UNIVERSITÉ LIBRE DE BRUXELLES

ELEC-Y404- OPERATING SYSTEMS AND SECURITY

Mini-project : KeyLogger

Name :
ANASS DENGUIR

Professor :
MARTIN TIMMERMAN

January 31, 2018

Abstract

"Keylogger" : by Anass DENGUIR
Université Libre de Bruxelles, 2017-2018.

People store more and more sensitive data on computers and servers. Although computers are more convenient to work with, they are also more likely to be spied on by malicious viruses. Indeed, in this project a Keylogger virus has been developed on Windows 10 using Python programming language. The latter registers each pressed character on the target's keyboard. An algorithm to extract the user′s passwords and other sensitive data from this logs has also been implemented. Afterwards, the spied data have been discretely sent to the attacker through a Slack channel. In order to induce the victim to download the Keylogger, the latter has been hidden with a Snake game program, thus forming a Trojan horse. Moreover, the virus has been designed in order to be registered in the Windows Registry. It is thus executed at every starting of the operating system. Finally, a detection analysis of the Keylogger on 65 different anti-viruses has lead to a detection ratio less than 13%.
Key words : Keylogger, virus, spy, Trojan, password, Windows 10, detection

Contents

1 Introduction

Nowadays people use more and more computers and other digital devices. A lot of information about them are stored or shared through the Internet. Due to this hyper-connectivity, security issues have arisen, endangering the private property of everyone. Indeed, a lot of malicious softwares have been developed since the last 20 years such as ransomwares or spywares. These kind of viruses are designed to steal sensitive information about the victim, which are used to blackmail him. In this project, a Keylogger virus will be implemented in order to show how easy it is to spy on someone's computer. Moreover the latter will be tested on a virtual machine to observe its effects. The Keylogger will finally be analyzed by a few anti-viruses to verify its detectability.

2 Design Plan

The design of the Keylogger will be done in 5 steps as illustrated in Figure 1. At first, the implementation of an off-line Keylogger will be done. Its duty is only to manage to catch Windows events such as keystrokes. Afterwards, the data will be filtered using a password detection algorithm. The filtered data will be logged on a Slack server to effectively steal the data. Once these steps finished, an executable file of the designed Keylogger will be built and a Trojan virus will be built from this executable file. Finally, detection analyzes will be applied on the Trojan Keylogger. The analyzes results will be fed back to improve the detectability of the Keylogger.

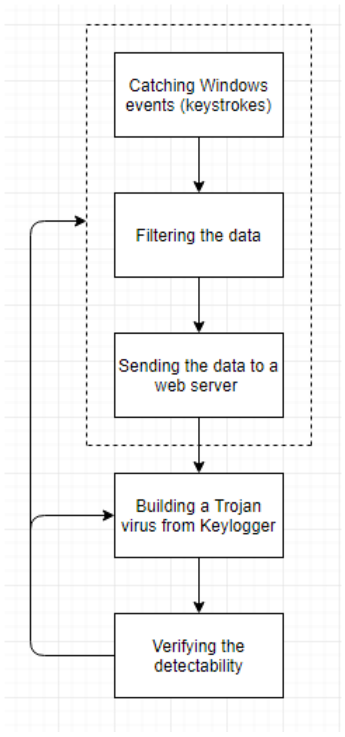

Catching Windows events (keystrokes)
Filtering the data
Sending the data to a web server
Building a Trojan virus from Keylogger
Verifying the detectability

Figure 1: Design Plan

3 Keylogger implementation

3.1 Code Architecture

In order to implement the Keylogger, a code architecture has been designed in Python programming language. The architecture design consists of 2 types of entities communicating together:

- The Keyboard Listener, which will detect keyboard events
- The Event Handler, which will act on a keyboard event

To achieve its goal, the Keyboard Listener mainly performs two tasks. In the first place, it has to constantly listen to Windows events. This is done using the Win32 package of Python which notifies every global input Windows events with the Win32 *PumpMessages* method. However, in order to detect a specific event, a *HookManager* is needed. Such an object, provided by the *pyHook* library, handles a myriad of events (including mouse and keyboard events). In this case, the HookManager has been used to hook the target's keyboard. This means that the program will callback a specific function when the desired event occurs. As it happens, an event triggered function called *on_keyboard* has been bound to the *KeyDown* event. Thus, whenever the user types on his keyboard, the *on_keyboard* function is called back and executed. The role of this function is simple: encode the keyboard events using Ascii lookup table and store the result in a FIFO (First In First Out) buffer. The whole process of the Keyboard Listener is summarized in the left part of Figure 2.

At the same time, the Event Handler constantly checks the state of the buffer. If the buffer is full, the data is extracted and analyzed by an algorithm which determines whether or not the typed data are worth to be spied on. In the negative, the data is discarded, otherwise the data is logged on a server (see Section 5). The algorithm which evaluates the relevance of the data will be explained in details in Section 4. The global process of the Event Handler is depicted in the right side of Figure 2.

3.2 Timing issues

Both the Keyboard Listener and the Event Handler operate simultaneously using Threads. At first glance, one could find unnecessary to use Threads as the tasks of the Event Handler could theoretically be executed just after a keystroke by the triggered *on_keyboard* function. Nevertheless, the HookManager needs this callback function to be as fast as possible, otherwise the event could be discarded. That is why the only job of *on_keyboard* is to add the detected keyboard events into a queue. In parallel, the Event Handler constantly checks this buffer and applies successively the filtering and the logging of the data. The use of a queue as a buffer ensures that the Event Handler will handle each keystroke in the right order, even if it is operating concurrently.

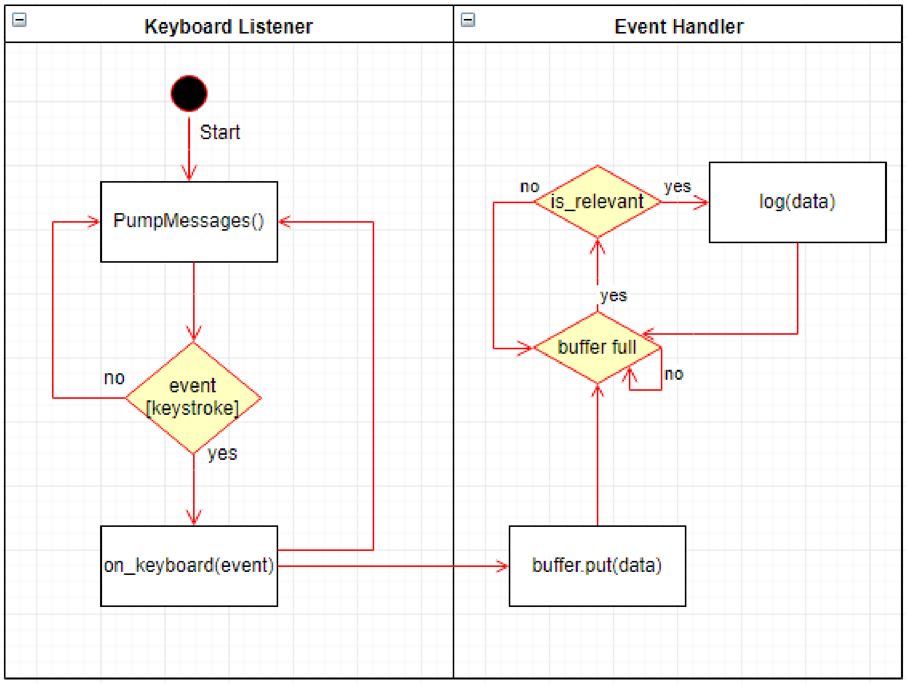


Figure 2: Keylogger Process Overview

4 Extracting passwords

4.1 Algorithm

Once the data are collected in the buffer, the Event Handler has to decide whether or not the data will be logged. This step of the process is important to prevent logging superfluous keystrokes on the internet. This decreases the probability of the virus to be detected either by the Firewall or by the target himself. A malicious keylogger is often designed to spy relevant data about the target like passwords, private conversations, etc. The algorithm implemented in this project easily extracts this kind of data from the logs. It consists of defining a filter which is actually a list of key words and it verifies if at least one of the defined key words appears in the currently opened window name of the target. Typically these key words will be a list of application names the hacker wants to spy on (ex: Facebook, Gmail, etc). Concretely, the algorithm operates as follows:

1. Define a list of application names, namely the filter
2. Convert each key word of the filter in uppercase (or lowercase)
3. At each keystroke event, store the window name where the keystroke occurred, in uppercase (or lowercase)
4. Compare each key word of the filter with the registered window name
5. If at least one key word is in the window name, log the data (with the corresponding window name)

Let's illustrate the implemented detection algorithm on a Python piece of code:

```python
def is_relevant_data(event):
    # define the filter:
    filter = ['FACEBOOK', 'GMAIL'] # in uppercase
    # get the Window name associated with the keystroke:
    keystroke = event.Ascii
    uppercase_window_name = event.WindowName.upper() # in uppercase
    # look if any words of the filter appears in the window name:
    for key_word in filter:
        if key_word in uppercase_window_name:
            return True
    return False
```

4.2 Weaknesses of the algorithm

This simple algorithm is useful to filter relevant data but does not guarantee that the password of the target will be extracted. Indeed, the identifier and the password are sometimes registered and pre-typed by the web browser. This avoids the user to always type his identifiers to log on a website. Thus the Keylogger will not be able to catch the target's password. However, this practice is not recommended as such information can easily be stolen by other means. Another way to forestall the Keylogger is to enable the 2FA (2 Factor Authentication) on websites where sensitive data (or money) are stored. This is a second level of account security that only authorizes a user to connect to a given website if the latter knows the account's password AND an extra 6-digit code that is updated in the user's cellphone about every 30 seconds.

5 Sending the logs

Once the logs are filtered, these are ready to be sent to a Slack server. Afterwards, the attacker will be able to freely access the data of the victim. To do so, a Slack channel has been created and a webhook URL has been bound to this channel. This URL contains an API secret key which allows a bot to post messages on the linked channel without proceeding to authentication. Thereby, the Keylogger will post the data stored in the buffer and then throw it. Data are posted to the server using HTTP POST requests. This is done using the *requests* library of Python. To perform such a request, one needs the URL of the server. Next, the data must be encoded using the *json* format (see [2]). Finally the data can be uploaded using a POST request to the server. As the target will continuously post data in the server, an HTTP Session has been created between the client and the server. It allows to reuse a same persistent TCP connection to transfer data. This avoids to constantly recreate an HTTP connection when data have to be sent. Hence the connection speed of the target will not be overly altered and the probability to be detected will be lower. This connection type, also known as Keep-alive connection, is illustrated in Figure 3.

The format of each log is the following: *Date + Hour :: Window Name :: Text*. This allows to record all the data within the context these were typed, that is to say the exact time and the window name the target was consulting when typing these keystrokes. A sample of the Slack Server records is shown in Figure 4. One can see that the data are uploaded to the server when the length of the

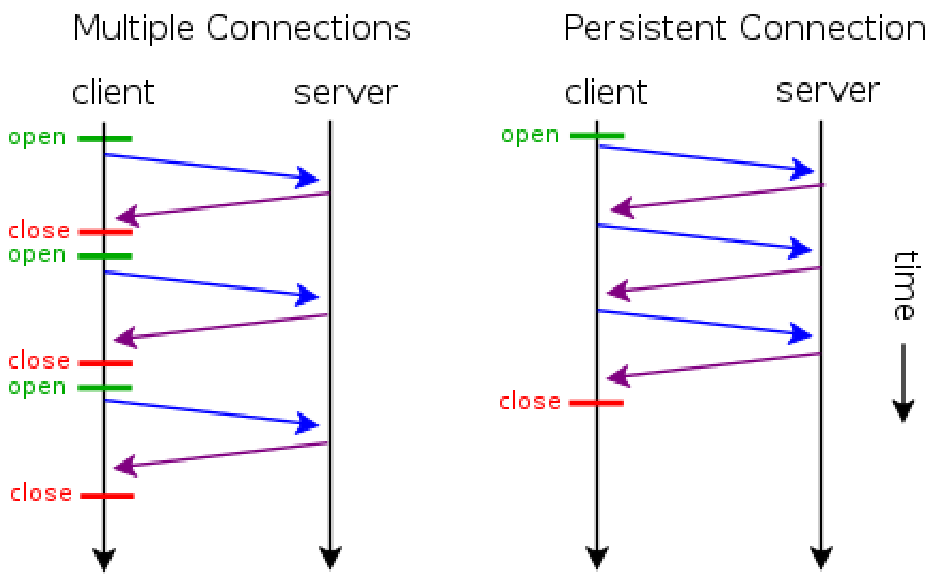
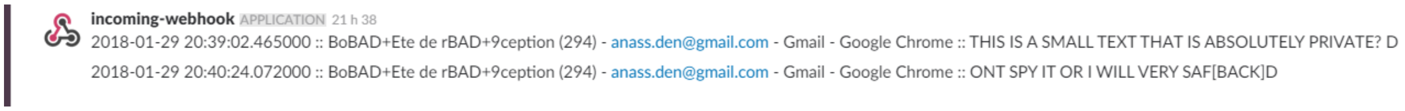


Figure 3: Multiple Connections vs Persistent Connection

buffer reaches 50 (typically 50 keystrokes). However, as shown on line 2 of Figure 4, whenever the user corrects a typo, the inscription *[BACK]* appears to underline the typo. In the same way, every time the user hints the [Return] button, the latter clearly appears in the logs. This is done in order to ease the analysis of the logs.



```
incoming-webhook APPLICATION 21 h 38
2018-01-29 20:39:02.465000 :: BoBAD+Ete de rBAD+9ception (294) - anass.den@gmail.com - Gmail - Google Chrome :: THIS IS A SMALL TEXT THAT IS ABSOLUTELY PRIVATE? D
2018-01-29 20:40:24.072000 :: BoBAD+Ete de rBAD+9ception (294) - anass.den@gmail.com - Gmail - Google Chrome :: ONT SPY IT OR I WILL VERY SAF[BACK]D
```

Figure 4: Slack Logging Server

6 Persistence in Registry

To make the virus more malicious, one can program the Keylogger to run as soon as the operating system is started. This consists of adding the executable file root of the virus in the Windows Registry. It is a hierarchical database storing all low-level settings of the Windows operating system but it can also be used to store settings of applications. Each application can subscribe to this database by defining a subkey and a corresponding value. Indeed the Windows Registry contains basic keys, referred to as root keys, and each subkey is a branch of one of these root keys. The different root keys are summarized in the Table below.

Key name	Abbreviation name
HKEY_LOCAL_MACHINE	*HKLM*
HKEY_CURRENT_CONFIG	*HKCC*
HKEY_CLASSES_ROOT	*HKCR*
HKEY_CURRENT_USER	*HKCU*
HKEY_USER	*HKU*

Table 1: Root Keys of Windows Registry

The Keylogger has been registered in the *HKCU* root key, which manages the user's applications that are run at the starting of the computer. This root is defined by the following subkey:
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

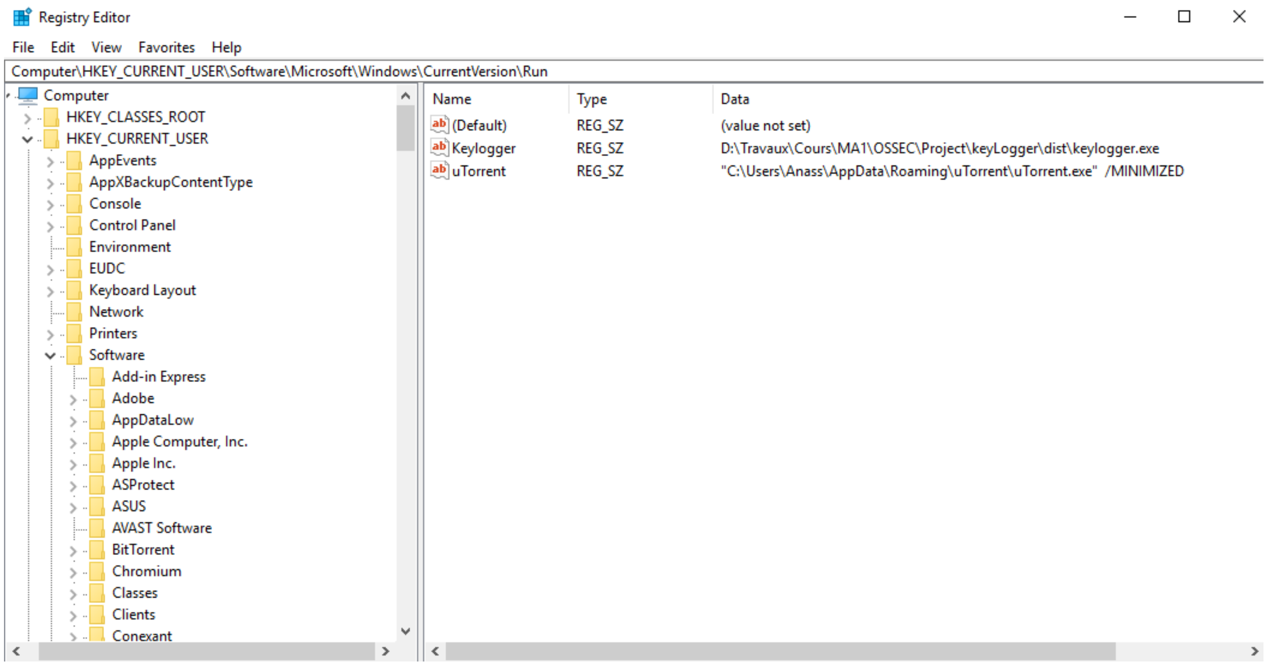
To open the specified register path, an *OpenKey* built-in function has been used from Python *_winreg* library. This method allows to open the subkey registry path with a defined access right. Afterwards, the corresponding value of the database has been set using the *SetValueEx* function. It attributes a specific path to the executable file to be accessed at the starting of the computer, the name of the application and the encoding form of the data. The different steps are illustrated in the piece of code below:

```
 1  def persist():
 2        '''Adds the executable file at startup Windows app'''
 3        # lines 4 and 5 evaluate the absolute path of the executable file:
 4        directory = os.path.dirname(os.path.realpath(__file__))
 5        path_to_exe = directory + '\\' + sys.argv[0].split('\\')[-1]
 6        # open the subkey with full access rights
 7        sub_key = 'Software\Microsoft\Windows\CurrentVersion\Run'
 8        with OpenKey(HKEY_CURRENT_USER, sub_key, 0, KEY_ALL_ACCESS) as key:
 9              # attribute the defined path to the registry value
10              SetValueEx(key, "Keylogger", 0, REG_SZ, path_to_exe)
```

Where:

- The selected path attributed to the value at registration, namely *path_to_exe*, is the absolute path of the executable file
- *REG_SZ* is the encoding type of the specified value. It corresponds to a null-terminated string

With this functionality, it simply requires the victim to execute the virus once to allow the Keylogger to run on its own at each startup of Windows. However, the application registry can be removed by accessing the Registry Editor as shown on Figure 5.



Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

```
Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
    AppEvents
    AppXBackupContentType
    Console
    Control Panel
    Environment
    EUDC
    Keyboard Layout
    Network
    Printers
    Software
      Add-in Express
      Adobe
      AppDataLow
      Apple Computer, Inc.
      Apple Inc.
      ASProtect
      ASUS
      AVAST Software
      BitTorrent
      Chromium
      Classes
      Clients
      Conexant
```

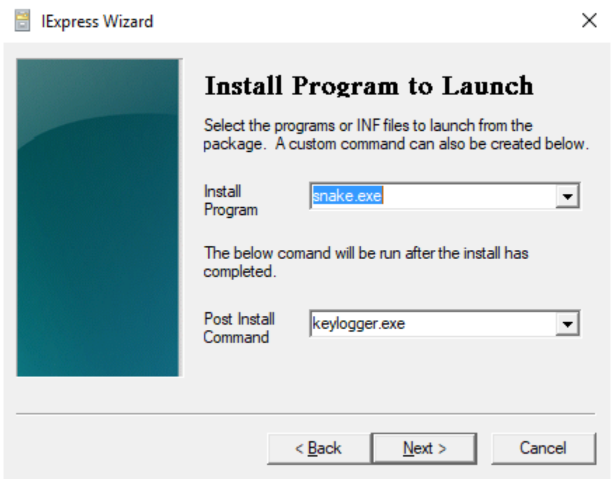
Name	Type	Data
(Default)	REG_SZ	(value not set)
Keylogger	REG_SZ	D:\Travaux\Cours\MA1\OSSEC\Project\keyLogger\dist\keylogger.exe
uTorrent	REG_SZ	"C:\Users\Anass\AppData\Roaming\uTorrent\uTorrent.exe" /MINIMIZED

Figure 5: Windows Registry Editor

7 Making a Trojan virus

Malicious viruses have to dupe the target in order to be downloaded. For sure, one will not intentionally download a program knowing that it is a virus. That is why these are often hidden behind a useful application, making them attractive. Such programs are known as Trojan viruses as the malicious software is wrapped into gift paper. In practice, a Trojan virus can be realized by merging the executable file of the Keylogger with the one of another program which is the seeming main application. This can be easily done using IExpress Windows tool which allows to merge two *.exe* files into one single executable file. This procedure is illustrated in Figure 6. In this project, the Keylogger has been bound to a Snake game to get a *snake_logger.exe* file (see Figure 7). Thereby, the target thinks he has downloaded a game but in fact, as soon as the user will exit the game, the Keylogger will be executed right after (and at each startup if the persistence is activated).

If the target decides to replay the Snake game, another instance of the Keylogger will be launched. This is an unwanted effect as redundant data will be logged on the Slack channel, leading to an overloading of the CPU and more connection lags in the victim side. These two factors involve a higher detection probability of the virus. In order to be as stealthy as possible, the Keylogger has to ensure that only one instance of the program is running at a given time. That is why, at each start of the program, a lock file with a filename based on the full path to the script file is generated. This lock file allows only one single instance of the considered script file. In practice, this is ensured with the following line of code, using Python *tendo* library:



IExpress Wizard

Install Program to Launch

Select the programs or INF files to launch from the package. A custom command can also be created below.

Install Program: snake.exe

The below comand will be run after the install has completed.

Post Install Command: keylogger.exe

Figure 6: IExpress Wizard

```
instance = singleton.SingleInstance()
```

This single line of code is put at the beginning of the script. It will check if another instance of the program is running. In this case, it will quit the program with a *-1 code*. Otherwise, it will only return *True*.

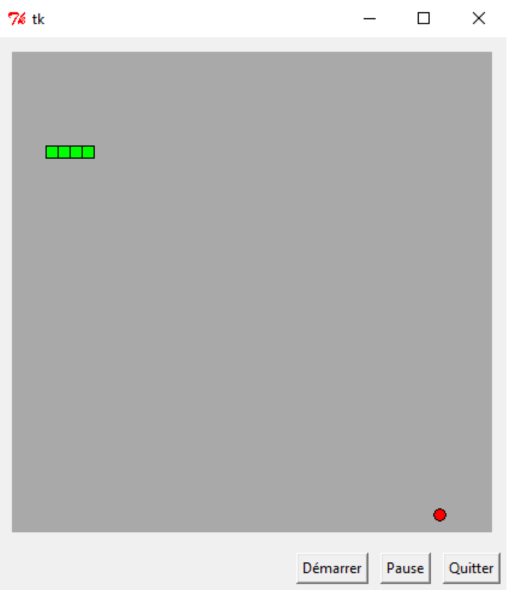
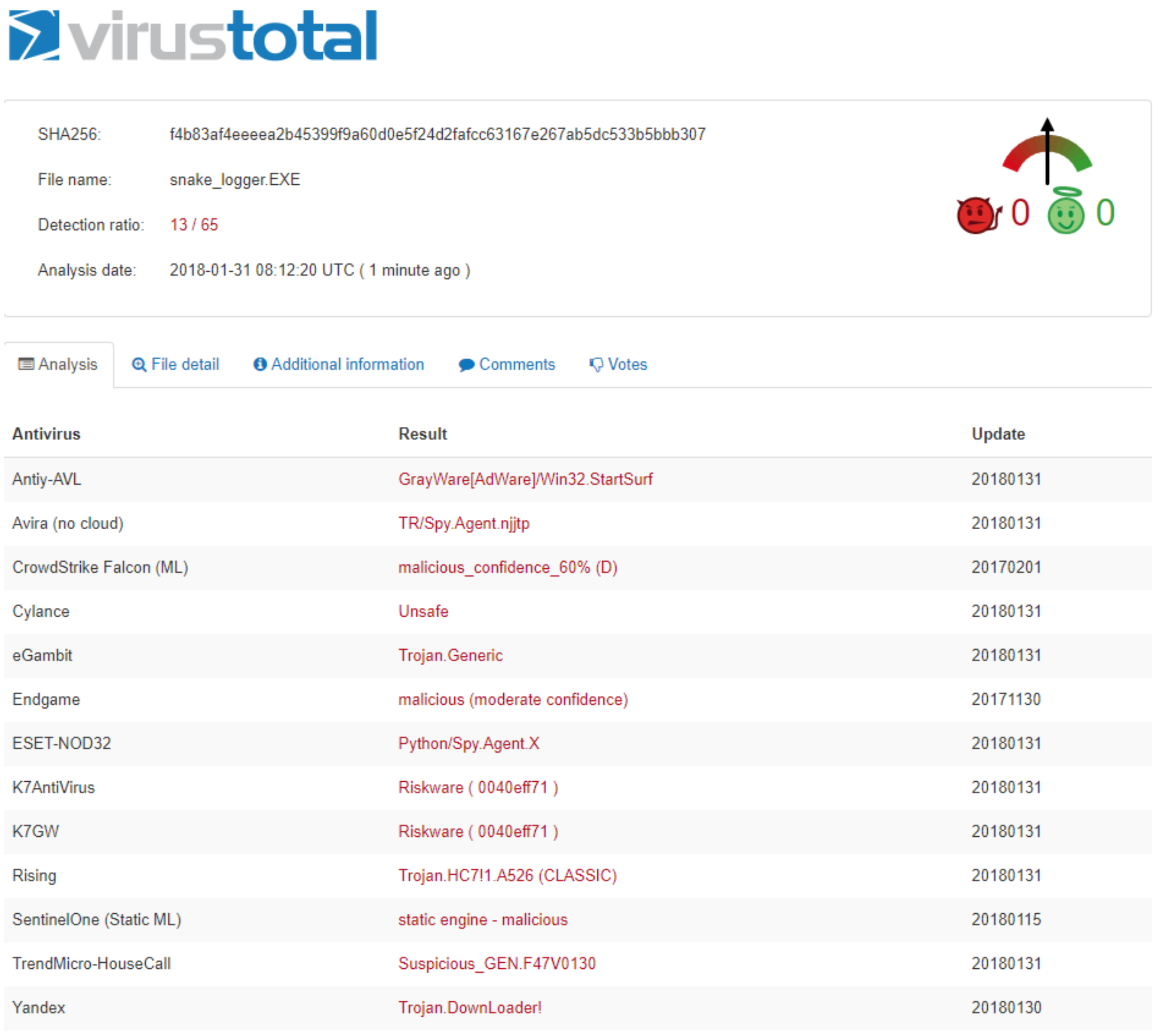


Figure 7: Trojan virus: *snake_logger.exe*

8 Detectability

In order to check the detectability of the Trojan virus built in Section 7, the latter has been tested on 65 different anti-viruses provided by the *VirusTotal* website (see [1]). The analysis results are shown in Figure 8. It appears that 13 anti-viruses detected the Trojan among the 65, which corresponds to a detection rate of 20%. However, these results are not totally consistent if these are not compared with the detection rate of the uncorrupted executable file, namely the *snake.exe* file. A similar analysis of the clean executable file *snake.exe* is reported in Figure 9. One can see that the detection rate is approximately the same as the Trojan one, knowing that this is not a virus. Moreover, it appears that the two programs have been detected by 8 anti-viruses in common. One can conclude that the detection criteria of these anti-viruses are quite strict.

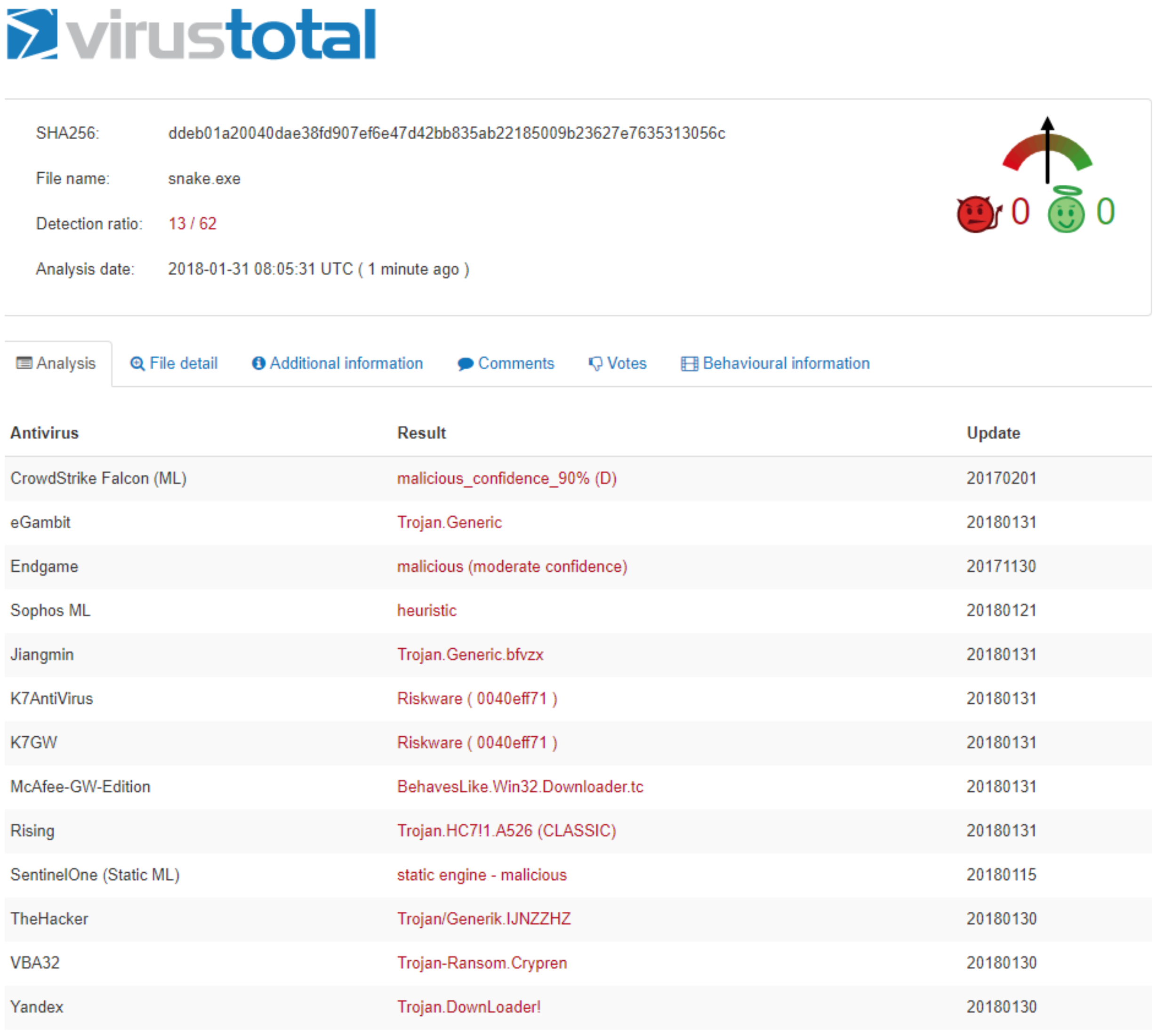
To provide a more reliable detection rate, one has to analyze a Trojan Keylogger hidden inside a perfectly clean executable file. This has been done using the executable file of a well known application which aims to teach traffic rules, namely *feuvert.exe*. As shown on Figure 10, this file is perfectly clean. After having merged this file with the designed Keylogger, the *VirusTotal* website returned a detection rate of about 13% for the *feuvert_logger.exe* file, which is better than the previous result (see Figure 11). Finally, it appears that the Keylogger is undetected by the most popular anti-viruses like Avast, Microsoft, Kaspersky or McAfee. This lowers further the detection ratio of the virus.



SHA256:	f4b83af4eeeea2b45399f9a60d0e5f24d2fafcc63167e267ab5dc533b5bbb307
File name:	snake_logger.EXE
Detection ratio:	13 / 65
Analysis date:	2018-01-31 08:12:20 UTC (1 minute ago)

Antivirus	Result	Update
Antiy-AVL	GrayWare[AdWare]/Win32.StartSurf	20180131
Avira (no cloud)	TR/Spy.Agent.njjtp	20180131
CrowdStrike Falcon (ML)	malicious_confidence_60% (D)	20170201
Cylance	Unsafe	20180131
eGambit	Trojan.Generic	20180131
Endgame	malicious (moderate confidence)	20171130
ESET-NOD32	Python/Spy.Agent.X	20180131
K7AntiVirus	Riskware (0040eff71)	20180131
K7GW	Riskware (0040eff71)	20180131
Rising	Trojan.HC7!1.A526 (CLASSIC)	20180131
SentinelOne (Static ML)	static engine - malicious	20180115
TrendMicro-HouseCall	Suspicious_GEN.F47V0130	20180131
Yandex	Trojan.DownLoader!	20180130

Figure 8: Detection result of Trojan *snake_logger.exe*



SHA256: ddeb01a20040dae38fd907ef6e47d42bb835ab22185009b23627e7635313056c

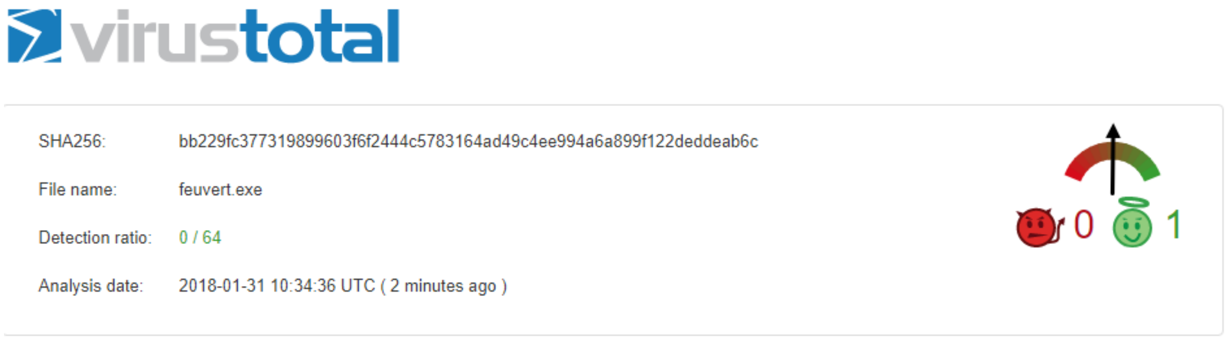
File name: snake.exe

Detection ratio: 13 / 62

Analysis date: 2018-01-31 08:05:31 UTC (1 minute ago)

Antivirus	Result	Update
CrowdStrike Falcon (ML)	malicious_confidence_90% (D)	20170201
eGambit	Trojan.Generic	20180131
Endgame	malicious (moderate confidence)	20171130
Sophos ML	heuristic	20180121
Jiangmin	Trojan.Generic.bfvzx	20180131
K7AntiVirus	Riskware (0040eff71)	20180131
K7GW	Riskware (0040eff71)	20180131
McAfee-GW-Edition	BehavesLike.Win32.Downloader.tc	20180131
Rising	Trojan.HC7!1.A526 (CLASSIC)	20180131
SentinelOne (Static ML)	static engine - malicious	20180115
TheHacker	Trojan/Generik.IJNZZHZ	20180130
VBA32	Trojan-Ransom.Crypren	20180130
Yandex	Trojan.DownLoader!	20180130

Figure 9: Detection result of *snake.exe*



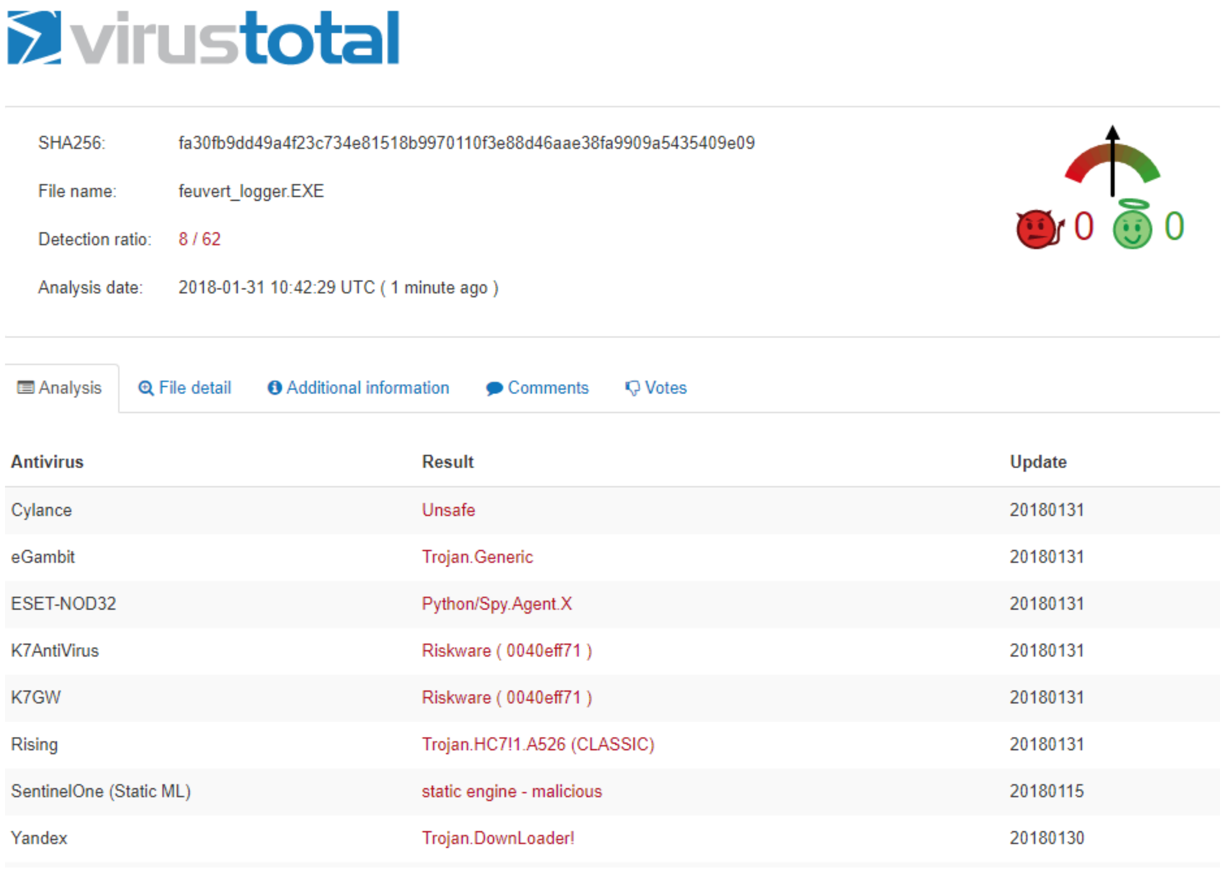
SHA256: bb229fc377319899603f6f2444c5783164ad49c4ee994a6a899f122deddeab6c

File name: feuvert.exe

Detection ratio: 0 / 64

Analysis date: 2018-01-31 10:34:36 UTC (2 minutes ago)

Figure 10: Detection result of *feuvert.exe*



SHA256: fa30fb9dd49a4f23c734e81518b9970110f3e88d46aae38fa9909a5435409e09

File name: feuvert_logger.EXE

Detection ratio: 8 / 62

Analysis date: 2018-01-31 10:42:29 UTC (1 minute ago)

Antivirus	Result	Update
Cylance	Unsafe	20180131
eGambit	Trojan.Generic	20180131
ESET-NOD32	Python/Spy.Agent.X	20180131
K7AntiVirus	Riskware (0040eff71)	20180131
K7GW	Riskware (0040eff71)	20180131
Rising	Trojan.HC7!1.A526 (CLASSIC)	20180131
SentinelOne (Static ML)	static engine - malicious	20180115
Yandex	Trojan.DownLoader!	20180130

Figure 11: Detection result of *feuvert_logger.exe*

9 Conclusion

In conclusion, a Trojan horse has been designed. The latter hides a Keylogger inside what seems to be a simple game. The virus logs in a Slack channel every data that are considered as relevant according to a specific algorithm. Finally, the detection analyzes show that the detectability ratio of the implemented virus itself goes under 13%.

References

[1] Virustotal. Found at https://www.virustotal.com/en/.

[2] Douglas Crockford. Jsonrequest. Found at http://www.json.org/JSONRequest.html.

[3] Python documentation. Windows registry access. Found at https://docs.python.org/2/library/_winreg.html.

[4] Ilya Grigorik. High performance browser networking. Found at https://hpbn.co/http1x Ilya Grigorik is a web performance engineer at Google and co-chair of the W3C Web Performance Working Group. Follow him on his blog and Twitter for the latest web performance news, tips, and talks.

[5] Wikipedia. Windows registry. Found at https://en.wikipedia.org/wiki/Windows_Registry.