sect.iij.ad.jp /blog/2025/06/ransomware-nailaolocker/

ランサムウェアNailaoLockerの調査

Naoki Takayama ⋮ ⋮ 6/20/2025

2025年2月、複数のセキュリティベンダーがNailaoLockerというランサムウェアに関するレポートを公開しました[1][2]。

このランサムウェアによる攻撃の特異な点として、PlugXやShadowPadといった特定の国家を背景とする攻撃者が使用するマルウェア(RAT[3])が感染までの攻撃活動で用いられていたことが挙げられます。PlugXやShadowPadは諜報活動を目的とした攻撃で使用されてきたRATであり、今回のように、侵害した組織内で最終的にランサムウェアを拡散・実行する事例はこれまで殆ど報告されていませんでした。

この記事では既存のレポートで取り上げられていない内容を中心に、NailaoLockerを解析することで得られた知見について共有します。

検体の情報

今回解析したNailaoLockerのSHA256ハッシュ値は以下の通りです。

SHA256	説明
2b069dcde43b874441f66d8888dcf6c24b451d648c8c265dffb81c7dffafd667	暗号化されたNailaoLocker
ee5fc8c8b3a9f7412655da01eb27a8959c82732988c0de593c1c90afeda1ef55	NailaoLocker

NailaoLockerは独自の方式によって暗号化された状態でファイルシステム上に存在し、NailaoLoader (ローダー)によってメモリ上で復号された上で実行されます。暗号化されたNailaoLockerは、以下のPythonスクリプトを用いることで静的に復号することができます。この記事ではNailaoLoaderについて詳しく取り上げませんが、興味がある方は併せてご確認ください。

```
import ctypes

with open('**暗号化されたNailaoLocker**', 'rb') as dat_file:
    dat = bytearray(dat_file.read())

for i in range(len(dat)):
    tmp = ctypes.c_uint8(dat[i] + 75).value
    tmp ^= 0x3F
    tmp = ctypes.c_uint8(tmp - 75).value
    dat[i] = tmp

with open('**出力先**', 'wb') as result_file:
    result_file.write(dat)
```

NailaoLockerの処理の流れ

NailaoLockerは以下の流れで感染端末上のファイルを暗号化します。後述しますが、Mutex名はNailaoLockerのバージョンによって異なるのでご注意ください。

1. NailaoLoaderのアンロード・削除
2. Mutex `Global\lockv7`の作成
3. エンコードされたデータ領域のデコード
4. ログファイルの作成
5. ファイルの暗号化・脅迫文の作成

エンコードされたデータ領域

脅迫文のファイル名、内容、各種設定値など、NailaoLocker内の一部のデータはシングルバイトXOR (0x3F)でエンコードされています。今回解析した検体では、実行バイナリのオフセット0x26D700から0x27132Cまでのデータ領

域がエンコードされていました。

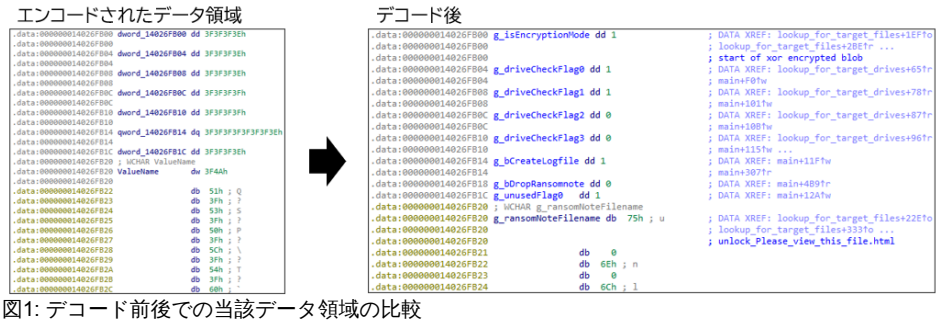

図1: デコード前後での当該データ領域の比較

これらのデータが存在する領域はNailaoLocker実行時にデコードされ、必要に応じてアクセスされます。

```
if ( (unsigned __int64)(int)v7 < 0x3C2C )
{
  v9 = (char *)&isEncryptionMode + (int)v7;
  do
  {
    *v9++ ^= 0x3Fu;
    ++v7;
  }
  while ( v7 < 0x3C2C );
}
```
図2: 当該データ領域をデコードする処理

エンコードされたデータ領域の構造については、記事末尾のAppendix Aをご確認ください。

ログファイル

NailaoLockerは`%ProgramData%\lock.log`にログファイルを作成し、暗号化等が行われた際にログを書き込むように設計されています。今回解析した検体では、以下のようなログが出力されます。

- ファイル暗号化時
  - 成功時: `OK Encrypt [**対象ファイル名**]`
  - 失敗時: `*** Encrypt [**対象ファイル名**] error with code **エラーコード**`
- ファイル復号時 (詳細は後述)
  - 成功時: `OK Decrypt [**対象ファイル名**]`
  - 失敗時: `*** Decrypt [**対象ファイル名**] error with code **エラーコード**`

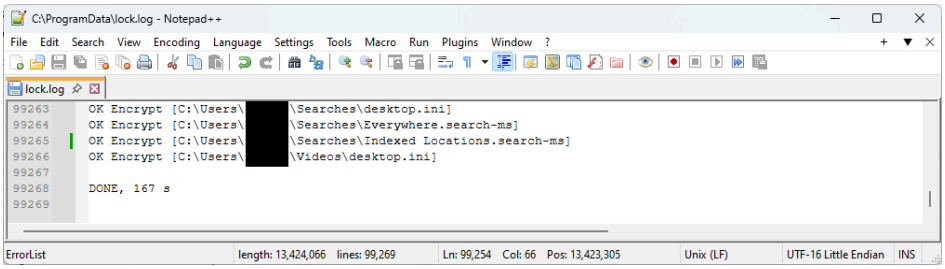

図3: NailaoLockerが出力するログファイルの例

また、今回の検体では有効化されていない機能でしたが、NailaoLocker実行時に任意のプロセスを実行する機能も実装されていました。

実行に成功した場合、失敗した場合に以下のログが出力されます。

- 任意プロセス実行時[4]
  - 成功時: `OK Excute [**プロセス名**] OK`
  - 失敗時: `*** Excute [**プロセス名**] error with code **エラーコード**`

ログファイルが作成されるか否かは、先述したエンコードされたデータ領域内の設定値に依存します。どのような処理が為されたか確認する上で有用なファイルなので、NailaoLockerへの感染が疑われる場合は確認すると良いでしょう。

除外対象

以下のドライブ・フォルダ・ファイルはNailaoLockerによる暗号化の対象から除外されます。

種別	除外対象
ドライブ	`A:\` , `B:\` [5] , `W:\`
フォルダ	`Boot` , `Windows` , `Program Files` , `Program Files (x86)` , `ProgramData` , `AppData` , `Application Data`
ファイル	`boot.ini` , `*.exe` , `*.dll` , `*.sys`

ファイル暗号化

NailaoLockerは静的リンクされたOpenSSL 3.3.2のライブラリ関数をファイルの暗号化に利用します。具体的な暗号化の流れは以下の通りです。

1. 32バイトのランダムなバイト列(AES鍵)、16バイトのランダムなバイト列(IV)を生成する

```
21  RAND_bytes_ex((__int64)cstruct.random32Bytes, 32);
22  RAND_bytes_ex((__int64)cstruct.random16Bytes, 16);
```

図4: AES鍵およびIVを生成

2. ハードコードされたRSA公開鍵を用いてAES鍵とIVを暗号化する

```
39    if ( (unsigned int)EVP_PKEY_encrypt(
40                          cstruct.evp_pkey_ctx1,
41                          (unsigned int)cstruct.encryptedRandom32Bytes,
42                          (unsigned int)&encryptedRandom32BytesLength,
43                          (unsigned int)cstruct.random32Bytes,
44                          32) != 1 )
45      goto LABEL_21;
46    cstruct.encryptedRandom32BytesLength = encryptedRandom32BytesLength;
47    encryptedRandom16BytesLength = (unsigned int)cstruct.encryptedRandom16BytesLength;
48    if ( (unsigned int)EVP_PKEY_encrypt(
49                          cstruct.evp_pkey_ctx1,
50                          (unsigned int)&cstruct.encryptedRandom16Bytes,
51                          (unsigned int)&encryptedRandom16BytesLength,
52                          (unsigned int)cstruct.random16Bytes,
53                          16) != 1 )
```

図5: AES鍵およびIVをRSAで暗号化

3. AES-256-CBCによる暗号化の準備を行う

図6: AES-256-CBCで暗号化を実施するための初期化処理

4. ファイルの先頭0x10000 (65,536)バイトを読み込む
5. 4をファイルサイズおよびAES-256-CBCで暗号化したデータで上書きする

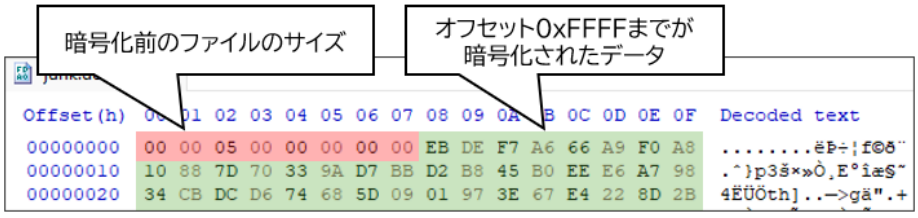

図7: 暗号化されたファイルの先頭8バイト

6. ファイルの末尾に暗号化されたAES鍵、IVなど様々な情報を含むフッターを書き込む

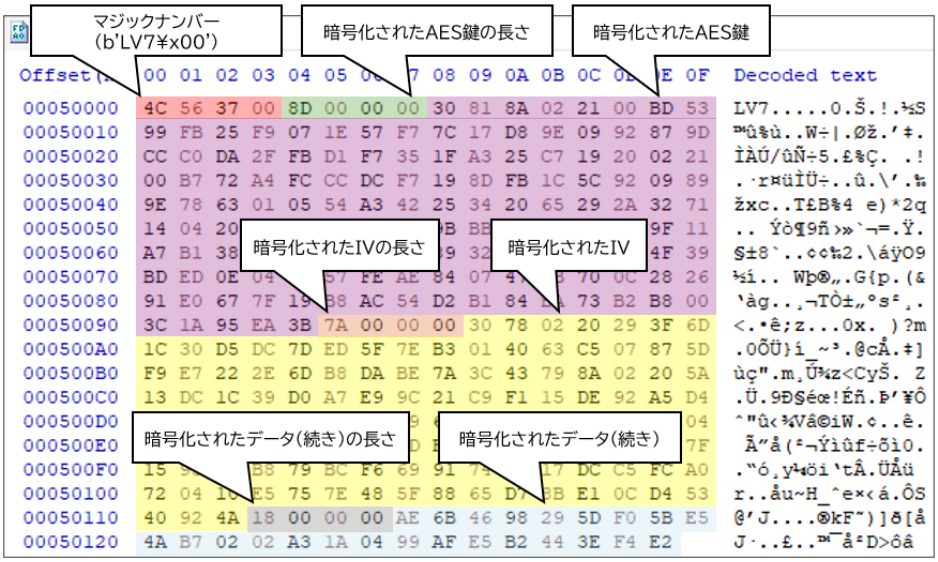

図8: 暗号化されたファイルの末尾

7. ファイル名の末尾に.lockedを付加する
8. SetFileTime関数を用いてファイルの$SIタイムスタンプを元ファイルと同じものに改ざんする
9. 上記の手順を暗号化対象のファイルの数だけ繰り返す

一部のみを暗号化

ファイルサイズが0x10000 (65,536)バイトを超える大きさのファイルが暗号化された場合、0x10000バイト以降のデータは平文のまま保持されます。これは他のランサムウェアでも頻繁に用いられる手法で、サイズが大きいファイルの暗号化に要する時間を削減するため、ファイルの一部のみ(多くの場合ヘッダーを含む先頭部分)を暗号化しているのだと考えられます。

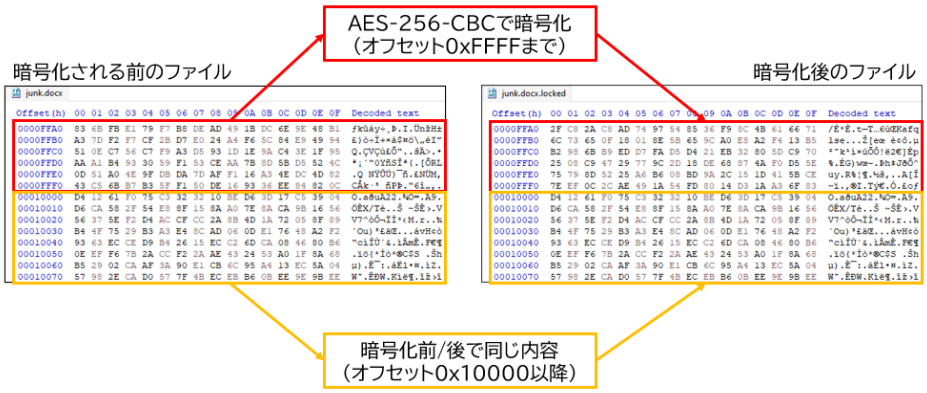

図9: オフセット0x10000以降は平文状態で暗号化後も残っている

脅迫文

NailaoLockerは暗号化対象のファイルが存在するフォルダ内に脅迫文を作成します。脅迫文のHTMLファイルのコメント内にKodex Ransomwareと関係する文字列が含まれていることから、この脅迫文はKodex Ransomwareのものをそのまま流用している可能性が高いです。

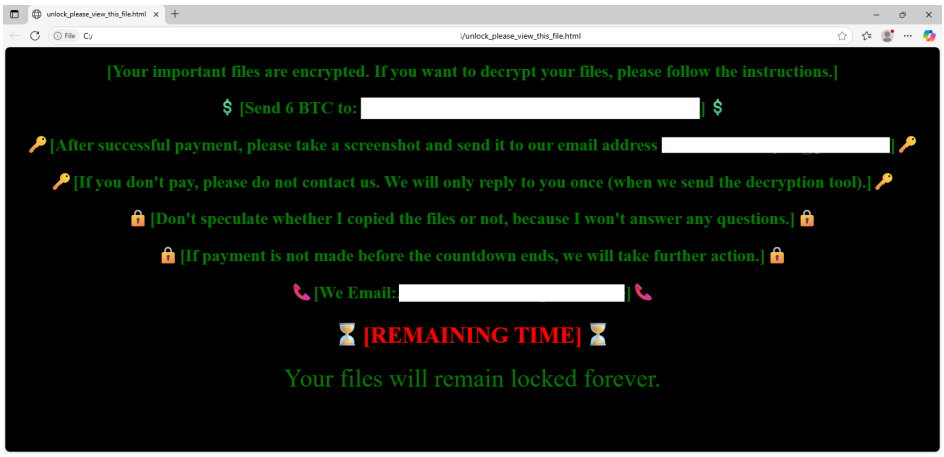

図10: NailaoLockerの脅迫文

また、今回の検体では有効化されていない機能でしたが、%ALLUSERSPROFILE%下に脅迫文を作成し、それをレジストリキー{HKLM/HKCU}\SOFTWARE\Microsoft\Windows\CurrentVersion\Runに設定することで自動実行させるような処理も確認できました。

ファイル復号機能の存在

今回解析した検体では、NailaoLockerが設定値によって暗号化されたファイルの復号ツールとしても動作するような実装を確認できました。具体的には、本来ファイルの暗号化時に用いるRSA公開鍵が格納されている領域からRSA秘密鍵を読み込み、復号用の関数を呼び出すことでファイルを復号することが可能となっていました。

暗号化モード時の初期化処理（RSA公開鍵を読み込む）

```
26  if ( (_BYTE)isEncryptionMode )                // isEncryptionMode == TRUE then encrypt the file
27  {
28    pp = &g_PublicKey;
29    publicKey = d2i_PUBKEY_0(0, (int)&pp, g_PublicKeyLength);
30    cstruct.publicKey = publicKey;
```

復号モード時の初期化処理（RSA秘密鍵を読み込む）

```
57  }
58  else                                          // isEncryptionMode == FALSE then decrypt the file
59  {
60    v12 = &g_PublicKey;
61    pkey = d2i_AutoPrivateKey_0(0, (__int64 *)&v12, g_PublicKeyLength);
62    cstruct.privateKey = pkey;
```

RSA公開鍵とRSA秘密鍵で参照しているアドレスが同一

```
.data:0000000140270F20 g_PublicKeyLength dd 5Bh                ; DATA XREF: CryptoThreadMain+75↑r
.data:0000000140270F20                                         ; main+18E↑w ...
.data:0000000140270F24 g_PublicKey       xmmword 2A080601023DCE48862A070613305930h
.data:0000000140270F24                                         ; DATA XREF: CryptoThreadMain+6E↑o
.data:0000000140270F24                                         ; main+16B↑w ...
```

図11: RSAの公開鍵と秘密鍵を同じアドレスから読み込む

このことから、身代金を支払ったユーザーにはRSA秘密鍵を含む復号モードのNailaoLockerを提供する(あるいは攻撃者が実行する)ようなオペレーションを想定していたと考えられます。

なお、脅迫文に含まれるBitcoinアドレスを調査しても一切の取引の痕跡が確認できなかったことから、攻撃者に身代金を支払ったユーザーはいない可能性が高いです。

APIフックによる検知妨害

NailaoLockerは、実行直後にGetModuleHandleExW関数に対してAPIフックを行うように実装されています。GetModuleHandleExWの先頭アドレス、あるいは既にJMP命令によってAPIフックされている場合、ジャンプした先のアドレスから12バイトの命令を書き換えて、図13の関数にジャンプするようにしています。これはEDR製品等による検知を困難にして、確実に暗号化処理を実施するためだと考えられます。

```
11  ptrHookingFunc = GetModuleHandleExW;
12  if ( !GetModuleHandleExW )
13    return GetLastError();
14  if ( *(_BYTE *)GetModuleHandleExW == 0xEB )   // EB ---> jmp
15    ptrHookingFunc = (BOOL (__stdcall *)(DWORD, LPCWSTR, HMODULE *))((char *)GetModuleHandleExW
16                                                                     + *((char *)GetModuleHandleExW + 1)
17                                                                     + 2);
18  *(_WORD *)patchedInstructions = 0xB848;       // 48 B8 ---> mov rax, sub_140002C40
19  *(_QWORD *)&patchedInstructions[2] = sub_140002C40;
20  *(_WORD *)&patchedInstructions[10] = 0xE0FF;  // FF E0 ---> jmp rax
21  flOldProtect[0] = 64;
22  VirtualProtect(ptrHookingFunc, 0xCu, PAGE_EXECUTE_READWRITE, flOldProtect);
23  v4 = *(_DWORD *)&patchedInstructions[8];
24  *(_QWORD *)ptrHookingFunc = *(_QWORD *)patchedInstructions;
25  *((_DWORD *)ptrHookingFunc + 2) = v4;
26  VirtualProtect(ptrHookingFunc, 0xCu, flOldProtect[0], flOldProtect);
27  CurrentProcess = GetCurrentProcess();
28  FlushInstructionCache(CurrentProcess, ptrHookingFunc, 0xCu);
```

図12: 命令を書き換えている処理

```
1 // Hidden C++ exception states: #wind=1
2 __int64 __fastcall sub_140002C40(int dwFlags, __int64 lpModuleName, HMODULE *phModule)
3 {
4   if ( dwFlags != 5 )
5     return 0;
6   *phModule = GetModuleHandleA(0);
7   return 1;
8 }
```

図13: ジャンプ先の関数のデコンパイル結果

v3とv7

今回解析したNailaoLockerは実行時に`Global\lockv7`というMutexを作成しますが、末尾部分のv7はバージョンであると考えられます[6]。それを裏付けるように、他の暗号化されたNailaoLockerの中に`Global\lockedv3`というMutexを作成するものが確認できました[7]。よって、そのような検体がNailaoLocker v3、今回解析した検体はNailaoLocker v7であったといえるでしょう。

```
59  if ( !CreateMutexW(0, 0, L"Global\\lockedv3") || GetLastError() == ERROR_ALREADY_EXISTS )
60    ExitProcess(0);
```

図14: NailaoLocker v3におけるMutexの作成処理

NailaoLocker v3とNailaoLocker v7のコンパイルタイムスタンプを比較してみると、およそ2か月程度の差しかないことがわかりました。ここに実際のコンパイル日時が格納されているとすると、開発者は積極的にNailaoLockerをアップデートして、機能を強化していると考えられます。

名称	SHA256	コンパイルタイムスタンプ
NailaoLocker v3	c7e5cd90da79d40818d6e781c9204e95d1b2a0973b0413282e82d2c125776e9b	2024-08-18 08:43:20 (UTC)
NailaoLocker v7	ee5fc8c8b3a9f7412655da01eb27a8959c82732988c0de593c1c90afeda1ef55	2024-10-19 08:02:47 (UTC)

NailaoLocker v3とNailaoLocker v7の最大の違いは暗号化時に使用するライブラリ関数です。NailaoLocker v7は静的リンクされたOpenSSLの関数を使用しているのに対して、NailaoLocker v3はCNG APIを使用して暗号化を実施しています。

CNGではなくOpenSSLを使用するよう変更を加えた理由として、暗号化関連APIの呼び出しによってアンチウイルス製品に検知・妨害されないようにしたこと、マルウェア解析者の負担を増やそうとしたこと等が考えられます。

```
15  a1->BCryptGenRandom = (__int64)BCryptGenRandom;
16  if ( !BCryptGenRandom )
17    return GetLastError();
18  BCryptOpenAlgorithmProvider = (NTSTATUS (__stdcall *)(BCRYPT_ALG_HANDLE *, LPCWSTR, LPCWSTR, ULONG))GetProcAddress((HMODULE)a1->field_40, "BCryptOpenAlgorithmProvider");
19  a1->BCryptOpenAlgorithmProvider = (__int64)BCryptOpenAlgorithmProvider;
20  if ( !BCryptOpenAlgorithmProvider )
21    return GetLastError();
22  BCryptImportKeyPair = (NTSTATUS (__stdcall *)(BCRYPT_ALG_HANDLE, BCRYPT_KEY_HANDLE, LPCWSTR, BCRYPT_KEY_HANDLE *, PUCHAR, ULONG, ULONG))GetProcAddress((HMODULE)a1->field
23  a1->BCryptImportKeyPair = (__int64)BCryptImportKeyPair;
24  if ( !BCryptImportKeyPair )
25    return GetLastError();
26  BCryptGenerateSymmetricKey = (NTSTATUS (__stdcall *)(BCRYPT_ALG_HANDLE, BCRYPT_KEY_HANDLE *, PUCHAR, ULONG, PUCHAR, ULONG, ULONG))GetProcAddress((HMODULE)a1->field_40, "
27  a1->BCryptGenerateSymmetricKey = (__int64)BCryptGenerateSymmetricKey;
28  if ( !BCryptGenerateSymmetricKey )
29    return GetLastError();
```

図15: CNG API関数のアドレスを動的に解決している処理

NailaoLocker v3にのみ存在する検知回避手法として、動作に必要な正規ライブラリ(bcrypt.dll)を任意の場所(%temp%\locked.dll)にコピーして、それをロードするというものがあります[8]。

```
99      ExpandEnvironmentStringsW(L"%windir%\\system32\\bcrypt.dll", Dst, 0x104u);
100     ExpandEnvironmentStringsW(L"%temp%\\locked.dll", Filename, 0x104u);
101     if ( CopyFileW(Dst, Filename, 0) && (v16 = LoadLibraryW(Filename), (v8->field_40 = (__int64)v16) != 0) )
102       LastError = resolve_cng_api_functions(v8);
103     else
104       LastError = GetLastError();
```

図16: bcrypt.dllをコピーしてロードする

NailaoLocker v7に本手法が実装されていなかったのは、OpenSSLを利用することでbcrypt.dllをロードする必要がなくなったからだと考えられます。

これ以外にも、ファイルの暗号化をマルチスレッドで行うようになったり、ログファイルに出力する情報が増えたりなど、NailaoLocker v7にはNailaoLocker v3から進化している点が随所に見られました。

おわりに

現時点でNailaoLockerはランサムウェアとして発展途上な存在だといえます。ファイルを暗号化するスピードはLockbitなど他のランサムウェアと比較して遅く、暗号化中に感染端末がシャットダウンした場合のリカバリー機能がないなど、機能面でも昨今のモダンなランサムウェアに劣っています。

しかし、継続的な開発が行われている可能性があるため、今後さらに検知が困難かつ高機能なランサムウェアとなっていくのかもしれません。少なくとも公開情報の範囲では、日本の企業や組織を標的とした攻撃で観測されていませんが、今後も警戒が必要な対象だと言えるでしょう。

暗号化されたNailaoLockerを検知するYARAルールをAppendix Bに記載していますのでご確認ください。

Appendix A: エンコードされたデータ領域の構造

オフセット	説明	解析した検体での設定値
0x0000	実行モード(0 = 暗号化モード, 1 = 復号モード)	0
0x0004	Aドライブの暗号化設定(0 = 暗号化する , 1 = 暗号化しない)	1
0x0008	Bドライブの暗号化設定(0 = 暗号化する , 1 = 暗号化しない)	1
0x000C	Cドライブの暗号化設定(0 = 暗号化する , 1 = 暗号化しない)	0
0x0010	Dドライブの暗号化設定(0 = 暗号化する , 1 = 暗号化しない)	0
0x0014	ログファイルの設定(0 = 作成しない , 1 = 作成する)	1
0x0018	脅迫文の自動実行設定(0 = 無効 , 1 = 有効)	0
0x001C	不明(使用されていない)	1
0x0020	脅迫文のファイル名	省略
0x0220	実行時に同時実行する任意プロセス名	空
0x1420	RSA公開鍵あるいは秘密鍵のサイズ	0x5B
0x1424	RSA公開鍵あるいは秘密鍵(最大0x70バイト)	省略
0x1494	不明(使用されていない)	0
0x1824	脅迫文のサイズ	0x98A
0x1828	脅迫文の内容	省略

Appendix B: YARAルール

```
rule ENC_NailaoLocker_Ransomware {
    meta:
      description = "Detects encrypted NailaoLocker ransomware payload"
      author = "Internet Initiative Japan Inc."
      hash1 = "a2e937d0b9d5afa5b638cd511807e0fcb44ec81b354e2cf0c406f19e5564e54e"
      hash2 = "2b069dcde43b874441f66d8888dcf6c24b451d648c8c265dffb81c7dffafd667"
      hash3 = "27b313243daf145c9105f5372e01f1cea74c62697195c1a21c660be5f7ee788c"
```

```
    strings:
        $rule1 = { 52 29 6F 29 4D 29 29 }
        $rule2 = { 04 29 59 29 37 29 3A 29 42 29 37 29 48 29 3C 29 65 29 48 29 35 29
48 29 04 29 4D 29 3D 29 3A 29 46 29 3E 29 FB 29 3D 29 3A 29 42 29 29 }

    condition:
        uint16(0) == 0x4F5C and
        all of ($rule*)
}
```

1. https://www.orangecyberdefense.com/global/blog/cert-news/meet-nailaolocker-a-ransomware-distributed-in-europe-by-shadowpad-and-plugx-backdoors ↩
2. https://www.trendmicro.com/en_us/research/25/b/updated-shadowpad-malware-leads-to-ransomware-deployment.html ↩
3. 正称はRemote Access TrojanやRemote Administration Toolなど。攻撃者による端末の遠隔操作を可能とするマルウェア。 ↩
4. `Execute`ではなく`Excute`となっていますが、これは攻撃者のタイプミスと思われます。 ↩
5. エンコードされたデータ領域内の設定値によります。今回の検体ではAドライブ・Bドライブ内のファイルは暗号化されないように設定がされていました。 ↩
6. 暗号化されたファイルのフッターのマジックナンバー`b'LV7\x00'`も解析対象がNailaoLockerのバージョン7だという説を補強しています。 ↩
7. SHA256: 27b313243daf145c9105f5372e01f1cea74c62697195c1a21c660be5f7ee788c ↩
8. Winnti Loaderにも似た検知回避手法が実装されています。詳細についてはRevivalStone：Winnti Groupによる日本組織を狙った攻撃キャンペーン | LAC WATCHをご確認ください。 ↩

カテゴリー:マルウェア解析

タグ:MalwareTargeted AttackRansomware