**ФЕДЕРАЛЬНО ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«**Национальный исследовательский университет ИТМО**»**

Факультет безопасности информационных технологий



**Дисциплина:**

«Основы системного программирования»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1-2**

**Выполнил:**

**Студент:**



Чан Ван Хоанг

Группы: N3249

**Преподаватель:**

Грозов В.А.

Санкт Петербург

2021

**Вариант 14: Опция: --exe <значение>**

**Назначение: поиск исполняемых файлов в форматах ELF, PE32, a.out и COFF.**

**Признаком формата считать соответствующее магическое число в заголовке**

**файла. Значением опции является строка, в которой перечисляются через запятую**

**без пробелов форматы, которые требуется найти.**

**Пример: --exe pe32,elf,a.out**

**I. Исходные тексты программ с комментариями**

**1. Makefile**

CFLAGS=-Wall -Wextra -Werror -O2

TARGETS=lab1test lab1tvhN3249 libtvhN3249.so libtvhN3249-2.so

.PHONY: all clean

all: $(TARGETS)

clean:

rm -rf \*.o $(TARGETS)

lab1test: lab1test.c plugin\_api.h

gcc $(CFLAGS) -o lab1test lab1test.c -ldl

lab1tvhN3249: lab1tvhN3249.c plugin\_api.h

gcc $(CFLAGS) -o lab1tvhN3249 lab1tvhN3249.c -ldl

libtvhN3249.so: libtvhN3249.c plugin\_api.h

gcc $(CFLAGS) -shared -fPIC -o libtvhN3249.so libtvhN3249.c -ldl -lm

libtvhN3249-2.so: libtvhN3249-2.c plugin\_api.h

gcc $(CFLAGS) -shared -fPIC -o libtvhN3249-2.so libtvhN3249-2.c -ldl -lm

**2. valgrind**

==13090== HEAP SUMMARY:

==13090== in use at exit: 0 bytes in 0 blocks

==13090== total heap usage: 62 allocs, 62 frees, 219,215 bytes allocated

==13090==

==13090== All heap blocks were freed -- no leaks are possible

==13090==

==13090== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

**3. lab1tvhN3249.c**

#include <errno.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <dlfcn.h>

#include <dirent.h>

#include "plugin\_api.h"

typedef int (\*ppf\_func\_t)(const char\*, struct option\*, size\_t); //for plugin\_process\_file

typedef int (\*pgi\_func\_t)(struct plugin\_info\*); // for plugin\_get\_info;

struct longopt {

struct option \*all\_opt ; //array for all longopt

size\_t all\_opt\_len ;

struct option \*opts\_to\_pass ;

size\_t opts\_to\_pass\_len ;

ppf\_func\_t func;

pgi\_func\_t info;

};

int count\_so (const char\* dirname, int\* len) {

DIR\* dir = opendir(dirname);

if (dir == NULL) {

fprintf(stderr, " ERROR: No directory %s\n", dirname);

return -1;

}

struct dirent\* entity;

entity = readdir(dir);

\*len = 0;

while (entity != NULL) {

int flen = strlen(entity->d\_name);

if ((entity->d\_type == DT\_REG) && (entity->d\_name[flen-1] == 'o') && (entity->d\_name[flen-2] == 's') && (entity->d\_name[flen-3] == '.')) {

(\*len)++;

fprintf(stdout,"lib\_name: %s\n",entity->d\_name);

}

entity = readdir(dir);

}

closedir(dir);

return 0;

}

int option\_p (const char\* dirname, void\* dl[], int len) {

DIR\* dir = opendir(dirname);

if (dir == NULL) {

fprintf(stderr, " ERROR: No directory %s\n", dirname);

return -1;

}

struct dirent\* entity;

entity = readdir(dir);

int index = 0;

while (entity != NULL && index < len) {

int flen = strlen(entity->d\_name);

if ((entity->d\_type == DT\_REG) && (flen > 3) && (entity->d\_name[flen-1] == 'o') && (entity->d\_name[flen-2] == 's') && (entity->d\_name[flen-3] == '.')) {

size\_t file\_name\_len = strlen(dirname) + strlen(entity->d\_name) + 2 ;

char\* file\_name = malloc(file\_name\_len);

sprintf(file\_name, "%s/%s", dirname, entity->d\_name);

dl[index] = dlopen(file\_name, RTLD\_LAZY);

if (dl[index] == NULL) {

fprintf(stderr, "ERROR: Failed to dlopen %s\n%s\n", entity->d\_name, dlerror());

return -1;

}

else {

index++;

}

free(file\_name);

}

entity = readdir(dir);

}

closedir(dir);

return 0;

}

//fun file recursive search;

int res\_file (const char\* dirname, int tlen, struct longopt sup\_all[], int is\_or, int is\_not) {

DIR\* dir = opendir(dirname);

if(dir !=NULL ){

struct dirent\* entity;

entity = readdir(dir);

while (entity != NULL) {

if(strcmp(entity->d\_name, ".") != 0 && strcmp(entity->d\_name, "..") != 0){

// printf("lol : %s\n",entity->d\_name);

size\_t path\_len = strlen(dirname) + strlen(entity->d\_name) + 2;

char\* path = malloc(path\_len);

snprintf(path, path\_len, "%s/%s", dirname, entity->d\_name);

if(entity->d\_type == DT\_DIR){

int res = res\_file(path, tlen, sup\_all, is\_or, is\_not);

if (res){

free(path);

return -1;

}

}

if(entity->d\_type == DT\_REG) {

int ret\_true = 0; // if plugin retrun true ret++;

int plugins\_call = 0; //\_count the number of plugins called

for (int i=0; i < tlen; i++){

if(sup\_all[i].opts\_to\_pass\_len > 0) {

plugins\_call++ ;

int ret\_fun = sup\_all[i].func(path, sup\_all[i].opts\_to\_pass, sup\_all[i].opts\_to\_pass\_len);

// fprintf(stdout, "%d %d \n", i , ret\_fun);

if (ret\_fun == 0) ret\_true++;

if (ret\_fun < 0){

free(path);

errno = 0;

fprintf(stdout, "Error information: %s\n", strerror(errno));

return -1;

}

}

}

if(plugins\_call){

// short\_opt A and no opt;

if ( ret\_true == plugins\_call && is\_or == 0 && is\_not == 0) fprintf(stdout, "%s\n", entity->d\_name);

//short\_opt O;

else if (ret\_true > 0 && is\_or == 1 && is\_not == 0) fprintf(stdout, "%s\n", entity->d\_name);

//short\_opt NA;

else if (ret\_true < plugins\_call && is\_or ==0 && is\_not ==1) fprintf(stdout, "%s\n", entity->d\_name);

//short\_opt NO;

else if (ret\_true == 0 && is\_or == 1 && is\_not == 1) fprintf(stdout, "%s\n", entity->d\_name);

}

}

free(path);

}

entity = readdir(dir);

}

closedir(dir);

}

return 0;

}

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

struct longopt \*sup\_all = 0;

char \*f\_name = 0;

opterr = 0;

int is\_o = 0, is\_n = 0 , is\_v = 0, is\_h = 0 ,is\_P = 0 ;

// short\_option A is\_a = 1 (if is\_o == 0 and is\_n == 0)

int len = 0;

void\*\* dl = 0;

char \*\*new\_argv = (char\*\*) malloc (argc \* sizeof(char\*));

if(!new\_argv){

fprintf(stdout,"ERROR: could not allocate for argv copy\n");

}

memcpy(new\_argv, argv, argc \* sizeof(char\*));

// Minimum number of arguments is 2:

// $ program\_name --opts file to ch

if (argc < 2) {

fprintf(stdout, "Usage: ./main -short\_opt --[options\_for\_lib] /path/to/file\n");

fprintf(stdout, "Short\_options:\n");

fprintf(stdout, "\t\t-P: Plugin directory\n");

fprintf(stdout, "\t\t-A: Combine plugin options using the 'AND' operation\n");

fprintf(stdout, "\t\t-O: Combine plugin options using the 'OR' operation.\n");

fprintf(stdout, "\t\t-N: Inverting the search term (after combining options plugins with -A or -O)\n");

fprintf(stdout, "\t\t-v: Displaying performer's full name, group number,lab version number o\n");

fprintf(stdout, "\t\t-h: Display help for options.\n");

fprintf(stdout, "Long\_options in plugin:\n");

is\_h = 1;

goto START;

}

int ret\_shrt = 0;

while((ret\_shrt = getopt(argc,new\_argv, "P:vhAON"))!=-1) {

switch(ret\_shrt) {

case 'P':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

is\_P = 1;

if (count\_so(optarg, &len)){

fprintf(stderr, "ERROR: unable to count file.so\n");

goto END;

}

dl = calloc (len, sizeof(void\*));

if (option\_p(optarg, dl, len)){

fprintf(stderr, "ERROR: unable to open file.so\n");

goto END;

}

if( optind == argc ) {

is\_h = 1;

goto START;

}

if(argv[optind][1] == '-') {

goto START;

}

break;

case 'v':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

is\_v = 1 ;

if( optind == argc ) goto START;

if( argv[optind][1] == '-' ) goto START;

break;

case 'h':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

is\_h = 1;

if( optind == argc ) goto START;

if( argv[optind][1] == '-' ) goto START;

break;

case 'A':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

if(argv[optind][1] == '-') goto START;

break;

case 'O':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

is\_o = 1;

if(argv[optind][1] == '-') goto START;

break;

case 'N':

fprintf(stdout, "Short option -%c is detected!\n", ret\_shrt);

is\_n = 1;

if(argv[optind][1] == '-') goto START;

break;

}

}

START:

if(!is\_P) {

if (count\_so(".", &len)) {

fprintf (stderr, "ERROR: unable to count in curren dir\n");

goto END;

}

dl = calloc (len, sizeof(void\*));

if (option\_p(".", dl, len)){

fprintf (stderr,"ERROR: unable to dlopen libs in a current dir\n");

goto END;

}

}

sup\_all = calloc (len, sizeof(struct longopt));

for (int i = 0; i < len; i++){

// Check for plugin\_get\_info()

sup\_all[i].info = dlsym(dl[i], "plugin\_get\_info");

if(!sup\_all[i].info){

fprintf(stderr, "ERROR: dlsym() failed: %s\n", dlerror());

goto END;

}

struct plugin\_info pi = {0};

if (sup\_all[i].info(&pi) < 0){

fprintf(stderr, "ERROR: plugin\_get\_info() failed\n");

goto END;

}

if (pi.sup\_opts\_len == 0){

fprintf(stderr, "ERROR: library supports no options! How so?\n");

goto END;

}

// Plugin info and printf short option v , h;

if(is\_v == 1){

fprintf(stdout, "Plugin purpose:\t\t%s\n", pi.plugin\_purpose);

fprintf(stdout, "Plugin author:\t\t%s\n", pi.plugin\_author);

if (i == len -1)goto END;

}

if(is\_h == 1){

fprintf(stdout, "Supported options: ");

if (pi.sup\_opts\_len > 0) {

fprintf(stdout, "\n");

for (size\_t j = 0; j < pi.sup\_opts\_len; j++) {

fprintf(stdout, "\t--%s\t\t%s\n", pi.sup\_opts[j].opt.name, pi.sup\_opts[j].opt\_descr);

}

}

else{

fprintf(stdout, "none (!?)\n");

}

fprintf(stdout, "\n");

if (i == len - 1) goto END;

}

// Get pointer to plugin\_process\_file()

sup\_all[i].func = dlsym(dl[i], "plugin\_process\_file");

if(!sup\_all[i].func) {

fprintf(stderr, "ERROR: no plugin\_process\_file() function found\n");

goto END;

}

// Prepare array of options for getopt\_long

sup\_all[i].all\_opt\_len = pi.sup\_opts\_len;

sup\_all[i].all\_opt = calloc(pi.sup\_opts\_len + 1, sizeof(struct option));

if (!sup\_all[i].all\_opt){

fprintf(stderr, "ERROR: calloc() failed:%s\n", strerror(errno));

goto END;

}

// copy option information

for (size\_t j = 0; j < pi.sup\_opts\_len; j++) {

memcpy(&sup\_all[i].all\_opt[j], &pi.sup\_opts[j].opt, sizeof(struct option));

}

// Prepare array of actually used options that will be passed to

// plugin\_process\_file() (Maximum pi.sup\_opts\_len options)

sup\_all[i].opts\_to\_pass\_len = 0;

sup\_all[i].opts\_to\_pass = calloc(pi.sup\_opts\_len, sizeof(struct option));

if(!sup\_all[i].opts\_to\_pass) {

fprintf(stderr, "ERROR: calloc() failed: %s\n", strerror(errno));

goto END;

}

}

// Now process options for the lib

for (int i = 0; i < len; i++) {

optind = 1;

memcpy(new\_argv, argv, argc \* sizeof(char\*));

while (1){

int opt\_ind = 0;

int ret = getopt\_long\_only(argc, new\_argv, "", sup\_all[i].all\_opt, &opt\_ind);

if (ret == -1) break;

if(ret != 0) continue;

// Check how many options we got up to this moment

if ((size\_t) sup\_all[i].opts\_to\_pass\_len == sup\_all[i].all\_opt\_len){

fprintf(stderr, "ERROR: too many options!\n");

goto END;

}

// Add this option to array of options actually passed to plugin\_process\_file()

memcpy(sup\_all[i].opts\_to\_pass + sup\_all[i].opts\_to\_pass\_len, sup\_all[i].all\_opt + opt\_ind, sizeof(struct option));

// Argument (if any) is passed in flag

if ((sup\_all[i].all\_opt + opt\_ind)->has\_arg) {

// Mind this!

// flag is of type int\*, but we are passing char\* here (it's ok to do so).

(sup\_all[i].opts\_to\_pass + sup\_all[i].opts\_to\_pass\_len)->flag = (int\*)strdup(optarg);

}

sup\_all[i].opts\_to\_pass\_len++;

}

}

if (getenv("LAB1DEBUG")) {

for (int i = 0; i < len; i++){

fprintf(stderr, "DEBUG: opts\_to\_pass\_len = %ld\n", sup\_all[i].opts\_to\_pass\_len);

for (size\_t j = 0; j < sup\_all[i].opts\_to\_pass\_len; j++) {

fprintf(stderr, "DEBUG: passing option '%s' with arg '%s'\n",

(sup\_all[i].opts\_to\_pass[j]).name,

(char\*)(sup\_all[i].opts\_to\_pass[j]).flag);

}

}

}

fprintf(stdout, "The options are passed to libs!!! \n");

fprintf(stdout,"--------------------------------------\n");

fprintf(stdout,"The list of files that satisfy the requirements is:\n");

// Call fun recursive search and plugin\_process\_file()

errno = 0;

f\_name = strdup(argv[argc-1]);

int ret\_main = res\_file(f\_name, len, sup\_all, is\_o, is\_n);

fprintf(stdout,"--------------------------------------\n");

fprintf (stdout, "fun res\_file() returned %d\n",ret\_main);

if(ret\_main < 0){

fprintf(stderr, "error infomation: %s\n", strerror(errno));

}

END:

if (sup\_all){

for (int i = 0; i < len; i++){

for(size\_t j = 0; j < sup\_all[i].opts\_to\_pass\_len; j++){

if (sup\_all[i].opts\_to\_pass[j].flag) free(sup\_all[i].opts\_to\_pass[j].flag);

}

if (sup\_all[i].opts\_to\_pass) free(sup\_all[i].opts\_to\_pass);

if (sup\_all[i].all\_opt) free(sup\_all[i].all\_opt);

}

free(sup\_all);

}

if(new\_argv) free(new\_argv);

if(f\_name) free(f\_name);

if (dl){

for (int i = 0; i < len; i++){

if(dl[i]) dlclose(dl[i]);

}

free (dl);

}

return 0;

}

**4. libtvhN3249.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <dirent.h>

#include "plugin\_api.h"

static char \*g\_lib\_name = "libtvhN3249.so";

static char \*g\_plugin\_purpose = "Find type of file";

static char \*g\_plugin\_author = "Tran Van Hoang - N3249";

#define OPT\_exe "exe"

static struct plugin\_option g\_po\_arr[] = {

/\*

struct plugin\_option {

struct option {

const char \*name;

int has\_arg;

int \*flag;

int val;

} opt,

char \*opt\_descr

}

\*/

{

{

OPT\_exe,

required\_argument,

0, 0,

},

"File file with type elf, pe32, coff, a.out "

},

};

static int g\_po\_arr\_len = sizeof(g\_po\_arr)/sizeof(g\_po\_arr[0]);

int plugin\_get\_info(struct plugin\_info\* ppi) {

if (!ppi) {

fprintf(stderr, "ERROR: invalid argument\n");

return -1;

}

ppi->plugin\_purpose = g\_plugin\_purpose;

ppi->plugin\_author = g\_plugin\_author;

ppi->sup\_opts\_len = g\_po\_arr\_len;

ppi->sup\_opts = g\_po\_arr;

return 0;

}

static char \*input\_type = NULL;

int plugin\_process\_file(const char \*fname,

struct option in\_opts[],

size\_t in\_opts\_len) {

int ret = -1;

char \*DEBUG = getenv("LAB1DEBUG");

if (!fname || !in\_opts || !in\_opts\_len) {

errno = EINVAL;

return -1;

}

int got\_input\_type = 0;

int tmp\_type = 0;

//check value op (type input)

#define OPT\_CHECK(opt\_var, tmp\_type) \

if (got\_##opt\_var) { \

if (DEBUG) { \

fprintf(stderr, "DEBUG: %s: Option '%s' was already supplied\n", \

g\_lib\_name, in\_opts[i].name); \

} \

errno = EINVAL; \

return -1; \

} \

else { \

char \*endptr = NULL; \

tmp\_type = strtol((char\*)in\_opts[i].flag, &endptr, 10); \

if(tmp\_type!=0){\

if (DEBUG) { \

fprintf(stderr, "DEBUG: %s: Failed to convert '%s'\n", \

g\_lib\_name, (char\*)in\_opts[i].flag); \

} \

errno = EINVAL; \

return -1; \

}\

opt\_var=endptr; \

got\_##opt\_var = 1; \

}

for (size\_t i = 0; i < in\_opts\_len; i++) {

if (!strcmp(in\_opts[i].name, OPT\_exe)) {

OPT\_CHECK(input\_type, tmp\_type)

}

else {

errno = EINVAL;

return -1;

}

}

FILE \*fp;

// Get file name from user. The file should be

// either in current folder or complete path should be provided

// Open the file

fp = fopen(fname, "rb");

static unsigned char magic[4];

// Check if file exists

if (fp == NULL) {

fprintf(stderr, "Error : Failed to open entry file - %s\n", strerror(errno));

return -1;

}

size\_t k = fread(magic,1,4,fp);

k++;

fclose(fp);

// Extract characters from file and store in character c

int loop;

int i;

i=0;

loop=0;

static char output[9];

while(magic[loop] != '\0')

{

sprintf((char\*)(output+i),"%02x", magic[loop]);

loop+=1;

i+=2;

}

//insert NULL at the end of the output string

output[i++] = '\0';

char c[100];

strcpy(c,input\_type);

if (strcmp(output,"7f454c46")==0 && strstr(c,"elf")!=NULL) {

return ret = 0;

}

if (strstr(output,"ffd8ff")!=NULL && strstr(c,"jpeg")!= NULL) {

return ret = 0;

}

if (strstr(output,"424d") != NULL && strstr(c,"bmp")!= NULL) {

return ret = 0;

}

if (strstr(output,"47494638")!=NULL && strstr(c,"gif")!=NULL) {

return ret = 0;

}

if (strstr(output, "89504e47") != NULL && strstr(c,"png") != NULL) {

return ret = 0;

}

else if (strstr(output,"4d5a")!=NULL && strstr(c,"pe32")!= NULL) {

return ret = 0;

}

// else if (strstr(output,"4c01")> 0) {

// if (strstr(input\_type, "coff")!=NULL)

// return ret = 0;

// else return ret = 1;

// }

else if ((strstr(output,"0410")!=NULL || strstr(output,"0413")!=NULL) && strstr(c,"a.out")!= NULL) {

return ret = 0;

}

else return ret = 1;

}

**5. libtvhN3249-2.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <dirent.h>

#include "plugin\_api.h"

static char \*g\_lib\_name = "libtvhN3249-2.so";

static char \*g\_plugin\_purpose = "Find all file with size smaller than input size";

static char \*g\_plugin\_author = "Tran Van Hoang - N3249";

#define OPT\_size "size"

static struct plugin\_option g\_po\_arr[] = {

/\*

struct plugin\_option {

struct option {

const char \*name;

int has\_arg;

int \*flag;

int val;

} opt,

char \*opt\_descr

}

\*/

{

{

OPT\_size,

required\_argument,

0, 0,

},

"Find file with size smaller than input size"

},

};

static int g\_po\_arr\_len = sizeof(g\_po\_arr)/sizeof(g\_po\_arr[0]);

int plugin\_get\_info(struct plugin\_info\* ppi) {

if (!ppi) {

fprintf(stderr, "ERROR: invalid argument\n");

return -1;

}

ppi->plugin\_purpose = g\_plugin\_purpose;

ppi->plugin\_author = g\_plugin\_author;

ppi->sup\_opts\_len = g\_po\_arr\_len;

ppi->sup\_opts = g\_po\_arr;

return 0;

}

static int input\_size = 0;

int plugin\_process\_file(const char \*fname,

struct option in\_opts[],

size\_t in\_opts\_len) {

int ret = -1;

char \*DEBUG = getenv("LAB1DEBUG");

if (!fname || !in\_opts || !in\_opts\_len) {

errno = EINVAL;

return -1;

}

g\_lib\_name="libtvhN3249-2.so";

int got\_input\_size = 0;

#define OPT\_CHECK(opt\_var) \

if (got\_##opt\_var) { \

if (DEBUG) { \

fprintf(stderr, "DEBUG: %s: Option '%s' was already supplied\n", \

g\_lib\_name, in\_opts[i].name); \

} \

errno = EINVAL; \

return -1; \

} \

else { \

char \*endptr = NULL; \

opt\_var = strtol((char\*)in\_opts[i].flag, &endptr, 10); \

if (strcmp(endptr,"")!=0) { \

if (DEBUG) { \

fprintf(stderr, "DEBUG: %s: Failed to convert '%s'\n", \

g\_lib\_name, (char\*)in\_opts[i].flag); \

} \

errno = EINVAL; \

return -1; \

} \

got\_##opt\_var = 1; \

}

for (size\_t i = 0; i < in\_opts\_len; i++) {

if (!strcmp(in\_opts[i].name, OPT\_size)) {

OPT\_CHECK(input\_size)

}

else {

errno = EINVAL;

return -1;

}

}

if (!got\_input\_size) {

if (DEBUG) {

fprintf(stderr, "DEBUG: %s: The input size value was not supplied.\n",

g\_lib\_name);

}

errno = EINVAL;

return -1;

}

FILE \*fp;

fp = fopen(fname, "rb");

if (fp == NULL) {

fprintf(stderr, "Error : Failed to open entry file - %s\n", strerror(errno));

return -1;

}

fseek(fp, 0L, SEEK\_END);

int res = ftell(fp);

fclose(fp);

if (res < input\_size)

return ret = 0;

else return ret = 1;

}

**6. lab1test.c**

// Test lab 1 .so files for formal conformance

//

// Compile with:

// gcc -o lab1test lab1test.c -ldl

//

// (c) Alexei Guirik, 2021

// This source is licensed under CC BY-NC 4.0

// (https://creativecommons.org/licenses/by-nc/4.0/)

//

#include <stdio.h>

#include <string.h>

#include <dlfcn.h>

#include "plugin\_api.h"

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

if (argc == 1) {

fprintf(stdout, "Usage: lab1test /path/to/lib1.so ... libN.so\n");

return 0;

}

// Try all names that are passed via argv

for (int i = 1; i < argc; i++) {

fprintf(stdout, "Trying %s:\n", argv[i]);

struct plugin\_info pi = {0};

void \*dl = dlopen(argv[i], RTLD\_LAZY);

if (!dl) {

fprintf(stderr, "ERROR: dlopen() failed: %s\n", dlerror());

continue;

}

// Check for plugin\_get\_info() func

void \*func = dlsym(dl, "plugin\_get\_info");

if (!func) {

fprintf(stderr, "ERROR: dlsym() failed: %s\n", dlerror());

goto END;

}

typedef int (\*pgi\_func\_t)(struct plugin\_info\*);

pgi\_func\_t pgi\_func = (pgi\_func\_t)func;

int ret = pgi\_func(&pi);

if (ret < 0) {

fprintf(stderr, "ERROR: plugin\_get\_info() failed\n");

goto END;

}

// Plugin info

fprintf(stdout, "Plugin purpose:\t\t%s\n", pi.plugin\_purpose);

fprintf(stdout, "Plugin author:\t\t%s\n", pi.plugin\_author);

fprintf(stdout, "Supported options: ");

if (pi.sup\_opts\_len > 0) {

fprintf(stdout, "\n");

for (size\_t i = 0; i < pi.sup\_opts\_len; i++) {

fprintf(stdout, "\t--%s\t\t%s\n", pi.sup\_opts[i].opt.name, pi.sup\_opts[i].opt\_descr);

}

}

else {

fprintf(stdout, "none (!?)\n");

}

// Warn if plugin\_process\_file() is not found

func = dlsym(dl, "plugin\_process\_file");

if (!func) {

fprintf(stderr, "WARNING: no plugin\_process\_file() function found\n");

}

END:

if (dl) dlclose(dl);

}

return 0;

}

**7. plugin\_api.h**

#ifndef \_PLUGIN\_API\_H

#define \_PLUGIN\_API\_H

#include <getopt.h>

/\*

Структура, описывающая опцию, поддерживаемую плагином.

\*/

struct plugin\_option {

/\* Опция в формате, поддерживаемом getopt\_long (man 3 getopt\_long). \*/

struct option opt;

/\* Описание опции, которое предоставляет плагин. \*/

const char \*opt\_descr;

};

/\*

Структура, содержащая информацию о плагине.

\*/

struct plugin\_info {

/\* Назначение плагина \*/

const char \*plugin\_purpose;

/\* Автор плагина, например "Иванов Иван Иванович, N32xx" \*/

const char \*plugin\_author;

/\* Длина списка опций \*/

size\_t sup\_opts\_len;

/\* Список опций, поддерживаемых плагином \*/

struct plugin\_option \*sup\_opts;

};

int plugin\_get\_info(struct plugin\_info\* ppi);

/\*

plugin\_get\_info()

Функция, позволяющая получить информацию о плагине.

Аргументы:

ppi - адрес структуры, которую заполняет информацией плагин.

Возвращаемое значение:

0 - в случае успеха,

< 0 - в случае неудачи (в этом случае продолжать работу с этим плагином нельзя).

\*/

int plugin\_process\_file(const char \*fname,

struct option in\_opts[],

size\_t in\_opts\_len);

/\*

plugin\_process\_file()

Фунция, позволяющая выяснить, отвечает ли файл заданным критериям.

Аргументы:

fname - путь к файлу (полный или относительный), который проверяется на

соответствие критериям, заданным с помощью массива in\_opts.

in\_opts - список опций (критериев поиска), которые передаются плагину.

struct option {

const char \*name;

int has\_arg;

int \*flag;

int val;

};

Поле name используется для передачи имени опции, поле flag - для передачи

значения опции (в виде строки). Если у опции есть аргумент, поле has\_arg

устанавливается в ненулевое значение. Поле val не используется.

in\_opts\_len - длина списка опций.

Возвращаемое значение:

0 - файл отвечает заданным критериям,

> 0 - файл НЕ отвечает заданным критериям,

< 0 - в процессе работы возникла ошибка

В случае, если произошла ошибка, переменная errno должна устанавливаться

в соответствующее значение.

\*/

#endif

**II. Примеры работы программы:**

1. LAB1DEBUG=1 ./lab1tvhN3249 --exe elf,pe32 ./file\_test

Text

Description automatically generated

1. LAB1DEBUG=1 ./lab1tvhN3249 --size 1000 ./file\_test

Text

Description automatically generated

1. LAB1DEBUG=1 ./lab1tvhN3249 -A --exe elf,pe32 --size 20000 ./file\_test

Text

Description automatically generated

1. LAB1DEBUG=1 ./lab1tvhN3249 -O --exe elf,pe32 --size 1000 ./file\_test

Text

Description automatically generated

1. LAB1DEBUG=1 ./lab1tvhN3249 -A -N --exe elf,pe32 –size 20000 ./file\_test

Text

Description automatically generated

1. LAB1DEBUG=1 ./lab1tvhN3249 -P ./file\_test --lines-count 8 lines-count-comp ne file\_test

A screenshot of a computer

Description automatically generated with medium confidence