

Scheduled Task Tampering

labs.f-secure.com/blog/scheduled-task-tampering/

Introduction

Microsoft recently published an article that [documented](#) how the HAFNIUM threat actor leveraged a flaw in how scheduled tasks are stored in the registry to hide their presence. This made it immediately clear that it was likely that the one presented was not the only flaw that affected the scheduled task component.

We began researching how the registry structure of the scheduled tasks could be abused to accomplish various goals, such as lateral movement and persistence.

Specifically, we investigated what were the minimum conditions for a task to be created, without going through the classic interfaces such as Remote Procedure Calls (RPC).

Both Microsoft's article and SpectreOps' [research on capability abstraction](#) confirmed that all the scheduled tasks eventually will be stored in the registry under the following registry keys:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks  
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree
```

The question was, would it be possible to create the relevant registry entries and create a task, and possibly, without generating events?

In this post we will explore two approaches that can be used to achieve the same result: create or modify a scheduled task and execute it, without generating the relevant telemetry. First, we will explore how direct registry manipulation could be used to create or modify tasks and how this did not generate the usual entries in the eventlog. Finally, an alternative route based on tampering with the Task Scheduler ETW will be presented that will completely suppress most of logging related to the Task Scheduler.

Available Telemetry

Before attempting to evade or circumvent how the Task Scheduler log its events, a brief introduction of what type of logging the Task Scheduler offers.

When creating, modifying, running or deleting a task, the following telemetry will be generated:

- 'Microsoft-Windows-TaskScheduler' - ETW provider that offers raw telemetry on task scheduler activity. Various security solutions that are based on ETW can tap directly into this data source.

- 'Microsoft-Windows-TaskScheduler/Operational' events from the Eventlog - If task History is enabled, this events will reflect what will be captured by the ETW provider mentioned above.
- Events 4698 to 4702 - In order to obtain these events, Object Auditing needs to be configured in the Local Security Policy's advanced auditing options. This source contains similar information as above but will end in the "Security" event log. Later on we will see what is the difference between this and 'Microsoft-Windows-TaskScheduler/Operational'. These events log the creation and modification of tasks, but not when a task is executed or which action a specific task executes. Throughout the post, this will be simply referred as the "Security" event log.

By analysing common attacker's knowledge frameworks such as ATT&CK, we can see that in the recommended auditing of the "Scheduled Task" technique we have both the 'Microsoft-Windows-TaskScheduler/Operational' and the 4XXX events from the Object Auditing. Since they contain similar information, it is normal to think that they can be used interchangeably. In fact, open source detection rule repositories like Sigma search for events generated by one source, but not both.

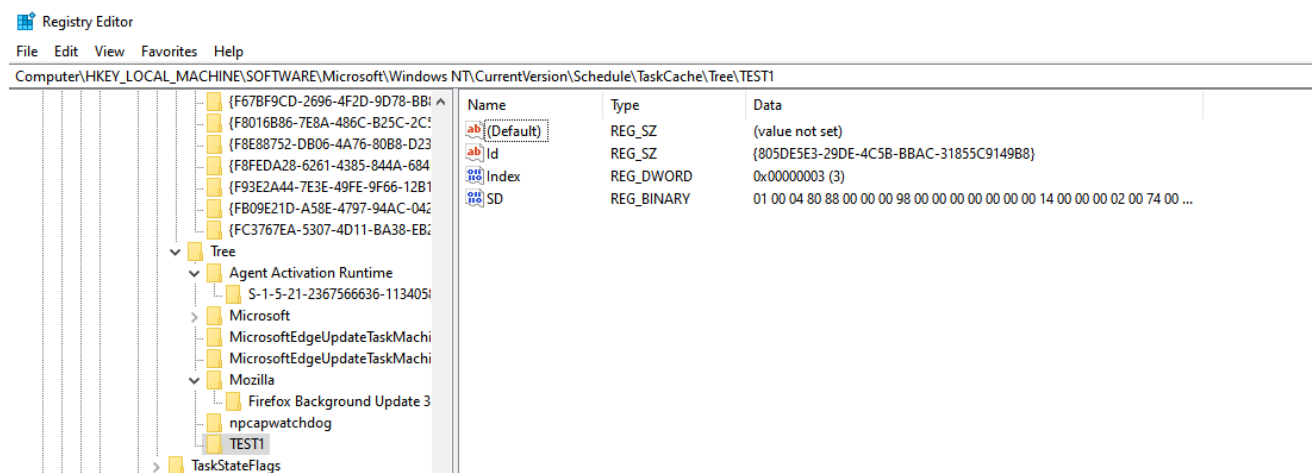
The following sections will document all the various techniques that were used to evade which types of events and how practical that would be in a real life scenario.

Registry Structure

If we analyse the structure of the aforementioned registry keys, it is possible to observe that the entry under the 'Tree\<taskName>' key the following values are present:

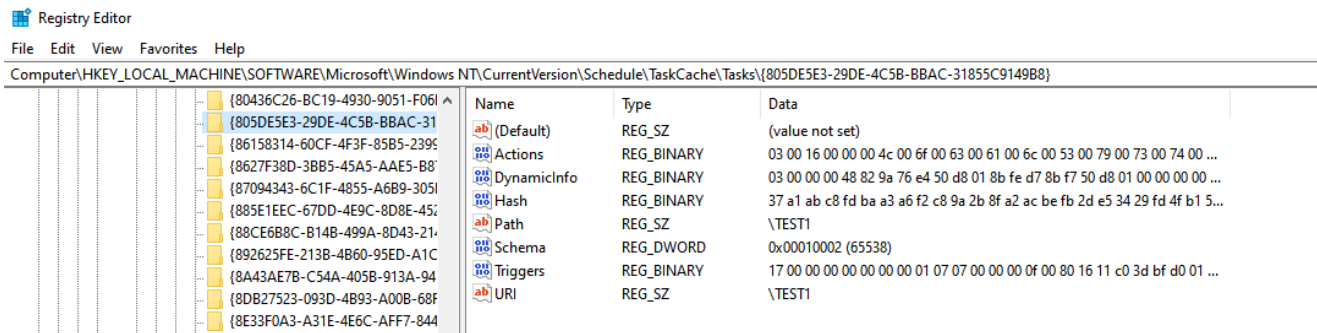
- 'Default' - Left to empty
- 'Id' - Is a GUID that indicates the subkey under the 'Tasks' key.
- 'Index' - DWORD value, could not understand what this was used for
- 'SD' - The security descriptor of the task, which was already discussed by Microsoft

The image below is an example of that:



The values of the 'Tasks' subkey are mostly binary blobs and encoded strings that indicate information such as:

- The task triggers
- The task action
- The hash of the XML file in the 'C:\windows\system32\tasks' folder



In addition to the above, another undocumented registry key was identified that played a big role in the task creation:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\Schedule\TaskCache\Plain\{GUID}
```

Where GUID is the unique identifier of the task as discussed above. No values were configured under the GUID subkey. We will see in the task creation section how this was fundamental.

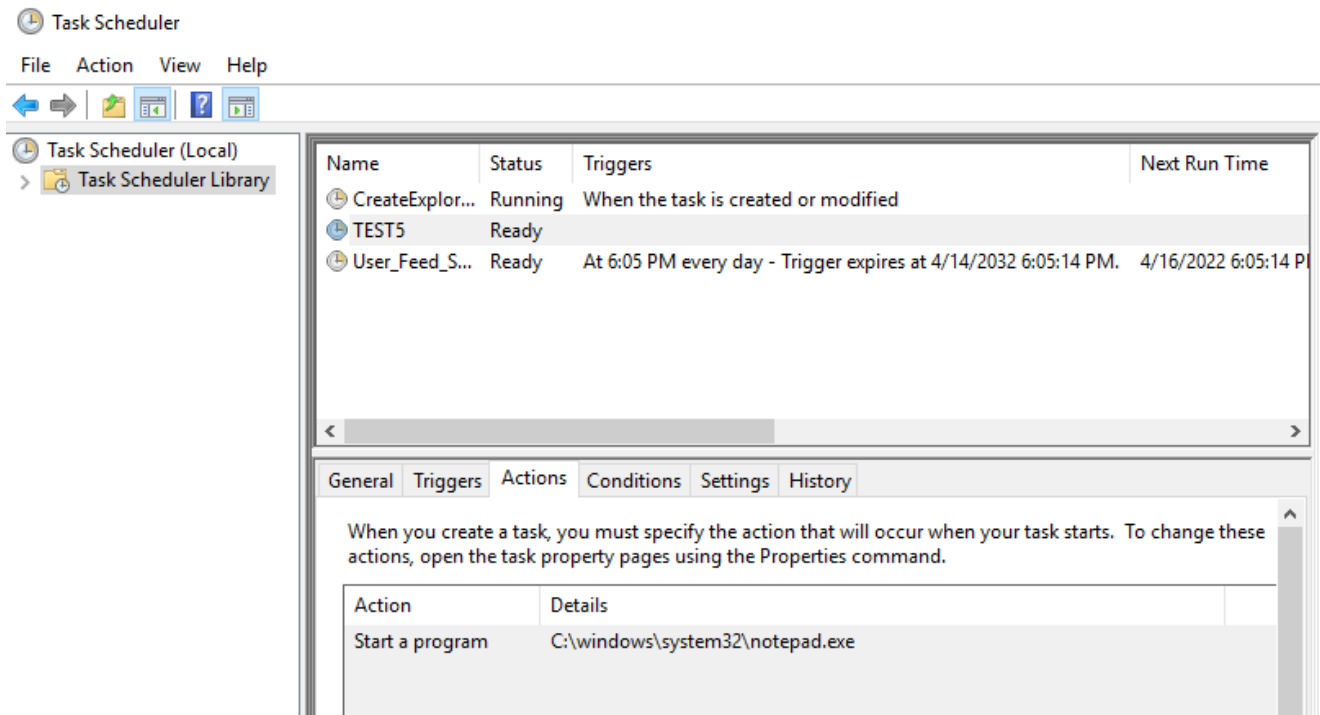
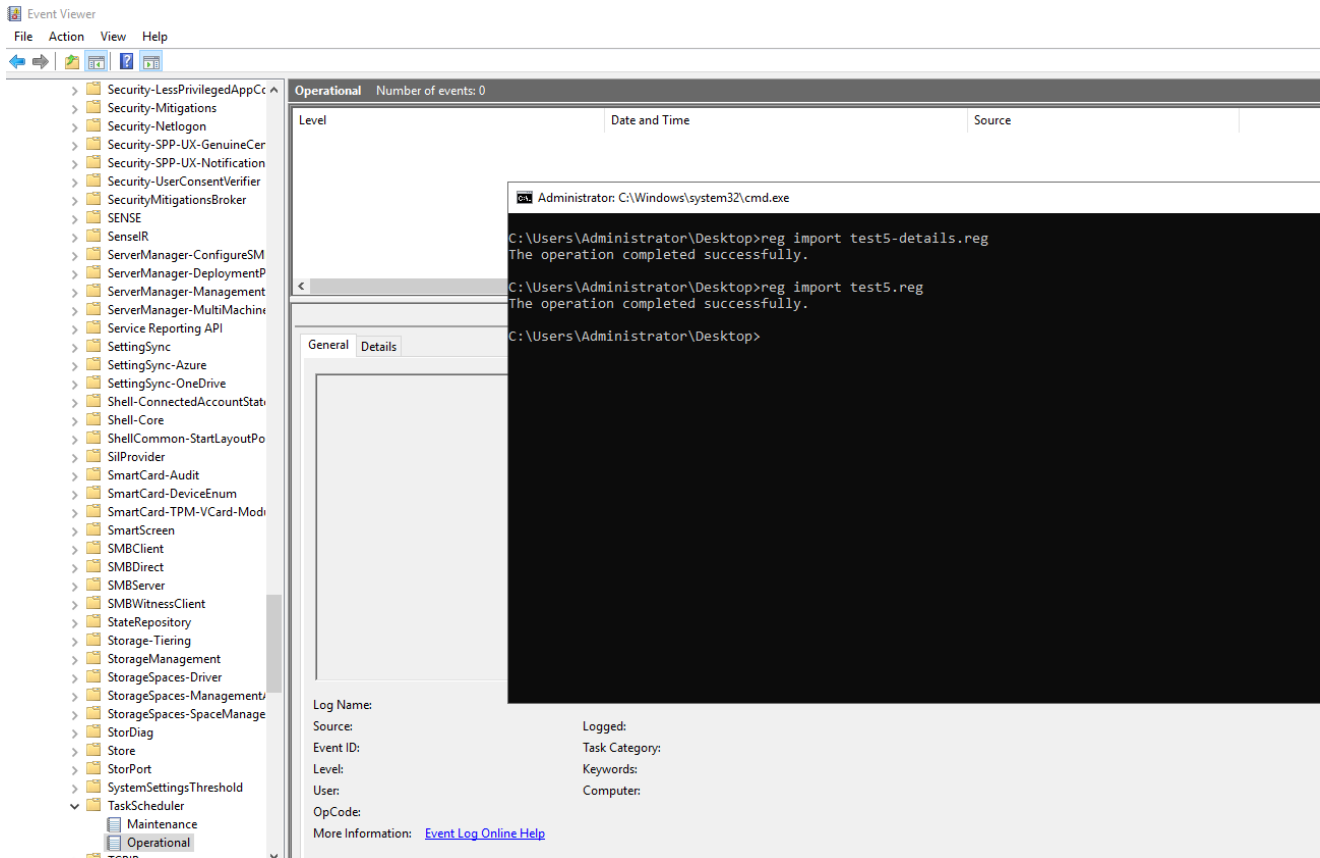
As mentioned above, our research focused on how to create or tamper with a task without going through the RPC interfaces, but only using direct registry manipulation. The following sections will explore how both the objectives were achieved and their respective caveats whilst understanding how tasks are saved in the registry.

Task Creation

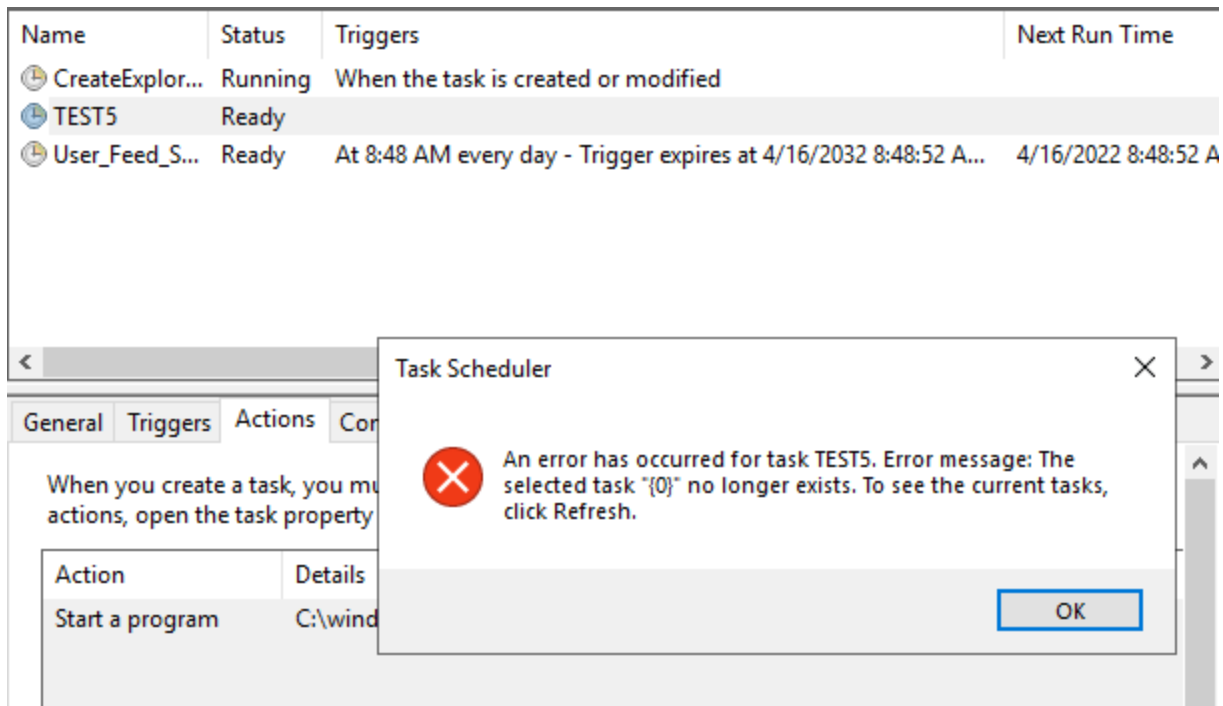
Without going too deep on reverse-engineering the RPC server responsible for the task creation, we exported the registry values of a previously created task, modified some values such as the task GUID, URI, and the Path, and imported the keys into the registry.

NOTE: In order to import the keys it is necessary to elevate to SYSTEM

After the import was done, it was possible to see that the task appeared in the Task Scheduler GUI but **no task creation events were generated, both in the EventLog or the Task Scheduler ETW feed.**



However, this technique had one big problem. When the created task was started, the following error was obtained:



This made the technique pretty useless, we had a cool way of creating tasks but no way to run them!

After some trial and error, two solutions were identified for this problem:

- Restarting the 'Schedule' service would "load" the task in the memory of the relevant svchost process, allowing the task to run
- Modification of the task definition would also force the load of the task in memory

The first approach, despite being noisier and more disruptive, **did not generate any additional events and now it was possible to execute the newly created task. Note that it is mandatory to create the GUID subkey under the 'Plain' key as discussed in the section above, otherwise this will not work.**

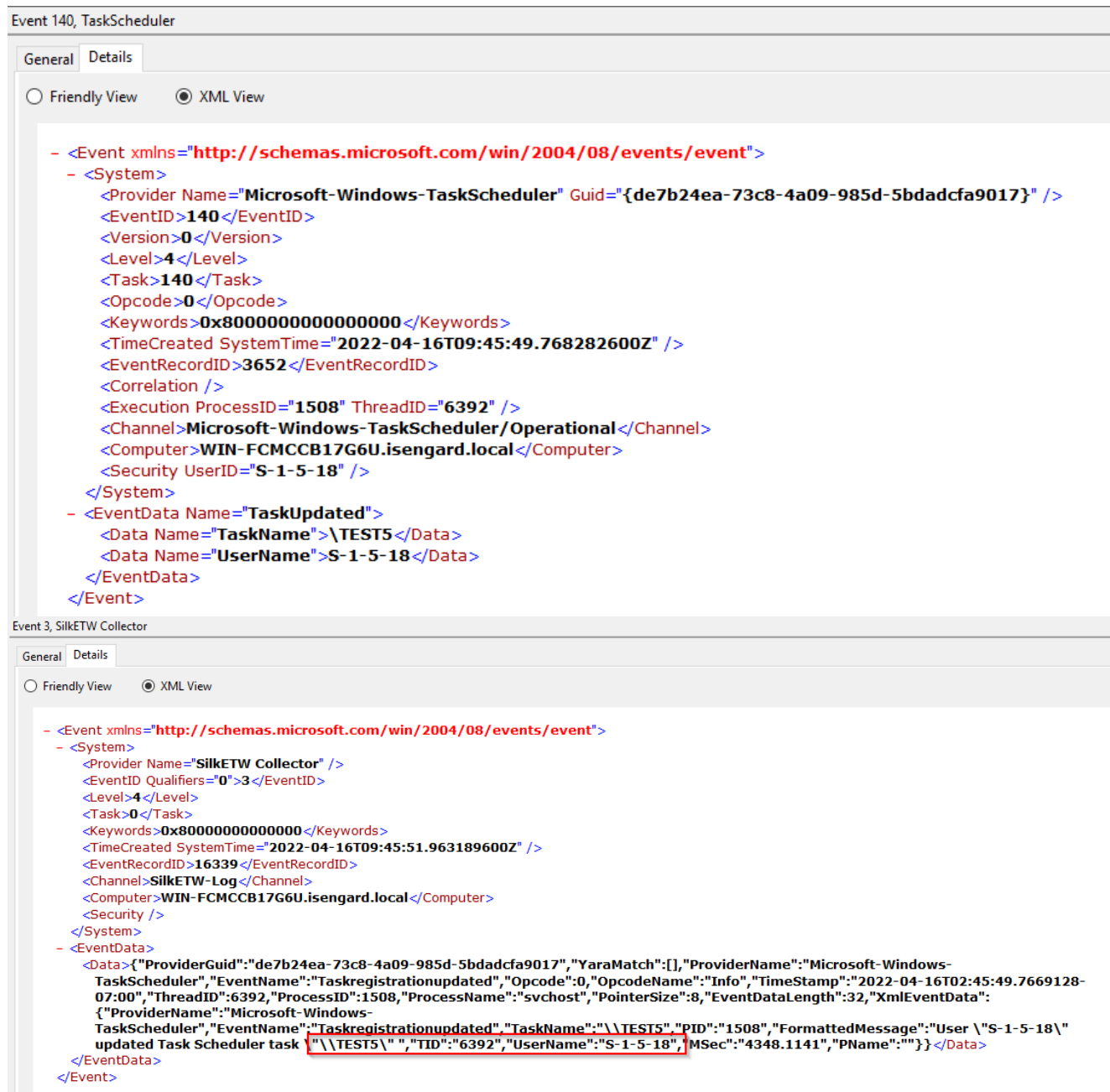
The second approach did not require any killing of innocent processes and restart of services and mainly consisted of the update of the newly created task definition.

```
C:\Users\Administrator\Desktop>schtasks /tn TEST5 /change /ru SYSTEM
SUCCESS: The parameters of scheduled task "TEST5" have been changed.
C:\Users\Administrator\Desktop>_
```

Note that despite the example showing the usage of the 'schtasks' utility, the same result could be achieved using the Task Scheduler RPC interface.

Now, the interesting part is that the update of a task definition will generate an entry in the event log.

The screenshots below show the log entries of the TaskScheduler auditing and the TaskScheduler ETW feed:



Interestingly, despite the task definition change generated events, the 'Microsoft-Windows-TaskScheduler/Operational' events contained only the information about the change that was applied. In the case above, we changed just the user that was used to execute the task, and therefore no additional information was present in the event information.

This was not the case with the Task Scheduler Security logs, as every modification in the task definition generated an entry in the event log that contained the whole task definition XML, as shown here:

Event 4702, Microsoft Windows security auditing.

General Details

Friendly View XML View

+ System

- EventData

SubjectUserSid S-1-5-18
SubjectUserName DESKTOP-BDTOO27\$
SubjectDomainName WORKGROUP
SubjectLogonId 0x3e7
TaskName \Microsoft\Windows\UpdateOrchestrator\Schedule Maintenance Work
TaskContentNew <?xml version="1.0" encoding="UTF-16"?> <Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task"> <RegistrationInfo>
<URI>\Microsoft\Windows\UpdateOrchestrator\Schedule Maintenance Work</URI> <SecurityDescriptor>D:P(A;FA;;;SY)(A;FRFX;;;LS)(A;FRFX;;;BA)
</SecurityDescriptor> </RegistrationInfo> <Triggers /> <Principals /> <Principal id="Author"> <UserId>S-1-5-18</UserId>
<RunLevel>LeastPrivilege</RunLevel> </Principal> </Principals> <Settings> <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
<DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries> <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
<AllowHardTerminate>true</AllowHardTerminate> <StartWhenAvailable>false</StartWhenAvailable>
<RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable> <IdleSettings> <Duration>PT10M</Duration> <WaitTimeout>PT1H</WaitTimeout>
<StopOnIdleEnd>true</StopOnIdleEnd> <RestartOnIdle>false</RestartOnIdle> </IdleSettings> <AllowStartOnDemand>true</AllowStartOnDemand>
<Enabled>false</Enabled> <Hidden>false</Hidden> <RunOnlyIfIdle>false</RunOnlyIfIdle> <WakeToRun>false</WakeToRun>
<ExecutionTimeLimit>PT72H</ExecutionTimeLimit> <Priority>7</Priority> </Settings> <Actions Context="Author"> <Exec> <Command>%systemroot%
\system32\usoclient.exe</Command> <Arguments>StartMaintenanceWork</Arguments> </Exec> </Actions> </Task>

ClientProcessStartKey 6473924464345406
ClientProcessId 2412
ParentProcessId 652
RpcCallClientLocality 0
FQDN DESKTOP-BDTOO27

Task Modification

Whilst exporting a task that was created on the attacker's host works, it was still necessary to understand the meaning of some of the structures in the registry. This would allow us to modify existing tasks directly on the compromised host and possibly hiding our activities.

Since attackers are likely going to be interested in modifying the action that a task performs, we will analyse the 'Actions' value from the registry. An example is shown below:

As it is possible to see, two Unicode strings are present in the structure, the value "Author" and the command that will be executed.

A deeper examination of the values within the binary blob led to the following:

Static Value

Sizeof Author String

Author string

Edit Binary Value

Value name:
Actions

Value data:

00000000	03 00	0C 00	00 00	00 00	41 00 A .
00000008	75 00	74 00	68 00	6F 00	u . t . h . o .	
00000010	72 00	66 66	00 00	00 00	r . f f	
00000018	36 00	00 00	43 00	3A 00	6 C . : .	
00000020	5C 00	77 00	69 00	6E 00	\\ . w . i . n	
00000028	64 00	6F 00	77 00	73 00	d . o . w . s	
00000030	5C 00	73 00	79 00	73 00	\\ . s . y . s	
00000038	74 00	65 00	6D 00	33 00	t . e . m . 3	
00000040	32 00	5C 00	63 00	6D 00	2 . \\ . c . m	
00000048	64 00	2E 00	65 00	78 00	d e . x	
00000050	65 00	16 00	00 00	2F 00	e /	
00000058	63 00	20 00	63 00	61 00		

Sizeof Unicode String

Value of 0x66,0x66 seems related to tasks that start a program

Unicode String (task action)

OK Cancel

- '03 00' - seem to be static bytes
- Sizeof author string, seems to indicate who's the user that is supposed to run the task, other tasks were found to have 'LocalSystem' as an example, others 'Authenticated Users'
- '66 66' - seems to be static for tasks that start another program, other COM bases tasks were found to have different values
- Sizeof action string - a value that indicates how long the action string will be
- Action string - UNICODE representation of the string that indicates the action to perform

Although it was not possible to fully understand the meaning of each field, some common patterns were identified.

It must be noted that attackers don't necessarily need to know the meaning of each bit of the key mentioned above. In fact, it is possible to create a task with the desired malicious action locally, export the key and then import it into the target system.

With this knowledge, attackers can edit the relevant fields of a Task's 'Action' and execute malicious actions. It must be noted that the task should be "loaded" in memory using any of the approaches proposed above.

If the attackers restarted the scheduler service to load the modified task in memory, no events were generated in the Security eventlog or in the 'Microsoft-Windows-TaskScheduler/Operational' logs.

Alternative Approach: ETW Tampering

ETW patching is a technique whereby an attacker abuses flaws in the ETW architecture with the aim of preventing either a specific process or the whole system from generating ETW telemetry. This was popularised by MDSec's research "[Hiding Your .NET - ETW](#)" and Palantir's [Tampering with Windows Event Tracing](#). As an example, MDSec's research was focused towards hiding .NET related events, however, considering how ubiquitous ETW is on Windows systems, many other abuse opportunities are present.

During the research, it was in fact possible to verify that all the Task Scheduler logs were generated by the Eventlog service, but the event information was sent by the Scheduler service using ETW. ****This meant that if an attacker is able to tamper with the ETW on the Scheduler service, no logs will be generated.****

Recent research on [patchless AMSI bypasses by CCob](#) showed that it was possible to combine hardware breakpoints and Vectored Exception Handlers (VEH) in order to modify how a function behaved. In case of the AMSI bypass, the breakpoint was set to the start of the 'AmsiScanBuffer' function and the custom VEH handler simply returned a predefined value and restored the execution flow as if the 'AmsiScanBuffer' function terminated normally. We won't go into the technical details on how the technique works as the blog post above does an excellent job in explaining the required concepts.

Adapting this to ETW tampering was easy enough, starting from the [initial PoC](#) published by the blog's author. The only major modification was that the VEH and breakpoints in the provided PoC were applied only to the current thread, and therefore to make it work for the whole process it was necessary to iterate through all the process' threads and configure the breakpoints and VEH accordingly. In addition, a logic to scan for new threads on a periodic basis was also added, because threads that were newly created by the Scheduler service would not be subject to this bypass.

The modified PoC was compiled as a Windows DLL and injected into the svchost.exe process that was hosting the Scheduler service. The following video shows the results of the attack:

As it is possible to see, before the attack the various events were generated as expected after tasks were executed or modified. However, after injecting the malicious DLL onto the target svchost process, no more events were sent to the eventlog.

Despite the video shows the usage of process hacker to perform DLL injection, in a real life scenario it would be possible to convert the DLL to a Reflective DLL and inject it using stealthier process injection techniques.

This tampering approach presents multiple OpSec advantages over the classic ETW function patching, as no modification was done against the in-memory DLL. This would avoid all the detections based on Write on Copy that are also implemented in open-source scanners like [Moneta](#). In addition, removing the VEH and breakpoints is relatively trivial once the attackers performed their actions and should leave minimal traces on the attacked process. Similar results can be achieved with different hooking techniques that do not rely on patching the memory of the '.text' section of a DLL, such as Page Guard hooking or similar.

Finally, using this technique would allow attackers to avoid more convoluted paths such as the ones that were previously described that involved modifying binary values in the registry. Using the classic scheduled task deployment that go via COM or RPC in conjunction with this bypass would effectively have the same effect in terms of defense evasion.

To summarise, the table below shows the various task scheduler log sources and how they are affected by the proposed techniques:

Technique	Security Logs	Microsoft-Windows-TaskScheduler/Operational	Microsoft-Windows-TaskScheduler ETW
Task creation/modification via RPC	Yes	Yes	Yes
Task creation/modification via registry manipulation	No	No	No
Task creation/modification with tampered ETW	Yes	No	No
Task modification via RPC	Yes	Yes	Yes
Task modification via Registry and Scheduler restart	No	No	No
Task Executed via RPC	No	Yes	Yes

NOTE: We are not planning to release any tool that automates this attack anytime soon.

Abuse Cases

To summarise, the following abuse case were identified:

- An attacker creates a malicious scheduled task via the registry to establish persistence and wants to hide their activity
- An attacker tampers with an existing and benign task to inject a malicious action
- An attacker creates a malicious task via either RPC or registry and masks it as a benign task

The third case presents an interesting scenario, since if an attacker can create a task - either with the classic techniques or this variation - they can then "spooof" the 'Actions' value to make the task appear legitimate. If the 'Actions' field is modified, its value will be reflected immediately in the Task Scheduler GUI. However, if the task definition is not updated or the host is rebooted, each time the task will be executed, the malicious action will be triggered instead.

To better explain this, an example is provided below:

- Attacker creates a malicious task A on the victim host that will execute 'C:\evil.exe'
- The attacker then modifies the registry entry of the newly created task and changes the action to be 'C:\legitimate.exe'
- The attacker can run the task, and 'C:\evil.exe' will be executed
- Defenders trying to spot malicious activity on the host will see that the created task should have executed 'C:\legitimate.exe' but no logs of task modification are present.

Note that in this proposed hypothetical scenario, the attacker might not necessarily want to hide the initial task creation activity. This is mainly to impede manual investigation on the host, as the Action in the Task Scheduler GUI would differ from the actual task that is being executed.

Extra: Lateral Movement

Due to permission restrictions, as mentioned by other researchers in [publicly available tools](#), normally manually modifying registry keys associated to scheduled tasks is not possible. This is because the relevant registry keys have ACLs such that only the SYSTEM user can modify them.

To validate that, let's try to add one of the keys required for the task creation using Impacket's reg.py script, with an account that has admin rights over the remote host:

```
reg.py isengard.local/administrator@WIN-FCMCCB17G6U.ISENGARD.LOCAL add -keyName
'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\
{61687CDA-FEBB-4F23-8E3B-5F2D8778CA7B}'
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth
Corporation
```

```
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```

Well, now the question is, can we use the SYSTEM account remotely and use what we discovered to perform this attack remotely? The short answer is yes, but let's explore what are the caveats.

In our experiments, we found that it was possible to use the Silver Ticket technique to craft a ticket that had the SYSTEM SID (S-1-5-18) in the PAC. It goes without saying that in order to perform a Silver Ticket attack, you must possess the NTLM or AES key associated with a computer account. This can be obtained in multiple ways, such as using the "Shadow Credentials" attack or by extracting it from the remote host using remote registry with tool such as Impacket's secretsdump.py.

Now, let's assume that we obtained the NTLM hash of a computer account, what's next?

Using Impacket's ticketer we can forge the silver ticket with the "-extra-sid" parameter:

```
ticketer.py -nthash [NTLM] -domain-sid S-1-5-21-861978250-176888651-3117036350 -
domain isengard.local -dc-ip 192.168.182.132 -extra-sid S-1-5-18 -spn HOST/WIN-
FCMCCB17G6U.isengard.local WIN-FCMCCB17G6U$
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth
Corporation
```

```
[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for isengard.local/WIN-FCMCCB17G6U$
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in WIN-FCMCCB17G6U$.ccache
```

Now that the ticket is saved, we can just use it with other impacket tools and apply the changes to the registry:

```
export KRB5CCNAME=/tmp/WIN-FCMCCB17G6U\$.ccache
```

```
reg.py -k -no-pass 'ISENGARD.LOCAL/WIN-FCMCCB17G6U$'@WIN-FCMCCB17G6U.ISENGARD.LOCAL  
add -keyName 'HKLM\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\Schedule\TaskCache\Tasks\{61687CDA-FEBB-4F23-8E3B-5F2D8778CA7B}'  
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth  
Corporation
```

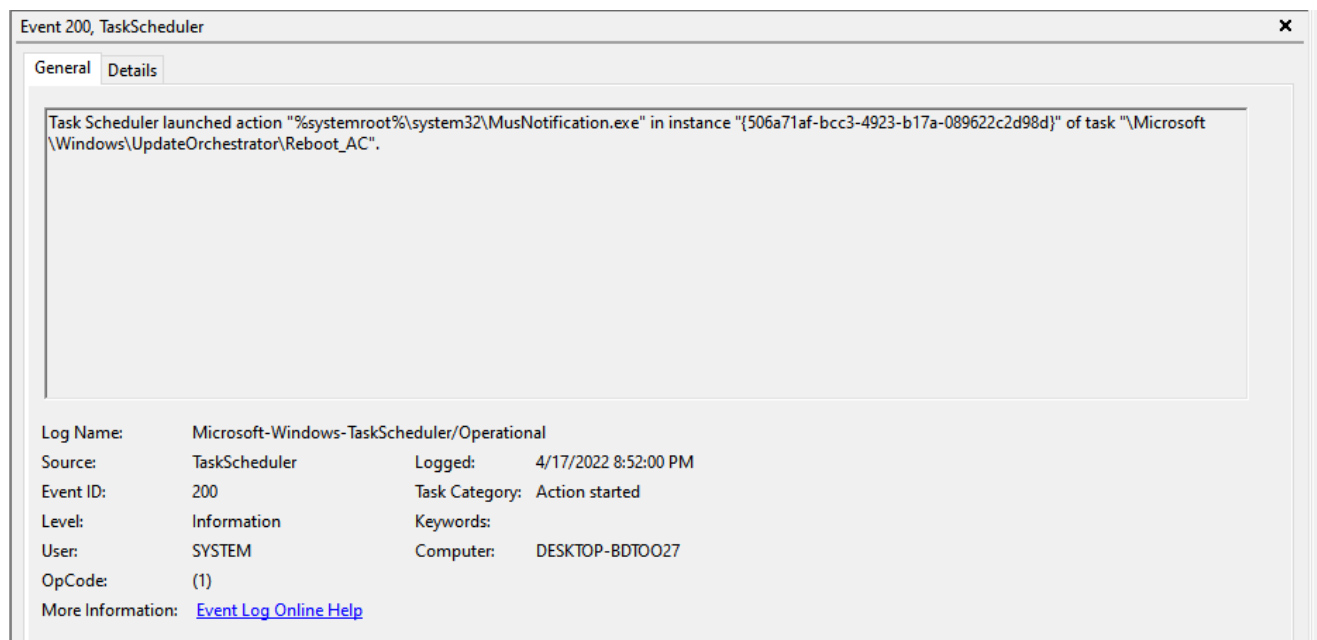
```
Successfully set subkey HKLM\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\Schedule\TaskCache\Tasks\{61687CDA-FEBB-4F23-8E3B-5F2D8778CA7B}
```

Despite the command above shows only the creation of one of the various keys required for a task to be created, it would be simple enough to automate the whole process.

Detection and Hunting

From a defensive standpoint, the first thing we should mention is that the task creation and modification did not generate events, the action of starting a task will.

Specifically, every time a task starts an action such as running an EXE you will have an event that looks like the following:

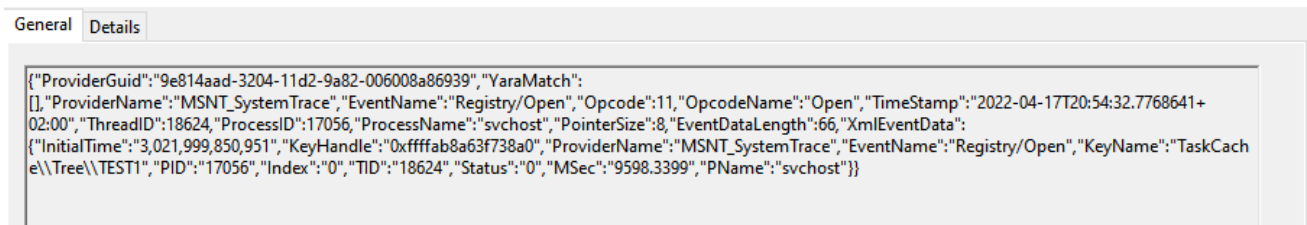


Note that this event is not present in the Task Scheduler's Security events, that will log only creation, deletion and modification of a task. Therefore, if ETW is tampered on the Scheduler service, the action started by a task will not be captured.

An alternative and more robust approach would include the parent-child process relationship, as all the processes started by the Task Scheduler will have a specific svchost as a parent.

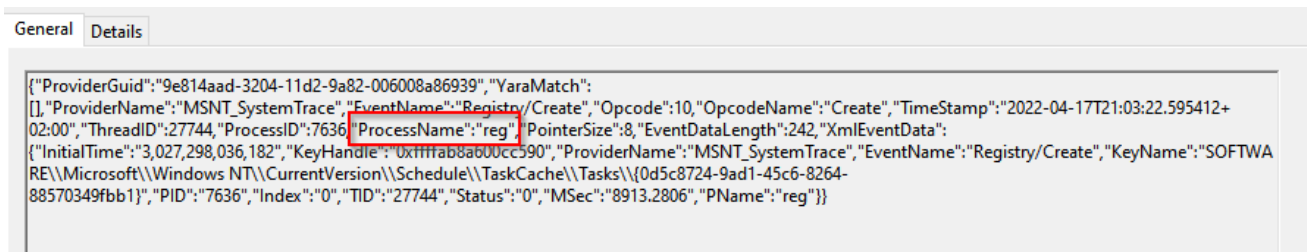
From a hunting and detection perspective, looking for registry operations that affect the keys mentioned in this article that are not originated from a svchost process is probably a good way to spot attackers that perform this attack without too much sophistication. The first

image below shows a benign event generated from svchost:



```
General Details
{"ProviderGuid":"9e814aad-3204-11d2-9a82-006008a86939","YaraMatch":
[],"ProviderName":"MSNT_SystemTrace","EventName":"Registry/Open","Opcode":11,"OpcodeName":"Open","TimeStamp":"2022-04-17T20:54:32.7768641+
02:00","ThreadID":18624,"ProcessID":17056,"ProcessName":"svchost","PointerSize":8,"EventDataLength":66,"XmlEventData":
{"InitialTime":"3,021,999,850,951","KeyHandle":"0xfffffab8a63f738a0","ProviderName":"MSNT_SystemTrace","EventName":"Registry/Open","KeyName":"TaskCach
e\\Tree\\TEST1","PID":"17056","Index":"0","TID":"18624","Status":"0","MSec":"9598.3399","PName":"svchost"}}
```

This image shows a similar registry operation, but performed by the 'reg.exe' process rather than svchost:



```
General Details
{"ProviderGuid":"9e814aad-3204-11d2-9a82-006008a86939","YaraMatch":
[],"ProviderName":"MSNT_SystemTrace","EventName":"Registry/Create","Opcode":10,"OpcodeName":"Create","TimeStamp":"2022-04-17T21:03:22.595412+
02:00","ThreadID":27744,"ProcessID":7636,"ProcessName":"reg","PointerSize":8,"EventDataLength":242,"XmlEventData":
{"InitialTime":"3,027,298,036,182","KeyHandle":"0xfffffab8a600cc590","ProviderName":"MSNT_SystemTrace","EventName":"Registry/Create","KeyName":"SOFTWA
RE\\Microsoft\\Windows NT\\CurrentVersion\\Schedule\\TaskCache\\Tasks\\{0d5c8724-9ad1-45c6-8264-
88570349fbb1}","PID":"7636","Index":"0","TID":"27744","Status":"0","MSec":"8913.2806","PName":"reg"}}
```

It goes without saying that attackers can easily spawn an svchost process or inject into an existing one to perform this attack, but that might (should?) trigger other controls.

On the ETW tampering side, the recommendations are not specific to the Scheduler abuse but should be mainly focused towards:

- Detecting memory injection against system processes - In order to deploy the ETW patches, attackers are likely to inject code into a target process. Detecting process injection is a topic that goes way beyond the scope of this research and therefore we won't spend too much time on that.
- Detecting abuse of VEH - NCC published an [interesting post](#) on how this could be approached. However, the proposed detections were mainly focused on detecting handlers that pointed to memory regions that are not backed by a file on disk. Advanced attackers could easily avoid those indicators by either hiding their code on memory pages that are associated with files that exist on disk or simply removing the VEH after they performed their actions.
- Detecting changes in ETW log volume - rapid changes to the volume of logs generated by a specific feed should be treated as suspicious, although attackers can still revert back the changes after the malicious actions are complete. Note that implementing this in a big enough estate could be extremely difficult to accomplish.

The following Sigma rule can be used to hunt for attackers that attempted to manually modify scheduled tasks from the registry:

```

title: Task Tampering Detection
status: experimental
description: Detects manual Scheduled Task tampering via registry modification
author: Riccardo Ancarani
date: 03/05/2022
level: high
logsource:
  product: windows
  service: sysmon
detection:
  selection:
    EventID: 13
    TargetObject: 'HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Schedule\TaskCache\*'
  filter:
    - Image|endswith: '\svchost.exe'
  condition: selection and not filter

```

The Sigma rule above leverages Sysmon's logging capabilities and looks for the event ID 13 (Registry write) on the specific keys used for this attack. It also filters all the activity generated from "svchost.exe".

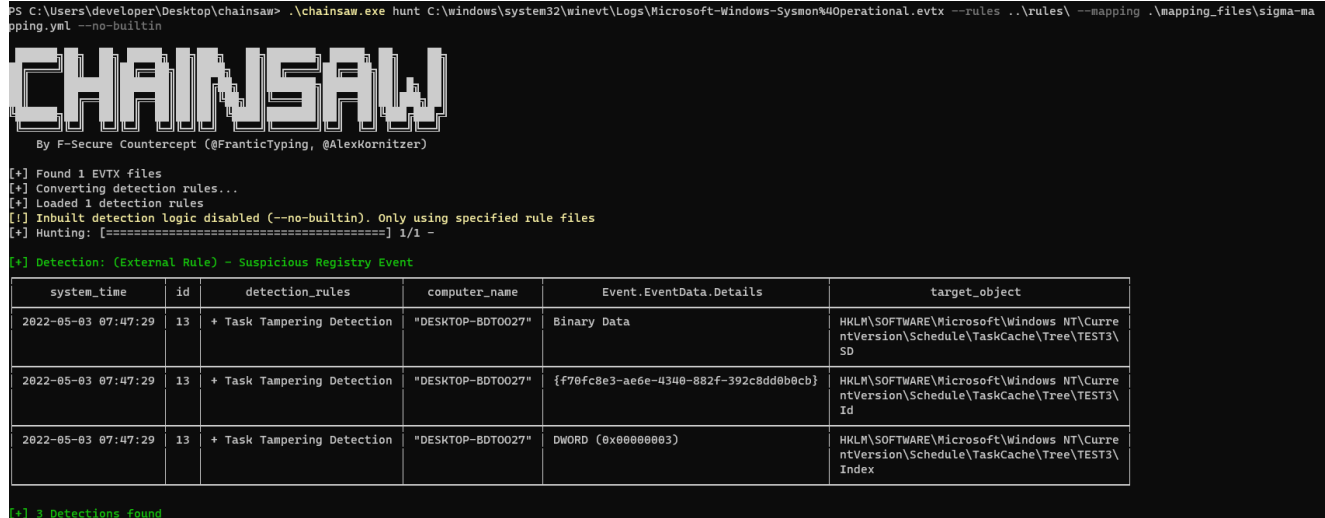
The rule was tested against Sigma's baseline [EVTX files](#) and did not produce any false positive.

An example of a true positive successfully identified using the [Chainsaw](#) tool:

```

PS C:\Users\developer\Desktop\chainsaw> .\chainsaw.exe hunt C:\windows\system32\winevt\Logs\Microsoft-Windows-Sysmon%4Operational.evtx --rules .\rules\ --mapping .\mapping_files\sigma-mapping.yml --no-builtin

```



system_time	id	detection_rules	computer_name	Event.EventData.Details	target_object
2022-05-03 07:47:29	13	+ Task Tampering Detection	"DESKTOP-BDT0027"	Binary Data	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\TEST3\SD
2022-05-03 07:47:29	13	+ Task Tampering Detection	"DESKTOP-BDT0027"	{f70fc8e3-ae6e-4340-882f-392c8dd0b0cb}	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\TEST3\Id
2022-05-03 07:47:29	13	+ Task Tampering Detection	"DESKTOP-BDT0027"	DWORD (0x00000003)	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\TEST3\Index

```

[+] 3 Detections found

```

Conclusions

In this post we examined various techniques that attackers can use to create or tamper with scheduled tasks using mostly registry operations. From a defense evasion perspective, this would limit the logs that defenders can use to spot malicious activity.

High-level detection strategies were provided to help blue team members to hunt for such abuses.