# Conti

## Analisi malware


aprile 2022

TLP:WHITE

# Indice

# 1 Executive summary

Nell'ambito dell'attività istituzionale dell'Agenzia per la Cybersicurezza Nazionale è stata effettuata un'analisi tecnica approfondita sul malware *Conti,* una minaccia ransomware di livello avanzato, al pari di ransomware del calibro di *DarkSide (BlackMatter)*, *Ryuk*, *Egregor*, *Maze* e dei non più attivi *Sodinokibi/REvil* e *NetWalker*, osservata per la prima volta a dicembre 2019 e distribuita secondo il modello del *Big Game Hunting*, la nota metodologia di selezione delle vittime che punta a colpire organizzazioni particolarmente sensibili ai tempi di inattività, pertanto maggiormente inclini al pagamento del riscatto.

Il ransomware, che nei poco più di due anni di vita ha dimostrato di poter efficacemente perpetrare il suo scopo estorsivo, colpendo più di mille organizzazioni nei soli Stati Uniti[1], è noto per la rapidità con cui viene distribuito nelle reti della vittima dopo l'accesso iniziale. *Conti*, come molti altri ransomware del panorama mondiale, viene operato secondo un modello *Ransomware-as-a-Service (RaaS)* e sfrutta tecniche di *double-extorsion* supportate da un *data leak site (DLS)* per indurre la vittima a pagare il riscatto; di recente, il gruppo di distribuzione ha avviato anche una campagna di vendita e acquisto[2] di accessi alle reti delle vittime compromesse (*Initial Access Broker*). Da gennaio 2019, *Crowdstrike* e *Mandiant/FireEye* tracciano il principale *threat actor* che distribuisce il ransomware rispettivamente con le nomenclature *Wizard Spider* e *FIN12*, ma esso è anche semplicemente noto come *Conti Ransomware Gang*. Il gruppo è uno dei principali *eCrime actor* odierni e fa uso anche di altre famiglie, quali *Trickbot*, *BazarLoader*, *IcedID* (*BokBot*) e il ransomware *Ryuk*.

Il 28 febbraio 2022 una fonte anonima su Twitter[3] ha rilasciato il codice sorgente di *Conti*, congiuntamente ad altri file attinenti all'arsenale del *threat actor*, quali file di log, messaggi di chat interni e codice sorgente del *decryptor*. A seguito di tale evento sono stati pubblicati su fonti aperte diversi documenti tecnici di descrizione del ransomware, tuttavia tali sorgenti si riferivano in realtà alla versione *v2* del ransomware, risalente al 15 settembre 2020. Il 20 marzo 2022, la stessa fonte anonima ha rilasciato un nuovo archivio, contenente questa volta il codice sorgente dell'ultima

[1] https://www.cisa.gov/uscert/ncas/alerts/aa21-265a
[2] https://blog.google/threat-analysis-group/exposing-initial-access-broker-ties-conti/
[3] https://twitter.com/ContiLeaks

versione nota (v*3*) in uso alla *Gang*. Il gruppo, a seguito del *data leak*, ha cambiato i domini e l'infrastruttura del *DLS* e si configura come una minaccia ancora attiva.

In questo report verranno descritte le caratteristiche distintive del ransomware, quali tecniche di *discovery* delle unità di rete e metodologie di offuscamento, con particolare attenzione alle nuove funzionalità introdotte nell'ultimo anno e mezzo; l'analisi è stata effettuata su entrambi i sorgenti rilasciati dalla fonte anonima, ma anche a partire da attività di *reverse engineering* compiute su alcuni eseguibili reperiti da una nota piattaforma di *repository* malware.

La pubblicazione dell'arsenale di un *threat actor*, compreso il codice sorgente di un malware avanzato come *Conti*, può portare ad una nuova, nonché parallela, proliferazione del ransomware a causa dell'adozione da parte di altri gruppi meno evoluti, della tecnologia malevola, pertanto si fornisce altresì una regola di rilevazione del malware in formato *YARA*.

# 2 Analisi Tecnica

I sorgenti di settembre 2020 suggeriscono che il ransomware è stato sviluppato in *C++* su ambiente di sviluppo *Microsoft Visual Studio 2015* con piattaforma target principale *Windows 10* ma con supporto alle versioni precedenti fino a *Windows XP*. Sebbene la maggior parte degli eseguibili di *Conti* siano in formato `.exe x86`, è bene notare che negli ultimi mesi il processo di distribuzione è stato visto anche con utilizzo di `.dll` e architettura `x64`. I nuovi sorgenti, apparentemente datati gennaio 2022, separano infatti la soluzione di Visual Studio (che questa volta si riferisce alla versione 2017) nei progetti *cryptor* e *cryptor_dll* (oltre che *decryptor*) per la produzione di release semanticamente identiche ma rispettivamente in formato `.exe` e `.dll`.

Al fine di valutare l'attendibilità della data dei sorgenti è stata effettuata un'analisi di similarità tra alcuni sample usati realmente in attacchi passati e gli eseguibili ottenuti compilando i sorgenti di settembre 2020 e gennaio 2022. I risultati, mostrati in *Figura 1*, provano che la versione compilata dai sorgenti di settembre 2020 è molto simile agli eseguibili di agosto e ottobre 2020 reperibili da fonti aperte, viceversa, la versione compilata dai sorgenti di gennaio 2022 è molto simile soltanto all'eseguibile di novembre 2021.

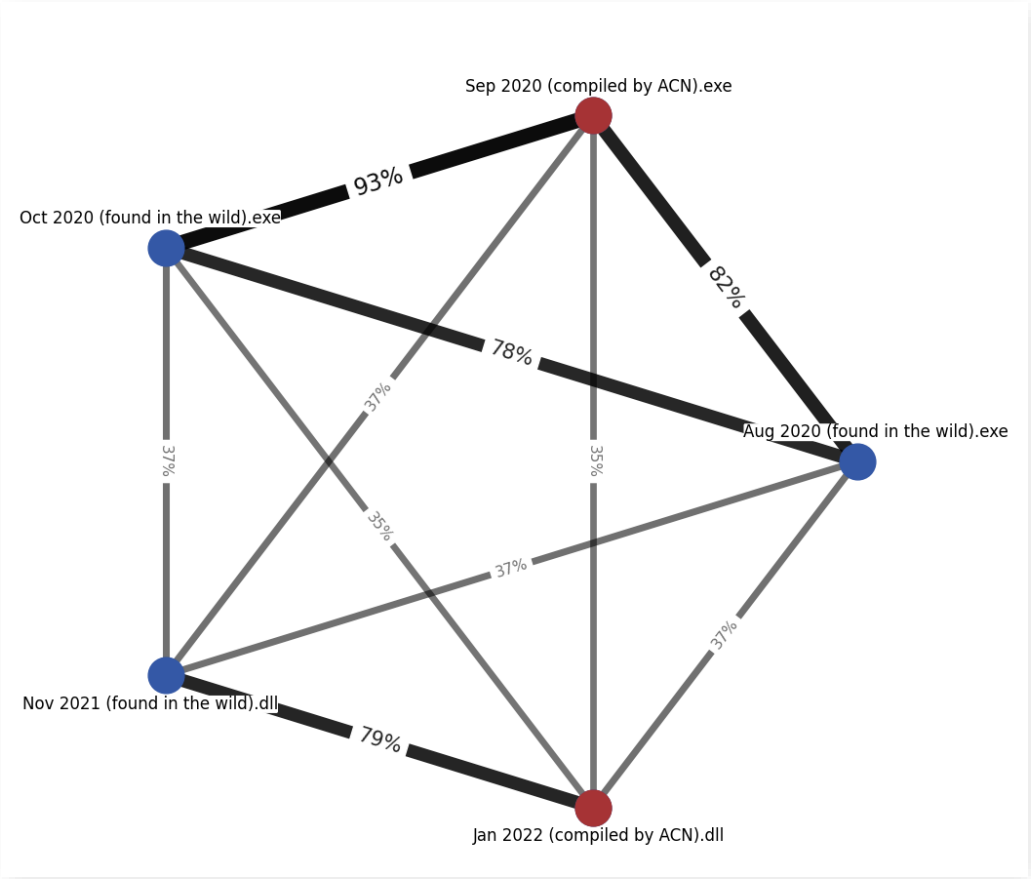


Figura 1 – Grafo delle similarità tra gli eseguibili compilati da ACN a partire dai sorgenti *leaked* (nodi rossi) e gli eseguibili compilati e usati in attacchi reali (nodi blu)

## 2.1 Argomenti dell'eseguibile

*Conti* fornisce al *threat actor* alcuni parametri opzionali da linea di comando che permettono di personalizzarne il comportamento. Di seguito si fornisce una descrizione di ciascun argomento.

| `-nomutex` |
| --- |
| Il comportamento standard di ***Conti*** prevede la creazione di una mutex con nome *hardcoded* nel sample. Quando questa flag viene impostata, *Conti* tralascia la creazione della mutex, consentendo a più istanze del ransomware di essere eseguite sullo stesso sistema. Questa opzione non è presente nei sorgenti di settembre 2020 in quanto implementata successivamente. |

| `-log <file>` |
| --- |

Viene impiegato il file `<file>` per registrare le operazioni che il ransomware compie. *Conti* non effettua alcuna attività di *logging* quando questo argomento viene omesso. Nei sorgenti di settembre 2020 l'opzione non permetteva di specificare il file **`<file>`** ma lo creava a prescindere nel percorso **`C:\CONTI_LOG.txt`**.

**`-size <encryption_mode_id>`**

Imposta la modalità di cifratura dei file di grandi dimensioni (superiori a 5 MB). Il parametro `<encryption_mode_id>` deve essere un numero compreso tra **10, 15, 20, 25, 30, 35, 40, 50, 60, 70 e 80,** ma **maggiori dettagli** verranno forniti nelle successive sezioni. Questa opzione non è presente nei sorgenti di settembre 2020 in quanto è stata implementata successivamente.

**`-m (local|net|all|backups)`**

**`local`**: cifra solamente le cartelle e i file locali

**`net`**: cifra solamente le unità di rete

**`all`**: modalità predefinita di *Conti* che corrisponde alla combinazione di *local* e *net*

**`backups`**: l'opzione è presente dal 2020 ma non è ancora stata effettivamente implementata, neppure nella versione ***v3*** del ransomware.

**`-p <directory>`**

Da usare in alternativa all'opzione **-m**, cifra solamente la cartella puntata da `<directory>`, usando un singolo thread. Tale modalità di funzionamento è stata probabilmente inserita dallo sviluppatore a scopo di *debugging*, tuttavia si rivela utile anche per l'analisi del ransomware. Nella versione di settembre 2020 l'argomento `<directory>` doveva in realtà puntare ad un file di testo contenente una lista di cartelle da cifrare.

**`-h <file>`**

Specifica un file `<file>` che contiene gli indirizzi IPv4 degli host da scansionare in rete. Questa opzione è stata rimossa già a partire da dicembre 2020.

Gli attuali argomenti da linea di comando possono essere riassunti nei seguenti *usage*:

| |
|---|
| `conti.exe [-nomutex] [-log <file>] [-size <mode_id>] [-m (all\|local\|net\|backups)]` |
| `conti.exe [-nomutex] [-log <file>] [-size <mode_id>] [-p <directory>]` |

## 2.2 Offuscamento API e stringhe

La maggior parte delle stringhe del ransomware è cifrata e la loro decifratura segue uno schema di tipo *lazy*, ossia viene effettuata solamente nel momento in cui le stringhe vengono utilizzate; ciò vale in tutte le versioni del ransomware prese in esame. In condizioni normali, ossia in assenza di codice sorgente, tale meccanismo complica l'analisi statica e rende necessario il *debugging* del ransomware.

Il metodo di *hardcoding* delle stringhe è fondato sulla definizione di due macro chiamate OBFA (per le stringhe *ANSI*) e OBFW (per le stringhe *Wide*) le quali, sfruttando tecniche di meta-programmazione, prendono in input una stringa e a *compile time* la restituiscono in forma cifrata. In questo modo lo sviluppatore astrae la procedura di offuscamento delle stringhe dal loro effettivo utilizzo, avvantaggiandosi del fatto di poterle utilizzare in chiaro nel codice con la consapevolezza che in maniera trasparente saranno inserite nell'eseguibile soltanto in forma cifrata. Ad un'analisi di *reverse engineering* infatti, le stringhe appaiono come cifrate nello *stack* e memorizzate carattere per carattere (*stack string*), rendendo vano l'utilizzo di strumenti quali `strings`. Ad esempio, la semplice linea di codice in *Figura 2* viene compilata da Visual Studio nel codice in *Figura 3*.

```
516     LPCSTR Netapi32DLL = OBFA("Netapi32.dll");
```

Figura 2 – Esempio di utilizzo della macro `OBFA` per cifrare a compile time la stringa `Netapi32.dll`

```
qmemcpy(v30, ".N", 2);
v30[2] = 5;
v30[3] = 89;
v30[4] = 16;
v30[5] = 67;
v30[6] = 25;
v30[7] = 123;
v30[8] = 7;
v30[9] = 49;
v30[10] = 27;
v30[11] = 27;
v30[12] = 70;
_ = (char *)v4;
for ( j = 0; j < 0xD; ++j )
  v30[j] = (35 * (70 - (unsigned __int8)v30[j]) % 127 + 127) % 127;// Netapi32.dll
```

Figura 3 – Codice decompilato della linea di codice mostrata in *Figura 2*

L'algoritmo di cifratura delle stringhe fissa due chiavi $k_1$ e $k_2$ e cifra in $C_i$ ogni byte $X_i$ della stringa secondo la seguente formula:

$$C_i = (k_1 X_i + k_2) \, mod \, 127$$

L'algoritmo di decifratura deve quindi risolvere la seguente equazione per poter riottenere i byte $X_i$ della stringa in chiaro e lo effettua (grazie all'applicazione dell'algoritmo esteso di Euclide, *Figura 4*) soltanto nel momento in cui la stringa viene referenziata:

$$k_1 X_i \equiv C_i - k_2 \; (mod \; 127)$$

Ogni stringa è quindi crittografata in modo differente in base alla scelta di $k_1$ e $k_2$, le quali vengono estratte pseudo-casualmente.

```
73    constexpr unsigned char __forceinline encrypt(unsigned char byte) const
74    {
75        return (A * byte + B) % 127;
76    }
77
78    constexpr unsigned char __forceinline decrypt(unsigned char byte) const
79    {
80        return positive_modulo(ExtendedEuclidian<127, A>::y * (byte - B), 127);
81    }
```

Figura 4 – Funzioni di cifratura e decifratura delle stringhe

Le funzioni di decifratura sono *inline*, ossia sono funzioni il cui codice viene direttamente inserito nella funzione chiamante. Questa, che di solito è una conseguenza delle ottimizzazioni automatiche del compilatore, sottolinea ancora una volta l'attenzione ai meccanismi di offuscamento da parte dello sviluppatore, il quale, facendo uso della parola chiave `__forceinline`, impone l'attivazione di questo meccanismo al fine di limitare la leggibilità del codice. Secondo l'indicazione fornita da alcuni commenti presenti nel codice, l'intero algoritmo di offuscamento delle stringhe è stato copiato e riadattato dal progetto open source *ADVobfuscator*[4].

Gli sforzi dello sviluppatore per affinare gli schemi di offuscamento si estendono anche alla risoluzione delle API di Windows. In questo caso, le funzioni di sistema vengono anch'esse risolte dinamicamente e con un approccio *lazy*, ma questa volta sfruttano un'unica funzione custom chiamata `GetProcAddressEx2` che restituisce il puntatore alla funzione richiesta:

[4] https://github.com/andrivet/ADVobfuscator

```
GetProcAddressEx2(<dll_id>, <function_name_hash>, <cache_index>)
```

Il primo parametro della funzione, `<dll_id>`, è un ID interno al ransomware che identifica la libreria DLL in cui è presente la funzione richiesta. La prima operazione effettuata all'avvio del ransomware prevede infatti il caricamento di tutte le librerie necessarie al suo funzionamento e l'assegnazione dell'identificativo. L'operazione di caricamento viene effettuata dinamicamente, ovvero tramite l'invocazione dell'API `LoadLibrary`, a sua volta risolta dinamicamente tramite l'enumerazione del *Process Environment Block (PEB)*. L'associazione tra l'ID `<dll_id>` e i nomi delle librerie DLL può variare in base alla versione di *Conti*, ma da poco più di un anno è staticamente determinata dalla seguente tabella:

| ID | DLL |
|---|---|
| 15 | Kernel32.dll |
| 16 | Advapi32.dll |
| 17 | Netapi32.dll |
| 18 | Iphlpapi.dll |
| 19 | Rstrtmgr.dll |
| 20 | User32.dll |
| 21 | Ws2_32.dll |
| 22 | Shlwapi.dll |
| 23 | Shell32.dll |
| 24 | Ole32.dll |
| 25 | OleAut32.dll |
| 26 | Ntdll.dll |

Tabella 1 – Associazione tra `<dll_id>` e librerie Microsoft

Il secondo parametro, `<function_name_hash>`, è l'hash (calcolato con l'algoritmo *Murmur Hash2A*) del nome della funzione richiesta, mentre il terzo, `<cache_index>`, è l'indice di una tabella *cache* in cui *Conti* memorizza i puntatori alle funzioni risolte fino a quel momento, in modo da non dover ripetere la lenta operazione di risoluzione più volte per le stesse API.

In *Figura 5* si mostra un esempio di utilizzo della funzione `GetProcAddressEx2` per risolvere `CreateThread`.

```
__forceinline HANDLE WINAPI pCreateThread(
    LPSECURITY_ATTRIBUTES   lpThreadAttributes,
    SIZE_T                  dwStackSize,
    LPTHREAD_START_ROUTINE  lpStartAddress,
    __drv_aliasesMem LPVOID lpParameter,
    DWORD                   dwCreationFlags,
    LPDWORD                 lpThreadId
)
{
    HANDLE(WINAPI * pFunction)(LPSECURITY_ATTRIBUTES, SIZE_T, LPTHREAD_START_ROUTINE, LPVOID, DWORD, LPDWORD);
    pFunction = (HANDLE(WINAPI*)(LPSECURITY_ATTRIBUTES, SIZE_T, LPTHREAD_START_ROUTINE, LPVOID, DWORD, LPDWORD))
        getapi::GetProcAddressEx2(NULL, KERNEL32_MODULE_ID, 0x8687ce53, 82);//GetProcAddress(hKernel32, OBFA("CreateThread"));
    return pFunction(lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, dwCreationFlags, lpThreadId);
}
```

Figura 5 – Esempio di chiamata alla funzione `GetProcAddressEx2` per risolvere l'API `CreateThread`

Nel codice sono inoltre presenti altri meccanismi di offuscamento quali condizioni e cicli ridondanti. Alcuni esempi sono visualizzabili in Figura 7, in particolare il ciclo `for` (righe 9-10), la condizione `if-then-else` (righe 17-21) e il ciclo `do while` (righe 23-25). Senza di essi la funzione si ridurrebbe a sole 8 righe (quelle evidenziate in rosso). Questa nuova funzionalità di offuscamento è stata inserita a partire da dicembre 2020. L'implementazione è basata sulla definizione di una funzione *custom* chiamata `morphcode`, che viene manualmente invocata dallo sviluppatore ogni qualvolta voglia inserire del codice ridondante. Secondo quanto riportato nei commenti in lingua russa dallo stesso sviluppatore, tale meccanismo viene descritto come "*uno strumento che rende il codice polimorfico (in modo che gli stessi sorgenti durante la compilazione producano diverse sequenze di istruzioni) al fine di complicare la creazione di una regola di rilevazione*". In *Figura 6* viene mostrato come il codice della funzione `GetProcAddressEx2` viene compilato nel codice mostrato in *Figura 7*; si noti che la funzione `GetProcAddressEx`, non corrisponde all'omonima API di Windows ma ad un'ulteriore funzione *custom* del ransomware.

```
LPVOID getapi::GetProcAddressEx2(__in LPSTR Dll, __in DWORD ModuleId, __in DWORD Hash, __in int CacheIndex)
{
    // Функция возвращает адрес функции используя кэш
    LPVOID Addr = NULL;

    Addr = g_ApiCache[CacheIndex];
    morphcode(Addr);

    if (!Addr) {

        // Функции нет в кэше. Получаем её адрес и добавляем в кэш
        Addr = GetProcAddressEx(Dll, ModuleId, Hash);

        morphcode(Addr);

        g_ApiCache[CacheIndex] = Addr;

    }
    return Addr;
}
```

Figura 6 – Codice della funzione `GetProcAddressEx2` nei sorgenti di gennaio 2022

```
1  char *__usercall ma_get_api@<eax>(int dll_id@<edx>, int function_name_hash, int cache_index)
2  {
3    char *api; // esi
4    int _; // [esp+20h] [ebp+Ch]
5    int __; // [esp+20h] [ebp+Ch]
6
7    cache_index *= 4;
8    api = (char *)api_cache_table[cache_index];
9    for ( _ = (int)(api + 0x2DA731); !(_ % 4); ++_ )
10     ;
11
12   if ( api )
13     return api;
14
15   api = ma_solve_api(0, dll_id, function_name_hash);
16   __ = (int)(api + 0x2DA731);
17   if ( (int)(api + 0x2DA731) % 4 )
18   {
19     *(int *)((char *)api_cache_table + cache_index) = (int)api;
20     return api;
21   }
22
23   do
24     ++__;
25   while ( !(__ % 4) );
26
27   *(int *)((char *)api_cache_table + cache_index) = (int)api;
28   return api;
29 }
```

Figura 7 – Codice decompilato della funzione `GetProcAddressEx2` mostrata in *Figura 6*. Le righe non evidenziate in rosso sono ridondanti e sono causate dall'invocazione di `morphcode`.

È infine presente un meccanismo di anti-analisi e anti-detection, atto ad ostacolare il *debugging* e la rilevazione del malware attraverso la rimozione di *hook* e *breakpoint* dalle librerie di sistema elencate in *Tabella 1*, eventualmente inseriti dall'analista o da sistemi di *endpoint security* di tipo *EDR*. La tecnica viene implementata sovrascrivendo a *runtime* il modulo caricato dal malware con il contenuto dei file dll presente sul disco. In questo modo viene ripristinato il contenuto del file originale e tutte le modifiche collaterali, causate dagli *hook* e dai *breakpoint* eventualmente inseriti, vengono rimosse.

## 2.3 Ransom notes, estensione file e portale Web

*Conti* memorizza le note del riscatto in un file di testo denominato `readme.txt`, in passato `R3ADM3.txt`, il cui contenuto è mostrato in *Figura 8*; il file viene collocato all'interno di ogni *directory* cifrata.

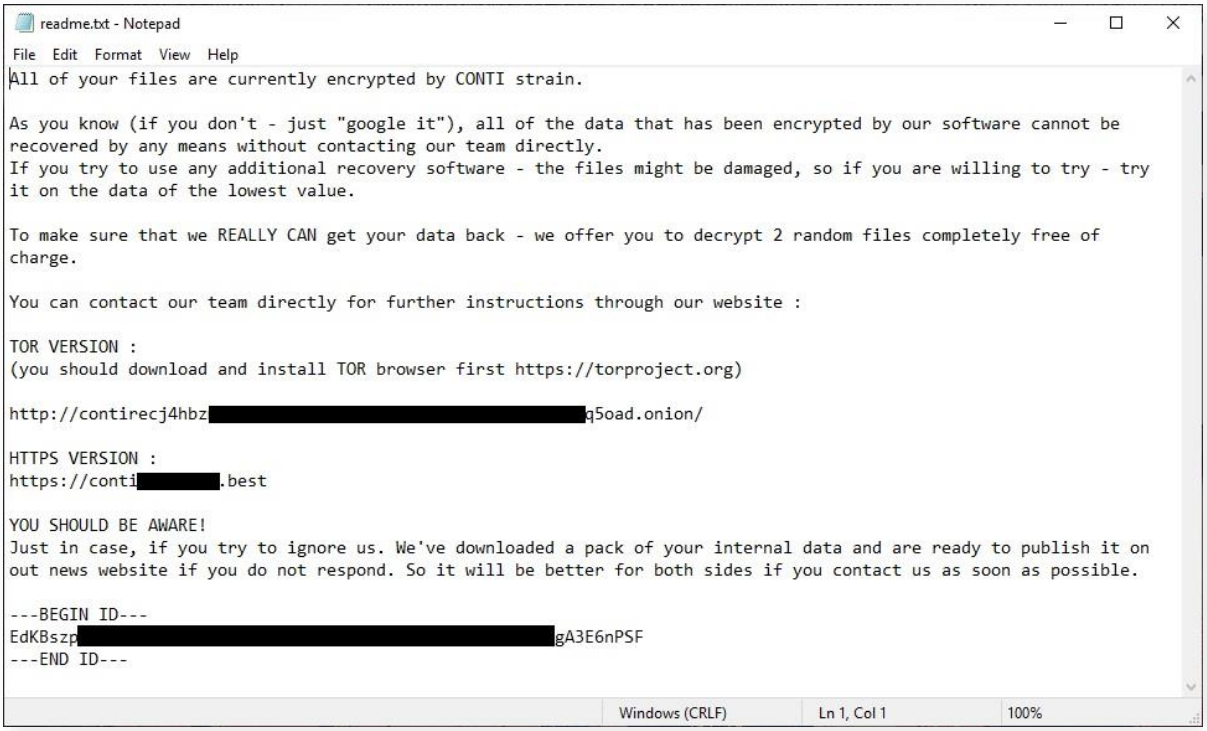


Figura 8 – *Ransom notes* di *Conti*

Il corpo del file, così come l'estensione dei file cifrati, la coppia di chiavi *RSA* dell'attaccante, l'ID della vittima e, soltanto nelle vecchie varianti, il nome della *mutex*, sono *hardcoded,* ma cambiano tra un sample e l'altro in quanto fanno parte dei parametri di output generati dal builder. In particolare lo sviluppatore predispone delle variabili inizializzate con un *placeholder* (*Figura 9*) e, dopo la compilazione del ransomware, invoca il *builder* (anch'esso rilasciato nel *data leak* ma soltanto come eseguibile compilato) per la generazione dei contenuti e il loro inserimento nell'eseguibile di *Conti*. Per ogni vittima, l'attaccante produce dunque un sample personalizzato.

```
STATIC WCHAR g_Extention[7] = L".EXTEN";
STATIC CHAR g_DecryptNote[2048] = "__DECRYPT_NOTE__";
STATIC INT g_EncryptMode = ALL_ENCRYPT;
STATIC BOOL g_IsProcKillerEnabled = FALSE;
STATIC LPCWSTR g_EncryptPath = NULL;
STATIC BYTE g_EncryptSize = 50;
//STATIC CHAR g_MutexName[65] = "__MUTEX_NAME__";
```

Figura 9 – Alcuni dei *placeholder* presenti nei sorgenti di Conti

Nelle note del riscatto non è presente nessun portafogli di criptovaluta, quest'ultimo viene concordato tra attaccante e vittima in una conversazione privata sul portale ufficiale del ransomware, presente sia in versione *.onion* (navigabile solamente attraverso TOR) che in versione web. Infatti, secondo il processo previsto, la vittima, dopo aver scoperto di essere stata infettata, deve effettuare l'upload del file `readme.txt` sul portale (*Figura 10*) e avviare una chat con l'operatore del ransomware (*Figura 11*).

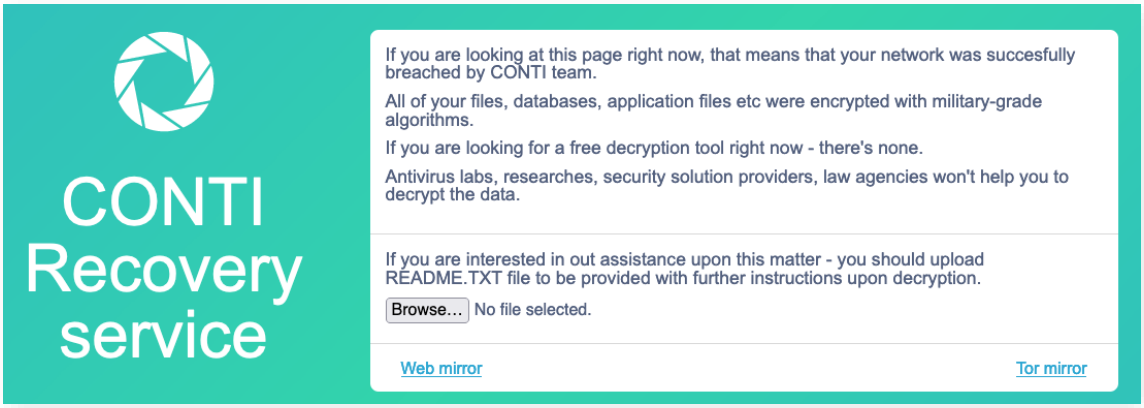


Figura 10 – *Homepage* del portale ufficiale di *Conti*

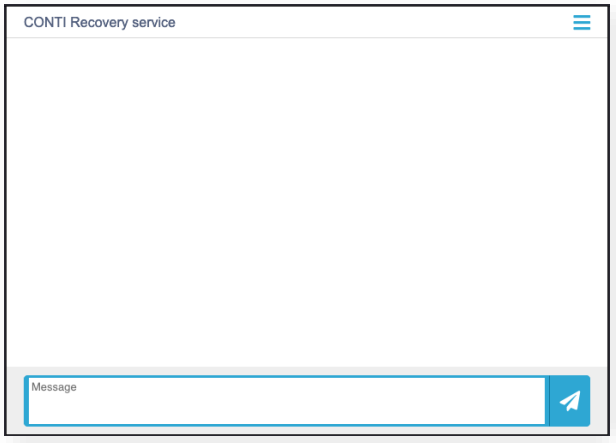


Figura 11 – Chat che appare sul portale ufficiale dopo l'upload del file *readme.txt*

Secondo quanto osservato, alcune evidenze suggeriscono che l'attaccante mantiene una blocklist di identificativi di vittime per i quali l'upload del file `readme.txt` viene rifiutato: si tratta probabilmente di ID associati a vittime che hanno già pagato il riscatto.

## 2.4 Rimozione copie shadow

La rimozione delle copie *shadow* permette ai ransomware di eliminare le copie di backup dei file della vittima presenti nel *File System* locale. *Conti*, così come la maggior parte dei ransomware, effettua questa operazione sfruttando *WMI (Windows Management Instrumentation)*, un'implementazione Microsoft del protocollo *WBEM* che offre ad applicazioni e script un'interfaccia per amministrare un sistema Windows locale o remoto. La modalità specifica con il quale le copie shadow vengono rimosse, descritta di seguito, presenta tuttavia delle peculiarità rispetto ad altri ransomware, che di solito sfruttano in forma diretta gli eseguibili Microsoft `vssadmin.exe` o `wmic.exe`.

Inizialmente il thread principale di *Conti* risolve ed esegue le chiamate di sistema `CoInitializeEx`, `CoInitializeSecurity` e `CoCreateInstance` per istanziare un oggetto dell'interfaccia `IWbemLocator`. Tale interfaccia serve ad ottenere un puntatore a un oggetto dell'interfaccia `IWbemServices`, a partire dal quale è possibile accedere ai servizi di WMI. Utilizzando l'oggetto `IWbemLocator`, *Conti* quindi invoca il metodo `IWbemLocator::ConnectServer` e ottiene un puntatore a un oggetto `IWbemServices`. Con

tale oggetto, al fine di estrarre informazioni sulle copie shadow presenti sul sistema, il ransomware esegue la seguente query *WQL (WMI Query Language)*:

```
SELECT * FROM Win32_ShadowCopy
```

e, enumerando queste informazioni, estrae l'ID di ogni copia. A partire da questo identificativo viene infine lanciato il seguente comando di rimozione della copia:

```
cmd.exe /c C:\Windows\System32\wbem\WMIC.exe shadowcopy where "ID='%s'" delete
```

## 2.5 Strategia multi-thread

Per la cifratura dell'intero sistema i ransomware ricorrono a diverse strategie, più o meno efficienti in base alla conoscenza da parte dello sviluppatore di tecniche di programmazione concorrente. L'obiettivo dello sviluppatore corrisponde in questo caso all'ideazione di un algoritmo *multi-thread* in grado di esplorare e cifrare il *file system* nel minor tempo possibile.

*Conti* gestisce il carico di lavoro necessario per portare a termine le due operazioni istanziando una *synchronized queue* condivisa (coda di cifratura), implementata come una lista doppiamente concatenata in cui gli inserimenti (*enqueue*) avvengono in coda, le rimozioni (*dequeue*) in testa e al cui interno vengono memorizzati i percorsi delle cartelle da cifrare. Per l'implementazione effettiva della coda lo sviluppatore importa staticamente la libreria open source *queue.h*[5].

Viene istanziato un numero di thread pari al doppio del numero di processori logici presenti sul sistema in uso, anche se in questa fase in alcuni sample di dicembre 2020 è stato osservato un bug in cui tale numero veniva estratto dal sistema a partire dal campo `dwActiveProcessorMask`, il quale restituisce una maschera di bit che rappresenta l'insieme di processori configurati nel sistema (il bit i-esimo si riferisce all'esistenza nel sistema del processore i-esimo), rispetto al campo corretto `dwNumberOfProcessors`.

I nuovi thread rimangono in *loop* sulla lista in attesa di nuove cartelle, accendovi in maniera sincronizzata attraverso le `CRITICAL_SECTION` di Windows e le relative chiamate `EnterCriticalSection` e `LeaveCriticalSection`. Quando una nuova cartella viene trovata, il thread la estrae dalla lista e in autonomia effettua sia l'esplorazione delle sotto-cartelle, eseguita con il tradizionale approccio ricorsivo (`FindFirstFileW` e `FindNextFileW`), sia la cifratura di tutti i file discendenti.

[5] https://opensource.apple.com/source/xnu/xnu-124.8/bsd/sys/queue.h.auto.html

Il thread principale, in base alla modalità con il quale viene avviato (argomento `-m`) e solo dopo aver avviato tutti gli altri thread, inserisce nella coda condivisa le cartelle root dei drive presenti nell'host (`-m local` oppure `-m all`) e/o le unità di rete (`-m net` oppure `-m all`) e rimane in attesa che tutti i thread istanziati concludano il proprio compito.

Secondo quanto descritto, il carico di lavoro per ogni thread varia in base alle dimensioni dell'unità che sta cifrando, quindi il tempo di cifratura dell'intero sistema è equivalente al tempo necessario a un thread per crittografare l'unità più grande. Tale strategia si rivela inefficiente e rende quasi inconcludente l'utilizzo del *multi-threading*, in quanto nella maggior parte dei sistemi il numero di drive o di unità di rete è basso e i file sono principalmente localizzati nel `drive C`.

## 2.6 Crittografia

*Conti* utilizza una tradizionale combinazione tra crittografia simmetrica (*Chacha8*) e asimmetrica (*RSA*) per cifrare i file ma, a differenza di altri ransomware, usa le API di sistema `wincrypt` per alcune primitive crittografiche.

Il ransomware inizialmente usa la chiamata `CryptAcquireContext` per inizializzare il *Cryptographic Service Provider (CSP)* con il tipo `PROV_RSA_AES`, mentre estrae la chiave RSA pubblica dell'attaccante con la funzione `CryptImportKey`. La chiave ha una lunghezza di 4096 bit ed è memorizzata in un blob di tipo `PUBLICKEYBLOB`[6]. L'attaccante, invocando il *builder*, genera una coppia di chiavi RSA diversa per ogni eseguibile, in modo che la strategia di estorsione non sia vulnerabile a un attacco di tipo *pay once, decrypt-many*.

Attraverso la chiamata `CryptGenRandom` (`CryptGenKey` per le varianti del 2020), il ransomware genera pseudo-casualmente e in modo crittograficamente sicuro una chiave di 32 byte e una nonce di 8 byte e, dopo averli usati per cifrare il file vittima con l'algoritmo *Chacha8*, utilizza l'API `CryptEncrypt` per cifrarli con la chiave pubblica dell'attaccante; il blob di dati risultante viene aggiunto al termine del file (*Figura 12*). Soltanto l'attaccante, essendo in possesso della chiave *RSA* privata, può decifrare il blob e riottenere la chiave e la nonce, pertanto l'eventuale software di *decryption* fornito alla vittima in seguito al pagamento del riscatto dovrà contenere la chiave privata dell'attaccante. Effettivamente, il builder, nel momento in cui genera la coppia di chiavi RSA dell'attaccante, effettua il *patching* dei due eseguibili `locker.exe` (o

[6] https://docs.microsoft.com/en-us/windows/win32/seccrypto/rsa-schannel-key-blobs#public-key-blobs

`locker.dll`) e `decryptor.exe` inserendone rispettivamente la chiave pubblica e la chiave privata. Per l'implementazione di *Chacha8* lo sviluppatore ha importato staticamente la libreria open source fornita dall'ideatore dell'algoritmo[7].

Ogni file del sistema viene crittografato con una chiave e una nonce diversa, dunque le ultime operazioni descritte vengono ripetute per ciascun file.

*Conti*, per bilanciare efficienza ed efficacia dell'attacco, adotta una strategia di cifratura che varia con le dimensioni e con il formato del file da cifrare.

I file piccoli, ossia di dimensione inferiore a 1 MiB, oppure tutti i file che hanno un'estensione associabile a un formato di *storage* tra quelli indicati nella *Tabella 2*, vengono cifrati interamente.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4dd | 4dl | abcddb | abs | abx | accdb | accdc | accde |
| accdr | accdt | accdw | accft | adb | ade | adf | adn |
| adp | alf | arc | ask | bdf | btr | cat | cdb |
| ckp | cma | cpd | dacpac | dad | dadiagrams | daschema | db |
| db-shm | db-wal | db2 | db3 | dbc | dbf | dbs | dbt |
| dbv | dbx | dcb | dct | dcx | ddl | dlis | dp1 |
| dqy | dsk | dsn | dtsx | dxl | eco | ecx | edb |
| epim | exb | fcd | fdb | fic | fm5 | fmp | fmp12 |
| fmpsl | fol | fp3 | fp4 | fp5 | fp7 | fpt | frm |
| gdb | grdb | gwi | hdb | his | hjt | ib | icg |
| icr | idb | ihx | itdb | itw | jet | jtx | kdb |
| kexi | kexic | kexis | lgc | lut | lwx | maf | maq |
| mar | mas | mav | maw | mdb | mdf | mdn | mdt |
| mpd | mrg | mud | mwb | myd | ndf | nnt | nrmlib |
| ns2 | ns3 | ns4 | nsf | nv | nv2 | nwdb | nyf |
| odb | oqy | ora | orx | owc | p96 | p97 | pan |
| pdb | pdm | pnz | qry | qvd | rbf | rctd | rod |
| rodx | rpd | rsd | sas7bdat | sbf | scx | sdb | sdc |
| sdf | sis | spq | sql | sqlite | sqlite3 | sqlitedb | te |
| temx | tmd | tps | trc | trm | udb | udl | usr |
| v12 | vis | vpd | vvv | wdb | wmdb | wrk | xdb |
| xld | xmlff | | | | | | |

Tabella 2 - Formati di file cifrati interamente.

Ai file medi, ossia compresi tra 1 e 5 MiB, viene cifrato soltanto il primo MiB. I file grandi invece, ossia superiori a 5 MiB, oppure tutti i file associabili a macchine virtuali o immagini di memoria, ossia i seguenti:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| vdi | vhd | vmdk | pvm | vmem | vmsn | vmsd | nvram | vmx | raw |
| qcow2 | subvol | bin | vsv | avhd | vmrs | vhdx | avdx | vmcx | iso |

[7] https://cr.yp.to/chacha.html

vengono divisi in *chunk* e soltanto alcuni tra essi vengono cifrati. Il valore di default della dimensione, del numero e dell'offset dei chunk può cambiare per specifico sample ma può anche essere forzato con l'argomento `-size <encryption_mode_id>`, in cui il parametro `<encryption_mode_id>` si riferisce a un identificativo delle modalità di cifratura supportate, ossia un numero compreso tra 10, 15, 20, 25, 30, 35, 40, 50, 60, 70 e 80 che corrisponde all'incirca alla percentuale del file da cifrare. Ad esempio la modalità con identificativo 20, ossia quella solitamente predefinita, imposta con la seguente formula la dimensione dei chunk:

$$ChunkSize = \frac{7}{100} FileSize = 7\%$$

imposta il seguente offset:

$$ChunkOffset = \frac{1}{2}(FileSize - 3 * ChunkSize) = \frac{79}{200} FileSize$$

e ne cifra soltanto tre, per un totale del 21%, ossia quelli ai seguenti offset:

$$\{0,\ \ ChunkOffset,\ \ 2 * ChunkOffset\}$$

In ciascuno dei casi descritti, per sovrascrivere il contenuto in chiaro dei file con la rispettiva controparte cifrata vengono usate le chiamate `ReadFile`, `SetFilePointerEx` e `WriteFile`.

Prima della cifratura, come accennato, vengono aggiunte al file alcune info utili al decryptor per poter decifrare il file. Tali informazioni possono essere riassunte dalla struttura in *Figura 12*.

```
struct ENCRYPTION_METADATAS {
  BYTE encrypted_key_and_nonce[524];    // Contiene chiave e nonce Chacha8
  BYTE file_type_id;                    // Indica il formato del file
  BYTE encryption_mode_id;              // L'id della modalità utilizzata
  LARGE_INTEGER file_size;              // La dimensione originale del file
}
```

Figura 12 – Struttura che descrive le informazioni che servono al *decryptor* per decifrare i file vittima e che vengono aggiunte al termine dei file cifrati

## 2.7 Blocklist e gestione dei file occupati

Rispetto alla maggior parte dei ransomware, *Conti* non effettua la terminazione di processi e servizi che potrebbero intralciarne l'esecuzione o limitarne l'efficacia, quali

software antivirus o di backup, piuttosto utilizza il *Restart Manager* di Windows per terminare i processi che tengono i file occupati, una pratica molto semplice usata anche da altri ransomware come *Babuk*, *REvil* e *RegretLocker*. I casi di errore che comportano l'invocazione di questa funzionalità sono i seguenti:

- `ERROR_SHARING_VIOLATION`: il ransomware non può accedere al file perché è in uso da un altro processo;
- `ERROR_LOCK_VIOLATION`: il ransomware non può accedere al file perché un altro processo ne ha bloccato una parte.

La tecnica si basa sulla sequenza delle chiamate `RmStartSession`, `RmRegisterResources` e `RmGetList`, le quali, a partire da un determinato file, restituiscono la lista dei processi occupanti. Una volta ottenuta la lista, il ransomware verifica che non sia presente sé stesso o `explorer.exe` e termina forzatamente i processi con la chiamata `RmShutdown`.

## 2.8 Whitelist

Prima di cifrare un file o una cartella, *Conti* si assicura che essi non appartengano a quelli indicati nella seguente tabella:

| File | Cartelle |
|---|---|
| .exe | Tmp |
| .dll | winnt |
| .lnk | temp |
| .sys | thumb |
| .msi | $Recycle.Bin |
| readme.txt | $RECYCLE.BIN |
| CONTI_LOG.txt | System Volume Information |
| .bat | Boot |
| .<estensione_ransomware> | Windows |
| | Trend Micro |
| | perflogs |

Tabella 3 - Esclusione di file e cartelle dal processo di cifratura.

L'obiettivo principale è quello di non compromettere l'usabilità del sistema per consentire alla vittima di contattare l'attaccante direttamente dallo stesso e dar modo di eseguire l'eventuale decryptor.

## 2.9 Scansione e cifratura della rete

La gestione della cifratura delle risorse di rete è una caratteristica unica di *Conti*, in quanto il ransomware usa un algoritmo di *SMB discovery* custom per individuare i server con cartelle condivise raggiungibili, anziché usare il metodo predefinito che prevede l'impiego delle chiamate `WNetOpenEnum` e `WnetEnumResourceA`. La comprensione dell'algoritmo di discovery non mostra particolari difficoltà, in quanto, in sintesi, esegue un *port scan* orizzontale su porta 445 verso le sottoreti /24 presenti nella tabella ARP del sistema e inserisce nella coda di cifratura le risorse dei server che rispondono entro 30 secondi. L'implementazione e conseguentemente l'analisi invece risulta molto complessa, sia perché fa uso di parecchie strutture e liste custom (che in assenza di sorgenti vanno ricostruite prima di poterle analizzare), sia perché basa il proprio funzionamento sul meccanismo delle *I/O Completion Port*, un threading model di Windows che, tramite un pool di thread automaticamente istanziato e coordinato, gestisce efficientemente le richieste di input/output asincrone in un'applicazione. Di seguito si fornisce una descrizione dettagliata dei passi compiuti da *Conti* per implementare l'algoritmo e cifrare le cartelle di rete.

*Conti* inizialmente istanzia alcune strutture di supporto, costituite da una *I/O completion port* e tre *synchronized queue* vuote che tengono traccia delle seguenti informazioni:

- **`SubnetsList`**: lista di sottoreti /24 che contengono indirizzi potenzialmente raggiungibili (ma la cui raggiungibilità è da verificare);
- **`SocketsList`**: lista di socket aperte sul sistema verso gli indirizzi IP delle subnet della lista **`SubnetsList`**;
- **`NetResourcesList`**: lista di server interni (con nome DNS e indirizzo IP) per il quale la raggiungibilità è stata verificata;

Successivamente scansiona la tabella ARP del sistema con la chiamata `GetIpNetTable` e salva nella lista **`SubnetsList`** tutte le sottoreti /24 ottenute dagli IP presenti in tabella che iniziano con "`172.`", "`192.168.`", "`10.`" e "`169.`". L'intento del ransomware è quello di tenere traccia di tutte le sottoreti private contattate di recente dal sistema, dunque con buona probabilità raggiungibili.

Vengono poi lanciati due thread `PRODUCER` e `CONSUMER`:

- il thread `CONSUMER` si occupa semplicemente della gestione sincronizzata di ogni server che viene aggiunto nella lista **`NetResourcesList`**. Essendo tali server

sicuramente raggiungibili, il thread `A` li enumera con la funzione `NetShareEnum` e aggiunge nella coda di cifratura tutti i percorsi di rete trovati;
- il thread `PRODUCER` effettua un port scan orizzontale su porta 445 verso tutti gli IP presenti all'interno delle subnet /24 contenute nella lista **`SubnetsList`** e inserisce quelli raggiungibili nella lista **`NetResourcesList`**.

```
g_IocpHandle = pCreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, NULL, 0);
if (g_IocpHandle == NULL) { ... }

TAILQ_INIT(&g_SubnetList);
TAILQ_INIT(&g_HostList);
TAILQ_INIT(&g_ConnectionList);

if (!GetSubnets(&g_SubnetList)) { ... }

hHostHandler = pCreateThread(NULL, 0, &HostHandler, NULL, 0, NULL);
if (hHostHandler == INVALID_HANDLE_VALUE) { ... }

hPortScan = pCreateThread(NULL, 0, &PortScanHandler, NULL, 0, NULL);
if (hPortScan == INVALID_HANDLE_VALUE) { ... }

pPostQueuedCompletionStatus(g_IocpHandle, 0, START_COMPLETION_KEY, NULL);
pWaitForSingleObject(hPortScan, INFINITE);

AddHost(STOP_MARKER);
pWaitForSingleObject(hHostHandler, INFINITE);
```

Figura 13 – Creazione della prima *I/O completion port*, delle tre liste di supporto e dei thread `PRODUCER` (chiamato `PortScanHandler` nel codice) e `CONSUMER` (`HostHandler`)

In particolare, il thread `PRODUCER` è costituito da un *message loop*, ossia, attraverso la chiamata sincrona `GetQueuedCompletionStatus`, rimane in attesa di messaggi inviati con la chiamata `PostQueuedCompletionStatus` da un altro thread. Tale messaggio può essere di tre tipi:

- **`START_COMPLETION_KEY`**: viene inviato manualmente dal thread principale per indicare che il setup delle strutture di supporto e dei thread `PRODUCER` e `CONSUMER` è terminato. Il `PRODUCER` interpreta questo messaggio come un comando per avviare la gestione delle sottoreti contenute nella lista **`SubnetsList`**. L'operazione viene effettuata con i seguenti passi:
  - rimozione del nodo in testa alla lista **`SubnetsList`**. Se la lista è vuota allora il thread `PRODUCER` termina perché vuol dire che ha gestito tutte le sottoreti da verificare;
  - apertura di una socket per ciascuno dei 254 indirizzi IP che contiene la sottorete estratta (`a.b.c.`**`1`**`-a.b.c.`**`254`**);
  - connessione delle 254 socket al *pool* di thread automatico della *completion port*;

  - aggiunta delle 254 socket alla lista **`SocketsList`**;
  - gestione della lista **`SocketsList`**, in particolare a tutte le socket viene comandato di eseguire un contatto sulla porta 445. Il comando viene lanciato con la chiamata `ConnectEx`, pertanto saranno i thread della completion port ad occuparsene in maniera trasparente;
  - avvio di un timer di 30 secondi con la chiamata `CreateTimerQueueTimer`. Il timer si limita ad inviare un messaggio di **`TIMER_COMPLETION_KEY`** nel message loop.
- **`CONNECT_COMPLETION_KEY`**: viene inviato in automatico dal pool di thread che gestisce le socket, soltanto quando un server ha risposto alla chiamata sulla porta 445. Il `PRODUCER` interpreta questo messaggio come un comando per rimuovere la socket dalla lista **`SocketsList`** e per inserire il relativo server (adesso verificato) nella lista **`NetResourcesList`**. Se dopo la rimozione la lista è vuota allora significa che l'intera sottorete è stata processata, pertanto il `PRODUCER` avvia la gestione della sottorete successiva in **`SubnetLists`**.
- **`TIMER_COMPLETION_KEY`**: viene inviato dal timer di 30 secondi istanziato prima. In questo caso il `PRODUCER` capisce che sono passati 30 secondi e annulla con la chiamata `CancelIo` tutte le socket che ancora non hanno ricevuto risposta dal rispettivo server. Poi avvia la gestione della sottorete successiva in **`SubnetLists`**. Tale messaggio serve ad impostare un *timeout* al *port scan*.

In *Figura 14* viene mostrato uno schema grafico che descrive le strutture di supporto utilizzate nell'algoritmo di *network discovery*.

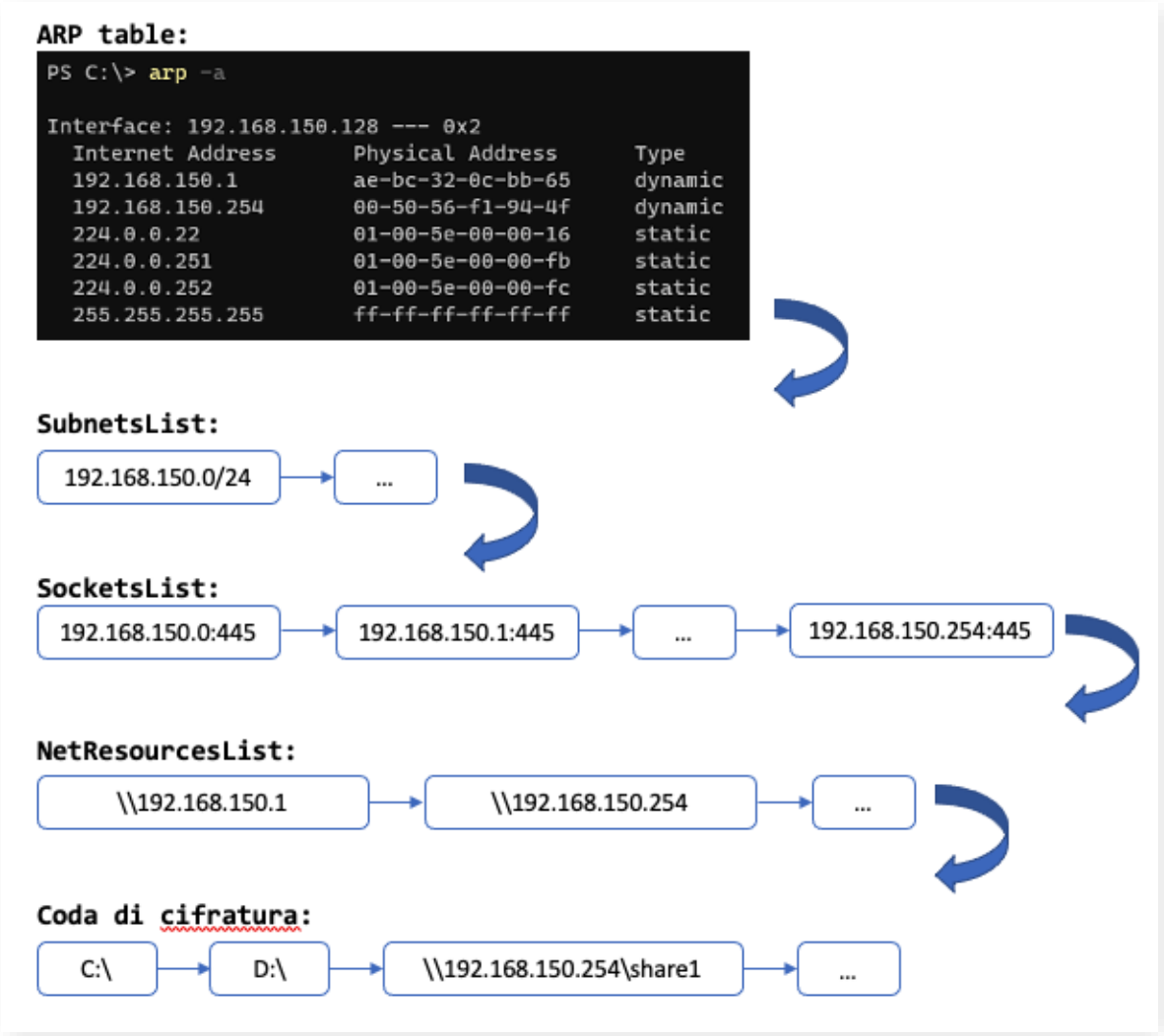


Figura 14 – I passi compiuti dall'algoritmo di *network discovery*: dalla tabella ARP alla coda di cifratura

# 3 Contromisure e raccomandazioni

Benché sia difficile contrastare l'acquisizione di informazioni potenzialmente utili al fine di perpetrare attacchi informatici (*reconnaissance*) e la preparazione di artefatti malevoli (*weaponization*) da parte degli innumerevoli *activity-group* esistenti, è opportuno minimizzare le informazioni pubbliche da cui l'avversario possa trarre vantaggio, come alcune informazioni non necessarie rilasciate pubblicamente. È stato verificato che le tecniche per ottenere il cosiddetto *foothold* all'interno delle infrastrutture target contemplano l'invio di *malspam* (*delivery*) o l'impiego di *exploit* per lo sfruttamento di vulnerabilità (*exploitation*) sui servizi Internet esposti, come VPN, *load-balancer* o applicazioni Web. È dunque necessario un addestramento specifico del personale preposto alla ricezione, apertura e lettura di mail oltre ad un'ordinaria valutazione delle vulnerabilità dell'infrastruttura utilizzata. Sarebbe utile adottare una politica più stringente circa la ricezione di determinate tipologie di file ed evitare sempre e comunque l'apertura di link direttamente ricevuti nelle mail. In ogni caso, è sempre consigliato impiegare un sistema di monitoraggio e di rilevamento di eventuali eventi di sicurezza che possano indicare il tentativo o l'avvenuta compromissione delle reti e dei sistemi in uso. Ai fini dell'analisi di eventuali incidenti, inoltre, è auspicabile l'impiego di un sistema di *logging* dei citati eventi di sicurezza, sia a livello host che network.

Di seguito si elencano alcune raccomandazioni di base che è sempre consigliabile implementare:

- effettuare regolari Backup dei dati critici, preferibilmente conservati in supporti non connessi in modo permanente alla rete o ai sistemi;
- impiegare su tutti i sistemi soluzioni di Endpoint Detection & Response (EDR) che contengano almeno la componente anti-malware avendo cura di mantenerlo aggiornato;
- abilitare un firewall sui sistemi garantendo esclusivamente il traffico verso i servizi e sistemi necessari;
- disattivare i servizi non necessari, sia nelle postazioni utente che sui server;
- utilizzare preferibilmente l'autenticazione multi fattoriale per gli accessi in VPN e, ove possibile, per l'accesso ai servizi esposti su Internet;
- mantenere aggiornati i software e i sistemi ed in particolare quelli impiegati per i servizi di accesso remoto;

- configurare in modo sicuro i servizi di connessione remota come quelli basati su RDP impostando limiti di accesso e password complesse e ove possibile, sistemi di autenticazione multifattoriale;
- non aprire senza opportune verifiche allegati o collegamenti in e-mail;
- verificare le comunicazioni tramite email security gateway;
- prevedere per il personale periodiche sessioni di formazione finalizzate a riconoscere il phishing e le minacce associate alla posta elettronica;
- garantire il giusto grado di consapevolezza tra i dipendenti;
- verificare la presenza di vulnerabilità che impattano prodotti e applicazioni di accesso remoto rivolti al pubblico, con particolare riferimento alle recenti vulnerabilità del protocollo RDP (CVE-2020-0609, CVE-2020-0610, CVE-2020-6896, CVE-2019-1489, CVE-2019-1225, CVE-2019-1224, CVE-2019-1108);
- valutare la capacità di rilevare e bloccare l'uso di Cobalt Strike sulla rete;
- analizzare e ridurre quanto possibile la superficie di attacco di Active Directory secondo le best practices di riferimento di Microsoft;
- limitare quanto possibile il numero e l'uso di account privilegiati, adottando il principio del privilegio minimo per tutti i task di amministrazione (just-in-time/just-enough);
- monitorare gli eventi di Active Directory per rilevare eventuali indicatori di intrusione e compromissione;
- introdurre restrizioni sull'impiego di tool di amministrazione come BitsAdmin, WMIC e PowerShell sulla rete;
- rilevare l'impiego improprio di tool di amministrazione come BitsAdmin, WMIC e PowerShell sulla rete;
- implementare regole di base per il controllo degli script;
- rilevare l'uso di tool di esecuzione remota come Psexec;
- rilevare eventuali movimenti laterali non previsti;
- segmentare la rete;
- crittografare i documenti sensibili sulla rete per impedirne la possibile divulgazione;

- impiegare soluzioni di Data Loss/Leak Prevention (DLP);
- implementare un piano di risposta agli attacchi informatici;
- impedire l'esecuzione di macro nei prodotti MS Office, consentendone l'esecuzione solo agli utenti che ne hanno comprovata necessità e, ove possibile, esclusivamente per le macro firmate digitalmente;
- visitare le pagine MITRE ATT&CK riportate nella sezione successiva per valutare ulteriori e personalizzate strategie finalizzate alla mitigazione e al rilevamento.

Lo sfruttamento di vulnerabilità o di configurazioni di sicurezza non ottimali che interessano le tecnologie e servizi esposti sulla rete perimetrale è tra i punti di accesso privilegiati nelle incursioni ransomware. Le attività di difesa devono pertanto prevedere cicli di verifica, ad esempio tramite *vulnerability assessment* (VA), al fine di garantire la tempestiva risoluzione delle stesse.

Un'ulteriore superficie di attacco, sfruttata in recenti compromissioni, è rappresentata da domini, accessi e sistemi di interoperabilità con partner e società terze: è quindi necessario porre particolare attenzione ad attività non previste e/o che si discostino dal normale utilizzo.

# 4 Regola Yara

```
rule Conti {
    meta:
        author = "Agenzia per la Cybersicurezza Nazionale (ACN)"
        date = "April 2022"
        malware = "Conti"

    strings:
        $ransom_notes_content_1 = "encrypted" nocase ascii wide
        $ransom_notes_content_2 = "data" nocase ascii wide
        $ransom_notes_content_3 = "TOR" ascii wide
        $ransom_notes_content_4 = "download" nocase ascii wide
        $ransom_notes_content_5 = "contact" nocase ascii wide
        $ransom_notes_content_6 = "CONTI strain" nocase ascii wide
        $ransom_notes_content_7 = "---BEGIN ID---" ascii wide
        $ransom_notes_content_8 = "---END ID---" ascii wide
        $ransom_notes_content_9 = ".onion" ascii wide
        $ransom_notes_content_10 = "https://torproject.org" ascii wide
        $ransom_notes_placeholder = "__DECRYPT_NOTE__" ascii wide

        $chacha_1 = "expand 32-byte k" ascii wide
        $chacha_2 = "expand 16-byte k" ascii wide

        $rsa_key_blob = {52 53 41 (31 | 32) 00 (04 | 08 | 10 | 20) 00 00}
        $rsa_placeholder = "__publickey__" ascii wide

        $string_decryption_x86 = {
            8A 07
            8D 7F ??
            0F B6 C0
            B9 ?? 00 00 00
            2B C8
            6B C1 ??
            99
            F7 FE
            8D 42 ??
            99
            F7 FE
            88 57 ??
        }

        $string_decryption_x64 = {
            41 0F B6 11
            4D 8D 49 ??
            83 EA ??
            B8 09 04 02 81
            6B CA ??
            F7 E9
            03 D1
            C1 FA 06
            8B C2
            C1 E8 1F
            03 D0
            6B C2 7F
            2B C8
            B8 09 04 02 81
            83 C1 7F
            F7 E9
            03 D1
```

```
            C1 FA 06
            8B C2
            C1 E8 1F
            03 D0
            6B C2 7F
            2B C8
            41 88 49 FF
            49 83 EA 01
        }

    condition:
        (5 of ($ransom_notes_content_*) or $ransom_notes_placeholder)
        and all of ($chacha_*)
        and ($rsa_key_blob or $rsa_placeholder)
        and any of ($string_decryption_*)
        and filesize > 90KB and filesize < 400KB
}
```