Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
7ο εξάμηνο, Ακαδημαϊκή περίοδος 2021 - 2022
http://www.cslab.ece.ntua.gr/

Εργαστήριο Λειτουργικών Συστημάτων

1η Εργαστηριακή Άσκηση: riddle

Η άσκηση αποτελείται από το εκτελέσιμο αρχείο riddle το οποίο περιέχει διάφορες δοκιμασίες που πρέπει να περάσουμε κάνοντας αλλαγές πάνω στο σύστημα που δουλεύουμε (αρχεία, περιβάλλον, εκτέλεση κώδικα κ.λ.π.). Ο κώδικας γράφτηκε σε μοναδικό αρχείο με το όνομα file.c. Η μελέτη του εκτελέσιμου έγινε με τις εντολές strace και ltrace. Και οι 2 εντολές μας δίνουν πληροφορίες για τα διάφορα system calls που καλούνται κατά την εκτέλεση του προγράμματος, καθώς και του δικού μας εκτελέσιμου (file) αλλά και των παιδιών διεργασιών που γεννιούνται σε αυτά (με την χρήση των σημαιών -f, -ff). Η κύρια διαφορά τους είναι ότι η strace βλέπει τα system calls που καλούνται από τις βιβλιοθήκες της C απευθείας στο Kernel του Linux ενώ η ltrace τις βλέπει στο επίπεδο του προγράμματος (εφαρμογής). Η strace κάνει χρήση της ptrace η οποία βλέπει ποια system calls έχει καλέσει η διεργασία (riddle και τα παιδιά της κ.λ.π.).

Βρίσκουμε μικρές διαφορές στην κλήση strace και ltrace. Για παράδειγμα, κάνοντας strace, βλέπουμε ότι καλείται το system call openat() ενώ στην ltrace, βλέπουμε τα system calls σε επίπεδο “κώδικα”, άρα βλέπουμε ότι καλείται η open().

TIER 1:

0. Hello There
Παρατηρούμε ότι η riddle προσπαθεί να ανοίξει ένα αρχείο με το όνομα “.hello_there” με την χρήση της open(). Η κλήση της open() αποτυγχάνει γιατί επιστρέφει τον int -1.
Δημιουργούμε ένα αρχείο με το ίδιο όνομα χρησιμοποιώντας την εντολή touch → *touch .hello_there*

1. Gatekeeper
Η riddle προσπαθεί να ανοίξει το ίδιο αρχείο, .hello_there, αυτή την φορά με την σημαία O_WRONLY. Η πρώτη μαντεψιά είναι να αφαιρέσουμε το δικαίωμα εγγραφής στο αρχείο καθώς το hint γράφει “I found the doors unlocked”. Αυτό το κάνουμε τρέχοντας την εντολή chmod a-w .hello_there από το bash. Η συγκεκριμένη, διαβάζοντας ορθώς το a - w, σημαίνει ότι αφαιρεί(-) από όλους τους χρήστες(a) το δικαίωμα εγγραφής(w).

ΜΕΛΕΤΗ: File Permissions

2. A time to kill
Εδώ βλέπουμε την υλοποίηση ενός Signal Handler. Βλέπουμε ότι καλείται η sigaction(), με πρώτο όρισμα την SIGCONT. Υποθέσαμε ότι θα θέλει το πρόγραμμα να καλέσουμε το σήμα SIGCONT στην διεργασία riddle. Αυτό το καταφέραμε ανοίγοντας 2ο bash terminal, τρέχοντας την riddle, κάνοντας ps -aux στο 2ο terminal, βλέποντας το PID της riddle και μετά στέλνοντας ένα σήμα SIGCONT με την εντολή kill -18 PID, όπου PID της riddle και -18 ο κωδικός της SIGCONT. (Η ps -aux φανερώνει όλες τις διεργασίες που τρέχουν στο σύστημα από τον χρήστη καθώς και διάφορες πληροφορίες για αυτές, όπως το PID τους, ποσοστά που δεσμεύουν από CPU και μνήμη, στιγμή έναρξης κ.α.).

ΜΕΛΕΤΗ: Signals, Signal Handler, Sigaction, kill
3. What is the answer to life the universe and everything
Παρατηρούμε την κλήση της συνάρτησης getenv(), η οποία ψάχνει μια environment variable, συγκεκριμένα με το όνομα “ANSWER”. Ένα καλό και αρκετά γνώριμο environment variable είναι το PATH, το οποίο περιέχει την λίστα όλων των σημείων που ψάχνουν τα Linux για την εκτέλεση εκτελέσιμων αρχείων όταν τρέχουμε εμείς ένα command.
Η προφανής απάντηση στο hint “What is the answer to life the universe and everything” είναι ο αριθμός 42. Θέτουμε ένα καινούργιο environment variable με την τιμή 42 τρέχοντας την εντολή export ANSWER=42

Από εδώ και εμπρός, παρατηρούμε ότι μας δίνονται καινούργια hints αν θέσουμε μια οποιαδήποτε τιμή στο environment variable I_NEED_TECH_HINTS, την οποία αξιοποιούμε κάθε φορά που ξαναξεκινάμε να δουλεύουμε στην άσκηση
ΜΕΛΕΤΗ: Environment Variables

4. First-in, First-out
Αξιοποιώντας τα tech hints, μαζί με το ότι καλείται η open και αποτυγχάνει στο να ανοίξει ένα αρχείο με το όνομα “magic_mirror”, καταλαβαίνουμε ότι πρέπει να δημιουργήσουμε ένα named pipe με το ίδιο όνομα. Αυτό το κάνουμε με την εντολή mkfifo magic_mirror. Η διαφορά των named pipes με των παραδοσιακών pipes είναι η εξής:
Τα απλά pipes, είναι “ανώνυμα” και έχουν διάρκεια ζωής όσο και οι διεργασίες που τα αξιοποιούν για την επικοινωνία τους. Σε αντίθεση, τα named pipes έχουν διάρκεια ζωής, και γενικά λόγο ύπαρξης, ακόμα και αν δεν υπάρχει κάποια διεργασία συνδεδεμένη μαζί τους. Παρουσιάζονται ως αρχεία στο περιβάλλον του ΛΣ και μπορούμε να τα συνδέσουμε σε διεργασίες για να γίνει FIFO επικοινωνία μεταξύ τους.

ΜΕΛΕΤΗ: FIFO, IPC (InterProcess Communication), Named PIPES
Εδώ ξεκινάει για εμάς η εγγραφή κώδικα σε εξωτερικό αρχείο, ονόματι file.c. Ουσιαστικά, γράφουμε κώδικα C σε αυτό και στο τέλος (ή όπου χρειαστεί) καλούμε την riddle (αρχικά με την χρήση της συνάρτησης system(), αργότερα, λίγο πιο ορθά, με την χρήση της execve().)

5. My favourite fd is 99

Η riddle καλεί την fstat() με το όρισμα F_GETFD, στο fd 99. Ψάχνοντας, με βοήθεια του hint, τις συναρτήσεις dup και dup2, καταλάβαμε ότι πρέπει να ανοίξουμε ένα dummy αρχείο και να κάνουμε μεταβολή το fd του ώστε να γίνει 99. Αυτό έγινε εύκολα, με λίγο κώδικα c

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


int main(int argc, char* argv[]) {
  int fd5;
  //5. Solution
  fd5 = open("test",O_CREAT|O_RDWR);
  dup2(fd5,99);
}
```

ΜΕΛΕΤΗ: dup,dup2, File Descriptors

6. Ping pong

Το hint “help us play”, καθώς και η χρήση της fork() για την δημιουργία άλλων processes μας παραπέμπει γρήγορα σε επικοινωνία μεταξύ processes. Τα signals εύκολα διαγράφονται σαν επιλογή, καθώς βλέπουμε ότι η riddle προσπαθεί να διαβάσει από το fd 33. Στην αρχή, με δοκιμαστικό αρχείο είδαμε ότι αν φτιάξουμε αρχείο με fd 33 για διάβασμα και αρχείο με fd 34 για γράψιμο, τότε ζητάει αρχείο με fd 53 για διάβασμα.

Εκεί, παρατήσαμε τα δοκιμαστικά αρχεία και τα αντικαταστήσαμε με 2 πίνακες 2 τιμών τα οποία αξιοποιήσαμε σαν pipes. Έτσι, χρησιμοποιώντας την pipe() και την dup2(). Φτιάξαμε σύστημα επικοινωνίας μεταξύ των διεργασιών για να γίνει η εγγραφή και το διάβασμα των λέξεων PING και PONG.

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


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

  int Pipe1[2], Pipe2[2];
```

```
  //6. Solution
        pipe(Pipe1);
        pipe(Pipe2);
        dup2(Pipe1[0],33);
        dup2(Pipe1[1],34);
        dup2(Pipe2[0],53);
        dup2(Pipe2[1],54);
}
```

ΜΕΛΕΤΗ: Unnamed Pipes

7. What's in a name?

Δημιουργούμε hard link (το αρχείο .hey_there) το οποίο "δείχνει" στο ίδιο σημείο όπως και το αρχείο .hello_there. Δηλαδή, στο ενδεχόμενο που διαγράφαμε το .hello_there, το .hey_there θα κρατούσε την ίδια σύνδεση που είχε το .hello_there στην μνήμη. Αυτό το κάναμε με την εντολή ln .hello_there .hey_there

ΜΕΛΕΤΗ: Hard Links

8. Big Data

Φτιαχνουμε 10 αρχεια bfxx, γραφουμε στο συγκεκριμενο offset καθε φορα με την lseek().
Με ΥΠΕΡΒΟΛΙΚΟ ψάξιμο, και αρκετό διάβασμα για κατανόηση της lseek(), κάναμε τα εξής. Είδαμε πως αρχικά, υπήρχε πρόβλημα στο άνοιγμα αρχείου ονόματι "bf00", το οποίο και δημιουργήσαμε. Μετά, προσπαθούσε 10 φορές στο ίδιο αρχείο να αλλάξει το σημείο διαβάσματος από το 0 στο 1073741824 και να διαβάσει από εκεί 16 χαρακτήρες. Μετά, έκλεινε το αρχείο. Μετά από αρκετές ώρες καταλάβαμε ότι έπρεπε εμείς να γράψουμε σε αυτό το σημείο τουλάχιστον 1 χαρακτήρα, στο αρχείο "bf00". Κάνοντάς το αυτό με κώδικα C, μετά η riddle ζητούσε ακριβώς την ίδια διαδικασία για ένα αρχείο με το όνομα "bf01". Άρα στην τελική φτιάξαμε 10 αρχεία με ονόματα από "bf00" εώς "bf09", με κάπως "τεμπέλικο" τρόπο και γράψαμε ένα μικρό string στο συγκεκριμένο σημείο που ζητούσε η riddle με την βοήθεια της lseek().

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


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

  char* bfxx[10] = {"bf00","bf01","bf02","bf03","bf04","bf05","bf06","bf07","bf08","bf09"};
  int fd[10]; //to fd[10] den xreiazetai, ena aplo int fd arkei.
  //8.Solution
  for (int i = 0; i < 10; i++){
```

```
                fd[i] = open(bfxx[i], O_RDWR|O_CREAT);
                lseek(fd[i], 1073741824, SEEK_SET);
                write(fd[i], "hihi uwu", 16);
                close(fd[i]);
        }
}
```

ΜΕΛΕΤΗ: lseek(), sparse files

9. Connect

Η riddle προσπαθεί να στείλει ένα μήνυμα στην πόρτα 49842. Σε 2ο terminal, τρέχουμε την εντολή nc -l 49842, η οποία επιτρέπει να κάνουμε listen σε αυτή την θύρα και μετά τρέχουμε την riddle. Αφού μας ζητήσει να κάνουμε μια αριθμητική πράξη, τελειώνει η δοκιμασία.

ΜΕΛΕΤΗ: netcat, listen (linux και C)

10. ESP

Η riddle ανοίγει ένα αρχείο (“secret_number”), κάνει unlink() το αρχείο ώστε να μην μπορούμε (με απλά μέσα) να το ανοίξουμε εμείς, γράφει τον αριθμό που “σκέφτεται” και μετά μας τον ζητάει. Στο συγκεκριμένο, το μυαλό μας γρήγορα πήγε σε hard link, ώστε να σωθεί η πληροφορία που γράφτηκε στο hard link που θα δημιουργήσουμε και θα ενώσουμε με το αρχείο “secret_number” που ανοίγει η riddle. Κάνουμε touch secret_number, μετά ln secret_number dummy, ώστε να δημιουργηθεί το hard link με όνομα dummy. Σε 2ο terminal, ανοίγουμε την riddle και μετα το dummy, το οποίο όντως έχει σωσμένη την πληροφορία την οποία δίνουμε στο riddle.

ΜΕΛΕΤΗ: unlink(), mmap(), Hard Links

11. ESP-2

Προφανώς, δεν θα μας άφηνε να περάσουμε την επόμενη δοκιμασία με τον ίδιο τρόπο. Τρέχοντας την fstat, η riddle μπορούσε να δει εάν υπήρχε hard link στο secret_number και έκανε απευθείας fail εάν το έβλεπε.

Άμεση σκέψη, να υλοποιήσουμε πρόγραμμα που θα διαβάζει κομμάτι της μνήμης που κάνει map η mmap(). Παρατηρούμε ότι η mmap() έχει το όρισμα MAP_SHARED, δηλαδή, επιτρέπεται να μοιραστεί μεταξύ διεργασιών.

Αποφασίσαμε να χρησιμοποιήσουμε την fork() και μετά να κάνουμε έλεγχο για την διεργασία πατέρα και την διεργασία παιδί:

- Αν είμαστε η διεργασία πατέρας, τότε κάνουμε sleep() ώστε να προλάβει να τρέξει η διεργασία παιδί και μετά διαβάζουμε ένα μέγεθος 100 θέσεων σε ενα buffer από το fd του secret_number και κάνουμε print αυτόν τον buffer.
- Αν είμαστε διεργασία παιδί, απλά τρέχουμε την riddle με την execve().

ΜΕΛΕΤΗ: unlink(), mmap(), shared memory with fork() and mmap()

12. A delicate change
Το πρόγραμμα περιμένει να δει έναν χαρακτήρα επιλογής του σε συγκεκριμένη διεύθυνση της εικονικής μνήμης. Βλέπουμε ότι όσες φορές και να τρέξουμε την strace, η διαφορά μεταξύ της διεύθυνσης εικονικής μνήμης που επιστρέφει η mmap() με την διεύθυνση μνήμης που μας ζητάει η riddle είναι πάντα 0x06f. Άρα έχουμε ένα σταθερό offset το οποίο μπορούμε να αξιοποιήσουμε με την lseek(). Επίσης, στο strace βλέπουμε την riddle να ανοίγει ένα temporary memory map file στο root.

Γράφουμε κώδικα C η οποία παίρνει 2 ορίσματα στο κάλεσμα του, το πρώτο είναι το path του temporary αρχείου μνήμης και το δεύτερο είναι ο χαρακτήρας που θέλει να δει η riddle. Ανοίγουμε με open() αυτό το αρχείο και αλλάζουμε την θέση του write() με την χρήση της lseek() κατά offset 0x06f. Τέλος, γράφουμε τον χαρακτήρα στην θέση αυτή με την χρήση της write().

ΜΕΛΕΤΗ: lseek(), mmap(), memory map files (temporary)
13. Bus error
Η riddle τρέχει την truncate() η οποία αλλάζει το size του αρχείου που είχαμε ανοίξει (fd=4 για το .hello_there). Αυτό το κάνει αρχικά για μια τιμή μεγέθους truncate(fd, 32768). Μετά την mmap(), ξανατρέχει την truncate αυτή την φορά για truncate(fd, 16384). Αυτό προκαλεί σοβαρό πρόβλημα καθώς είχαμε κάνει memory map με συγκεκριμένο τρόπο το αρχείο και έπειτα του αλλάζουμε το μέγεθος. Για να το διορθώσουμε αυτό, πρέπει να επαναφέρουμε το αρχείο στο αρχικό του μέγεθος με το οποίο κάναμε το memory map. Αυτό γίνεται με την χρήση πάλι της truncate().

Γράφουμε κώδικα C που κάνει ακριβώς αυτή την ενέργεια, δηλαδή, ανοίγει με την open() το αρχείο “.hello_there” και μετά τρέχει την truncate για το fd που γύρισε το open και τιμή 32768 → truncate(fd, 32768).

ΜΕΛΕΤΗ: truncate()

14. Are you the One?
Η riddle περιμένει να τρέξει με συγκεκριμένο PID 32767. Για να το κάνουμε αυτό, τρέχουμε loop κάνοντας συνέχεια fork() και ελέγχουμε για το πότε φτάνουμε σε αυτό το PID, στο οποίο μετά τρέχουμε την riddle με χρήση της execve(). Τον έλεγχο αυτό τον κάνουμε χρησιμοποιώντας την getpid(), η οποία επιστρέφει το pid της διεργασίας που την κάλεσε.

ΜΕΛΕΤΗ: fork(), getpid()