸÄ±nÄ± anlamayÄ± zorlaÅŸtÄ±rabilirsiniz.
3. **JNI Hata YÃ¶netimi:** Hatalar detaylÄ± mesajlarla paylaÅŸÄ±lmaz, bu da saldÄ±rganlarÄ±n analizine karÅŸÄ± koruma saÄŸlar.

#### 1.7.11.1 2.4. Statik Dizelerin Obfuske Edilmesi (Obfuscation of Static Strings) Teorik

**AÃ§Ä±klama:** Statik dizeler, saldÄ±rganlarÄ±n geri mÃ¼hendislik iÅŸlemleri sÄ±rasÄ±nda kullanabileceÄŸi Ã¶nemli bilgiler iÃ§erir. Bu dizelerin obfuske edilmesi, gÃ¼venliÄŸi artÄ±rÄ±r.

**Uygulama Ã–rnekleri:**

1. Statik dizelerin ÅŸifrelenmesi ve Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lmesi.
2. Dizelerin obfuske edilerek anlamlarÄ±nÄ±n gizlenmesi.
3. Rastgele dize oluÅŸturma ve manipÃ¼lasyon teknikleri ile gÃ¼venliÄŸi artÄ±rma.

### 1.7.12 2.4.1. Statik Dizelerin Åžifrelenmesi ve Ã‡alÄ±ÅŸma AnÄ±nda Ã‡Ã¶zÃ¼lmesi

**AÃ§Ä±klama:**
Statik dizeler genellikle Ã¶nemli bilgiler iÃ§erir (Ã¶rneÄŸin, API anahtarlarÄ± veya kullanÄ±cÄ± bilgileri) ve geri mÃ¼hendislik yapan bir saldÄ±rgan bu bilgilere kolayca eriÅŸebilir. Bu nedenle, statik dizeler şifrelenir ve Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lerek gÃ¼venlik artÄ±rÄ±lÄ±r.

**Java Kodu:**

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class StaticStringObfuscation {

    private static final String KEY = "1234567890123456"; // Åžifreleme anahtarÄ± (16-byte AES)

    // Statik dizenin ÅŸifrelenmiÅŸ hali (Ã¶rneÄŸin bir API anahtarÄ±)
    private static final String ENCRYPTED_STRING = "Y2hlY2tfdGhpcyBzdHJpbmdf";

    // Åžifreleme fonksiyonu (dizeleri ÅŸifrelemek iÃ§in)
    public static String encrypt(String data) throws Exception {
        SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Åžifre Ã§Ã¶zme fonksiyonu
    public static String decrypt(String encryptedData) throws Exception {
        SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
```

```
        byte[] decodedBytes = Base64.getDecoder().decode(encryptedData);
        byte[] decryptedBytes = cipher.doFinal(decodedBytes);
        return new String(decryptedBytes);
    }

    public static void main(String[] args) {
        try {
            // Statik dizenin ÅŸifresini Ã§Ã¶zme
            String decryptedString = decrypt(ENCRYPTED_STRING);
            System.out.println("Ã‡Ã¶zÃ¼len dize: " + decryptedString);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**AÃ§Ä±klama:**

- `ENCRYPTED_STRING` deÄŸiÅŸkeni ÅŸifrelenmiÅŸ bir statik dizeyi temsil eder. Program Ã§alÄ±ÅŸÄ±rken bu ÅŸifre Ã§Ã¶zÃ¼lerek gerÃ§ek veri elde edilir.
- Åžifreleme AES algoritmasÄ±yla yapÄ±lmÄ±ÅŸtÄ±r ve Ã§Ã¶zÃ¼m Base64 kodlamasÄ± ile gerÃ§ekleÅŸtirilmiÅŸtir.

### 1.7.13 2.4.2. Dizelerin Obfuske Edilerek AnlamlarÄ±nÄ±n Gizlenmesi

**AÃ§Ä±klama:**
Dizelerin doÄŸrudan kodda yer almasÄ±, bu dizelerin anlamlarÄ±nÄ± aÃ§Ä±k eder. Dizeleri obfuske etmek, saldÄ±rganlarÄ±n anlamlarÄ±nÄ± anlamalarÄ±nÄ± zorlaÅŸtÄ±rÄ±r.

**Java Kodu:**

```
public class StaticStringObfuscation {

    // Obfuske edilmiÅŸ dizeler
    private static final char[] OBFUSCATED_STRING = { 'J', 'a', 'v', 'a', 'I', 's', 'S', 'e', 'c', 'u',

    // Dizeyi Ã§Ã¶zÃ¼mleyerek gerÃ§ek deÄŸerini elde etme
    public static String decodeObfuscatedString() {
        StringBuilder decodedString = new StringBuilder();
        for (char c : OBFUSCATED_STRING) {
            decodedString.append(c);
        }
        return decodedString.toString();
    }

    public static void main(String[] args) {
        // Obfuske edilmiÅŸ dizeyi Ã§Ã¶zme ve gÃ¶sterme
        String decoded = decodeObfuscatedString();
        System.out.println("Ã‡Ã¶zÃ¼len dize: " + decoded);
    }
}
```

**AÃ§Ä±klama:**

- `OBFUSCATED_STRING` dizisi, dizeyi karakter karakter saklayarak doÄŸrudan gÃ¶rÃ¼nmesini engeller.
- `decodeObfuscatedString()` metodu, obfuske edilmiÅŸ dizeyi Ã§Ã¶zer ve anlamlÄ± hale getirir. Bu yaklaÅŸÄ±m, dizelerin aÃ§Ä±kÃ§a gÃ¶rÃ¼nmemesini saÄŸlar.

### 1.7.14 2.4.3. Rastgele Dize OluÅŸturma ve ManipÃ¼lasyon Teknikleri ile GÃ¼venliÄŸi ArtÄ±rma

**AÃ§Ä±klama:**
Rastgele dize oluÅŸturma ve bu dizeleri Ã§alÄ±ÅŸma anÄ±nda manipÃ¼le etme, dizelerin aÃ§Ä±kÃ§a kodda gÃ¶rÃ¼nmesini engeller ve analiz edilmesini zorlaÅŸtÄ±rÄ±r.

**Java Kodu:**

```
import java.util.Random;

public class RandomStringObfuscation {

    // Rastgele bir dize oluÅŸturma fonksiyonu
    public static String generateRandomString(int length) {
        String characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        StringBuilder randomString = new StringBuilder();
        Random random = new Random();

        for (int i = 0; i < length; i++) {
            randomString.append(characters.charAt(random.nextInt(characters.length())));
        }

        return randomString.toString();
    }

    // ManipÃ¼le edilen dizeyi elde etme
    public static String manipulateString(String input) {
        return input.substring(0, input.length() / 2); // Dizenin yarÄ±sÄ±nÄ± kullan
    }

    public static void main(String[] args) {
        // Rastgele bir dize oluÅŸturma
        String randomString = generateRandomString(16);
        System.out.println("OluÅŸturulan rastgele dize: " + randomString);

        // Dizeyi manipÃ¼le etme
        String manipulatedString = manipulateString(randomString);
        System.out.println("ManipÃ¼le edilmiÅŸ dize: " + manipulatedString);
    }
}
```

**AÃ§Ä±klama:**

- `generateRandomString()` metodu, rastgele bir dize oluÅŸturur.
- `manipulateString()` metodu bu diziyi manipÃ¼le eder. Bu teknik, sabit bir dize yerine dinamik ve manipÃ¼le edilmiÅŸ bir dize kullanarak dizelerin analiz edilmesini zorlaÅŸtÄ±rÄ±r.

### 1.7.15 Ã–zet:

1. **Statik Dizelerin Åžifrelenmesi ve Ã‡alÄ±ÅŸma AnÄ±nda Ã‡Ã¶zÃ¼lmesi:** Statik dizeler ÅŸifrelenerek saklanÄ±r ve Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lÃ¼r.
2. **Dizelerin Obfuske Edilerek Gizlenmesi:** Dizeler karakter dizisi olarak saklanÄ±r ve Ã§alÄ±ÅŸma anÄ±nda Ã§Ã¶zÃ¼lerek gizlenir.
3. **Rastgele Dize OluÅŸturma ve ManipÃ¼lasyon Teknikleri:** Rastgele oluÅŸturulmuÅŸ dizeler, statik olmaktan Ã§Ä±kar ve manipÃ¼le edilerek gÃ¼venlik artÄ±rÄ±lÄ±r.

## 1.8 HaftanÄ±n Ã–zeti ve Gelecek Hafta

### 1.8.1 Bu Hafta:

- **Kod GÃ¼Ã§lendirme Teknikleri (C/C++ ve Java)**
- **Obfuske Teknikleri ve UygulamalarÄ±**

### 1.8.2 Gelecek Hafta:

- **SaldÄ±rÄ± AÄŸaÃ§larÄ± ve GÃ¼venlik Modelleri**
- **SaldÄ±rÄ± YÃ¶ntemleri ve GÃ¼venli Ä°letiÅŸim**

$4. Hafta - Sonu$