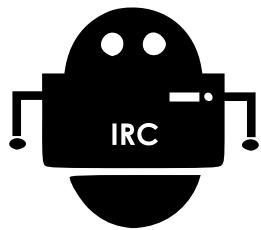


# Analisi di Sample.exe, un Malware IRC Bot

---

## Progetto Finale Malware

Xu Sunyi VR435956  
Crosara Marco VR434403



# Indice

<b>1 Introduzione</b>	<b>4</b>
1.1 Il controllo sulla data . . . . .	4
<b>2 Analisi Statica</b>	<b>6</b>
2.1 PEStudio . . . . .	6
2.1.1 UPX . . . . .	7
2.1.2 Analisi delle sezioni . . . . .	7
2.1.3 Analisi delle librerie e dei relativi imports . . . . .	7
2.1.4 Analisi delle stringhe . . . . .	9
2.1.5 Virus Total . . . . .	11
2.1.6 IRC Bot - Una valida ipotesi . . . . .	12
<b>3 Analisi Dinamica</b>	<b>13</b>
3.1 Ambiente di Testing . . . . .	13
3.2 Comportamento apparente di <i>sample.exe</i> . . . . .	13
3.3 Comportamento apparente di <i>sample_unp_mod.exe</i> . . . . .	13
3.4 Una nuova parentesi su PEStudio . . . . .	15
3.5 Comportamento apparente del file ignoto da 2kb . . . . .	18
3.6 ProcMon . . . . .	20
3.6.1 Librerie relative alle operazioni effettuate ( <i>sample.exe</i> ) . . . . .	20
3.6.2 Librerie relative alle operazioni effettuate ( <i>sample_unp_mod.exe</i> ) . . . . .	21
3.6.3 Operazioni sul File System ( <i>sample.exe</i> ) . . . . .	21
3.6.4 Operazioni sul File System ( <i>sample_unp_mod.exe</i> ) . . . . .	23
3.6.5 Operazioni sul Registro di Sistema ( <i>sample.exe</i> ) . . . . .	25
3.6.6 Operazioni sul Registro di Sistema ( <i>sample_unp_mod.exe</i> ) . . . . .	28
3.6.7 Operazioni di creazione altri Processi e Thread ( <i>sample.exe</i> ) . . . . .	33
3.6.8 Operazioni di creazione altri Processi e Thread ( <i>sample_unp_mod.exe</i> ) . . . . .	34
3.7 Analisi dinamica sul file da 2kb . . . . .	35
3.8 Script batch generati . . . . .	35
3.8.1 File I1726961194.bat . . . . .	36
3.8.2 File K1726961194.bat . . . . .	36
3.8.3 File U1726961194.bat . . . . .	36
3.9 RegShot . . . . .	37
3.9.1 Report per <i>sample.exe</i> . . . . .	37
3.9.2 Report per <i>sample_unp_mod.exe</i> . . . . .	38
3.10 Wireshark e FakeNet . . . . .	39
3.10.1 Analisi di <i>sample.exe</i> . . . . .	39
3.10.2 Analisi di <i>sample_unp_mod.exe</i> . . . . .	40
3.10.3 Protocollo IRC in un contesto malevolo . . . . .	40
3.10.4 Comandi IRC . . . . .	41
3.10.5 Pacchetti scambiati tra <i>sample_unp_mod.exe</i> e il server . . . . .	42
3.10.6 Tentativi di comunicazione con il malware . . . . .	43
3.10.7 Prime conclusioni . . . . .	44
3.11 L'incognita dei file da 2kb . . . . .	45
3.12 Comandiamo il bot IRC: proviamo alcuni comandi . . . . .	45
<b>4 Reverse Engineering</b>	<b>50</b>
4.1 Impostazioni per OllyDBG . . . . .	50
4.2 Funzioni TLS callback . . . . .	53
4.2.1 Funzioni TlsCallback_0, TlsCallback_1 . . . . .	53
4.2.2 Funzioni set_critical_section, manage_critical_section . . . . .	53
4.3 Inizio del programma . . . . .	54
4.3.1 Funzione main_bot . . . . .	54

4.4	Funzione di controllo sandbox . . . . .	57
4.4.1	Funzione <code>is_sandbox</code> . . . . .	57
4.5	Recupero delle informazioni di sistema e definizione di variabili . . . . .	58
4.5.1	Funzione <code>define_variables</code> . . . . .	58
4.6	Configurazione per il client IRC . . . . .	60
4.6.1	Funzione <code>set_up_info</code> . . . . .	60
4.6.2	Funzioni di codifica e decodifica Base64 . . . . .	60
4.6.3	Funzione di cifratura RC4 . . . . .	61
4.7	Funzioni di installazione e persistenza . . . . .	62
4.7.1	Funzione <code>create_thread_persist</code> . . . . .	62
4.7.2	Funzione <code>persist</code> . . . . .	62
4.7.3	Funzione <code>set_registry_keys</code> . . . . .	63
4.8	Creazione e gestione della copia del malware . . . . .	66
4.8.1	Funzione <code>handle_main_copy</code> . . . . .	66
4.8.2	Funzione <code>thread_create_copy</code> . . . . .	67
4.8.3	Funzione <code>create_copy</code> . . . . .	67
4.8.4	Funzione <code>check_backup</code> . . . . .	67
4.9	Funzioni di scansione di cartelle/registri ed eliminazione file sospetti . . . . .	68
4.9.1	Funzione <code>scan_directories</code> . . . . .	68
4.9.2	Funzione <code>search_for_programs</code> . . . . .	68
4.9.3	<code>kill_process</code> . . . . .	69
4.9.4	<code>break_file</code> . . . . .	69
4.9.5	<code>scan_for_registries</code> . . . . .	71
4.10	Funzioni relative a IRC . . . . .	72
4.10.1	Funzione <code>connect_to_irc</code> . . . . .	72
4.10.2	Funzione <code>send_to_irc</code> . . . . .	73
4.10.3	Funzione <code>generate_random_nick</code> . . . . .	73
4.10.4	Funzione <code>values_for_irc</code> . . . . .	73
4.10.5	Funzione <code>parse_irc_command</code> . . . . .	74
4.10.6	Funzione <code>exec_command</code> . . . . .	75
4.11	Funzioni di disininstallazione e rimozione . . . . .	75
4.11.1	Funzione <code>uninstall_program</code> . . . . .	75
4.11.2	Funzione <code>remove_keys_files</code> . . . . .	76
4.12	Miscellaneous . . . . .	77
4.12.1	Funzione <code>gen_rand_8_char</code> . . . . .	77
4.12.2	Funzione <code>generate_random_numb</code> . . . . .	77
4.12.3	Funzione <code>set irc client</code> . . . . .	78
4.13	Athena vs sample.exe . . . . .	78
4.13.1	Conclusioni . . . . .	79

# 1 Introduzione

Nel progetto il nostro obiettivo è stato quello di analizzare il malware *sample.exe* all'interno della cartella `C:\Users\Malware\Desktop\Samples\Reverse Engineering`. Ci è stato chiesto di capire il comportamento dell'eseguibile e di riportarne le parti più interessanti in un apposito report. Per le varie analisi è stata usata principalmente la stessa macchina virtuale utilizzata durante lo svolgimento del corso, anche se talvolta si è ricorso a soluzioni alternative, per eludere possibili sistemi che il malware potrebbe possedere per capire se viene o meno eseguito in una sandbox. Nelle sezioni descriveremo i vari software utilizzati per fare analisi e commenteremo i risultati ottenuti da questi ultimi.

## 1.1 Il controllo sulla data

La nostra ricerca si è svolta in modo particolare: inizialmente, dopo aver svolto una prima combinazione di analisi statica e dinamica per capire il comportamento generale del sample, risultava che il malware non producesse comportamenti malevoli rilevanti nelle macchine di test. Durante la fase di reverse engineering invece abbiamo scoperto che il comportamento anomalo del sample era dovuto ad un controllo sulla data corrente del sistema contro una certa “expiration date”, calcolata differendo di una decina di giorni da una “standard expiration date” che corrisponde al 15 Dicembre 2013 e salvata nel formato `yyymmdd`. Questa data può variare da un’ esecuzione all’ altra del malware. Siccome qualsiasi data prodotta per l’ “expiration date” è ovviamente trascorsa da molto tempo, il malware produce un comportamento “storpiato”, che ne impedisce dunque il funzionamento corretto. Pensiamo che questo controllo sia stato fissato perché l’autore non volesse che il malware funzionante circolasse per troppo tempo in rete e che volesse anche rendere l’ analisi più difficile in seguito per potenziali reversers. Per risolvere questo problema è possibile adottare tre diverse strategie:

- cambiare, tramite OllyDBG, il flusso di esecuzione del malware. Questo significa individuare, all’ interno dell’ assembly, tutti i punti in cui il controllo della data viene eseguito e cambiare i relativi jump. In Figura 1 viene mostrato un esempio relativo alla funzione `connect_to irc`. Cambiando JL (jump less) in JA (jump above) dopo il confronto delle date è possibile permettere al malware di continuare correttamente l’ esecuzione. Naturalmente questo risulta il metodo più lungo e tedioso.
- Cambiare direttamente la data della virtual machine e impostarla a una qualsiasi precedente a Dicembre 2013. Per fare ciò con VirtualBox da sistema Linux, bisogna prima usare il seguente comando da terminale `VBoxManage setextradata "Windows" "VBoxInternal/Devices/VMMDev/0/Config/GetHostTimeDisabled" 1`. Esso disabilita la sincronizzazione automatica della data del sistema host con quello della vm, permettendone quindi il cambiamento. Nonostante questa sia una soluzione molto semplice, durante i nostri test abbiamo notato che alcune volte, ripristinando la snapshot di una vm con una data modificata, essa ritornava a sincronizzarsi con quella del sistema host e bisognava modificarla manualmente di nuovo.
- Modificare la “standard expiration date” del malware tramite OllyDBG e produrre un nuovo eseguibile. Questa è stata la soluzione adottata perché ritenuta più veloce e affidabile. In Figura 2 si può notare che il valore esadecimale 2008F (131215 in decimale) è stato modificato in 498AF, che corrisponde alla data 31 Dicembre 2030. Le modifiche fatte all’ assembly si possono salvare nell’ eseguibile tramite OllyDBG facendo `tasto destro → Edit → Copy all modifications to executable` e poi salvando il file.

Durante l’ analisi statica utilizzeremo *sample.exe* o la sua versione unpacked *sample\_unp.exe*, quest’ ultima verrà usata anche nella parte di reverse engineering. Quando faremo analisi dinamica utilizzeremo invece *sample\_unp\_mod.exe* ovvero il malware modificato attuando la terza strategia.

The screenshot shows the OllyDbg Assemble window. The assembly code pane displays:

```

00408591 : 894424 28 MOV DWORD PTR SS:[ESP+28], EAX
00408595 : 8D4424 28 LEA EAX, [ESP+28]
004085D8 JA SHORT 004085EA

```

The instruction at address 004085D8 is selected. The assembly pane shows:

```

004085D8 JA SHORT 004085EA

```

Below the assembly pane are two checkboxes:

- Keep size
- Fill rest with NOPs

Buttons for "Assemble" and "Close" are at the bottom right.

On the right side of the window, there are several labels in red text:

- localtime
- strftime
- MSVCRT.\_atoi
- Arg1 => [435480] = 304
- WS2\_32.closesocket

Below the assembly pane, the memory dump pane shows:

```

004085D0 : E8 126E0100 CALL <JMP.&msvcrt._atoi>
004085D1 : 3B05 80C84201 CMP EAX, DWORD PTR DS:[42C880]
004085D2 : 7C 10 JL SHORT 004085EA
004085D3 : A1 80544300 MOV EAX, DWORD PTR DS:[435480]
004085D4 : 890424 MOV DWORD PTR SS:[ESP], EAX
004085D5 : E8 21C50000 CALL <JMP.&WS2_32.closesocket>
004085D6 : 83EC 04 SUB ESP, 4
004085D7 : > 9C74424 08 10 MOV DWORD PTR SS:[ESP+8], 10
004085D8 : 004424 28 EN ENV record

```

Figura 1: OllyDBG - Il jump viene cambiato per forzare il flusso di esecuzione voluto.

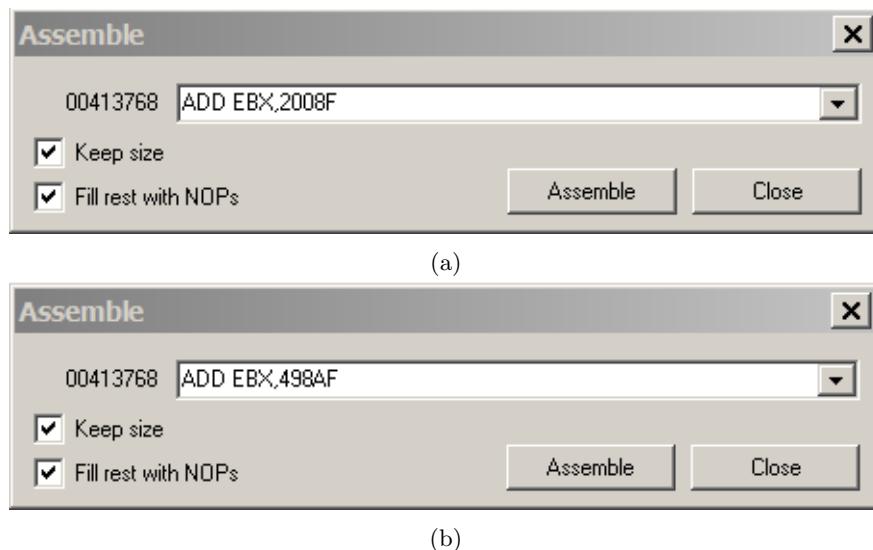


Figura 2: OllyDBG - La “standard expiration date” viene modificata da (a) a (b)

## 2 Analisi Statica

Nel nostro report iniziamo ovviamente dall'analisi statica del malware, per svolgerla sono stati utilizzati principalmente due applicativi: PEStudio e CFFExplorer tuttavia presentando il secondo le stesse informazioni del primo, a seguito analizzeremo solo i risultati ritornati da PEStudio.

## 2.1 PEStudio

**Definizione.** PEStudio è un tool freeware per ispezionare file binari relativi ad applicazioni a 32 o 64 bit senza la necessità di doverli eseguire. È in grado di aprire file di diverso tipo, come ad esempio \*.exe, \*.dll, \*.cpl, \*.ocx, \*.ax, \*.sys ed altro ancora. Il software riesce a fornire un insieme nutrito di informazioni sulle applicazioni, fra le quali si evidenziano le librerie utilizzate, le funzioni importate, le funzioni esportate e condivise con altre librerie. [9]

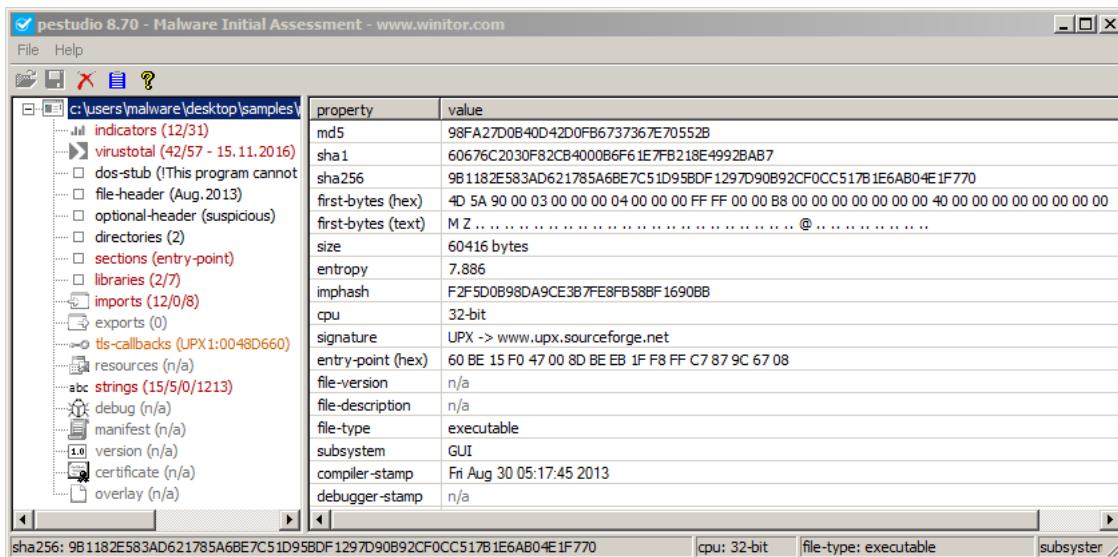


Figura 3: PEStudio - Malware Packed

Analizzando il malware *sample.exe* in PEStudio otteniamo sulla schermata principale i risultati in Figura 3, possiamo dunque analizzare le varie informazioni:

- I vari hash: md5, sha1 e sha256, utili per trovare informazioni online sul virus.
  - I primi bytes e la dimensione del file exe.
  - L'entropia che può essere usata per stabilire se un programma ha subito un processo di packing, come nel nostro caso.
  - L'architettura 32/64 bit, nel nostro caso a 32.
  - La firma in cui viene indicato, se presente, il packer utilizzato. Nel nostro caso il packer è UPX.
  - Altre informazioni come ad esempio l'entry-point e la data di compilazione del programma.

Avendo dunque scoperto che il nostro *sample.exe* è packed, procediamo dunque all'unpacking prima di continuare con l'analisi su PEStudio.

```

Administrator: C:\Windows\system32\cmd.exe
C:\Users\Malware>C:\Tools\upx\upx.exe -d "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample.exe" -o "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
                                         Ultimate Packer for eXecutables
                                         Copyright (C) 1996 - 2013
UPX 3.91w      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013
File size       Ratio      Format      Name
----- 174592 <- 60416  34.60%    win32/pe   sample_unp.exe
Unpacked 1 file.

C:\Users\Malware>

```

Figura 4: CMD - UPX unpacking del malware

### 2.1.1 UPX

**Definizione.** UPX è un programma Open Source per comprimere files creato da Markus F.X.J. Oberhumer e László Molnár. Non sostituisce normali programmi di compressione come Winzip ma li affianca, UPX comprime infatti solo .exe e .dll lasciandoli però eseguibili.

Il comando utilizzato per eseguire l'unpacking è quello in Figura 4, abbiamo quindi ottenuto *sample-unp.exe*. Fatto questo il report generato ovviamente cambia e risulta essere quello in Figura 5. Tra le tante informazioni che cambiano le principali risultano essere:

- l'aumento del numero delle stringhe,
- la variazione dell'entry point del programma,
- il campo firma, nel quale era prima presente UPX, ora risulta vuoto.

Fare l'unpacking dell'eseguibile ci consente di rendere nuovamente comprensibile il codice assembly che era stato compresso. Anche le stringhe potrebbero essere state rese irriconoscibili dal packer, dunque ci si aspetta che dall'analisi di un file unpacked rispetto al corrispettivo packed, anche il numero di stringhe individuate da PEStudio aumenti, in effetti è così. Per tale motivazione tutte le analisi delle altre schede di PEStudio, che faremo nelle seguenti sezioni, verranno fatte sull'eseguibile unpacked.

Analizzando la schermata *indicators* per il file unpacked (Figura 6), PEStudio ci propone diversi indicatori, tra i quali, i più interessanti, risultano i seguenti:

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• la modifica al registro di sistema</li> <li>• l'enumerazione dei processi in esecuzione, delle thread e dei moduli caricati</li> <li>• l'utilizzo di protocolli internet</li> </ul> | <ul style="list-style-type: none"> <li>• il controllo delle firme dei browser installati</li> <li>• la verifica della presenza di una sandbox</li> <li>• la modifica di privilegi</li> </ul> |
|--|--|

### 2.1.2 Analisi delle sezioni

La parte delle sezioni su PEStudio non contiene moltissime informazioni rilevanti. Se si effettua l'analisi del file packed, si trovano le sezioni UPX0, UPX1 e UPX2, a ulteriore conferma che il file è packed con UPX. Invece nel file unpacked le sezioni sono in numero maggiore, rispettivamente: *.text* (contenente l'entry point), *.data*, *.rdata*, *.eh.fram* (contenente la parte di gestione delle eccezioni), *.bss*, *.idata*, *.CRT*, *.tls* (di cui parleremo a seguito).

### 2.1.3 Analisi delle librerie e dei relativi imports

Nelle schede *libraries* (Figura 7) e *imports* (Figura 8) di PEStudio sono visualizzate le librerie importate e le chiamate alle funzioni di queste ultime nel codice. Tutte queste informazioni permettono di capire meglio il comportamento del malware. Nel nostro caso PEStudio mostra 7 librerie, di cui 2 sospette (secondo una

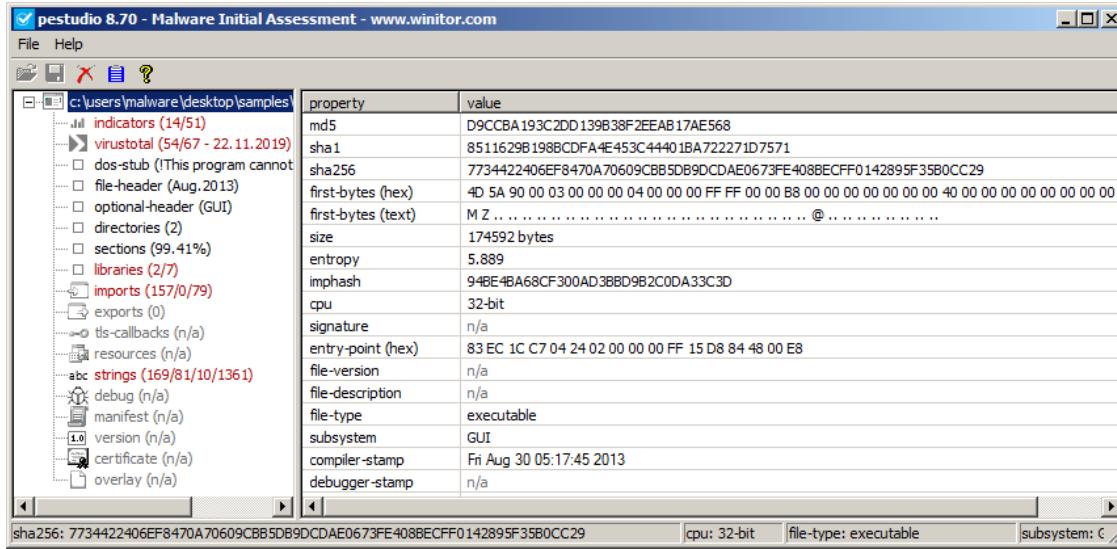


Figura 5: PEStudio - Malware Unpacked

indicator (51)		severity
The file modifies the Registry		1
The file enumerates the list of running processes		1
The file enumerates the list of loaded modules		1
The file references the File Transfer Protocol (FTP) API		1
The file enumerates the list of registered windows		1
The file references the protection of the Virtual Address space		1
The file references Alternate Data Stream (ADS)		1
The file fingerprints Web browsers		1
The file fingerprints Sandboxes		1
The file enumerates the list of running threads		1
The file is scored (54/67) by virustotal		1
The file references (2) blacklisted library(ies)		1
The file references a URL pattern (www.microsoft.com)		1
The file references (1) Windows built-in privilege(s)		1
The file references the Windows Socket (winsock) API		2
The file spawns another process		2
The file references the Windows Debug Helper API		2
The file enumerates files		2
The file references the Global Atom Table		2
The file references Dynamic Data Exchange (DDE)		2
The file references the Domain Name System (DNS) API		2
The file references the Windows Internet (WinINet) library		2
The file references the Internet Control Message Protocol (ICMP)		2
The file references the RPC Network Data Representation (NDR) Engine		2
The file installs an Exception Handler		2
The file references (169) blacklisted string(s)		2
The file references a MIME64 encoding string		2
The file imports (79) blacklisted function(s)		2

Figura 6: PEStudio - Malware Unpacked - Indicatori

blacklist interna al programma) e molte chiamate, 79 delle quali sono a metodi che potrebbero generare un comportamento malevolo.

A seguito verranno elencate le varie librerie mostrate. Nota: per capire la funzione e l'utilizzo delle librerie si è cercato per lo più di usare la documentazione che *Microsoft* fornisce [20]. Laddove non vi fossero dati relativi alla *dll* su quest'ultima, si sono incrociate informazioni acquisite su siti e forum online.

- **wininet.dll:** consente all'applicazione di interagire con i protocolli FTP e HTTP per accedere a risorse

Internet. Con l'evoluzione degli standard e l'introduzione di nuovi protocolli, queste funzioni gestiscono le modifiche dei protocolli sottostanti. Tra le chiamate più rilevanti troviamo:

- InternetConnectA, InternetOpenA, InternetOpenUrlA e InternetReadFile che possono significare che l'eseguibile tenta di reperire risorse da pagine web o più genericamente ricevere/inviare informazioni da e verso url online.
- **ws2\_32.dll**: implementa l'API Winsock, che fornisce funzioni di rete TCP/IP e implementa anche una parziale compatibilità con altre API di rete. Corrisponde nelle versioni più dattate di Windows a wsock.dll e wsock32.dll. Dalla presenza di questa e della precedente libreria, ipotizziamo che il malware utilizzi funzioni di rete. Tra le varie funzioni quella che riteniamo più rilevante è:
  - gethostbyname: funzione utile a risolvere un host name in un indirizzo IP.
- **kernel32.dll**: è la Dynamic Link Library a 32 bit del kernel Windows. Permette la gestione della memoria, delle operazioni di input/output e degli interrupt di sistema. Tra le chiamate più interessanti al fine della nostra analisi ci sono:
  - CreateProcessA, TerminateProcess, CreateThread, CreateMutexA, OpenMutexA e Sleep permettono di creare e gestire processi figli e thread figlie, permettono inoltre la gestione dei semafori.
  - CreateDirectoryA, CopyFileA, DeleteFile e WriteFile vengono usate per manipolare il filesystem.
- **advapi32.dll** (Advanced Windows 32 Base API): fornisce accesso alle risorse fondamentali disponibili per un sistema Windows. Sono inclusi elementi come file system, dispositivi, processi e thread, accesso al registro di Windows e gestione degli errori. Tra le chiamate più rilevanti troviamo:
  - RegCreateKeyExA, RegFlushKey, RegSetValueExA e RegDeleteValueA permettono di aggiungere nuove chiavi nel registro di Sistema di Windows e di manipolare i valori delle chiavi esistenti.
  - OpenProcessToken, AdjustTokenPrivileges e LookUpPrivilegeValueA servono alla modifica di privilegi.
- **mscrv.dll** (DLL di runtime Microsoft Visual C++): questa DLL è associata a versioni di Visual C ++ precedenti alla VS2005 e implementa varie funzionalità di Windows.
- **shell32.dll**: contiene le funzioni dell'API Shell di Windows, lo scopo di questa libreria è fornire supporto per l'accesso ai file.
  - ShellExecute: il malware esegue probabilmente comandi nella Shell di Windows.
- **user32.dll**: libreria in cui Windows memorizza le istruzioni per elementi grafici come ad esempio finestre di dialogo. Le funzioni usate dall'eseguibile potrebbero servire a ottenere informazioni sugli altri processi in esecuzione.

#### 2.1.4 Analisi delle stringhe

Le stringhe del programma sono visualizzate nella scheda *strings* di PEStudio. Cerchiamo ora di suddividerle per capire qual'è lo scopo di ognuna di esse:

- **Stringhe codificate**: nell'eseguibile sono presenti molte stringhe che sembrano codificate in *Base64*, come ad esempio LcQ1vUua5UwEGjzgwx... e 2PC8GGALSsrsttKc.....
- “Chiavi” **Base64**: nel malware sono presenti delle stringhe che vengono probabilmente usate per decodificare stringhe in *Base64*. Queste si contraddistinguono per contenere una sequenza di caratteri in ordine alfabetico, per esempio ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/.

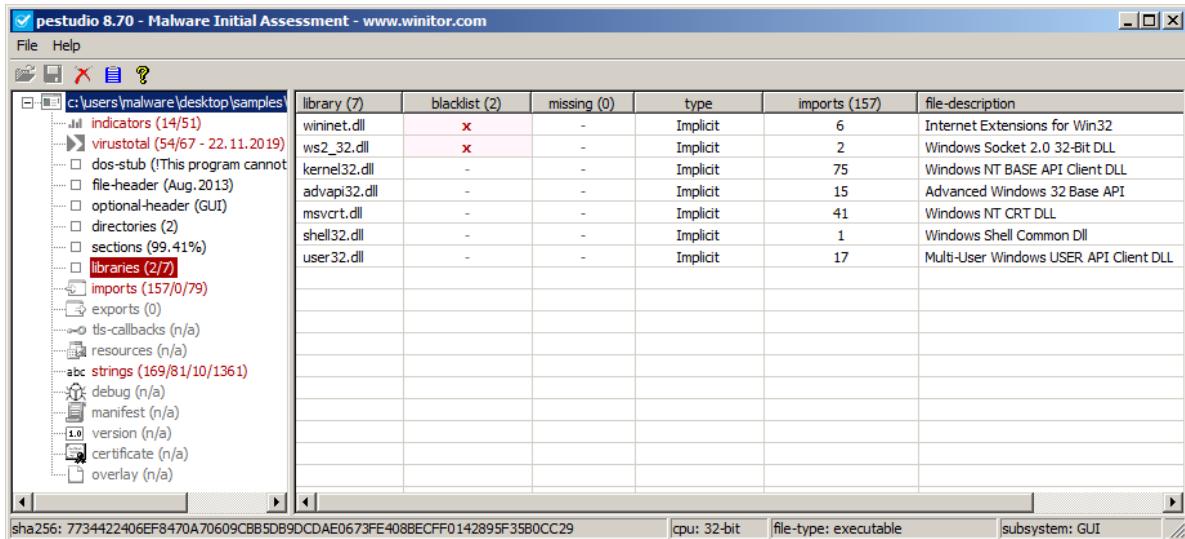


Figura 7: PEStudio - malware unpacked - Librerie

symbol (157)	blacklist (79)	library (7)
AddAtomA	x	kernel32.dll
CopyFileA	x	kernel32.dll
CreateDirectoryA	x	kernel32.dll
CreateMutexA	x	kernel32.dll
CreateProcessA	x	kernel32.dll
CreateThread	x	kernel32.dll
CreateToolhelp32Snapshot	x	kernel32.dll
DeleteFileA	x	kernel32.dll
DuplicateHandle	x	kernel32.dll
FindAtomA	x	kernel32.dll
FindClose	x	kernel32.dll
FindFirstFileA	x	kernel32.dll
FindNextFileA	x	kernel32.dll
FreeConsole	x	kernel32.dll
GetAtomNameA	x	kernel32.dll
GetCurrentProcess	x	kernel32.dll
GetCurrentProcessId	x	kernel32.dll
GetCurrentThreadId	x	kernel32.dll
GetLogicalDrives	x	kernel32.dll
GetModuleFileNameA	x	kernel32.dll
GetModuleHandleA	x	kernel32.dll
GetProcAddress	x	kernel32.dll
GetProcessHeap	x	kernel32.dll
GetSystemInfo	x	kernel32.dll
GetSystemPowerStatus	x	kernel32.dll
GetTickCount	x	kernel32.dll
GlobalMemoryStatus	x	kernel32.dll
LoadLibraryA	x	kernel32.dll

Figura 8: PEStudio - malware unpacked - Alcuni import delle Librerie

- **Nomi di librerie:** sono presenti numerosi nomi di librerie. Esse potrebbero essere caricate a runtime o utilizzate per verificare un preciso stato del sistema (ad esempio se ci si trova su un sistema reale o su una VM). Esempi di queste librerie sono: `rundll32`, `dnsapi.dll`, `SbieD1l.dll`, `snxhk.dll`, `dbghelp.dll`. Tra queste librerie sono contenute anche quelle dell'eseguibile stesso.
- **Chiavi di registro:** sono presenti molte stringhe che si riferiscono a chiavi o indirizzi del registro di sistema. Probabilmente il malware cercherà di leggere o modificare i valori delle suddette chiavi. Tra queste alcuni esempi rilevanti sono: `Software\Microsoft\Windows.NT\CurrentVersion` e

RunOnceEx.

- **Stringhe relative a protocolli di rete:** indicano che il malware in qualche modo utilizza protocolli internet per comunicare con l'esterno. Esempi di queste stringhe sono `http`, `https`, `firefox.exe` e altri nomi di browser, POST, PING e molte altre.
- **Stringhe relative al protocollo IRC:** sono stringhe riguardanti il protocollo *IRC*, tra queste abbiamo ad esempio JOIN, PING, kill user, IRC War, Flood ma anche siti, software o servizi vari legati al protocollo stesso: `lightIRC.com`, `HexChat.2.9.3.[x64]`, etc.
- **Stringhe con placeholders:** nell'eseguibile sono presenti molte stringhe di formato che contengono placeholders. A runtime verranno probabilmente utilizzate per produrre stringhe con data/ora (%y%m%d), indirizzi mail come ad esempio %s@s e indirizzi IRC compresi di porta %s@s:%i, di questi ultimi ne parleremo meglio a seguito.
- **Eseguibili di sistema:** si tratta di eseguibili del sistema Windows: `explorer.exe`, `regedit.exe`, `cmd.exe`, `msconfig.exe`, `taskmgr.exe`.
- **Domini Web:** sono stringhe relative a domini web vari `www.update.microsoft.com`, `yahoo.com`, `hotmail.com`, `facebook.com`, `live.com` e altri. L'intuizione è che alcuni di questi possano essere utilizzati per formare indirizzi email.
- **Directory:** sono stringhe che rappresentano path o nomi di directory e file di sistema: `WindowsNT`, `System32`, `\System32\drivers\etc\protocol`

### 2.1.5 Virus Total

La sezione *virustotal* del tool, mostrata in Figura 9, ritorna il livello di rischio che vari antivirus associano all'eseguibile. Il fatto che 54 su 67 di questi segnalino questo eseguibile come pericoloso, ci conferma la natura malevola di *sample.exe*. Tra le altre informazioni notiamo in particolare che molti antivirus segnalano il malware come *IRC Bot*, tale dato risulta essere il punto di partenza per capire l'intento del programma.

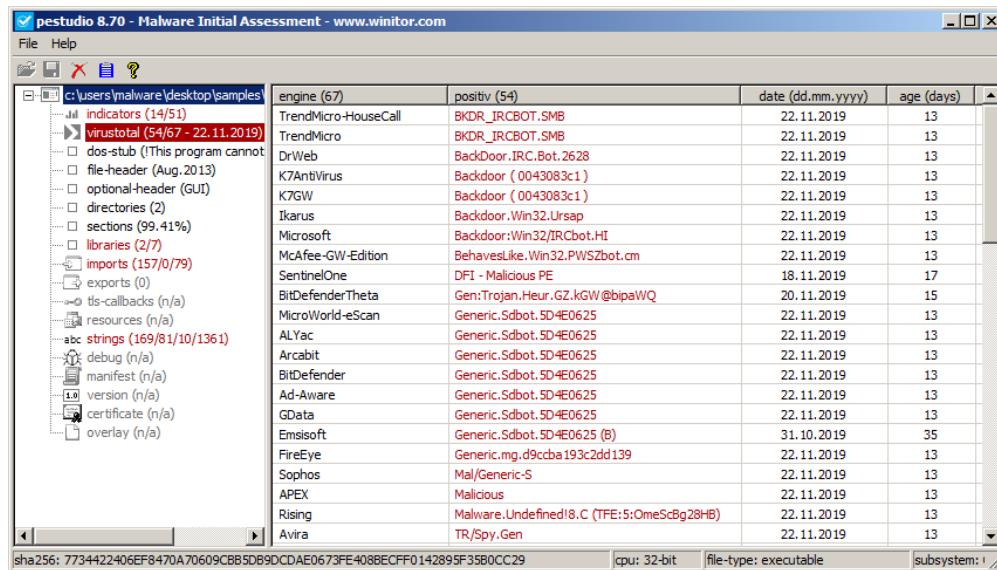


Figura 9: PEStudio - malware unpacked - Virustotal sections

Oltre alle informazioni ottenute da PEStudio ne abbiamo cercate di altre sul malware anche online, sia sul sito *virustotal.com* (Figura 10) che tramite una ricerca delle stringhe generica su *Google*. I risultati ottenuti verranno discussi attentamente nella sezione successiva.

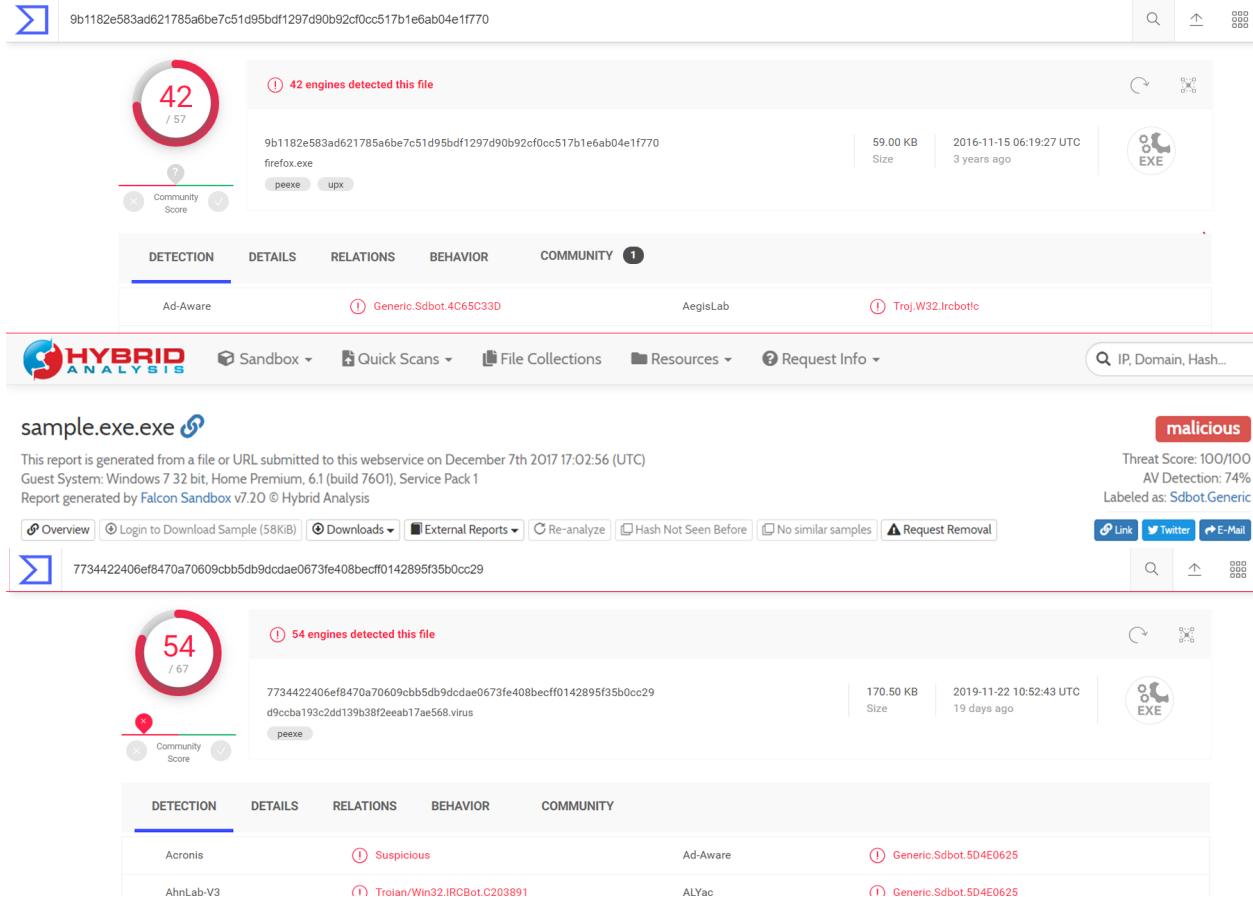


Figura 10: Virustotal/Hybrid - (a) (b) malware packed (c) malware unpacked

### 2.1.6 IRC Bot - Una valida ipotesi

Dalle analisi fatte fino ad ora possiamo già trarre alcune importanti conclusioni, che verranno confermate anche dalla successiva analisi dinamica e dal Reverse Engineering. La grossa intuizione è che l'attività del nostro malware ruoti attorno, come già accennato, all'utilizzo del protocollo *IRC*.

**Definizione.** *IRC (Internet Relay Chat)* è un protocollo di messaggistica istantanea su Internet o, detto in altri termini, un sistema che consente conversazioni (chat) in tempo reale. *IRC* consente il dialogo tra due o più persone contemporaneamente. Il tutto avviene all'interno di stanze virtuali chiamate canali.

Da una prima analisi non sembra errato ipotizzare che l'eseguibile utilizzi uno o più canali *IRC* per trasferire informazioni a un dispositivo remoto, potrebbe inoltre ricevere comandi da quest'ultimo per eseguirli sul computer infettato.

Cercando su *Google* alcune delle stringhe visualizzate su PEStudio si trovano parecchi riferimenti a una repository *GitHub*, *3val/Athena* [1]. Se si analizza il progetto si scopre che sono moltissime le stringhe che sono in comune tra i due. L'ipotesi è dunque che il creatore del malware si sia basato su di quest'ultimo per la creazione di *sample.exe*. *Athena* è un *IRC Bot* per Windows, programmato in *c++*, capace di compiere varie operazioni in ambito di canali *IRC*, ne parleremo meglio a seguito e in particolare nella parte di Reverse Engineering.

## 3 Analisi Dinamica

Avendo completato l'analisi statica si procede ora all'analisi dinamica. Abbiamo quindi osservato il comportamento del malware in vari ambienti controllati per capire quali fossero gli effetti sul sistema. Verranno separate le operazioni effettuate da *sample.exe* (ovvero dal malware non attivo a causa del controllo sulla data di sistema), da quelle di *sample\_unp\_mod.exe*.

### 3.1 Ambiente di Testing

Prima di procedere nella descrizione delle varie analisi e dei relativi software utilizzati, diamo l'elenco degli ambienti usati per il testing:

- Virtual Machine usata durante il corso - SO: Windows 7 a 32bit (completa di vari software per l'analisi di malware)
- Virtual Machine con Windows 7 32 bit ma priva di software di analisi installati e priva delle Guest Addition fornite da VirtualBox.
- Ambiente reale: computer con Windows XP collegato in una piccola rete locale con un altro pc che monitorava il traffico di rete con Wireshark.

Le ultime due soluzioni erano state adottate per evitare che l'eseguibile potesse rilevare di essere eseguito in un ambiente di analisi. Tale precauzione era stata messa in piedi durante le prime analisi, poichè nonostante gli sforzi, non si riusciva a rilevare, da parte di *sample.exe*, nessun comportamento realmente malevolo per la macchina di test. L'analisi successiva, utilizzando la macchina XP e quella priva di software di analisi, ha dimostrato che nemmeno in questi due ambienti il malware aveva alcun comportamento malevolo. Successivamente si è scoperto che l'unica ragione per cui il sample non si comportava correttamente era il controllo sulla data.

### 3.2 Comportamento apparente di *sample.exe*

Abbiamo iniziato con l'osservare il comportamento del malware senza utilizzare software di analisi, guardando cioè solo le modifiche visibili nel sistema. Nella prima fase di analisi, prima cioè di scoprire il controllo fatto dal sample sulla data di sistema, ci siamo interrogati sul motivo del mancato funzionamento dello stesso. Si è infatti notato che:

- *sample.exe* non sembra svolgere nessuna azione realmente malevola per il pc: dopo essere stato avviato rimane in esecuzione senza modificare apparentemente le impostazioni della macchina o almeno non in modo significativo.
- Non sembra esserci alcuna persistenza: osservando i processi con il Task Manager di Windows, dopo un riavvio, il sample non è più in esecuzione, neppure sotto altri nomi. Controllando manualmente le chiavi di registro Run e RunOnce ci siamo accorti che l'eseguibile inserisce una barra verticale come valore di una chiave. Ci siamo quindi chiesti il perché lo faccia, visto che nell'ecosistema Windows il comando “|” non produce alcun risultato preso singolarmente. Ne discuteremo meglio a seguito.

Verificando poi tali anomalie anche con i tool di analisi, all'inizio pensavamo i problemi fossero legati al rilevamento della Macchina Virtuale da parte del sample. Come detto sopra, l'esecuzione del malware su un ambiente reale (pc con Windows XP collegato in una piccola rete locale) ci ha fatto cambiare idea. Forse il creatore aveva smesso di sfruttare da remoto il malware e ora quest'ultimo risultava essere inattivo? La risposta, come sappiamo, non si allontana molto da questa idea.

### 3.3 Comportamento apparente di *sample\_unp\_mod.exe*

Abbiamo successivamente verificato l'operato dalla versione modificata del sample sul sistema, come vedremo ora le operazioni malevoli svolte da quest'ultima sono evidenti fin da subito. I comportamenti osservati sono i seguenti:

- Appena il malware viene avviato, sparisce dalla directory in cui si trova ma resta comunque in esecuzione.
- Il malware crea persistenza infatti, dopo il riavvio del sistema, si può osservare che il processo *sam-ple\_unp\_mod.exe* non esiste più, ma vi sono due nuovi processi, *AdobeARM.exe* e *ADService.exe*, che non erano presenti nella precedente esecuzione di Windows (Figura 11).
- Al primo riavvio successivo all'esecuzione del sample, le cartelle nascoste non sono più visibili nell'esplora risorse di Windows, il malware disabilita inoltre le opzioni sulla cartella (Figura 12). In questo modo rende più difficile la sua individuazione da parte di un utente con un'esperienza medio-bassa.
- Attraverso il task manager si può risalire facilmente ai path dei due eseguibili, il primo è presente in *C:\Users\Malware\AppData\Roaming\1726961194* mentre il secondo in *C:\ProgramData*. Verificando manualmente le directory tramite il prompt dei comandi, si trovano effettivamente i due file exe.
- Il peso dei due eseguibili (170 KB circa) è identico, verificando a seguito con PEStudio potremo stabilire se sono esattamente lo stesso malware oppure il risultato di una mutazione del sample in più versioni diverse.
- Controllando la chiave *Run* del registro di sistema si nota la presenza dell'eseguibile *AdobeARM.exe* con il relativo path dove è collocato, *ADService.exe* non è invece presente. L'ipotesi è che quest'ultimo venga creato ed eseguito da *AdobeARM.exe* all'avvio di Windows.
- Avviando più volte il malware si scopre che la posizione di *ADService.exe* nel file system sembra cambiare ad ogni esecuzione, mentre quella di *AdobeARM.exe* rimane invariata.
- Sia *AdobeARM.exe* che *ADService.exe* hanno l'attributo *Hidden*.
- Terminando uno dei due eseguibili l'altro viene subito riavviato. Chiaramente con questo metodo il malware garantisce di essere sempre in esecuzione.
- Controllando il sistema ulteriormente, anche Windows Update sembra aver subito modifiche, il programma rileva infatti delle impostazioni diverse, come si può notare in Figura 13.
- Quando si forza la terminazione di *AdobeARM.exe* oppure dopo alcuni riavvii, accade una cosa particolare: *ADService.exe* viene copiato nella medesima directory dove si trovava ma con una estensione che sembra essere randomica (e.g. *ADService.exe.uspmmjkk*). Inoltre il file originale *ADService.exe* non è più tale poiché osservando la dimensione del file ora pesa meno di 2kb. Da una prima ipotesi potrebbe essere anche questo un meccanismo attuato dal sample per evitare di essere cancellato.
- Un ultimo comportamento osservato all'avvio del malware è la chiusura di un processo di VBox Guest Additions. L'icona che era presente sulla System Tray (Figura 14) scompare.
- Facendo nuovi test ma avviando l'eseguibile come amministratore le operazioni svolte dal malware non sembrano cambiare.

Per non confondere il file *ADService.exe* con l'omonimo file che pesa solo 2kb circa, nel resto della relazione spesso ci riferiremo a quest'ultimo con l'appellativo di "file da 2kb".

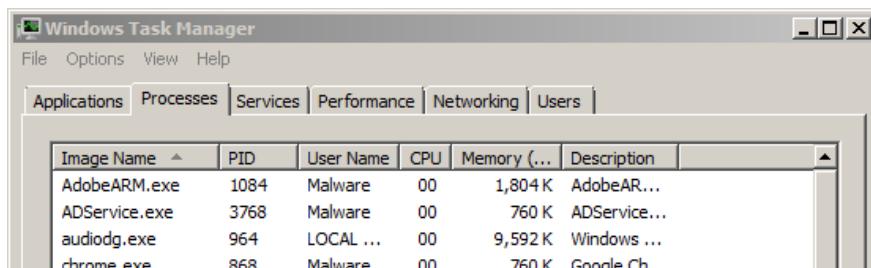


Figura 11: Task Manager - Processi *AdobeARM.exe* e *ADService.exe*.

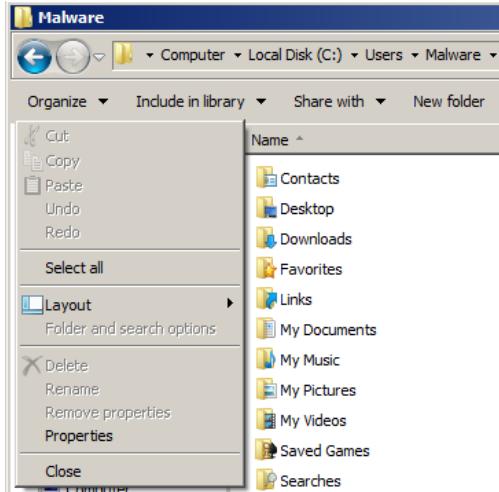


Figura 12: Esplora Risorse - “Opzioni cartella e ricerca” è disattivato.

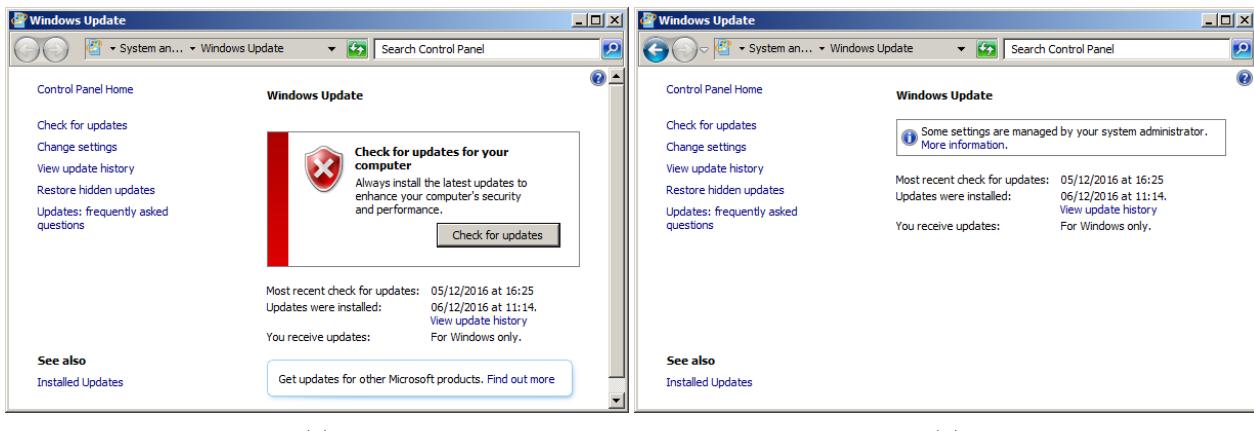


Figura 13: Windows Update - Stato degli aggiornamenti prima (a) e dopo (b) l'avvio del malware.

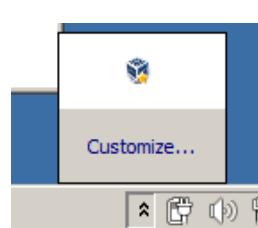


Figura 14: System Tray - VBox Guest Additions risulta attivo prima dell'avvio del malware, viene disattivato in seguito.

### 3.4 Una nuova parentesi su PEStudio

Con questa prima parte di analisi dinamica abbiamo scoperto nuovi eseguibili, necessitiamo ora di ritornare a usare PEStudio, per verificare che i malware siano effettivamente tutti lo stesso file, copiato più volte con nomi diversi e in path distinti. Come si può vedere in Figura 15, confrontando l'hash di *sample\_unp\_mod.exe* (l'eseguibile su cui abbiamo cliccato per avviare il malware) con quello di *AdobeARM.exe* e *ADService.exe*, risulta che si tratta effettivamente dello stesso eseguibile. Ovviamente l'hash di *sample\_unp.exe* (precedentemente analizzato) è diverso da quello di *sample\_unp\_mod.exe*, in quanto il primo è stato modificato con

un bypass sulla data di sistema per ottenere il secondo. Tuttavia, essendo l'unica modifica fatta, la parte delle stringhe e degli import è identica. Resta tuttavia da svolgere l'analisi statica per un ultimo file: si tratta dell'eseguibile da 2kb con lo stesso nome del malware (*ADService.exe*). Sottoponendolo a PEStudio, il risultato è quello in Figura 16. Secondo l'analizzatore il file in questione non è un eseguibile, non ha dunque senso concentrarci sull'entropia, anche successive analisi fatte ci hanno confermato che il file non è packed. Le poche stringhe presenti (Figura 17), sembrano un insieme di caratteri ascii casuali, non c'è alcuna libreria o import. Infine non risultano esserci informazioni sull'affidabilità del file da parte di Virustotal, infatti anche caricando il file sul corrispondente sito web, il report generato (Figura 18) [27] non ci dà alcuna informazione per determinare la pericolosità del programma. *ADService.exe* da 2kb sembra essere ignoto agli antivirus. Un'altra stranezza deriva dal fatto che confrontando il report di PEStudio relativo al medesimo file da 2kb, ma generato in momenti diversi dal malware, l'hash cambia ogni volta. Da questo ne consegue che questi file, pur non variando mai di molto la dimensioni in byte, hanno sempre almeno una piccola variazione nei dati al loro interno. Quest'ultimo potrebbe essere il motivo per il quale Virustotal non ha informazioni su di esso e al contrario, quando abbiamo caricato l'eseguibile sul sito, non ci è stata ritornata subito la pagina con il report ma è stata effettuata in quel momento la prima analisi in assoluto.

The figure consists of three vertically stacked screenshots of the PEStudio 8.70 interface, each showing the analysis of a different executable file. Each screenshot has a title bar indicating the file being analyzed and the PEStudio version.

- (a) sample\_unp\_mod.exe:** This screenshot shows the analysis of a 2KB executable. The left pane displays the file structure with sections like indicators (13/50), virustotal (n/a), dos-stub (labeled as 'This program cannot be run in DOS mode.'), file-header (Aug. 2013), and optional-header (GUI). The right pane is a table of properties and their values, including md5, sha1, sha256, and first-bytes (hex).
- (b) AdobeARM.exe:** This screenshot shows the analysis of another 2KB executable. The file structure is similar to (a), with sections for indicators (13/50), virustotal (n/a), dos-stub, file-header (Aug. 2013), and optional-header (GUI). The right pane shows identical property values to (a), including md5, sha1, sha256, and first-bytes (hex).
- (c) ADService.exe:** This screenshot shows the analysis of a third 2KB executable. The file structure is identical to the others. The right pane shows identical property values to (a) and (b), including md5, sha1, sha256, and first-bytes (hex).

Figura 15: PEStudio - Gli eseguibili *sample\_unp\_mod.exe* (a), *AdobeARM.exe* (b) e *ADService.exe* (c) sono identici come si può vedere dall'hash del file.

The screenshot shows two windows from PEStudio. The top window displays file properties for 'c:\programdata\adservice.exe'. The properties listed are:

property	value
md5	7D176CE66863924F669527F0706B6676
sha1	2938CAD40EDA167ED4394E804CC055A9758D250D
sha256	9CA35E1A6498C0721E43EC27E3F95D3060FDA6B3C9C6CE4B4C0BA517BB2F4D1C
first-bytes (hex)	04 DD 79 E5 7B 95 2E 12 78 00 21 98 ED 1B 1B 0E 08 E4 D4 F6 FA C8 3C 3B AA 70 8D 35 27 21 50 03 B1
first-bytes (text)	... y .. { ... x .. ! ..... < .. p .. 5' ! P ..
size	2000 bytes
entropy	7.879

The bottom window shows indicators for the same file, listing a single indicator: 'The file is not an executable file' with a severity of 9.

Figura 16: PEStudio - eseguibile da 2kb - hash e indicatori

This screenshot shows the 'strings' analysis for 'c:\programdata\adservice.exe'. The table lists 24 ASCII strings found in the file, each with its type, size, location, and value.

type	size	location	blacklist (0)	hint (0)	whitelist (0)	value (24)
ascii	4	0x0000...	-	-	-	5'P
ascii	5	0x0000...	-	-	-	:gE7W
ascii	4	0x0000...	-	-	-	\uS
ascii	4	0x0000...	-	-	-	\$yA
ascii	5	0x0000...	-	-	-	%enqs
ascii	4	0x0000...	-	-	-	)D(
ascii	4	0x0000...	-	-	-	<Qmn
ascii	6	0x0000...	-	-	-	gD:]43
ascii	4	0x0000...	-	-	-	})g4
ascii	4	0x0000...	-	-	-	f#LT
ascii	4	0x0000...	-	-	-	yg\u
ascii	4	0x0000...	-	-	-	g6!W
ascii	4	0x0000...	-	-	-	oF\X
ascii	4	0x0000...	-	-	-	Eiv4
ascii	5	0x0000...	-	-	-	[.W<
ascii	7	0x0000...	-	-	-	9wymBtD
ascii	4	0x0000...	-	-	-	9\vk

SHA256: 9CA35E1A6498C0721E43EC27E3F95D3060FDA6B3C9C6CE4B4C0BA517BB2F4D1C

Figura 17: PEStudio - eseguibile da 2kb - stringhe

This screenshot shows the VirusTotal report for the file '9ca35e1a6498c0721e43ec27e3f95d3060fda6b3c9c6ce4b4c0ba517bb2f4d1c'. The report indicates that no engines detected the file. The file details are:

- File Hash: 9ca35e1a6498c0721e43ec27e3f95d3060fda6b3c9c6ce4b4c0ba517bb2f4d1c
- Size: 1.95 KB
- Upload Date: 2020-03-31 19:18:56 UTC
- File Name: ADService.exe

The report includes tabs for DETECTION, DETAILS, and COMMUNITY. Under DETECTION, results from Ad-Aware, AhnLab-V3, AegisLab, and ALYac are shown, all marked as Undetected. There is also a comment section with a blue speech bubble icon.

Figura 18: Virustotal - report sull'eseguibile da 2kb

### 3.5 Comportamento apparente del file ignoto da 2kb

Per capire a cosa servisse il file da 2kb, inizialmente si è pensato di provare ad eseguirlo. A seguito i risultati ottenuti.

1. Avviando la prima volta il presunto eseguibile (anche se PEStudio non lo classifica come tale), appare un messaggio di errore, mostrato in Figura 19.
2. Verificando le proprietà, si scopre che il file ha tutti i privilegi bloccati per l'utente corrente (Figura 20a). Si procede alla modifica manuale di tali impostazioni, per procedere a un nuovo tentativo di avvio.
3. Dopo un ulteriore tentativo, il file viene bloccato dalle impostazioni di sicurezza di Windows, si procede allo sblocco e si tenta ancora.
4. Ora Windows ritorna un errore relativo al fatto che il file è già usato da un altro processo (Figura 20b). Dopo alcuni tentativi si scopre che il file è aperto in *explorer.exe*, i motivi non sono chiari.
5. Dopo aver riavviato Esplora Risorse, l'eseguibile finalmente inizia l'esecuzione, l'interfaccia è quella comandata ma i risultati sono apparentemente deludenti:
  - L'eseguibile lascia aperta una interfaccia DOS, che si chiude subito dopo qualche secondo.
  - In altri casi l'eseguibile produce alcuni errori, mostrati in Figura 20c e 20d.

In ogni caso, sia attraverso gli errori prodotti dall'eseguibile che tramite l'utilizzo del task manager (Figura 21), si osserva che la breve esecuzione di questo file coinvolge *NTVDM.exe*.

**Definizione.** *NTVDM (NT Virtual Dos Machine)* è una macchina virtuale che consente di eseguire app a 16 bit, insieme a WoW16 nelle moderne versioni di Windows. [26]

Successivamente si è passati a verificare se le restrizioni relative ai privilegi di avvio/modifica fossero le stesse anche per i file *AdobeARM.exe* e *ADService.exe* (quello della stessa dimensione dell'eseguibile iniziale), l'esito è negativo: il file da 2kb risulta quindi essere l'unico con tali impostazioni di sicurezza.

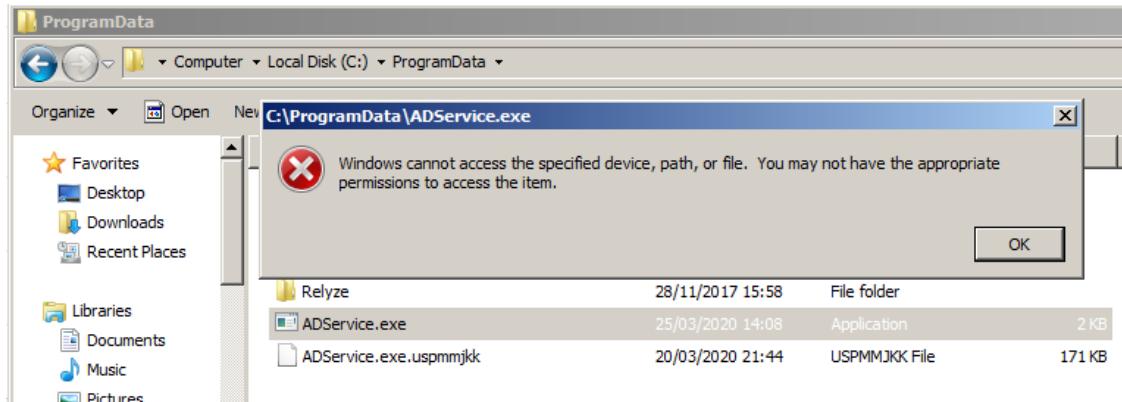


Figura 19: Primo tentativo di esecuzione del file da 2kb

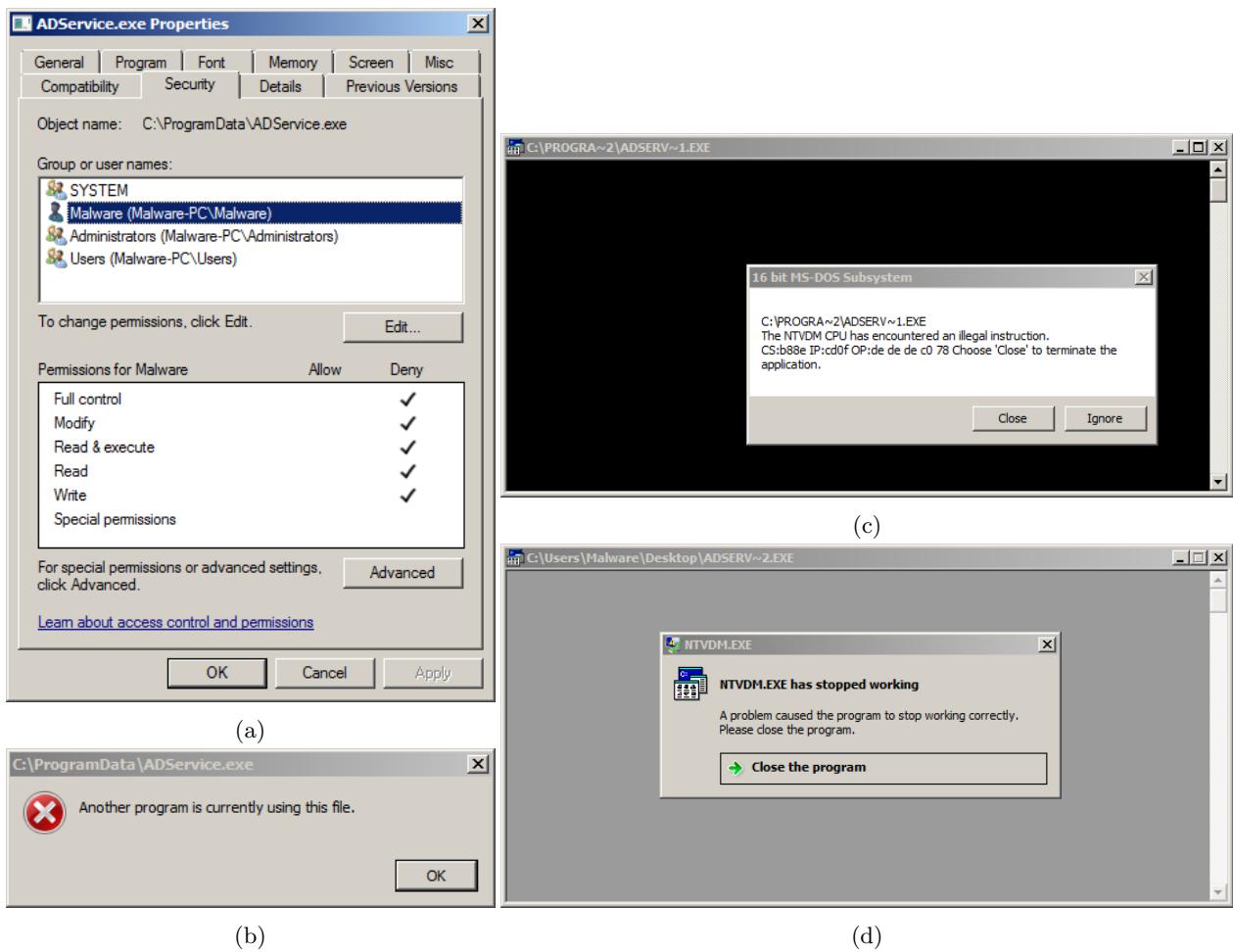


Figura 20: File da 2 kb: (a) privilegi dell'utente corrente, (b) errore file in uso da un altro programma, (c) (d) errori durante l'esecuzione.

Windows Task Manager						
File Options View Help						
Applications Processes Services Performance Networking Users						
Image ...	PID	User Name	CPU	Memory (...)	Description	
dumpcap.exe	3472	Malware	00	1,676 K	Dumpcap	
GoogleCrashH...	1408	SYSTEM	00	384 K	Google Cr...	
lsass.exe	436	SYSTEM	00	2,688 K	Local Secu...	
lsm.exe	444	SYSTEM	00	920 K	Local Sess...	
ntvdm.exe	2176	Malware	99	1,068 K	NTVDM.EXE	
SearchIndexe...	2312	SYSTEM	00	8,404 K	Microsoft ...	

Figura 21: Task Manager - Il file da 2kb in esecuzione appare come NTVDM.exe

## 3.6 ProcMon

**Definizione.** *Process Monitor(ProcMon)* rappresenta l'unione di due software freeware già ampiamente conosciuti in Rete: *Filemon* e *Regmon*. Il primo consente di verificare in tempo reale quali file vengono gestiti da parte del sistema operativo e delle applicazioni installate, il secondo invece può essere impiegato per stabilire quali operazioni vengono compiute sul registro di sistema di Windows. I due programmi vengono spesso utilizzati per mettere a nudo l'eventuale presenza di spyware e malware sul sistema in uso. [10]

Le informazioni ritornate da ProcMon sono moltissime, risulta essere dunque utile impostare i giusti filtri che permettano di osservare le informazioni rilevanti. Il primo filtro da impostare, che dovrà essere sempre presente nei successivi punti, è quello relativo ai nomi del processo (*sample.exe*, *AdobeARM.exe*, etc.), verranno tutti inseriti simultaneamente, in modo tale da seguire tutte le operazioni effettuate dal malware in ogni sua copia. È fondamentale infatti visualizzare solo le modifiche apportate dal sample e non dagli altri processi che erano in esecuzione al momento dell'analisi. A seguito vengono impostati inoltre altri filtri, per osservare un determinato tipo di operazione piuttosto che un altro. Vista la mole di dati raccolti, non inseriremo i filtri tutti assieme per evitare di avere troppe informazioni visualizzate.

Come già anticipato *sample\_unp\_mod.exe* crea persistenza, perciò alcune operazioni potrebbero essere visibili solo dopo un riavvio. Non riuscendo a riavviare effettivamente la VM, poiché dopo tale operazione non potremmo eseguire ProcMon in tempo, alcune volte durante l'analisi si sono utilizzati degli script *bat* appositamente realizzati per “simulare il riavvio”. Un esempio è quello in Figura 22 che termina tutti i processi del malware che potrebbero essere in esecuzione, facendo il kill simultaneo infatti si evita che un processo possa riavviare l'altro che è stato terminato. Successivamente lo script riavvia il malware tramite lo stesso comando che viene eseguito all'avvio del sistema operativo, tale comando/path sarà chiaro a seguito, quando parleremo delle modifiche relative al registro di sistema.

```
taskkill /f /im "AdobeARM.exe" /im "ADService.exe"  
/im "sample.exe" /im "sample_unp_mod.exe"  
C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe
```

Figura 22: Script bat che “riavvia” il malware

A seguito di una breve parentesi sulle librerie utilizzate dall'eseguibile, faremo l'analisi delle operazioni relative al file system, poi quelle riguardanti il registro di sistema e così via, distinguendo sempre tra i risultati ottenuti con *sample.exe* e quelli ottenuti con *sample\_unp\_mod.exe*.

### 3.6.1 Librerie relative alle operazioni effettuate (*sample.exe*)

Quasi tutte le operazioni effettuate dal malware sul sistema avvengono attraverso l'utilizzo di librerie. Applicando un filtro sulle *DLL* nel report di ProcMon, si scopre che il malware utilizza all'incirca 29 librerie, presenti nei path elencati a seguito.

- C:\Windows\System32\,
- C:\Users\Malware\Desktop\Samples\Reverse Engineering\: path di *sample.exe* (librerie importate staticamente dal sample),
- C:\Windows\winsxs\

Le librerie sono le seguenti:

- |               |                |                 |                 |
|---------------|----------------|-----------------|-----------------|
| • sechost.dll | • iphlpapi.dll | • nlaapi.dll    | • winrnr.dll    |
| • version.dll | • winnsi.dll   | • NapiNSP.dll   | • dhcpcsvc6.dll |
| • imm32.dll   | • tzres.dll    | • pnrpnsnsp.dll | • dhcpcsvc.dll  |
| • icmp.dll    | • dnsapi.dll   | • msasn1.dll    | • cryptbase.dll |

- ntmartha.dll
- shell32.dll
- shdocvw.dll
- rasadhlp.dll
- rpcss.dll
- comctl32.dll
- secur32.dll
- wship6.dll
- propsys.dll
- apphelp.dll
- sspicli.dll
- wshtcpip.dll
- api-ms-win-downlevel-advapi32-1-0.dll

In particolare dhcpcsvc.dll e dhcpcsvc6.dll consentono operazioni sul DHCP e altre operazioni di rete.

### 3.6.2 Librerie relative alle operazioni effettuate (*sample\_unp\_mod.exe*)

Nel caso della versione del malware sbloccata con il bypass sulla data di sistema e le sue copie, le librerie non cambiano molto ma i path sì. Non essendo presenti *DLL* di sistema nelle directory elencate a seguito (tranne ovviamente *System32*), l'ipotesi è che ciò sia dovuto alla replicazione del malware. Quando un eseguibile utilizza librerie importate staticamente nel codice, queste appaiono con il path dove si trova l'*exe*. Tenendo a mente l'analisi precedente, notiamo che i percorsi in più rispetto a prima sono relativi a directory dove il malware si è copiato durante la sua esecuzione, quindi quando quest'ultimo fa chiamate a libreria da queste posizioni ProcMon ce le fa vedere dai relativi path:

- C:\Windows\System32
- C:\Users\Malware\Desktop\Samples\Reverse\_Engineering\
- C:\Users\Malware\AppData\Roaming\1726961194\
- C:\Users\Malware\AppData\Local\Temp\
- C:\Users\Malware\
- C:\ProgramData\

Vi sono solo due librerie in più, rispetto a quelle di *sample.exe*.

- fwpuclnt.dll
- swdrm.dll

### 3.6.3 Operazioni sul File System (*sample.exe*)

Per analizzare le operazioni che avvengono nel File System dobbiamo settare i seguenti filtri (in aggiunta a quello sui nomi di processo):

- **Operation - is - CreateFile:** indica che un processo apre o crea un determinato file.
- **Operation - is - WriteFile:** indica che un processo esegue un'operazione di scrittura su un determinato file. Questa operazione, rispetto alla precedente, risulta essere molto più rilevante al fine della nostra analisi, poiché ci interessa maggiormente comprendere se il malware modifichi o scriva un file piuttosto che se lo visualizzi solamente.
- **Operation - is - QuerySecurityFile, SetSecurityFile:** ci permettono di vedere se il sample modifica i permessi correnti.

Con questi filtri i risultati rilevanti alla fine della nostra analisi sono quelli che seguono. Nota: tutte le operazioni qui descritte sono eseguite con successo se non altrimenti specificato.

- CreateFile, QuerySecurityFile di C:\Users\Malware\AppData\Local\Temp\I1726961194.bat: parleremo di questo script in una sezione apposita successiva.

- CreateFile, QuerySecurityFile di C:\Users\Malware\AppData\Roaming\1726961194: crea questa directory nascosta, ma le motivazioni saranno chiare a seguito. Da notare che osservando le operazioni su ProcMon e da successive verifiche, nel caso di *sample.exe*, tale directory rimane completamente vuota durante tutta l'esecuzione.
- CreateFile di C:\Users\Malware\Desktop\Samples\Reverse Engineering\|: questa operazione ritorna il risultato: NAME INVALID (Figura 23). Questo ci dice che l'eseguibile tenta di creare una cartella o un file con tale nome ma il sistema non lo permette poiché, nell'ecosistema Windows, la barra verticale ‘|’ è un carattere non permesso nel nome di un file.
- CreateFile di C:\Users\Malware\AppData\Roaming\KMSpico-setup.exe
- CreateFile di C:\Users\Malware\AppData\Roaming\terra.exe: questo e il precedente eseguibile corrispondono a due programmi già presenti sulla macchina virtuale, tali *open* possono indicare che il malware ha fatto una scansione della cartella per cercarne gli eseguibili contenuti.
- CreateFile di C:\Windows\System32\drivers\etc\hosts: indica che l'eseguibile ha visualizzato ma non modificato il file hosts, questo è un file di testo che memorizza i nomi host e gli indirizzi IP a cui sono associati. La sua funzione è quella di indirizzare correttamente la navigazione in Internet a partire dal nome di un sito web, risolvendo i nomi in indirizzi IP.
- CreateFile di C:\Windows\System32\drivers\etc\protocol: come il precedente ma con la differenza che in questo caso il file protocol è un file di testo che risolve i nomi dei protocolli, nei rispettivi numeri di protocollo RFC su una rete TCP/IP.
- WriteFile di C:\Users\Malware\NTUSER.DAT
- WriteFile di C:\Users\Malware\ntuser.dat.LOG1
- WriteFile di C:\Windows\System32\config\SOFTWARE
- WriteFile di C:\Windows\System32\config\SOFTWARE.LOG1

Questi ultimi quattro, fanno parte dei file nascosti su cui il sistema operativo Windows salva il registro di sistema e i log delle modifiche al registro stesso. Tra questi file particolari abbiamo ad esempio: *system.dat*, *user.dat*, *ntuser.dat*, *default.dat* e altri. Le modifiche a questi file sono la conseguenza delle modifiche fatte a chiavi e valori di registro, esse verranno analizzate in una sezione a seguito.



Figura 23: ProcMon - Name Invalid: |

### 3.6.4 Operazioni sul File System (*sample\_unp\_mod.exe*)

Eseguendo *sample\_unp\_mod.exe* ci sia aspetta che le operazioni rilevanti siano un numero molto maggiore, rispetto all'esecuzione del malware che si blocca sul controllo della data di sistema, in effetti è così. A seguito un report che abbiamo ottenuto riassumendo tutti i log di Procmon, relativi alle operazioni su File System. Spiegheremo quindi tutte le operazioni interessanti che non sono già state trattate nella sezione precedente (poichè presenti anche in *sample.exe*).

Operazioni	Path	Risultato
CF WF QSF SSF	C:\ProgramData\ADService.exe	SUCCESS
CF	C:\ProgramData\ADService.exe.cczzaxxu	SUCCESS
CF	C:\ProgramData\ADService.exe.exe	NAME NOT FOUND
CF SSF	C:\Users\Malware\ADService.exe	SUCCESS
CF	C:\Users\Malware\ADService.exe.exe	NAME NOT FOUND
CF	C:\Users\Malware\ADService.exe.tqvwtttq	SUCCESS
CF WF QSF SSF	C:\Users\Malware\AppData\Local\Temp\ADService.exe	SUCCESS
CF	C:\Users\Malware\AppData\Local\Temp\ADService.exe.exe	NAME NOT FOUND
CF	C:\Users\Malware\AppData\Local\Temp\ADService.exe.xuropmm	SUCCESS
CF WF QSF	C:\Users\Malware\AppData\Local\Temp\I1726961194.bat	SUCCESS
CF WF QSF	C:\Users\Malware\AppData\Local\Temp\K1726961194.bat	SUCCESS
CF QSF SSF	C:\Users\Malware\AppData\Roaming\1726961194	SUCCESS
CF WF QSF SSF	C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe	SUCCESS
CF	C:\Users\Malware\AppData\Roaming\17...\\AdobeARM.exe.Local	NAME NOT FOUND
CF	C:\Users\Malware\AppData\Roaming\KMSpico-setup.exe	SUCCESS
CF	C:\Users\Malware\AppData\Roaming\terra.exe	SUCCESS
CF	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
CF	C:\Windows\Globalization\Sorting\SortDefault.nls	SUCCESS
CF WF QSF SSF	C:\Windows\System32\VBoxTray.exe	SUCCESS
CF	C:\Windows\System32\VBoxTray.exe.jgheeebb	SUCCESS
CF QSF	C:\Windows\System32\cmd.exe	SUCCESS
CF	C:\Windows\System32\cmd.exe.khheefcc	NAME NOT FOUND
CF	C:\Windows\System32\drivers\etc\hosts	SUCCESS
CF	C:\Windows\System32\en-US\propsys.dll.mui	SUCCESS
CF	C:\Windows\System32\en-US\setupapi.dll.mui	SUCCESS
CF	C:\Windows\System32\en-US\tzres.dll.mui	SUCCESS
CF	C:\Windows\explorer.exe	SUCCESS
WF	C:\Users\Malware\NTUSER.DAT	SUCCESS
WF	C:\Users\Malware\ntuser.dat.LOG1	SUCCESS
WF	C:\Windows\System32\config\SOFTWARE.LOG1	SUCCESS
WF	C:\Windows\System32\config\SOFTWARE.LOG1	SUCCESS

[CF = CreateFile, WF = WriteFile, QSF = QuerySecurityFile, SSF = SetSecurityFile]

Nel report, per questioni di spazio, non è stato riportato quale tra le varie replicazioni del malware esegue l'operazione in questione. Tale dato risulta tuttavia ridondante nella maggior parte dei casi, nell'eventualità che non lo sia, verrà specificato a seguito. Le operazioni rilevanti sono:

- CreateFile, QuerySecurityFile e SetSecurityFile di C:\Users\Malware\AppData\Roaming\1726961194
- CreateFile, WriteFile, QuerySecurityFile e SetSecurityFile di C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe

Il malware crea una cartella nascosta (1726961194) dove si sposta subito dopo essere andato in esecuzione, nello spostamento avviene la rinomina in *AdobeARM.exe*. Il nome serve sicuramente a confondere l'utente che è portato a pensare che non si tratti di un programma malevolo, bensì di un componente relativo a un prodotto della nota casa di software. Per verificare che effettivamente si tratti di

un'operazione di rinomina del nostro malware, siamo andati a cercare su ProcMon un evento di tipo SetRenameInformationFile, in effetti quest'ultimo è presente, come si può vedere nella Figura 24a.

- CreateFile, WriteFile, QuerySecurityFile e SetSecurityFile di C:\ProgramData\ADService.exe
- CreateFile e SetSecurityFile di C:\Users\Malware\ADService.exe
- CreateFile, WriteFile, QuerySecurityFile e SetSecurityFile di C:\Users\Malware\AppData\Local\Temp\ADService.exe

Durante l'esecuzione il malware crea un'altra copia che chiama *ADService.exe*, sempre in riferimento alla software house statunitense, tale copia non si trova sempre sulla stessa directory ma viene posta randomicamente in uno dei tre path sopra elencati. Nel report sono presenti tutti e tre i percorsi, poichè avendo provato a terminare il malware dal task manager, l'altra copia del malware in esecuzione crea nuovamente *ADService.exe* in una delle tre directory.

- CreateFile di C:\ProgramData\ADService.exe.cczzaxxu
- CreateFile di C:\Users\Malware\ADService.exe.tqvwtttq
- CreateFile di C:\Users\Malware\AppData\Local\Temp\ADService.exe.xuuroppm

Otteniamo questi file provando a terminare *AdobeARM.exe* dal task manager. Essi corrispondono semplicemente alla copia esatta di *ADService.exe* ma rinominata con l'aggiunta di un'estensione composta da 8 caratteri casuali (Figura 24b). Come già detto precedentemente le motivazioni per cui vengono

(a)

Date:	25/03/2020 15:05:55,8434548
Thread:	2404
Class:	File System
Operation:	SetRenameInformationFile
Result:	SUCCESS
Path:	C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp_mod.exe
Duration:	0.0009757
ReplaceIfExists:	False
FileName:	C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe

(b)

Date:	25/03/2020 15:15:54,2000697
Thread:	908
Class:	File System
Operation:	SetRenameInformationFile
Result:	SUCCESS
Path:	C:\Users\Malware\ADService.exe
Duration:	0.0011360
ReplaceIfExists:	False
FileName:	C:\Users\Malware\ADService.exe.tqvwtttq

Figura 24: ProcMon - (a) Il malware all'inizio dell'infezione si sposta e si rinomina in *AdobeARM.exe*, (b) *ADService.exe* viene rinominato con l'aggiunta di una estensione composta da caratteri casuali

creati non è chiara, sappiamo tuttavia che in caso *AdobeARM.exe* venga terminato forzatamente, viene creato un nuovo file *ADservice.exe* da 2kb e prima che questo avvenga l'eseguibile originale viene rinominato con l'estensione casuale.

- CreateFile e QuerySecurityFile di C:\Users\Malware\AppData\Local\Temp\K1726961194.bat: come per il precedente I1726961194.bat, parleremo di questo script in una apposita sezione successiva.
- CreateFile di C:\Windows\AppPatch\sysmain.sdb: database Shim (sdb) per l'Application Compatibility Toolkit (ACT). Questi database contengono i dettagli specifici su come Windows dovrebbe manipolare (in altre parole "shim") un programma target, con "Correzioni" predefinite. Lo scopo è quello di far funzionare programmi realizzati per versioni datate di Windows anche sulle versioni attuali. L'ACT contiene centinaia di configurazioni per migliaia di programmi [19]. La presenza di questo file può indicare che il sample sia stato realizzato per una versione datata di Windows o che sfrutti l'ACT in modo malevolo per effettuare alcune particolari chiamate a libreria.
- CreateFile di C:\Windows\Globalization\Sorting\SortDefault.nls: online non esistono informazioni precise in merito a questo path ma la directory Globalization sembra contenere degli strumenti utili a Windows per gestire l'interfaccia grafica e le impostazioni sulle lingue di sistema.
- CreateFile, WriteFile, QuerySecurityFile e SetSecurityFile di C:\Windows\System32\VBoxTray.exe
- CreateFile di C:\Windows\System32\VBoxTray.exe.jgheeebb

Questi due eventi sono probabilmente la causa della terminazione del processo relativo a VBox Guest Additions che avevamo notato nella fase iniziale di osservazione. Come ha fatto per la sua copia *ADService.exe*, anche in questo caso il malware rinomina l'eseguibile target con la nuova estensione (*VBoxTray.exe.jgheeebb*), lasciando al suo posto un piccolo file da due 2kb circa (*VBoxTray.exe*). Questo provoca probabilmente la terminazione del processo stesso che sparisce dalla System Tray. L'ipotesi è che il nostro sample esegua questa operazione ricorrente per chiudere i processi che potrebbero essere una minaccia al suo funzionamento, nascondere l'eseguibile in questione tramite una rinomina, potrebbe essere infatti una tecnica per impedirne il riavvio.

- CreateFile e QuerySecurityFile di C:\Windows\System32\cmd.exe
  - CreateFile di C:\Windows\explorer.exe
- Controllando i componenti di Windows in questione con PEStudio, si è verificato che gli hash degli eseguibili non cambiano prima e dopo l'esecuzione del nostro sample, ciò ci dice che i due file non vengono modificati. Eseguendo il malware come amministratore, la cosa non cambia.
- CreateFile di C:\Windows\System32\en-US\propsys.dll.mui
  - CreateFile di C:\Windows\System32\en-US\setupapi.dll.mui
  - CreateFile di C:\Windows\System32\en-US\tzres.dll.mui

Anche in questo caso online non esistono informazioni precise su questi file. L'estensione *mui* (Multilingual User Interface) serve a indicare una tipologia di file utilizzata per il contenimento di risorse utili a gestire l'interfaccia di Windows quando si necessita l'implementazione di lingue diverse.

### 3.6.5 Operazioni sul Registro di Sistema (*sample.exe*)

Per le operazioni sul registro dobbiamo usare i seguenti filtri:

- **Operation - is - RegOpenKey:** apre una chiave di registro esistente.
- **Operation - is - RegCreateKey:** crea una nuova chiave di registro.
- **Operation - is - RegSetValue:** crea o sovrascrive uno specifico valore di una chiave di registro.
- **Operation - is - RegFlushKey:** scrive tutti gli attributi della chiave corrente nel registro.

Come nel caso del File System anche qui le operazioni più interessanti sono quelle relative alle modifiche di registro (RegCreateKey, RegSetValue)

- RegCreateKey di `HKLM\System\CurrentControlSet\Services\Tcpip\Parameters`: è una chiave che contiene dei valori che rappresentano tutti i parametri TCP/IP della connessione di rete Windows.
- RegOpenKey di `HKLM\System\CurrentControlSet\services\Disk\Enum`
- RegOpenKey e RegSetValue di `HKLM\System\CurrentControlSet\services\Disk\Enum\0` con valore `xobv`: la chiave contiene sostanzialmente il nome e l'ID seriale del disco primario del computer. L'accesso e la modifica di questa chiave è una nota tecnica per rilevare se l'esecuzione sti avvenendo in una macchina virtuale [15]. I caratteri del valore non sono a caso ma comunemente usati dai software di gestione delle macchine virtuali: `awmw`, `xobv` e `umeq`, in altre parole fanno riferimento a VMware, VirtualBox e QEMU.
- RegCreateKey di `HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zone Map\`
- RegSetValue di `HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zone Map\UNCAsIntranet` e valore 0: `UNCAsIntranet` è un'impostazione che permette di includere o meno tutti i path di rete. Usata da Internet Explorer.
- RegSetValue di `HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zone Map\AutoDetect` e valore 1: `AutoDetect` sta per l'impostazione “Automatically detect intranet network” e come la chiave precedente è un settaggio usato principalmente da Internet Explorer.
- RegCreateKey, RegOpenKey e RegFlushKey di `HKCU\Software\Microsoft\Windows\Current Version\RunOnce`: su Windows le chiavi di registro `Run` e `RunOnce`, posizionate sia in `HKLM` che in `HKCU`, fanno sì che i programmi vengano eseguiti ogni volta che un utente accede al computer. In modo predefinito, nel caso di una chiave `RunOnce`, il valore viene eliminato prima dell'esecuzione della riga di comando. Si tengano presente queste informazioni anche per le successive chiavi che riguardano `RunOnce`.
- RegSetValue di `HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194` e valore "`|`" (virgolette incluse), visibile in Figura 25. Per impostazione predefinita, la chiave viene ignorata all'avvio del computer in modalità provvisoria, tuttavia il nome delle chiavi `RunOnce` preceduto da un asterisco (\*) forza l'esecuzione del programma anche in tale modalità. Come si può notare dal nome della chiave, è questo il nostro caso. Il nome della chiave (nel nostro caso il numero di dieci cifre preceduto dall'asterisco) è puramente descrittivo, mentre il valore è la linea di comando che verrà eseguita all'avvio di Windows. Qui nasce un problema, poiché la barra verticale, impostata come valore dal nostro sample, non ha alcun significato, nè funzione: per dirlo in modo diverso `|` in un comando DOS sarebbe la pipe ma presa singolarmente non ha alcun valore. Tale conclusione è confermata anche dal risultato visibile in Figura 26, proposto dal software Autoruns che ci fa notare l'inesistenza di un file con tale path.
- RegSetValue di `HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194` e valore "`|`" (virgolette incluse): come la precedente ma su `HKLM` invece di `HKCU`.

Oltre a queste modifiche al registro ci sono anche molti accessi in lettura (RegOpenKey) usate per verificare le impostazioni correnti del sistema. Riproponiamo a seguito quelli che sembrano essere più interessanti.

- RegOpenKey di `HKLM\Software\Microsoft\Windows NT\CurrentVersion\....`: varie chiavi che contengono informazioni sul computer e sul sistema operativo.
- RegOpenKey di `HKCU\Software\Classes\.bat`
- RegOpenKey di `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.bat`

- RegOpenKey di HKCU\Software\Classes\batfile  
Queste tre e molte altre chiavi simili indicano che il sample opera su file batch.
- RegOpenKey di HKLM\System\CurrentControlSet\Services\Disk\Enum: un elenco di dischi installati con i relativi ID e altre informazioni.
- RegOpenKey di HKLM\Software\Policies\Microsoft\Windows NT\DnsClient
- RegOpenKey di HKLM\System\CurrentControlSet\Services\DNS  
Ci permettono di capire che il sample esegue anche operazioni riguardanti i DNS.
- RegOpenKey di HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Interfaces
- RegOpenKey di HKLM\System\CurrentControlSet\Services\Tcpip\Linkage
- RegOpenKey di HKLM\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore

Durante l'analisi abbiamo utilizzato l'interfaccia standard al registro di sistema di Windows (*regedit*) per verificare le modifiche fatte dal malware.

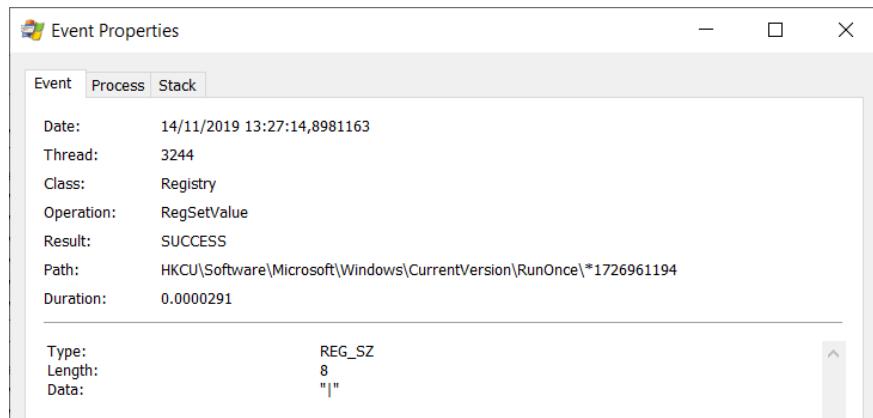


Figura 25: ProcMon - Aggiunta di un valore a RunOnce

The screenshot shows the 'Autoruns' tool interface. The main window displays a list of running tasks:

Autostart Entry	Description	Publisher	Image Path	Timestamp	VirusTotal
<input checked="" type="checkbox"/> VBoxTray	VirtualBox Guest Additions Tray Appl... (Verified)	Oracle Corporation	c:\windows\system32\vboxtray.exe	10/10/2019 18:46	
<input checked="" type="checkbox"/> HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce				20/12/2019 10:05	
<input checked="" type="checkbox"/> *1726961194				File not found:	
<input checked="" type="checkbox"/> HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce				20/12/2019 10:05	
<input checked="" type="checkbox"/> *1726961194				File not found:	
<input checked="" type="checkbox"/> C:\Users\Malware\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup				06/12/2016 14:19	
<input checked="" type="checkbox"/> Drive.bat			c:\users\malware\appdata\roaming\...	06/12/2016 14:19	

Figura 26: Autoruns - File not found: |

### 3.6.6 Operazioni sul Registro di Sistema (*sample\_unp\_mod.exe*)

- [RegCreateKey] -	
Chiave	Risultato
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce	SUCCESS (EXISTING KEY)
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	SUCCESS (EXISTING KEY)
HKLM\System\CurrentControlSet\Services\Tcpip\Parameters	SUCCESS (EXISTING KEY)
HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS (EXISTING KEY)
HKCU\Software\Mic...\\CurrentVersion\Internet Settings\ZoneMap\	SUCCESS (EXISTING KEY)

- [RegSetValue] -	
Chiave	Risultato
HKCU\Software\Microsoft\Windows\CurrentVersion\Hidden	
--> Value = 2	SUCCESS
HKCU\Software\Mic...\\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	
--> Value = 1	SUCCESS
HKCU\Software\Mic...\\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	
--> Value = 0	SUCCESS
HKCU\Software\Mic...\\CurrentVersion\Policies\Explorer\NoFolderOptions	
--> Value = 1	SUCCESS
HKCU\Software\Mic...\\CurrentVersion\Policies\Explorer\NoWindowsUpdate	
--> Value = 1	SUCCESS
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194	
--> Value = "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"	SUCCESS
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194	
--> Value = "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"	SUCCESS
HKLM\System\CurrentControlSet\services\Disk\Enum\0	
--> Value = xobv	
HKLM\System\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations	
--> Value = \??\C:\Program Files\Google\Chrome, ,	
\??\C:\Windows\system32\VBoxTray.exe.jgheeebb, ,	
\??\C:\Windows\system32\VBoxTray.exe, ,	
\??\C:\Program Files\Google\Update\1.3.35.442, ,	
\??\C:\Windows\system32\cmd.exe.khheefcc, ,	
\??\C:\Users\Malware\ADService.exe.tqvwtttq, ,	
\??\C:\Users\Malware\ADService.exe, ,	
\??\C:\ProgramData\ADService.exe.cczzaxxu, ,	
\??\C:\ProgramData\ADService.exe, ,	
\??\C:\Users\Malware\AppData\Local\Temp\ADService.exe.xuropmm, ,	
\??\C:\Users\Malware\AppData\Local\Temp\ADService.exe, ,	
\??\C:\Windows\system32\cmd.exe.urrroolm, ,	
\??\C:\Users\Malware\AppData\Local\Temp\ADService.exe.hemmjgje, ,	
\??\C:\Users\Malware\AppData\Local\Temp\ADService.exe, ,	
\??\C:\ProgramData\ADService.exe.kkkhhefz, ,	
\??\C:\ProgramData\ADService.exe, ,	
	SUCCESS
- [RegFlushKey] -	
Chiave	Risultato
HKCU\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce	SUCCESS
HKLM\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS

HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	SUCCESS
- [RegOpenKey] -	
Chiave	Risultato
HKCR\batfile\shell\open\command	SUCCESS
HKCR\SystemFileAssociations\.bat	NAME NOT FOUND
HKCU\Software\Classes\batfile\...	
HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\...	
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Linkage	SUCCESS
HKLM\SYSTEM\CurrentControlSet\Services\Disk\Enum	SUCCESS
HKLM\System\CurrentControlSet\Services\DNS	NAME NOT FOUND
HKLM\Software\Policies\Microsoft\System\DNSClient	NAME NOT FOUND
HKLM\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore	NAME NOT FOUND
HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\K1726961194.bat	NAME NOT FOUND
HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\I1726961194.bat	NAME NOT FOUND
HKLM\Software\Microsoft\Windows NT\CurrentVersion\...	
HKLM\Software\Mic...\\CurrentVersion\AppCompatFlags\Custom\K1726961194.bat	NAME NOT FOUND
HKLM\SOFTWARE\Mic...\\CurrentVersion\AppCompatFlags\Custom\AdobeARM.exe	NAME NOT FOUND
HKLM\Software\Mic...\\CurrentVersion\AppCompatFlags\Custom\ADService.exe	NAME NOT FOUND

Proseguiamo ora all’analisi delle operazioni su registro fatte da *sample-unp-mod.exe*. Al solito verranno proposti a seguito solo gli eventi che non erano presenti nell’analisi di *sample.exe* o che hanno subito delle variazioni rispetto a quest’ultimo.

- RegSetValue di HKCU\Software\Microsoft\Windows\CurrentVersion\Hidden con valore 2 (Figura 27a): tale chiave di registro serve a gestire la visualizzazione di file e cartelle nascosti su Esplora Risorse di Windows, in particolare può assumere i due valori 1 o 2, rispettivamente per:
  - 1 → visualizzare file e cartelle nascoste
  - 2 → non visualizzare file e cartelle nascoste

Osservando il valore impostato dall’evento corrente siamo nel secondo caso. La motivazione è sicuramente dovuta al fatto che il malware vuole nasconde i suoi file e le sue directory (che hanno l’attributo *Hidden*) all’utente. In alcuni casi, all’avvio del malware, i file nascosti sono ancora visibili su Esplora Risorse, questo è dovuto al fatto che tale componente di Windows non aggiorna subito le impostazioni dal registro, sarà infatti necessario aspettare qualche minuto o riavviare *explorer.exe* per vedere applicate le impostazioni scelte dal sample.

- RegSetValue di HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\NoFolderOptions con valore 1 (Figura 27b): questa chiave di registro serve ad abilitare o disabilitare la voce di menù “Opzioni Cartella e Ricerca” presente su Esplora Risorse di Windows, in particolare può assumere i due valori 0 o 1:
  - 0 → per abilitare la voce di menù
  - 1 → per disabilitare la voce di menù

Il nostro sample imposta il valore 1, ciò significa che per l’utente non sarà più possibile modificare le impostazioni contenute nella suddetta voce. In particolare non ci sarà più la possibilità di riattivare la visualizzazione file e cartelle nascoste direttamente da Esplora Risorse.

- RegSetValue di HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\NoWindowsUpdate con valore 1 (Figura 27c): permette di bloccare l’accesso a Windows Update per l’utente corrente. In un uso corretto la chiave può essere sfruttata dall’amministratore di sistema per impedire che un utente della macchina possa interagire in modo errato con il sistema di aggiornamenti del SO. Essa può assumere due valori: [23]

- 0 → (o chiave non presente nel registro). La policy è disabilitata o non configurata. L’utente può utilizzare Windows Update.
- 1 → La policy è abilitata. L’utente non può utilizzare Windows Update.

Nel nostro caso, impostando il valore 1, il malware blocca la possibilità di aggiornare la macchina all’utente corrente, in tale modo cerca di evitare che update futuri al SO danneggino il suo operato.

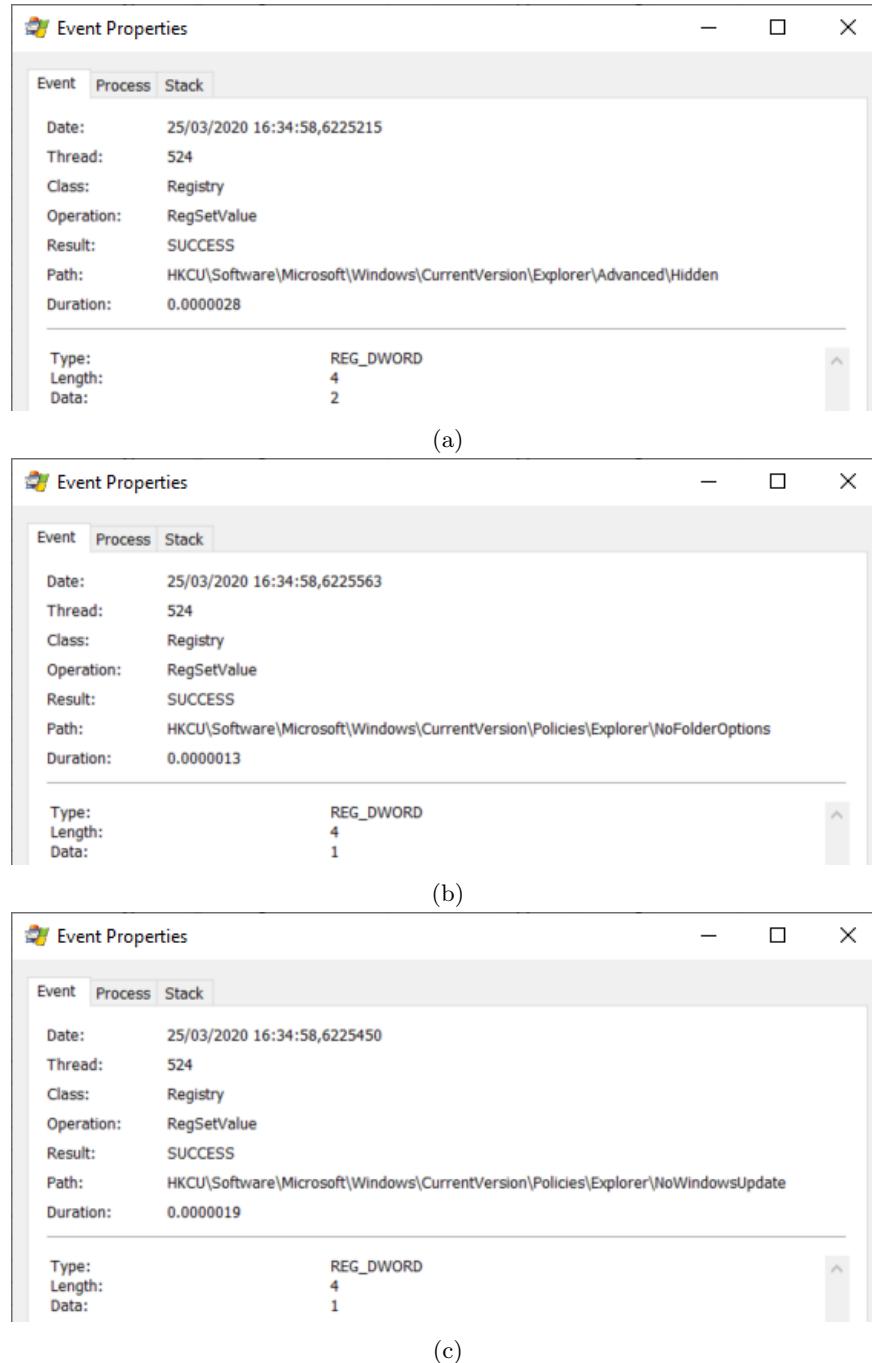
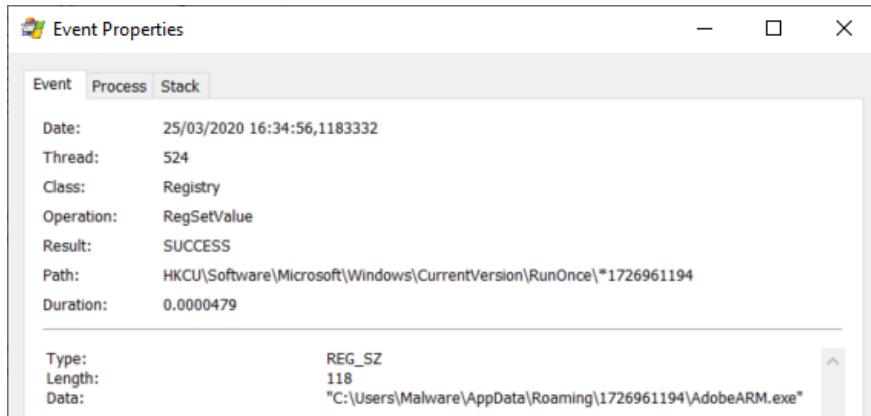


Figura 27: ProcMon - Il malware disabilita la visualizzazione di file e cartelle nascoste (a), le opzioni cartelle e ricerca (b) e Windows Update (c)

- RegSetValue di HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\\*1726961194 con valore "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe" (virgolette incluse): Abbiamo già spiegato con *sample.exe* la funzione di questa chiave di registro. Tuttavia *sample\_unp\_mod.exe*, a differenza della versione bloccata dal controllo sulla data di sistema, tramite questa chiave effettua la persistenza in maniere corretta. Il valore corrisponde infatti al path di *AdobeARM.exe* (Figura 28a), che avevamo già osservato in una sezione precedente parlando delle operazioni del malware su File System. Al riavvio del sistema, Windows esegue *AdobeARM.exe*, che a sua volta avvierà la sua seconda copia *ADService.exe*. Come si può vedere in Figura 28b, anche Autoruns ci mostra la presenza del bot tra i programmi avviati in automatico dal SO.
- RegSetValue di HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce\\*1726961194 con valore "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe" (virgolette incluse): come la precedente ma su HKLM invece di HKCU.
- RegSetValue di HKLM\System\CurrentControlSet\Control\Session Manager\PendingFileRename Operations con il valore molto lungo indicato nel report: tale chiave può consentire l'eliminazione, lo spostamento e la ridenominazione dei file al riavvio della macchina. In particolare tali operazioni vengono eseguite all'inizio del processo di avvio, quindi possono essere completate prima che i programmi possano bloccare i file [12].

Citando un articolo che parla in merito alla chiave “PendingFileRenameOperations” [12], eventuali malware potrebbero utilizzare questa chiave nei seguenti modi:

- Tentare di eliminarsi al successivo riavvio, ma rimanere in esecuzione sul computer fino a tale momento.
- Cambiare il nome e/o la posizione del file, in combinazione con la modifica del meccanismo di persistenza.



(a)

The screenshot shows the 'Autoruns' tool interface. The table lists registry entries under the 'RunOnce' key:

Autourun Entry	Description	Publisher	ImagePath	Timestamp
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\AlternateShell			c:\windows\system32\cmd.exe	14/07/2009 04:37
<input checked="" type="checkbox"/> cmd.exe	Windows Command Processor	(Verified) Microsoft Windows	c:\windows\system32\cmd.exe	20/11/2010 09:00
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce			c:\users\malware\appdata\roaming\1726961194\adobearm.exe	25/03/2020 14:01
<input checked="" type="checkbox"/> 1726961194			c:\users\malware\appdata\roaming\1726961194\adobearm.exe	30/08/2013 04:17
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce			c:\users\malware\appdata\roaming\1726961194\adobearm.exe	25/03/2020 14:01
<input checked="" type="checkbox"/> 1726961194			c:\users\malware\appdata\roaming\1726961194\adobearm.exe	30/08/2013 04:17
<input checked="" type="checkbox"/> C:\Users\Malware\AnnData\Roaming\Microsoft\Windows\Start Menu\Programs\Start in				06/12/2016 14:19

(b)

Figura 28: ProcMon (a), Autoruns (b) - Persistenza del malware tramite chiave RunOnce

- Un file programma o .dll, malevolo o meno, può scambiarsi con un altro file, usando due funzioni di rinomina; una per rinominare il file valido e un’altra per rinominare un file errato con il nome del vecchio file valido.
- I malware possono “rompere” un software legittimo eliminando/rinominando un file dipendente.

Prima di parlare delle modifiche fatte dal nostro sample, dobbiamo capire come avvengono le operazioni di ridefinizione ed eliminazione file attraverso questo registro [12] [3]. L’aggiunta di una voce a “PendingFileRenameOperations” è complicata perché è un valore `REG_MULTI_SZ` non la normale stringa o la `DWORD`. Per questo motivo se volessimo modificare manualmente tale chiave, esistono due procedimenti distinti: uno per lo spostamento del file, l’altro per l’eliminazione. Per aggiungere una voce relativa a una ridefinizione/spostamento di file, si procede a inserire sul valore della chiave di registro il percorso completo del file, precedendolo da `\??\` (e.g. `\??\C:\source\programma.exe`). Successivamente, in una nuova riga subito sotto a quest’ultima, va inserito il path di destinazione, sempre preceduto da `\??\` (e.g. `\??\C:\target\programma.exe`). Se invece volessimo eliminare un file al riavvio, il procedimento è leggermente diverso. Dopo aver inserito il path sorgente, preceduto da `\??\`, bisogna inserire i valori esadecimali `00 00` sulla successiva riga. Modificando manualmente la chiave, questo non è possibile dalla visualizzazione stringa. Per inserire i quattro zeri bisognerebbe quindi aprire la visualizzazione binaria del valore della chiave. È possibile inserire più operazioni sulla chiave, una sotto l’altra, ogni coppia di linee presenti sul valore corrispondono a una singola operazione che si desidera pianificare al riavvio (Figura 29a). Da notare che aprendo la finestra relativa alla visualizzazione binaria dei dati, le operazioni di eliminazione avrebbero otto zeri finali (`00 00 00 00`) e non quattro (Figura 29b). Ciò è dovuto al fatto che il codice esadecimale `00 00` sta anche per “new line”. I primi quattro zeri saranno dunque relativi alla nuova linea che segue il path del file, i successivi quattro indicano che quest’ultimo deve essere eliminato. Al riavvio, dopo aver eseguito l’operazione specificata, il SO elimina il valore o direttamente la chiave “PendingFileRenameOperations” che dovrà essere ricreata per pianificare nuove operazioni.

Conoscendo l’utilizzo e il funzionamento di questa chiave, risulta ora semplice capire le operazioni pianificate dal nostro malware. ProcMon separa le nuove righe presenti sul valore con il carattere “virgola”, perciò quando troviamo “`,` ,” significa che è presente una linea vuota, ciò indica che come spiegato sopra, il path precedente sarà relativo a un file da eliminare. Osservando il report relativo agli eventi su registro, notiamo che le operazioni impostate dal malware sono tutte eliminazioni di file. Il nostro sample chiede la cancellazione dei file che ha creato con l’estensione casuale (e.g. `\??\C:\Windows\system32\VBoxTray.exe.jgheebb, ,`), di tutti i file `ADService.exe` nei diversi path, di `VBoxTray.exe` e di alcune directory relative al browser Chrome. Tenendo in considerazione le motivazioni per cui un malware solitamente utilizza questa chiave di registro (elencate precedentemente), l’ipotesi è che queste operazioni servano al nostro sample sia per eliminare al riavvio programmi che ostacolano il suo funzionamento, che per eliminare la sua copia `ADService.exe`. Quest’ultima, servendo solo a garantire la non terminazione di `AdobeARM.exe`, può essere eliminata senza problemi in quanto al riavvio verrà ricreata nuovamente.

Il sample non inserisce immediatamente tutti i path nella chiave di registro, questi ultimi vengono aggiunti progressivamente. In Figura 30 si può notare che durante l’esecuzione del malware, man mano che il tempo passa, il valore della chiave “PendingFileRenameOperations” diventa sempre più lungo, poichè conterrà sempre più operazioni di eliminazione.

Concludendo questa sezione, nel report sono presenti molti eventi di tipo `RegOpenKey` con risultato “NAME NOT FOUND”. Questo può essere un altro indicatore che il malware è stato realizzato per versioni date di Windows. L’ipotesi è che tali risultati siano dovuti al fatto che su Windows 7 e successivi, le chiavi coinvolte non siano più presenti nel registro di sistema.

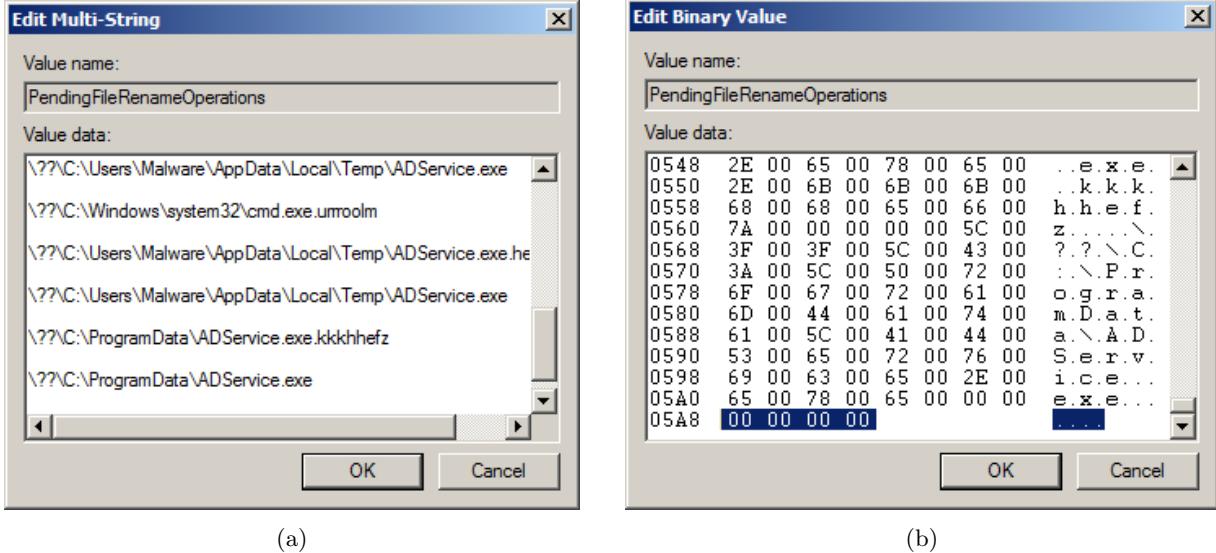


Figura 29: Regedit - Chiave “PendingFileRenameOperations” in modifica stringa (a) e in modifica dati binari (b).

Path	Detail
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 168, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb..
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 244, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 422, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.., \??\C:\Pro
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 512, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.., \??\C:\Pro
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 584, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.., \??\C:\Pro
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 670, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.., \??\C:\Pro
HKLM\System\CurrentControlSet\Control..	Type: REG_MULTI_SZ, Length: 738, Data: \??\C:\Program Files\Google\Chrome.., \??\C:\Windows\system32\VBoxTray.exe.jhheeb.., \??\C:\Windows\system32\VBoxTray.exe.., \??\C:\Pro

Figura 30: Procmon - Chiave PendingFileRenameOperations

### 3.6.7 Operazioni di creazione altri Processi e Thread (*sample.exe*)

Per l’analisi su processi e thread dobbiamo usare i seguenti filtri:

- **Operation - is - Process Create:** per la creazione di un processo.
- **Operation - is - Thread Create:** per la creazione di una thread.
- **Operation - is - Process Exit:** per la terminazione di un processo.
- **Operation - is - Thread Exit:** per la terminazione di una thread.

I risultati ottenuti ci indicano che durante l’esecuzione del malware è stato generato un altro processo e sono state invece create moltissime thread. Da un’analisi approfondita è risultato che a numeri siamo su 60 thread/secondo, la maggior parte delle quali rimangono in vita per pochi decimi di secondo, se non in alcuni casi addirittura centesimi. Risulta invece essere molto interessante il *Process Create*, visibile in Figura 31a, fatto tramite il comando C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\I1726961194.bat". Si tratta dell’esecuzione dello script batch che era stato creato dal sample precedentemente (*WriteFile* in una precedente sezione). Sul file I1726961194.bat vengono effettuate anche operazioni di SetEndOfFileInformationFile e di QuerySecurityFile. Discuteremo dei file batch in seguito in una apposita sezione. Potremmo controllare anche se sono presenti operazioni di rete ma per quelle è meglio usare altri strumenti appropriati. Durante l’analisi su ProcMon sono stati provati altri filtri, tuttavia qui sono stati riportati solo quelli che effettivamente producevano dei risultati rilevanti rispetto alle operazioni apportate dal sample.

### 3.6.8 Operazioni di creazione altri Processi e Thread (*sample\_unp\_mod.exe*)

Nel caso di *sample\_unp\_mod.exe* la situazione relativa alle thread create non cambia molto rispetto a *sample.exe* (aumenta leggermente il numero di thread create in un secondo), tuttavia come si può vedere in Figura 32, ci sono delle nuove operazioni di tipo *Process Create*, analizzate qui a seguito.

- Il processo creato tramite il file **I1726961194.bat** (Figura 31a) era già presente in *sample.exe*.
- Vi è una nuova creazione di processo simile alla precedente, relativa al file **K1726961194.bat** (Figura 31b), questo processo viene avviato tramite il comando `C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\K1726961194.bat"`.
- Avvio del processo *ADService.exe*. Ci sono più eventi con la variazione sul path dove si può trovare questa copia del malware. Questo avviene poiché, avendo tentato di terminare il sample, *AdobeARM.exe* ha creato *ADService.exe* in directory diverse. Avendo già osservato in precedenza l'operato del nostro bot e delle relative copie, ci aspettavamo questi eventi.
- Creazione del processo *AdobeARM.exe* nel caso sia stato precedentemente terminato dal task manager.

(a)

(b)

Event Type	Date	Thread	Class	Operation	Result	Path	PID	Command Line
Process Create	25/03/2020 16:05:55,9333351	2412	Process	Process Create	SUCCESS	C:\Windows\system32\cmd.exe	3832	C:\Windows\system32\cmd.exe /c ""C:\Users\Malware\AppData\Local\Temp\I1726961194.bat""
Process Create	25/03/2020 16:05:55,9289610	2444	Process	Process Create	SUCCESS	C:\Windows\system32\cmd.exe	1844	C:\Windows\system32\cmd.exe /c ""C:\Users\Malware\AppData\Local\Temp\K1726961194.bat""

Figura 31: ProcMon - Processi e Thread, particolare di esecuzione di due linee di comando: una relativa alla creazione di **I1726961194.bat** (a), e una relativa alla creazione di **K1726961194.bat** (b).

\Marco\Desktop\Logfile3.PML				
File	Options	Help		
Process Name	PID	Operation	Path	Detail
sample_unp_-	2252	Process Create	C:\Windows\system32\cmd.exe	PID: 1844, Command line: C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\K1726961194.bat"
sample_unp_-	2252	Process Create	C:\Windows\system32\cmd.exe	PID: 3832, Command line: C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\I1726961194.bat"
AdobeARM.exe	3924	Process Create	C:\Users\Malware\ADService.exe	PID: 2556, Command line: C:\Users\Malware\ADService.exe
AdobeARM.exe	3924	Process Create	C:\Windows\system32\cmd.exe	PID: 3164, Command line: C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\I1726961194.bat"
ADService.exe	2556	Process Create	C:\Users\Malware\AppData\Roaming\I1726961194\AdobeARM.exe	PID: 2364, Command line: C:\Users\Malware\AppData\Roaming\I1726961194\AdobeARM.exe
AdobeARM.exe	2364	Process Create	C:\Windows\system32\cmd.exe	PID: 1692, Command line: C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\K1726961194.bat"
AdobeARM.exe	2364	Process Create	C:\Windows\system32\cmd.exe	PID: 3344, Command line: C:\Windows\system32\cmd.exe /c "C:\Users\Malware\AppData\Local\Temp\I1726961194.bat"
AdobeARM.exe	2364	Process Create	C:\ProgramData\ADService.exe	PID: 3640, Command line: C:\ProgramData\ADService.exe

Figura 32: ProcMon - Processi e Thread

### 3.7 Analisi dinamica sul file da 2kb

Dall'analisi fatta con PEStudio, il file da 2kb non sembra aver le potenzialità per compiere comportamenti nocivi, e in generale sulla carta non sembra poter svolgere azioni rilevanti di alcun tipo. In ogni caso per toglierci il dubbio abbiamo svolto con ProcMon l'analisi del piccolo file. I risultati, proposti qui a seguito, sono quelli che ci aspettavamo.

- Caricamento di poche librerie standard.
- Poche letture sul file system, scrittura di 2 file temporanei che risultano essere vuoti:
  - CreateFile e WriteFile di C:\Users\Malware\AppData\Local\Temp\scs2083.tmp
  - CreateFile e WriteFile di C:\Users\Malware\AppData\Local\Temp\scs62.tmp
- Nessuna scrittura sul registro di sistema, poche letture, nessuna delle quali interessante.
- Nessun processo creato, poche thread istanziate.
- Nessuna operazione di rete.

### 3.8 Script batch generati

Il malware durante la sua esecuzione crea degli script .bat che hanno varie funzioni, più o meno chiare. Durante l'analisi con ProcMon abbiamo osservato la creazione di I1726961194.bat e di K1726961194.bat. In questa sezione parleremo anche di un terzo file, U1726961194.bat, che è stato catturato in un preciso momento dell'esecuzione del nostro sample, riconducibile alla disinstallazione del malware.

Per catturare i file e vederne il contenuto si può fare in parte riferimento a *VirusTotal* (che mostra il contenuto del primo file batch) oppure procedere attraverso il Reverse Engineering. Nel nostro caso, visto che tutti e tre i file vengono generati nella medesima cartella temporanea, è stato utilizzato primariamente uno script appositamente creato per fare i backup dei file con estensione .bat che appaiono in qualsiasi momento in tale directory. Tale precauzione è stata messa in piedi perchè, come vedremo, i file batch alla fine dell'esecuzione eliminano se stessi. Lo script, proposto in Figura 33, contiene quindi un comando che copia i file con estensione .bat dalla cartella "C:\Users\Malware\AppData\Local\Temp" alla directory "C:\Users\Malware\Desktop\BackupDat".

```

@ECHO OFF
cd C:\Users\Malware\AppData\Local\Temp\
REM Si tratta della cartella in cui cercare i file .bat da salvare
:loop
FOR %%f IN (*.bat) DO XCOPY C:\Users\Malware\AppData\Local\Temp\%%f"%
C:\Users\Malware\Desktop\BackupDat\ /m /y
goto loop

```

Figura 33: Script creato appositamente per catturare i file batch

### 3.8.1 File I1726961194.bat

```
@echo off  
DEL %0>NUL
```

Figura 34: I1726961194.bat

Iniziamo con il discutere del primo file, visibile in Figura 34. Per impostazione predefinita, un file batch visualizza i comandi e i relativi risultati durante l'esecuzione. Lo scopo del primo comando, `@echo off`, è di disattivare questo display. Il secondo comando, `DEL %0`, consente invece allo script di eliminare se stesso, grazie a `>NUL` eventuali errori verranno soppressi. Nonostante l'apparente inutilità di questi script, l'ipotesi più attendibile è che lo scopo sia probabilmente quello di testare la possibilità di scrivere su un determinato percorso oppure la capacità di eseguire comandi terminale tramite la libreria “*shell32*” citata nelle precedenti sezioni.

### 3.8.2 File K1726961194.bat

```
@echo off  
TASKKILL /F /T /PID 1564>NUL  
DEL %0>NUL
```

Figura 35: K1726961194.bat

Il secondo script, in Figura 35, contiene i comandi `@echo off` e `DEL %0>NUL` presenti anche nel precedente file batch, ma in più consente al malware di terminare un processo tramite il comando `TASKKILL`. In particolare:

- Il parametro `\F` indica che la terminazione del processo deve avvenire forzatamente, mentre il parametro `\T` indica che oltre al processo in questione devono essere terminati anche tutti i processi figli di quest'ultimo.
- `\PID` indica che si vuole “uccidere” il processo specificato tramite il suo PID ID, il numero che segue questo parametro è l'identificativo in questione.
- `>NUL` indica al solito che eventuali errori generati durante l'esecuzione del comando dovranno essere soppressi.

Resta soltanto da capire a che processo si riferisca il PID del comando. Facendo un'ulteriore verifica con il task manager ci siamo accorti che l'identificativo di processo è quello di *ADService.exe*. In particolare accade quanto segue: quando l'utente termina *AdobeARM.exe*, la copia *ADService.exe* in esecuzione rimette in esecuzione il primo. Quest'ultimo esegue lo script *K1726961194.bat*, ma diversamente da quanto ci si potrebbe aspettare, *ADService.exe* non viene terminato.

### 3.8.3 File U1726961194.bat

```
@echo off  
DEL C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe>NUL  
:REPEAT  
DEL C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe>NUL  
if exist C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe goto REPEAT>NUL  
DEL %0>NUL
```

Figura 36: U1726961194.bat

L'ultimo script, in Figura 36, contiene sempre i comandi `@echo off` e `DEL %0>NUL`, nonostante sembri complesso, gli altri comandi non fanno nient'altro che eliminare l'eseguibile `AdobeARM.exe` dalla relativa cartella. L'unica accortezza è il continuare a riprovare l'eliminazione nel caso in cui l'operazione fallisca. Quando il file `AdobeARM.exe` non è più presente lo script batch termina. Per ora abbiamo solamente capito il funzionamento del file `U1726961194.bat`, le condizioni di creazione e le motivazioni legate all'utilizzo di quest'ultimo verranno spiegate in una sezione successiva.

### 3.9 RegShot

**Definizione.** *RegShot è una piccola utility che permette di creare un'istantanea del registro di sistema di Windows, in modo da poterla paragonare con un'altra precedentemente effettuata. Si tratta quindi di un utile programma che permette di verificare, ad esempio, quali modifiche sono state apportate al registro di sistema dopo l'installazione di un nuovo software. Il report delle differenze tra un'immagine e l'altra può essere generato in formato testo puro oppure in formato HTML. E' possibile verificare non solo le modifiche sul registro di sistema, ma anche su cartelle specificate. [10]*

Tramite l'uso di RegShot si è verificato che nella nostra analisi non manchino importanti operazioni che potrebbero esserci sfuggite tramite l'analisi con ProcMon. Per funzionare correttamente il programma necessita che vengano acquisite due immagini del registro di sistema. La prima deve essere effettuata subito prima di avviare il malware, la seconda quando riteniamo che il sample abbia completato le operazioni che ci interessa monitorare sul sistema.

#### 3.9.1 Report per `sample.exe`

```
Values added:3
HKL\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194: ""|"""
HKS-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-917
HKS-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194: ""|"""

Values modified:12
HKL\SYSTEM\ControlSet001\services\Disk\Enum\0: "IDE\DiskVBOX_HARDDISK" 1.0 \5&33d1638a&0&0.0.0"
HKL\SYSTEM\ControlSet001\services\Disk\Enum\0: "xobv"
HKL\SYSTEM\CurrentControlSet\services\Disk\Enum\0: "IDE\DiskVBOX_HARDDISK" 1.0 \5&33d1638a&0&0.0.0"
HKL\SYSTEM\CurrentControlSet\services\Disk\Enum\0: "xobv"

Files [attributes?] modified:6
C:\Users\Malware\AppData\Local\Microsoft\Windows\UsrClass.dat
C:\Users\Malware\AppData\Local\Microsoft\Windows\UsrClass.dat.LOG1
C:\Users\Malware\NTUSER.DAT
C:\Users\Malware\ntuser.dat.LOG1
C:\Windows\System32\config\SOFTWARE
C:\Windows\System32\config\SOFTWARE.LOG1

Folders added:1
C:\Users\Malware\AppData\Roaming\1726961194

Folders attributes changed:4
C:\ProgramData
C:\Users\All Users\
C:\Users\All Users\..
C:\Users\Malware\AppData\Local\Temp
```

Figura 37: RegShot - Report `sample.exe` (editato)

Nel caso di `sample.exe`, gli eventi che avevamo sottolineato precedentemente come rilevanti risultano gli stessi evidenziate da RegShot. In Figura 37 si può vedere il report html generato. Si tenga tuttavia conto che il file html è stato modificato, rimuovendo i registri superflui, per poter essere qui riproposto. Nel report sono dunque presenti: le informazioni che identificano il tentativo di creare persistenza su `RunOnce`, gli accessi relativi alle informazioni del disco primario e la creazione della cartella vuota sulla directory `AppData\Roaming` dell'utente. Le uniche informazioni su cui non si era posta particolare attenzione prima

sono quelle relative alle modifiche degli attributi di alcune cartelle (come si può vedere dalle ultime righe del report). Tuttavia, verificando con procmon di cosa si trattasse nello specifico, sono risultati essere eventi di poca importanza: alcune modifiche della data/ora di ultimo accesso/modifica.

### 3.9.2 Report per *sample\_unp\_mod.exe*

Anche analizzando il report di ProcMon relativo alla versione del sample modificata con il bypass sulla data del sistema, abbiamo constatato di aver già analizzato tramite ProcMon tutti gli eventi rilevanti. Sono presenti infatti le modifiche alle chiavi **Hidden**, **NoWindowsUpdate**, **NoFolderOptions** e **RunOnce**, la creazione dei file **AdobeARM.exe**, **ADService.exe** e di quelli con l'estensione randomica. L'unico dettaglio che non abbiamo precedentemente notato è quello relativo all'evento RegDeleteValue di **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\VBoxTray**. Probabilmente il malware oltre a eliminare fisicamente dal disco gli eseguibili che potrebbero ostacolare la sua esecuzione, li rimuove anche, se presenti, dall'avvio automatico tramite chiave *Run* o *RunOnce*.

#### **Values deleted:1**

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\VBoxTray: "C:\Windows\system32\VBoxTray.exe"
```

#### **Values added:7**

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194: ""C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
HKU\S-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\Policies\Explore\NoWindowsUpdate: 0x00000001
HKU\S-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\Policies\Explore\NoFolderOptions: 0x00000001
HKU\S-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce\*1726961194: ""C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe""
```

#### **Values modified:16**

```
HKLM\SYSTEM\ControlSet001\services\Disk\Enum\0:
"IDE\DiskVBOX_HARDDISK" 1.0 \5&33d1638a&0&0.0.0"
HKLM\SYSTEM\ControlSet001\services\Disk\Enum\0: "xobv"
HKLM\SYSTEM\CurrentControlSet\services\Disk\Enum\0:
"IDE\DiskVBOX_HARDDISK" 1.0 \5&33d1638a&0&0.0.0"
HKLM\SYSTEM\CurrentControlSet\services\Disk\Enum\0: "xobv"
HKU\S-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden: 0x00000001
HKU\S-1-5-21-2794639071-3859556952-360872733-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden: 0x00000002
```

#### **Files added:8**

```
C:\ProgramData\ADService.exe
C:\ProgramData\ADService.exe.ddattqqn
C:\Users\All Users\ADService.exe
C:\Users\All Users\ADService.exe.ddattqqn
C:\Users\Malware\AppData\Local\Temp\ADService.exe
C:\Users\Malware\AppData\Local\Temp\K1726961194.bat
C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe
C:\Windows\System32\VBoxTray.exe.mmjkkhhe
```

#### **Files deleted:1**

```
C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp_mod.exe
```

#### **Files [attributes?] modified:10**

```
C:\Users\Malware\AppData\Local\Microsoft\Windows\UsrClass.dat
C:\Users\Malware\AppData\Local\Microsoft\Windows\UsrClass.dat.LOG1
C:\Users\Malware\NTUSER.DAT
C:\Users\Malware\ntuser.dat.LOG1
C:\Windows\AppCompat\Programs\RecentFileCache.bcf
C:\Windows\System32\config\SOFTWARE
C:\Windows\System32\config\SOFTWARE.LOG1
C:\Windows\System32\config\SYSTEM
C:\Windows\System32\config\SYSTEM.LOG1
C:\Windows\System32\VBoxTray.exe
```

#### **Folders added:1**

```
C:\Users\Malware\AppData\Roaming\1726961194
```

Figura 38: RegShot - Report *sample\_unp\_mod.exe* (editato)

## 3.10 Wireshark e FakeNet

Dall'ipotesi fondata che il nostro malware fosse un *IRC Bot*, risulta fondamentale fare un controllo del traffico di rete per monitorare la connessione che il malware potrebbe creare con entità esterne. Nel caso di *sample.exe* l'analisi del traffico è risultata essere inconcludente poichè, come vedremo a seguito, l'eseguibile dopo aver instaurato una connessione a un server DNS e aver ricevuto risposta da quest'ultimo non genera altro traffico se non tale operazione in loop. L'esecuzione di *sample\_unp\_mod.exe* genera invece dei pacchetti molto interessanti che confermano a pieno l'ipotesi iniziale. Per fare l'analisi di rete abbiamo dapprima utilizzato FakeNet e poi Wireshark.

**Definizione.** *FakeNet è uno strumento di simulazione di rete per Windows, progettato per l'analisi di malware. Il tool reindirizza tutto il traffico in uscita da una macchina verso l'host locale (incluso il traffico IP e il traffico DNS) e implementa diversi protocolli per garantire che il codice dannoso continui a essere eseguito e possa essere osservato da un analista.*

**Definizione.** *WireShark è un packet sniffer ovvero un software che, partendo dall'utilizzo del medesimo canale di comunicazione condiviso tra più utenti, consente di esaminare il contenuto di tutti i pacchetti dati in transito. Il programma, grazie a speciali criteri di ordinamento e filtraggio consente di estrarre i dati di interesse traendo vantaggio dall'utilizzo della cosiddetta "modalità promiscua". Grazie a Wireshark è quindi possibile verificare in tempo reale che cosa avviene sulla rete locale [10].*

Il funzionamento dei due software è completamente diverso, come si può capire dalla breve descrizione data. Per l'analisi di rete, all'inizio ci siamo limitati a usare FakeNet, mantenendo la macchina virtuale non connessa a Internet. Successivamente, non ottenendo traffico interessante su *sample.exe* e sospettando che il malware rilevasse la rete falsa, per provare a provocare nuovi comportamenti, abbiamo configurato la macchina con la connessione a Internet e una volta fatto questo è stato utilizzato Wireshark per l'analisi. Nonostante tale procedura, nulla di nuovo è stato osservato, abbiamo scoperto solo successivamente che i problemi erano in realtà legati al controllo sulla data. Nel caso di *sample.exe*, FakeNet e Wireshark producono i medesimi risultati, mentre nel caso di *sample\_unp\_mod.exe* FakeNet presenta solo una piccola parte dei pacchetti prodotti da Wirehsark poichè ovviamente non è in grado di rispondere come il server vero. Per queste motivazioni mostreremo e commenteremo solo i risultati ottenuti con Wireshark.

### 3.10.1 Analisi di *sample.exe*

Nella Figura 39 vengono mostrate le richieste più interessanti che vengono fatte durante l'esecuzione di *sample.exe*. Come si può notare dal protocollo, sono richieste DNS, viene richiesta infatti la risoluzione di due nomi di dominio: `www.update.microsoft.com` e `irc.spotchat.org`. Ogni richiesta è seguita della relativa risposta da parte del server DNS. Queste query non vengono fatte una sola volta ma continuamente, a distanza di qualche decina di secondi, in un loop infinito. Tra le due, la connessione al DNS per `irc.spotchat.org` (Figura 40) risulta essere la più interessante, soprattutto visto il fatto che sospettavamo già che il sample fosse un *IRC Bot*. Andando a cercare informazioni sul dominio in questione online, facendo anche riferimento a siti che permettono di ottenere il record di registrazione del dominio (Figura 41), si scopre che `spotchat.org` è un dominio valido e affidabile. Se si va sul sito si comprende che si tratta di un'interfaccia web attraverso la quale, inserendo un nome utente, è possibile chattare su canali IRC. Oltre alle richieste DNS non sono presenti altri pacchetti anomali.

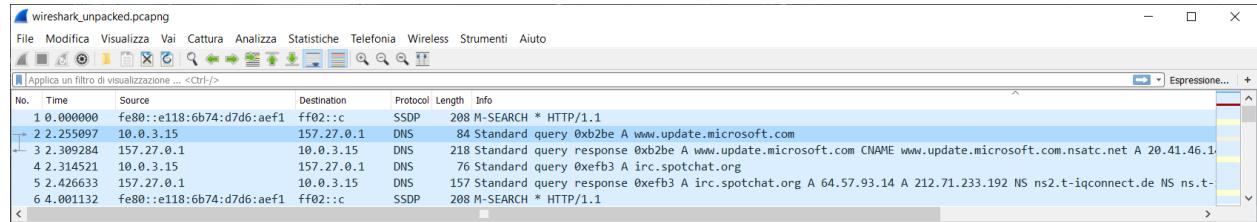


Figura 39: Wireshark - I pacchetti più rilevanti.

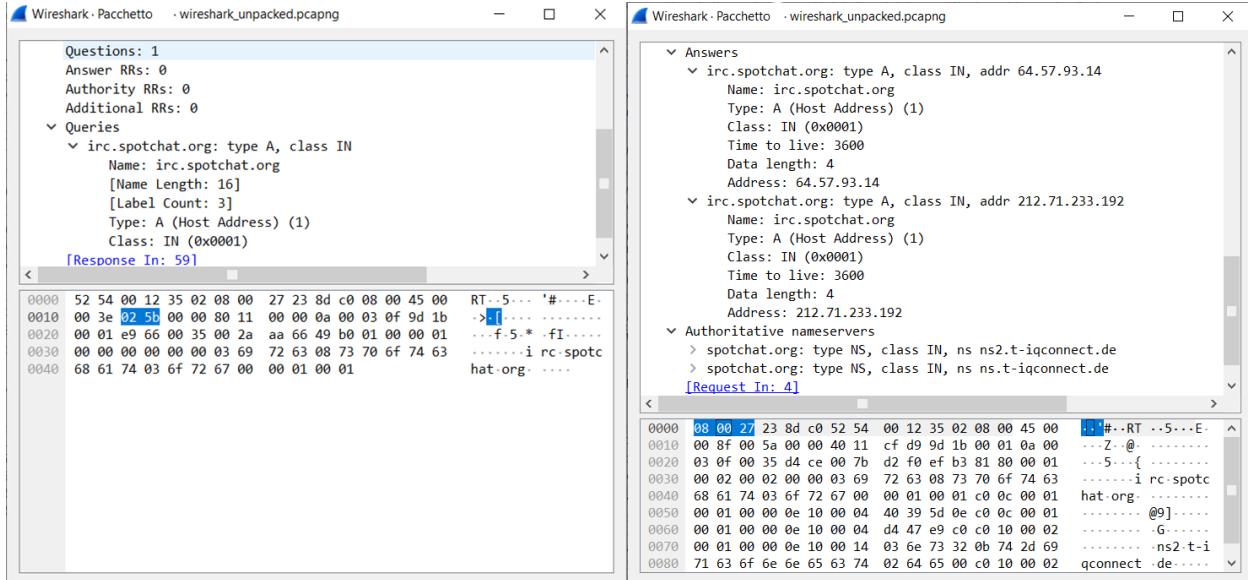


Figura 40: Wireshark - DNS per *irc.spotchat.org*: richiesta al server e relativa risposta.

Whois Record for SpotChat.org	
Domain Profile	
Registrant Country	de
Registrar	Key-Systems GmbH IANA ID: 269 URL: <a href="http://www.key-systems.net">http://www.key-systems.net</a> Whois Server: whois.rrpproxy.net abuse@key-systems.net (p) 4968949396850
Registrar Status	clientTransferProhibited
Name Servers	NS.T-IQCONNECT.DE (has 1 domains) NS2.T-IQCONNECT.DE (has 1 domains)
Tech Contact	—
IP Address	46.253.127.139 - 19 other sites hosted on this server
IP Location	Germany - Hessen - Offenbach - Aixit GmbH
ASN	AS29551 HGCOMP-ASN, DE (registered Oct 09, 2003)

Figura 41: [whois.domaintools.com](http://whois.domaintools.com) [6] - Dati di Registrazione del dominio *spotchat.org*.

### 3.10.2 Analisi di *sample\_unp\_mod.exe*

A differenza della precedente esecuzione, avviando *sample\_unp\_mod.exe* le cose si fanno più interessanti: osservando attentamente il report generato da Wireshark si possono notare oltre a tutti i pacchetti generati normalmente da Windows e a molte richieste DNS, anche una serie di messaggi IRC (Figura 42) scambiati con il server 212.71.233.192. Quest'ultimo è proprio il server IRC di Spotchat, come avevamo già visto precedentemente in Figura 39. Avendo verificato che tutti i pacchetti relativi agli altri protocolli non riguardano l'esecuzione del malware, procederemo dunque ad analizzare tutta la comunicazione che avviene tramite IRC. Prima però bisogna capire le basi relative al funzionamento di tale protocollo, visto che per ora l'abbiamo solo introdotto brevemente nelle precedenti sezioni.

### 3.10.3 Protocollo IRC in un contesto malevolo

Come già precedentemente introdotto, IRC è un protocollo di messaggistica istantanea su Internet o, detto in altri termini, un sistema che consente conversazioni (chat) in tempo reale. In passato IRC veniva usato dagli hacker per comunicare con i malware installati sulle macchine infettate e per creare le cosiddette IRC War: battaglie virtuali combattute nei canali IRC tra diversi teams, allo scopo di guadagnare pregiò tra i partecipanti allo scontro. Ora questa pratica è ormai scomparsa per via della chiusura di vari server e in generale del fallimento di IRC [29]. Il fatto che il nostro malware utilizzi questo protocollo ci fornisce anche una possibile data di creazione dello stesso, il protocollo IRC nasce nel 1988 ad opera di Jarkko "WiZ" Oikarinen e le IRC War più famose furono combattute tra il 1994 e il 2004 [17]. È dunque possibile che

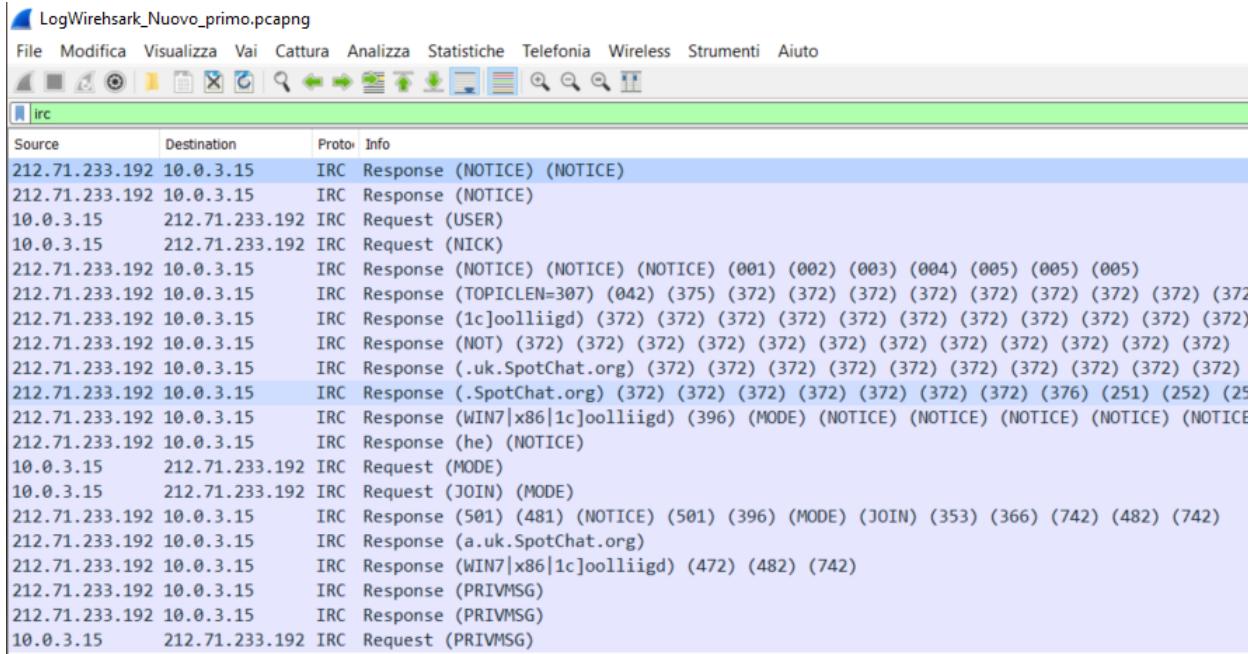


Figura 42: Wireshark - Pacchetti IRC scambiati con il server di Spotchat

il nostro malware sia stato realizzato in quegli anni (ben prima del primo commit sul repository Github *3val/Athena*).

L'*RFC 1459* descrive i protocolli di comunicazione base utilizzati per l'IRC, ci sono state nel tempo altre proposte di estensione a questo protocollo. In tutto il mondo ci sono diverse centinaia di reti IRC attive e di relativi server che eseguono diverse implementazioni del protocollo. Per comunicare bisogna accedere a un “canale” attraverso cui è possibile chattare con altri utenti, il nome del canale è di fantasia ma deve sempre iniziare con il carattere “#”. Nella concezione standard il primo utente che entra in un canale ne acquisisce automaticamente i privilegi, che può poi cedere a qualsiasi altro utente presente. Nelle implementazioni più recenti è anche possibile “registrare” i canali, in modo che i diritti di accesso non vengano persi alla disconnessione dell’ultimo utente operatore. [29]

Chi è meno avvezzo con tale protocollo, può iniziare la comunicazione su un canale collegandosi all’interfaccia web di uno dei server che offre il servizio. In alternativa è possibile utilizzare un client desktop come ad esempio ChatZilla, Pidgin, jIRC, XChat o HexChat (fork di XChat). Quest’ultimo è stato utilizzato anche da noi durante i test di interazione con il malware.

### 3.10.4 Comandi IRC

Tramite alcuni comandi definiti nei vari standard IRC (e dunque ai relativi pacchetti generati dal protocollo), è possibile eseguire tutte le operazioni di interazione con i canali e di configurazione del proprio utente. Esistono moltissimi comandi, presenti o meno sulle varie implementazioni del protocollo, tra quelli comuni i più importanti sono i seguenti: [14]

- **/whois Panda:** ritorna alcune informazioni sull’utente Panda.
- **/join #bamboo:** permette di entrare nel canale #bamboo.
- **/part #bamboo:** permette di uscire dal canale #bamboo.
- **/query Koala:** consente di parlare in privato con una persona, in questo caso con l’utente Koala.
- **/nick Kiwi:** consente di cambiare il proprio nick in Kiwi

- `/mode #channel +(-)mode [value]`: consente all'amministratore del canale o a un operatore (utente incaricato dall'amministratore) di impostare alcuni parametri sull'accesso al canale e sulla relativa gestione.
- `/kick #channel nick why`: espelle l'utente con il nick specificato dal canale `#channel`, permettendo di specificare la motivazione “`why`”.
- `/topic #channel friends`: permette di cambiare l'argomento di discussione all'interno del canale `#channel` in “`friends`”.

Parlando dell'identità dell'utente bisogna inoltre distinguere tra:

- user: è l'identità dell'utente quando comunica con un server IRC
- nick: è il soprannome dello user quando comunica all'interno di un canale.
- realname: è il nome “reale” che l'utente ha fornito durante la registrazione sul server IRC.

### 3.10.5 Pacchetti scambiati tra `sample_unp_mod.exe` e il server

Descriptoremo ora la comunicazione che avviene tra il nostro malware e Spotchat, spiegando i pacchetti presenti in Figura 42.

1. Nei primi due pacchetti Spotchat manda alcuni messaggi di benvenuto, assegnando al malware che ha richiesto la connessione un identificativo casuale ottenuto risolvendo il suo indirizzo ip: `host239-176-dynamic.17-79-r.retail.telecomitalia.it`.
2. Il malware risponde e imposta il suo user in `dabbyyyv` (Figura 43a). Come si può intuire è un nome casuale che cambia ad ogni esecuzione diversa del bot.
3. Successivamente allo username, il sample sceglie il nick con cui comunicherà all'interno dei canali: `n[GBR|A|L|WIN7|x86|1c]oolliigd` (Figura 43b). Come si può vedere all'interno del nome utente il nostro malware include anche alcune informazioni sulla macchina infettata, come ad esempio la versione del SO. Anche in questo caso gli ultimi caratteri sono casuali e cambiano ad ogni esecuzione.
4. Seguono una moltitudine di messaggi che il server invia per confermare l'utente e il nick scelto dal malware (Figura 43e) e dare alcune informazioni sul server e sui comandi che si possono usare.
5. Il malware esegue il comando `MODE` (Figura 43c) per cercare di impostare alcuni parametri sull'account e sui canali ai quali si connetterà da qui in poi. Non vedremo di preciso quale siano questi parametri poichè non risultano essere molto interessanti, ma volendo fare un esempio `+V` permette di bloccare la ricezione di inviti da parte di altri utenti.
6. Il malware prosegue con un comando fondamentale, che ci ha permesso di capire quale fosse il canale di comunicazione con il possibile hacker: viene eseguito infatti il `JOIN` al canale `#sdds` (Figura 43d). Da qui in poi siamo stati in grado di collegarci tramite HexChat al medesimo canale per vedere i dati trasmessi dal malware e cercare di capire come impartire (se possibile) comandi a quest'ultimo.
7. Spotchat risponde con vari messaggi contenenti informazioni relative all'utilizzo del comando `MODE` e altri dati relativi all'ingresso al canale. In questi pacchetti potrebbe essere anche presente il comando `RPL_TOPIC` (332), che corrisponde al “Topic” attualmente settato sul canale, se l'amministratore lo ha impostato. Nella Figura (Figura 42) non si vede poichè il “Topic” non era presente ma in un test successivo, visibile in Figura 44, si può vedere un esempio di un pacchetto contenente tale comando. Se è presente l'argomento di discussione, il messaggio 332 viene sempre inviato al client che si connette.
8. Nella comunicazione ci sono poi tutti i pacchetti `PRIVMSG` che corrispondono ai messaggi inviati all'interno del canale dagli user.
9. Infine se si lascia aperta a lungo la conversazione, appaiono dei messaggi di tipo “PING” e “PONG” che servono al server per capire se l'utente sia ancora connesso o meno. Se il client dell'utente non è attivo, non effettuerà il “PONG” e quindi il server terminerà la connessione.

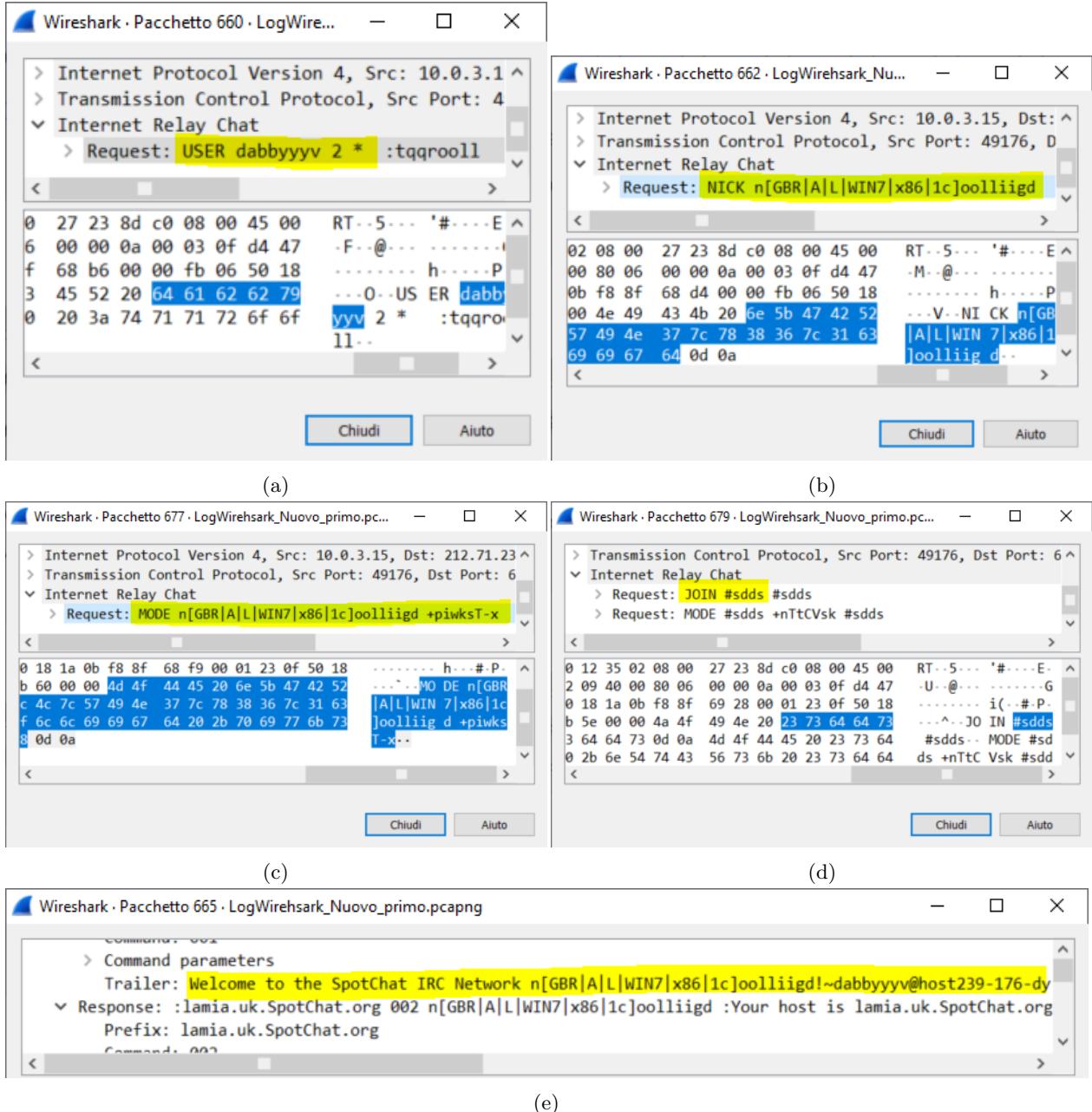


Figura 43: Wireshark - Il malware imposta il suo user (a) e il suo nick (b) attraverso il relativo comando IRC, il server di Spotchat risponde con un messaggio di conferma (e). Il sample utilizza il comando MODE (c) e fa il JOIN al canale (d).

### 3.10.6 Tentativi di comunicazione con il malware

Quando ci siamo connessi per la prima volta a #sdds ci siamo accorti che quando il bot si connette (la VM viene accesa) e si sconnette (la VM viene spenta), vengono generati due messaggi che indicano la connessione dell'utente al canale:

- n[GBR|A|L|WIN7|x86|1c]oollrigd ( dabbyyyv@host....telecomitalia.it) has joined: il malware si connette al canale quando viene avviato la prima volta o al riavvio di Windows.

- n[GBR|A|L|WIN7|x86|1c]oolliigd has quit (Quit: Windows is shutting down): nel caso in cui il malware si disconnette a causa dello spegnimento di Windows.
- n[GBR|A|L|WIN7|x86|1c]oolliigd has quit (Connection closed): quando il malware si disconnette perchè forziamo la sua terminazione.

Un'altra cosa che ha subito attirato la nostra attenzione è che pur lasciando il canale #sdds aperto per ore su HexChat, il nostro malware sulla macchina di test era l'unico a connettersi. Ciò ci ha fatto concludere che il malware ormai non è più attivo su alcun computer infettato in passato, oppure che l'hacker non abbia mai diffuso attivamente il sample creato. Per avere il pieno controllo sul canale, durante i nostri tentativi di comunicazione abbiamo deciso di registrare #sdds, in modo tale da poter eseguire tutti i comandi con i privilegi di amministratore. A questo punto della nostra analisi abbiamo dato peso a un *pdf* che avevamo già individuato in passato sul repository *3val/Athena*: il documento in questione [30] contiene, tra le altre, una serie di comandi (con relative descrizioni) che sarebbe possibile impartire al nostro bot.

All'inizio abbiamo passato parecchio tempo a cercare il modo di impartire gli ordini al malware. Inviando in chat i comandi che iniziano con il carattere “!” non accadeva nulla. Alla fine però, osservando i pacchetti su Wireshark e soprattutto grazie alla fase di Reverse Engineering, ne siamo venuti a capo. Esistono quindi diverse modalità con cui è possibile mandare comandi al bot, tra i quali:

1. modificare il Topic del canale inserendo, come argomento di discussione, il comando che si vuole far eseguire al malware in ascolto. E.g. con /topic !info è possibile ad esempio ottenere alcune informazioni sulla macchina infettata. Con questo metodo il comando viene eseguito a ogni connessione al canale, quando cioè viene mandato dal server il pacchetto contenente il comando 332 (Figura 44). È questo il motivo per cui prima abbiamo sottolineato l'importanza di RPL\_TOPIC (332).
2. mandando un messaggio in chat dove il comando è preceduto da una particolare stringa di “attivazione” riconosciuta dal bot. Un formato riconosciuto può essere di questo tipo: @my.vhost TOPIC #sdds :<comando> (e.g. @my.vhost TOPIC #sdds :!info). Con questo metodo il malware esegue subito il comando impartito nel canale.

Non inseriremo qui gli esempi di alcuni comandi con la relativa risposta da parte del malware poichè ne parleremo dettagliatamente a seguito su una sezione dedicata.

### 3.10.7 Prime conclusioni

Abbiamo finalmente compreso le vere potenzialità del malware che ci è capitato per le mani. Il nostro sample è un IRC Bot capace di creare persistenza e replicarsi in modo intelligente, per evitare di essere terminato

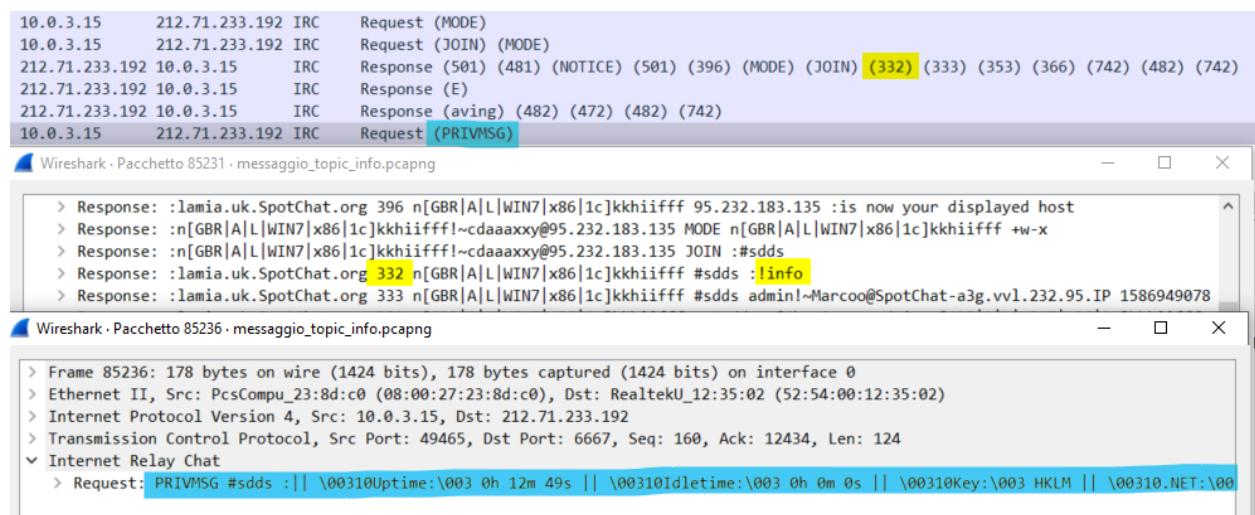


Figura 44: Wireshark - Pacchetto con comando 332 e risposta dal parte del malware

o eliminato da un utente con conoscenze nella media. Tra le funzionalità vi è anche incluso il controllo sulla data, che consente al malware di non eseguirsi da un determinato giorno in poi. I comandi impartiti tramite il canale IRC rendono possibili diverse operazioni, atte a rubare informazioni all'utente ma anche a effettuare attacchi durante una *IRC War*. Infettando diversi computer si è potenzialmente in grado di creare una vera e propria *Bot Net* capace di rispondere pienamente al creatore.

### 3.11 L'incognita dei file da 2kb

Riguardo all'operato del malware, ci mancano pochi dettagli da chiarire che saranno pienamente spiegati nella sezione di Reverse Engineering. Tra questi, l'incognita che ruota attorno alla creazione dei file da 2kb ci ha impegnato a lungo, vale perciò la pena di discuterne, concluderemo poi il discorso definitivamente quando parleremo dell'analisi fatta con OllyDBG. Come sappiamo, durante la sua esecuzione, *sample.unp-mod.exe* rinomina alcuni eseguibili aggiungendo una estensione composta da caratteri casuali e al posto del file originale, ne rimane uno omonimo dal peso di 2kb circa. Questa azione che il malware esegue, avviene sia per la sue copia *ADService.exe*, che per altri eseguibili già presenti nel sistema, come ad esempio *VBoxTray.exe*. Abbiamo quindi inizialmente elaborato alcune teorie in merito:

- All'inizio pensavamo che il file da 2kb fosse un “risultato inatteso” dell’operazione di rinomina effettuata dal malware: tentando di rinominare *ADService.exe* mentre quest’ultimo era in esecuzione poteva provocare questo comportamento non voluto. A conferma di tale ipotesi c’era il fatto che il piccolo eseguibile sembrasse “vuoto” e che rimanesse aperto in esplora risorse, a confutarla c’era invece la presenza degli strani permessi di sicurezza sul file e il fatto che cercando su internet episodi simili non si trovasse nulla. Quando durante il Reverse Engineering abbiamo trovato le istruzioni relative alla creazione di questo file, abbiamo abbandonato completamente la teoria.
- Un’altra ipotesi è che il malware usasse questa tecnica come ulteriore metodo per clonarsi, sostituendosi a programmi esistenti. Il fatto che il file da 2kb, una volta in esecuzione, non faccia pressoché nulla di interessante, ci ha allontanati da questa idea.
- La teoria che ci è venuta in mente a seguito, riguarda invece il fatto che il piccolo eseguibile potesse semplicemente servire a evitare il riavvio del programma originale, sia da parte del sistema che da parte dell’utente. Il fatto che tale comportamento avvenga anche per *ADService.exe* (copia del malware) ci crea dei dubbi, tuttavia bisogna tener conto anche del fatto che durante un’ esecuzione normale del sample (non forzando la terminazione del malware dal task manager), la rinominazione con estensione casuale e la generazione del file da 2kb avvengono solo per altri programmi presenti sul computer e non per *ADService.exe*. Il nostro file sarebbe quindi solo un piccolo eseguibile non funzionante (derivante da *NTVDM.exe*), avente il solo scopo di evitare che l’utente possa accorgersi che il programma originale è stato eliminato. Questa ultima teoria combacia meglio con tutti i risultati ottenuti durante l’analisi statica e dinamica.

Trarremo la conclusione definitiva in merito al file da 2kb sulla parte di Reverse Engineering, dove sarà chiaro lo scopo dell’eseguibile. Con l’ultima teoria elaborata non ci siamo allontanati molto dalla soluzione.

### 3.12 Comandiamo il bot IRC: proviamo alcuni comandi

Passiamo ora a testare i vari comandi presenti nella documentazione di Athena [30], che scopriremo corrispondere esattamente a quelli del nostro malware. In questa fase, come già anticipato, utilizzeremo il client IRC HexChat, collegandoci al server di SpotChat da quest’ultimo sarà possibile eseguire il JOIN al canale **#sdds** e testare alcuni comandi. Nella documentazione sono presenti diverse tipologie di comandi:

- **Comandi generici:** servono a ottenere alcune informazioni relative al pc infettato e per configurare varie impostazioni del malware (e.g. `!info`, `!version`, `!uninstall`).
- **Comandi di download:** consentono principalmente di scaricare un eseguibile da una sorgente online e di avviarlo (e.g. `!download`).

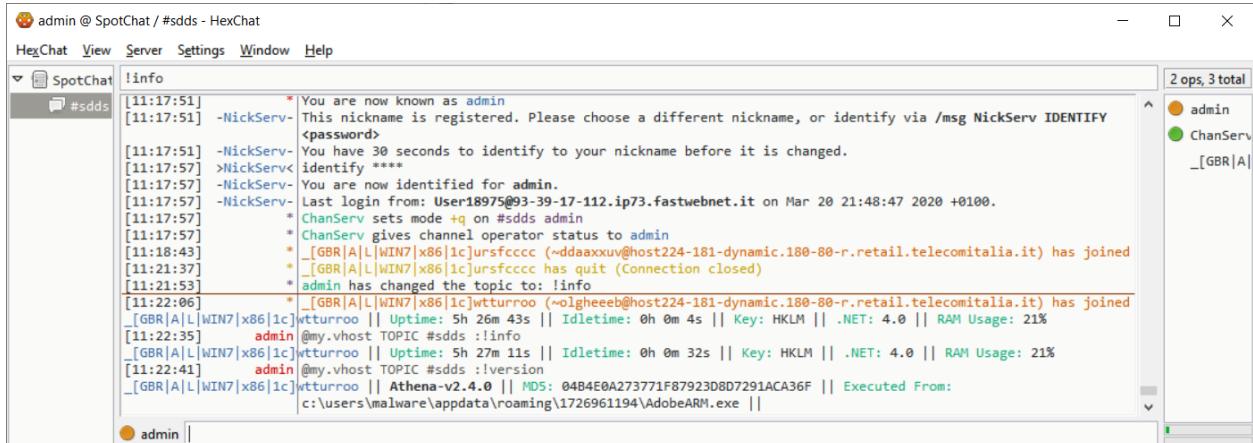
- **Comandi IRC:** permettono di far eseguire al malware alcuni comandi in ambito IRC, ad esempio `!irc.join #channel` comunica al malware di effettuare il join a un altro canale deciso dall'hacker al momento.
- **Comandi DDoS:** questi comandi servono a far partire un attacco DDoS dal pc verso un certo target online. Ovviamente questo comando ha maggior effetto, tanti più sono le macchine infettate con il malware (e.g. `!ddos.http.slowloris`).
- **Comandi di ricerca, furto e modifica dei file:** alcuni esempi sono `!ftp.upload`, `!recovery.ftp` e `!filesearch`.
- **Comandi di apertura siti:** consentono principalmente di visitare siti web in modalità finestra, visibile cioè dall'utente (e.g. `!view`) e in modalità nascosta (e.g. `!view.hidden`).
- **Comandi IRC War:** comandi che consentono di partecipare a una IRC War o di effettuare alcune operazioni relative a questo ambito (e.g. `!war.connect`, `!war.status`).

La scelta di quali comandi provare a eseguire nella nostra fase di test, naturalmente è dovuta non solo alla semplicità della relativa operazione ma anche alla visibilità degli effetti sul sistema host. Per ovvie motivazioni, non abbiamo invece provato alcun comando relativo a attacchi DDoS e a IRC War.

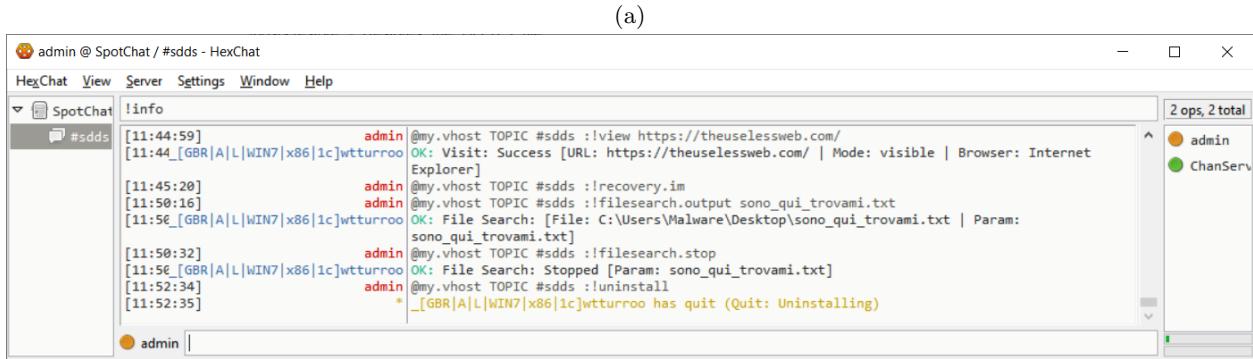
Nelle Figure 45a e 45b si possono vedere il modo in cui i comandi sono stati inviati dal canale al bot e le risposte di quest'ultimo. Da notare che nel momento dei test il malware si era collegato al canale usando il nick `_-[GBR|A|L|WIN7|x86|1c]wtturroo` mentre noi siamo autenticati come `admin`. Nei test seguenti per inviare i comandi al malware si è deciso di usare la forma `@my.vhost TOPIC #sdds :!<comando>` per avere una risposta immediata da parte del sample. A seguito la descrizione dei comandi testati e dei loro effetti:

- **`!info`** [11:21:53] [11:22:35]: permette di conoscere alcune informazioni relative al computer infettato, nello specifico, da quanto tempo il computer è acceso, il tempo di inattività, il tipo di chiave di registro installata, versione di .NET presente nel SO (se installato) e RAM utilizzata.
- **`!version`** [11:22:35]: stampa la versione del bot e il path dove è in esecuzione il malware.
- **`!view https://theuselessweb.com`** [11:44:59]: Visualizza la pagina web relativa all'url specificato (nel nostro caso `https://theuselessweb.com`), su un browser preso casualmente tra quelli presenti sulla macchina infettata. A differenza di `!view.hidden`, questo comando mostra una finestra visibile, questo vuol dire che l'utente si accorgerà della pagina aperta. Nel nostro test, il sito in questione è stato effettivamente aperto in una finestra del browser Internet Explorer, in Figura 46 si può vedere l'evento ProcMon relativo all'azione svolta dal malware.
- **`!recovery.im`** [11:45:20]: Permette di recuperare alcune credenziali salvate sul sistema host. Nel nostro caso il malware non effettua l'output poichè probabilmente non sono presenti credenziali, essendo la nostra una VM di test.
- **`!filesearch.output sono_qui_trovami.txt`** [11:50:16]: Cerca nell'intero file system del computer infettato un file il cui nome contiene la stringa specificata. L'opzione `.output` permette di impostare che il malware fornisca il path dove è stato trovato tale file. Questo comando in un utilizzo malevolo potrebbe servire a cercare file di testo contenenti password o altri documenti sensibili, potrebbe inoltre essere seguito dal comando `!ftp.upload` che consente all'hacker di caricare su una destinazione online tali contenuti, rubandoli effettivamente all'utente ignaro. Nel nostro esempio abbiamo preventivamente posizionato il file `sono_qui_trovami.txt` sul desktop della VM e abbiamo quindi chiesto al bot di cercarlo nella macchina. Come si può vedere in Figura 45b, nel canale il sample ci fornisce effettivamente il path del file. In Figura 47 si può osservare invece l'evento che ha permesso al malware di scansionare la cartella Desktop dell'utente, per conoscerne il contenuto.
- **`!filesearch.stop`** [11:50:32]: comunica al malware di terminare la ricerca di file. Tramite questo comando è possibile terminare una ricerca precedentemente avviata, nel nostro caso se non avessimo usato `!filesearch.stop` il sample avrebbe continuato a cercare il file `sono_qui_trovami.txt` per verificare se quest'ultimo fosse presente anche su altri path.

- `!uninstall` [11:52:34]: questo comando molto interessante permette all'hacker di disattivare il malware a distanza. Durante questa fase viene generato (Figura 49a) ed eseguito (Figura 49b) lo script `U1726961194.bat` che avevamo già commentato precedentemente nella sezione riguardante i file batch. Sottolineiamo che l'esecuzione di tale script è una tecnica usata dal malware per terminare forzatamente se stesso, evitando che i vari cloni possano rimettersi in vita l'un l'altro. In Figura 48 si può vedere l'evento ProcMon relativo alla rimozione della persistenza, viene cancellato sia il valore su `HKLM` che il valore presente in `HKCU`. Effettivamente, a un successivo riavvio di Windows, il malware non risulta più essere in esecuzione. Stranamente in alcuni casi, eseguendo il comando, l'esegubile `AdobeARM.exe` è ancora presente nel relativo path.



(a)



(b)

Figura 45: HexChat - (a) Apertura e chiusura della connessione da parte del malware e alcuni comandi semplici. (b) Comandi avanzati e disinstallazione del malware.

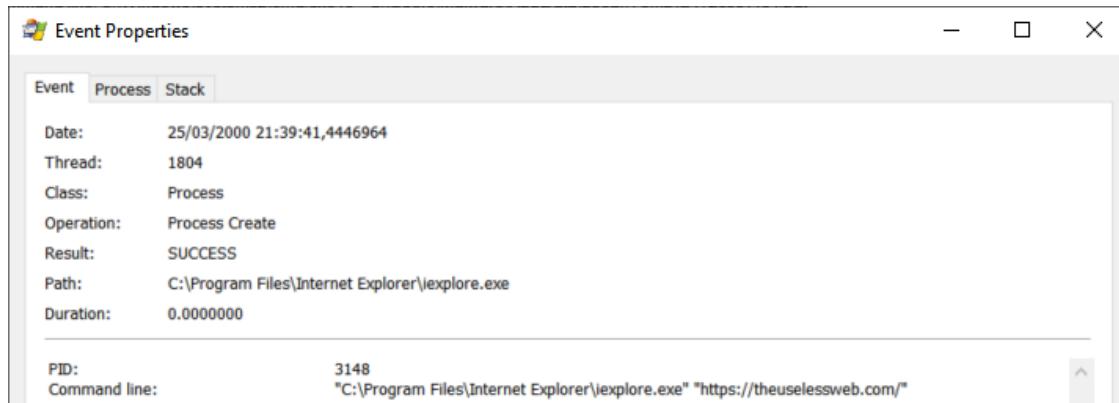


Figura 46: ProcMon - Apertura della pagina web sul browser Internet Explorer

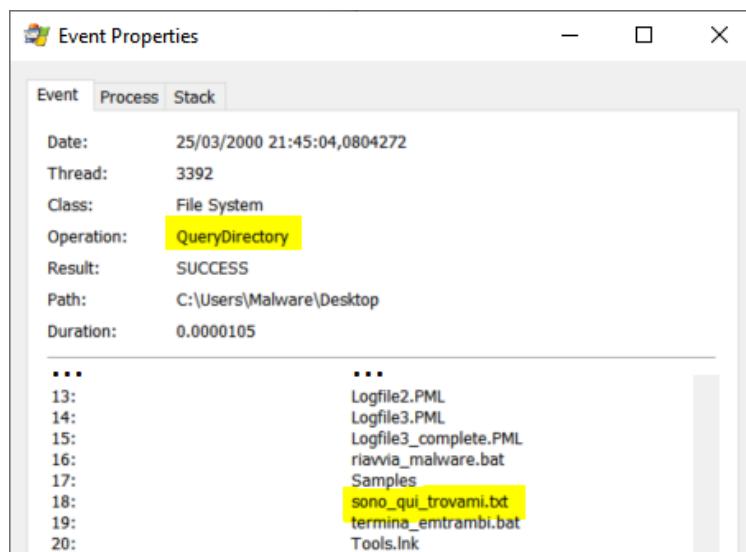


Figura 47: ProcMon - Query per la ricerca del file `sono_qui_trovami.txt`

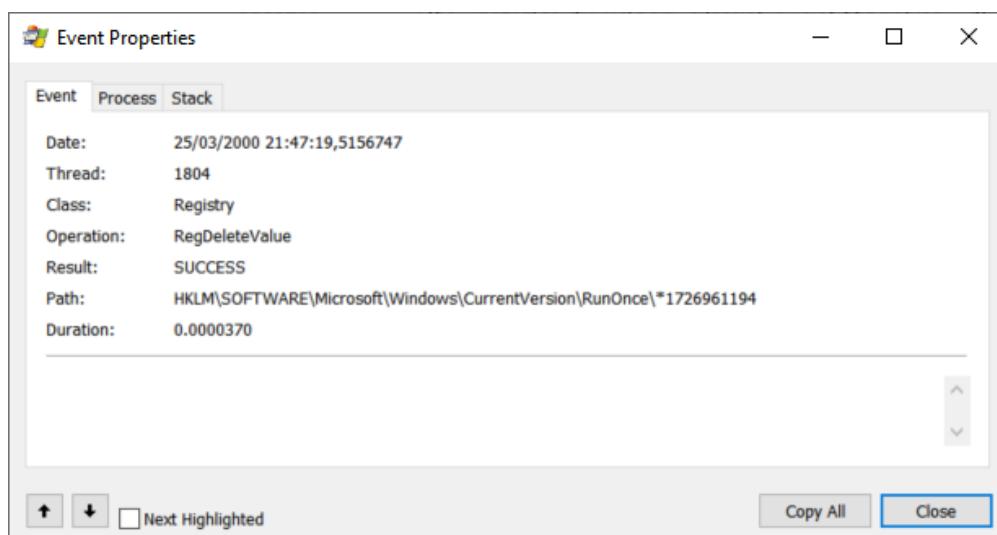
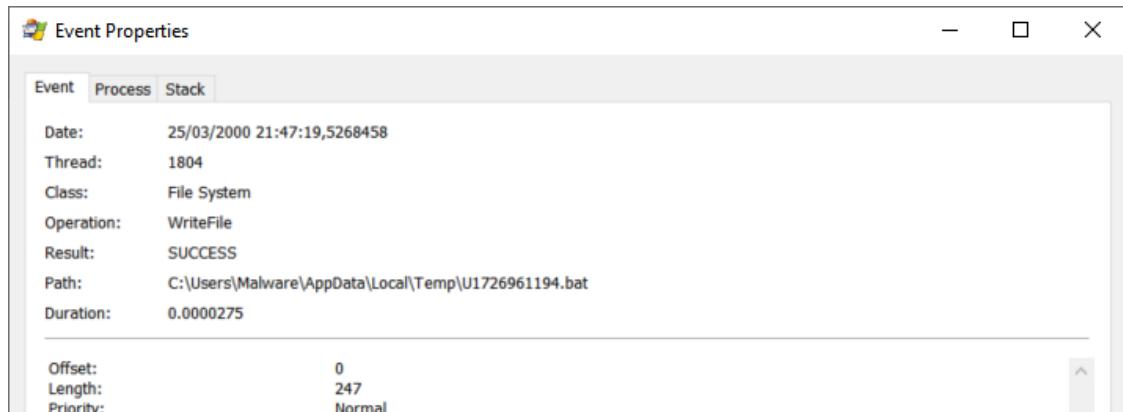


Figura 48: ProcMon - Rimozione della persistenza



(a)



(b)

Figura 49: ProcMon - Creazione (a) ed esecuzione (b) dello script U1726961194.bat

## 4 Reverse Engineering

Per il Reverse Engineering su *sample\_unp.exe* sono stati utilizzati principalmente IDA e OllyDBG. Per aiutarsi nell'analisi si è anche fatto ricorso agli strumenti di decompilazione di Ghidra.

**Definizione.** *L'Interactive Disassembler, più comunemente conosciuto con il nome IDA, è un disassembler largamente usato per il reverse engineering. Supporta numerosi formati di file eseguibili per diversi processori e sistemi operativi [29].*

**Definizione.** *Ghidra è uno tool di reverse engineering gratuito e open source sviluppato dalla National Security Agency (NSA). Le funzionalità sono molto simili a quelle offerte da IDA ma, a differenza di quest'ultimo, offre un utile decompilatore che produce codice C.*

**Definizione.** *OllyDbg è un debugger per Windows che fa analisi a livello assembler. L'enfasi sull'analisi del codice binario lo rende particolarmente utile nei casi in cui il codice sorgente sia disponibile [25].*

Naturalmente sono state analizzate solo le funzioni che presentavano le caratteristiche più interessanti e utili. Si noti che, in questa sezione, abbiamo ovviamente tenuto in considerazione anche il codice sorgente precedentemente citato, trovato su *GitHub*, per i metodi più ostici da analizzare usando solo gli strumenti sopra. Nei metodi interessati verrà esplicitamente indicato il differente comportamento tra l'esecuzione dei due sample, *sample\_unp.exe* (malware unpacked) e *sample\_unp\_mod.exe* (malware con data modificata). Se invece non è detto nulla allora si sottintende che non c'è alcuna differenza nella logica della funzione, cioè che nel metodo non è presente nessun controllo sulla data ed esso viene eseguito indifferentemente da essa.

### 4.1 Impostazioni per OllyDBG

Per il debugging si è deciso di utilizzare la versione 2.01 di OllyDBG perché ritenuta più affidabile e con più funzionalità rispetto alla 1.10 [24]. Per poter sfruttare al meglio le funzionalità del debugger è stato necessario modificare alcune impostazioni poiché si sono riscontrati diversi problemi. Avviando entrambe le versioni di OllyDBG si è notata una discrepanza nell'entry point visualizzata dai due debugger. Mentre la versione 1.10 mostra 00401DAA0 come indirizzo iniziale di esecuzione, la versione 2.01 mostra 004012A0, così come IDA. Tale differenza è dovuta al fatto che sono presenti due funzioni di TLS callback, che vengono eseguite prima di ogni altra istruzione nel codice. Per poter analizzare quindi le funzioni nell'ordine corretto (*TlsCallback\_0*, *TlsCallback\_1*, *start*) è stato necessario modificare le impostazioni di default dello start di OllyDBG 2.01 (da *WinMain (if location is known)* a *System breakpoint*, Figura 53) e settare manualmente dei breakpoint agli indirizzi di *TlsCallback\_0* (0041DAA0) e *TlsCallback\_1* (0041DA50). Gli indirizzi delle due funzioni TLS sono stati trovati tramite IDA (Figura 50).

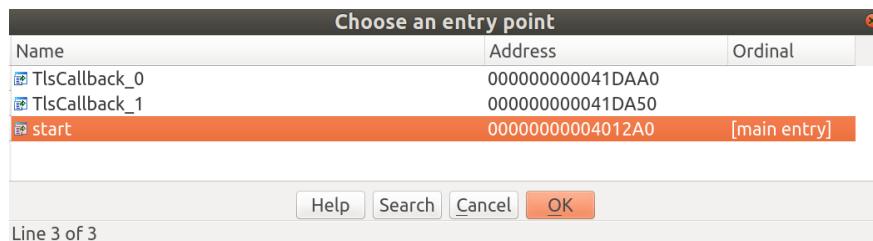


Figura 50: IDA - Possibili entry point (Ctrl + E)

Un altro problema in cui si incorre è la gestione delle eccezioni (accessi a zone di memoria non autorizzati (Figura 51), istruzioni non valide, etc.). Per evitare che il tool si blocchi ogni volta che si verifica uno di questi eventi, si è settato il debugger affinché ignorasse tutte le eccezioni di default (Options → Options → Debugging → Exceptions → Selezionare tutte le caselle sotto a Ignore following exceptions).

Si è dovuto anche aggiungere manualmente tutte le eccezioni specifiche del malware che sorgevano durante l'esecuzione (Options → Options → Debugging → Exceptions → Ignore also the following custom exceptions or ranges → Add current) (Figura 54) e impostare il debugger affinché le eccezioni non processate fossero gestite dall'Unhandled Exception Filter, per evitare errori come quello in Figura 55 e 52. Le thread del

programma si possono visualizzare con il comando (Alt + T) (Figura 56). Fatto ciò l'esecuzione del sample continua fino alla fine senza solitamente errori, tranne quelli generati dalle thread, che mandano in pausa l'esecuzione del debugger e di cui non si è trovata soluzione. Riteniamo che l'autore del malware abbia usato le eccezioni come tecnica anti-debugging per ostacolare l'analisi [18]. Molto probabilmente usa il side effect del debugger per alterare il flusso di esecuzione del malware. Una delle tecniche più comuni per fare ciò è implementare parte della logica di un programma malevolo negli exception handler. Se fosse OllyDBG a gestire le eccezioni, senza richiamare gli handler, potremmo dunque avere un flusso di esecuzione alterato. Ignorando le eccezioni però, e quindi lasciando che sia il programma stesso a gestirli e non OllyDBG, possiamo essere sicuri che il malware continui la sua esecuzione normalmente senza distorsioni nel flusso.



Figura 51: OllyDBG - Errore: accesso non autorizzato a zona di memoria



Figura 52: OllyDBG - Errore generato da un'altra thread passata in esecuzione che blocca l'applicazione.

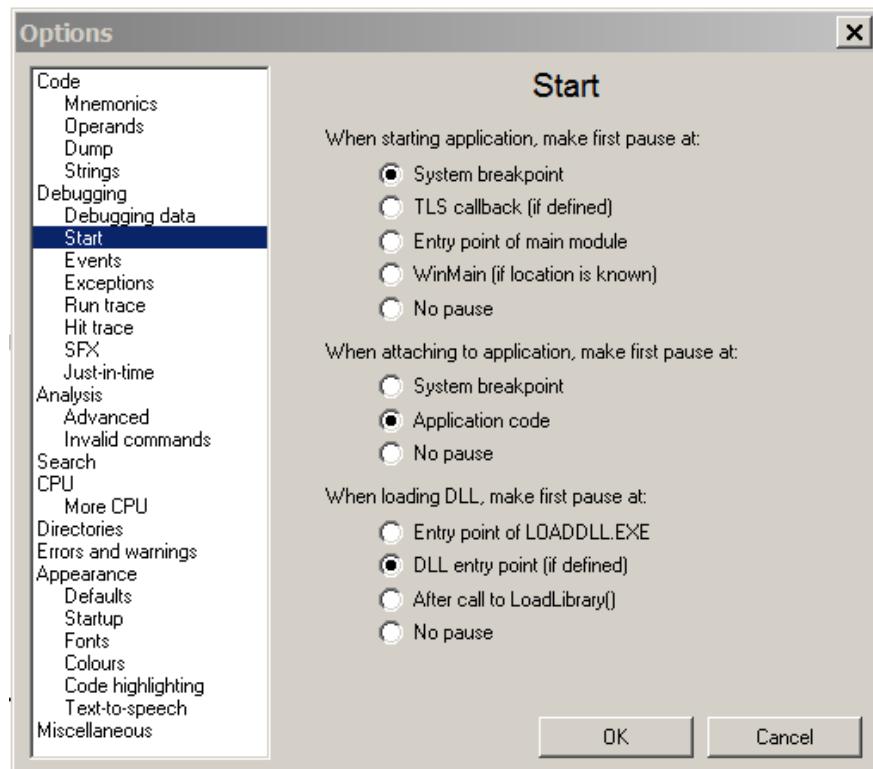


Figura 53: OllyDBG - Impostazioni per l'entry point (Options → Options → Debugging → Start)

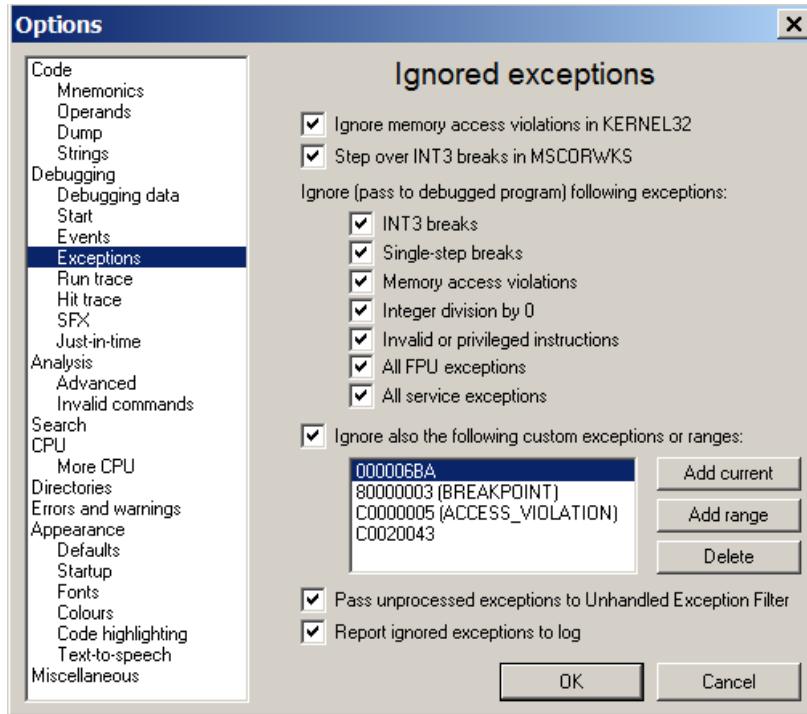


Figura 54: OllyDBG - Impostazioni per ignorare tutte le eccezioni del malware, indipendentemente dalla natura.



Figura 55: OllyDBG - Errore da eccezione custom non gestita che fa terminare inaspettatamente l'esecuzione del debugger.

Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
1.	00000ESC		WSAECONNREFUSED (0x00324D)	sample_unp.<ModuleEntryPoint> 004012H0	7FFDF000	0.	Normal	0.2500 s	0.0000 s
2.	00000ESC	The Wireshark Network Analyzer	ERROR_SUCCESS (0x00000000)	004012H0	7FFDC000	1.	Normal	0.0000 s	0.0000 s
3.	00000898		ERROR_SUCCESS (0x00000000)	00405B0E	7FFDB000	1.	Normal	0.0000 s	0.0000 s
4.	00000898		ERROR_SUCCESS (0x00000000)	77B8FC03	7FFDB000	1.	Normal	0.0000 s	0.0000 s
5.	00000898		ERROR_NO_WRITE_FILES (0x00000012)	004053H1	7FFD7000	1.	Normal	0.0000 s	0.0000 s
6.	00000870		ERROR_NO_REMOTE_STORAGE (0x00000087)	sample_unp.PERSIST 00406EF0	7FFD7000	1.	Normal	0.0000 s	0.0000 s
161.	00000818		ERROR_SUCCESS (0x00000000)	0040CB79	7FFDA000	0.	Normal	0.0000 s	0.0000 s

Figura 56: OllyDBG - Thread generate e in esecuzione del programma (Alt + T).

## 4.2 Funzioni TLS callback

**Definizione.** L'archiviazione thread-local (TLS, Thread-Local Storage) è il meccanismo attraverso il quale ogni thread in un dato processo multithread alloca lo spazio di archiviazione per i dati specifici dei thread. Nei programmi multithread standard, i dati vengono condivisi da tutti i thread di un determinato processo, mentre l'archiviazione thread-local è il meccanismo che consente di allocare i dati per singoli thread. [4]

Le funzioni TLS callback possono essere utilizzate come efficace tecnica anti-debugging siccome esse vengono eseguite prima del tradizionale entry point del programma e non vengono solitamente rilevate dai debugger (come discusso sopra nel caso di OllyDBG 2.01) [8]. Un possibile attaccante quindi potrebbe scrivere l'intera logica di un codice malevole, o comunque parte di essa, in questi tipi particolari di metodi. Al suo avvio in un debugger, senza che l'analista se ne accorga, il programma potrebbe aver già eseguito buona parte delle sue funzionalità. Nel nostro caso, analizzando attentamente le due funzioni TlsCallback\_0 (Figura 57) e TlsCallback\_1 (Figura 58), siamo arrivati alla conclusione che non è presente in esse nessun comportamento malevolo e che quindi non sono state usate per ostacolare un eventuale reverser.

### 4.2.1 Funzioni TlsCallback\_0, TlsCallback\_1

Entrambe le funzioni gestiscono l'inizializzazione/eliminazione di oggetti di tipo critical section attraverso la funzione `set_critical_section` controllando dei parametri. Gli oggetti critical section vengono utilizzati per la sincronizzazione di thread appartenenti ad un unico processo, che prenderanno il possesso dell'oggetto in maniera mutualmente esclusiva [5].

```

1 || 
2 /* WARNING: Removing unreachable block (ram,0x0041dae5) */
3 /* WARNING: Removing unreachable block (ram,0x0041dae7) */
4 /* WARNING: Removing unreachable block (ram,0x0041daf2) */
5 /* WARNING: Removing unreachable block (ram,0x0041daf4) */
6 /* WARNING: Removing unreachable block (ram,0x0041dafb) */
7 
8 undefined4 tls_callback_0(undefined4 param_1,int param_2)
9 {
10    if (DAT_004877b0 != 2) {
11        DAT_004877b0 = 2;
12    }
13    if ((param_2 != 2) && (param_2 == 1)) {
14        set_critical_section(param_1,1);
15    }
16    return 1;
17 }
18 }
```

Figura 57: Ghidra - Codice ad alto livello per TlsCallback\_0 prodotto dal decompilatore. Da notare che ci sono delle zone irraggiungibili che vengono ignorate.

```

1 || 
2 undefined4 tls_callback_1(undefined4 param_1,int param_2)
3 
4 {
5    if ((param_2 != 0) && (param_2 != 3)) {
6        return 1;
7    }
8    set_critical_section(param_1,param_2);
9    return 1;
10 }
11 }
```

Figura 58: Ghidra -Codice ad alto livello per TlsCallback\_1 prodotto dal decompilare.

### 4.2.2 Funzioni set\_critical\_section, manage\_critical\_section

La funzione `set_critical_section` (sub\_41E1A0) (Figura 59) si occupa di:

- inizializzare un oggetto critical section, tramite il metodo di libreria Windows `InitializeCriticalSection`, la quale riceve come parametro un puntatore all'indirizzo di tale oggetto [20].
- Eliminare un oggetto critical section, tramite la funzione di libreria Windows `DeleteCriticalSection`, la quale riceve come parametro un puntatore all'indirizzo dell'oggetto [20].
- Gestire l'accesso e le operazioni al critical object tramite il metodo `manage_critical_section`.

```

1 | 
2 | undefined4 __cdecl set_critical_section(undefined4 param_1,int param_2)
3 | {
4 |     if (param_2 != 1) {
5 |         if (param_2 == 0) {
6 |             if (DAT_004877b4 != 0) {
7 |                 manage_critical_section();
8 |             }
9 |             if (DAT_004877b4 == 1) {
10 |                 DAT_004877b4 = 0;
11 |                 DeleteCriticalSection((LPCRITICAL_SECTION)&DAT_004877b8);
12 |             }
13 |         }
14 |     }
15 |     else {
16 |         if ((param_2 == 3) && (DAT_004877b4 != 0)) {
17 |             manage_critical_section();
18 |         }
19 |     }
20 |     return 1;
21 | }
22 | if (DAT_004877b4 == 0) {
23 |     InitializeCriticalSection((LPCRITICAL_SECTION)&DAT_004877b8);
24 | }
25 | DAT_004877b4 = 1;
26 | return 1;
27 |
28 }
```

Figura 59: Ghidra - Codice ad alto livello per `set_critical_section` prodotto dal decompilatore.

La funzione `manage_critical_section` (sub\_41E020) (Figura 60) si occupa di:

- gestire il possesso del critical section object tramite la funzione `EnterCriticalSection`.
- Rilasciare l'object tramite la funzione `LeaveCriticalSection`.
- Recuperare il valore memorizzato nel TLS della thread in un determinato indice (`TlsGetValue`), controllare l'ultimo errore generato dalla thread (`GetLastError`) e impostare alcuni valori nella TLS.

### 4.3 Inizio del programma

*sample\_unp.exe* ha il proprio entry point all'indirizzo 004012A0, che corrisponde all'inizio della funzione `start`. Il metodo fa prima una chiamata a `_set_app_type` per impostare il tipo di applicazione e poi chiama, subito dopo, la subroutine `main` (sub\_401000) (Figura 61). Quest'ultima setta alcune impostazioni riguardanti operazioni su file (`_setmode`, `_p_fmode`) e variabili del process environment (`_p_environ`), per poi passare a chiamare il metodo `main_bot`.

#### 4.3.1 Funzione `main_bot`

La funzione `main_bot` (sub\_40B2DC) avvia prima di tutto una serie di operazioni necessarie per creare l'ambiente “ideale” per il malware (Figura 62). Le funzioni principali usate a tale scopo sono, in ordine di chiamata:

- `is_sandbox`: controllo della presenza di una sandbox.
- `srand`: creazione di un seme per le funzioni di generazione di valori numerici randomici.
- `set_up_info`: estrazione delle informazioni di configurazione per la connessione IRC.

- **define\_variables**: recupero di informazioni del sistema e definizione del valore di diverse variabili globali da utilizzare successivamente.
- **handle\_main\_copy**: gestione dell'esecuzione di due copie del malware.
- **create\_thread\_persist**: creazione di una thread che andrà ad eseguire la funzione **persist**.
- **dns\_flush\_resolver\_cache**: utilizzo del metodo omonimo dalla libreria *dnsapi.dll* per fare il flush della cache DNS.

Prima di creare le thread per la persistenza il malware controlla se il proprio nome sia uguale a quello nuovo definito in **define\_variables**. Se non lo è, crea una nuova cartella C:\Users\Malware\AppData\Roaming\1726961194 (**CreateDirectoryA**) e usa **MoveFileExA**, con flag **MOVEFILE\_WRITE\_THROUGH** [21], per spostarsi e cambiare nome. Nella Figura 63 si possono vedere i due possibili nuovi nomi per il malware. Nel primo caso l'operazione fallisce perché il nome non è valido, nel secondo invece ha successo e si sposta nella nuova cartella appena creata con nome **AdobeARM.exe**. Prima di terminare viene eseguita la funzione **f\_to irc** (Figura 64) che richiamerà a sua volta **connect\_to irc**, la quale si occupa principalmente della connessione IRC e dell'invio, ricezione e interpretazione dei messaggi dal canale IRC.

```

1 void manage_critical_section(void)
2 {
3     DWORD *pDVar1;
4     LPVOID pvVar2;
5     DWORD DVar3;
6
7     EnterCriticalSection((LPCRITICAL_SECTION)&DAT_004877b8);
8     pDVar1 = DAT_004877d0;
9     while (pDVar1 != (DWORD *)0x0) {
10         pvVar2 = TlsGetValue(*pDVar1);
11         DVar3 = GetLastError();
12         if ((DVar3 == 0) && (pvVar2 != (LPVOID)0x0)) {
13             (*(code *)pDVar1[1])(pvVar2);
14         }
15         pDVar1 = (DWORD *)pDVar1[2];
16     }
17     LeaveCriticalSection((LPCRITICAL_SECTION)&DAT_004877b8);
18     return;
19 }
20
21 }
```

Figura 60: Codice ad alto livello per **manage\_critical\_section** prodotto dal decompilare di Ghidra.

```

.text:004012A0 ; ===== S U B R O U T I N E =====
.text:004012A0
.text:004012A0 ; Attributes: noreturn
.text:004012A0
.text:004012A0          public start
.text:004012A0 start    proc near
.text:004012A0     argc    = dword ptr -1Ch
.text:004012A0     sub     esp, 1Ch
.text:004012A0     mov     [esp+1Ch+argc], 2 ; argc
.text:004012A0     call    ds:_set_app_type
.text:004012A0     call    main
.text:004012B0 start    endp
```

Figura 61: IDA - Funzione **start**

```

.text:0040B2F4          mov    esi, [ecx+4]
.text:0040B2F7          call   sub_41E000
.text:0040B2FC          call   is_sandbox
.text:0040B301          test  al, al
.text:0040B303          jz    short loc_40B31C
.text:0040B305          mov    [esp+1B8h+uExitCode], 0 ; uExitCode
.text:0040B30D          mov    [esp+1B8h+hProcess], OFFFFFFFFh ; hProcess
.text:0040B314          call   TerminateProcess
.text:0040B319          sub   esp, 8
.text:0040B31C          ; CODE XREF: main_bot+27+j
.text:0040B31C loc_40B31C:
.text:0040B31C          call   sub_40B45C
.text:0040B321          test  al, al
.text:0040B323          jz    loc_40B41A
.text:0040B329          call   FreeConsole
.text:0040B32E          call   sub_40BA9
.text:0040B333          call   sub_414AB0
.text:0040B338          mov    [esp+1B8h+hProcess], eax ; Seed
.text:0040B33B          call   srand
.text:0040B340          call   set_up_info
.text:0040B345          call   define_initial_variables
.text:0040B34A          mov    [esp+1B8h+uExitCode], esi
.text:0040B34E          mov    [esp+1B8h+hProcess], ebx

```

Figura 62: IDA - Alcune subroutine chiamate da `main_bot`

(a)

```

0040B372|| E8 69C5FFFF CALL 004078E0 | sample_unp.004078E0
0040B377|| C74424 04 E0 MOU DWORD PTR SS:[LOCAL.110],OFFSET 004; string2 => "!"
0040B37F|| C76424 A09F4: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; string1 => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B386|| E8 A1400100 CALL <JMP.&nsvprt.strcmp> | MSVCRT.strncmp
0040B388|| 85C0 TEST EAX,EAX
0040B38D|| C74424 04 00 MOU DWORD PTR SS:[LOCAL.110],0 | pSecurity => NULL
0040B397|| C74424 009F4: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; Flags => MOVEFILE_WRITE_THROUGH
0040B39E|| E8 C9420100 CALL <JMP.&KERNEL32.CreateDirectoryA> | New => "!"
0040B393|| 83EC 08 SUB ESP,8 | Existing => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B396|| C74424 08 08 MOU DWORD PTR SS:[LOCAL.109],8 | KERNEL32.CreateDirectoryA
0040B39E|| C74424 04 E0 MOU DWORD PTR SS:[LOCAL.110],OFFSET 004; Flags => MOVEFILE_WRITE_THROUGH
0040B393|| C76424 A09F4: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; Existing => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B386|| E8 62420100 CALL <JMP.&KERNEL32.MoveFileExA> | KERNEL32.MoveFileExA
0040B3C2|| 83EC 0C SUB ESP,0C

```

(b)

```

0040B372|| E8 69C5FFFF CALL 004078E0 | sample_unp.004078E0
0040B377|| C74424 04 E0 MOU DWORD PTR SS:[LOCAL.110],OFFSET 004; string2 => "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
0040B37F|| C76424 A09F4: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; string1 => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B386|| E8 A1400100 CALL <JMP.&nsvprt.strcmp> | MSVCRT.strncmp
0040B388|| 85C0 TEST EAX,EAX
0040B38D|| C74424 04 00 MOU DWORD PTR SS:[LOCAL.110],0 | pSecurity => NULL
0040B397|| C74424 C9A94: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; Flags => MOVEFILE_WRITE_THROUGH
0040B39E|| E8 C9420100 CALL <JMP.&KERNEL32.CreateDirectoryA> | New => "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
0040B393|| 83EC 08 SUB ESP,8 | Existing => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B396|| C74424 08 08 MOU DWORD PTR SS:[LOCAL.109],8 | KERNEL32.CreateDirectoryA
0040B39E|| C74424 04 E0 MOU DWORD PTR SS:[LOCAL.110],OFFSET 004; Flags => MOVEFILE_WRITE_THROUGH
0040B393|| C76424 A09F4: MOU DWORD PTR SS:[LOCAL.111],OFFSET 004; Existing => "C:\Users\Malware\Desktop\Samples\Reverse Engineering\sample_unp.exe"
0040B386|| E8 62420100 CALL <JMP.&KERNEL32.MoveFileExA> | KERNEL32.MoveFileExA

```

Figura 63: OLLYDBG - Il sample crea una nuova cartella, si rinomina e si sposta. In Figura (a) il nuovo nome è `!`, in (b) è `AdobeARM.exe`.

```

.text:004089EF          ; ===== S U B R O U T I N E =====
.text:004089EF          proc near
.text:004089EF          sub   esp, 0Ch
.text:004089EF          call  connect_to irc
.text:004089F2          add   esp, 0Ch
.text:004089F7          retn
.text:004089FA          endp
.text:004089FA          f_to irc
.text:004089FA          ; CODE XREF: main_bot+139+p

```

Figura 64: IDA - Funzione `f_to irc` che si occupa solo di fare una chiamata a `connect_to irc`

#### 4.4 Funzione di controllo sandbox

**Definizione.** Una sandbox è un meccanismo per eseguire applicazioni in uno spazio limitato. Solitamente fornisce un ristretto e controllato set di risorse al programma che deve essere testato, come un'area ristretta di memoria o un insieme di chiamate di sistema limitate; di norma, l'accesso alla rete, la possibilità di ispezionare il sistema ospite o leggere dai dispositivi di input, sono disabilitati o altamente ristretti. Data la capacità di fornire un ambiente estremamente controllato, le sandbox possono essere viste come uno esempio specifico di virtualizzazione, solitamente utilizzata per eseguire programmi non testati o non attendibili, non verificati o provenienti da terze parti non riconosciute (come utenti o siti web), senza rischiare di infettare il dispositivo dove viene eseguita l'applicazione, ad esempio effettuare test su programmi non verificati che possono contenere virus o codice maligno, senza permettere al software di infettare il dispositivo ospite. [28]

**Definizione.** Il *productId* è un valore univoco di 20 cifre derivato dal *product key* e dalla configurazione hardware del sistema che serve ad identificare un determinato sistema operativo Windows. [2]

#### 4.4.1 Funzione is\_sandbox

La funzione `is_sandbox` (sub\_40BD0C) controlla che il malware non si trovi in una sandbox. Per fare ciò cerca, all'interno del sistema, tramite la subroutine `GetModuleHandleA` le librerie appartenenti a noti software per sandbox:

- SbieDll.dll, appartenente a Sandboxie;
  - snxhk.dll, appartenente ad Avast sandbox;
  - dbghelp.dll, appartenente a ThreatExpert [2].

Il metodo inoltre controlla il ProductId del sistema (`RegQueryValueExA`), all'interno del registro `SOFTWARE/Microsoft/Windows NT/CurrentVersion`, per verificare che non sia uguale a uno dei seguenti, appartenenti a sandbox note:

- 76487-640-1457236-23837, Anubis sandbox;
  - 76487-644-3177037-23510, CWSandbox;
  - 55274-640-2673064-23950, JoeBox;
  - 76497-640-6308873-23835, CWSandbox.

Il test sopra viene eseguito solo se la versione del sistema operativo, trovata tramite la funzione di libreria `GetVersion`, è uguale a 5 (Windows 2000 oppure Windows XP [16]). Nel nostro caso specifico la virtual machine ha Windows 7 quindi viene controllata solo la presenza delle librerie (Figura 65). Se viene effettivamente rilevata una sandbox il programma termina immediatamente.

00400D1F	F3:DQ	REP STOS PTR ES:[EDI]	
00400D21	CD4424 28 04	MOV DWORD PTR SS:[LOCAL_68],104	
00400D29	E8 56390100	CALL <JMP.&KERNEL32.GetVersion>	KERNEL32.GetVersion
00400D2E	S9C3	MOV EBX,EAX	
00400D39	C70424 14704	MOV DWORD PTR SS:[LOCAL_78],OFFSET 00427	[ModuLeName => "Sbie.dll"]
00400D37	E8 68380100	CALL <JMP.&KERNEL32.GetModuleHandleA>	KERNEL32.GetModuleHandleA
00400D3D	S8EC 04	SUB ESP,4	
00400D3F	BF 01000000	MOV EDI,1	
00400D44	S8C0	TEST EAX,ERX	
00400D46	BF85 3301000	JNE 00400BE7F	
00400D4C	C70424 28704	MOV DWORD PTR SS:[LOCAL_78],OFFSET 00427	[ModuLeName => "snxhk.dll"]
00400D51	E8 3AC80100	CALL <JMP.&KERNEL32.GetModuleHandleA>	KERNEL32.GetModuleHandleA
00400D58	S8EC 04	SUB ESP,4	
00400D5B	S8C0	TEST EAX,EAX	
00400D5D	BF85 1C010000	JNZ 00400BE7D	[ModuLeName => "dbghelp.dll"]
00400D63	C70424 28704	MOV DWORD PTR SS:[LOCAL_78],OFFSET 00427	KERNEL32.GetModuleHandleA
00400D6A	E8 35380100	CALL <JMP.&KERNEL32.GetModuleHandleA>	
00400D6F	S8EC 04	SUB ESP,4	
00400D72	S8C0	TEST EAX,ERX	
00400D74	BF85 F900000	JNE 00400BE73	
00400D7B	S8FB 05	CMP BL,5	
00400D7D	BF85 F7000000	JNE 00400BE70	
00400D83	S9D424 24	LEA EAX,[LOCAL_69]	
00400D89	S9D424 10	MOV DWORD PTR SS:[LOCAL_74],EAX	
00400D8B	C74424 9C 19	MOV DWORD PTR SS:[LOCAL_75],2019	
00400D93	F74424 98 00	MOV DWORD PTR SS:[LOCAL_261],0	
Jump is taken			
Dest:sample_wnd_00400BE7A			

Figura 65: OllyDBG - Dopo il controllo delle librerie viene saltato quello del productId e is\_sandbox termina.

## 4.5 Recupero delle informazioni di sistema e definizione di variabili

### 4.5.1 Funzione define\_variables

Il metodo `define_variables` (sub\_4133EF) si occupa di recuperare tutta una serie di informazioni dal sistema operativo e inizializzare molteplici variabili globali per poterle utilizzare nelle funzioni successive. In particolare:

- attraverso l'uso del metodo di libreria Windows `getenv` il malware ottiene il valore di diverse variabili d'ambiente [7]. Variabili come WINDIR, APPDATA, TEMP,PROGRAMFILES, USERPROFILE, ALLUSERSPROFILE e PROGRAMDATA memorizzano, al loro interno, il percorso alle relative cartelle (Figura 66).
- Viene controllato il tipo di sistema operativo.
- Vengono inizializzate molteplici variabili tra cui le più interessanti sono quella contenente il valore della “expiration date” (Figura 67) e quella che memorizza il nome della cartella 1726961194, calcolata a runtime con la funzione `itoa` (Figura 68).
- Vengono trovati i browsers installati nel sistema con il metodo `find_browsers`.

Anche il nuovo nome da dare al malware viene definito in questa funzione tramite due meccanismi diversi. Il metodo chiama, a un certo punto, la subroutine `sub_412FEF` che calcola e ritorna la stringa `AdobeARM.exe` solo se la data corrente non supera la “expiration date” (Figura 69). In caso contrario il valore `l` viene settato come nuovo nome alla fine di `define_variables`, dopo l'ennesimo controllo sulla data (Figura 70).

```

0041342F | . C605 66864301 MOV BYTE PTR DS:[438666],0
00413436 | . C70424 358E41 CALL QWORD PTR SS:[LOCAL.270],OFFSET 00413436
00413440 | . E8 5BFB0000 CALL <JMP.&msvcrt._getenv>
00413442 | . 894424 04 MOV DWORD PTR SS:[LOCAL.2691],EAX
00413446 | . C70424 095441 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 00413446
0041344D | . E8 5BFB0000 CALL <JMP.&msvcrt._scanf>
00413452 | . C70424 3C5841 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 00413452
00413459 | . E8 5BFB0000 CALL <JMP.&msvcrt._getenv>
0041345E | . 894424 04 MOV DWORD PTR SS:[LOCAL.2691],EAX
00413462 | . C70424 605A41 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 00413462
00413469 | . E8 3EBF0000 CALL <JMP.&msvcrt._strcpy>
0041346E | . C70424 445841 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 0041346E
00413475 | . E8 A2BF0000 CALL <JMP.&msvcrt._getenv>
00413479 | . 894424 00 MOV DWORD PTR SS:[LOCAL.2691],EAX
0041347E | . C70424 405941 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 0041347E
00413485 | . E8 22BF0000 CALL <JMP.&msvcrt._strcpy>
00413488 | . C70424 498B41 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 00413488
00413491 | . E8 86BF0000 CALL <JMP.&msvcrt._getenv>
00413496 | . 894424 00 MOV DWORD PTR SS:[LOCAL.2691],EAX
00413499 | . C70424 095541 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 00413499
004134A1 | . E8 6BFB0000 CALL <JMP.&msvcrt._strcpy>
004134A6 | . C70424 555841 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 004134A6
004134AD | . E8 6BFB0000 CALL <JMP.&msvcrt._getenv>
004134B2 | . 894424 04 MOV DWORD PTR SS:[LOCAL.2691],EAX
004134B6 | . C70424 005741 MOU DWORD PTR SS:[LOCAL.2701],OFFSET 004134B6
004134BD | . E8 ERBE0000 CALL <JMP.&msvcrt._strcpy>
004134C2 | . C70424 655841 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 004134C2
004134C9 | . E8 4EBF0000 CALL <JMP.&msvcrt._getenv>
004134CE | . 894424 04 MOV DWORD PTR SS:[LOCAL.2691],EAX
004134D2 | . C70424 E05541 MOV DWORD PTR SS:[LOCAL.2701],OFFSET 004134D2
004134D9 | . E8 CEBE0000 CALL <JMP.&msvcrt._strcpy>
004134DE | . E8 F9DBFFFF CALL 0041180C
004134E3 | . B8 A9E84300 MOU EAX,OFFSET 00439BA0
004134E8 | . B8 728B4200 MOU ES1,OFFSET 00428B72
004134ED | . B9 1D000000 MOU ECX,1D

[vars]
varname => "WINDIR"
MSVCRT._getenv
src
dest => "C:\Windows"
MSVCRT.__mbscopy
varname => "APPDATA"
MSVCRT._getenv
src
dest => "C:\Users\Malware\AppData\Roaming"
MSVCRT.__mbscopy
varname => "TEMP"
MSVCRT._getenv
src
dest => "C:\Users\Malware\AppData\Local\Temp"
MSVCRT.__mbscopy
varname => "USERPROFILE"
MSVCRT._getenv
src
dest => "C:\Users\Malware"
MSVCRT.__mbscopy
varname => "ALLUSERSPROFILE"
MSVCRT._getenv
src
dest => "C:\ProgramData"
MSVCRT.__mbscopy
varname => "PROGRAMFILES"
MSVCRT._getenv
src
dest => "C:\Program Files"
MSVCRT.__mbscopy
sample_upn_0041100C
ASCII '\"Start Menu\Programs\Startup'
ASCII '\"Start Menu\Programs\Startup'


```

Figura 66: OllyDBG - Informazioni relative all'ambiente del sistema host

```

00413744 | . C705 E40E4301 MOV DWORD PTR DS:[430EE4],0
0041374E | . C70424 0A0001 MOU DWORD PTR SS:[LOCAL.270],0A
00413755 | . E8 89F9FFFF CALL 004130E3
0041375A | . 89C3 MOU EBX,EAX
0041375C | . C70424 0A0001 MOU DWORD PTR SS:[LOCAL.270],0A
00413763 | . E8 7BF9FFFF CALL 004130E3
00413768 | . 81C3 8F000201 ADD EBX,2008F
0041376E | . 29C3 SUB EBX,EAX
00413770 | . B91D 80C84201 MOU DWORD PTR DS:[42C880],EBX
00413776 | . C705 1CA34301 MOU DWORD PTR DS:[43A831C],0
00413780 | . C705 18034301 MOU DWORD PTR DS:[43B03181],0

[Registers]
EBX=0002008C (current registers)
0042C8801=0002008C

```

Figura 67: OllyDBG - Assembly per calcolare e impostare “expiration date”. 2008F è la “standard expiration date” che rappresenta il 15 Dicembre 2013.

```

0041367D0 | . A9 889F4300 MOV DWORD PTR DS:[439F88],EAX
00413680 | . C74424 08 8C MOV DWORD PTR SS:[LOCAL_268],0A
00413681 | . C74424 04 8C MOV DWORD PTR SS:[LOCAL_269],OFFSET 004; radix => 10.
00413682 | . 890424 MOV DWORD PTR SS:[LOCAL_270],EAX buffer
00413683 | . E8 BABC0000 CALL <JMP.&msvcrt._itoa> value
00413684 | . B8 E58C4200 MOV EAX,OFFSET 0043A2E5
00413685 | . BE 058C4200 MOV ESI,OFFSET 00428C05
00413686 | . B9 11000000 MOV ECX,11 ASCII ":Zone.Identifier"
00413687 | . 89C7 MOU EDI,EAX
00413688 | . 89C7 MOU EDI,EAX

```

ERX=66EF562A  
Stack [0022F920]=sample\_unp.00439BA0, ASCII "C:\Users\Malware\AppData\Roaming(null)64.57.98.14irc.spotchat.org6667#sddsAthena-v2.4.0"

```

0041367D0 | . A9 889F4300 MOV DWORD PTR DS:[439F88],EAX
00413680 | . C74424 08 0A MOV DWORD PTR SS:[LOCAL_268],0A
00413681 | . C74424 04 8C MOV DWORD PTR SS:[LOCAL_269],OFFSET 004; radix => 10.
00413682 | . 890424 MOV DWORD PTR SS:[LOCAL_270],EAX buffer => "1726961194"
00413683 | . E8 BABC0000 CALL <JMP.&msvcrt._itoa> value
00413684 | . B8 E58C4200 MOV EAX,OFFSET 0043A2E5
00413685 | . BE 058C4200 MOV ESI,OFFSET 00428C05
00413686 | . B9 11000000 MOV ECX,11 ASCII ":Zone.Identifier"
00413687 | . 89C7 MOU EDI,EAX
00413688 | . 89C7 MOU EDI,EAX

```

MSVCRT.\_itoa returned EAX = "1726961194"  
Imm=sample\_unp.00439B8C, ASCII "1726961194"

(b)

Figura 68: OllyDBG - Il nome della cartella 1726961194 viene creato dinamicamente

```

00413072 | . 3B05 80C8420 CMP EAX,DWORD PTR DS:[42C880]
00413073 | . 7D 59 JGE SHORT 004130D3
00413074 | . A1 889F4300 MOV EAX,DWORD PTR DS:[439F88]
00413075 | . B9 17000000 MOV ECX,17
00413076 | . BA 00000000 MOU EDX,0
00413077 | . F7F1 DIV ECX
00413078 | . 0FBFC5 MOVSX EAX,BP
00413079 | . 0100 ADD EAX,EDX
0041307A | . 83F8 16 CMP EAX,16
0041307B | . 76 07 JBE SHORT 0041309C
0041307C | . 0FBFFD MOVSX EBP,BP
0041307D | . F7D5 NOT EBP
0041307E | . 01E8 ADD EAX,EBP
0041307F | . 801480 LEA EDX,[EAX*4+EAX]
00413080 | . 800450 LEA EAX,[EDX*2+EAX]
00413081 | . 8D7444 36 LEA ES1,[EAX*2+ESP+36]
00413082 | . BF E0284800 MOV EDI,OFFSET 004828E0
00413083 | . B9 16000000 MOV ECX,16
00413084 | . F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
00413085 | . BF E0284800 MOV EDI,OFFSET 004828E0
00413086 | . B0 00 MOU AL,0 ASCII "AdobeARM.exe"
00413087 | . B9 FFFFFFFF MOU ECX,-1
00413088 | . F2:AE REPNE SCAS BYTE PTR ES:[EDI]
00413089 | . F7D1 NOT ECX
0041308A | . C781 DF28480 MOV DWORD PTR DS:[ECX+4828DF],6578652E
0041308B | . C681 E3284800 MOU BYTE PTR DS:[ECX+4828E3],0 ASCII "AdobeARM.exe"
0041308C | . B8 E0284800 MOV EAX,OFFSET 004828E0
0041308D | . 80C4 3C02000 ADD ESP,23C ASCII "AdobeARM.exe"
0041308E | . 5B POP EBX

```

Figura 69: OllyDBG - Creazione del nuovo nome AdobeARM.exe per il malware

```

004139AD | . 895424 0C MOV DWORD PTR SS:[LOCAL_267],EDX
004139B0 | . C74424 08 AD MOV DWORD PTR SS:[LOCAL_268],OFFSET 004; ASCII "%w%w%d"
004139B1 | . C74424 04 141 MOV DWORD PTR SS:[LOCAL_269],14
004139B2 | . 891C24 MOV DWORD PTR SS:[LOCAL_270],EBX
004139B3 | . E8 138A0000 CALL <JMP.&msvrt.strftime>
004139B4 | . 891C24 MOV DWORD PTR SS:[LOCAL_270],EBX Jump to msvrt.strftime
004139B5 | . E8 138A0000 CALL <JMP.&msvrt.atoi>
004139B6 | . 3B05 80C84200 CMP EAX,DWORD PTR DS:[42C880] MSVCRT.atoi
004139B7 | . 7C 09 JL SHORT 004139E2
004139B8 | . 661C705 E0A1 MOV WORD PTR DS:[43A1E0],7C
004139B9 | . C74424 0C A01 MOV DWORD PTR SS:[LOCAL_267],OFFSET 004; %ws => "#sdds"
004139B0 | . C74424 08 E01 MOV DWORD PTR SS:[LOCAL_268],OFFSET 004; %ws => "irc.spotchat.org"
004139B1 | . C74424 04 821 MOV DWORD PTR SS:[LOCAL_269],OFFSET 004; Format => "%s%s"

```

Imm=007C  
[0043A1E0]=007C

Address	Hex dump	ASCII
0043A1E0	7C 00 5C 55 73 65 72 73 5C 4D 61 6C 77 61 72 65	\Users\Malware
0043A1F0	5C 41 70 70 44 61 74 61 5C 52 6F 61 60 69 6E 67	\AppData\Roaming
0043A200	5C 31 37 32 36 39 36 31 31 39 34 5C 00 00 00 00	\1726961194\
0043A210	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0043A220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figura 70: OllyDBG - Creazione del nuovo nome \ per il malware

## 4.6 Configurazione per il client IRC

I metodi presentati in questa sezione servono all'estrazione dei dati di configurazione per la connessione IRC e il suo successivo utilizzo.

### 4.6.1 Funzione set\_up\_info

La funzione `set_up_info` (sub\_40C032) si occupa di estrarre le informazioni per la configurazione iniziale (Figura 71):

- **Athena-v2.4.0**, versione della libreria Athena utilizzata
- **irc.spotchat.org**, server al quale collegarsi
- **6667**, porta
- **#sdds**, canale al quale iscriversi
- **#sdds**, chiave del canale
- **my.host**, authentication host
- **64.57.93.14**, owner

Tali informazioni, insieme ad altri valori quali checksum etc., non sono in chiaro. Il metodo utilizza infatti la stringa `LcQ1vUua5UwEGjzgw...` (visibile staticamente e codificata in Base64) per estrarre a run time tutti i dati, cioè la stringa viene passata attraverso vari metodi che ricostruiscono le varie informazioni dinamicamente (Figura 72). È interessante notare che a un certo punto la stringa in questione viene decodificata da Base64 e cifrata con RC4 usando la chiave `Imagination is more important than knowledge..`

Address	Hex dump	ASCII
0022D0E0	0C D2 22 00 E0 EE 42 00 00 00 00 00 00 00 00 00	.E".0~B.....
0022D0F0	41 74 68 65 6E 61 2D 76 32 2E 34 2E 30 00 69 72	Athena-v2.4.0.ir
0022D100	63 2E 73 70 6F 74 63 68 61 74 2E 6F 72 67 00 36	c.spotchat.org.6
0022D110	36 36 37 00 23 73 64 64 73 00 23 73 64 64 73 00	667.#sdds.#sdds.
0022D120	6D 79 2E 76 68 6F 73 74 00 36 34 2E 35 37 2E 39	my.vhost.64.57.9
0022D130	33 2E 31 34 00 30 00 31 33 32 37 36 36 38 32 34	3.14.0.132766024
0022D140	37 00 2D 36 38 35 35 31 34 39 38 00 38 37 36	7.-685551498.876
0022D150	36 35 34 35 32 38 00 2D 31 39 38 30 34 33 38 34	654528.-19804304
0022D160	30 38 00 2D 36 30 35 33 39 38 30 38 32 00 2D 39	08.-605398082.-9
0022D170	32 30 36 34 37 39 33 38 00 2D 31 34 31 33 32 35	20647938.-141325
0022D180	34 35 34 33 00 32 35 38 34 36 36 31 37 38 00 2D	4543.258466178.-
0022D190	31 31 38 32 36 33 31 36 36 39 00 31 33 00 31 36	1182631669.13.16
0022D1A0	00 35 00 35 00 38 00 31 00 50 52 49 56 4D 53 47	.5.5.8.1.PRIUMSG
0022D1B0	00 4A 4F 49 4E 00 50 41 52 54 00 55 53 45 52 00	.JOIN.PART.USER.
0022D1C0	4E 49 43 4B 00 50 41 53 53 00 50 4F 4E 47 00 35	NICK.PASS.PONG.5
0022D1D0	00 31 00 30 3F 00 00 00 00 00 00 00 00 00 00 00	.1.0?.....

Figura 71: OllyDBG - Dump di memoria - Stringa decodificata

### 4.6.2 Funzioni di codifica e decodifica Base64

**Definizione.** *Base64 è un sistema di codifica che consente la traduzione di dati binari in stringhe di testo ASCII, rappresentando i dati sulla base di 64 caratteri ASCII diversi. [29]*

La funzione `base64_encode(sub_403779)` codifica un array di byte (char) usando Base64 (RFC 3548) con l'utilizzo di una subroutine ausiliaria `_base64_encode_triple` che codifica gruppi di tre bytes (24 bits) in 4 caratteri ASCII (6 bit per carattere, i caratteri ASCII usati vengono definiti nella stringa ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/), visibile in IDA nella finestra Strings). La funzione `base64_decode` (sub\_40385E) invece decodifica stringhe precedentemente codificate in Base64 (Figura 73).

### 4.6.3 Funzione di cifratura RC4

**Definizione.** *RC4 è un algoritmo di cifratura a flusso a chiave simmetrica, in passato ampiamente diffuso in protocolli quali l'SSL ed il WEP ma non più ritenuto sicuro a causa di diversi attacchi che ne hanno provato la debolezza.* [29]

La funzione `rc4` (sub\_404708) cifra un testo in chiaro con l'algoritmo RC4. La chiave usata, per l'unica chiamata alla funzione, è la stringa `Imagination is more important than knowledge.` (Figura 74).

```

0040C456 . 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],1
0040C45E .~75 15 JNZ SHORT sample_u.0040C475
0040C460 . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C464 . C70424 E0EE421 MOV DWORD PTR SS:[ESP],sample_u.0042EEE1
0040C468 . E8 3C2F0100 CALL <JMP.&msvcrt._strcpy>
0040C470 .~E9 6B850000 JMP sample_u.0040C9E0
0040C475 > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],2
0040C47D .~75 15 JNZ SHORT sample_u.0040C494
0040C47F . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C483 . C70424 E0EA421 MOV DWORD PTR SS:[ESP],sample_u.0042EAE1
0040C48A . E8 1D2F0100 CALL <JMP.&msvcrt._strcpy>
0040C48E .~E9 4C050000 JMP sample_u.0040C9E0
0040C494 > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],3
0040C49C .~75 13 JNZ SHORT sample_u.0040C4B1
0040C49E . 898424 MOV DWORD PTR SS:[ESP],EAX
0040C4A1 . E8 3E2F0100 CALL <JMP.&msvcrt._atoi>
0040C4A6 . 66:A3 88DC4201 MOV WORD PTR DS:[42DC881],AX
0040C4AC .~E9 2F050000 JMP sample_u.0040C9E0
0040C4B1 > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],4
0040C4B9 .~75 15 JNZ SHORT sample_u.0040C4D0
0040C4BB . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C4BF . C70424 A0D8421 MOV DWORD PTR SS:[ESP],sample_u.0042D8A1
0040C4C6 . E8 E12E0100 CALL <JMP.&msvcrt._strcpy>
0040C4CB .~E9 10050000 JMP sample_u.0040C9E0
0040C4D0 > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],5
0040C4D8 .~75 15 JNZ SHORT sample_u.0040C4EF
0040C4DA . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C4DE . C70424 A0D4421 MOV DWORD PTR SS:[ESP],sample_u.0042D0A1
0040C4E5 . E8 C22E0100 CALL <JMP.&msvcrt._strcpy>
0040C4E9 .~E9 F1840000 JMP sample_u.0040C9E0
0040C4EF > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],6
0040C4F7 .~75 15 JNZ SHORT sample_u.0040C50E
0040C4F9 . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C4FD . C70424 A0D0421 MOV DWORD PTR SS:[ESP],sample_u.0042D0A1
0040C504 . E8 A32E0100 CALL <JMP.&msvcrt._strcpy>
0040C509 .~E9 D2840000 JMP sample_u.0040C9E0
0040C50E > 66:83BD B004F1 CMP WORD PTR SS:[EBP-2B50],7
0040C516 .~75 15 JNZ SHORT sample_u.0040C52D
0040C518 . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040C51C . C70424 A0CC421 MOV DWORD PTR SS:[ESP],sample_u.0042CC41
0040C523 . E8 842E0100 CALL <JMP.&msvcrt._strcpy>
0040C528 .~E9 D2840000 JMP sample_u.0040C9E0

```

Figura 72: OllyDBG - Informazioni estrapolate dinamicamente durante l'esecuzione

```

0040C0F2 | . 895424 08 MOV DWORD PTR SS:[ESP+8],EDX
0040C0F6 | . 8BBD B004FFF1 MOV EDI,DWORD PTR SS:[LOCAL.2772]
0040C0FC | . 897C24 04 MOV DWORD PTR SS:[ESP+4],EDI
0040C100 | . 8B85 B0D4FFF1 MOV EDX,DWORD PTR SS:[LOCAL.2773] ASCII "LcQ1vUua5UwEGjzgwxc0t3y00ceYgd8t03Eh4KvAfqRUsuknVgdXd/2HNqmDUP8F1g/0zh4
0040C105 | . 895424 0C MOV EDI,DWORD PTR SS:[ESP+8],EDX
0040C109 | . 8B85 B804FFF1 MOV EDX,DWORD PTR SS:[LOCAL.2773]
0040C113 | . 895424 0C MOV EDI,DWORD PTR SS:[LOCAL.2773],EDI ASCII "LcQ1vUua5UwEGjzgwxc0t3y00ceYgd8t03Eh4KvAfqRUsuknVgdXd/2HNqmDUP8F1g/0zh4
0040C118 | . 89C6 MOV EST,EAX
0040C11A | . 89C1 MOV ECX,EAX
0040C11C | . 40 INC EAX
0040C11D | . 30D2 XOR DL,DL
0040C11F | . 90 NOP
0040C120 | . 90 NOP
Stack 0022D284=sample_unp.0043A3A0, ASCII "LcQ1vUua5UwEGjzgwxc0t3y00ceYgd8t03Eh4KvAfqRUsuknVgdXd/2HNqmDUP8F1g/0zh4
ERK=sample_unp.0043A3A0, ASCII "LcQ1vUua5UwEGjzgwxc0t3y00ceYgd8t03Eh4KvAfqRUsuknVgdXd/2HNqmDUP8F1g/0zh4/F0rxtvDevw1G"

```

Figura 73: OllyDBG - Il metodo `base64_decode` viene usato per decodificare la stringa `LcQ1vUua5UwEGjzgw...`

```

0040C169 | . 8B95 ACD4FFF1 MOV EDX,DWORD PTR SS:[LOCAL.2773]
0040C16F | . 895424 0C MOV DWORD PTR SS:[ESP+8C],EDX
0040C173 | . C74424 08 70 MOV DWORD PTR SS:[ESP+8],DFSET 004271 ASCII "Imagination is more important than knowledge."
0040C17B | . 8974E4 04 MOV DWORD PTR SS:[ESP+4],ESI
0040C17F | . 8BBD B004FFF1 MOV EDI,DWORD PTR SS:[LOCAL.2772]
0040C185 | . 893C24 MOV DWORD PTR SS:[ESP],EDI
0040C188 | . E8 7B85FFFF CALL 00404708 RC4
0040C18D | .~EB 0A JMP SHORT 0040C199

```

Figura 74: OllyDBG - Chiave usata per fare la cifratura RC4

## 4.7 Funzioni di installazione e persistenza

Le subroutines presenti in questa sezione permettono al malware, attraverso varie tecniche, di mantenersi nascosto all'interno del sistema host e di avviarsi in automatico a ogni riavvio di Windows, senza più nessuna azione da parte dell'ignaro utente.

### 4.7.1 Funzione create\_thread\_persist

La funzione `create_thread_persist` (sub\_40669F) si occupa della creazione di una thread che esegue il metodo `persist` (Figura 75). La persistenza è quindi demandata a una thread figlia invece che a quella principale (Figura 76).

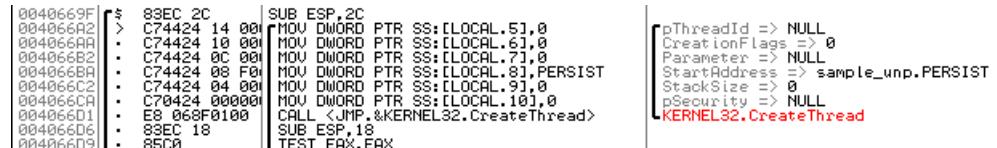


```

.text:004066A2 loc_4066A2: ; CODE XREF: create_thread_persist+6B+j
.text:004066A2        mov    [esp+2Ch+lpThreadId], 0 ; lpThreadId
.text:004066AA        mov    [esp+2Ch+dwCreationFlags], 0 ; dwCreationFlags
.text:004066B2        mov    [esp+2Ch+lpParameter], 0 ; lpParameter
.text:004066BA        mov    [esp+2Ch+lpStartAddress], offset persist ; lpStartAddress
.text:004066C2        mov    [esp+2Ch+dwStackSize], 0 ; dwStackSize
.text:004066CA        mov    [esp+2Ch+lpThreadAttributes], 0 ; lpThreadAttributes
.text:004066D1        call   CreateThread
.text:004066D6        sub    esp, 18h
.text:004066D9        test   eax, eax
.text:004066D8        jz    short loc_4066EC
.text:004066DB        mov    [esp+2Ch+lpThreadAttributes], eax ; hObject
.text:004066DD        call   CloseHandle

```

(a)



0040669F	\$ 83EC 2C	SUB ESP,2C
004066A2	> C74424 14 00	MOV DWORD PTR SS:[LOCAL.51],0
004066A9	· C74424 10 00	MOV DWORD PTR SS:[LOCAL.61],0
004066B2	· C74424 0C 00	MOV DWORD PTR SS:[LOCAL.71],0
004066B9	· C74424 08 F0	MOV DWORD PTR SS:[LOCAL.81],PERSIST
004066C2	· C74424 04 00	MOV DWORD PTR SS:[LOCAL.91],0
004066CA	· C70424 000000	MOV DWORD PTR SS:[LOCAL.101],0
004066D1	· E8 068F0100	CALL <JMP.&KERNEL32.CreateThread>
004066D6	· 83EC 18	SUB ESP,18
004066D9	· 85C0	TEST EAX,EAX

(b)

Figura 75: Creazione della thread per la persistenza (a) IDA (b) OllyDBG

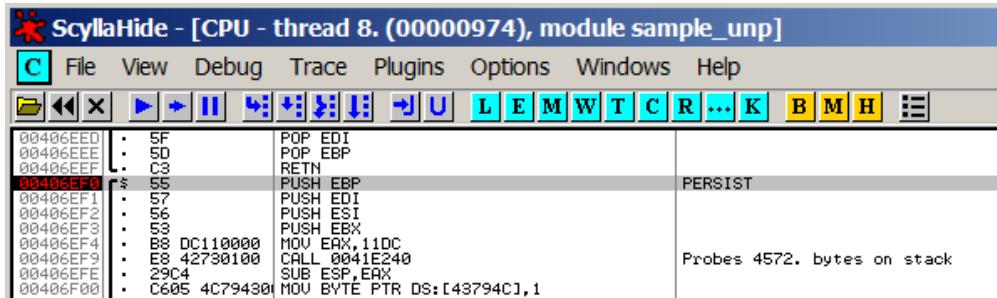


Figura 76: OllyDbg - La funzione `persist` viene eseguita da una thread figlia e non dalla main thread.

### 4.7.2 Funzione persist

La funzione `persist` (sub\_406EF0) si occupa di implementare la persistenza del malware all'interno del sistema e di creare un file batch all'interno della cartella `temp`. In particolare la procedura:

- chiama la subroutine `remove_keys_files` che elimina le modifiche principali apportate al sistema operativo.
- Chiama il metodo `set_registry_keys` che si occupa di settare i valori desiderati alle diverse chiavi del registro Windows.
- Crea una cartella nascosta `C:\Users\Malware\AppData\Roaming\1726961194` (Figura 77). Cerca poi di spostarsi da `C:\Users\Desktop\Samples\Reverse Engineering` alla cartella appena

creata, cambiando di nome. Da notare che la creazione della cartella fallisce perchè era già stata creata in precedenza nel metodo `main_bot`.

- Aggiungere al registro `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce` e `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`, la chiave `*1726961194` e come valore viene impostato il nuovo nome del malware (con eventuale path), in modo che esso venga eseguito ogni volta che viene avviato il sistema, come già visto nella parte di analisi dinamica.
- Crea il file batch chiamato `I1726961194.bat` nella cartella `C:\Users\Malware\AppData\Local\Temp` e lo esegue tramite la subroutine `SHELL32.ShellExecuteA` (Figura 78).

Il malware cerca di implementare la persistenza all'interno del sistema ma ci riesce solo in determinate condizioni. Come detto nella sezione precedente, in base alla data impostata, il malware può cercare di rinominarsi con due nomi distinti, `l` (esecuzione con `sample_unp.exe`) oppure `AdobeARM.exe` (esecuzione con `sample_unp_mod.exe`), prima di spostarsi nella cartella `\Roaming\1726961194`. Nel primo caso, cioè quando la data corrente supera quella di “expiration”, il sample non riesce a implementare la persistenza poichè il nome è invalido (Figura 23). Nel secondo caso invece ha pieno successo (Figura 28b).

```

00407007 | .v 0F85 D2020001 JNE 004072DF
0040700D | .C74424 04 00 MOV DWORD PTR SS:[ARG.1],0
00407015 | .C76424 C0H4: MOV DWORD PTR SS:[ARG.RETADDR],OFFSET
0040701B | .E8 B6858100 CALL <JMP.&KERNEL32.CreateDirectoryA>
00407021 | .83EC 08 SUB ESP,8
00407024 | .C76424 C0H4: MOV DWORD PTR SS:[ARG.RETADDR],OFFSET
00407028 | .E8 59980000 CALL 00411889
00407030 | .C74424 04 FF MOV DWORD PTR SS:[ARG.2],0
00407038 | .C74424 04 FF MOV DWORD PTR SS:[ARG.11],1F01FF
00407040 | .C76424 E0H14: MOV DWORD PTR SS:[ARG.RETADDR],OFFSET
00407047 | .E8 16A40000 CALL 00411462
00407051 | .89F8 MOV EAX,ESI
00407054 | .94F0 TEST ESI,ESI
00407058 | .v JZ SHORT 00407001
0040705B | .C74424 08 00 MOV DWORD PTR SS:[ARG.2],8
0040705D | .C74424 04 E0 MOV DWORD PTR SS:[ARG.11],OFFSET 004381H
00407062 | .C76424 A0H94: MOV DWORD PTR SS:[ARG.RETADDR],OFFSET
00407069 | .E8 B6858100 CALL <JMP.&KERNEL32.MoveFileExA>
0040706E | .83EC 0C SUB ESP,0C
00407071 | .C76424 E0H14: MOV DWORD PTR SS:[ARG.RETADDR],OFFSET
00407078 | .E8 0C800000 CALL 00411889

```

Figura 77: OllyDBG - Creazione della cartella 1726961194 in `C:\Users\Malware\AppData\Roaming`.

#### 4.7.3 Funzione set\_registry\_keys

La funzione `set_registry_keys` (`sub_40670C`) si occupa di impostare i valori di diverse chiavi nel registro Windows. Le seguenti chiavi, e i relativi valori, vengono modificati solamente in `sample_unp_mod.exe` ma non in `sample_unp.exe` (Figura 80):

- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden`
- `HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore\DisableSR`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Folder\SuperHidden\CheckedValue`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\NoWindowsUpdate`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\NoFolderOptions`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Folder\SuperHidden>ShowSuperHidden`

Funzionalità e valore di ogni chiave sono già stati specificati nella parte di analisi dinamica. Le seguenti chiavi non vengono invece modificate perchè le chiavi non esistono nella nostra versione di Windows 7.

- `HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore\RPSessionInterval`

- HKEY\_LOCAL\_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore/RPGlobalInterval
- HKEY\_LOCAL\_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\SystemRestore/RPLifeInterval
- HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications>List

L'unica chiave di registro che viene impostata correttamente è HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Disk\Enum con valore “xobv” (Figura 79).

The screenshot shows assembly code from address 00407082 to 00407188. The code involves various system calls like `MSVCRT.printf`, `SHELL32.ShellExecuteA`, and `MSVCRT.mbscpy`. Annotations highlight specific memory locations and arguments, such as `Arg1 = > 0` and `Arg2 = > 0`. The assembly code includes instructions for reading from memory at `00407098` and writing to memory at `00407188`.

```

00407082: A1 889F4300    MOV EAX,DWORD PTR DS:[439F88]
00407083: 894424 00 40  MOU DWORD PTR SS:[ARG.3],EAX
00407088: C74424 00 40  MOU DWORD PTR SS:[ARG.2],OFFSET 004359
00407093: C74424 00 DC  MOU DWORD PTR SS:[ARG.1],OFFSET 004264
00407098: 891C24          MOU DWORD PTR SS:[ARG.RETADDR],EBX
0040709E: E8 F9820100    CALL <JMP.&msvcrt.sprintf>
004070A3: 891C24          MOU DWORD PTR SS:[ARG.RETADDR],EBX
004070A5: E8 C2H70000    CALL 0041186D
004070AB: 84C0            TEST AL,AL
004070AD: 75 18           JNZ SHORT 004070C7
004070AF: 4E              JMP SHORT 004070FF
004070B1: > C70424 E0A14: MOU DWORD PTR SS:[ARG.RETADDR],OFFSET ASCII "C:\Users\Malware\AppData\Local\Temp"
004070B8: E8 CCA70000    CALL 00411889
004070B9: E8 56A50000    CALL 00411618
004070C2: > E9 C4000000  JMP 00407188
004070C7: > C74424 14 00  MOU DWORD PTR SS:[ARG.5],0
004070CF: C74424 10 00  MOU DWORD PTR SS:[ARG.4],0
004070D7: C74424 00 00  MOU DWORD PTR SS:[ARG.3],0
004070DF: 895C24 08          MOU DWORD PTR SS:[ARG.2],EBX
004070E3: C74424 04 00  MOU DWORD PTR SS:[ARG.1],0
004070EB: C70424 000000  CALL <JMP.&SHELL32.ShellExecuteA>,0
004070F2: E8 00848100    CALL 00407188
004070F7: 83EC 18          SUB ESP,18
004070FA: > E9 80000000  JMP 00407188
004070FF: > 895C24 04          MOU DWORD PTR SS:[ARG.1],EBX
00407103: 8D8424 180601  LEA EAX,[ARG.390]
0040710A: 890424          MOU DWORD PTR SS:[ARG.RETADDR],EAX
0040710D: E8 98820100    CALL <JMP.&msvcrt.strcpy>
00407112: BE E8644200    MOV ESI,OFFSET 004264E8
00407117: B9 16000000    MOV ECX,16
0040711C: 89EF            MOV EDI,EBP
0040711E: F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS
00407120: C74424 04 FE  MOU DWORD PTR SS:[ARG.1],OFFSET 004264
00407128: 8D8424 180601  LEA EAX,[ARG.390]
0040712F: 890424          MOU DWORD PTR SS:[ARG.RETADDR],EAX
00407132: E8 B8820100    CALL <JMP.&msvcrt.fopen>
00407137: 89C6            MOV ESI,EAX
00407139: 89C0            TEST EAX,EAX
0040713B: > 74 4E           JZ SHORT 0040718B
0040713D: 894424 04          MOU DWORD PTR SS:[ARG.1],EAX
00407141: 892C24          MOU DWORD PTR SS:[ARG.RETADDR1],EBP
00407144: E8 B8820100    CALL <JMP.&msvcrt.fputs>
00407149: 893424          MOU DWORD PTR SS:[ARG.RETADDR1],ESI
0040714C: E8 B8820100    CALL <JMP.&msvcrt.fclose>
00407151: C74424 14 00  MOU DWORD PTR SS:[ARG.5],0
00407159: C74424 10 00  MOU DWORD PTR SS:[ARG.4],0
00407161: C74424 00 00  MOU DWORD PTR SS:[ARG.3],0
00407169: 8D8424 180601  LEA EAX,[ARG.390]
00407170: 894424 08          MOU DWORD PTR SS:[ARG.2],EAX
00407174: C74424 04 00  MOU DWORD PTR SS:[ARG.1],0
0040717C: C70424 000000  CALL <JMP.&SHELL32.ShellExecuteA>
00407183: E8 7C830100    CALL 00407188
00407188: 83EC 18          SUB ESP,18

```

Figura 78: OllyDBG - Creazione ed esecuzione del file .bat.

sample_unp.exe	2844	RegOpenKey	HKLMSoftware\Microsoft\WindowsNT\CurrentVersion\SystemRestore	NAME NOT FOUND Desired Access: Write
sample_unp.exe	2844	RegOpenKey	HKLMSoftware\Microsoft\WindowsNT\CurrentVersion\SystemRestore	NAME NOT FOUND Desired Access: Write
sample_unp.exe	2844	RegOpenKey	HKLMSoftware\Microsoft\WindowsNT\CurrentVersion\SystemRestore	NAME NOT FOUND Desired Access: Write
sample_unp.exe	2844	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Services\Disk\Enum	NAME NOT FOUND Desired Access: Write
sample_unp.exe	2844	RegOpenKey	HKLMSystem\CurrentControlSet\Services\Disk\Enum	REPARSE Desired Access: Write
sample_unp.exe	2844	RegSetValue	HKLMSystem\CurrentControlSet\services\Disk\Enum	SUCCESS Desired Access: Write
				SUCCESS Type: REG_SZ, Length: 10, Data: xobv

Figura 79: Procmon - La chiave .../SystemRestore risulta non esistente mentre .../Disk/Enum viene impostato correttamente a “xobv”.

Jump is taken  
Dest=sample\_unp.00406AC0

```

0040689F | . 89F7      | MOV EDI,ESI
004068A1 | . F3:RA     | REP STOS BYTE PTR ES:[EDI]
004068A3 | . 895424 0C | MOV DWORD PTR SS:[ARG.3],EDX
004068A7 | . C74424 08 CA | MOV DWORD PTR SS:[ARG.2],OFFSET 004263 ASCII "%y%m%d"
004068A9 | . C74424 04 14 | MOV DWORD PTR SS:[ARG.1],14
004068B7 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068B9 | . E8 1D8B0100 | CALL <JMP.&msvcrt.strftime>
004068B9 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068B9 | . E8 1D8B0100 | CALL <JMP.&msvcrt.atoi>
004068C2 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068C7 | . 3B05 80C84201 | CMP EAX,DWORD PTR DS:[42C880]
004068C7 | . 804068D0 | JGE 00406AC0
004068D0 | . v 0F8D ED010000 | TEST BX,BX
004068D3 | . 66:85DB     | JNZ SHORT 00406919
004068D6 | . v 75 41     | LEA EAX,[ARG.158]
004068D8 | . 808424 780201 | MOV DWORD PTR SS:[ARG.1],EAX
004068DF | . 894424 04     | MOV DWORD PTR SS:[ARG.11],EAX
004068E3 | . 808424 702D01 | LEA EAX,[ARG.2908]
004068E9 | . 896424     | MOV DWORD PTR SS:[ARG.RETADDR],EAX
004068ED | . E8 BA8A0100 | CALL <JMP.&msvcrt.strcpy>
004068F2 | . 808424 600601 | LEA EAX,[ARG.408]
004068F2 | . 894424 04     | MOV DWORD PTR SS:[ARG.1],EAX

```

src dest MSVCRT.\_mbscopy

(a)

Jump is not taken  
Dest=sample\_unp.00406919

```

0040689F | . 89F7      | MOV EDI,ESI
004068A1 | . F3:RA     | REP STOS BYTE PTR ES:[EDI]
004068A3 | . 895424 0C | MOV DWORD PTR SS:[ARG.3],EDX
004068A7 | . C74424 08 CA | MOV DWORD PTR SS:[ARG.2],OFFSET 004263 ASCII "%y%m%d"
004068A9 | . C74424 04 14 | MOV DWORD PTR SS:[ARG.1],14
004068B7 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068B9 | . E8 1D8B0100 | CALL <JMP.&msvcrt.strftime>
004068B9 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068B9 | . E8 1D8B0100 | CALL <JMP.&msvcrt.atoi>
004068C2 | . 893424     | MOV DWORD PTR SS:[ARG.RETADDR],ESI
004068C7 | . 3B05 80C84201 | CMP EAX,DWORD PTR DS:[42C880]
004068C7 | . 804068D0 | JGE 00406AC0
004068D0 | . v 0F8D ED010000 | TEST BX,BX
004068D3 | . 66:85DB     | JNZ SHORT 00406919
004068D6 | . v 75 41     | LEA EAX,[ARG.158]
004068D8 | . 808424 780201 | MOV DWORD PTR SS:[ARG.1],EAX
004068DF | . 894424 04     | MOV DWORD PTR SS:[ARG.11],EAX
004068E3 | . 808424 702D01 | LEA EAX,[ARG.2908]
004068E9 | . 896424     | MOV DWORD PTR SS:[ARG.RETADDR],EAX
004068ED | . E8 BA8A0100 | CALL <JMP.&msvcrt.strcpy>
004068F2 | . 808424 600601 | LEA EAX,[ARG.408]
004068F2 | . 894424 04     | MOV DWORD PTR SS:[ARG.1],EAX

```

src dest MSVCRT.\_mbscopy

(b)

Figura 80: (a) OllyDBG - I valori di registro non vengono impostati nel caso (a) (il jump salta quella parte di codice) mentre le modifiche vengono fatte nel caso (b).

## 4.8 Creazione e gestione della copia del malware

Le funzioni proposte in questa sezione servono per permettere al sample di creare una copia di se stesso. In questo modo all'interno del sistema host ci sono sempre due istanze del malware in esecuzione in parallelo. Se uno dei due viene eliminato o terminato, l'altro prende il controllo e reinfetta il computer creando un'altra copia e mandandola in esecuzione. Per gestire la sincronizzazione dei due eseguibili vengono usati i mutex objects di Windows.

**Definizione.** Un mutex object è un oggetto di sincronizzazione il cui stato è impostato su “signaled” quando non è in possesso di alcuna thread e “nonsignaled” in caso contrario. Solo una thread alla volta può possedere un oggetto mutex, il cui nome deriva dal fatto che è utile nel coordinare l'accesso reciprocamente esclusivo a una risorsa condivisa. Ad esempio, per impedire che due thread scrivano contemporaneamente nella memoria condivisa, ogni thread attende il possesso di un oggetto mutex prima di eseguire il codice per accedere alla memoria. Dopo aver scritto nella memoria condivisa, la thread rilascia il mutex [22].

### 4.8.1 Funzione handle\_main\_copy

La funzione `handle_main_copy` (sub\_4078E0) gestisce la presenza all'interno del sistema di due istanze del malware in esecuzione parallela attraverso l'utilizzo di mutex objects condivisi tra i due processi. In particolare il metodo controlla l'esistenza di due mutex objects chiamati rispettivamente `MAIN_1726961194`, associato all'esecuzione di `AdobeARM.exe`, e `BACKUP_1726961194`, associato a `ADService.exe`. Per controllare la presenza di un mutex e per crearne uno vengono usate rispettivamente le subroutine `check_exists_mutex` (Figura 81) e `create_mutex` (Figura 82). In base alla presenza o meno di questi oggetti avremo un comportamento differente:

- se entrambi i mutex non esistono, siamo quindi alla prima esecuzione del malware, allora viene creato il mutex `MAIN_1726961194` ed eseguita la subroutine `thread_create_copy` per creare la copia (Figura 83).
- Se il mutex principale esiste ma non esiste il mutex di backup, implicando che l'esecuzione stia avvenendo da `ADService.exe`, allora viene creato il secondo mutex object e il programma si occuperà di controllare in loop la presenza del mutex di `MAIN_1726961194` (`check_recreate_main`, sub\_40CD88). Qualora il processo principale non fosse più in esecuzione, elimina, se presente, il file, ricrea un'altra copia e lo fa partire.
- Se esiste solo `BACKUP_1726961194` significa che l'istanza principale è stata terminata e ricreata. Viene quindi creato di nuovo il mutex `MAIN_1726961194`.
- Se entrambi i mutex esistono già, quindi sia `AdobeARM.exe` che `ADService.exe` sono già in esecuzione, allora il programma viene semplicemente terminato.

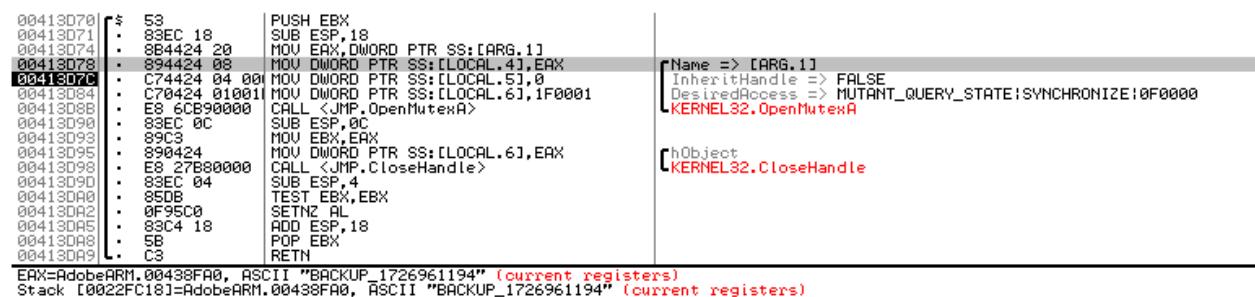


Figura 81: OllyDBG - Controllo dell'esistenza del mutex di backup

#### 4.8.2 Funzione thread\_create\_copy

La funzione `thread_create_copy` (sub\_40CA85) si occupa della creazione di una thread, la quale andrà ad eseguire la subroutine `create_copy` (Figura 84).

#### 4.8.3 Funzione create\_copy

La funzione `create_copy` (sub\_40CB79) viene utilizzata per creare una copia del malware all'interno del file system. La copia, che verrà rinominata `ADService.exe`, può trovarsi in 3 diversi percorsi:

- C:\Users\Malware\AppData\Local\Temp,
- C:\Users\Malware,
- C:\ProgramData.

La scelta del path è totalmente randomica, viene infatti usata la subroutine `generate_random_num` per decidere la cartella di destinazione. Una volta che la copia è stata generata, tramite la funzione `KERNEL32.CopyFileA`, viene mandata in esecuzione con la subroutine `KERNEL32.CreateProcessA` (Figura 85). Viene poi eseguita la funzione `check_backup`.

#### 4.8.4 Funzione check\_backup

Il metodo `check_backup` (sub\_40CAF2) si occupa principalmente di controllare, in loop, che il mutex `BACKUP_1726961194` esista, quindi di verificare che `ADService.exe` sia correntemente in esecuzione. Nel

The screenshot shows assembly code in OllyDbg. The tooltip on the right side of the screen provides details about the parameters for the `KERNEL32.CreateMutexA` function call. The parameters are:

- `Name`: `InitialOwner => TRUE`
- `pSecurity`: `> NULL`
- `hObject`: `KERNEL32.CloseHandle`

Figura 82: OllyDBG - Creazione del mutex per il processo principale

The screenshot shows assembly code in OllyDbg. The tooltip on the right side of the screen provides details about the parameters for the `KERNEL32.CreateThread` function call. The parameters are:

- `pThreadId`: `=> NULL`
- `CreationFlags`: `=> 0`
- `Parameter`: `=> NULL`
- `StartAddress`: `=> AdobeARM.40CB79`
- `StackSize`: `=> 0`
- `pSecurity`: `=> NULL`

Figura 83: OllyDBG - Creazione del mutex e della thread per la copia.

The screenshot shows assembly code in OllyDbg. The tooltip on the right side of the screen provides details about the parameters for the `KERNEL32.CreateThread` function call. The parameters are:

- `pThreadId`: `=> NULL`
- `CreationFlags`: `=> 0`
- `Parameter`: `=> NULL`
- `StartAddress`: `=> AdobeARM.40CB79`
- `StackSize`: `=> 0`
- `pSecurity`: `=> NULL`

Figura 84: OllyDBG - Creazione della thread per `create_copy`

```

0040CCB6 : 83EC 04 SUB ESP, 4
0040CCB6 : C74424 08 00 MOU DWORD PTR SS:[LOCAL.164],0
0040CCC1 : 895C24 08 00 LEA EBX, [LOCAL.132]
0040CCC8 : 895C24 04 MOU DWORD PTR SS:[LOCAL.165],EBX
0040CCC8 : C78424 A09F4: MOU DWORD PTR SS:[LOCAL.166],00439FA0
0040CCD3 : E8 B4280100 CALL <JMP.CopyFileA>
0040CCD8 : 89EC 00 SUB ESP, 0C
0040CCD9 : 894424 34 LER ERX,[LOCAL.153]
0040CCD9 : C74424 24 MOU DWORD PTR SS:[LOCAL.157],EAX
0040CCD9 : 894424 44 LER ERX,[LOCAL.149]
0040CE7 : 894424 20 MOU DWORD PTR SS:[LOCAL.158],EAX
0040CEB : C74424 1C 00 MOU DWORD PTR SS:[LOCAL.159],0
0040CF5 : C74424 18 00 MOU DWORD PTR SS:[LOCAL.160],0
0040CFB : C74424 14 00 MOU DWORD PTR SS:[LOCAL.161],0
0040CD83 : C74424 10 00 MOU DWORD PTR SS:[LOCAL.162],0
0040CD88 : C74424 08 00 MOU DWORD PTR SS:[LOCAL.163],0
0040CD93 : C74424 08 00 MOU DWORD PTR SS:[LOCAL.164],0
0040CD1B : 895C24 04 MOU DWORD PTR SS:[LOCAL.165],EBX
0040CD1F : C78424 00000 MOU DWORD PTR SS:[LOCAL.166],0
0040CD26 : E8 B4280100 CALL <JMP.CreateProcessA>
0040CD2B : 88EC 28 SUB ESP, 28
0040CD2E : 85C0 TEST ERX, ERX
EBX=01EFFD78, ASCII "C:\ProgramData\ADService.exe"
Stack 00EFECF4J=AdobeARM.0043A2E5

```

Figura 85: OllyDBG - Creazione ed esecuzione di *ADService.exe*

momento in cui non trova l'istanza di backup chiama la subroutine `create_thread_copy` per ricrearla (Figura 86). Da notare che il malware non cerca di rieseguire il processo terminato ma crea direttamente un'altra copia da mandare in esecuzione.

```

.text:0040CB33
.text:0040CB33 loc_40CB33:
.text:0040CB33
.text:0040CB33 ; CODE XREF: check_backup+23+j
.text:0040CB33 ; check_backup+6E+j
    mov    [esp+1Ch+dwMilliseconds], offset byte_438FA0 ; LPCSTR
    call   check_exists_mutex
    test   al, al
    jnz   short loc_40CB4A
    call   thread_create_copy
    jmp   short loc_40CB62

```

Figura 86: OllyDBG - Se il mutex di backup non viene più trovato allora viene ricreato il processo secondario *ADService.exe*

## 4.9 Funzioni di scansione di cartelle/registri ed eliminazione file sospetti

In questa sezione parleremo delle subroutine usate dal malware per controllare la presenza di file sospetti, all'interno di determinate cartelle del file system e di alcune chiavi di registro, e la loro successiva eliminazione. Tutti i metodi qui presenti non vengono eseguiti dalla thread principale ma da una thread figlia. Per testare i metodi si è creata una copia di *ADService.exe* nella cartella C:\ProgramData per imitare il comportamento del malware nelle situazioni descritte nella parte di analisi dinamica. In questa sezione vedremo inoltre la ragione dell'esistenza del file da 2kb di cui si è tanto parlato nella parte precedente.

### 4.9.1 Funzione scan\_directories

La funzione `scan_directories` (`sub_404DF0`) viene utilizzata per determinare le cartelle che dovranno essere controllate dal metodo `search_for_programs`. Esse sono:

- C:\Users\Malware\AppData\Local\Roaming,
- C:\Users\Malware\AppData\Local\Temp,
- C:\Users\Malware,
- C:\ProgramData.

Si può notare che queste sono le cartelle utilizzate dal malware durante la sua esecuzione.

### 4.9.2 Funzione search\_for\_programs

Il metodo `search_for_programs` (`sub_404B94`) viene utilizzato per analizzare tutti i file all'interno di una cartella per determinare se essi siano sospetti o meno. Se sono ritenuti tali vengono eliminati dal sistema. In particolare il malware:

- cerca file con estensione *.com*, *.exe*, *.src*, *.vbs*, *.cmd*, ignorando tutti gli altri (Figura 87).
- Individuato un file sospetto viene controllato che non sia il programma stesso oppure *AdobeARM.exe* e che sia un file nascosto oppure di sistema, attraverso (*GetFileAttributesA*).
- Se il programma è in esecuzione, si cerca di terminarlo con la subroutine *kill\_process*.
- Si “rompe” il file attraverso il metodo *break\_file*.

Nel nostro caso la copia di *ADService.exe* è in esecuzione. Supponiamo, in base ai criteri utilizzati per decidere se un file sia sospetto o meno, che il sample cerchi se all’interno della macchina siano presenti altri malware, più precisamente altri bot.

The screenshot shows assembly code from address 00404C00 to 00404C80. The code includes several calls to MSUCRT.strstr, which are highlighted in red. Red boxes on the right side of the screen indicate search results for specific strings:

- search => OFFSET LOCAL.75 string => "."
   
MSUCRT.strstr
- search => OFFSET LOCAL.75 string => ".com"
   
MSUCRT.strstr
- search => OFFSET LOCAL.75 string => ".exe"
   
MSUCRT.strstr
- search => OFFSET LOCAL.75 string => ".src"
   
MSUCRT.strstr
- search => OFFSET LOCAL.75 string => ".vbs"
   
MSUCRT.strstr
- search => OFFSET LOCAL.75 string => ".cmd"
   
MSUCRT.strstr

Figura 87: OllyDBG - Il malware cerca file con determinate estensioni nella cartella.

#### 4.9.3 kill\_process

La funzione *kill\_process* (sub\_413F48) crea un file batch, K1726961194.bat (di cui si è ampiamente parlato nella sezione di analisi dinamica), per eliminare un determinato processo conoscendone il PID (Figura 88). Se dopo l’esecuzione del file .bat il programma risulta ancora in esecuzione, si cerca nuovamente di terminarlo con il metodo *TerminateProcess*. Nel nostro test accade qualcosa di non previsto che però rispecchia i comportamenti visti in analisi dinamica. Nonostante il file batch venga eseguito, il processo *ADService.exe* non viene ucciso, rimanendo visibile in Task Manager. Il successivo tentativo di controllare se il processo è ancora in esecuzione tramite la subroutine di sistema *OpenProcess*, con accesso *PROCESS\_TERMINATE*, però fallisce e, anche se il processo è ancora attivo, viene ritornato un handle NULL (Figura 89). Viene dunque saltato un nuovo tentativo di terminazione. Stranamente, creando una copia di K1726961194.bat e facendola eseguire manualmente, il processo *ADService.exe* viene effettivamente terminato.

#### 4.9.4 break\_file

La funzione *break\_file* (sub\_404911), dato un determinato file, si occupa principalmente di:

- rinominare il file aggiungendo un’estensione casuale di 8 caratteri generati dalla funzione *gen\_rand\_8\_char* (Figura 90).
- Usando l’operazione *MoveFileExA* di Windows, predisporre il file originale e il file che verrà successivamente creato ad essere eliminati dal file system al reboot del sistema (Figura 91). Questo accade perché *MoveFileExA* con parametro *dwFlags* = *MOVEFILE\_DELAY\_UNTIL\_REBOOT* ritarda l’operazione di spostamento fino al reboot del sistema ma *lpNewFileName* = *NULL* indica che il file deve essere eliminato [21]. Questo corrisponde, nell’analisi dinamica, al comportamento relativo alla chiave “PendingFileRenameOperations”.

```

00414003: 890424 MOV DWORD PTR SS:[LOCAL_582],EAX
00414006: E9 49630000 CALL <JMP.&svrvt._itoa>
00414008: 897424 04 MOU DWORD PTR SS:[LOCAL_581],ESI
0041400F: 892C24 MOU DWORD PTR SS:[LOCAL_582],EBP
00414012: EB D5B30000 CALL <JMP.&svrvt.strcat>
00414017: 89EF MOU EDI,EBP
00414019: B8 00 MOU AL,0
0041401B: B9 FFFFFFFF MOU ECX,-1
00414020: F2:AE REPNE SCAS BYTE PTR ES:[EDI]
00414022: F7D1 NOT ECX
00414024: 8D4400 FF LEA EAX,[ECX+EBP-1]
00414028: BE CB8C4200 MOU ESI,OFFSET 00428CCB
0041402D: 89C7 MOU EDI,EAX
00414032: F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
00414036: A1 899F4200 MOU EAX,DWORD PTR DS:[439F88]
00414038: B9 894424 0C MOU DWORD PTR SS:[LOCAL_5791],EAX
0041403F: C74424 08 40 MOU DWORD PTR SS:[LOCAL_5801],OFFSET 004:
00414047: C74424 04 DC MOU DWORD PTR SS:[LOCAL_5811],OFFSET 004:
0041404F: 8D7424 2C LEA ESI,[LOCAL_571]
00414053: B93424 MOU DWORD PTR SS:[LOCAL_5821],ESI
00414056: EB 41B30000 CALL <JMP.&svrvt.sprintf>
0041405B: C74424 00 E8 MOU DWORD PTR SS:[LOCAL_5811],OFFSET 004:
00414063: B93424 MOU DWORD PTR SS:[LOCAL_5821],ESI
00414065: B8 00 MOU EAX,FOPEN
0041406C: 89C0 TEST EDX,EAX
0041406D: F3C0 TEST EDX,EAX
0041406F: 74 52 JZ SHORT 004140C3
00414071: 894424 04 MOU DWORD PTR SS:[LOCAL_5811],EAX
00414075: B80424 180509 LEA EAX,[LOCAL_256]
0041407C: B90424 MOU DWORD PTR SS:[LOCAL_5821],EAX
0041407F: EB 78B30000 CALL <JMP.&svrvt.fputs> ...

```

value => [ARG.1]  
**MSVCRT.\_itoa**  
 Arg2  
 Arg1 => OFFSET LOCAL.256  
**msvrt.\_mbscat**  
 ASCII "@echo off&TASKKILL /F /T /PID 2508"  
 ASCII ">NULL&DEL %>NULL"  
 %%li => [439F88] = 1726961194.  
 %%s => ";\Users\Malware\AppData\Local\Temp"  
 format => "%s\%lli.bat"  
 s => OFFSET LOCAL.571  
**MSVCRT.printf**  
 mode => "w"  
 filename  
**MSVCRT fopen**  
 stream  
 string => OFFSET LOCAL.256  
**MSVCRT.fputs**

Figura 88: OllyDBG - Creazione del file batch. Il PID del processo *ADService.exe* è 2508

```

File Machine View Input Devices Help
ScyllaHide - [CPU - thread 5. (00000F34), module sample_unp]
File View Debug Trace Plugins Options Windows Help
L E M W T C R ... K B M H
0041408C: C74424 14 00 MOU DWORD PTR SS:[LOCAL_5771],0
00414094: C74424 10 00 MOU DWORD PTR SS:[LOCAL_5781],0
0041409C: C74424 0C 00 MOU DWORD PTR SS:[LOCAL_5791],0
0041409E: B804424 2C LEA EAX,[LOCAL_571]
004140A8: B93424 08 MOU DWORD PTR SS:[LOCAL_5801],EAX
004140AC: C74424 04 00 MOU DWORD PTR SS:[LOCAL_5811],0
004140B4: C70424 0000 MOU DWORD PTR SS:[LOCAL_5821],0
004140BB: E8 44B40000 CALL <JMP.&SHELL32.ShellExecuteA>
004140C0: 83EC 18 SUB ESP,18
004140C3: > C70424 F40100 MOU DWORD PTR SS:[LOCAL_5821],1F4
004140C9: E8 14F0FFFF CALL 004130E3
004140CF: 05 F0000000 ADD EDX,0FA
004140D4: B90424 MOU DWORD PTR SS:[LOCAL_5821],EAX
004140D7: E8 F0B40000 CALL <JMP.&KERNEL32.Sleep>
004140DC: 83EC 04 SUB ESP,4
004140E0: E8 2B860000 CALL <JMP.&KERNEL32.GetCurrentProcessId>
004140E4: 398424 200900 CMP DWORD PTR SS:[ARG.1],EAX
004140E6: 74 42 JE SHORT 0041412F
004140ED: B888424 200900 MOU EDX,DWORD PTR SS:[ARG.1]
004140F4: B93424 08 MOU DWORD PTR SS:[LOCAL_5801],EAX
004140F8: C74424 04 00 MOU DWORD PTR SS:[LOCAL_5811],0
00414100: C70424 010000 MOU DWORD PTR SS:[LOCAL_5821],1
00414107: E8 20B50000 CALL <JMP.&KERNEL32.OpenProcess>
0041410C: 89EC 0C SUB ESP,0C
0041410F: B9C6 MOU ESI,EAX
00414111: 89C0 TEST EAX,EAX
00414113: 74 1F JZ SHORT 00414134

```

KERNEL32.OpenProcess returned EAX = NULL  
 Imm=0000000C (decimal 12.)  
 ESP=0231EAB0 (current registers)

Arg6 => 0  
 Arg5 => 0  
 Arg4 => 0  
 Arg3 => OFFSET LOCAL.571  
 Arg2 => 0  
 Arg1 => 0  
**SHELL32.ShellExecuteA**  
 Time  
**KERNEL32.Sleep**  
 ProcessID => [ARG.1]  
 InheritHandle => FALSE  
 Access => PROCESS\_TERMINATE  
**KERNEL32.OpenProcess**

Figura 89: OllyDBG - *ADService.exe* è ancora in esecuzione ma OpenProcess ritorna NULL.

- Creare un nuovo file con il nome originale. In esso vengono inseriti 2000 byte generati casualmente (Figura 93).
- Controlla che esista la system tray, attraverso FindWindow("Shell\_TrayWnd"), e duplica l'handle del file originale con DuplicateHandle di Windows.

Con queste operazioni il processo originale, ora rinominato (Figura 92), una volta terminato prima del reboot del sistema, non può più essere rieseguito a causa dell'estensione casuale non riconosciuta mentre il file da 2kb con il suo stesso nome non può essere eseguito perché contenente semplici byte randomici. In conclusione dunque questi comportamenti servono per impedire all'utente di avviare il programma originale, da una parte perchè ora quest'ultimo ha un'estensione strana che non lo rende direttamente eseguibile, dall'altra il file da 2kb inganna l'utente che pensa sia il programma vero. Una volta riavviato il sistema entrambi i file vengono eliminati. Nel nostro caso *ADService.exe* viene rinominato in *ADService.exe.wtturro*. Viene poi creato un nuovo file chiamato sempre *ADService.exe* che contenente 2000 byte causali. Stranamente, come visto in analisi dinamica, Windows prova ad eseguire questo file spazzatura con NTVDM.exe, probabilmente perchè non trova altre alternative che permettano di eseguirlo, visto che ha comunque estensione *exe*.

```

0040490A | . 66:C74433 FF MOU WORD PTR DS:[ESI+EBX-1],2E
00404901 | . E8 B2F00000 CALL 00413B48
00404906 | . 894C24 04 MOU DWORD PTR SS:[ESP+4],EAX
00404908 | . 891C24 MOU DWORD PTR SS:[ESP1],EBX
0040490D | . E8 40AA00100 CALL <JMP_&msvcrt._mbscat>
0040490E | . C74424 08 08 MOU DWORD PTR SS:[ESP+8],8
0040490F | . 895C24 04 MOU DWORD PTR SS:[ESP+4],EBX
0040490E | . 8855 08 MOU EDX,DWORD PTR SS:[ARG_1]
00404901 | . 891424 MOU DWORD PTR SS:[ESP],EDX
00404904 | . E8 6BAC0100 CALL <JMP_&KERNEL32.MoveFileExA>
00404909 | . 83FC 0C SUB ESP,0C
0040490C | . C74424 08 04 MOU DWORD PTR SS:[ESP+8],4
00404904 | . C74424 04 00 MOU DWORD PTR SS:[ESP+4],0
00404905 | . 891C24 MOU DWORD PTR SS:[ESP1],EBX
00404909 | . E8 59AC0100 CALL <JMP_&KERNEL32.MoveFileExA>
0040490A | . 83EC 0C SUB ESP,0C
0040490D | . C74424 08 04 MOU DWORD PTR SS:[ESP+8],4
0040490F | . C74424 04 00 MOU DWORD PTR SS:[ESP+4],0
00404901 | . 890424 MOU EAX,DWORD PTR SS:[ARG_1]
00404902 | . E8 32AC0100 CALL <JMP_&KERNEL32.MoveFileExA>
00404903 | . 83EF 0C SUB ESP,0C

```

EBX=0231EB7C, ASCII "C:\ProgramData\ADService.exe.wtturrrro"  
Stack [0231EB64]=sample\_unp.004048F9, ASCII "wtturrrro"

Figura 90: OllyDBG - *ADService.exe* viene rinominato aggiungendo un' estensione casuale.

(a)

0231EB54	00000008
0231EB58	0231F3C8
0231EB5C	004049B9
0231EB60	0231EB7C
0231EB64	00000000
0231EB68	00000004
0231EB6C	0231F68B

RETURN from kernel32.MoveFileExA to sample\_unp.004049B9  
Existing = "C:\ProgramData\ADService.exe.wtturrrro"  
New = NULL  
Flags = MOVEFILE\_DELAY\_UNTIL\_REBOOT

(b)

0231EB54	00000004
0231EB58	0231F3C8
0231EB5C	004049D4
0231EB60	0231F3FC
0231EB64	00000000
0231EB68	00000004
0231EB6C	0231F68B
0231EB70	00000000

RETURN from kernel32.MoveFileExA to sample\_unp.004049D4  
Existing = "C:\ProgramData\ADService.exe"  
New = NULL  
Flags = MOVEFILE\_DELAY\_UNTIL\_REBOOT

Figura 91: OllyDBG - Il file originale rinominato (a) e il file da 2Kb (b) vengono predisposti per essere eliminati al reboot del sistema.

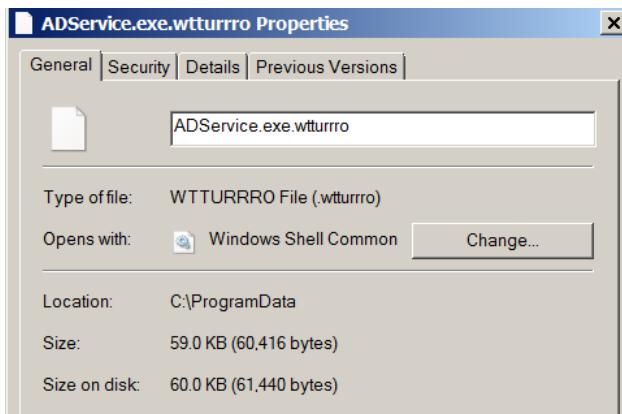


Figura 92: Task Manager - *ADService.exe* è ancora in esecuzione ma con il nuovo nome.

#### 4.9.5 scan\_for\_registries

La funzione `scan_registries` (sub\_404F0C) è molto simile a `scan_directories` ma invece di controllare cartelle controlla alcune chiavi di registro (nello specifico Run e RunOnce) per vedere se ci sono, all'interno dei suoi valori, programmi da eseguire che potrebbero essere sospetti. Un esempio è ciò che succede con *VBoxTray.exe*, presente nel registro Run.

The screenshot shows two assembly code snippets from OllyDbg, labeled (a) and (b), illustrating the creation and modification of a file.

**(a)** Assembly code for file creation:

```

004049F2 | . 83EC 0C SUB ESP, 0C
004049F5 | . C74424 18 00 MOU DWORD PTR SS:[ESP+18], 0
004049FD | . C74424 14 00 MOU DWORD PTR SS:[ESP+14], 0
004049F9 | . C74424 10 00 MOU DWORD PTR SS:[ESP+10], 0
004049FD | . C74424 0C 00 MOU DWORD PTR SS:[ESP+C], 0
004049F9 | . C74424 08 00 MOU DWORD PTR SS:[ESP+8], 0
004049FD | . C74424 04 00 MOU DWORD PTR SS:[ESP+4], 40000000
004049F5 | . MOU EDX, DWORD PTR SS:[ARG_11]
0040492B | . E8 84AB0100 CALL <JMP.&KERNEL32.CreateFileA>
00404930 | . 83EC 1C SUB ESP, 1C

```

Annotations for (a) show parameters for CreateFileA:

- `hTemplate => NULL`
- `Attributes => FILE_ATTRIBUTE_NORMAL`
- `CreateDisposition => CREATE_ALWAYS`
- `pSecurity => NULL`
- `ShareMode => 0`
- `DesiredAccess => GENERIC_WRITE`
- `ASCII "C:\ProgramData\ADService.exe"`
- `FileName => [ARG_11]`
- `KERNEL32.CreateFileA`

**(b)** Assembly code for file write:

```

00404A4B | . BE FF000000 MOU ESI, 0FF
00404A50 | . E8 B7A90100 CALL <JMP.&msvcrt.rand>
00404A55 | . 99 CDQ
00404A56 | . F7FE IDIV ESI
00404A59 | . 8813 MOU BYTE PTR DS:[EBV], DL
00404A5A | . 43 INC EBV
00404A5B | . 39FB CMP EBV, EDI
00404A5D | . 75 F1 JNE SHORT 00404A50
00404A5E | . C785 10F8FFFF MOU DWORD PTR SS:[LOCAL_508], 0
00404A63 | . C74424 10 00 MOU DWORD PTR SS:[ESP+10], 0
00404A71 | . 8085 10F8FFFF LEA EAX, [LOCAL_509]
00404A77 | . 894424 0C MOU DWORD PTR SS:[ESP+0C], EAX
00404A7D | . C74424 00 D0 MOU DWORD PTR SS:[ESP+8], 7D0
00404A83 | . 8085 10F8FFFF LEA EAX, [LOCAL_506]
00404A89 | . 894424 04 MOU DWORD PTR SS:[ESP+4], EAX
00404A9D | . 8B85 04F8FFFF MOU EAX, DWORD PTR SS:[LOCAL_511]
00404A9E | . 896424 MOU DWORD PTR SS:[ESP], EAX
00404A96 | . E8 21AB0100 CALL <JMP.&KERNEL32.WriteFile>
00404A9B | . 83EC 14 SUB ESP, 14

```

Annotations for (b) show parameters for WriteFile:

- `cMSVCRT.rand`
- `pOverlapped => NULL`
- `pBytesWritten => OFFSET LOCAL_508`
- `Size => 2000`
- `Buffer => OFFSET LOCAL_506`
- `hFile => [LOCAL_511]`
- `KERNEL32.WriteFile`

Figura 93: OllyDBG - (a) Creazione di un nuovo file vuoto chiamato *ADService.exe* (b) Write del file con 2000 byte randomici.

## 4.10 Funzioni relative a IRC

### 4.10.1 Funzione connect\_to irc

`connect_to irc` (sub\_4083E7) è la funzione principale per la connessione IRC e la gestione dei messaggi inviati/ricevuti. Il metodo si occupa di:

- fare una richiesta DNS per l'indirizzo IP di *irc.spotchat.org* (Figura 40).
- Aprire un socket verso l'esterno (`ws2_32.socket`) e fare la successiva connessione al server IRC (`ws2_32.connect`).
- Effettuare l'iscrizione al canale dopo aver generato randomicamente nick (subroutine `generate_random_nick`) e user tramite la funzione `send_to irc`, la quale si occupa di mandare un determinato messaggio tramite IRC.
- Visualizzare i valori impostati per ricevere comandi da IRC (subroutine `values_for irc`).
- Aprire un socket per la ricezione dei messaggi dal canale e interpretarli tramite la funzione `parse irc command`.

Come notato nella prima parte di analisi dinamica, il malware sembra andare in loop in questa funzione continuando a fare continue richieste DNS per lo stesso indirizzo IP, *irc.spotchat.org*. Questo è causato, come in altri casi, dal problema della data. Siccome la data corrente è maggiore della data di “expiration”, il malware procede a chiudere il socket aperto precedentemente, il quale porta di conseguenza al fallimento del tentativo di connessione al server IRC che avviene subito dopo. *sample\_unp.exe* dunque ripete l'apertura del socket e la richiesta DNS finché la connessione non ha successo, quindi all'infinito. Usando invece *sample\_unp\_mod.exe* (la versione modificata per bypassare il controllo sulla “expiration”) si riesce a fare la connessione al server IRC, al quale si procede dunque a mandare due messaggi di tipo USER e NICK per identificarsi. Il server risponderà con tutta una serie di informazioni. Il malware apre poi un socket al server per ricevere le risposte e invia un messaggio di tipo JOIN e MODE per iscriversi al canale `#sdds` con i settaggi specificati, come già spiegato nell'analisi dinamica. Rimane poi continuamente in ascolto sul socket per ricevere comandi.

#### 4.10.2 Funzione send\_to\_irc

Il metodo `send_to_irc` (sub\_408B81) serve per inviare un messaggio testuale al canale IRC tramite la funzione di libreria `ws2_32.send`. In Figura 94 è presente un esempio di invio di un messaggio di tipo NICK contenente il nickname casuale generato.

```

004088833 | . 83EC 14 SUB ESP,14
004088836 | . 8B5C24 20 MOV EBX,DWORD PTR SS:[ARG_1]
004088839 | . 803D 4B794301 CMP BYTE PTR DS:[43794B],0
004088831 | < 74 04 JE SHORT 004088B47
004088833 | . C74424 04 46 MOV DWORD PTR SS:[LOCAL_5],OFFSET 0043B
004088836 | . 891C24 MOV DWORD PTR SS:[LOCAL_6],EBX
004088839 | . E9 19680100 CALL <JMP.&msvcrt._strcat>
00408883E | . 89C0 TEST EBX,EBX
00408883F | < 75 53 JNZ SHORT 004088BFA
004088840 | > 895C24 08 MOV DWORD PTR SS:[LOCAL_4],EBX
004088843 | . C74424 04 C4 MOV DWORD PTR SS:[LOCAL_5],OFFSET 00426
004088835 | . C74424 885241 MOV DWORD PTR SS:[LOCAL_6],OFFSET 00435
004088838 | . E8 DD670100 CALL <JMP.&msvcrt._sprintf>
00408883B | . BF 89524300 MOV EDI,OFFSET 00435280
00408883C | . 80 00 ASCII "NICK _[GBR]A:L!WIN7!x86!1cJzwwwurro"
00408883D | . 89C0 FFFFFFFF MOV ECX,-1
00408883E | . 8B240E REPNE STOS BYTE PTR ES:[EDI]
00408883F | . F7D1 NOT ECX
004088840 | . 4991 DEC ECX
004088841 | . C74424 0C 00 MOV DWORD PTR SS:[LOCAL_3],0
004088842 | . 894C24 08 MOV DWORD PTR SS:[LOCAL_4],ECX
004088843 | . C74424 04 00 MOV DWORD PTR SS:[LOCAL_5],OFFSET 00435
004088844 | . A1 80544300 MOV EBX,DWORD PTR DS:[4354801]
004088845 | . 890424 MOV DWORD PTR SS:[LOCAL_6],EBX
004088846 | . F115 3C1C420 CALL <JMP.&ws2_32.send>
004088847 | . 83EC 10 SUB ESP,10
004088848 | . R9 482B4300 MOV DWORD PTR DS:[43B2B48],EAX
004088849 | > 83C4 14 ADD ESP,14
00408884A | . 5B POP EBX
00408884B | . 5F POP EDI
00408884C | . C3 RETN
00408884D | $ A1 20A34300 MOV EBX,DWORD PTR DS:[43A3201]
00408884E | . 83F8 01 CMP EBX,1
00408884F | < 74 24 JE SHORT 004088C2E
004088850 | . 83F8 02 CMP EBX,2
004088851 | . 83C4 40 ADD ESP,40
004088852 | . 83C4 40 ADD ESP,40

```

sample\_unp.00408C00(guessed void)

[0042C83C]=75836C19 (ws2\_32.send)

Figura 94: OllyDBG - Invio di messaggi al canale IRC.

#### 4.10.3 Funzione generate\_random\_nick

La funzione `generate_random_nick` (sub\_408C58) ha il compito di generare dei nick randomici da far utilizzare al bot per le interazioni con il canale IRC. Oltre a testo casuale (creato da `gen_rand_8_char`) usa anche informazioni relative al computer host (per esempio il sistema operativo e l'architettura) per creare nick univoci. Nella Figura 95 si possono vedere diversi esempi di nick generati.

```

00408D04E | . 8D4424 34 LEA EAX,[LOCAL_5]
00408D052 | . 894424 08 MOV DWORD PTR SS:[LOCAL_16],EAX
00408D055 | . C74424 04 F1 MOV DWORD PTR SS:[LOCAL_17],OFFSET 00421
00408D056 | . C70424 A09B41 MOV DWORD PTR SS:[LOCAL_18],OFFSET 00431
00408D057 | . E8 32660100 CALL <JMP.&msvcrt._sprintf>
00408D058 | . BB A09B4300 MOV EBX,OFFSET 00439BA0
00408D059 | . 83C4 40 ADD ESP,40

```

(a)

```

00408D04E | . 8D4424 34 LEA EAX,[LOCAL_5]
00408D052 | . 894424 08 MOV DWORD PTR SS:[LOCAL_16],EAX
00408D055 | . C74424 04 F1 MOV DWORD PTR SS:[LOCAL_17],OFFSET 00421
00408D056 | . C70424 A09B41 MOV DWORD PTR SS:[LOCAL_18],OFFSET 00431
00408D057 | . E8 32660100 CALL <JMP.&msvcrt._sprintf>
00408D058 | . BB A09B4300 MOV EBX,OFFSET 00439BA0
00408D059 | . 83C4 40 ADD ESP,40

```

(b)

Figura 95: (a) OllyDBG - Creazione di nick randomici per IRC.

#### 4.10.4 Funzione values\_for\_irc

Il metodo `values_for_irc` (sub\_4089FC) imposta i valori di diverse variabili per IRC tramite la funzione di libreria `sprintf`. Le stringhe generate, visibili nella Figura 96, vengono utilizzate come “trigger” per il malware. Infatti esso esegue determinati comportamenti solo se riceve un messaggio contenente una di queste stringhe dal canale IRC. Ogni messaggio scambiato tra client e server contiene un codice oppure un nome univoco che ne identifica il tipo, per esempio NICK, PRIVMSG, NOTICE, etc., e ha un formato diverso. In particolare:

- 376 <nick>: il codice 376 indica la fine di MOTD (Message of the day), un messaggio inviato dal server al momento della connessione di un client e contenente le principali caratteristiche e restrizioni

dell'IRC server e l'IRC Network. <nick> corrisponde al nick del client. Il messaggio viene inviato dal server all'instaurazione della connessione [11].

- 422 <nick>: il codice 422 indica un errore relativo al MOTD. Il messaggio viene inviato dal server se il file di MOTD è mancante.
- 332 <nick>: il codice 332 indica un messaggio inviato dal server per indicare il topic impostato sul canale, se presente.
- @my.vhost TOPIC #sdds :!
- @my.vhost PRIVMSG #
- @my.vhost PRIVMSG <nick> :! : le ultime 3 sono stringhe predefinite con cui un utente dal canale può mandare un comando, scritto dopo # o :!, da far eseguire al malware.

Da notare che solo ultima delle tre stringhe invia un comando a uno specifico bot mentre con le altre due il comando viene eseguito da tutti i malware in ascolto sul canale.

```

004089FC  $ 83EC 2C    SUB ESP, 2C
004089FF  . C74424 08 60    MOV DWORD PTR SS:[LOCAL.8],OFFSET 00432
00408A07  . C74424 04 18    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408A0F  . C70424 60234    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408A16  . E8 81698100    CALL <JMP.&msvcrt.printf>
00408A1B  . C74424 08 60    MOV DWORD PTR SS:[LOCAL.8],OFFSET 00432
00408A23  . C74424 04 20    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408A2B  . C70424 601F4    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408A32  . E8 65698100    CALL <JMP.&msvcrt.printf>
00408A37  . C74424 08 60    MOV DWORD PTR SS:[LOCAL.8],OFFSET 00432
00408A3F  . C74424 04 28    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408A47  . C70424 60184    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408A4E  . E8 49698100    CALL <JMP.&msvcrt.printf>
00408A53  . C74424 08 A0    MOV DWORD PTR SS:[LOCAL.7],OFFSET 0042D
00408A5B  . C74424 08 A0    MOV DWORD PTR SS:[LOCAL.8],OFFSET 0042D
00408A63  . C74424 04 31    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408A6B  . C70424 60174    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408A72  . E8 25698100    CALL <JMP.&msvcrt.printf>
00408A77  . C74424 0C 46    MOV DWORD PTR SS:[LOCAL.7],OFFSET 0043B
00408A7F  . C74424 08 A0    MOV DWORD PTR SS:[LOCAL.8],OFFSET 0042D
00408A87  . C74424 04 41    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408A8F  . C70424 60134    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408A96  . E8 01698100    CALL <JMP.&msvcrt.printf>
00408A9B  . C74424 08 60    MOV DWORD PTR SS:[LOCAL.6],OFFSET 00432
00408A93  . C74424 0C 46    MOV DWORD PTR SS:[LOCAL.7],OFFSET 0043B
00408AAB  . C74424 08 A0    MOV DWORD PTR SS:[LOCAL.8],OFFSET 0042D
00408ABB  . C74424 04 4A    MOV DWORD PTR SS:[LOCAL.9],OFFSET 00426
00408AC2  . C70424 600F4    MOV DWORD PTR SS:[LOCAL.10],OFFSET 0043
00408AC7  . E8 D56880100    CALL <JMP.&msvcrt.printf>
00408ACA  . 83C4 2C    ADD ESP, 2C
00408ACA  . C3    RETN

```

sample\_unp\_004089FC(guessed void)  
`%s` => "[GBR]!L!WIN7!x86!1c]qnnnklli"  
format => "%s" & my.vhost  
`%s` => "422 %s"  
`%s` => "332 %s"  
`%s` => "332 \_[GBR]!L!WIN7!x86!1c]qnnnklli"  
`%s` => "422 \_[GBR]!L!WIN7!x86!1c]qnnnklli"  
`%s` => "332 \_[GBR]!L!WIN7!x86!1c]qnnnklli"  
`%s` => "#sdds"  
`%s` => "my.vhost"  
format => "%s %s :!"  
`%s` => "@my.vhost TOPIC %s :!"  
`%s` => "@my.vhost PRIVMSG %s :!"  
`%s` => "PRIVMSG"  
`%s` => "my.vhost"  
format => "%s %s #"  
`%s` => "@my.vhost PRIVMSG #"  
`%s` => "[GBR]!L!WIN7!x86!1c]qnnnklli"  
`%s` => "PRIVMSG"  
`%s` => "my.vhost"  
format => "%s %s :!"  
`%s` => "@my.vhost PRIVMSG \_[GBR]!L!WIN7!x86!1c]qnnnklli :!"  
`%s` => "@my.vhost PRIVMSG \_[GBR]!L!WIN7!x86!1c]qnnnklli ;!"  
`%s` => "MSVCRT.printf"

Figura 96: OllyDBG - stringhe valide per inviare i comandi dal canale IRC.

#### 4.10.5 Funzione parse\_irc\_command

Il metodo `parse_irc_command` (sub\_408D76) si occupa di ispezionare i vari messaggi che arrivano dal server/canale. In base al tipo di messaggio e al loro contenuto esegue differenti azioni. A seguito un elenco dei comportamenti più interessanti:

- se il messaggio è di tipo PRIVMSG (qualsiasi testo scritto nella chat da un utente, che non sia un comando di IRC, viene inviato come PRIVMSG) e non contiene nessuna delle ultime tre stringhe indicate sopra allora viene ignorato. Se invece ne contiene almeno una il metodo estrae il comando associato, che si trova dopo :! (Figura 45a e Figura 45b), e chiama la funzione `exec_command`.
- Se il messaggio è di tipo 332, cioè il server sta inviando al client il topic settato nel canale quando esso entra la prima volta, il malware controlla che il topic contenga la stringa :!, estrae il comando associato e lo passa alla funzione `exec_command`.
- Se il messaggio è di tipo KICK, cioè il bot è stato cacciato, esso prova a unirsi di nuovo al canale.
- Se il messaggio è relativo a un nickname già in uso, ne crea un altro e riprova ad entrare in #sdds.
- Se il messaggio è di tipo PING, inviato dal server, viene inviato un PONG di risposta. Questo meccanismo viene usato per determinare se una connessione sia morta o non responsiva. Se il server non riceve nessun PONG entro un determinato tempo la connessione viene chiusa.

#### 4.10.6 Funzione exec\_command

Il metodo `exec_command` (sub\_40CFFF) viene utilizzato per gestire i vari comandi inviati al bot tramite il canale IRC. Esso si occupa di controllare che il comando arrivato sia effettivamente valido e di eseguire le relative operazioni. Come si può notare dal control flow graph generato da IDA (Figura 97) la funzione è enorme, con una moltitudine di if innestati. Questo è dovuto al fatto che i comandi accettati sono innumerevoli. Facendo una comparazione tra le stringhe presenti nel metodo e la documentazione Athena [1], contenente l'elenco di tutti i comandi che si possono dare al bot, si nota che esse siano le stesse. Il malware dunque può eseguire tutti i comportamenti descritti in Athena.

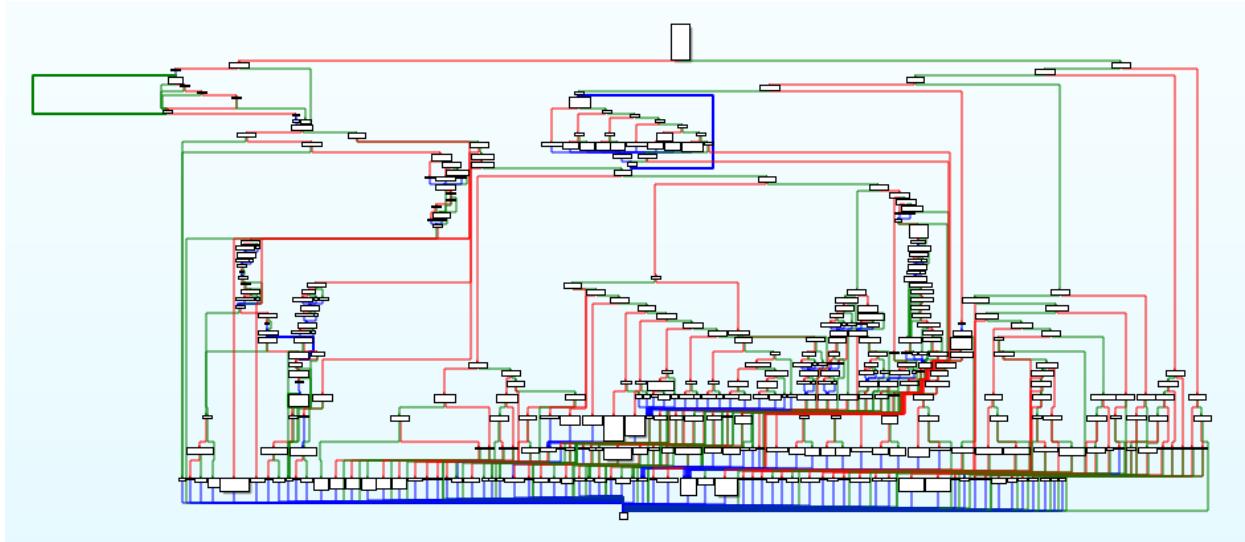


Figura 97: IDA - Control Flow Graph della funzione `exec_command`.

#### 4.11 Funzioni di disinistallazione e rimozione

I metodi in questa sezione servono al malware per eliminarsi dal sistema operativo senza lasciare tracce visibili del proprio passaggio. Naturalmente il sample normalmente non cerca di disinistallare se stesso, a meno di errori o comando specifico dal canale IRC, quindi per poter analizzare la funzione `uninstall_program` si è dovuto forzarne l'esecuzione con OllyDBG. Nella Figura 98 si può notare che è stato modificato un jump nel metodo `connect_to irc` per far eseguire la subroutine che ci interessa.



Figura 98: OllyDBG - Il JNE è stato cambiato in JE per forzare il malware ad eseguire le procedure di disinistallazione dal sistema.

##### 4.11.1 Funzione `uninstall_program`

La funzione `uninstall_program` (sub\_4072F1) si occupa di rimuovere il malware e le modifiche che ha creato all'interno del sistema host. Per fare ciò usa la subroutine `remove_keys_files`, che elimina chiavi, directory e file precedentemente creati, per poi generare un file bat, U1726961194.bat, all'interno di C:

\Users\Malware\AppData\Local\Temp. Esso, come già visto approfonditamente nella sezione di analisi dinamica, non fa altro che tentare di eliminare se stesso fintanto che si trova presente nella cartella.

#### 4.11.2 Funzione remove\_keys\_files

Il metodo `remove_keys_files` (sub\_4063E4) si occupa di rimuovere dal sistema operativo chiavi, file e directory precedentemente create. In particolare:

- elimina entrambe le chiavi \*1726961194 dal registro Windows usando la subroutine `sub_41485D` (Figura 99);
- elimina il malware e la cartella 1726961194 dal file system (Figura 100);
- elimina le possibili copie di *ADService.exe* generate precedentemente nei percorsi C:\Users\Malware\AppData\Local\Temp, C:\Users\Malware e C:\ProgramData (Figura 101).

```

0041485D 5 58 PUSH EBX
0041485E 5 58 SUB ESP, 38
00414861 5 C74424 2C 00 MOV DWORD PTR SS:[LOCAL.3], 0
00414869 5 8D4424 2C LEA EAX, [LOCAL.3]
00414871 5 C74424 18 00 MOV DWORD PTR SS:[LOCAL.10], EAX
00414871 5 C74424 0C 02 MOV DWORD PTR SS:[LOCAL.11], 2
00414871 5 C74424 08 00 MOV DWORD PTR SS:[LOCAL.12], 0
00414881 5 8B4424 44 MOV EAX, DWORD PTR SS:[ARG.2]
00414885 5 894424 04 MOV DWORD PTR SS:[LOCAL.13], EAX
00414889 5 8B4424 48 MOV EAX, DWORD PTR SS:[ARG.1]
00414890 5 896424 04 MOV DWORD PTR SS:[LOCAL.14], EAX
00414896 5 E8 F74B0000 CALL <JMP.&ADVAPI32.RegOpenKeyExA>
00414897 5 83EC 14 SUB ESP, 14
00414898 5 85C0 TEST EAX, EAX
00414899 5 C74424 0C JNP SHORT 004148D0
0041489C 5 8B4424 48 MOV EAX, DWORD PTR SS:[ARG.3]
0041489E 5 894424 04 MOV DWORD PTR SS:[LOCAL.13], EAX
004148A0 5 8B4424 2C MOV EAX, DWORD PTR SS:[LOCAL.3]
004148A1 5 896424 04 MOV DWORD PTR SS:[LOCAL.14], EAX
004148A2 5 E8 444C0000 CALL <JMP.&ADVAPI32.RegDeleteValueA>
004148B0 5 83EC 08 SUB ESP, 8
004148B3 5 85C0 TEST EAX, EAX
004148B5 5 0F94C3 SETZ BL
004148BF 5 8B4424 2C MOV EAX, DWORD PTR SS:[LOCAL.3]
004148C0 5 896424 04 SUB ESP, 4
004148C4 5 E8 D84B0000 CALL <JMP.&ADVAPI32.RegFlushKey>
004148C6 5 83EC 04 SUB ESP, 4
004148C7 5 C74424 2C MOV EAX, DWORD PTR SS:[LOCAL.3]
004148CB 5 896424 04 MOV DWORD PTR SS:[LOCAL.14], EAX
004148CE 5 E8 D14B0000 CALL <JMP.&ADVAPI32.RegCloseKey>
004148D0 5 83EC 04 SUB ESP, 4
004148D1 5 C74424 02 JNP SHORT 004148D0
004148D2 5 8908 00 MOV AL, BL
004148D4 5 8908 00 MOV AL, BL
004148DC 5 83C4 38 ADD ESP, 38
004148DF 5 8B 00 POP EBX
004148E0 5 C3 RETN

```

Figura 99: OllyDbg - La subroutine `sub_41485D` rimuove le chiavi indicate dal registro Windows.

```

00406490 5 C70424 E00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:ASCII "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
00406491 5 E8 B90F0000 CALL 004119D4
00406492 5 C70424 E00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:ASCII "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
00406493 5 E8 1FB50000 CALL 004119D4
00406495 5 C70424 E00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:Name => "C:\Users\Malware\AppData\Roaming\1726961194\AdobeARM.exe"
00406496 5 E8 13910100 CALL <JMP.&KERNEL32.DeleteFileA> KERNEL32.DeleteFileA
004064C1 5 83EC 04 SUB ESP, 4
004064C4 5 C74424 08 00 MOV DWORD PTR SS:[LOCAL.3881], 0
004064CC 5 C74424 04 FF MOV DWORD PTR SS:[LOCAL.3891], FF01FF
004064D4 5 C70424 C00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:ASCII "C:\Users\Malware\AppData\Roaming\1726961194"
004064D5 5 E8 B90F0000 CALL 004119D4
004064E0 5 C70424 C00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:ASCII "C:\Users\Malware\AppData\Roaming\1726961194"
004064E7 5 E8 E5B40000 CALL 004119D4
004064EC 5 C70424 C00141 MOV DWORD PTR SS:[LOCAL.3901], OFFSET 004:Name => "C:\Users\Malware\AppData\Roaming\1726961194"
004064F3 5 E8 DC900100 CALL <JMP.&KERNEL32.DeleteFileA> KERNEL32.DeleteFileA
004064F8 5 83EC 04 SUB ESP, 4

```

Figura 100: OllyDbg - `sample_unp.exe` e la cartella 1726961194 vengono eliminati.

00406683	: 894424 04	MOV DWORD PTR SS:[LOCAL_3891],EAX	Arc2
00406687	: 891C24	MOV DWORD PTR SS:[LOCAL_3901],EBX	Arc1
0040668B	: E8 5D080100	CALL <JMP.&msvcrt._strcat>	nsvoirt._mbscat
0040668F	: 891C24	MOV DWORD PTR SS:[LOCAL_3901],EBX	Name
00406692	: E8 3D0F0100	CALL <JMP.&KERNEL32.DeleteFileA>	KERNEL32.DeleteFileA
00406697	: 83EC 04	SUB ESP,4	
0040669A	: E9 6FFFFFFF	JMP 0040650E	
0040669F	: 83EC 2C	SUB ESP,2C	
004066A2	> C74424 10 00	MOV DWORD PTR SS:[LOCAL_51],0	pThreadId => NULL
004066A6	C74424 10 00	MOV DWORD PTR SS:[LOCAL_61],0	CreationFlags => 0
004066B2	C74424 00 00	MOV DWORD PTR SS:[LOCAL_71],0	Parameter => 0
004066C2	C74424 00 00	MOV DWORD PTR SS:[LOCAL_81],0,PERSIST	StartAddress => sample_unp.PERSIST
004066C8	C74424 04 00	MOV DWORD PTR SS:[LOCAL_91],0	StackSize => 0
004066D1	E8 060F0100	CALL <JMP.&KERNEL32.CreateThread>	pStackSize => NULL
004066D6	: 83EC 18	SUB ESP,18	KERNEL32.CreateThread
004066D9	: 85C0	TEST EAX,EAX	
	31C9	JE SHORT .LBB0_2	

Figura 101: OllyDBG - Il malware cerca di eliminare le varie copie di *ADService.exe*.

## 4.12 Miscellaneous

#### 4.12.1 Funzione gen\_rand\_8\_char

La funzione `gen_rand_8_char` (sub\_413A48) genera una stringa di 8 caratteri alfabetici in minuscolo (Figura 102).

```
.text:00413A48 gen_rand_8_char proc near ; CODE XREF: break_file+80+p
.text:00413A48
.text:00413A48
.text:00413A48 var_1C      = dword ptr -1Ch
.text:00413A48
.text:00413A48     push    esi
.text:00413A49     push    ebx
.text:00413A4A     sub     esp, 14h
.text:00413A4D     mov     ebx, offset unk_4828F9
.text:00413A52     mov     esi, offset unk_482901
.text:00413A57
.text:00413A57 loc_413A57:          ; CODE XREF: gen_rand_8_char+23+j
.text:00413A57     mov     [esp+1Ch+var_1C], 1Ah
.text:00413A5E     call    generate_random_numb
.text:00413A63     add     eax, 61h
.text:00413A66     mov     [ebx], al
.text:00413A68     inc     ebx
.text:00413A69     cmp     ebx, esi
.text:00413A6B     jnz    short loc_413A57
.text:00413A6D     mov     eax, offset unk_4828F9
.text:00413A72     add     esp, 14h
.text:00413A75     pop     ebx
.text:00413A76     pop     esi
.text:00413A77     retn
.text:00413A77 gen_rand_8_char endp
```

Figura 102: IDA - Creazione di una stringa con 8 caratteri ascii.

#### 4.12.2 Funzione generate\_random\_num

Il metodo `generate_random_numb` (`sub_4078E0`) viene utilizzato per generare un numero casuale, usando `rand` e `srand`, tra 0 e il parametro dato in input (Figura 103).

```
.text:0041313F loc_41313F:                 ; CODE XREF: generate_random_numb+3C+j
.text:0041313F     mov     edx, 1505h
.text:0041313F                                         ; CODE XREF: generate_random_numb+5A+j
.text:00413144 loc_413144:                 ; CODE XREF: generate_random_numb+5A+j
.text:00413144     cmp     edx, ds:dword_43B764
.text:00413144     jz      short loc_41315B
.text:00413144     call    rand
.text:00413151     mov     ecx, 0Ah
.text:00413156     cdq
.text:00413157     idiv   ecx
.text:00413159     jmp     short loc_413188
```

Figura 103: IDA - Creazione di un numero pseudocasuale usando `rand`.

#### 4.12.3 Funzione set\_irc\_client

La funzione `set_irc_client` (`sub_409DD5`) imposta il client IRC da utilizzare da un elenco predefinito:

- mIRC v7.27 Khaled Mardam-Bey
- xchat 2.8.8 Linux 3.2.0-4-amd64 [x86\_64/1.10GHz/SMP]
- irssi v0.8.15
- <http://www.mibbit.com> ajax IRC Client:3972:3972
- HexChat 2.9.3 [x64] / Windows 7 [3.31GHz]
- ZNC 0.202 - <http://znc.in>
- lightIRC.com 1.3 Build 118, Okt 22 2012 12:07 on Windows 7

Il client viene scelto in maniera casuale, la funzione `generate_random_numb` infatti ritorna un numero, tra 0 e 140, in base al quale verrà scelto il client da utilizzare (Figura 104).

```
.text:00409DD5 get_irc_client proc near          ; CODE XREF: sub_40A280+10D+p
.text:00409DD5
.text:00409DD5 var_1C      = dword ptr -1Ch
.text:00409DD5
.text:00409DD5 push    edi
.text:00409DD5 push    esi
.text:00409DD5 sub     esp, 14h
.text:00409DD5 mov     [esp+1ch+var_1C], 8Ch
.text:00409DD5 call    generate_random_numb
.text:00409DE1 cmp     eax, 13h
.text:00409DE1 ja     short loc_409E03
.text:00409DE1 mov     eax, offset unk_42C220
.text:00409DE1 mov     esi, offset aMircV727Khaled ; "mIRC v7.27 Khaled Mardam-Bey"
.text:00409DF0 mov     ecx, 1Dh
.text:00409DF5 mov     edi, eax
.text:00409DFC rep    movsb
.text:00409DFE jmp    loc_409E98
.text:00409E03 ; -----
.text:00409E03 loc_409E03:                      ; CODE XREF: get_irc_client+14+j
.text:00409E03 cmp     eax, 27h
.text:00409E03 ja     short loc_409E1D
.text:00409E06 mov     eax, offset unk_42C220
.text:00409E08 mov     esi, offset aXchat288Linux3 ; "xchat 2.8.8 Linux 3.2.0-4-amd64 [x86_64"...
.text:00409E0D mov     ecx, 35h
.text:00409E12 mov     edi, eax
.text:00409E17 rep    movsb
.text:00409E19 jmp    short loc_409E98
```

Figura 104: IDA - Scelta dell'IRC Client da utilizzare.

### 4.13 Athena vs sample.exe

Dopo un'attenta analisi è risultato che, nonostante il malware sia fortemente simile al codice presente nella repository, alcune funzionalità presenti in quest'ultimo non sono presenti nel primo e viceversa. Le principali differenze tra i due sono:

- La struttura del main del malware è diversa rispetto a quello di Athena mentre la struttura generale del codice risulta simile.
- Alcune funzionalità mancano totalmente dal malware come il pannello e la command tool per selezionare i comandi da far eseguire al bot, ma questo può essere banalmente perché i comandi verranno inviati solamente tramite il canale IRC. Il malware sembra quindi basarsi principalmente sul codice sorgente del bot (*Athena/Source-bot/Source*) ma anche qui si sono notate delle discrepanze. Per esempio se in Athena è possibile fare la connessione a http in alternativa al client IRC, in *sample\_unp.exe* questo non è possibile.
- Nel sorgente del malware sono state aggiunte due funzioni `tls_callback`.
- Il valore di molte variabili globali e locali è stato modificato, per esempio il canale e il server IRC sono diversi, così come lo “standard expiration date”.

#### 4.13.1 Conclusioni

Se in un primo momento si pensava che il malware si basasse sulla repository GitHub di Athena, ora si crede fortemente che sia il contrario. Probabilmente è lo stesso autore del bot che ha reso pubblico il codice sorgente del malware, arricchendolo di nuove funzionalità. Questa ipotesi è avvalorata da diversi indizi:

- La repository di Athena è stata creata il 25 Settembre 2015 ma lo “standard expiration date” del malware è del 15 Dicembre 2013. Se il sample si basasse su di essa, sarebbe abbastanza bizzarro che un utente si sia preso la briga di comprendere tutto il codice e apportare le eventuali modifiche per creare un bot che non funziona.
- Una buona parte dei comandi del bot riguardano le IRC wars. Esse erano molto popolari verso la fine degli anni 90’ e nei primi anni del 2000 [17] ma sono ormai scomparse anche a causa del rapido declino dell’utilizzo di IRC dal 2003 [13].
- Spotchat.org è stato rilasciato il 20 Maggio del 2005 quindi negli anni in cui le IRC wars erano ancora in qualche modo popolari.
- Come visto nella funzione `set_registry_keys`, il sample controlla alcune specifiche chiavi di registro che in Windows 7, rilasciato nel 2009, non esistono più. Controlla inoltre che il sistema operativo sia versione 5, quindi Windows XP. Se il malware fosse stato creato dopo il 2015 sarebbe piuttosto insolito che vada a controllare versioni così dattate di Windows.
- Dalla data di compilazione, visibile su PEStudio (Figura 3), risulta che l’eseguibile risalga al 2013, tuttavia il codice dell’eseguibile potrebbe essere più datato.

Tutti questi indizi ci portano a credere che il malware sia stato creato intorno al 2005, dopo il rilascio di Spotchat e quando le bot wars e IRC erano ancora popolari.

## Riferimenti bibliografici

- [1] GRBower. 3val/athena, 2015. [www.github.com/3val/Athena](http://www.github.com/3val/Athena).
- [2] 20 aug win32/atrax.a. <https://thisissecurity.stormshield.com/2014/08/20/win32atrax-a/>.
- [3] 5 ways to view, schedule delete or move files on next windows bootup. <https://www.raymond.cc/blog/why-are-you-asked-to-restart-after-install-or-uninstall-software/>.
- [4] Archiviazione thread-local. <https://docs.microsoft.com/it-it/cpp/c-language/thread-local-storage?view=vs-2019>.
- [5] Critical section objects. <https://docs.microsoft.com/en-us/windows/win32/sync/critical-section-objects>.
- [6] Domaintools: informazioni di registrazione di nomi di dominio. <http://whois.domaintools.com/spotchat.org>.
- [7] getenv, \_wgetenv. <https://docs.microsoft.com/it-it/cpp/c-runtime-library/reference/getenv-wgetenv?view=vs-2019>.
- [8] How malware defends itself using tls callback functions. <https://isc.sans.edu/diary/How+Malware+Defends+Itself+Using+TLS+Callback+Functions/6655>.
- [9] Html.it. [www.download.html.it](http://www.download.html.it).
- [10] Il software. [www.ilsoftware.it](http://www.ilsoftware.it).
- [11] Internet relay chat protocol. <https://tools.ietf.org/html/rfc1459>.
- [12] Interpreting the pendingfilerenameoperations registry key. <https://forensicatorj.wordpress.com/2014/06/25/interpreting-the-pendingfilerenameoperations-registry-key-for-forensics/>.
- [13] Irc is dead, long live irc. <https://royal.pingdom.com/irc-is-dead-long-live-irc/>.
- [14] Irc/comandi. <https://it.wikibooks.org/wiki/IRC/Comandi>.
- [15] Joe security - anti-vm gone wrong. <https://www.joesecurity.org/blog/5208960489995723>.
- [16] List of microsoft windows versions. [https://en.wikipedia.org/wiki/List\\_of\\_Microsoft\\_Windows\\_versions](https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions).
- [17] Liste des irc war. [http://assiste.com.free.fr/p/abc/b/liste\\_irc\\_war.php](http://assiste.com.free.fr/p/abc/b/liste_irc_war.php).
- [18] Main thread causes an exception - how to bypass that? <https://reverseengineering.stackexchange.com/questions/11140/main-thread-causes-an-exception-how-to-bypass-that>.
- [19] Malicious application compatibility shims. <https://www.blackhat.com/docs/eu-15/materials/eu-15-Pierce-Defending-Against-Malicious-Application-Compatibility-Shims-wp.pdf>.
- [20] Microsoft documentation (link es. ws2\_32.dll). <https://docs.microsoft.com/en-us/windows/win32/winsock/initialization-2>.
- [21] Movefileexa function. <https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefileexa>.
- [22] Mutex objects. <https://docs.microsoft.com/it-it/windows/win32/sync/mutex-objects>.
- [23] Nowindowsupdate. [http://systemmanager.ru/win2k\\_registry.en/92848.htm](http://systemmanager.ru/win2k_registry.en/92848.htm).
- [24] Ollydbg 2.01. <http://www.ollydbg.de/version2.html>.
- [25] Ollydbg.de. [www.ollydbg.de](http://www.ollydbg.de).

- [26] Petr-akhlamov/ntvdm. <https://reactos.org/wiki/User:Petr-akhlamov/NTVDM>.
- [27] Report file 2kb. <https://www.virustotal.com/gui/file/9ca35e1a6498c0721e43ec27e3f95d3060fda6b3c9c6ce4b4c0ba517bb2f4d1c/detection>.
- [28] Sandbox. [https://it.wikipedia.org/wiki/Sandbox\\_\(sicurezza\\_informatica\)](https://it.wikipedia.org/wiki/Sandbox_(sicurezza_informatica)).
- [29] Wikipedia. [www.wikipedia.org](http://www.wikipedia.org).
- [30] \_Stoner. Athena irc documentation. [https://github.com/3val/Athena/blob/master/AthenaIRC\\_Documentation.pdf](https://github.com/3val/Athena/blob/master/AthenaIRC_Documentation.pdf).