// Файл monitoring.c

#define pr\_fmt(fmt) "ftrace\_hook: " fmt

#include <linux/ftrace.h>

#include <linux/kallsyms.h>

#include <linux/kernel.h>

#include <linux/linkage.h>

#include <linux/module.h>

#include <linux/slab.h>

#include <linux/uaccess.h>

#include <linux/version.h>

MODULE\_DESCRIPTION("Module hooking clone() and execve() via ftrace");

MODULE\_AUTHOR("Levushkin Ilya");

MODULE\_LICENSE("GPL");

#define USE\_FENTRY\_OFFSET 0

struct ftrace\_hook {

const char \*name;

void \*function;

void \*original;

unsigned long address;

struct ftrace\_ops ops;

};

static int fh\_resolve\_hook\_address(struct ftrace\_hook \*hook)

{

hook->address = kallsyms\_lookup\_name(hook->name);

if (!hook->address) {

pr\_debug("unresolved symbol: %s\n", hook->name);

return -ENOENT;

}

#if USE\_FENTRY\_OFFSET

\*((unsigned long\*) hook->original) = hook->address + MCOUNT\_INSN\_SIZE;

#else

\*((unsigned long\*) hook->original) = hook->address;

#endif

return 0;

}

static void notrace fh\_ftrace\_thunk(unsigned long ip, unsigned long parent\_ip,

struct ftrace\_ops \*ops, struct pt\_regs \*regs)

{

struct ftrace\_hook \*hook = container\_of(ops, struct ftrace\_hook, ops);

#if USE\_FENTRY\_OFFSET

regs->ip = (unsigned long) hook->function;

#else

if (!within\_module(parent\_ip, THIS\_MODULE))

regs->ip = (unsigned long) hook->function;

#endif

}

int fh\_install\_hook(struct ftrace\_hook \*hook)

{

int err;

err = fh\_resolve\_hook\_address(hook);

if (err)

return err;

hook->ops.func = fh\_ftrace\_thunk;

hook->ops.flags = FTRACE\_OPS\_FL\_SAVE\_REGS

| FTRACE\_OPS\_FL\_RECURSION\_SAFE

| FTRACE\_OPS\_FL\_IPMODIFY;

err = ftrace\_set\_filter\_ip(&hook->ops, hook->address, 0, 0);

if (err) {

pr\_debug("ftrace\_set\_filter\_ip() failed: %d\n", err);

return err;

}

err = register\_ftrace\_function(&hook->ops);

if (err) {

pr\_debug("register\_ftrace\_function() failed: %d\n", err);

ftrace\_set\_filter\_ip(&hook->ops, hook->address, 1, 0);

return err;

}

return 0;

}

void fh\_remove\_hook(struct ftrace\_hook \*hook)

{

int err;

err = unregister\_ftrace\_function(&hook->ops);

if (err) {

pr\_debug("unregister\_ftrace\_function() failed: %d\n", err);

}

err = ftrace\_set\_filter\_ip(&hook->ops, hook->address, 1, 0);

if (err) {

pr\_debug("ftrace\_set\_filter\_ip() failed: %d\n", err);

}

}

int remove\_hooks(struct ftrace\_hook \*hooks, size\_t i, int err)

{

while (i != 0) {

fh\_remove\_hook(&hooks[--i]);

}

return err;

}

int fh\_install\_hooks(struct ftrace\_hook \*hooks, size\_t count)

{

int err;

size\_t i;

for (i = 0; i < count; i++) {

err = fh\_install\_hook(&hooks[i]);

if (err)

return remove\_hooks(hooks, i, err);

}

return 0;

}

void fh\_remove\_hooks(struct ftrace\_hook \*hooks, size\_t count)

{

size\_t i;

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

fh\_remove\_hook(&hooks[i]);

}

#ifndef CONFIG\_X86\_64

#error Currently only x86\_64 architecture is supported

#endif

#if defined(CONFIG\_X86\_64) && (LINUX\_VERSION\_CODE >= KERNEL\_VERSION(4,17,0))

#define PTREGS\_SYSCALL\_STUBS 1

#endif

#if !USE\_FENTRY\_OFFSET

#pragma GCC optimize("-fno-optimize-sibling-calls")

#endif

#ifdef PTREGS\_SYSCALL\_STUBS

static asmlinkage long (\*real\_sys\_clone)(struct pt\_regs \*regs);

static asmlinkage long fh\_sys\_clone(struct pt\_regs \*regs)

{

long ret;

pr\_info("clone() before\n");

ret = real\_sys\_clone(regs);

pr\_info("clone() after: %ld\n", ret);

return ret;

}

#else

static asmlinkage long (\*real\_sys\_clone)(unsigned long clone\_flags,

unsigned long newsp, int \_\_user \*parent\_tidptr,

int \_\_user \*child\_tidptr, unsigned long tls);

static asmlinkage long fh\_sys\_clone(unsigned long clone\_flags,

unsigned long newsp, int \_\_user \*parent\_tidptr,

int \_\_user \*child\_tidptr, unsigned long tls)

{

long ret;

pr\_info("clone() before\n");

ret = real\_sys\_clone(clone\_flags, newsp, parent\_tidptr,

child\_tidptr, tls);

pr\_info("clone() after: %ld\n", ret);

return ret;

}

#endif

static char \*duplicate\_filename(const char \_\_user \*filename)

{

char \*kernel\_filename;

kernel\_filename = kmalloc(4096, GFP\_KERNEL);

if (!kernel\_filename)

return NULL;

if (strncpy\_from\_user(kernel\_filename, filename, 4096) < 0) {

kfree(kernel\_filename);

return NULL;

}

return kernel\_filename;

}

#ifdef PTREGS\_SYSCALL\_STUBS

static asmlinkage long (\*real\_sys\_execve)(struct pt\_regs \*regs);

static asmlinkage long fh\_sys\_execve(struct pt\_regs \*regs)

{

long ret;

char \*kernel\_filename;

kernel\_filename = duplicate\_filename((void\*) regs->di);

pr\_info("execve() before: %s\n", kernel\_filename);

kfree(kernel\_filename);

ret = real\_sys\_execve(regs);

pr\_info("execve() after: %ld\n", ret);

return ret;

}

#else

static asmlinkage long (\*real\_sys\_execve)(const char \_\_user \*filename,

const char \_\_user \*const \_\_user \*argv,

const char \_\_user \*const \_\_user \*envp);

static asmlinkage long fh\_sys\_execve(const char \_\_user \*filename,

const char \_\_user \*const \_\_user \*argv,

const char \_\_user \*const \_\_user \*envp)

{

long ret;

char \*kernel\_filename;

kernel\_filename = duplicate\_filename(filename);

pr\_info("execve() before: %s\n", kernel\_filename);

kfree(kernel\_filename);

ret = real\_sys\_execve(filename, argv, envp);

pr\_info("execve() after: %ld\n", ret);

return ret;

}

#endif

#ifdef PTREGS\_SYSCALL\_STUBS

#define SYSCALL\_NAME(name) ("\_\_x64\_" name)

#else

#define SYSCALL\_NAME(name) (name)

#endif

#define HOOK(\_name, \_function, \_original) \

{ \

.name = SYSCALL\_NAME(\_name), \

.function = (\_function), \

.original = (\_original), \

}

static struct ftrace\_hook demo\_hooks[] = {

HOOK("sys\_clone", fh\_sys\_clone, &real\_sys\_clone),

};

static int fh\_init(void)

{

int err;

err = fh\_install\_hooks(demo\_hooks, ARRAY\_SIZE(demo\_hooks));

if (err)

return err;

pr\_info("module loaded\n");

return 0;

}

static void fh\_exit(void)

{

fh\_remove\_hooks(demo\_hooks, ARRAY\_SIZE(demo\_hooks));

pr\_info("module unloaded\n");

}

module\_init(fh\_init);

module\_exit(fh\_exit);

// Makefile

CURRENT = $(shell uname -r)

KDIR = /lib/modules/$(CURRENT)/build

PWD = $(shell pwd)

DEST = /lib/modules/$(CURRENT)/misc

TARGET = monitoring

obj-m := $(TARGET).o

default:

$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:

@rm -f \*.o .\*.cmd .\*.flags \*.mod.c \*.order

@rm -f .\*.\*.cmd \*.symvers \*~ \*.\*~ TODO.\*

@rm -fR .tmp\*

@rm -rf .tmp\_versions