ağ güvenliği- hijacking savunma

Data ile kontrol bilgisi her zaman ayrı kanallardan sağlanmalıdır!

Bu güvenlikteki en temel tasarım ilkesidir.

Örnek: Eskiden(1971) telefonla konuşurken belli bir frekansla sinyal alarak görüşmenin ücretinin ödenip ödenmediği bilgisi santrale iletiliyordu. Bu frekansı öğrendiklerinde insanlar bu frekansı taklit ederek ücret ödemeden görüşmeleri gerçekleştirmeye başlamışlar. Burada bu temel prensipin uygulanmadığını görüyoruz

Buffer overflow da bu ilkenin uygulanmaması nedeniyle imkan bulan bir durumdur. return adres'i bir kontrol bilgisi olarak düşündüğümüzde data override ederek kontrol bilgisini değiştirir. İkisi ayrı kanallarda olsa bu mümkün olmazdı.

Hijacking'in Engellenmesi

1- Bugların düzeltilmesi

1.a - Yazılımın incelenip düzeltilmesi → Bunun için toollar bulunuyor.

1.b - Yazılımın type safe bir dilde tekrar yazılması → Java, Go, Rust

2- Platformun savunulması

Saldırı sırasında aktarılan kodun çalışmasının engellenmesi

3- Koda overflow durumlarını tespit edecek/engelleyecek kod parçalarının eklenmesi

3.a- Halt process: Düzgünce gerçekleştirilemeyen control hijacking saldırısı DoS saldırısına dönüşecektir. Bu maliyetli seçenek.

3.b- StackGuard, CFI, LibSafe...

Platform Defenses

1-) Memory'i non-exec yapmak

Normalde heap ve stackin kodla işi yoktur ancak overflow yapıldığında buralar exec için kullanılabilir. Öyleyse buraları non-exec yapalım ki run edilemesin. Non-exec komutları: NX→ AMD, XD→Intel, XN→ARM.

Bu önlem tek başına yeterli değil, ama olması önemli.

Saldırı: Return Oriented Programming

Kod enjekte etmeden, override ederek bir return adresin değerini değiştirebiliriz. Bu durumda stackte exec yapmamış olsak da return adrese exec'i işaret ettirerek çalıştırmış oluruz. Bu da anlattığımız savunmayı geçersiz kılar.

Bir başka fonksiyonu "call" ile çağırmadığımız için, stack frame pointer değişmediği için, yalnız return address ile çağırdığımız için o fonksiyonun sonundaki return dönerken aynı yere getiriyor. Normalde fonksiyonu çağırmadan önce push yapılıyor, "call" ile çağırılınca stacke return address i basıyor.

2-) Address Space Layout Randomization

C run time library'si içinde fonksiyonların hangi adreste bulunacağı belirli olduğundan saldırgan bahsettiğimiz saldırıyı kolaylıkla gerçekleştirebiliyor. Bunu önlemek için bu adresleri randomize ediyoruz.

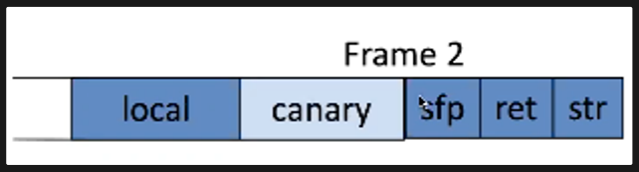
İlk olarak Win7'de kullanılıyor. Her sistem açıldığında kütüphanenin bulunduğu adresi güncelliyorlar ama yeterli olmuyor çünkü sadece 256 farklı adresleme yapabiliyorlar randomizasyon ile.

3-) kBouncer (Kernel Bouncer)

Return döndüğünde call ile çağırılmış olması lazım. Ama önceki bahsettiğimiz saldırıda bu durum söz konusu değil. Bunu tespit edip kernel'a instructionların girmesini engelleyecek bir "bodyguard" gibi düşünebiliriz kBouncer'ı. Kernel'a girmeden önceki son 16 return çağrısının normal olduğunu doğruluyor. Çok daha zorlaştırdık ancak tamamen engelleyemedik.

4-) StackGuard

Biliyoruz ki buffer overflow olduğunda stackin yapısı bozuluyor. Kanarya: maden ocaklarında kanarya bulunduruyorlar ki zararlı gaz salını gerçekleşmeye başladığında insanlar ölmeden kanaryanın ölümüyle bu durum anlaşılsın.



Canary değiştiyse stack overflow yapılmıştır.

Random canary/Terminator canary

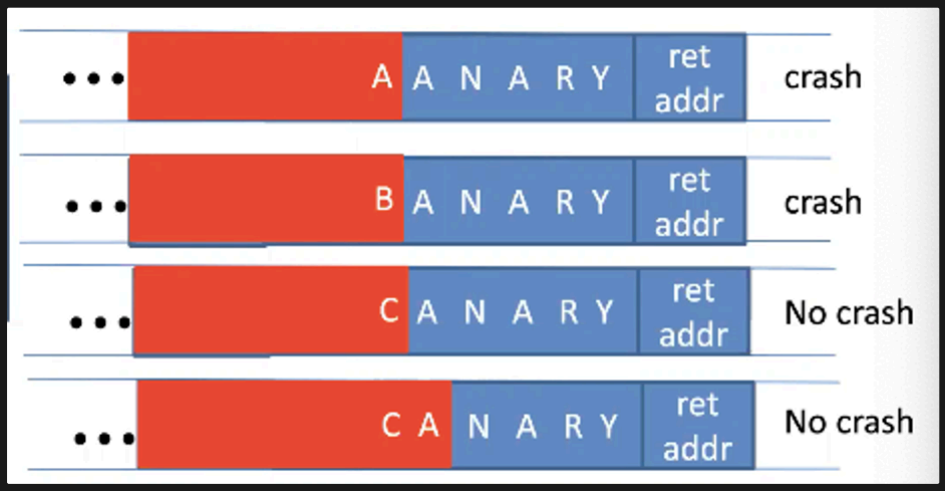
Canary kontrolü fonksiyondan geri dönerken gerçekleşir, öyleyse fonksiyon çalışırken tetiklenme olursa stackguard bunu çözemez.

5-) SAFESEH (Safe Struction Exception Handling) & SEHOP

Nelerin exception üretmesi gerektiğini biliyoruz, öyleyse exception üretmesini beklediğimiz instructionları güvenli listesine alırız, güvenli listesinde olmayan exceptionlar görürsen programı sonlandırırız.

- /SAFESEH: linker flag
  - Linker produces a binary with a table of safe exception handlers
  - System will not jump to exception handler not on list
- /SEHOP: platform defense (since win vista SP1)
  - Observation: SEH attacks typically corrupt the "next" entry in SEH list.
  - SEHOP: add a dummy record at top of SEH list
  - When exception occurs, dispatcher walks up list and verifies dummy record is there. If not, terminates process.

Kötü bir senaryo:



Servis çöktüğünde otomatik olarak restart yapıyoruz. Servis kendini yükler. Bu şekilde byte byte canary'nin değerini öğrenebiliriz.

6-) Libsafe

Son yüklenen kütüphane güvenlidir.

7-) Shadow Stack

Stack'in bir kopyasını memory'de tutarız(saldırganın buraya erişemediğini varsayıyoruz) ve buradan değişim olup olmadığını kontrol ederiz.

8-) CFI(Control Flow Integrity)

Bir call'un akış grafını ortaya çıkartıyoruz. Compilation sırasında nerelere gidebileceğini biliyoruz, farklı bir yere giderse program sonlandırılıyor. kBouncer bu çalışmaların bir ürünü.

9-) Control Flow Guard

Dolaylı fonksiyon çağrılarında oraya gitmeye izin olup olmadığı kontrol ediliyor.