

# 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 olarak 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/engellenecek 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 deęerini deęiřtirebiliriz. Bu durumda stackte exec yapmamıř olsak da return adrese exec'i iřaret ettirerek alıřtırmıř oluruz. Bu da anlattıęımız savunmayı geersiz kılar.

Bir bařka fonksiyonu "call" ile aęırmadıęımız iin, stack frame pointer deęiřmedięi iin, yalnız return address ile aęırdıęımız iin o fonksiyonun sonundaki return donerken aynı yere getiriyor. Normalde fonksiyonu aęırmadan nce push yapılıyor, "call" ile aęırılınca stacke return address i basıyor.

## 2-) Address Space Layout Randomization

C run time library'si iinde fonksiyonların hangi adreste bulunacaęı belirli olduęundan saldırgan bahsettięimiz saldırıyı kolaylıkla gerekleřtirebiliyor. Bunu nlemek iin bu adresleri randomize ediyoruz.

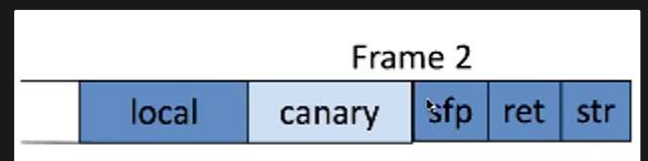
İlk olarak Win7'de kullanılıyor. Her sistem aıldığında ktphanenin bulunduęu adresi gncelliyorlar ama yeterli olmuyor nk sadece 256 farklı adresleme yapabiliyorlar randomizasyon ile.

## 3-) kBouncer (Kernel Bouncer)

Return dndęnde call ile aęırılmış olması lazım. Ama nceki bahsettięimiz saldırıda bu durum sz konusu deęil. Bunu tespit edip kernel'a instructionların girmesini engelleyecek bir "bodyguard" gibi dřnebiliriz kBouncer'ı. Kernel'a girmeden nceki son 16 return aęrısının normal olduęunu doęruluyor. ok daha zorlařtırdık ancak tamamen engelleyemedik.

## 4-) StackGuard

Biliyoruz ki buffer overflow olduęunda stackin yapısı bozuluyor. Kanarya: maden ocaklarında kanarya bulunduruyorlar ki zararlı gaz salını gerekleřmeye bařladıęında insanlar lmeden kanaryanın lmyle bu durum anlařılsın.



Canary deęiřtiyse stack overflow yapılmıřtır.

Random canary/Terminator canary

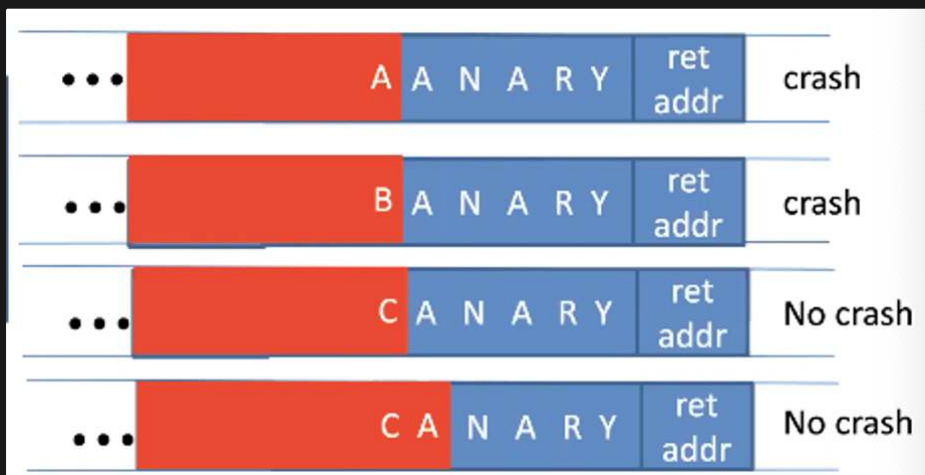
Canary kontrol fonksiyondan geri donerken gerekleřir, yleyse fonksiyon alıřırken tetiklenme olursa stackguard bunu özemez.

## 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.