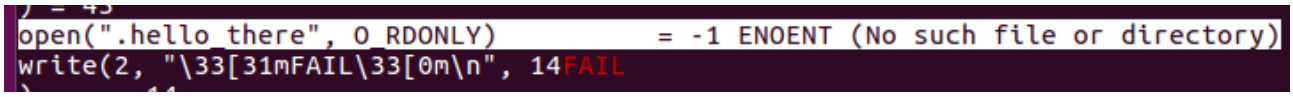
Riddle

Διαμάντη Ιωάννα 03115035
Ντούνης Πέτρος 03115091

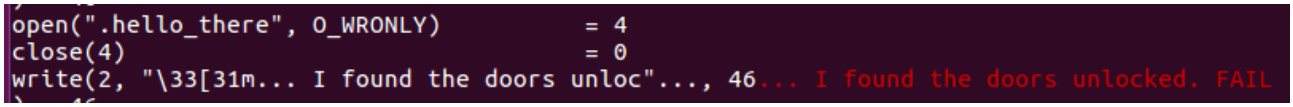
Challenge 0:

Τρέξαμε αρχικά την εντολή strace ./riddle και παρατηρήσαμε ότι το πρόγραμμα κολλάει στην εντολη open(".hello_there", O_RDONLY) = -1 ENOENT (No such file or directory) οπότε δημιουργήσαμε ένα κρυφό αρχείο μέσω της εντολής:
touch .hello_there



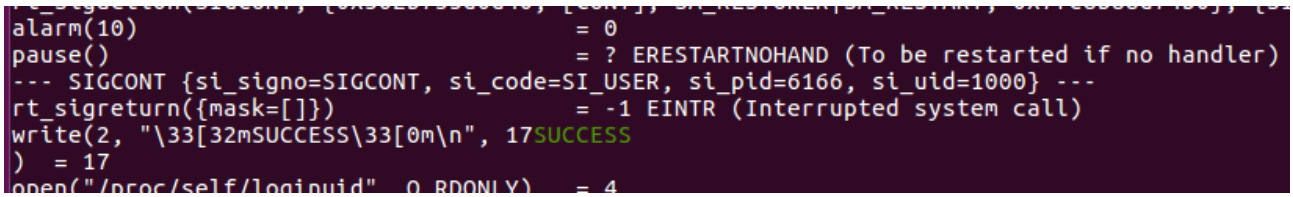
Challenge 1:

Πάλι τρέχοντας strace ./riddle παρατηρούμε ότι εδώ κολλάει το riddle επειδή μπορει να ανοίξει ελεύθερα το .hello_there και να γράψει, οπότε του στερούμε αυτη την δυνατότητα μέσω της εντολής:
sudo chmod a=r .hello_there που σημαίνει ότι το αρχείο μας είναι πλέον διαθέσιμο μόνο για γράψιμο.



Challenge 2:

Εδώ το πρόγραμμα κολλαει στην pause(10) οπότε για να συνεχίστει βρισκουμε το pid του riddle μεσω της εντολής ps -ax | grep riddle και στη συνέχεια εκτελούμε kill SIGCONT pid.



Challenge 3:

Εδώ τρέχοντας ltrace ./riddle παρατηρούμε ότι δεν τρέχει επειδή η μεταβλητή περιβάλλοντος ANSWER δεν υπάρχει. Η ερώτηση είναι what

is the answer to life the universe and everything? Που ξέρουμε ,μέχρι και το google calculator απαντάει, ότι είναι 42. Άρα τη θέτουμε μέσω της εντολής export ANSWER=42, όποτε περνάμε και αυτή τη δοκιμασία.

```
)                                       = 25
getenv("ANSWER")                                                        = "42"
strtol(0x7ffc8ffd5d74, 0, 10, 11)                                       = 42
fprintf(0x7fae5d53c540, "\033[32m%s\033[0m\n", "SUCCESS"SUCCESS
)                                       = 17
```

Challenge 4:

Πάλι σε αυτό το ερώτημα βλέπουμε μέσω ltrace ότι κολλάει το πρόγραμμα λόγω της μη ύπαρξης του magic_mirror. Αν το φτιάξουμε ως απλό αρχείο παρατηρούμε το εξής error:

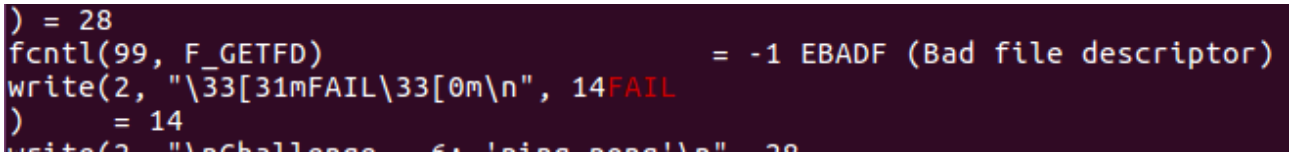
```
open("magic_mirror", 2, 010000400000)                                   = 4
rand(0x56235d0f2f40, 2, 0xfffff80040020000, 0x7fa4dc84c040)             = 0x2adfa67
write(4, "B", 1)                                                        = 1
read(4, "", 1)                                                          = 0
fprintf(0x7fa4dcb1a540, "\033[31m%s\033[0m\n", "I cannot see my reflection. FAIL"...I cannot see my reflection. FAIL
)   = 42
```

Οπότε φτιάξαμε ένα named pipe με όνομα magic_mirror μέσω της εντολής mkfifo magic_mirror, όπου και ξεκλειδώσαμε την επόμενη δοκιμασία.

```
open("magic_mirror", 2, 010000400000)                                   = 4
rand(0x55dc99977f40, 2, 0xfffff80040020000, 0x7f11e9759040)             = 0x9feaf82
write(4, "G", 1)                                                        = 1
read(4, "G", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x1316d556
write(4, "I", 1)                                                        = 1
read(4, "I", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x3f141e26
write(4, "C", 1)                                                        = 1
read(4, "C", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x2d00f3ca
write(4, "Q", 1)                                                        = 1
read(4, "Q", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x6431948c
write(4, "Q", 1)                                                        = 1
read(4, "Q", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x1946cdd9
write(4, "J", 1)                                                        = 1
read(4, "J", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x51695b42
write(4, "U", 1)                                                        = 1
read(4, "U", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0xa44a40a
write(4, "K", 1)                                                        = 1
read(4, "K", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x25cc157
write(4, "J", 1)                                                        = 1
read(4, "J", 1)                                                         = 1
rand(4, 0x7ffdd150dcbf, 1, 0x7f11e9759260)                              = 0x193b0274
write(4, "Y", 1)                                                        = 1
read(4, "Y", 1)                                                         = 1
close(4)                                                                = 0
fprintf(0x7f11e9a27540, "\033[32m%s\033[0m\n", "SUCCESS"SUCCESS
)                                       = 17
```

Challenge 5:

Σε αυτήν την δοκιμασία παρατηρούμε ότι το riddle προσπαθεί να κάνει κλήση fcntl σε αρχείο με fd = 99.



Επειδή τέτοιο αρχείο δεν υπάρχει δημιουργήσαμε τον εξής κώδικα:

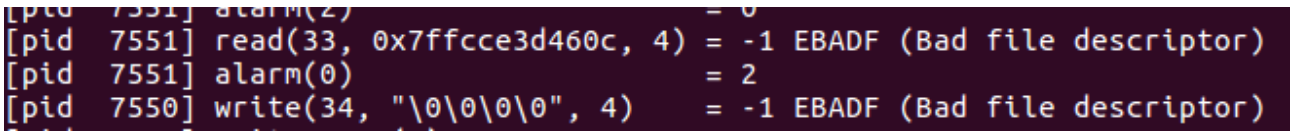
```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include<sys/types.h>
int main(int argc, char** argv){
        int filedesc = open("temp.txt", O_WRONLY | O_APPEND);
        dup2(filedesc,99);
        extern char** environ;
        execve(argv[1],argv,environ);

        return 0;
}
```

Όπου τρέχοντας τον παραπάνω κώδικα με όρισμα το riddle αφού έχουμε δημιουργήσει ένα αρχείο με όνομα temp.txt, αρχικά αλλάζει το fd του σε 99 και στην συνέχει τρέχει το riddle μέσω της εντολής execve, οπότε όταν τρέχει το riddle να υπάρχει ένα αρχείο με fd = 99, συγκεκριμένα το temp.txt.

Challenge 6:

Εδώ βλέπουμε ότι γίνεται fork() και έχουμε πλέον δυο διεργασίες και τρέχοντας strace -f ./riddle παρατηρούμε το εξής:



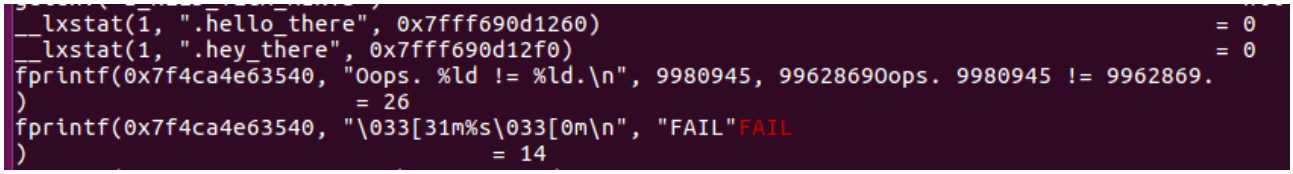
Οπότε κατασκευάζουμε ένα pipe και θέτουμε τα fd των άκρων ίσα με 33 και 34. Τότε παρατηρούμε ότι χρειάζεται ακόμη ένα pipe με fd άκρων 53 και 54. Οπότε συνοψίζοντας τρέχοντας πρώτα το παρακάτω πρόγραμμα με όρισμα το riddle, περνάμε το ερώτημα. Ο κώδικας είναι ο εξής:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<fcntl.h>

int main(int argc, char** argv){
        int pfd[2],pfd2[2];
        pipe(pfd);
        pipe(pfd2);
        dup2(pfd2[0],53);
        dup2(pfd2[1],54);
        dup2(pfd[0],33);
        dup2(pfd[1],34);
        extern char** environ;
      execve(argv[1],argv,environ);
}
```

Challenge 7:

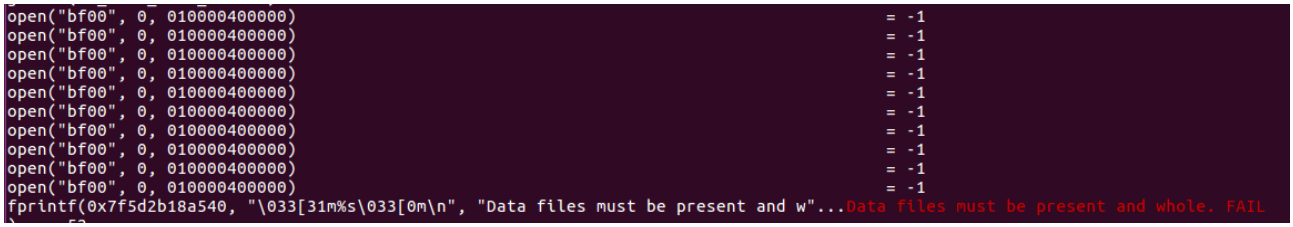
Τρέχοντας ltrace παρατηρούμε το εξής:



Οπότε απλά δημιουργούμε ένα hard-link του αρχείου .hello there με όνομα .hey_there μέσω της εντολής: ln .hello_there .hey_there και αρκεί για να περάσουμε.

Challenge 8:

Σε αυτό το στάδιο τρέχοντας ltrace ./riddle παρατηρώ το εξής:



Οπότε δημιουργήσαμε ένα αρχείο με όνομα bf00 που να έχει γραμμένο χαρακτήρα κάθε 1073741824 θέσεις 10 φορές, όμως τότε ζητήθηκε ένα αρχείο με όνομα bf01 και όταν το φτιάξαμε και ένα με όνομα bf02 κτλ

οπότε φτιάξαμε το παρακάτω πρόγραμμα κατασκευής αρχείων bf00-bf99 και ανάλογου γεμίσματος όπου μετά περάστηκε η δοκιμασία:

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

int main(){
	FILE *fp;
	int i,k;
	for(k=0; k<99; k++){
	char bf[] = "bf";
	char num[]="init";
	sprintf(num,"%d",k);
	if(k<10) strcat(bf,"0");
	strcat(bf,num);
	fp = fopen(bf, "w");
	for(i=0 ; i<10 ; i++){
		fseek(fp, 1073741824, SEEK_CUR);
		fprintf(fp,"X");

	}
	fclose(fp);
}
return 0;
}
```

Challenge 9:

Σε αυτήν την κατάσταση τρέχουμε strace και παρατηρούμε ότι το λάθος είναι εδώ:

```
socket(2, 1, 0)                                                                 = 4
inet_aton("127.0.0.1", { 0x100007f })                                           = 1
fputs("I am trying to contact you...\\n", 0x7f7c1f329540I am trying to contact you...\n)                    = 1
connect(4, 0x7ffcf4e86fa0, 16, 0x7f7c1f05b2c0)                                  = -1
fputs("...but you don't seem to listen.", 0x7f7c1f329540...but you don't seem to listen.\n)                 = 1
fprintf(0x7f7c1f329540, "\033[31m%s\033[0m\n", "FAIL"FAIL
```

Προσπαθεί δηλαδή το riddle να συνδεθεί με ένα socket, χωρίς να υπάρχει ο αντίστοιχος server στο pc μας οπότε τον δημιουργούμε με τον ακόλουθο κώδικα:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
```

```
#include <string.h>

int main( int argc, char *argv[] ) {
   int sockfd, newsockfd, portno, clilen;
   char buffer[256];
   struct sockaddr_in serv_addr, cli_addr;
   int  n;
   char sub[1000];
   /* First call to socket() function */
   sockfd = socket(AF_INET, SOCK_STREAM, 0);

   if (sockfd < 0) {
      perror("ERROR opening socket");
      exit(1);
   }

   /* Initialize socket structure */
   bzero((char *) &serv_addr, sizeof(serv_addr));
   portno = 5001;

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   serv_addr.sin_port = htons(49842);

   /* Now bind the host address using bind() call.*/
   if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
      perror("ERROR on binding");
      exit(1);
   }

   /* Now start listening for the clients, here process will
      * go in sleep mode and will wait for the incoming connection
   */

   listen(sockfd,5);
   clilen = sizeof(cli_addr);

   /* Accept actual connection from the client */
   newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);

   if (newsockfd < 0) {
      perror("ERROR on accept");
      exit(1);
   }

   /* If connection is established then start communicating */
   bzero(buffer,256);
   n = read( newsockfd,buffer,255 );

   if (n < 0) {
      perror("ERROR reading from socket");
      exit(1);
```

```
  }
        for(int i=12 ; i<17 ; i++) sub[i-12]=buffer[i];
        sub[6]='\0';
        int l = atoi(sub);
        l++;
        char res[100];
        sprintf(res, "%d", l);

  printf("Here is the message: %s\n",buffer);

  /* Write a response to the client */
  n = write(newsockfd,res,5);

  if (n < 0) {
    perror("ERROR writing to socket");
    exit(1);
  }

  return 0;
}
```

Μετά από αυτή την εγκατάσταση του server επιτυγχάνουμε σε αυτό το challenge.

Challenge 10:

Μέσω ltrace βλέπουμε το εξής:

```
open("secret_number", 578, 0600)                                    = 5
unlink("secret_number")                                             = 0
sysconf(30, 578, 384, 0x7f87198f6a97)                               = 4096
malloc(4096)                                                        = 0x55f5bb259240
write(5, "The number I am thinking of righ"..., 4096)               = 4096
mmap(0, 4096, 3, 1)                                                 = 0x7f8719fef000
close(5)                                                            = 0
```

Δημιουργείται εέα αρχείο με όνομα secret_number, όπου είναι γραμμένος ο ζητούμενος αριθμός και στη συνέχεια γίνεται unlink. Επειδή η unlink() δεν διαγράφει hardlinks, μπορούμε να δημιουργήσουμε εκ των προτέρων το αρχείο secret_number και ένα hard link του, πχ μέσω
ln secret_number peek. Τρέχουμε το riddle και όταν μας ζητάει τον αριθμό βλέπουμε στο αρχείο peek αυτό:

```
The number I am thinking of right now is [uppercase matters]: 688F569B
```

οπότε περνάμε στο 2ο ESP που έχει το ίδιο ζητούμενο χωρίς να επιτρέπεται η χρήση hard-link.

Challenge 11:

Πάλι από το manpage της unlink() βλέπουμε ότι η unlink δεν διαγράφει ανοιχτά αρχεία, οπότε χρησιμοποιούμε τον εξής κώδικα:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
	FILE *fp;
	while(1){
		fp=fopen("secret_number","r");
		if(fp!=NULL) break;
	}
	sleep(1);
	char c='a';
	for(int i=0; i<3000; i++){
		c=fgetc(fp);
		printf("%c",c);
	}
	return  0;
}
```

Τρέχουμε τον παραπάνω κώδικα ο οποίος περιμένει να δημιουργηθεί το αρχείο secret_number και όταν το βρει μετά από 1 sec το διαβάζει. Οπότε τρέχουμε το παραπάνω και στη συνέχει το riddle, οπότε το παραπάνω μας τυπώνει το περιεχόμενο του secret_number του οποίου εφόσον ήταν ανοιχτό η unlink δεν μπορούσε να το διαγράψει.

Challenge 12:

Δοκιμάζοντας την ltrace παρατηρούμε τα εξής:

```
mkstemp(0x7ffcf6df9620, 0x7ffcf6df6ff0, 0xffffffffffff45142, 15)                    = 5
ftruncate(5, 4096)                                                                  = 0
mmap(0, 4096, 3, 1)                                                                 = 0x7f529cd1e000
memset(0x7f529cd1e000, '\0', 4096)                                                  = 0x7f529cd1e000
rand(0x7f529cd1efa0, 0x7f529cd1f000, 4096, 0xfffffe0)                               = 0xb5b8cb1
fprintf(0x7f529c8f3540, "%s '%c' at %p\n", "I want to find the char", 'F', 0x7f529cd1e06fI want to find the char 'F' at 0x7f529cd1e06f
) = 46
```

Το riddle δημιουργεί ένα αρχείο της μορφής riddle-xxxxxx κάτω από τον φάκελο /tmp και το αντιστοιχίζει μέσω mmap() σε μια περιοχή εικονικής μνήμης. Εκεί θέλει να αλλάξουμε τον χαρακτήρα 6F = 111 σε ‘F’. Για να το κάνουμε αυτό, αρχικά εκμεταλλευόμαστε τα sleep, δίνοντας του από το terminal σήμα SIGSTOP. Στη συνέχεια τρέχουμε τον παρακάτω κώδικα ο οποίος επιτελεί το ζητούμενο:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
```

```
int main(){
	char cont[1000000]="\0",c,file[13],cha;
	int i;
	cha='I';
	int fp=open("/tmp/riddle-s8lL8d",O_RDWR);
	int ft=open("/tmp/riddle-s8lL8d",O_RDWR);
	i=0;
	char a[112];
	int r=read(fp,a,sizeof(a));
	a[111]=cha;
	write(ft,a,sizeof(a));
	close(ft);
	close(fp);

	return 0;
}
```

Εφόσον έχουμε κάνει SIGSTOP αλλάζουμε τη μεταβλητή char στο γράμμα που θέλουμε να τυπώσουμε και στις open το όνομα του αρχειου κάτω από το /tmp.

Challenge 13:

Κάνοντας ltrace αρχικά βλέπουμε ότι πρέπει να ξαναδώσω write permission στο .hello_there, το οποίο κάνω με chmod a=rw .hello_there
Στη συνέχει παρατηρώ το εξής:

```
open(".hello_there", 66, 0600)                                  = 4
ftruncate(4, 32768)                                             = 0
mmap(0, 0x8000, 3, 1)                                           = 0x7f0073e20000
ftruncate(4, 16384)                                             = 0
read(0A
, "A", 1)                                                 = 1
--- SIGBUS (Bus error) ---
```

Το αρχείο κόβεται στο μισό του μέγεθος, αφού έχει γίνει πρώτα mmap, οπότε δημιουργείται Bus Error. Για να διορθωθεί, απλά καλούμε πάλι την frtruncate επιστρέφοντας το αρχείο στο αρχικό του μέγεθος. Αυτό επιτυγχάνεται με τον εξής κώδικα:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<signal.h>
#include<fcntl.h>
int main(){
int fd;
fd=open(".hello_there",O_RDWR);
ftruncate(fd,32768);
}
```

Challenge 14:

Σε αυτό το τελευταίο challenge μας ζητείται να τρέχεται το riddle με pid = 32767 που είναι και το μεγαλύτερο δυνατό pid. Για αυτό χρησιμοποιήθηκε ο παρακάτω κώδικας ο οποίος προήλθε από το internet με κάποιες κατάλληλες παραλλαγές:

```
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
int fd, pid;
char buf[32];
fd = open("/proc/sys/kernel/ns_last_pid", O_RDWR | O_CREAT, 0644);
flock(fd, LOCK_EX);
pid = atoi(argv[1]);
snprintf(buf, sizeof(buf), "%d", pid - 1);printf("Writing pid-1 to ns_last_pid...\n");
write(fd, buf, strlen(buf));
int new_pid;
new_pid = fork();
if (new_pid == 0) {
execve(argv[2],argv,NULL);
}
else  {
printf("I'm parent. My child got right pid!\n");
}
flock(fd, LOCK_UN);
close(fd);
return 0;
}
```