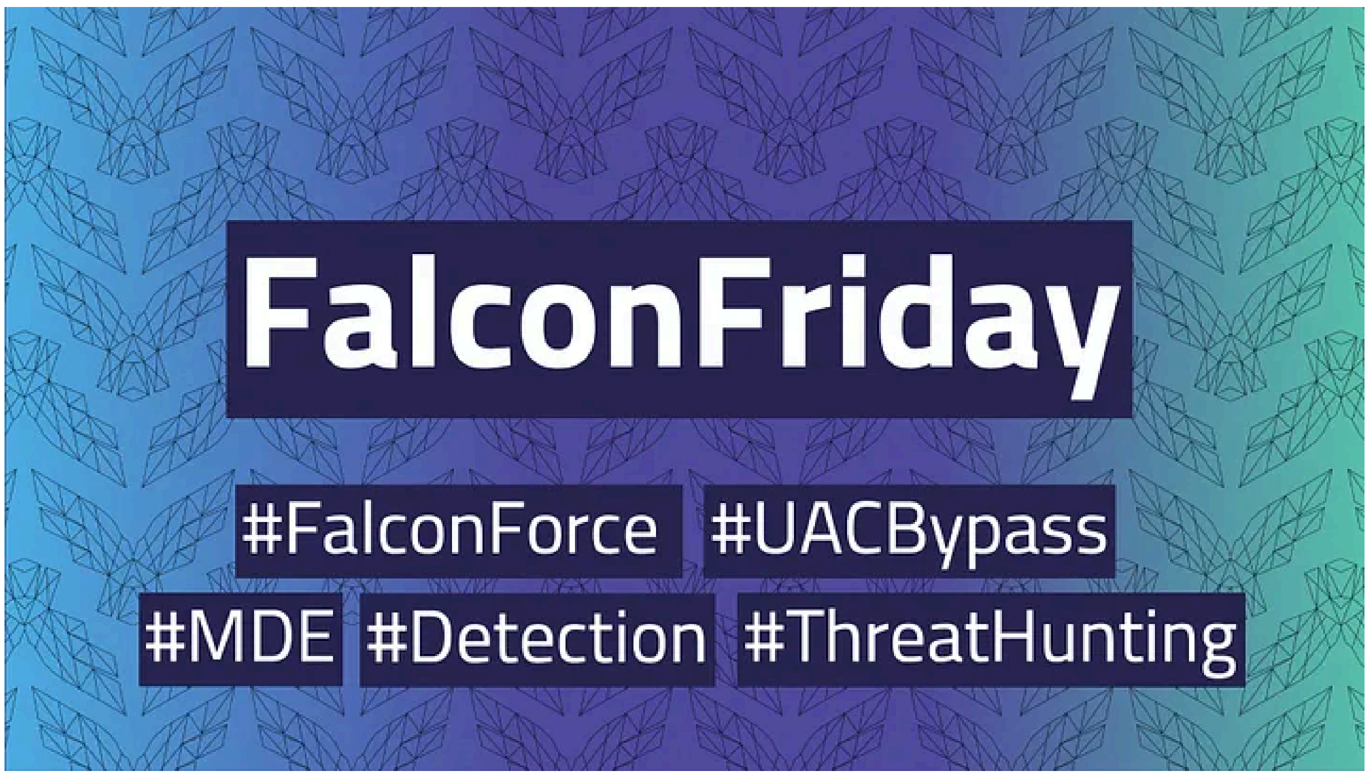
FalconFriday — Detecting UAC Bypasses — 0xFF16

Gijs Hollestelle · Follow
Published in FalconForce · 5 min read · Aug 20, 2021

Attackers often require full administrative privileges on a machine to be able to use their full attack capabilities. Many attacks originate from a regular user account running with low or medium integrity. Therefore one of the first things an attacker needs to do is bypass User Account Control (UAC) to get access to a process running with high integrity. If the targeted user has administrative privileges this can be achieved by using social engineering techniques to have them approve a UAC prompt or by using a UAC bypass technique that bypasses the UAC prompt altogether. In this blog post we take a look at a collection of UAC bypasses published in the UACME Github repository and investigate how they can be detected using Microsoft Defender for Endpoint (MDE).

TL;DR for blue teams: Many publicly documented UAC bypasses exist that work against a fully patched Windows 10 machine. This blog post provides detection rules for several of these UAC bypasses that will allow detection of techniques that are not detected by default using MDE.

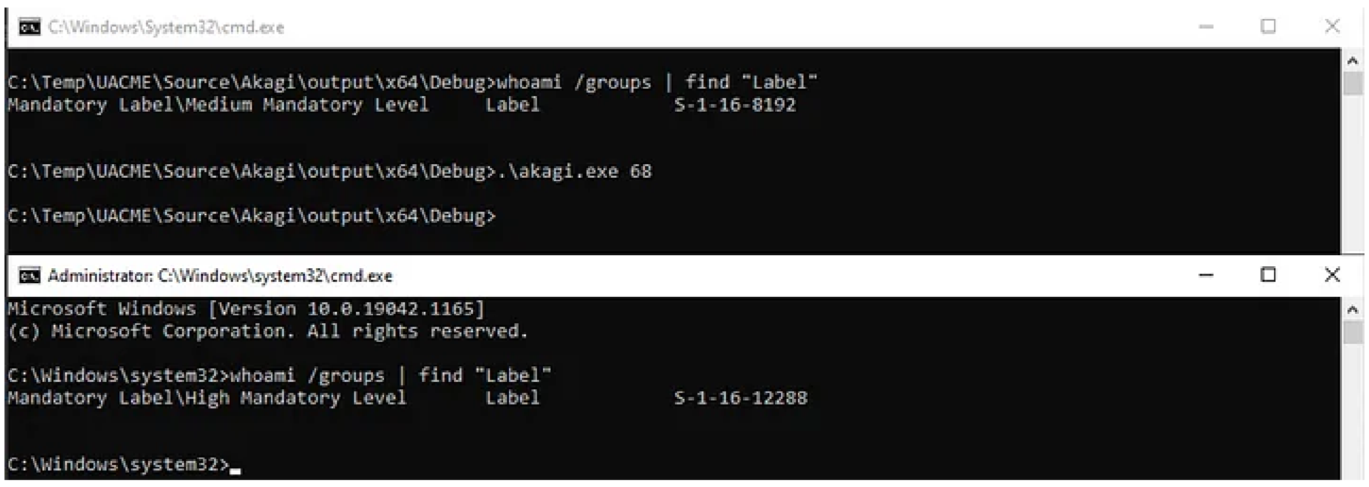
TL;DR for red teams: While many functional UAC bypass techniques are available, many of them allow for relatively easy detection. It might be beneficial to research your own techniques instead of relying on widely known techniques that can be easily detected.



. . .

Many UAC bypass methods have been identified and published in the past. An excellent source for such bypasses is the UACME Github repository maintained by hFireF0X. At the time of writing this blog post: August 2021, the repository contains 69 different UAC bypasses.

The techniques in UACME are simply numbered 1 to 69 and can be invoked by running the "akagi" tool available in the Github repository and providing the technique number as the first command line argument. An additional argument providing the binary to launch can be provided but in our tests we omitted it, causing the default (which is "cmd.exe") to be launched.


C:\Windows\System32\cmd.exe

```
C:\Temp\UACME\Source\Akagi\output\x64\Debug>whoami /groups | find "Label"
Mandatory Label\Medium Mandatory Level     Label            S-1-16-8192

C:\Temp\UACME\Source\Akagi\output\x64\Debug>.\akagi.exe 68

C:\Temp\UACME\Source\Akagi\output\x64\Debug>
```

Administrator: C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami /groups | find "Label"
Mandatory Label\High Mandatory Level       Label            S-1-16-12288

C:\Windows\system32>_
```

Example of using the UACME technique 68 to run a cmd.exe session with High integrity, bypassing UAC

The first step is identifying which of these techniques are functional on a fully patched Windows 10 machine. To determine that we can simply run all the techniques one-by-one and verify which of the techniques provide us with a high integrity cmd.exe when launched.

This results in a list of 12 techniques (techniques 33, 34, 41, 43, 53, 56, 59, 61, 62, 65, 67, and 68) that worked on our default Windows 10 installation. Note that it is possible that additional techniques could also be made to work using additional tuning but we simply tried the techniques that worked out of the box since we expect a majority of attackers will use similar easy to use techniques.

We also investigated whether or not the techniques are detected by Microsoft Defender for Endpoint (MDE) out of the box. This yields 6 techniques that are detected using the 'UAC bypass was detected' alert (techniques 33, 34, 56, 59, 62, and 67), and one technique that is detected using the 'Behavior:Win32/UACBypassExp.F!sdclt' alert (technique 53).

The other 5 functional techniques are not detected by MDE out of the box and require custom detection rules to be detected. Below we go through these techniques, investigate how they work and how they can be detected.

. . .

Techniques 41, 43 and 65 — Elevated COM interface

Three of the functional UAC bypasses rely on Component Object Model (COM). COM objects can specify that they need to run using Elevation, meaning with High process integrity. Under normal circumstances this will cause the UAC prompt to appear for the user. It is also possible to enable “Auto Approval” meaning that no UAC prompt will appear for the user and the COM object will run in an elevated process without triggering a UAC prompt, achieving a UAC bypass.

The three mentioned techniques all make calls to COM objects that have “Auto Approval” enabled. By looking through the UACME source code we can identify the CLSIDs and DLLs used by these three techniques:

- 41 — {3E5FC7F9–9A51–4367–9063-A120244FBEC7} — cmstplua.dll
- 43 — {D2E7041B-2927–42fb-8E9F-7CE93B6DC937} — colorui.dll
- 65 — {E9495B87-D950–4AB5–87A5-FF6D70BF3E90} — wscui.cpl

With an excellent tool called OleViewDotNet (released by James Forshaw of Google Project Zero), we can view the properties of COM objects. If we use this tool to view one of the entries above we can see that these COM objects indeed provide elevation using “Auto Approval”.

OleViewDotNet showing the properties of the “Security Center” COM object

All of these COM objects are implemented to run via the *dllhost.exe* DLL Surrogate process. This means that actions performed via these COM objects will originate from a *dllhost.exe* process. The CLSID GUID of the COM object is included on the *dllhost.exe* command line.

We can use this information to create a detection rule to identify processes spawned with High integrity from *dllhost.exe* with a command line that contains one of the three mentioned CLSID GUIDs.

```
DeviceProcessEvents
| where InitiatingProcessFileName =~ "dllhost.exe"
| where ProcessIntegrityLevel == "High"
| where InitiatingProcessCommandLine has_any ("E9495B87-D950-4AB5-87A5-FF6D70BF3E90", "3E5FC7F9-9A51-4367-9063-A120244FBEC7", "D2E7041B-2927-42fb-8E9F-7CE93B6DC937")
```

Another approach would be to create a list of the CLSIDs of COM objects that are known to spawn child processes with high integrity under normal circumstances and look for process creation from other CLSIDs. This would also allow identifying newly discovered Elevated COM interfaces in the future. This more advanced version of the rule is included in our subscription service, which provides access to over 150 custom detection rules for Azure Sentinel and MDE, contact us for details in case you are interested.

. . .

Technique 68 — Modify ms-windows-store protocol

This technique is well documented, for example on the LOLBAS website. It involves modifying a specific registry key and then launching the *wsreset* executable that resets the “Windows Store settings”.

The relevant registry key appears not to be logged in MDE but the technique can still be detected since the UAC elevated process spawns directly from *wsreset.exe* which normally does not spawn any child processes with high integrity.

```
DeviceProcessEvents
| where InitiatingProcessFileName =~ "wsreset.exe"
| where ProcessIntegrityLevel == "High"
```

. . .

Technique 61 — ChangePK / SLUI Registry tampering

This technique was disclosed by @mattharr0ey on his blog, it involves setting a registry key under the HKCU hive which is writable by the user and then launching *slui.exe* which is related to Windows Activation. This will then launch *ChangePK.exe* which in turn will execute an arbitrary process with high integrity without a UAC prompt, achieving a UAC bypass.

This chain of events with *slui.exe,* launching *ChangePK.exe* which launches a process with high integrity is very rare under normal use conditions and therefore can be detected with a simple rule:

```
DeviceProcessEvents
| where InitiatingProcessParentFileName =~ "slui.exe"
| where InitiatingProcessFileName =~ "changepk.exe"
| where ProcessIntegrityLevel == "High"
```

With these three simple detection rules we can cover all of the five undetected UAC bypasses implemented in UACME. As the state of both attack techniques and detection techniques changes continuously, there is obviously a lot of ground to cover implementing a good detection capability. Keep an eye out for our next FalconFriday articles where we will dive into other areas.

Falconfriday Uac Defender For Endpoint