Woody RAT: A new feature-rich malware spotted in the wild

Threat Intelligence Team ⋮⋮ 8/3/2022

This blog post was authored by Ankur Saini and Hossein Jazi

The Malwarebytes Threat Intelligence team has identified a new Remote Access Trojan we are calling Woody Rat that has been in the wild for at least one year.

This advanced custom Rat is mainly the work of a threat actor that targets Russian entities by using lures in archive file format and more recently Office documents leveraging the Follina vulnerability.

Based on a fake domain registered by the threat actors, we know that they tried to target a Russian aerospace and defense entity known as OAK.

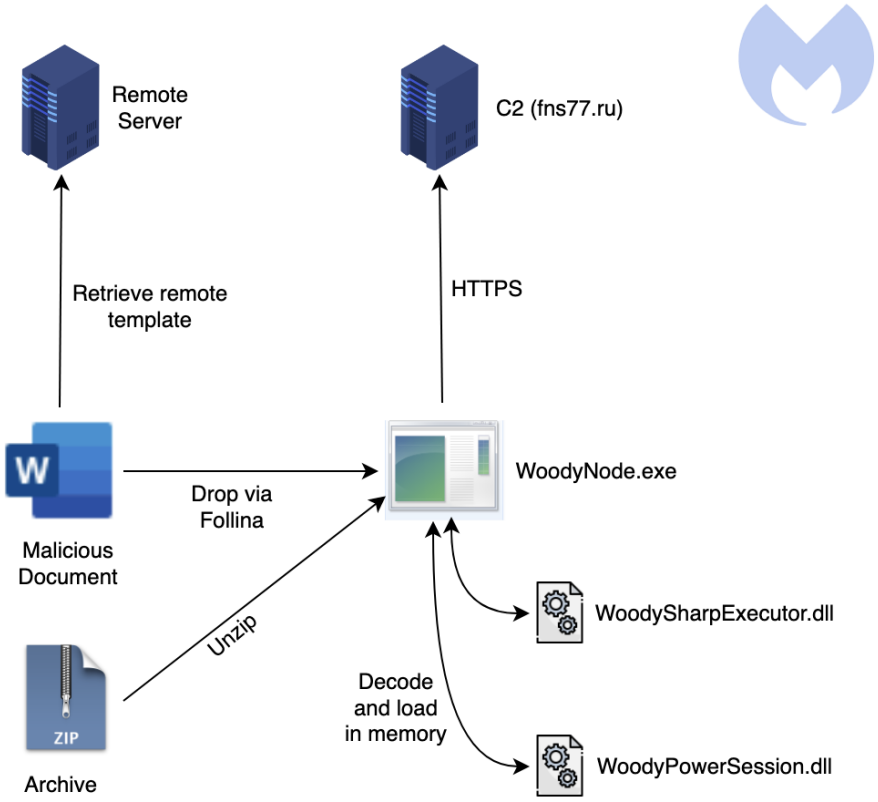
In this blog post, we will analyze Woody Rat's distribution methods, capabilities as well as communication protocol.

Distribution methods

Based on our knowledge, Woody Rat has been distributed using two different formats: archive files and Office documents using the Follina vulnerability.

The earliest versions of this Rat was typically archived into a zip file pretending to be a document specific to a Russian group. When the Follina vulnerability became known to the world, the threat actor switched to it to distribute the payload, as identified by @MalwareHunterTeam.

The following diagram shows the overall attack flow used by the threat actor to drop Woody Rat:



Woody Rat distribution methods

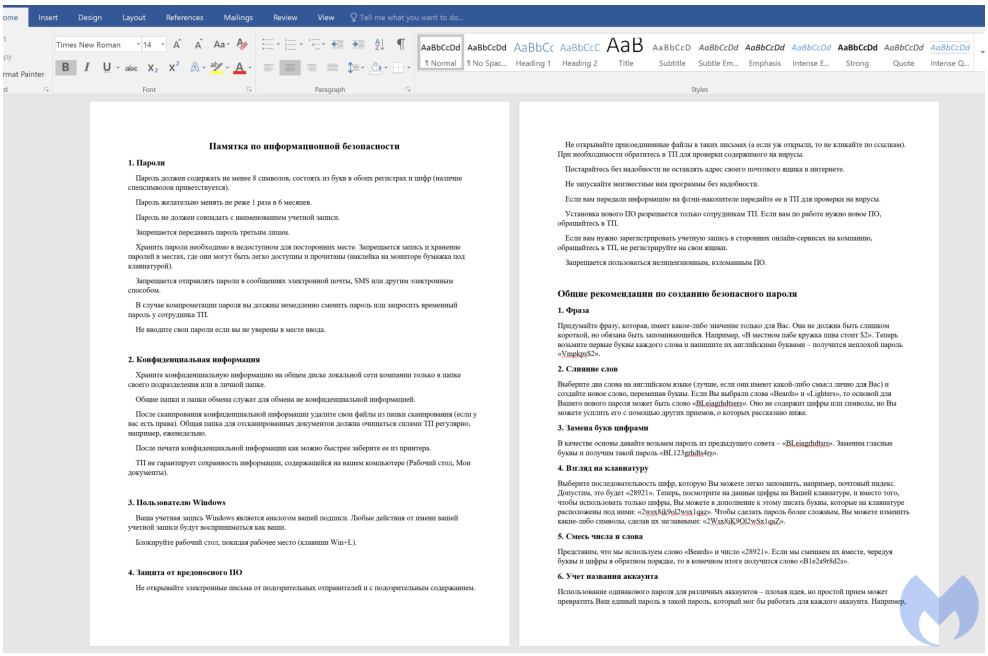
Archive files

In this method, Woody Rat is packaged into an archive file and sent to victims. We believe that these archive files have been distributed using spear phishing emails. Here are some examples of these archive files:

- *anketa_brozhik.doc.zip*: It contains Woody Rat with the same name: *Anketa_Brozhik.doc.exe*.
- *zayavka.zip*: It contains Woody Rat pretending to be an application (application for participation in the *selection.doc.exe*).

Follina vulnerability

The threat actor is using a Microsoft Office document (*Памятка.docx*) that has weaponized with the Follina (CVE-2022-30190) vulnerability to drop Woody Rat. The used lure is in Russian is called "*Information security memo*" which provide security practices for passwords, confidential information, etc.



Памятка по информационной безопасности

1. Пароли

Пароль должен содержать не менее 8 символов, состоять из букв в обоих регистрах и цифр (наличие спецсимволов приветствуется).

Пароль желательно менять не реже 1 раза в 6 месяцев.

Пароль не должен совпадать с именованием учетной записи.

Запрещается передавать пароль третьим лицам.

Хранить пароли необходимо в недоступном для посторонних месте. Запрещается запись и хранение паролей в местах, где они могут быть легко доступны и прочитаны (наклейка на мониторе бумажка под клавиатурой).

Запрещается отправлять пароли в сообщениях электронной почты, SMS или другим электронным способом.

В случае компрометации пароля вы должны немедленно сменить пароль или запросить временный пароль у сотрудника ТП.

Не вводите свои пароли если вы не уверены в месте ввода.

2. Конфиденциальная информация

Храните конфиденциальную информацию на общем диске локальной сети компании только в папке своего подразделения или в личной папке.

Общие папки и папки обмена служат для обмена не конфиденциальной информацией.

После сканирования конфиденциальной информации удалите свои файлы из папки сканирования (если у вас есть права). Общая папка для отсканированных документов должна очищаться силами ТП регулярно, например, еженедельно.

После печати конфиденциальной информации как можно быстрее заберите ее из принтера.

ТП не гарантирует сохранность информации, содержащейся на вашем компьютере (Рабочий стол, Мои документы).

3. Пользователю Windows

Ваша учетная запись Windows является аналогом вашей подписи. Любые действия от имени вашей учетной записи будут восприниматься как ваши.

Блокируйте рабочий стол, покидая рабочее место (клавиши Win+L).

4. Защита от вредоносного ПО

Не открывайте электронные письма от подозрительных отправителей и с подозрительным содержанием.

Не открывайте присоединенные файлы в таких письмах (а если уж открыли, то не кликайте по ссылкам). При необходимости обратитесь в ТП для проверки содержимого на вирусы.

Постарайтесь без надобности не оставлять адрес своего почтового ящика в интернете.

Не запускайте неизвестные вам программы без надобности.

Если вам передали информацию на флэш-накопителе передайте ее в ТП для проверки на вирусы.

Установка нового ПО разрешается только сотрудникам ТП. Если вам по работе нужно новое ПО, обращайтесь в ТП.

Если вам нужно зарегистрировать учетную запись в сторонних онлайн-сервисах на компанию, обращайтесь в ТП, не регистрируйте на свои ящики.

Запрещается пользоваться нелицензионным, взломанным ПО.

Общие рекомендации по созданию безопасного пароля

1. Фраза

Придумайте фразу, которая, имеет какое-либо значение только для Вас. Она не должна быть слишком короткой, но обязана быть запоминающейся. Например, «В местном пабе кружка пива стоит $2». Теперь возьмите первые буквы каждого слова и напишите их английскими буквами – получится неплохой пароль «Vimpkps$2».

2. Слияние слов

Выберите два слова на английском языке (лучше, если они имеют какой-либо смысл лично для Вас) и создайте новое слово, перемешав буквы. Если Вы выбрали слова «Beards» и «Lighters», то основой для Вашего нового пароля может быть слово «BLeiagrhdtsers». Оно не содержит цифры или символы, но Вы можете усилить его с помощью других приемов, о которых рассказано ниже.

3. Замена букв цифрами

В качестве основы давайте возьмем пароль из предыдущего совета – «BLeiagrhdtsrs». Заменим гласные буквы и получим такой пароль «BL123ghdts4rs».

4. Взгляд на клавиатуру

Выберите последовательность цифр, которую Вы можете легко запомнить, например, почтовый индекс. Допустим, это будет «28921». Теперь, посмотрите на данные цифры на Вашей клавиатуре, и вместо того, чтобы использовать только цифры, Вы можете в дополнение к этому писать буквы, которые на клавиатуре расположены под ними: «2wsxSiKl9oI2wsx1qaz». Чтобы сделать пароль более сложным, Вы можете изменить какие-либо символы, сделав их заглавными: «2WsxSiK9OI2wSx1qaZ».

5. Смесь числа и слова

Представим, что мы используем слово «Beards» и число «28921». Если мы смешаем их вместе, чередуя буквы и цифры в обратном порядке, то в конечном итоге получится слово «B1e2a9r8d2s».

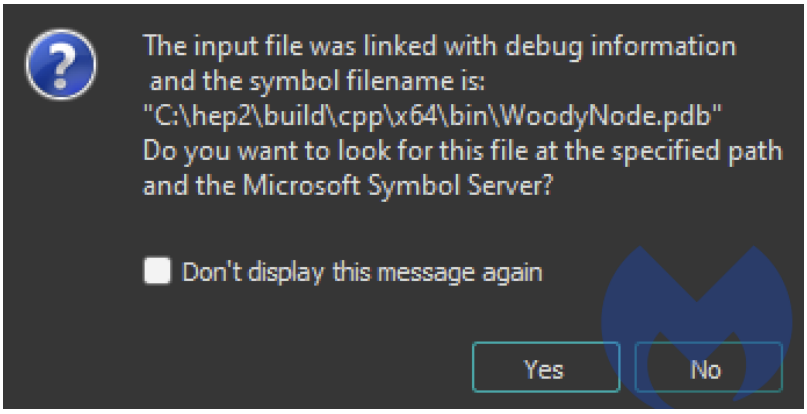
6. Учет названия аккаунта

Использование одинакового пароля для различных аккаунтов – плохая идея, но простой прием может превратить Ваш единый пароль в такой пароль, который мог бы работать для каждого аккаунта. Например,

Document lure

Woody Rat Analysis

The threat actor has left some debugging information including a pdb path from which we derived and picked a name for this new Rat:



Debug Information

A lot of CRT functions seem to be statically linked, which leads to IDA generating a lot of noise and hindering analysis. Before initialization, the malware effectively suppresses all error reporting by calling SetErrorMode with 0x8007 as parameter.

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    SetErrorMode(0x8007u);
6    woody_main(v3, v4);
7    return 0;
8  }
```
main function

As we will see later, that malware uses multiple threads and so it allocates a global object and assigns a mutex to it to make sure no two clashing operations can take place at the same time. This object enforces that only one thread is reaching out to the C2 at a given time and that there are no pending requests before making another request.

Deriving the Cookie

The malware communicates with its C2 using HTTP requests. To uniquely identify each infected machine, the malware derives a cookie from machine specific values. The values are taken from the adapter information, computer name and volume information, and 8 random bytes are appended to this value to avoid any possible cookie collisions by the malware.

A combination of *GetAdaptersInfo*, *GetComputerNameA* and *GetVolumeInformationW* functions are used to retrieve the required data to generate the cookie. This cookie is sent with every HTTP request that is made to the C2.

```
1  __int64 __fastcall get_cookie_data(__int64 a1)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    v16 = a1;
6    *a1 = 0i64;
7    *a1 = 0i64;
8    *(a1 + 8) = 0i64;
9    *(a1 + 16) = 0i64;
10   v15 = 1;
11   v18 = a1;
12   derive_AdapterInfo(v13, &v14);
13   nSize = 1024;
14   GetComputerNameA(CompName, &nSize);
15   v3 = 0;
16   for ( i = 0i64; i < nSize; i += 4i64 )
17   {
18     v2 = CompName[i] | ((CompName[i + 1] | ((CompName[i + 2] | (CompName[i + 3] << 8)) << 8)) << 8);
19     v3 ^= v2;
20   }
21   pos_lib_func(&v18, v2, 4ui64, v3);
22   _RAX = 0i64;
23   __asm { cpuid }
24   v19 = _RAX;
25   v20 = _RBX;
26   v21 = _RCX;
27   v22 = _RDX;
28   pos_lib_func(&v18, _RDX, 2ui64, (WORD1(_RDX) + _RDX + WORD1(_RCX) + _RCX + WORD1(_RBX) + _RBX + WORD1(_RAX) + _RAX));
29   nSize = 0;
30   GetVolumeInformationW(L"c:\\", 0i64, 0, &nSize, 0i64, 0i64, 0i64, 0);
31   pos_lib_func(&v18, (nSize + HIWORD(nSize)), 2ui64, (nSize + HIWORD(nSize)));
32   pos_lib_func(&v18, v10, 2ui64, v13[0]);
33   pos_lib_func(&v18, v11, 2ui64, v14);
34   return a1;
35 }
```
get_cookie_data function

Data encryption with HTTP requests

To evade network-based monitoring the malware uses a combination of RSA-4096 and AES-CBC to encrypt the data sent to the C2. The public key used for RSA-4096 is embedded inside the binary and the malware formulates the

RSA public key blob at runtime using the embedded data and imports it using the *BCryptImportKeyPair* function.

The malware derives the key for AES-CBC at runtime by generating 32 random bytes; these 32 bytes are then encrypted with RSA-4096 and sent to the C2. Both the malware and C2 simultaneously use these bytes to generate the AES-CBC key using *BCryptGenerateSymmetricKey* which is used in subsequent HTTP requests to encrypt and decrypt the data. For encryption and decryption the malware uses *BCryptEncrypt* and *BCryptDecrypt* respectively.

```
int __thiscall Encrypt_RSA(const void **this, int a2, PUCHAR a3, ULONG cbInput, int a5)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v18 = 1;
  *a2 = 0i64;
  *(a2 + 8) = 0;
  phAlgorithm = 0;
  phKey = 0;
  pcbResult = 0;
  pPaddingInfo[0] = L"SHA1";
  pPaddingInfo[1] = 0;
  pPaddingInfo[2] = 0;
  *a2 = 0;
  *(a2 + 4) = 0;
  *(a2 + 8) = 0;
  if ( BCryptOpenAlgorithmProvider(&phAlgorithm, L"RSA", 0, 0) >= 0 )
  {
    v6 = this[3];
    v7 = *this;
    pbInput[5] = 16;
    *&pbInput[6] = 0;
    *&pbInput[12] = 512;
    *&pbInput[8] = 3;
    strcpy(pbInput, "RSA1");
    *&pbInput[16] = 0;
    *&pbInput[20] = 0;
    *&pbInput[24] = *v6;
    pbInput[26] = v6[2];
    qmemcpy(&pbInput[27], v7, 0x200u);          // Copy the RSA key blob
    if ( BCryptImportKeyPair(phAlgorithm, 0, L"RSAPUBLICBLOB", &phKey, pbInput, 0x21Bu, 8u) >= 0
      && BCryptEncrypt(phKey, a3, cbInput, pPaddingInfo, 0, 0, 0, 0, &pcbResult, 4u) >= 0 )
    {
      v12 = pcbResult;
      ProcessHeap = GetProcessHeap();
      v9 = HeapAlloc(ProcessHeap, 0, v12);
      if ( v9 )
      {
        if ( BCryptEncrypt(phKey, a3, cbInput, pPaddingInfo, 0, 0, v9, pcbResult, &pcbResult, 4u) >= 0 )
          sub_40F360(v9, pcbResult);
        v10 = GetProcessHeap();
        HeapFree(v10, 0, v9);
      }
    }
  }
  if ( phKey )
    BCryptDestroyKey(phKey);
  if ( phAlgorithm )
    BCryptCloseAlgorithmProvider(phAlgorithm, 0);
  if ( cbInput || a3 )
    j_j__free(a3);
```

RSA Encryption routine

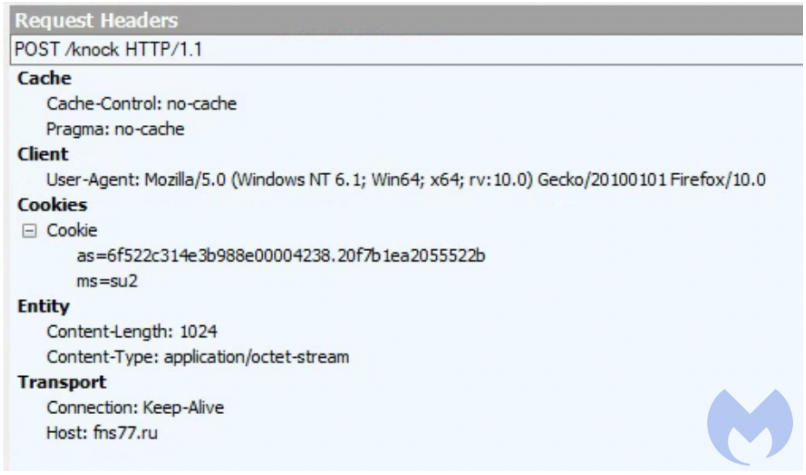
```
  ProcessHeap = GetProcessHeap();
  v10 = HeapAlloc(ProcessHeap, 0, cbOutput);
  sub_3BE40(v10, *a3, a3[1]);
  if ( BCryptOpenAlgorithmProvider(&phAlgorithm, L"AES", 0i64, 0) < 0
    || BCryptGetProperty(phAlgorithm, L"ObjectLength", pbOutput, 4u, &pcbResult, 0) < 0 )
  {
    v13 = 0i64;
  }
  else
  {
    v11 = *pbOutput;
    v12 = GetProcessHeap();
    v13 = HeapAlloc(v12, 0, v11);
    if ( v13 )
    {
      if ( BCryptSetProperty(phAlgorithm, L"ChainingMode", L"ChainingModeCBC", 0x20u, 0) >= 0
        && BCryptGenerateSymmetricKey(phAlgorithm, &phKey, v13, *pbOutput, *(a1 + 40), *(a1 + 48), 0) >= 0 )
      {
        v14 = GetProcessHeap();
        v15 = HeapAlloc(v14, 0, 0x10ui64);
        v6 = v15;
        if ( v15 )
        {
          *v15 = *a1;
          if ( BCryptEncrypt(phKey, v10, v7, 0i64, v15, 0x10u, v10, cbOutput, &pcbResult, a4 != 0) >= 0 )
            sub_E900(a2, v10, pcbResult);
          v16 = a2[1];
          if ( v16 >= 0x10 )
          {
            *a1 = **memset_0(a2, Val, (v16 - 16));
            if ( v24 || *Val )
              j_j_free(*Val);
          }
        }
      }
    }
  }
```

AES Encryption Routine

C2 HTTP endpoint request

knock – This is the first HTTP request that the malware makes to the C2. The machine-specific cookie is sent as part of the headers here. This is a POST request and the data of this request contains 32 random bytes which are used to derive AES-CBC key, while the 32 bytes are RSA-4096 encrypted.

The data received as response for this request is decrypted and it contains the url path to submit (/submit) the additional machine information which the malware generates after this operation.



knock request headers

submit – This endpoint request is used to submit information about the infected machine. The data sent to the C2 is AES-CBC encrypted. Data sent via submit API includes:

- OS
- Architecture
- Antivirus installed
- Computer Name
- OS Build Version
- .NET information
- PowerShell information
- Python information (Install path, version etc.)
- Storage drives – includes Drive path, Internal name etc.
- Environment Variables
- Network Interfaces
- Administrator privileges
- List of running processes
- Proxy information
- Username
- List of all the User accounts

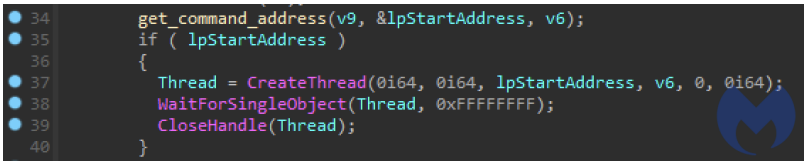
The malware currently detects 6 AVs through Registry Keys; these AVs being Avast Software, Doctor Web, Kaspersky, AVG, ESET and Sophos.

ping – The malware makes a ping GET http request to the C2 at regular intervals. If the C2 responds with "_CRY" then the malware proceeds to send the knock request again but if the C2 responds with "_ACK" the response contains additional information about which command should be executed by the malware.

The malware supports a wide variety of commands which are classified into _SET and _REQ requests as seen while analyzing the malware. We will dive into all these commands below in the blog.

C2 Commands

The malware uses a specific thread to communicate with the C2 and a different one to execute the commands received from the C2. To synchronize between both threads, the malware leverages events and mutex. To dispatch a command it modifies the state of the event linked to that object. We should note all the communications involved in these commands are AES encrypted.



Command execution routine

_SET Commands

- **PING** – This command is used to set the sleep interval between every ping request to the C2.
- **PURG** – Unknown command
- **EXIT** – Exit the command execution thread.

_REQ Commands

- **EXEC** (Execute)- Executes the command received from the C2 by creating a cmd.exe process, the malware creates two named pipes and redirects the input and output to these pipes. The output of the command is read using *ReadFile* from the named pipe and then "_DAT" is appended to this data before it is AES encrypted and sent to the C2.

```
143  PipeAttributes.nLength = 24;
144  PipeAttributes.lpSecurityDescriptor = 0i64;
145  PipeAttributes.bInheritHandle = 1;
146  if ( CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0) )
147  {
148    if ( CreatePipe(&hFile, &hObject, &PipeAttributes, 0) )
149    {
150      StartupInfo.cb = 104;
151      StartupInfo.dwFlags = 257;
152      StartupInfo.hStdError = hObject;
153      StartupInfo.hStdOutput = hObject;
154      StartupInfo.hStdInput = hReadPipe;
155      StartupInfo.wShowWindow = 0;
156      if ( CreateProcessW(0i64, CommandLine, 0i64, 0i64, 1, 0x8000000u, 0i64, 0i64, &StartupInfo, &ProcessInformation) )
157      {
158        CloseHandle(hObject);
159        CloseHandle(hReadPipe);
160        sub_3240(v52);
161        memset(Buffer, 0, sizeof(Buffer));
162        NumberOfBytesRead = 0;
163        while ( ReadFile(hFile, Buffer, 0x400u, &NumberOfBytesRead, 0i64) )
164        {
165          sub_BAD0(v52, Buffer);
166          memset(Buffer, 0, sizeof(Buffer));
167        }
168        CloseHandle(hFile);
169        CloseHandle(hWritePipe);
170        CryptAcquireContext_impl(v56);
171        set_null(v44);
172        unknown_libname_104(v44);
173        set_null_0(&v40);
174        _lambda_9a32fed5bf61b6b509b2d3f6003082a1_::_lambda_9a32fed5bf61b6b509b2d3f6003082a1_(&v40, v44);
175        v21 = append(v30, "_DAT");
176        sub_EDB0(&v40, v21);
```

EXEC command

- **UPLD** (Upload) – The Upload command is used to remotely upload a file to the infected machine. The malware makes a GET request to the C2 and receives data to be written as file.
- **INFO** (Submit Information) – The INFO command is similar to the "submit" request above; this command sends the exact information to the C2 as sent by the "submit" request.

```
1  void __fastcall __noreturn info_command(__int64 *a1)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    alloc_obj();
6    val = get_val(a1 + 4, 0i64);
7    v3 = sub_D990(val, v5);
8    multitowide(v7, v3);
9    v4 = some_lib_func(v6, v7);
10   submit_routine(v4);                    // get the info and submit encrypted data
11   if ( a1 )
12     free_0(a1);
13   ExitThread(0);
14 }
```

INFO command

- **UPEX** (Upload and Execute) – This is a combination of UPLD and EXEC command. The commands first writes a file received from the C2 and then executes that file.
- **DNLD** (Download) – The DNLD command allows the C2 to retrieve any file from the infected machine. The malware encrypts the requested file and sends the data via a POST request to the C2.
- **PROC** (Execute Process) – The PROC command is similar to the EXEC command with slight differences, here the process is directly executed instead of executing it with cmd.exe as in EXEC command. The command uses the named pipes in similar fashion as used by the EXEC command.
- **UPPR** (Upload and Execute Process) – This is a combination of UPLD and PROC command. The command receives the remote file using the upload command then executes the file using PROC command.
- **SDEL** (Delete File) – This is used to delete any file on the infected system. It also seems to overwrite the first few bytes of the file to be deleted with random data.
- **_DIR** (List directory) – This can list all the files and their attributes in a directory supplied as argument. If no directory is supplied, then it proceeds to list the current directory. File attributes retrieved by this command are:
  - Filename
  - Type (Directory, Unknown, File)
  - Owner
  - Creation time
  - Last access time
  - Last write time
  - Size
  - Permissions

- **STCK** (Command Stack) – This allows the attacker to execute multiple commands with one request. The malware can receive a STCK command which can have multiple children commands which are executed in the same order they are received by the malware.
- **SCRN** (Screenshot) – This command leverages Windows GDI+ to take the screenshot of the desktop. The image is then encrypted using AES-CBC and sent to the C2.
- **INJC** (Process Injection) – The malware seems to generate a new AES key for this command. The code to be injected is received from the C2 and decrypted. To inject the code into the target process it writes it to the remote memory using WriteProcessMemory and then creates a remote thread using CreateRemoteThread.

```
v10 = VirtualAllocEx(hProcess, 0i64, v3, 0x3000u, 0x40u);
v11 = v10;
if ( v10 )
{
  if ( WriteProcessMemory(hProcess, v10, lpBuffer, v9, 0i64) )
    return CreateRemoteThread(hProcess, 0i64, 0x100000ui64, &v11[v8], 2, 0, &ThreadId);
}
```

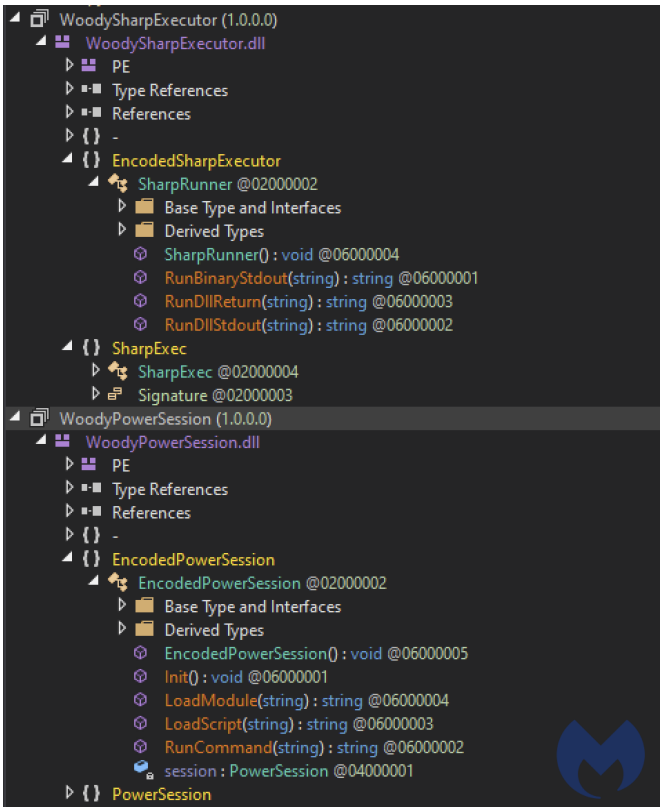
INJC routine

- **PSLS** (Process List) – Calls *NtQuerySystemInformation* with *SystemProcessInformation* to retrieve an array containing all the running processes. Information sent about each process to the C2:
  - PID
  - ParentPID
  - Image Name
  - Owner
- **DMON** (Creates Process) – The command seems similar to PROC with the only difference being the output of the process execution is not sent back to the C2. It receives the process name from the C2 and executes it using CreateProcess.
- **UPDM** (Upload and Create Process) – Allows the C2 and upload a file and then execute it using DMON command.

SharpExecutor and PowerSession Commands

Interestingly, the malware has 2 .NET DLLs embedded inside. These DLLs are named *WoodySharpExecutor* and *WoodyPowerSession* respectively. *WoodySharpExecutor* provides the malware ability to run .NET code received from the C2. *WoodyPowerSession* on the other hand allows the malware to execute PowerShell commands and scripts received from the C2.

WoodyPowerSession makes use of pipelines to execute these PS commands. The .NET dlls are loaded by the malware and commands are executed via the methods present in these DLLs:



SharpExecutor and PowerSession methods

We will look at the commands utilising these DLLs below:

- **DN_B** (DotNet Binary) – This command makes use of the RunBinaryStdout method to execute Assembly code with arguments received from the C2. The code is received as an array of Base64 strings separated by 0x20

character.

- **DN_D** (DotNet DLL) – This method provides the attacker a lot more control over the execution. An attacker can choose whether to send the console output back to the C2 or not. The method receives an array of Base64 strings consisting of code, class name, method name and arguments. The DLL loads the code and finds and executes the method based on other arguments received from the C2.
- **PSSC** (PowerSession Shell Command) – Allows the malware to receive a Base64 encoded PowerShell command and execute it.
- **PSSS** (PowerSession Shell Script) – This command allows the malware to load and execute a Base64 encoded PowerShell script received from the C2.
- **PSSM** (PowerSession Shell Module) – This command receives an array of Base64 encoded strings, one of which contains the module contents and the other one contains the module name. These strings are decoded and this module is imported to the command pipeline and then invoked.

Malware Cleanup

After creating the command threads, the malware deletes itself from disk. It uses the more commonly known *ProcessHollowing* technique to do so. It creates a suspended notepad process and then writes shellcode to delete a file into the suspended process using *NtWriteVirtualMemory*. The entry point of the thread is set by using the *NtSetContextThread* method and then the thread is resumed. This leads to the deletion of the malware from disk.

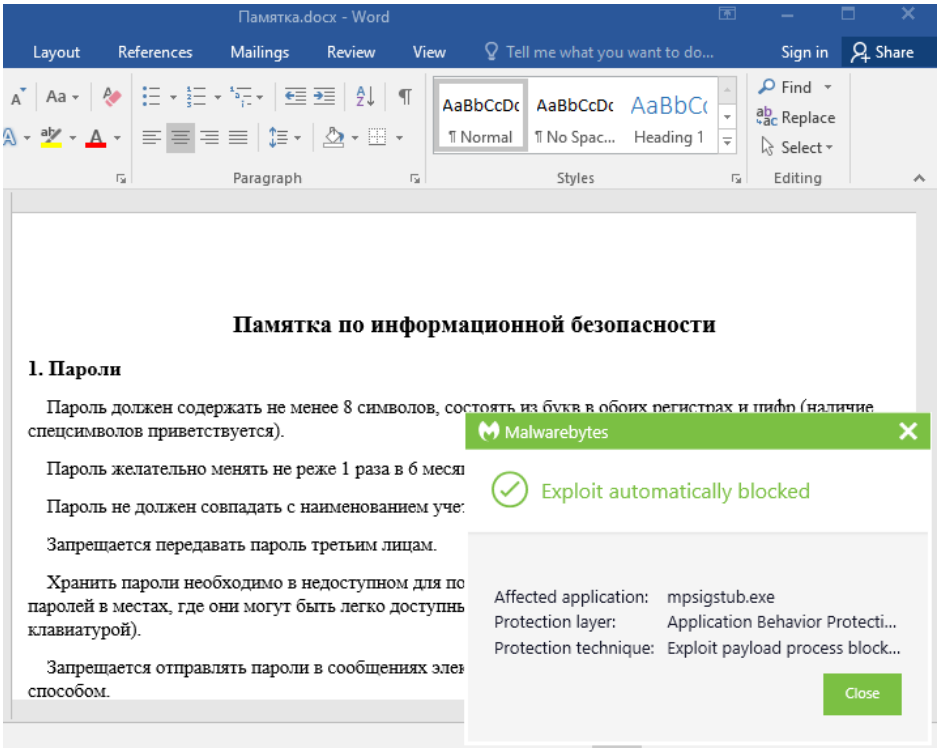
```
904     deletefile_sc[v101] = 0;
905     strcpy(CommandLine, "notepad");
906     Context.ContextFlags = 1048587;
907     memset(&StartupInfo, 0, sizeof(StartupInfo));
908     memset(&ProcessInformation, 0, sizeof(ProcessInformation));
909     CreateProcessA(0i64, CommandLine, 0i64, 0i64, 0, 4u, 0i64, 0i64, &StartupInfo, &ProcessInformation);
910     v102 = VirtualAllocEx(ProcessInformation.hProcess, 0i64, 0x3E8ui64, 0x3000u, 0x40u);
911     NtWriteVirtualMemory(ProcessInformation.hProcess, v102, deletefile_sc, 0x1DFui64, 0i64);
912     NtGetContextThread(ProcessInformation.hThread, &Context);
913     Context.Rcx = v102;
914     NtSetContextThread(ProcessInformation.hThread, &Context);
915     NtResumeThread(ProcessInformation.hThread, 0i64);
```

Malware deletes itself

Unknown threat actor

This very capable Rat falls into the category of unknown threat actors we track. Historically, Chinese APTs such as Tonto team as well as North Korea with Konni have targeted Russia. However, based on what we were able to collect, there weren't any solid indicators to attribute this campaign to a specific threat actor.

Malwarebytes blocks the Follina exploit that is being leveraged in the latest Woody Rat campaign. We also already detected the binary payloads via our heuristic malware engines.



IOCs

Woody Rat:

- 982ec24b5599373b65d7fec3b7b66e6afff4872847791cf3c5688f47bfcb8bf0

- 66378c18e9da070629a2dbbf39e5277e539e043b2b912cc3fed0209c48215d0b
- b65bc098b475996eaabbb02bb5fee19a18c6ff2eee0062353aff696356e73b7a
- 43b15071268f757027cf27dd94675fdd8e771cdcd77df6d2530cb8e218acc2ce
- 408f314b0a76a0d41c99db0cb957d10ea8367700c757b0160ea925d6d7b5dd8e
- 0588c52582aad248cf0c43aa44a33980e3485f0621dba30445d8da45bba4f834
- 5c5020ee0f7a5b78a6da74a3f58710cba62f727959f8ece795b0f47828e33e80
- 3ba32825177d7c2aac957ff1fc5e78b64279aeb748790bc90634e792541de8d3
- 9bc071fb6a1d9e72c50aec88b4317c3eb7c0f5ff5906b00aa00d9e720cbc828d

C2s:

- kurmakata.duckdns[.]org
- microsoft-ru-data[.]ru
- 194.36.189.179
- microsoft-telemetry[.]ru
- oakrussia[.]ru

Follina Doc:
Памятка.docx
ffa22c40ac69750b229654c54919a480b33bc41f68c128f5e3b5967d442728fb
Follina html file:
garmandesar.duckdns[.]org:444/uoqiuwef.html
Woody Rat url:
fcloud.nciinform[.]ru/main.css (edited)