1η Εργαστηριακή «Αναφορά»

Πρόβλημα 0:
Προσπαθεί να ανοίξει για ανάγνωση ένα αρχείο .hello_there το οποίο δεν υπάρχει.
Λύση:
Δημιουργία αρχείου με το ίδιο όνομα.

Πρόβλημα 1:
Δεν πρέπει να επιτρέψουμε στο πρόγραμμα να γράψει στο αρχείο hello_there.
Λύση:
Αφαιρούμε δικαίωμα εγγραφής μέσω chmod -w.

Πρόβλημα 2:
Το πρόγραμμα "κολλάει" στην αναμονή.
Λύση:
Στέλνουμε σήμα kill -SIGCONT [pid] από δεύτερο terminal. Το pid το βρίσκουμε με ps aux | grep riddle.

Πρόβλημα 3:
Ελέγχει την τιμή μιας μεταβλητής περιβάλλοντος.
Λύση:
Μέσω getenv βρίσκουμε τις μεταβλητές περιβάλλοντος και με export μεταβάλλουμε την τιμή της στο 42.

Πρόβλημα 4:
Το πρόγραμμα θέλει να διαβάζει ότι γράφει με την ίδια σειρά.
Λύση:
Φτιάχνουμε ένα named FIFO με το κατάλληλο όνομα.

Πρόβλημα 5:
Το πρόγραμμα ζητάει συγκεκριμένο fd σε συγκεκριμένο αρχείο.
Λύση:
Τρέχουμε exec 99<>[file] για ανακατεύθυνση του fd.

Πρόβλημα 6:
Το πρόγραμμα προσπαθεί μέσω fork να περάσει μηνύματα από το ένα παιδί στο άλλο.
Λύση:
Ανακατευθύνουμε τους file descriptor για να δείχνουν στο ίδιο named fifo ώστε να μπορούν να διαβάζουν τα μηνύματα.

Πρόβλημα 7:
Το πρόγραμμα ψάχνει τις ιδιότητες ενός αρχείου ελέγχοντας τα link του.
Λύση:
Φτιάχνουμε hard link με ln [link] [file.txt].

Πρόβλημα 8:
Το πρόγραμμα τρέχει 10 lseek ελέγχοντας το μέγεθος και την πληρότητα των αρχείων.
Λύση:
Αξιοποιούμε το special file /dev/urandom και μέσω της εντολής dd φτιάχνουμε ένα αρχείο μεγέθους 1 GiB, το οποίο μετά αντιγράφουμε με τα κατάλληλα ονόματα.

```
dd if=/dev/urandom of=./bf00 count=1.000.000 bs=1024
```

Πρόβλημα 9:
Το πρόγραμμα προσπαθεί να επικοινωνήσει μέσω sockets
Λύση:
Ανοίγουμε socket μέσω nc σε listening mode και socat και περνάμε το απαραίτητο μήνυμα

Πρόβλημα 10:
Βρες τον αριθμό
Λύση:
Βλέπουμε πως ο αριθμός γράφεται σε ένα αρχείο το οποίο μετά γίνεται unlink και δεν είναι απευθείας προσβάσιμο. Οπότε φτιάχνουμε ένα link και διαβάζουμε τον αριθμό μέσω του link

Πρόβλημα 11:
Βρες τον αριθμό 2.0
Λύση:
Η διαφορά εδώ είναι ότι ελέγχοντας τις ιδιότητες του αρχείου απαγορεύεται να φτιάξουμε link. Οπότε, εφόσον από τα system calls φαίνεται ότι τα αρχείο έχει απεικονιστεί στην μνήμη, κάνουμε απεικόνιση του χώρου μνήμης του προγράμματος μέσω dd και έτσι βρίσκουμε τον κωδικό.

```
sudo dd id=/proc/[pid]/mem bs=4096 skip=$((0x[addr]/4096)) count=1 | hexdump -c
```

Πρόβλημα 12:
Το πρόγραμμα ζητάει να βρει μια αλλαγή σε μια συγκεκριμένη θέση μνήμης
Λύση:
Στην πραγματικότητα αυτό που κοιτάει το πρόγραμμα είναι ο χειρισμός του offset, καθώς η διεύθυνση μνήμης είναι τυχαία, ενώ το offset σταθερό. Οπότε χρησιμοποιούμε σε δεύτερο terminal το κάτωθι πρόγραμμα για να ανοίξουμε το τυχαίο αρχείο, μορφής riddle-xxxx, στο /tmp/ και να γράψουμε στο ζητούμενο offset (=111) τον ζητούμενο χαρακτήρα.

```
int main(int argc, char const *argv[]) {

        const char *filename = argv[1];

        int fd = open(filename, O_RDWR | O_CREAT, 0600);

        void *buf = mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

        ((char *)buf)[111] = argv[2][0];

return 0;
}
```

Πρόβλημα 13:
Το πρόγραμμα προσπαθεί να διαβάσει από θέσεις που είναι πλέον invalid
Λύση:
Αυτό που πραγματικά συμβαίνει είναι μέσω δύο διαδοχικών truncate το αρχείο γίνεται πρώτα 32KiB και ύστερα 16KiB, με το πρόγραμμα να προσπαθεί να διαβάσει στο δεύτερο κομμάτι, 16-32. Οπότε τρέχουμε ftruncate για να το ξανακάνουμε 32.

Πρόβλημα 14:
Το πρόγραμμα "απαιτεί" να τρέξει με συγκεκριμένο pid
Λύση:
Γράφουμε το κάτωθι πρόγραμμα το οποίο κάνει fork και διαβάζει το pid του καινούργιο παιδιού. Αν το pid != 32767 τότε ξανακάνει fork, διαφορετικά κάνει execv και τρέχει το riddle

```
int main() {
    while (1) {
        pid_t pid = fork();
        if (pid == 0) {
                if (getpid() == 32767) {
                char *args[] = {"./riddle", NULL};

                execv(args[0], args);
                }
                exit(0);
        }
    }
return 0;
}
```

Πρόβλημα 15:
Το πρόγραμμα προσπαθεί να διαβάσει από την δυναμική βιβλιοθήκη tier2.so
Λύση:
Δημιουργούμε την δυναμική βιβλιοθήκη

Πρόβλημα 16:
Το πρόγραμμα θέλει να διαβάσει το pid του σε μια συγκεκριμένη θέση μνήμης
Λύση:
Τροποποιούμε την βιβλιοθήκη μας ώστε μέσω mmap, χωρίς όμως να αφήσουμε στο λειτουργικό την ανάθεση διεύθυνσης, να γράφει το pid του στην ζητούμενη θέση.

```
void setup_tier2() {
        int fd=  open("rand.txt", O_RDWR | O_CREAT, 0666);
        ftruncate(fd, 4096);

        void *buf = mmap((void *)0x[addr], 4096, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);


        *((long int *)buf) = (long int)getpid();
}
```

Πρόβλημα 17:
Ξεκλείδωμα του tier3
Λύση:
Το πρόβλημα στην πραγματικότητα είναι ότι η unlock_tier_3 προσπαθεί να γράψει σε invalid διεύθυνση το οποίο οδηγεί σε segmentation fault μέσω του σήματος SIGSEGV. Βλέπουμε όμως ότι πρώτα τρέχει η setup_tier2, οπότε βάζουμε signal handling σε αυτήν την συνάρτηση και βάζουμε εκεί να τρέξει το unlock_tier3. Μετά χρησιμοποιούμε jumps για να επαναφέρουμε το πρόγραμμα εκεί που είχε μείνει (αφού πρώτα καλέσουμε την authorize_tier_3)

```
jmp_buf buf;

void handle_signal(int sig) {
        void *handle = dlopen("./tier3.so", 2);
        int (* auth)() = (int (*)()) dlsym(handle, "authorize_tier3");

        int rand = auth();

        longjmp(buf, 1);
}

void setup_tier2() {
        signal(SIGSEGV, handle_signal);

        void *handle = dlopen("./tier3.so", 2);

        int (* unlock)() = (int (*)()) dlsym(handle, "unlock_tier3");

        if(!setjmp(buf)) {
                int un = unlock();
        }
        else {}
}
```