Министерство образования и науки Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Кафедра №304

Отчет по лабораторной работе №4

«**Межпроцессоное взаимодействие в Linux на примере программы – шифратора файлов**»

по дисциплине «Операционные системы»

Выполнил

Студент группы 3О-310Б

Сомов Д.Н.

Приняли

ассистент каф.304

Балян А.В.

ассистент каф.304

Аляутдин Р.Т.

Москва, 2017

**Задание:**

Ознакомиться с системными функциями Linux, позволяющими передавать данные между процессами. С помощью полученных знаний написать программу-шифратор, запускающую несколько процессов и настраивающую связь между ними. Тип шифрования – гаммирование (операция XOR). Проверить написанную программу с помощью Memory Sanitizer, checkpatch.pl. Использовать систему контроля версий git и удаленный репозиторий на GitHub.

**Описание:**

Программа xor при запуске запрашивает у пользователя путь и имя файла, которые нужно зашифровать. Далее она запускает с помощью функций fork() и exec() два процесса. Первый – это команда cat с параметром – введенным именем файла, который нужно будет читать. Для программы cat создается труба с помощью функции pipe(), и дескрипторы настраиваются таким образом, чтобы стандартный вывод для программы cat был на самом деле входным дескриптором трубы. Вторая программа – gen – это генератор случайной последовательности байт, которая используется как XOR-шифр для гаммирования. Генератор принимает входной дескриптор трубы как параметр и читает последовательность в неё. Выходные дескрипторы обеих труб – у программы xor. Она читает из обеих труб одинаковую порцию байт (т.е. кусочек файла и кусочек шифра), шифрует считанный кусок файла и записывает байты шифрованного файла и ключа на диск. Это продолжается до тех пор, пока в трубе программы cat не закончатся байты исходного файла, после чего программа xor закрывает все дескрипторы, прерывает генератор и завершается.

Программа unxor действует похожим образом; она форкает два процесса cat – один читает шифрованный файл, другой – файл с ключом, – создает и перенаправляет входы труб и затем, читая одинаковые порции байт, пока шифрованный файл не закончится, расшифровывает его и записывает на диск. Как только процесс расшифровки завершен, дескрипторы закрываются, cat’ы останавливаются, и программа завершается.

**Исходный код:**

/\*

\* Programming language – C

\* Author – Dmitriy Somov, 3O-310B

\*

\* **File: lab4.h**

\*/

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

#define BUFSIZE 512

/\*

\* Programming language – C

\* Author – Dmitriy Somov, 3O-310B

\*

\* **File: xor.c**

\*/

#include "lab4.h"

pid\_t genPid;

void AtExit(int exitCode)

{

kill(genPid, SIGINT);

// Parent waits for child's command completion.

wait(NULL);

exit(exitCode);

}

int main(void)

{

int gen\_descs[2];

int cat\_descs[2];

pid\_t catPid;

char \*out;

char path[BUFSIZE];

char buf[BUFSIZE];

// Read path and name of the file to cipher.

memset(path, '\0', BUFSIZE);

out = "What file you want to XOR?\n: ";

write(1, out, strlen(out));

read(0, path, BUFSIZE);

path[strlen(path)-1] = '\0'; // Get rid of '\n' at the end.

// Create a pipe to connect generator and cipher.

if (pipe(gen\_descs) == -1) {

perror("Error creating generator pipe");

AtExit(1);

}

// Fork and run the generator.

genPid = fork();

switch (genPid) {

case -1:

perror("Error forking the generator");

AtExit(2);

break;

case 0:

// We're sending pipe descriptor to the generator.

memset(buf, '\0', BUFSIZE);

sprintf(buf, "%d", gen\_descs[1]);

execlp("./gen\_m", "./gen\_m", buf, NULL);

break;

}

// Create a pipe for cat process.

if (pipe(cat\_descs) == -1) {

perror("Error creating catalog pipe");

AtExit(3);

}

// Fork and run cat-filewalker.

catPid = fork();

switch (catPid) {

case -1:

perror("Error forking the filewalker");

AtExit(4);

break;

case 0:

// Before we run cat, we need to redirect its output to our pipe.

close(cat\_descs[0]);

// Duplicate our pipe input descriptor as stdout.

if (dup2(cat\_descs[1], 1) == -1) {

perror("Error while redirecting cat input");

AtExit(5);

}

close(cat\_descs[1]);

execlp("cat", "cat", path, NULL);

break;

default:

close(cat\_descs[1]);

sleep(1); // Wait a bit for generator.

char xor[BUFSIZE];

int readed;

int genreaded;

int i;

int xorfile;

int xorkey;

// Get ready to write ciphed file and key to decipher it later.

memset(buf, '\0', BUFSIZE);

memset(xor, '\0', BUFSIZE);

strcat(buf, path);

strcat(buf, ".crypted");

xorfile = open(buf, O\_RDWR|O\_CREAT, 0600);

if (xorfile == -1) {

perror("Failed to create xor-file");

AtExit(6);

}

strcat(buf, ".key");

xorkey = open(buf, O\_RDWR|O\_CREAT, 0600);

if (xorkey == -1) {

perror("Failed to create key-file");

AtExit(7);

}

memset(buf, '\0', BUFSIZE);

// While cat haven't read the entire file...

while ((readed = read(cat\_descs[0], buf, BUFSIZE)) > 0) {

genreaded = read(gen\_descs[0], xor, readed);

// printf("readed = %i, genreaded = %i\n", readed, genreaded);

if (genreaded != readed) {

printf("Seems like generator works too slow.");

printf(" Results will be incorrect. Stopping.\n");

AtExit(8);

}

// Apply XOR operation and save readed key.

for (i = 0; i < readed; i++)

buf[i] = buf[i] ^ xor[i];

i = write(xorfile, buf, readed);

if (i != readed) {

perror("Error while writing into xor-file");

AtExit(9);

}

i = write(xorkey, xor, readed);

if (i != readed) {

perror("Error while writing into key-file");

AtExit(10);

}

memset(buf, '\0', BUFSIZE);

memset(xor, '\0', BUFSIZE);

}

close(cat\_descs[0]);

close(xorfile);

close(xorkey);

AtExit(0);

}

}

/\*

\* Programming language – C

\* Author – Dmitriy Somov, 3O-310B

\*

\* **File: unxor.c**

\*/

#include "lab4.h"

pid\_t catkeyPid;

pid\_t catfilePid;

void AtExit(int exitCode)

{

kill(catkeyPid, SIGINT);

kill(catfilePid, SIGINT);

// Parent waits for child's command completion.

wait(NULL);

exit(exitCode);

}

int main(void)

{

int catkey\_descs[2];

int catfile\_descs[2];

char \*out;

char path[BUFSIZE];

char buf[BUFSIZE];

// Read path and name of the file we want to decipher.

memset(path, '\0', BUFSIZE);

out = "What file you want to unXOR?\n: ";

write(1, out, strlen(out));

read(0, path, BUFSIZE);

path[strlen(path)-1] = '\0'; // Get rid of '\n' at the end

// Create pipe to connect decipher and XOR-key file.

if (pipe(catkey\_descs) == -1) {

perror("Error creating cat key-file pipe");

AtExit(1);

}

// Fork and run cat which will send key bytes to redirected stdout.

catkeyPid = fork();

switch (catkeyPid) {

case -1:

perror("Error forking cat key-file");

AtExit(2);

break;

case 0:

close(catkey\_descs[0]);

// Here is where redirection is performed:

if (dup2(catkey\_descs[1], 1) == -1) {

perror("Error while redirecting cat key input");

AtExit(3);

}

close(catkey\_descs[1]);

memset(buf, '\0', BUFSIZE);

strcat(buf, path);

strcat(buf, ".key");

execlp("cat", "cat", buf, NULL);

break;

}

// Create pipe for connecting ciphed file and decipher.

if (pipe(catfile\_descs) == -1) {

perror("Error creating file pipe");

AtExit(4);

}

// Again, fork, redirect and run filewalker...

catfilePid = fork();

switch (catfilePid) {

case -1:

perror("Error forking the filewalker");

AtExit(5);

break;

case 0:

close(catfile\_descs[0]);

if (dup2(catfile\_descs[1], 1) == -1) {

perror("Error while redirecting cat file input");

AtExit(6);

}

close(catfile\_descs[1]);

execlp("cat", "cat", path, NULL);

break;

default:

close(catfile\_descs[1]);

sleep(1); // Wait for keywalker to read some bytes ahead.

char xor[BUFSIZE];

int filereaded;

int keyreaded;

int i;

int xorfile;

// We should write down deciphed file.

memset(buf, '\0', BUFSIZE);

memset(xor, '\0', BUFSIZE);

strcat(buf, path);

buf[strlen(buf)-8] = '\0';

strcat(buf, ".decrypted");

xorfile = open(buf, O\_RDWR|O\_CREAT, 0600);

if (xorfile == -1) {

perror("Failed to create decrypted file");

AtExit(7);

}

memset(buf, '\0', BUFSIZE);

// While ciphed file haven't read to the end...

while ((filereaded = read(catfile\_descs[0], buf, BUFSIZE)) > 0) {

keyreaded = read(catkey\_descs[0], xor, filereaded);

//printf("filereaded = %i, keyreaded = %i\n", filereaded, keyreaded);

if (filereaded != keyreaded) {

printf("Seems like filewalker is waaaay ahead keywalker.");

printf(" Results will be incorrect. Stopping.\n");

AtExit(8);

}

// Apply XOR to decipher.

for (i = 0; i < filereaded; i++)

buf[i] = buf[i] ^ xor[i];

i = write(xorfile, buf, filereaded);

if (i != filereaded) {

perror("Error while writing into decrypted file");

AtExit(9);

}

memset(buf, '\0', BUFSIZE);

memset(xor, '\0', BUFSIZE);

}

close(catfile\_descs[0]);

close(catkey\_descs[0]);

close(xorfile);

AtExit(0);

}

}

/\*

\* Programming language – C

\* Author – Dmitriy Somov, 3O-310B

\*

\* **File: gen.c**

\*/

#include "lab4.h"

int pipeDesc;

void AtInterruption(int signal)

{

// The singal will be sent by xor/unxor program.

close(pipeDesc);

printf("Stopping generator...\n");

exit(0);

}

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

{

char randnum[10];

int i;

signal(SIGINT, AtInterruption);

srand(time(NULL));

// Get pipe descriptor...

if (sscanf(argv[1], "%d", &pipeDesc) != 1) {

perror("sscanf error in gen.c");

exit(1);

}

// Just continiously write random bytes into the pipe.

while (1) {

for (i = 0; i < 10; i++)

randnum[i] = (char)(rand() % 256);

write(pipeDesc, randnum, 10);

}

}

# Author – Dmitriy Somov, 3O-310B

#

# **File: makefile**

regular:

gcc -o gen gen.c

gcc -o xor xor.c

gcc -o unxor unxor.c

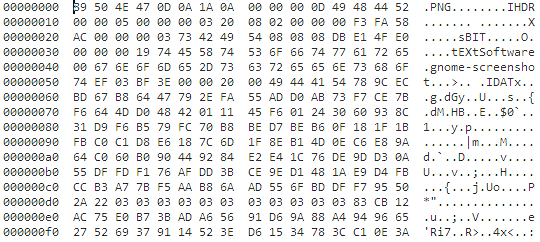
memsan:

gcc -o gen\_m gen.c -fsanitize=address

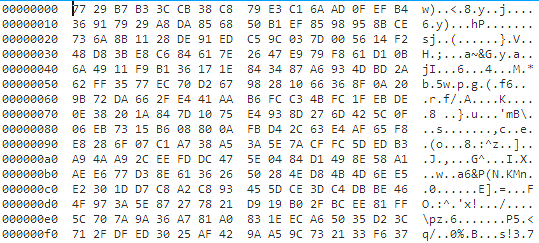
gcc -o xor\_m xor.c -fsanitize=address

gcc -o unxor\_m unxor.c -fsanitize=address

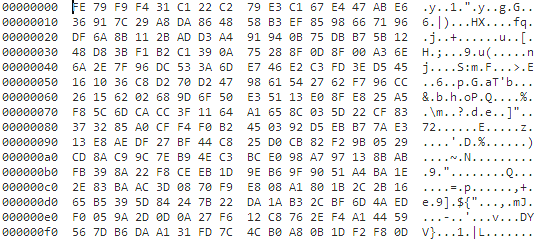
**Исходный файл (в hex-представлении):**



**Зашифрованный файл (в hex-представлении):**



**Файл-ключ (в hex-представлении):**



**Расшифрованный файл (в hex-представлении):**

