ağ güvenliği- hijcaking attacks

Temel kontrol hırsızlığı(Control hijacking) saldırıları. Bu tarz saldırılar ilk günden beri benzer şekillerde yapılıyor. Çalışmakta olan sistemleri kendi kontrollerinde çalıştırmayı hedefliyorlar.

Kolaydan zora, yaygından nadire, eskiden yeniye:

1-) Buffer overflow

2-) Integer overflow

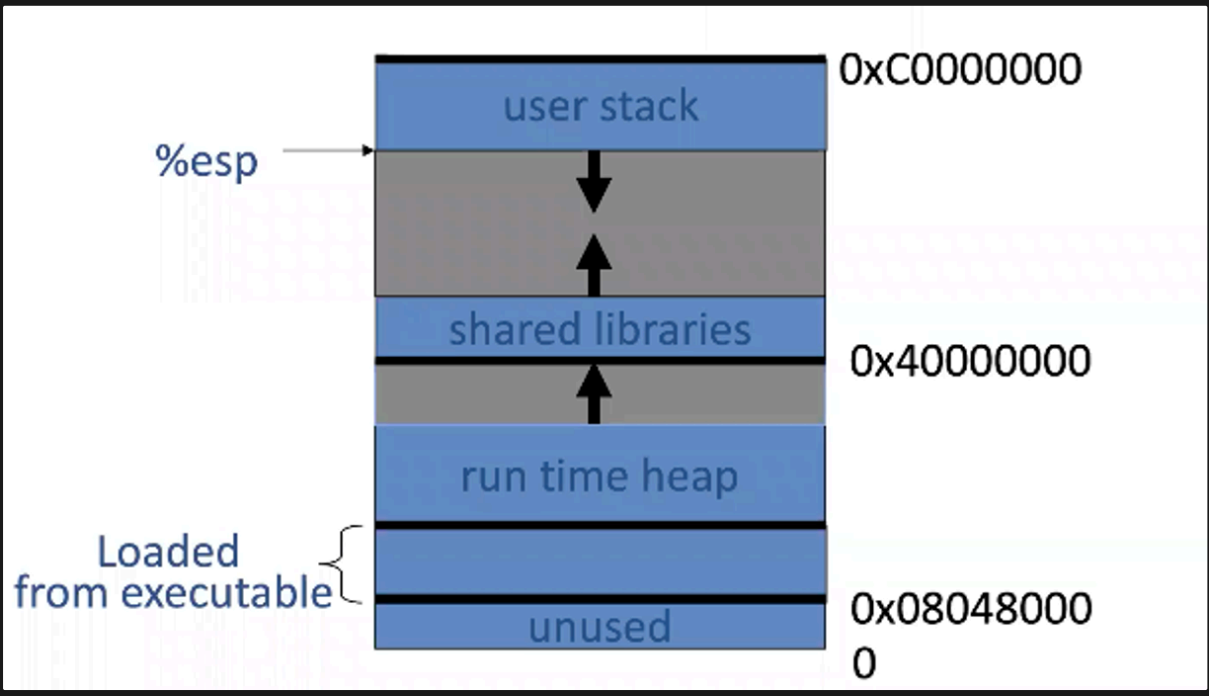
3-) Format String

4-) Use after free

Buffer Overflow(Stack Smashing) Saldırıları

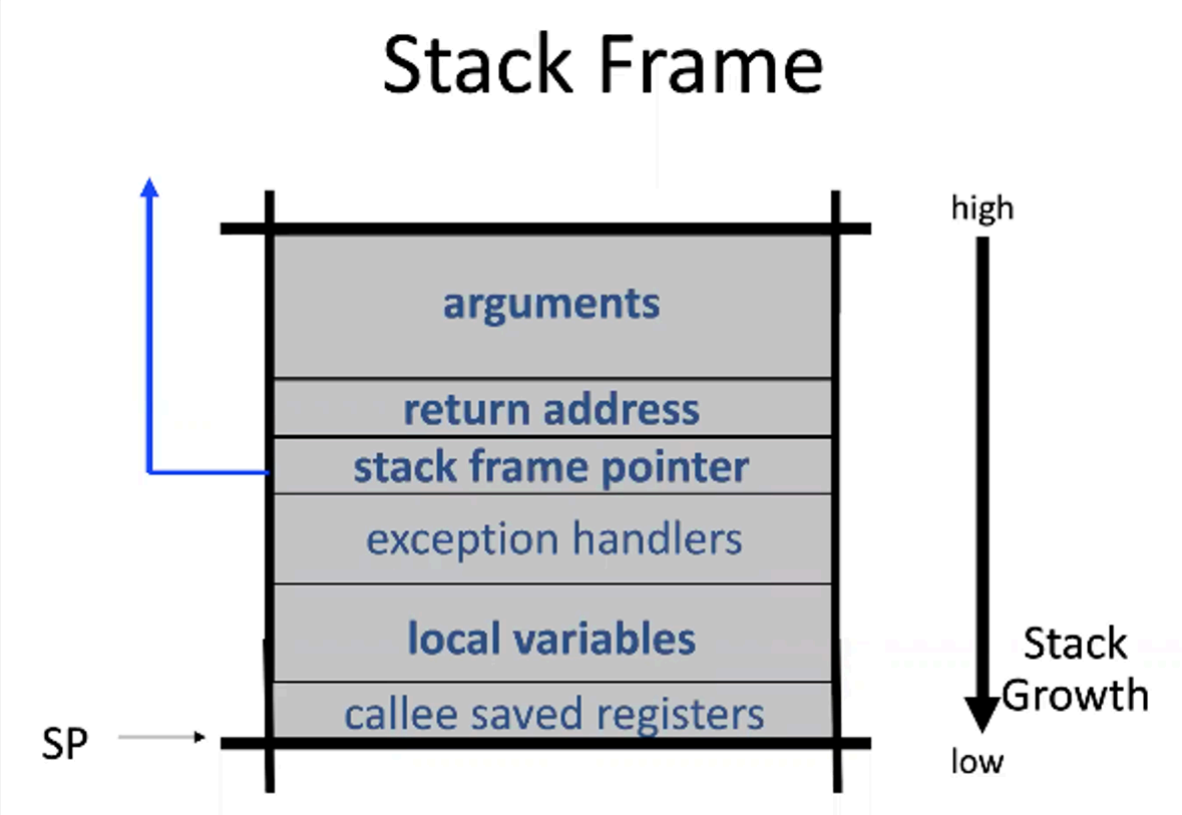
C/C++'da çok yaygın.

İhtiyaç duyduklarımız: C fonksiyonları, stack, heap, sistem çağrıları, exec()'i anlamak. Ayrıca hedefteki makinenin CPU ve OS'unun ne olduğunu bilmemiz gerekiyor.



Linux

Stack kullanıldıkça adresi azalır. Stack çok kullanılır ve shared lib veya run time heap üzerine gelirse stack overflow oluşur. Shared libraries programlar tarafından ihtiyaç duyulan kütüphanelerdir. Tüm temel işleri gerçekleştirir, toplama, çıkarma, input, output gibi. Run time heap dinamik olarak kullanılan bellek bölgesidir(malloc ve free fonk.ları gibi). Unused kısmı işlemciye ait orada o takılır. Bir program 0x080480000 adresinden başlar.

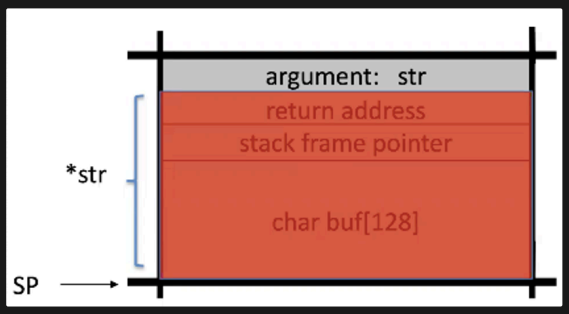


Her fonksiyon çağrılmadan önce stackte bu fonksiyon için oluşturulan veri yapısına Stack Frame denir.

Biz örnek olarak 32 bitlik yapıları kullanıyoruz. Return adres 32bittir o zaman. SFP: Fonksiyonu çağıran fonksiyonun stackteki adresidir. Bunu uygulama çökecek olursa kurtarma yapmada kullanırız. Exception handler: özel durum gerçekleştiğinde hangi adresten itibaren çalışılacağını gösterir. Local variables: sadece fonksiyondayken kullanılır, dışardan erişilemez. Callee saved registers: Fonksiyona gelindiğinde registerların değerleri burada saklanır, fonksiyonun çalışması bittiğinde bu değerlere set ederek registerların değerlerini korumuş oluruz.

```
void func(char *str) {
    char buf[128];
    strcpy(buf, str);
    do-something(buf);
}
```

gelen str 127 byte olması gerekiyor maks(çünkü stringde son char her zaman 0 olur)

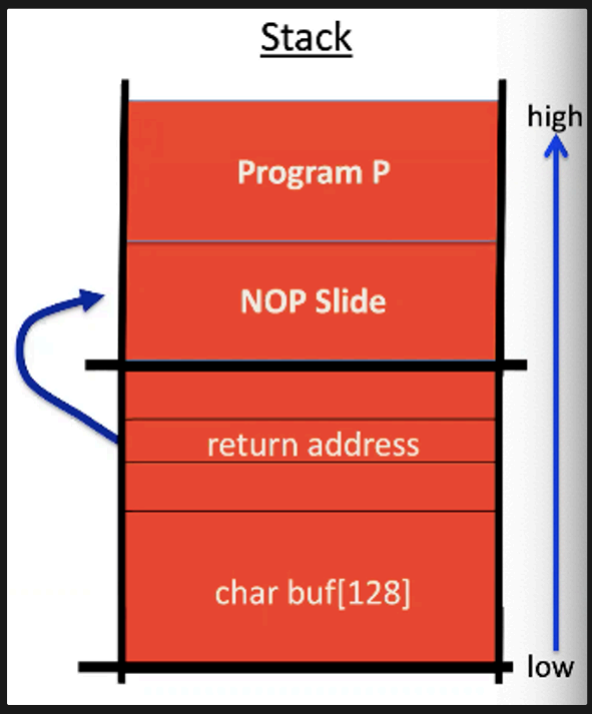


136 bytelık bir str geldiğinde buff için ayrılan alanın üzerine çıkarak erişilmemesi gereken yerleri değiştirir.

Sistem uygulamalarının büyük bir çoğunluğu root olarak çalıştırıldığından saldırganlar için çok caziptirler.

Çözümler

The NOP Slide



nop, xor eax eax, inc ax → bunlar nop için kullanılarabilir, birer bytelar

Güvenli olmayan fonksiyonlar(C): strcpy, strcat, gets, scanf, strncpy, strncat

Güvenli versiyonu: strcpy_s

Nerelerde buffer overflow gerçekleştirilebilir?

Exception handlers

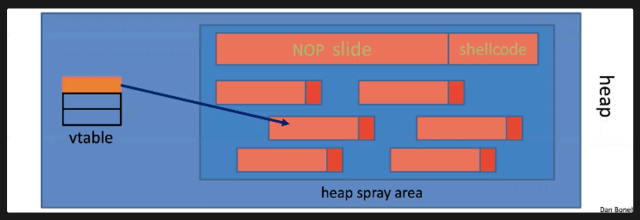
Function pointers

Longjmp buffers

Heap istismarları

Heap istismarlarını engellemek için kullanılan yöntemler:

Heap Spraying: bunu basitçe heap için nop slide uygulaması olarak düşünebiliriz.



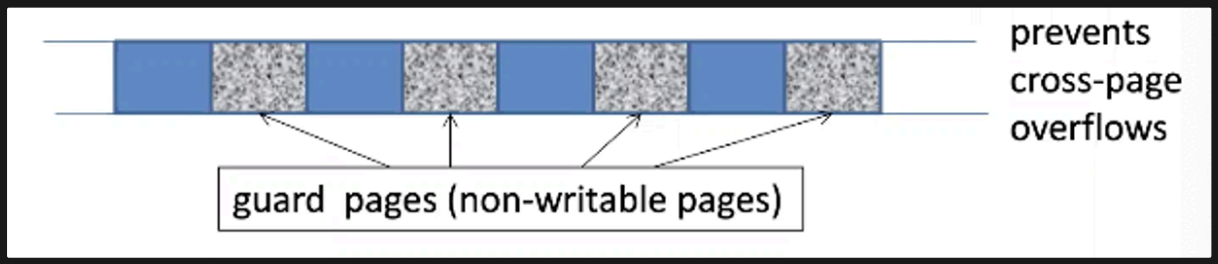
Heap spraying

```
var  nop = unescape("%u9090%u9090")
while (nop.length < 0x100000)  nop += nop;

var shellcode = unescape("%u4343%u4343%...");

var x = new Array ()
for (i=0;  i<1000;  i++) {
       x[i] = nop + shellcode;
}
```

Heap spraying kodu

Ad-hoc (geçici) Heap Overflow Azaltımları:

Bu şuan daha popüler bir yöntem. Araya yazılamayan sayfalar ekliyoruz.



Buffer overflowları nasıl tespit edebiliriz?

Fuzzing:

Farklı uzunluklarda istekler yollayıp sonuna "$$$$$" işaretlerini koyuyoruz. Program çökerse core dump'da "$$$$$" işaretlerini arıyoruz ve overflow olan adresi buluyoruz.

Double Free

Daha önce free edilen bir alan tekrar free edildiğinde memory manager'ın hiçbir şey yapmadan dönmesi beklenirken bu alana bir şeyler yazdığı tespit edilmiş(o dönemde).

Integer Overflows

İşlemler sonucunda çıkan değer registera sığmayınca integer overflow olur. Çok büyük bir işlem sonucu 1 0000 0000 gibi bir overflowla taşınca biz sonucu sıfır gibi görüp yanılabiliriz.

Format String Problemi

%d gibi tipleri yalnız kullanmaktan dolayı olabilir.

Zaafiyeti içeren fonksiyonlar: printf, fprintf, sprintf, vprintf, vfprintf, vsprintfi, syslog, err, warn.

Use After Free

malloc'la allocate ediyoruz, free ile serbest bırakıyoruz ama sonra free yapmamışız gibi kullanmaya devam ediyoruz.