CE407 Güvenli Programlama

Hafta-5

Native C/C++ için RASP Teknikleri

İndir PDF, DOCX, SLIDE, PPTX

Outline

- RASP (Çalışma Zamanı Uygulama Koruması) Nedir?
- Native C/C++ İçin RASP Teknikleri
- Caller APK Hash Doğrulama
- Root Tespiti ve LD Preload Koruması

Hafta-5: RASP (Runtime Application Self-Protection) Native C/C++ Tarafı

Runtime Application Self-Protection (RASP), uygulamaların çalışma zamanında kendi güvenliklerini sağlamalarını mümkün kılan bir güvenlik yaklaşımıdır. Native C/C++ uygulamalarında, RASP kullanarak çeşitli güvenlik kontrolleri gerçekleştirilebilir. Bu ders kapsamında RASP teknikleri detaylıca açıklanacak ve uygulama örnekleriyle pekiştirilecektir.

1. Çalışma Zamanında Kod Bloklarının Checksum Doğrulaması (Runtime CodeBlock Checksum Verification)

Teorik Açıklama: Çalışma zamanında belirli kod bloklarının hash veya checksum değerleri doğrulanarak, kodun değiştirilip değiştirilmediği tespit edilir. Bu yöntem, kod manipülasyonlarına ve kötü niyetli müdahalelere karşı bir koruma sağlar.

Uygulama Örnekleri:

1. Herhangi bir kod bloğunun checksum değerini hesaplama ve çalışma sırasında bu değeri karşılaştırma.
2. Değişiklik tespit edildiğinde programın kapanması veya hatalı bir sonuç üretmesi.
3. Önemli fonksiyonların ve kritik kod parçalarının checksum doğrulaması ile korunması.

2. Caller APK Hash ve İmza Doğrulaması (Caller APK Hash Verification & Signature Verification)

Teorik Açıklama: APK dosyalarının hash ve imza bilgileri doğrulanarak, uygulamanın yalnızca güvenilir ve imzalanmış APK'lar tarafından çağrılması sağlanır. Bu sayede, uygulamanın değiştirilmiş veya sahte APK'lar tarafından çalıştırılması engellenir.

Uygulama Örnekleri:

1. APK dosyasının hash değerini çalışma sırasında doğrulama.
2. APK'nın imza bilgisini kontrol ederek yalnızca orijinal imzalanmış APK'ların çalışmasına izin verme.
3. Hash ve imza değerlerinin saklanması ve dinamik doğrulama işlemleri.

Teorik Açıklama: Root yetkisine sahip cihazlar, güvenlik riskleri oluşturabilir. Rooted cihazların tespit edilmesi, bu cihazlarda uygulamanın çalışmasının engellenmesini sağlar.

Root Tespit Yöntemleri:

1. **/dev/kmem Dosyası:** Sistemde bu dosyanın varlığı kontrol edilir. Varsa, sistemde syscall table hook ediliyor olabilir ve cihaz root yetkisine sahip olabilir.
2. **/proc/kallsyms Dosyası:** sys_call_table ve compat_sys_call_table adreslerinin boş olup olmadığını kontrol etme.
3. **/default.prop ve /system/build.prop Dosyaları:** Bu dosyalar okunabiliyorsa cihaz rootlanmış olabilir.
4. **Diğer Root Tespit Yöntemleri:**
    - Superuser.apk dosyasının varlığı.
    - 27047 portuna bağlanma testi ile frida server'ın aranması.

Uygulama Örnekleri:

1. Belirtilen dosyaların varlığını kontrol ederek root tespiti yapma.
2. Frida gibi araçların varlığını test etme ve tespit etme.

4. İleri Seviye LD Preload Saldırı Tespiti (Advanced LD Preload Attack Detection)

Teorik Açıklama: LD_PRELOAD, dinamik olarak yüklenen kütüphaneleri manipüle etmek için kullanılan bir yöntemdir. Bu teknik, kötü amaçlı yazılımlar tarafından kullanılan bir saldırı vektörüdür. LD_PRELOAD saldırılarının tespit edilmesi, uygulamanın güvenliğini artırır.

Uygulama Örnekleri:

1. Çalışma zamanında LD_PRELOAD ortam değişkenlerinin kontrol edilmesi.
2. LD_PRELOAD saldırılarının tespiti için özel algoritmaların kullanılması.
3. Tespit edilen saldırılara karşı uygulamanın kendini korumaya alması.

5. GDB, Tracers ve Emülatör Tespiti (GDB, Tracers, and Emulator Detection)

Teorik Açıklama: GDB gibi hata ayıklama araçlarının, izleyici (tracer) ve emülatörlerin tespit edilmesi, saldırganların uygulamayı analiz etmelerini ve değiştirmelerini engeller.

Uygulama Örnekleri:

1. GDB ortamının tespit edilmesi ve uygulamanın bu ortamda çalışmamasını sağlama.
2. ltrace, strace gibi izleyicilerin kullanımını algılama ve engelleme.
3. Emülatör ortamında çalışırken uygulamanın kapanmasını veya farklı bir davranış sergilemesini sağlama.

6. Debugger Eklentisi Tespiti (Debugger Attachment Check)

Teorik Açıklama: Uygulamanın bir hata ayıklayıcıya (debugger) eklenip eklenmediği tespit edilerek, kötü niyetli kişilerin uygulamayı analiz etmesi engellenebilir.

Uygulama Örnekleri:

1. Debugger eklentisini algılayan kod parçalarının uygulamaya eklenmesi.
2. Debugger tespit edildiğinde uygulamanın çalışmasını durdurma veya farklı bir işlev sergilemesini sağlama.
3. Anti-debugging teknikleri ile uygulamanın güvenliğini artırma.

7. Bellek Koruması (Memory Protection)

Teorik Açıklama: Bellek koruma teknikleri, bellek erişimlerinin kontrol edilmesini sağlar. Bellek üzerinde yapılan manipülasyonlara karşı koruma sağlar. Clang'ın SafeStack özelliği, bellek erişimlerini izlenebilir hale getirir.

Uygulama Örnekleri:

1. SafeStack kullanarak bellek koruma işlemlerinin devreye sokulması.
2. Bellek üzerinde yapılan her türlü manipülasyonun tespit edilmesi.
3. Bellek koruma mekanizmaları ile uygulamanın güvenliğini artırma.

8. Diğer RASP Teknikleri

1. **LD Preload Custom Envoriment Detection:** Özelleştirilmiş LD_PRELOAD ortam değişkenlerinin tespiti.
2. **Tamper Device Detection:** Uygulama cihazının değiştirilip değiştirilmediğinin kontrol edilmesi.
3. **Control Flow Counter Checking:** Kontrol akışını izleyen sayaçlar ile kodun manipüle edilip edilmediğinin tespiti.
4. **Device Binding:** Uygulamanın belirli bir cihaza bağlı olarak çalışmasını sağlama.
5. **Version Binding:** Uygulamanın belirli bir versiyonda çalıştığından emin olma.

$5. Hafta - Sonu$