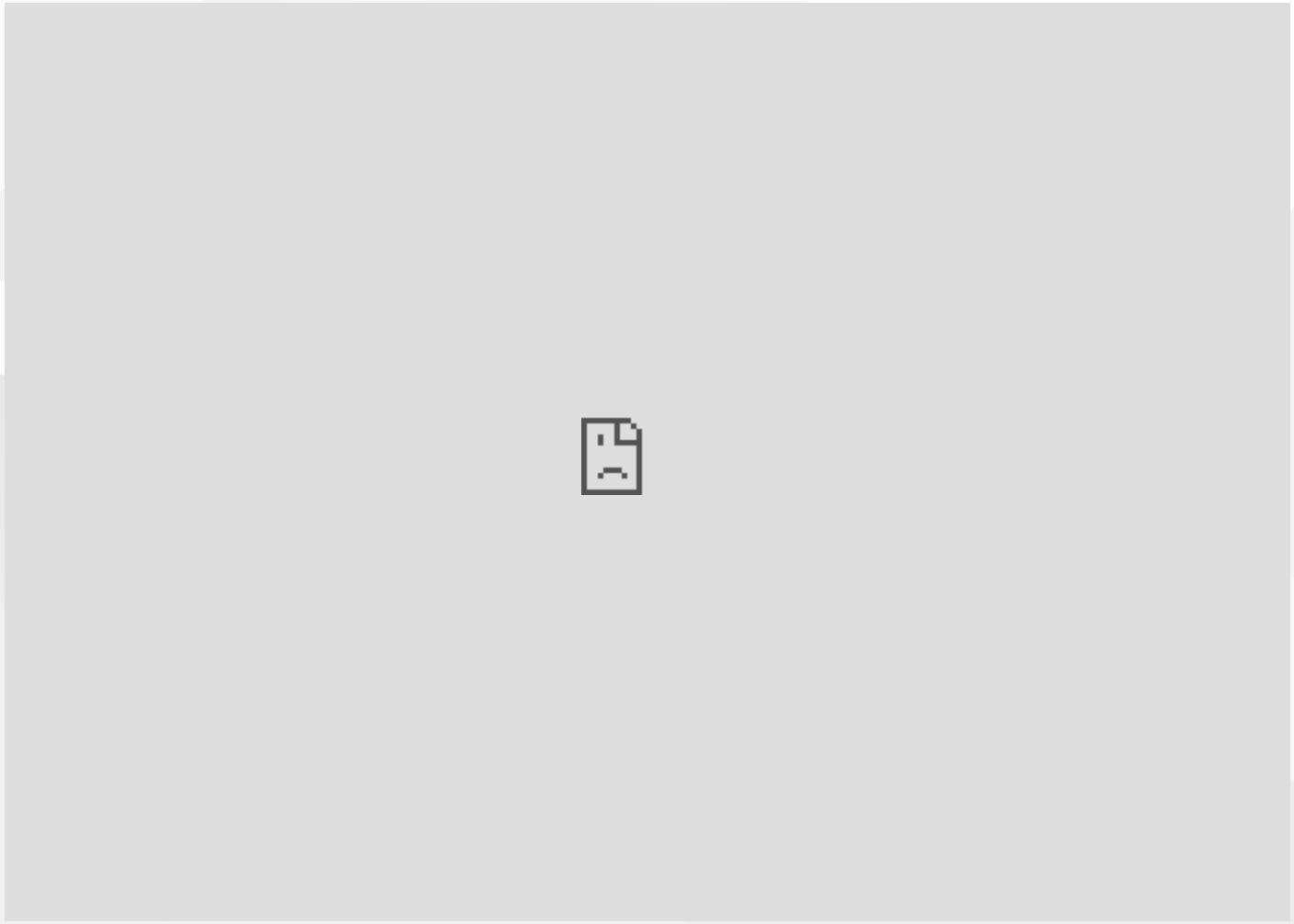
CEN429 Güvenli Programlama

Hafta-7

Kod Karartma (Obfuscation) ve Çeşitlendirme Teknikleri

İndir

- PDF
- DOC
- SLIDE
- PPTX



Outline

- Kod Karartma ve Çeşitlendirme Teknikleri
- Statik ve Dinamik Kod Karartma
- Sanallaştırma ve Şifreleme

Hafta-7: Kod Karartma (Code Obfuscation) ve Çeşitlendirme (Diversifications)

Kod karartma ve çeşitlendirme teknikleri, yazılımın güvenliğini artırmak amacıyla kaynak kodunun ve işlevlerinin karmaşık hale getirilmesini içerir. Bu hafta, bu teknikleri ve bunların uygulamalarını inceleyeceğiz. Bu yöntemler, özellikle yazılımların tersine mühendislikten korunması ve saldırıların zorlaştırılması için kritik öneme sahiptir.

1. Tigress Nedir?

Teorik Açıklama: Tigress, programları dönüştürmek, karartmak ve karmaşık hale getirmek için kullanılan bir araçtır. Karartma teknikleri ile yazılımların tersine mühendislikten korunmasını sağlar. Farklı karartma teknikleri sunarak kodun analizini zorlaştırır.

2. Kod Karartma Teknikleri (Types of Obfuscation)

Teorik Açıklama: Kod karartma, kodu insan ve araçlar tarafından anlaşılması zor hale getirir. Aşağıdaki teknikler kod karartmanın temel yöntemlerindendir:

- **Abstraction Transformations:** Modül yapıları, sınıflar, fonksiyonlar vb. yapıların yok edilmesi.
- **Data Transformations:** Veri yapılarını yeni temsillerle değiştirmek.
- **Control Transformations:** Kontrol yapılarının (if, while, repeat vb.) yok edilmesi.
- **Dynamic Transformations:** Programın çalışma zamanında değişiklik yapması.

3. Statik Kod Karartma (Static Obfuscation)

Teorik Açıklama: Statik karartma, programın çalışma zamanında sabit kalan karartma türüdür. Programın yapısını değiştirir ancak çalışırken değişmez. Aşağıdaki teknikler bu kategoridedir:

- **Bogus Control Flow:** Programın kontrol akışını karmaşık hale getirir. Gerçek olmayan kontrol yapıları eklenir, ölü dallar ve gereksiz dallar kullanılır.
- **Control Flow Flattening:** Kontrol yapılarının yapılarını bozarak kodu dümdüz hale getirir.

Uygulama Örnekleri:

1. Kodda gereksiz dallanmalar ve ölü dallar ekleyerek kontrol akışını zorlaştırmak.
2. Fonksiyonların içine sahte işlemler yerleştirmek.

4. Opaque Predicates ve Kırma (Breaking Opaque Predicates)

Teorik Açıklama: Opaque Predicates, her zaman sabit bir değere sahip olan, ancak dışarıdan bakıldığında değişiyormuş gibi görünen koşul ifadeleridir. Bu koşulların karmaşık matematiksel veya mantıksal ilişkilerle oluşturulması, kodun analiz edilmesini zorlaştırır.

Uygulama Örnekleri:

1. **Opaque Predicates** kullanarak sabit koşullar oluşturma.
2. Opaque predicates'i kırma teknikleri ile matematiksel analizler yaparak bu yapıları çözme.

5. Şifreleme Tabanlı Sayısal Dönüşümler (Encoding Integer Arithmetic)

Teorik Açıklama: Sayılar üzerinde karmaşık matematiksel dönüşümler kullanarak orijinal işlemleri gizleme. Örneğin, toplama işlemini karmaşık matematiksel ifadelerle değiştirme, tersine mühendisliği zorlaştırır.

Uygulama Örnekleri:

1. **x + y** gibi basit aritmetik işlemleri gizleyerek yerine daha karmaşık matematiksel işlemler yerleştirme.
2. Dönüştürülmüş sayısal işlemler üzerinde çalışarak orijinal aritmetik yapıyı geri çözme.

6. Linear Transformation ve Sayısal Dönüşümler (Linear Transformation and Number-Theoretic Tricks)

Teorik Açıklama: Doğrusal dönüşümler, orijinal veriyi karmaşık matematiksel dönüşümlerden geçirerek gizler. Bu dönüşümler geri döndürülemez değildir, ancak analiz edilmesi zordur.

Uygulama Örnekleri:

1. Mod 2^32 gibi büyük modüler aritmetiklerle doğrusal dönüşümler yaparak sayısal işlemleri gizleme.
2. Euclid'in Genişletilmiş Algoritması gibi matematiksel yöntemlerle ters dönüşümleri yapma.

7. Sanallaştırma (Virtualization)

Teorik Açıklama: Sanallaştırma, kodun doğrudan CPU'da çalıştırılması yerine bir sanal makine (interpreter) üzerinde çalıştırılmasını sağlar. Bu yöntemle, programın çalışma zamanında sürekli olarak çevrimi yapılır ve kodun tersine mühendisliği zorlaştırılır.

Uygulama Örnekleri:

1. Programın tüm komutlarını bir interpreter aracılığıyla çalıştırarak orijinal kodu gizlemek.
2. Interpreter bazlı sanallaştırmalarla kodun sürekli olarak değişken tutulması.

8. Çeşitlendirme (Diversity)

Teorik Açıklama: Çeşitlendirme, her bir programın farklı bir versiyonunu oluşturarak, kodun sabit bir yapıda olmamasını sağlar. Bu, virüslerin veya kötü niyetli yazılımların kodu analiz etmesini zorlaştırır.

Uygulama Örnekleri:

1. Aynı işlevi yerine getiren ancak farklı görünümlerdeki kod yapıları oluşturma.
2. Her kod versiyonunda küçük yapısal değişiklikler yaparak kodun analiz edilmesini zorlaştırma.

9. Şifreleme ve Sayısal Dönüşümler (Encoding and Transforming)

Teorik Açıklama: Kodun bazı bölümleri, özel şifreleme algoritmalarıyla gizlenebilir. Bu, kodun analizini zorlaştıran başka bir karartma tekniğidir. Özellikle sayılar üzerinde şifreleme ve dönüşümler uygulanabilir.

Uygulama Örnekleri:

1. Kod içinde kullanılan sayıları şifreleyerek bu sayıların analizini zorlaştırma.
2. Şifrelenmiş sayıların çözümlerini analiz ederek orijinal değerleri geri döndürme.

10. Opaque İfadeler ve Dinamik Karartma (Opaque Expressions and Dynamic Obfuscation)

Teorik Açıklama: Opaque ifadeler, kodun belirli kısımlarının karmaşık koşullar altında değerlendirilmesini sağlar. Dinamik karartma, kodun çalışma zamanında sürekli olarak dönüştürülmesi ve değişken tutulmasını içerir.

Uygulama Örnekleri:

1. Kodun çalıştığı sırada sürekli olarak dönüşümler uygulayarak analiz edilmesini zorlaştırmak.
2. Çalışma zamanında kodu yeniden yapılandırarak sabit kalmasını engellemek.

$7. Hafta - Sonu$