#NoFilter: Abusing Windows Filtering Platform for privilege escalation

Ron Ben Yizhak

About Me

Ron Ben Yizhak

- Security Researcher @ deep instinct
- Interested in malware campaigns, attack vector and evasion techniques
- Enjoys rock climbing and volleyball

Agenda

01 **Intro**
Technical background and known techniques

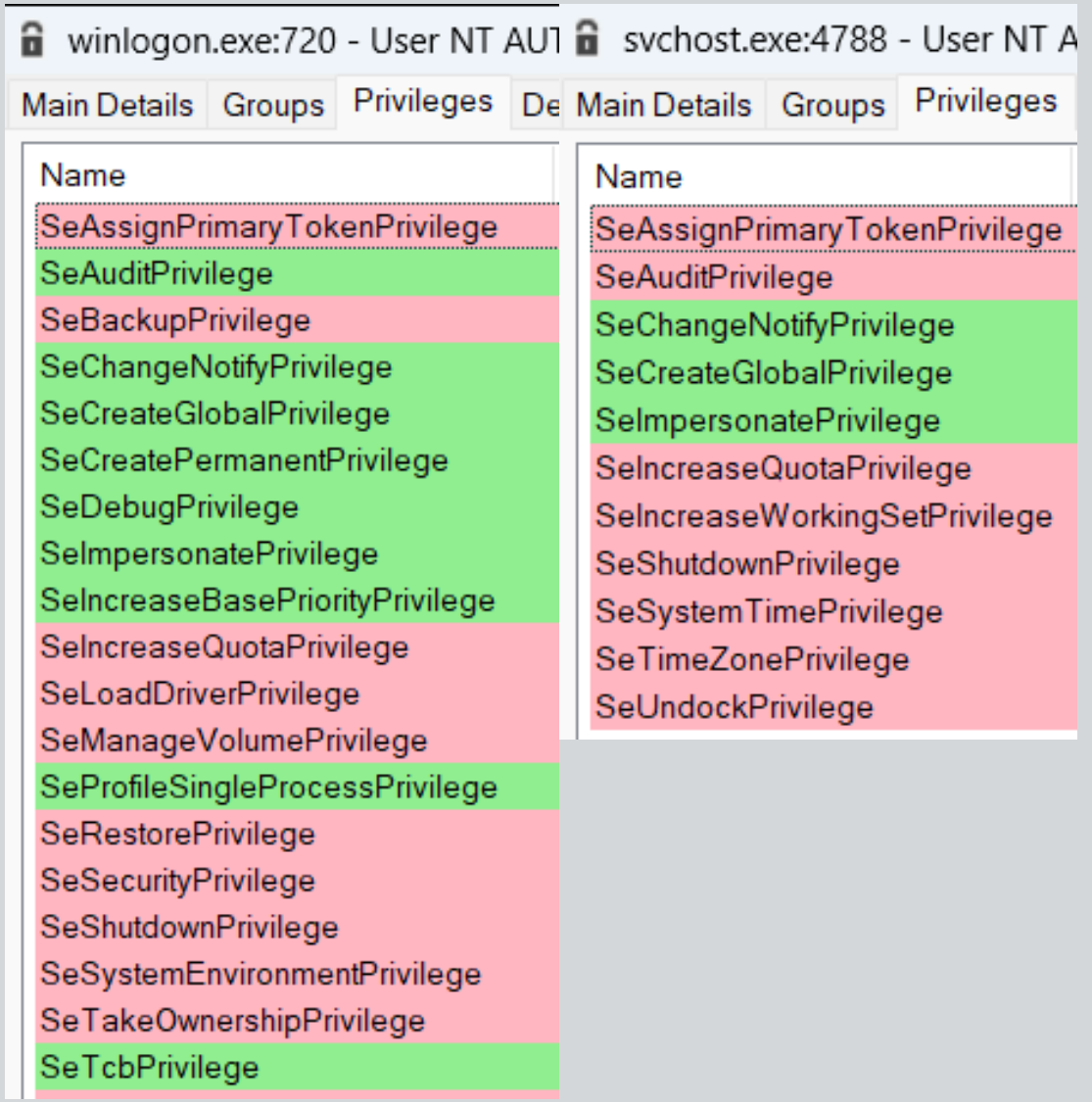
02 **Reverse Engineering**
Drivers and RPC components of WFP

03 **Attacks**
Duplicating tokens via Windows Filtering Platform

04 **Conclusion**
Detection methods and further research

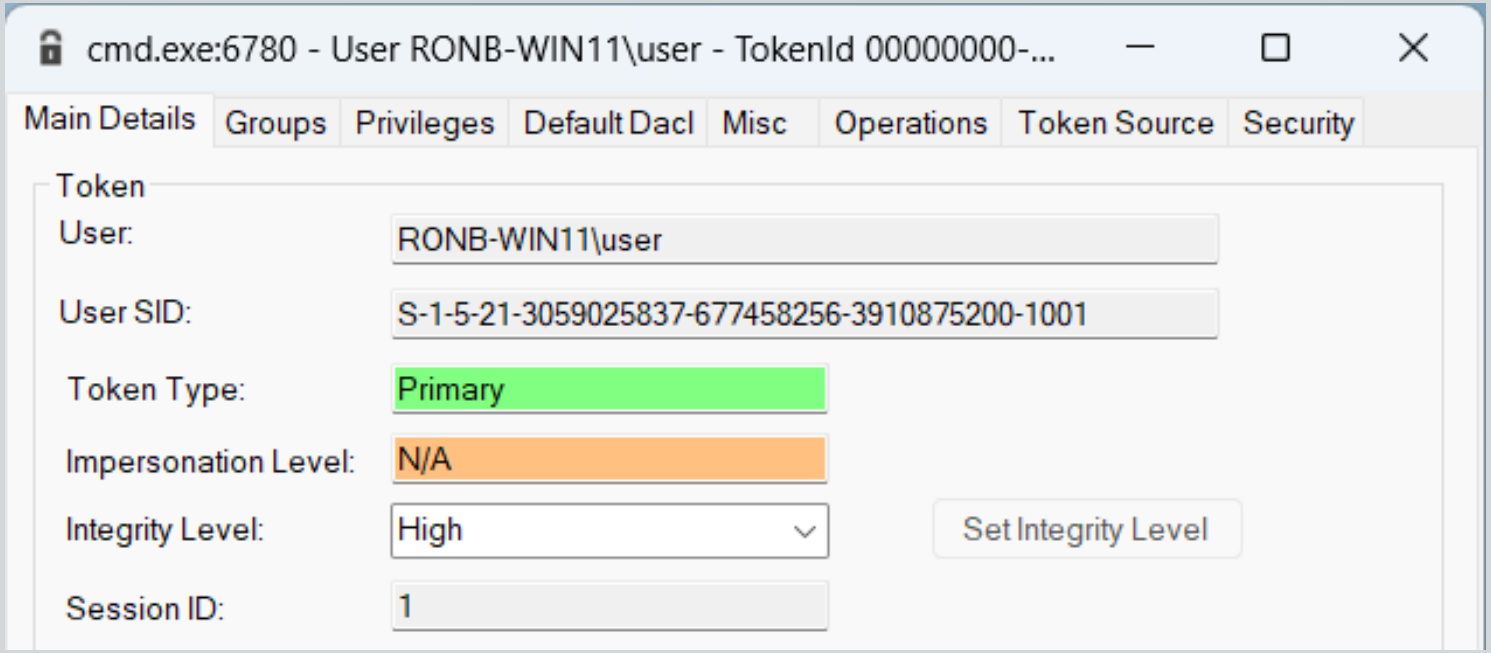
Privilege Escalation

- Executing code in a higher level of permissions on the system
- Useful when compromising a weak process
- Most privileged user is “NT AUTHORITY\SYSTEM”, required for:
  - Reading the SAM hive
  - LSASS Shtinkering
  - Dump Kerberos tickets with Rubeus



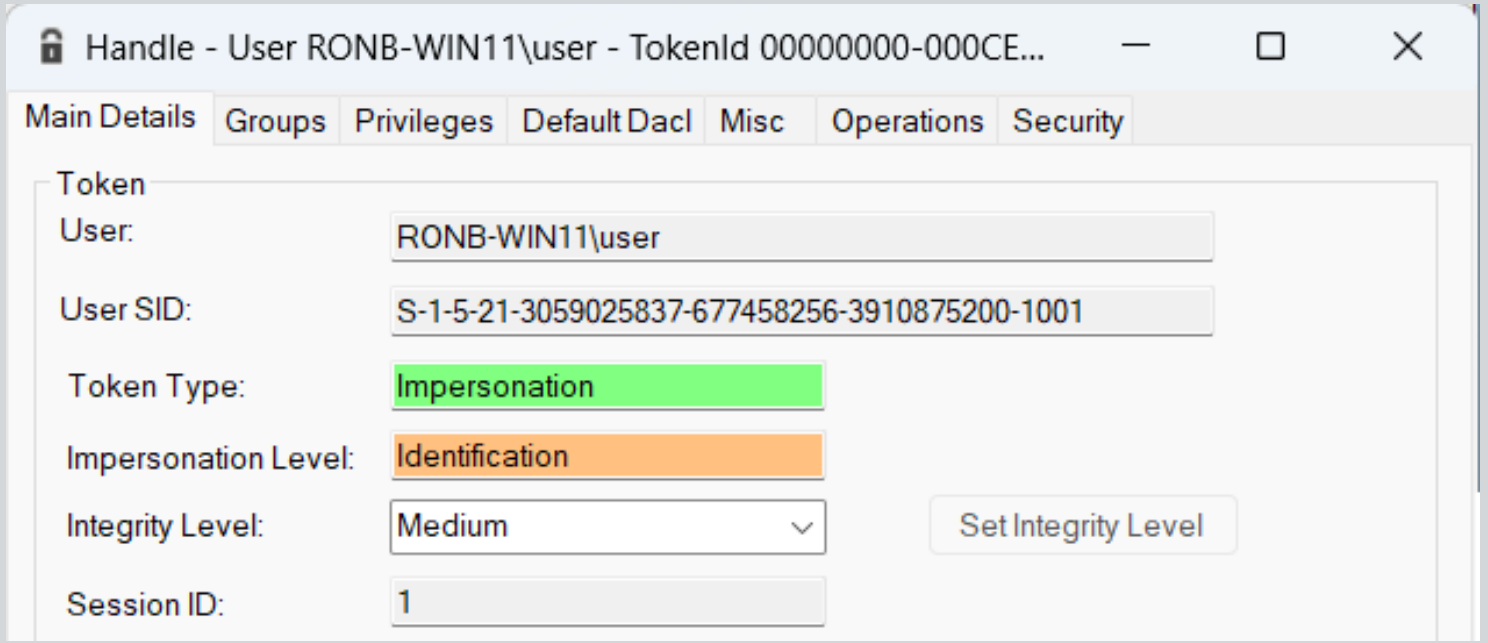
Access Tokens

- Describes the **security context** of a process or thread
- Used for identification when interacting with a securable object
- There are two token types:
  - Primary
  - Impersonation



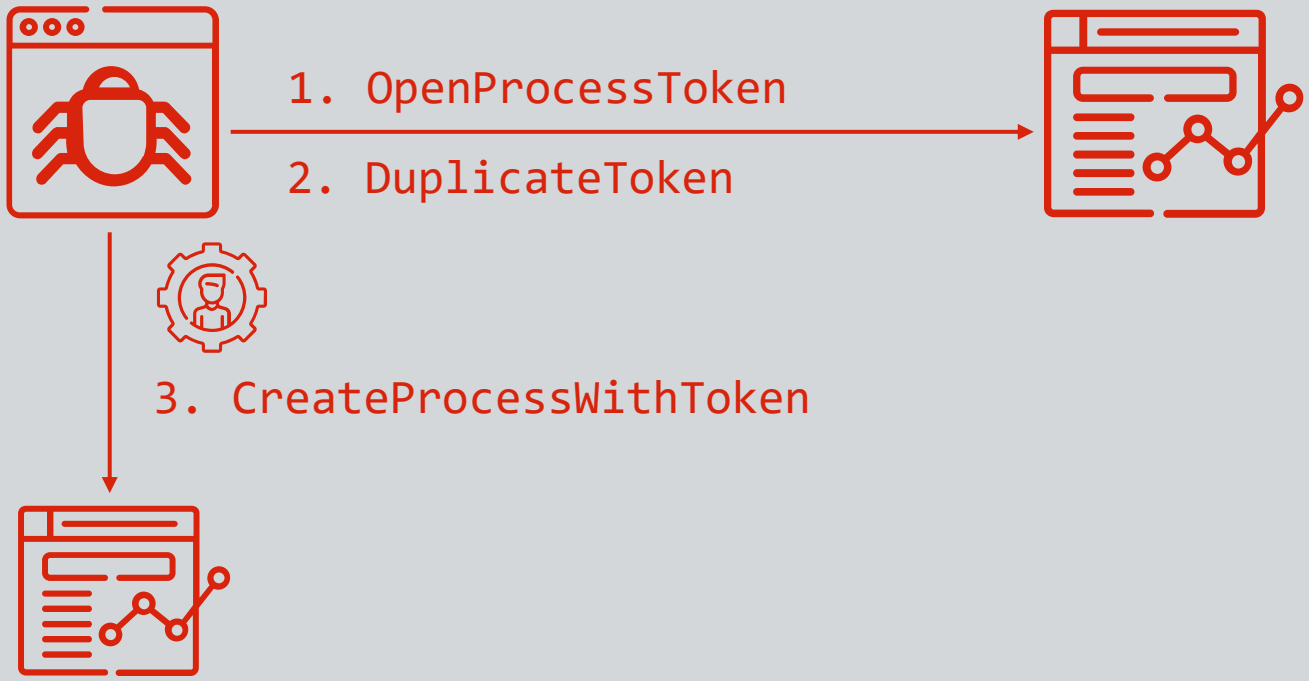
Impersonation Tokens

- Threads can use a different security context than the owning process
  - i.e., in a server application - impersonate a client to act on its behalf
- Impersonation levels:
  - Annonymous
  - Identification
  - Impersonation
  - Delegation



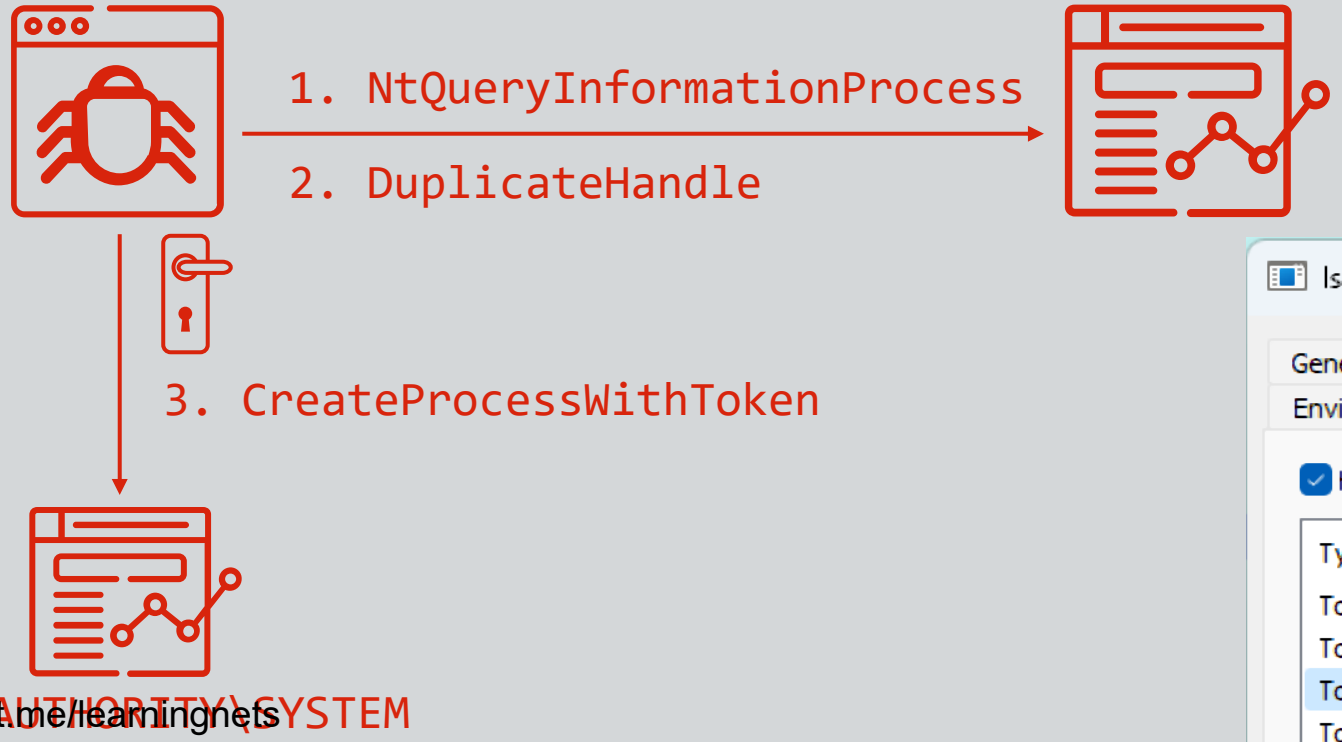
Known Techniques – DuplicateToken

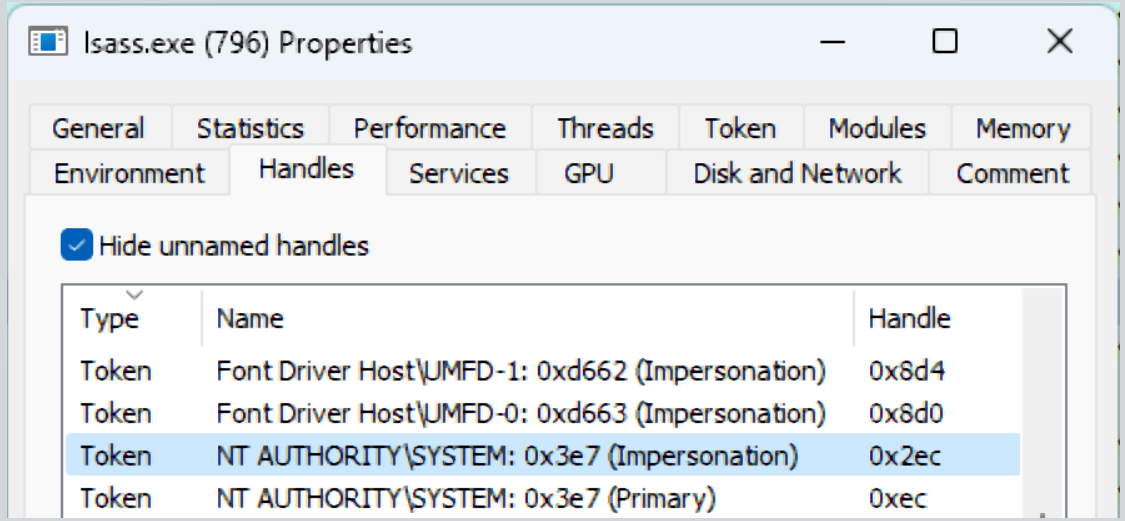
- The primary token of a process can be accessed
- Child process can be launched with a different token
- Used by PowerSploit and Rubeus



Known Techniques – DuplicateHandle

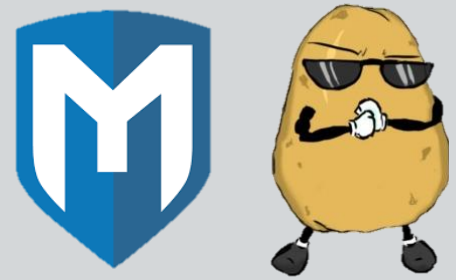
- The tokens held by a process are listed in the handle table
- Handle tables of other processes can be retrieved
- Used by meterpreter (getsystem -t 4) and mimikatz (token::elevate)

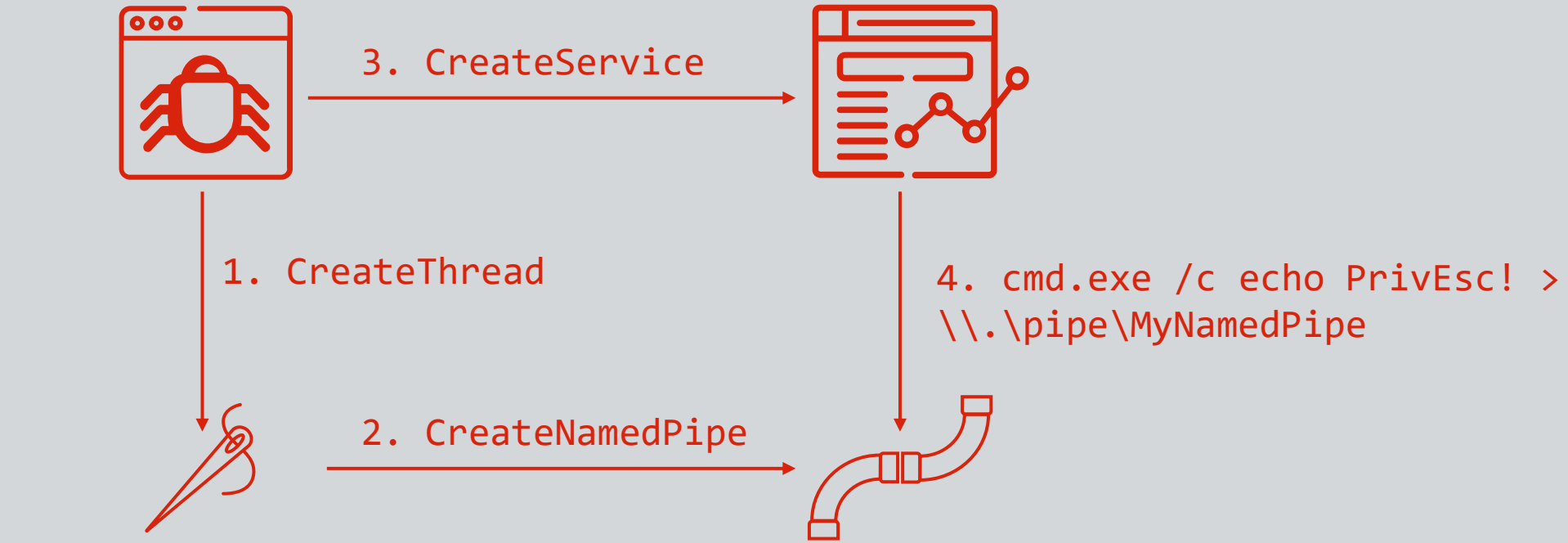




Known Techniques - Impersonation

- Impersonation is abused for privilege escalation
- Services are manipulated to connect to a listener thread
- Used by meterpreter (getsystem –t 1) and various potato tools





Known Techniques - Detection

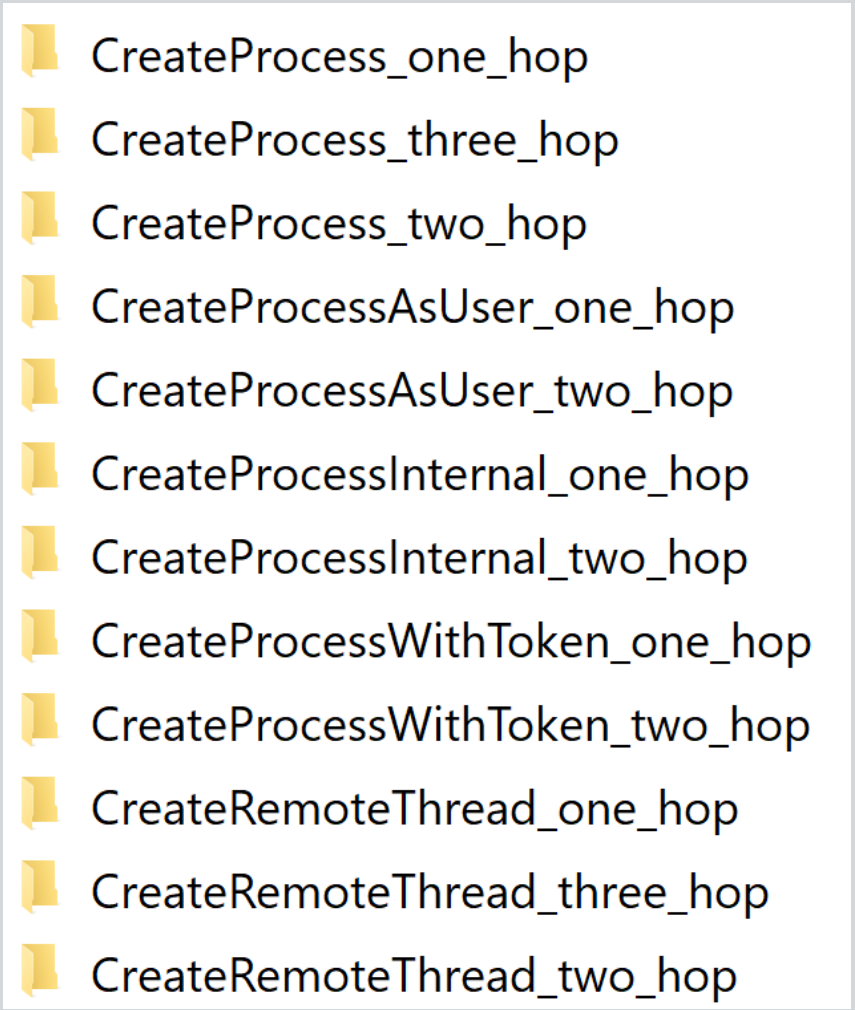
The techniques can be easily detected by tracking the following events:

- `DuplicateToken` / `DuplicateHandle` return a token for NT AUTHORITY\SYSTEM
- Thread impersonating NT AUTHORITY\SYSTEM after calling:
  - `ImpersonateNamedPipeClient`
  - `CoImpersonateClient`
  - `RpcImpersonateClient`

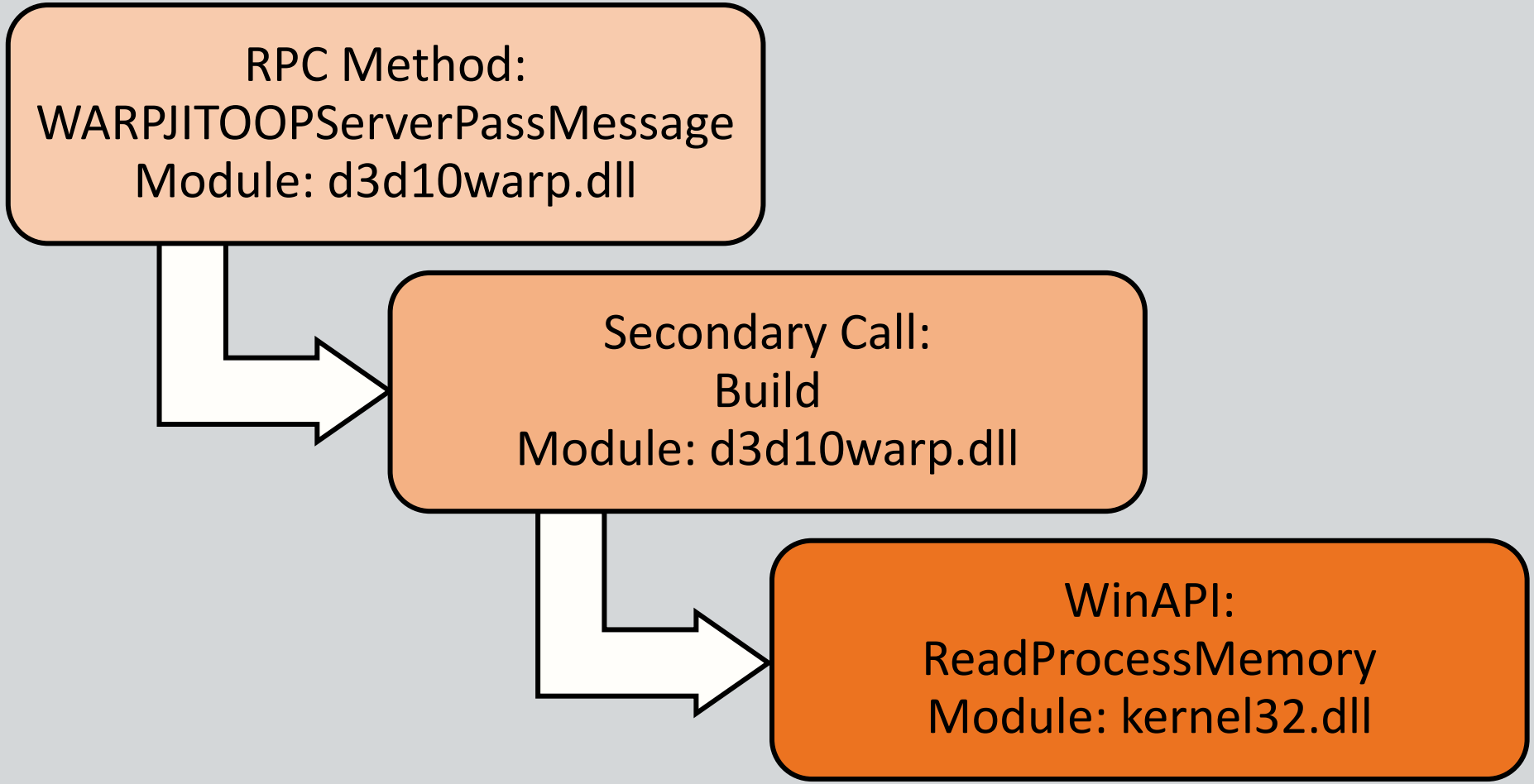
RPC Mapper

- RPC Mapper was developed by Sun Obziler
  - Aims to manipulate a benign service to perform a malicious actions
- RPC methods were extracted from every service on the machine
- Method was marked if it:
  - Leads to a WinAPI that can be abused (directly or after several hops)
  - Contains interesting keywords in its name

RPC Mapper



RPC Mapper



Utilizing RPC Mapper

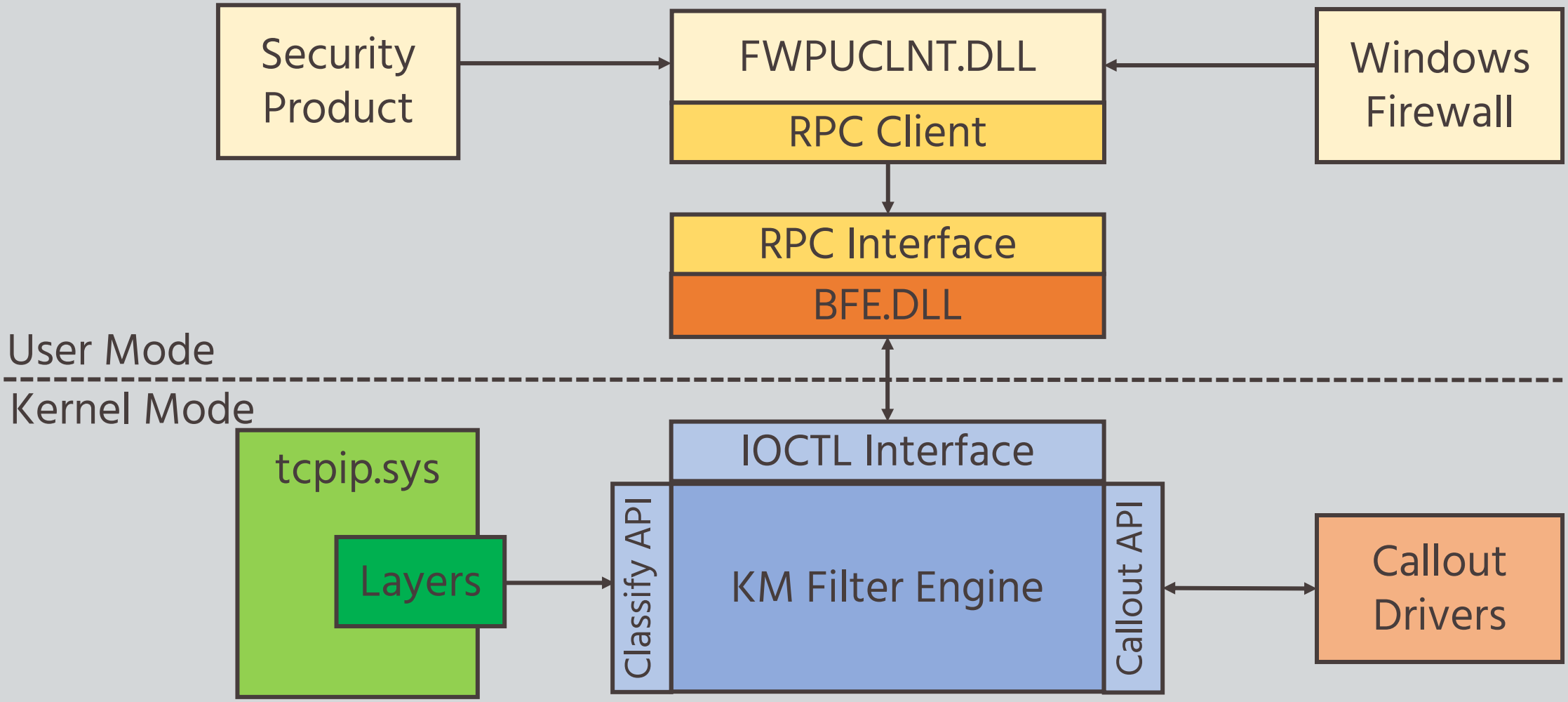
- One result of the RPC Mapper drew my attention to BFE.DLL
  - Exposes 194 RPC methods and calls interesting WinAPI
- The method `BfeRpcOpenToken` was picked based on its name
- This DLL is part of the Windows Filtering Platform (WFP)

```
PS C:\WINDOWS\system32> $RpcServer = Get-RpcServer C:\Windows\System32\BFE.DLL
-DbgHelpPath $Env:DbgHelp
foreach ($interface in $RpcServer) {
    foreach ($proc in $interface.procedures) {
        if ($proc.Name -like '*Token*') {$proc.Name} } }
BfeRpcOpenToken
BfeRpcCloseToken
```

Windows Filtering Platform

- Native platform in the Windows OS with a dedicated API
- Can block or allow network traffic at any layer in the system based on several fields
- Processes network traffic by hooking the network stack
    - Allows the development of security products
- Consists of several components:
    - Callout drivers
    - Filter Engine
    - Base Filtering Engine (BFE)

Windows Filtering Platform



https://learn.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform-architecture-overview

01 Intro
Technical background and known techniques

02 Reverse Engineering
Drivers and RPC components of WFP

03 Attacks
Duplicating tokens via Windows Filtering Platform

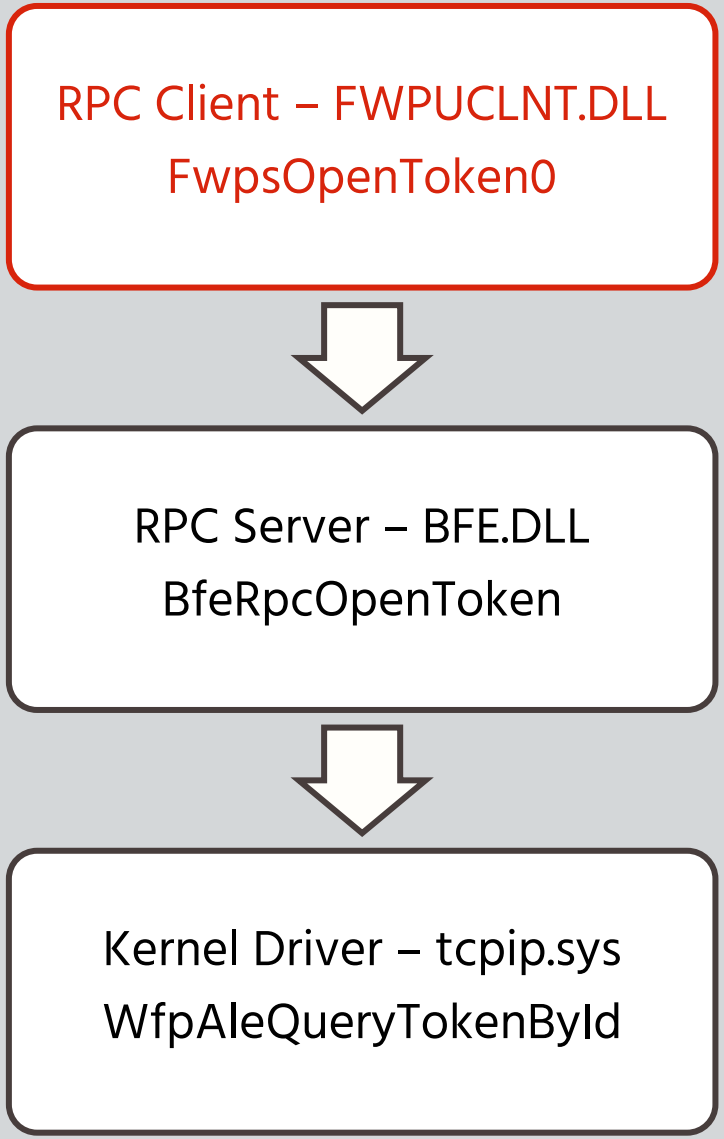
04 Conclusion
Detection methods and further research

FWPUCLNT.DLL

- FWPUCLNT.DLL exports documented functions that wrap RPC calls to BFE.DLL
- Engine handle is gained with `FwpmEngineOpen0`, but *`modifiedId`* is unclear

```
NTSTATUS FwpsOpenToken0(
    IN HANDLE engineHandle,
    IN LUID modifiedId,
    IN DWORD desiredAccess,
    OUT HANDLE* accessToken
);
```

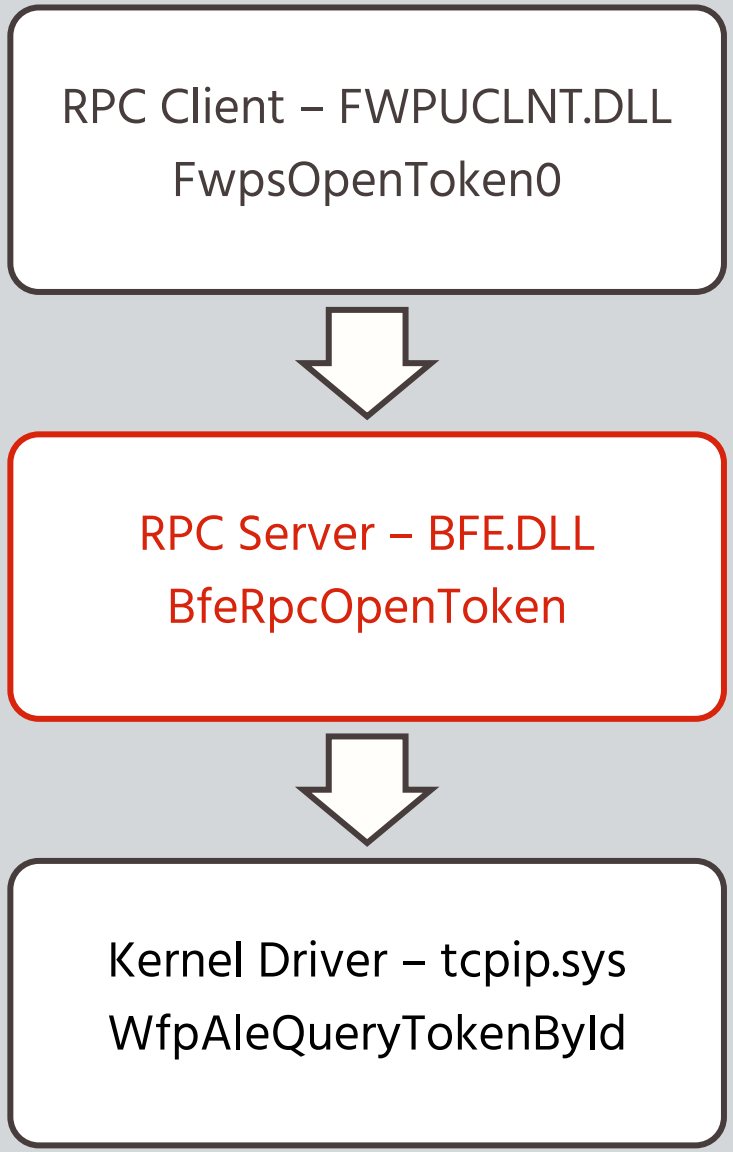
Token Query Flow



FWPUCLNT.DLL

```
DWORD FwpsOpenToken0(HANDLE engineHandle, LUID modifiedId, DWORD desiredAccess, HANDLE*
accessToken)
{
    NdrClientCall3(
        &pProxyInfo,
        0x93, //BfeRpcOpenToken
        0,
        *engineHandle,
        *(engineHandle + 1),
        modifiedId,
        &dwProcessId,
        &hSourceHandle);
    hSourceProcess = OpenProcess(PROCESS_DUP_HANDLE, 0, dwProcessId);
    DuplicateHandle(hSourceProcess, hSourceHandle, GetCurrentProcess(), accessToken,
        desiredAccess, 0, 0);
    [snip]
}
```

Token Query Flow

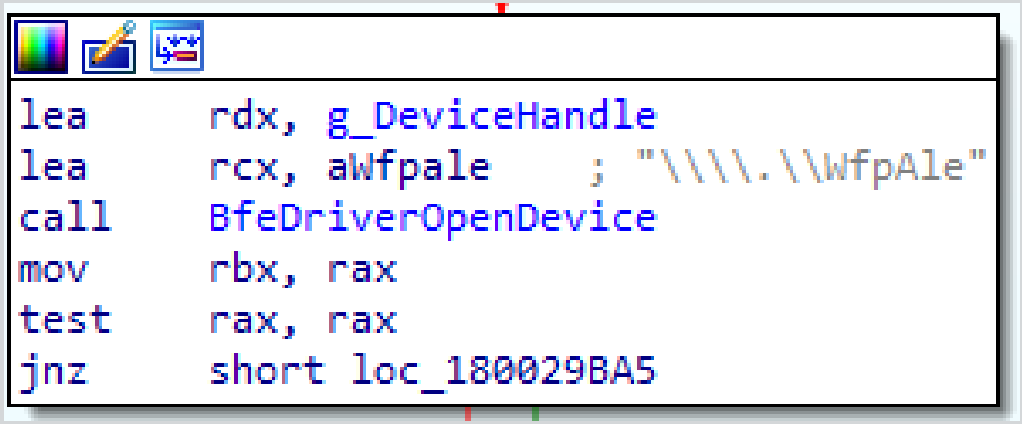


BFE.DLL

```
int BfeRpcOpenToken(RPC_BINDING_HANDLE bindingHandle, int engineHandle, LUID
modifiedId, DWORD* BfePid, HANDLE* HandleToDuplicate, HANDLE* HandleToClose)
{
    [snip]
    if (!BfeDriverTokenQuery(modifiedId, &tokenHandle))
    {
        CurrentProcess = GetCurrentProcess();
        *BfePid = GetProcessId(CurrentProcess);
        *HandleToDuplicate = tokenHandle;
    }
    [snip]
}
```

BFE.DLL

```
int BfeDriverTokenQuery(LUID ModifiedId, HANDLE* TokenHandle)
{
    LUID modifiedId = ModifiedId;
    return BfeDeviceIoControl(g_DeviceHandle, 0x124008, 8, &modifiedId, 8,
TokenHandle, 0, 0);
}
```

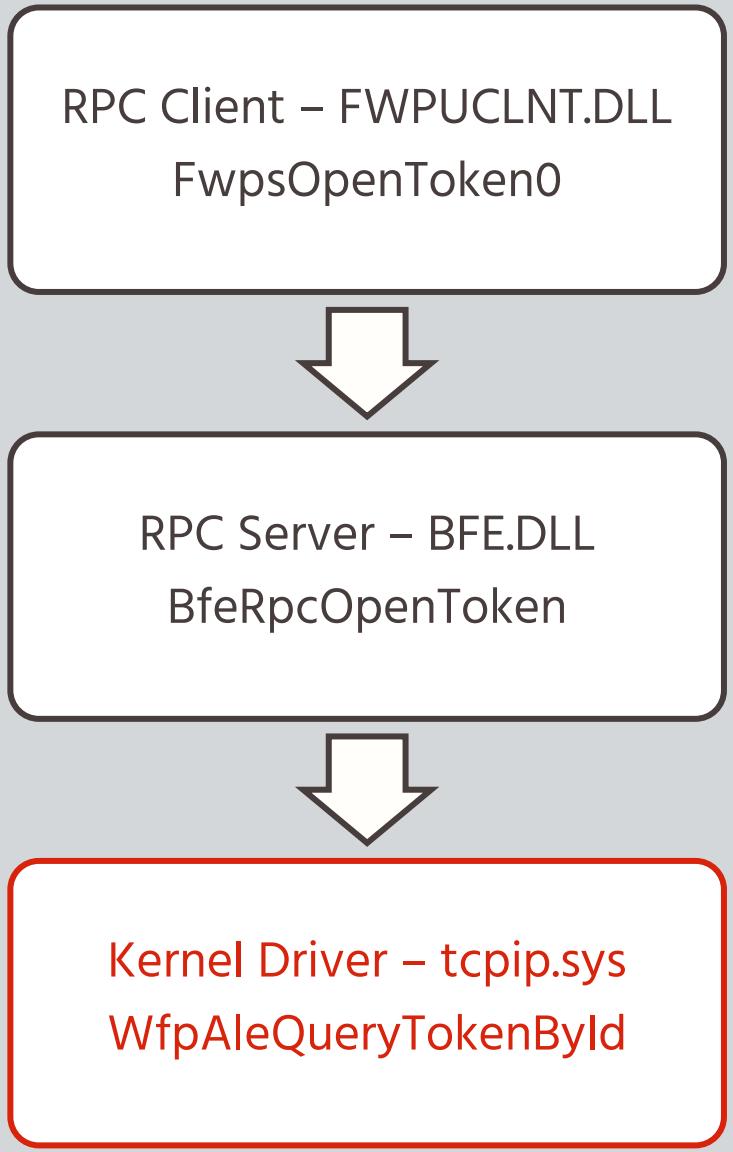


WfpAle

- Searching for this string under C:\Windows\System32\drivers leads to tcpip.sys
- The driver registers several additional devices: IPSECDOSP, NXTIPSEC, eQoS
- The function that handles IOCTL for WfpAle is `WfpAleDispatchControl`

Control Code	Tcpip Function	BFE Function
0x124008	WfpAleQueryTokenById	BfeRpcOpenToken
0x124018	WfpAleProcessEndpointPropertiesQuery	BfeRpcAleEndpointGetById
0x12401E	WfpAleProcessEndpointEnumIoctl	BfeRpcAleEndpointCreateEnumHandle
0x124020	sets tcpip!gMaxInboundSeqRanges global variable	BfeRpcEngineSetOption
0x128000	WfpAleProcessTokenReference	BfeDriverTokenAddRef
0x128004	WfpAleReleaseTokenInformationById	BfeDriverTokenRelease
0x128010	WfpAleProcessExplicitCredentialQuery	BfeRpcAleExplicitCredentialsQuery

Token Query Flow



TCPIP.SYS

```
int WfpAleQueryTokenById(LUID* pLuid, ACCESS_MASK DesiredAccess, TOKEN_TYPE TokenType,
void** OutputBuffer) {
    PTOKEN_ENTRY tokenEntry = 0;
    HANDLE NewTokenHandle = 0;
    LUID luid = *pLuid;
    if (!WfpAleAcquireTokenInformation(&luid, &tokenEntry)) {
        duplicationStatus = ZwDuplicateToken(
            tokenEntry->TokenValue,
            DesiredAccess, // TOKEN_DUPLICATE
            &ObjectAttributes,
            0,
            TokenType,     // TokenPrimary
            &NewTokenHandle);
        if (!duplicationStatus)
            *OutputBuffer = NewTokenHandle;
    }
    return status;
}
```

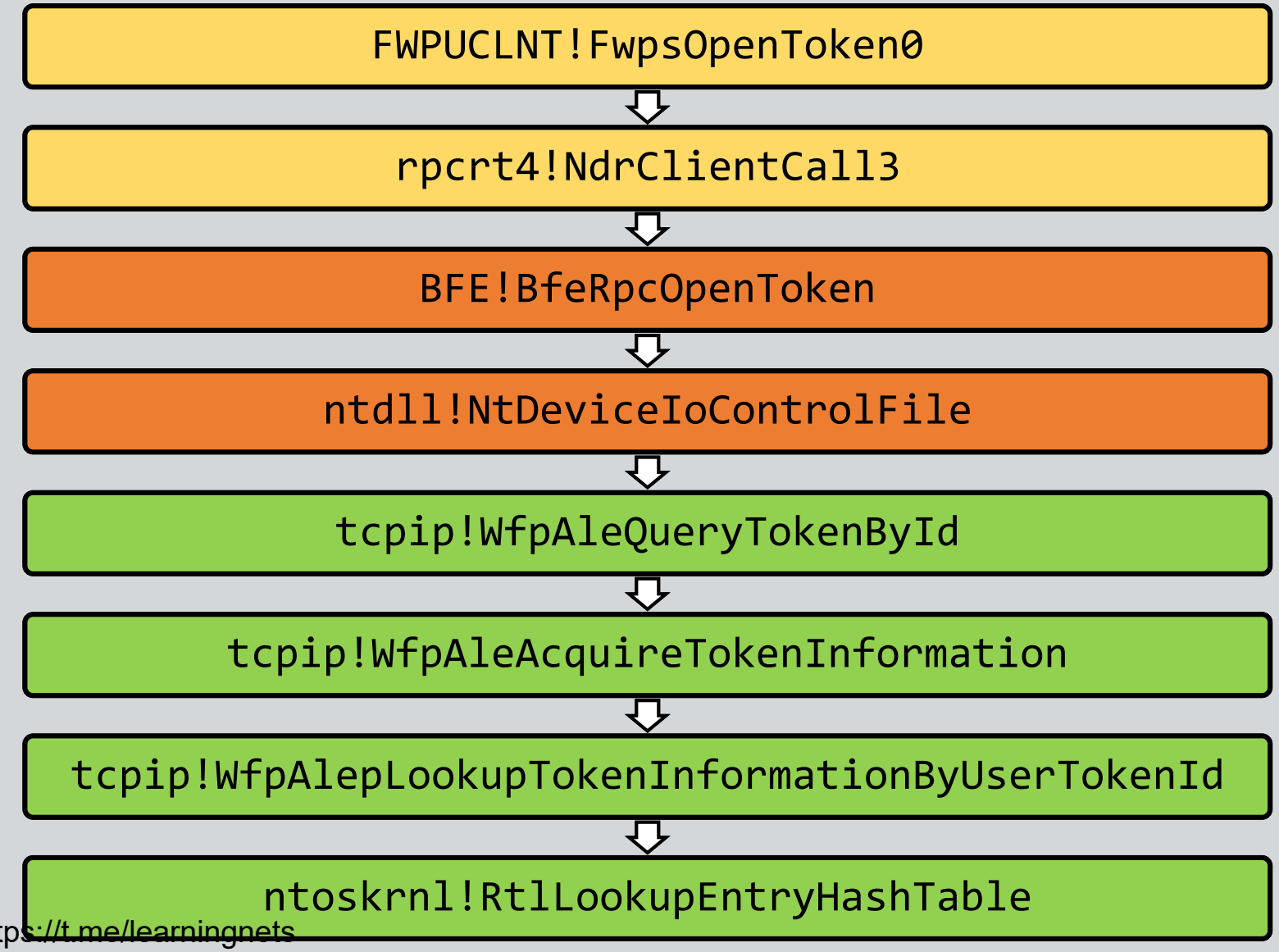
TCPIP.SYS

```
int WfpAleAcquireTokenInformation(LUID* Luid, PTOKEN_ENTRY* pTokenEntry)
{
    int hash = 0;
    *pTokenEntry = 0;
    WfpUpdateHash(Luid, 8, &hash);
    hash = (hash << gAleNumHashEntryBits) | 7;
    return  WfpAlepLookupTokenInformationByUserTokenId(Luid, &hash, pTokenEntry);
}
```

TCPIP.SYS

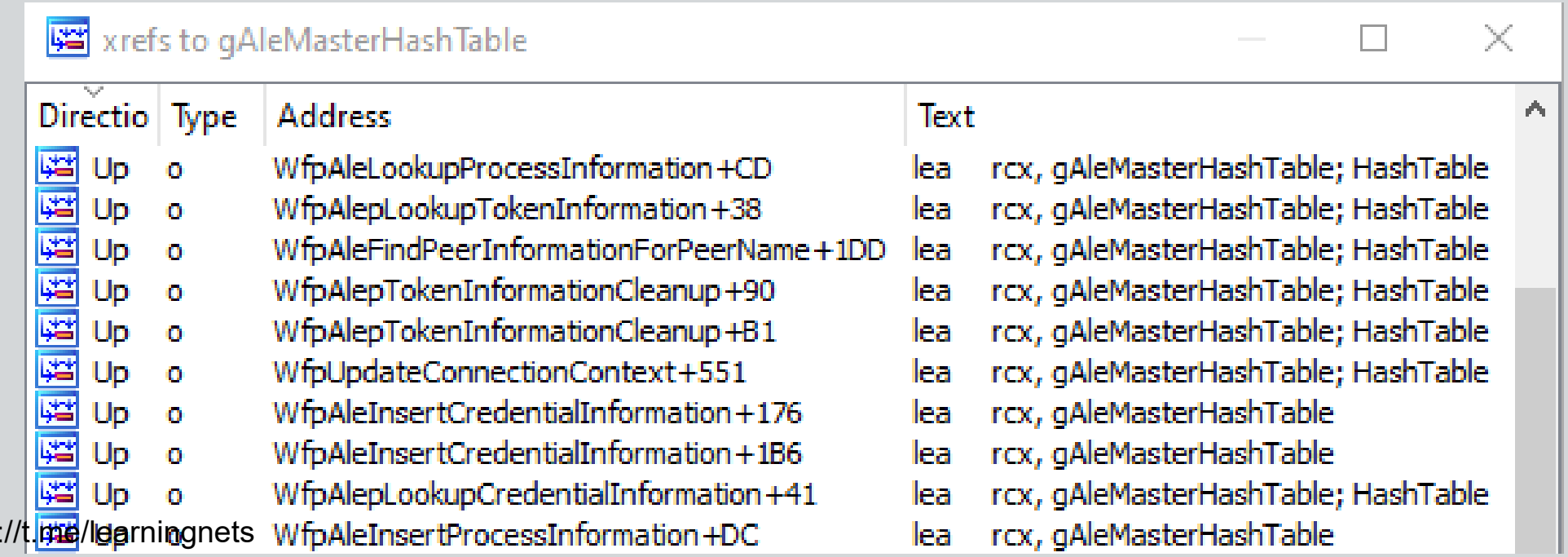
```
int WfpAlepLookupTokenInformationByUserTokenId(LUID* Luid, ULONG_PTR* HashPtr,
PTOKEN_ENTRY* pTokenEntry)
{
    [snip]
    PRTL_DYNAMIC_HASH_TABLE_ENTRY i;
    RTL_DYNAMIC_HASH_TABLE_CONTEXT Context;
    for (i = RtlLookupEntryHashTable(&gAleMasterHashTable, *HashPtr, &Context);
        i;
        i = RtlGetNextEntryHashTable(&gAleMasterHashTable, &Context))
    {
        PTOKEN_ENTRY currentEntry = CONTAINING_RECORD(i, TOKEN_ENTRY, hashTableEntry);
        if (currentEntry->Luid.LowPart == Luid->LowPart &&
            currentEntry->Luid.HighPart == Luid->HighPart)
        {
            *pTokenEntry = currentEntry;
        }
    }
}
```

Token Query Recap



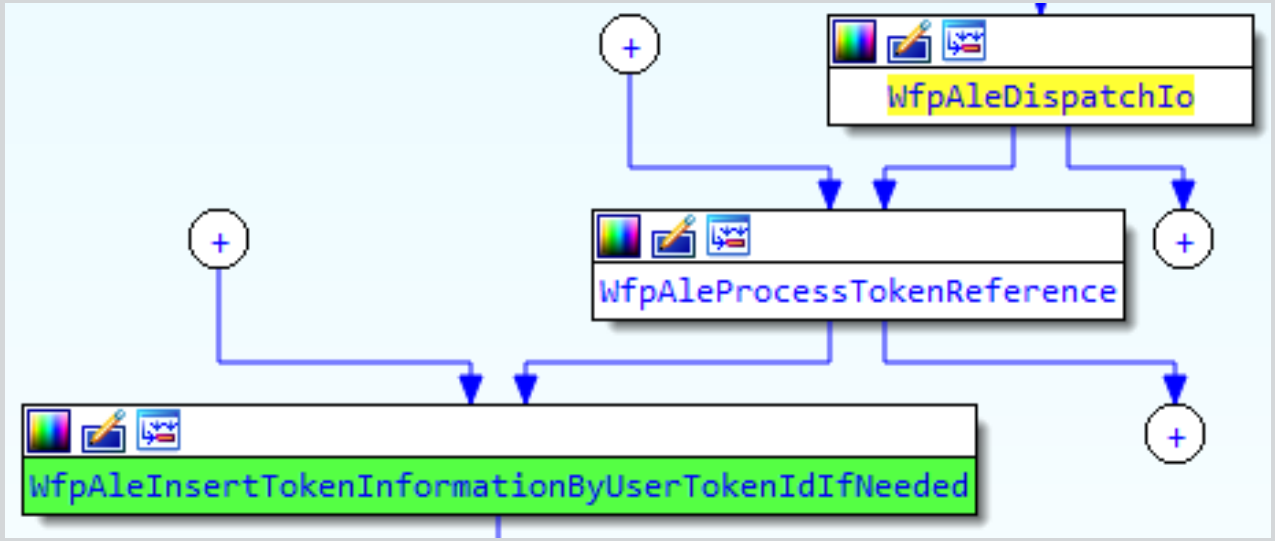
gAleMasterHashTable

- The table is used by over 30 functions and stores various undocumented structs
- Tokens is added by calling `WfpAleInsertTokenInformationByUserTokenIdIfNeeded`
- The function isn't called during the boot process
- The table doesn't contain token entries



Token Insertion

- The insertion function is called by `WfpAleProcessTokenReference (IOCTL 0x128000)`
- `BfeDriverTokenAddRef` sends this IOCTL, but isn't exposed through RPC
- Triggering this device IO request will insert a token to the table



Token Insertion

```
int WfpAleProcessTokenReference(PSET_TOKEN_STRUCT InputBuffer, LUID* OutputBuffer)
{
    void* Pid = InputBuffer->Pid; // BAD
    HANDLE Token = InputBuffer->Token;
    CLIENT_ID ClientId = { Pid,0 };

    ZwOpenProcess(&ProcessHandle, PROCESS_QUERY_INFORMATION, &ObjectAttributes, &ClientId);
    ObReferenceObjectByHandle(ProcessHandle, PROCESS_QUERY_INFORMATION, 0, 0, &pEPROCESS, 0);
    KeStackAttachProcess(pEPROCESS, &ApcState);
    ZwDuplicateToken(Token, TOKEN_ALL_ACCESS, &ObjectAttributes, 0, TokenPrimary, &newTokenHandle);

    if (!WfpAleCaptureTokenIdByHandle(newTokenHandle, luid)) {
        if (!WfpAleAcquireTokenInformationFromToken(0, newTokenHandle, luid, &newTokenEntry, v13)) {
            if (!WfpAleInsertTokenInformationByUserTokenIdIfNeeded(newTokenEntry))
                *OutputBuffer = newTokenEntry->Luid;
        }
    }
}
```

01 Intro
Technical background and known techniques

02 Reverse Engineering
Drivers and RPC components of WFP

03 Attacks
Duplicating tokens via Windows Filtering Platform

04 Conclusion
Detection methods and further research

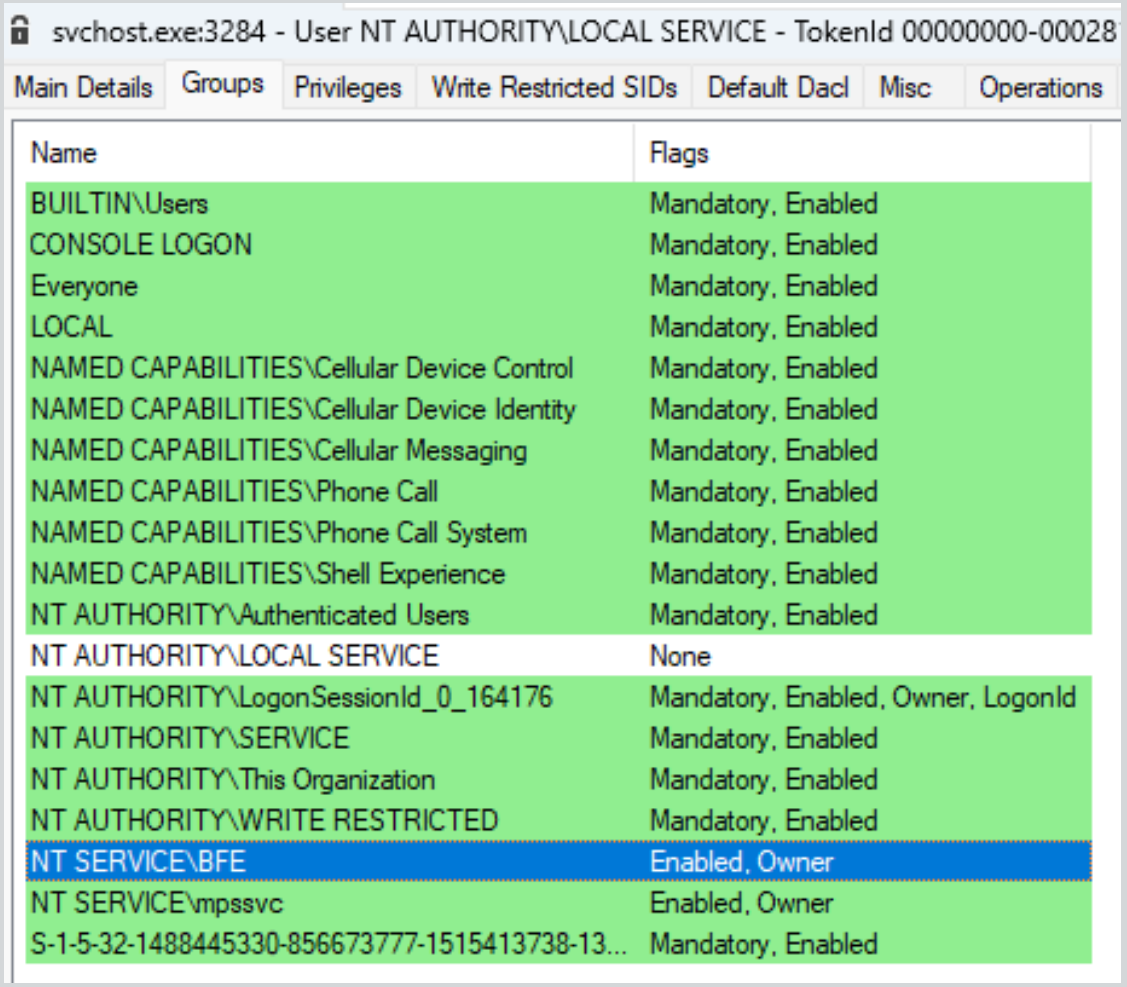
Accessing WfpAle

- Sending IOCTL to tcpip.sys will bypass limitations set by the BFE service
- Opening a handle to the device *WfpAle* results in ERROR_ACCESS_DENIED
- The device is created with a security descriptor by tcpip.sys

Accessing WfpAle

```
int WfpAleRegisterDevice(PDRIVER_OBJECT DriverObject, PDEVICE_OBJECT* ppDeviceObject)
{
    UNICODE_STRING DeviceName;
    UNICODE_STRING DestinationString;
    RtlInitUnicodeString(&DeviceName, L"\\Device\\WfpAle");
    NTSTATUS error = IoCreateDevice(DriverObject, 0, &DeviceName, FILE_DEVICE_NETWORK,
        FILE_DEVICE_SECURE_OPEN, 0, ppDeviceObject);
    if (!error) {
        PDEVICE_OBJECT pDeviceObject = *ppDeviceObject;
        deviceCreated = TRUE;
        if (!WfpAllowBfeGenericAll(pDeviceObject)) {
            RtlInitUnicodeString(&DestinationString, L"\\DosDevices\\WfpAle");
            if (!IoCreateSymbolicLink(&DestinationString, &DeviceName)) {
                symbolsLinkCreated = TRUE;
            }
        }
    }
    return error;
}
```

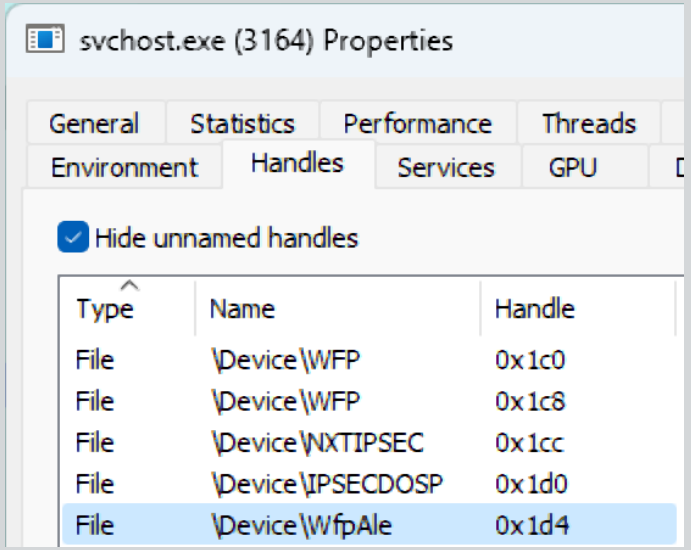
Accessing WfpAle



Accessing WfpAle

- The BFE service has an open handle to the device
- The device handle can be duplicated into another process
- The security descriptor doesn't block the duplication of the handle
- Prerequisites:
  - Debug privileges
  - Handle to the BFE service with

    `PROCESS_DUP_HANDLE | PROCESS_QUERY_INFORMATION`



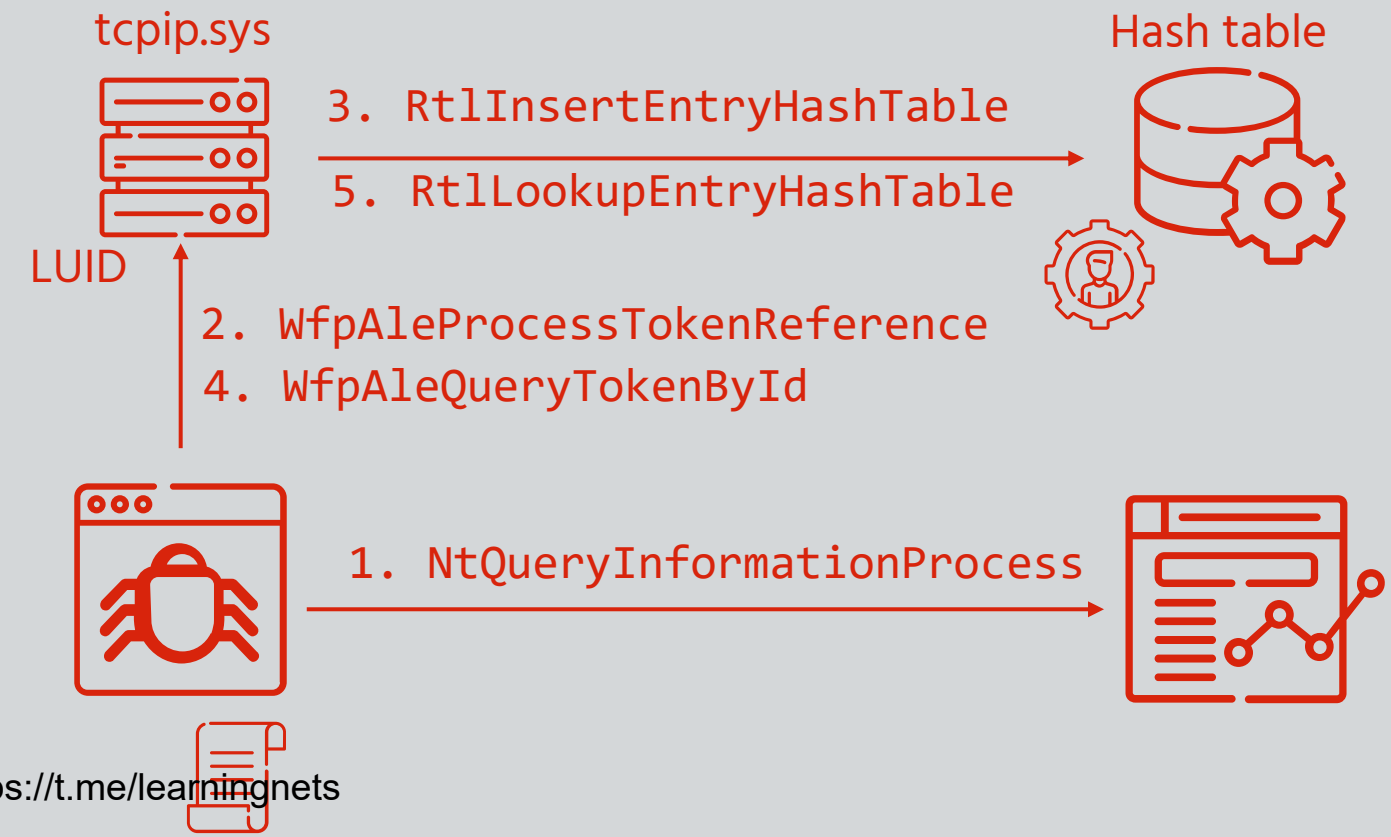
Bypassing Detection with WfpAle

- Calls to `DuplicateHandle` / `DuplicateToken` might be detected
- `FwpsOpenToken0` will trigger these detections
- `DuplicateHandle` duplicates the token from the BFE service to the current process
- The handle will only have the permission `TOKEN_DUPLICATE`
- `DuplicateToken` will need to be called after the handle is retrieved by `FwpsOpenToken0`

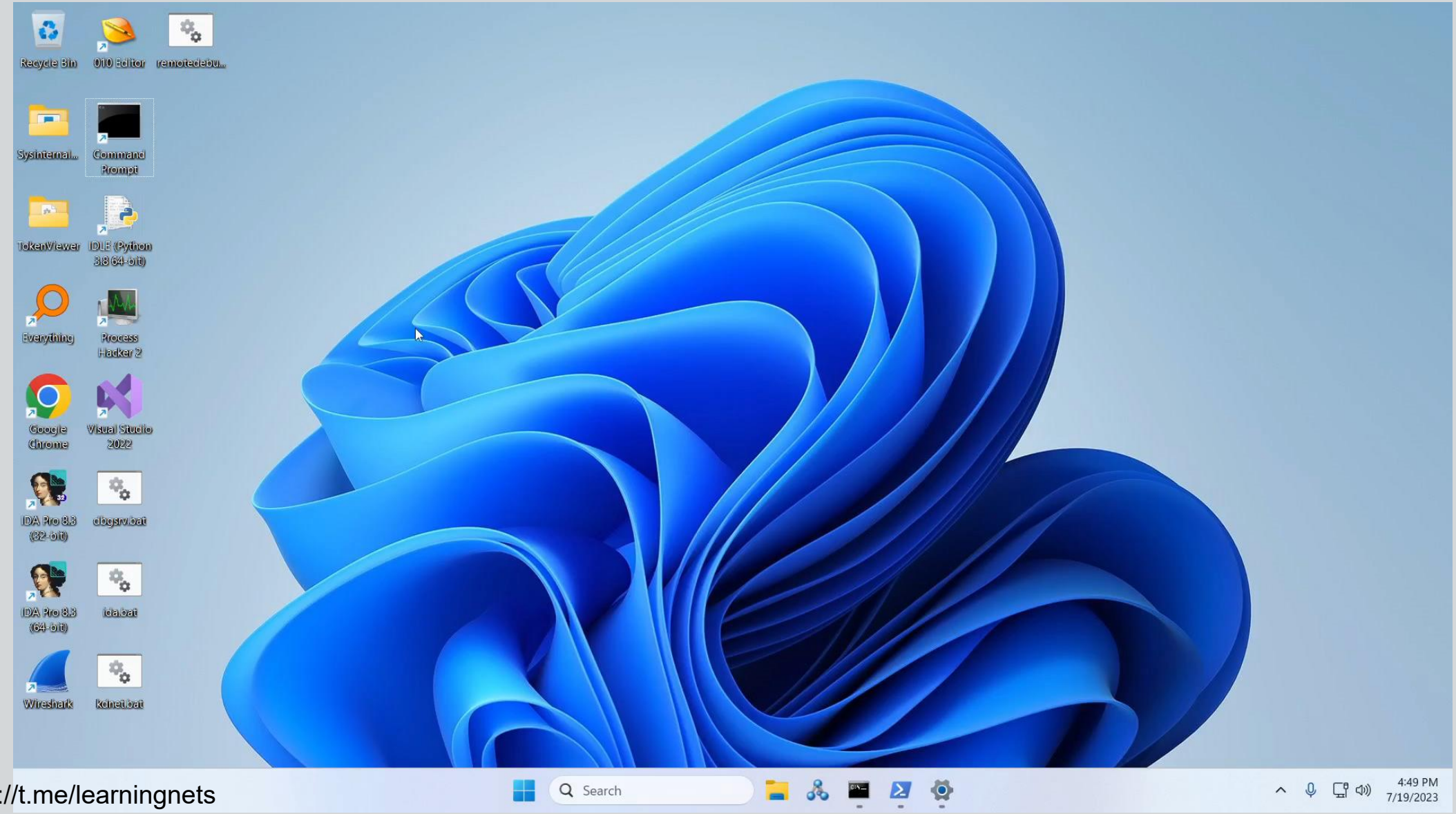
- Sending the device IO requests directly will bypass these detection
- Tcpip will duplicate the token for the current process instead of the BFE service
- `DuplicateHandle` changes the permissions from `TOKEN_DUPLICATE` to `TOKEN_ALL_ACCESS`

Attack #1 - Duplicating tokens via WFP

- The goal is to store a token in the hash table and then retrieve it
- The call to `DuplicateHandle` can be replaced with `WfpAleProcessTokenReference`



Attack #1 - Demo



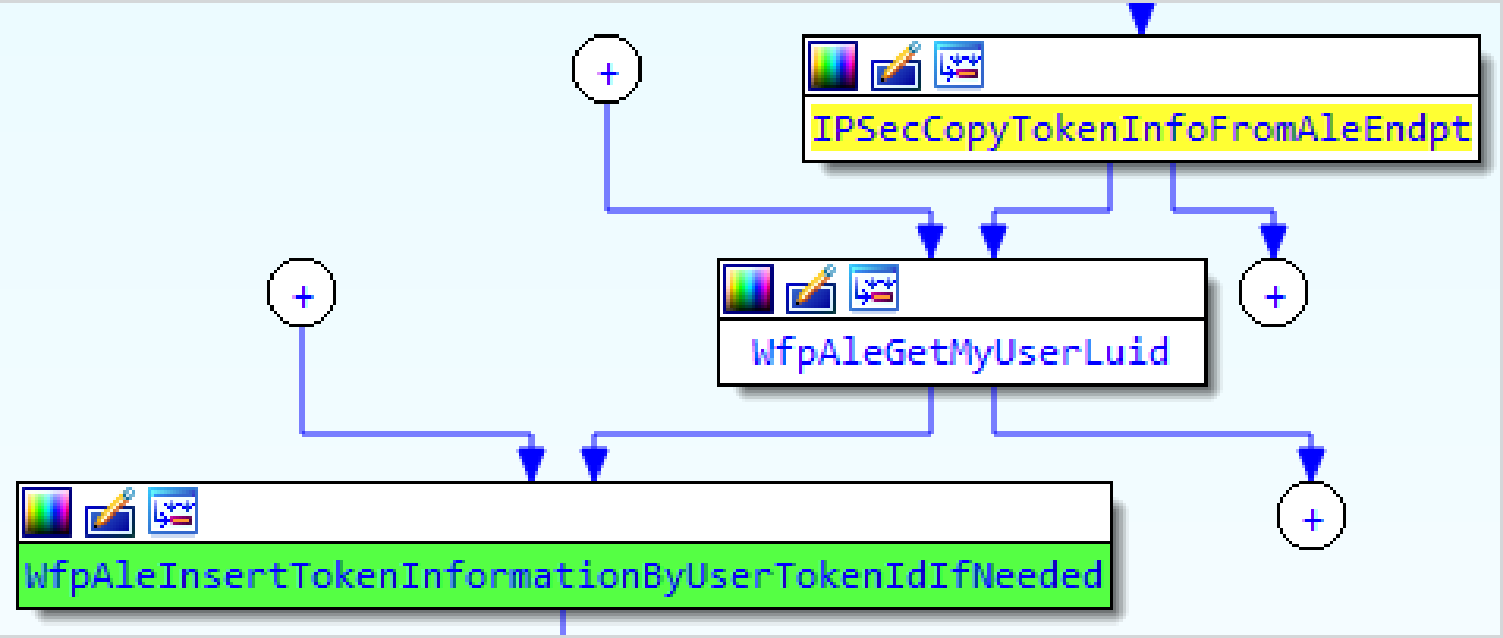
Attack #1 - Conclusion

Advantages:

- Handle to a token isn't duplicated from one process to another
- Duplicating a handle to *WfpAle* device is not suspicious
- The token of several services can be duplicated only by this method:
  - LSM
  - Winmgmt
  - Schedule

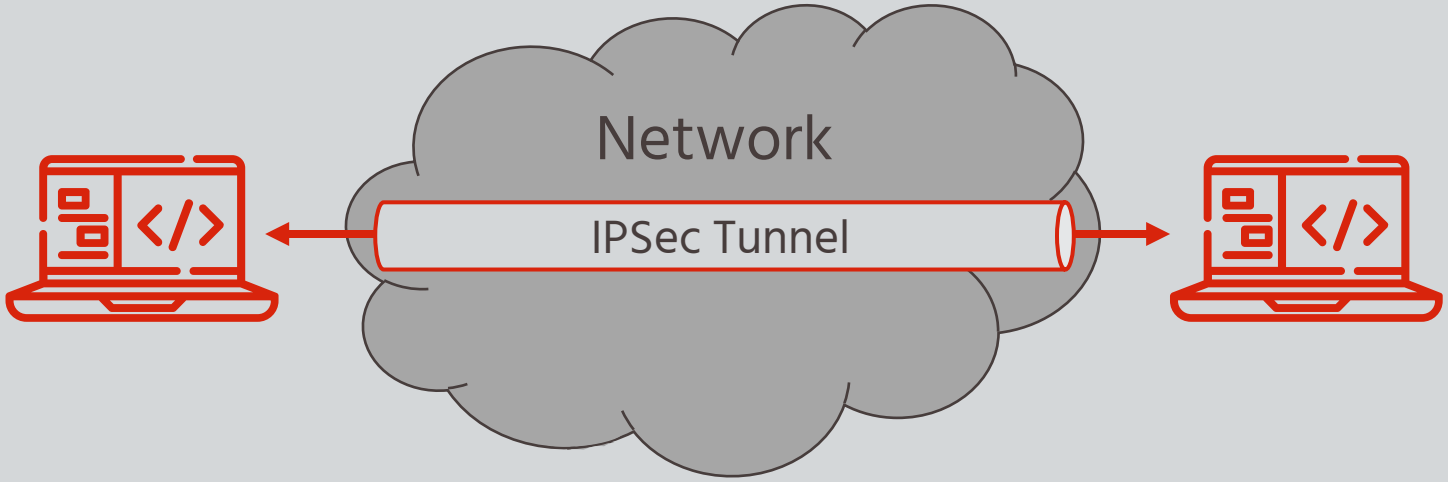
Attack #1 - Conclusion

- Can WFP be abused in a way that isn't based on a known technique?
- Additional cross-references to the token insertion function were found
- Maybe using IPSec will insert a token?



Internet Protocol Security

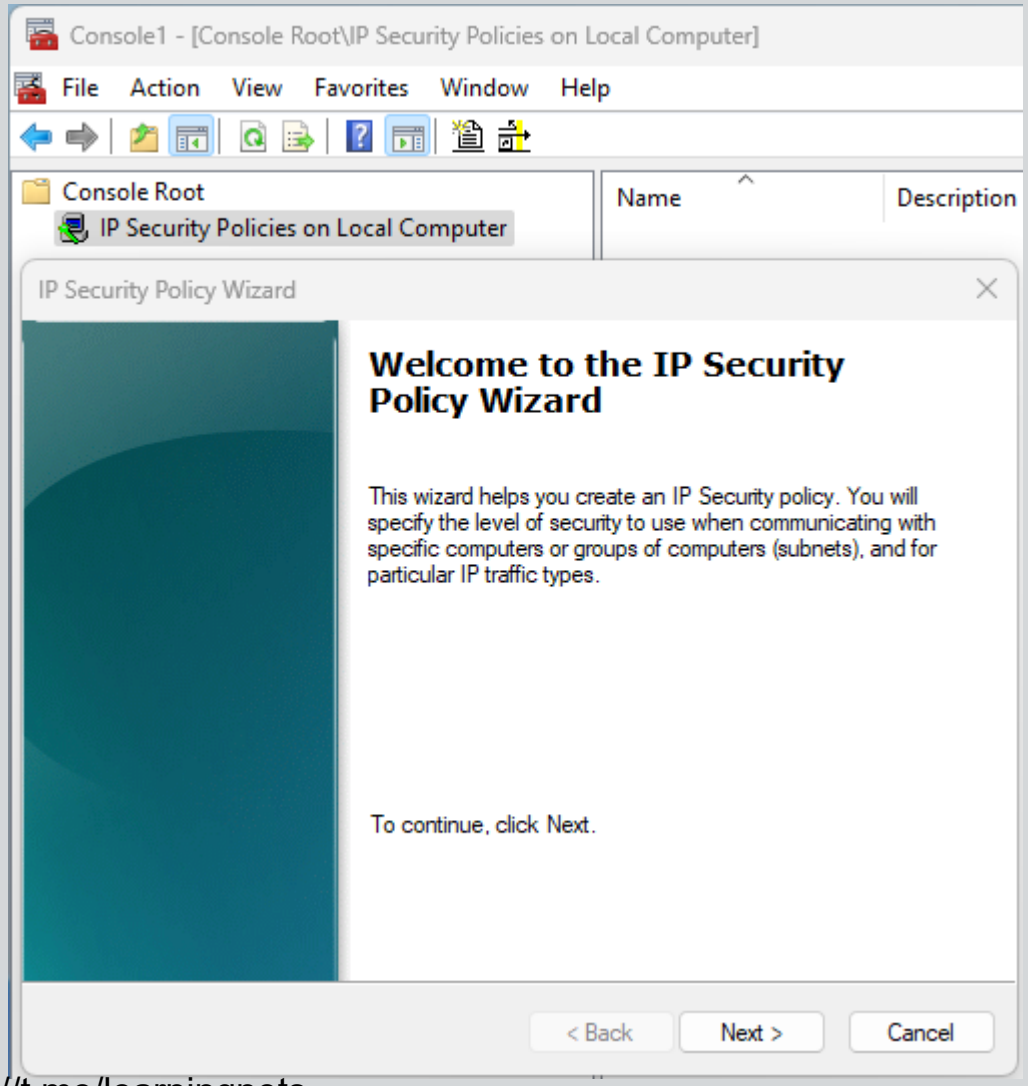
- Set of protocols that ensure secured and private communication
- Integrating it at the Internet layer provides security for almost all TCP/IP protocols
- The authentication and encryption process protects against the following:
  - Network sniffers
  - Data modification
  - Identity spoofing
  - Denial of service



Internet Protocol Security

- Secure connection is set up between two machines before exchanging data by the Internet Key Exchange (IKE) service
- Authentication options: Kerberos V5, certificates, or a pre-shared key
- AuthIP expands the IKE with more authentication options
- IPSec policy is configured via MMC or the WFP API

Internet Protocol Security



IPsec Functions

Article • 08/20/2020 • 4 contributors

Feedback

The Windows Filtering Platform (WFP) functions that interact with Internet Protocol Security (IPsec) are as follows.

- FwpmIPsecTunnelAdd:
  - FwpmIPsecTunnelAdd0 (Windows Vista)
  - FwpmIPsecTunnelAdd1 (Windows 7)
  - FwpmIPsecTunnelAdd2 (Windows 8)
- FwpmIPsecTunnelDeleteByKey0
- IPSEC_KEY_MANAGER_KEY_DICTATION_CHECK0
- IPSEC_KEY_MANAGER_DICTATE_KEY0
- IPSEC_KEY_MANAGER_NOTIFY_KEY0
- IPSEC_SA_CONTEXT_CALLBACK0
- IPsecDospGetSecurityInfo0
- IPsecDospGetStatistics0
- IPsecDospSetSecurityInfo0
- IPsecDospStateCreateEnumHandle0

Configuring IPSec Policy

- IPSec connection needed to be established
- Example from MSDN was used
- The IKE provider didn't return an error code
- `FwpmGetAppIdFromFileName0` was reverse-engineered
- Breakpoint on the insertion function was set
- IPSec policy was configured between two machines

```
DWORD ConfigureIPsecTunnelMode(
__in HANDLE engine,
__in PCWSTR policyName,
__in_opt const GUID* providerKey,
__in const SOCKADDR* localAddr,
__in const SOCKADDR* remoteAddr,
__in const FWP_BYTE_BLOB* presharedKey
)
```

https://learn.microsoft.com/en-us/windows/win32/fwp/using-tunnel-mode

Configuring IPSec Policy

- **Establishing IPSec connections results in inserting tokens to gAleMasterHashTable**
- Metadata is kept on IPSec connections
- This metadata includes the token of the process
- It is probably to impersonate the process

What is AuthIP

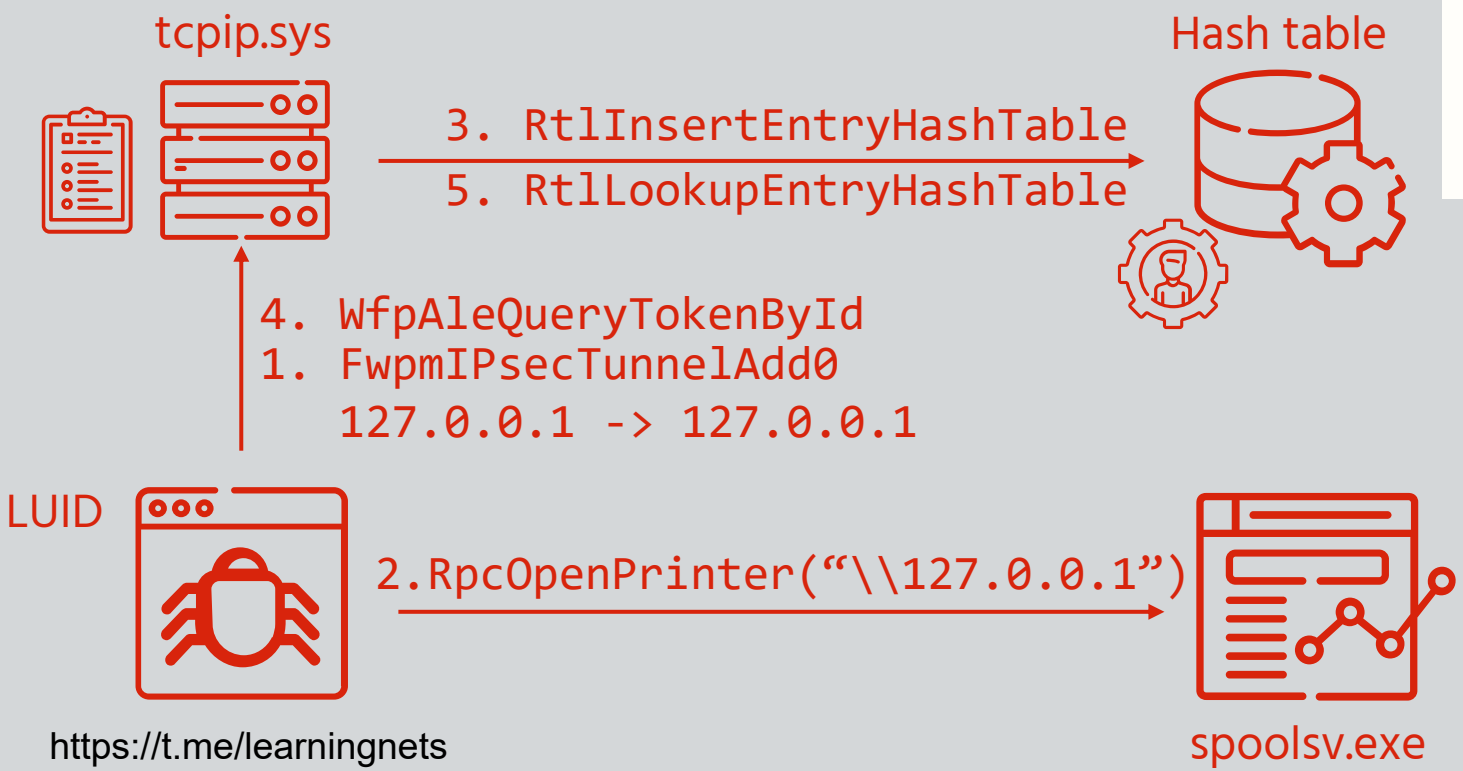
Authenticated Internet Protocol (AuthIP) is a new key exchange protocol that expands IKE as follows.
[snip]
AuthIP can be used with secure sockets to implement application-based IPsec secured traffic. It provides:

- Per-socket authentication and encryption. See WSASetSocketSecurity for more information.
- Client impersonation. (IPsec impersonates the security context under which the socket is created.)
- Inbound and outbound peer name validation. See WSASetSocketPeerTargetName for more information.

https://learn.microsoft.com/en-us/windows/win32/fwp/ipsec-configuration#what-is-authip

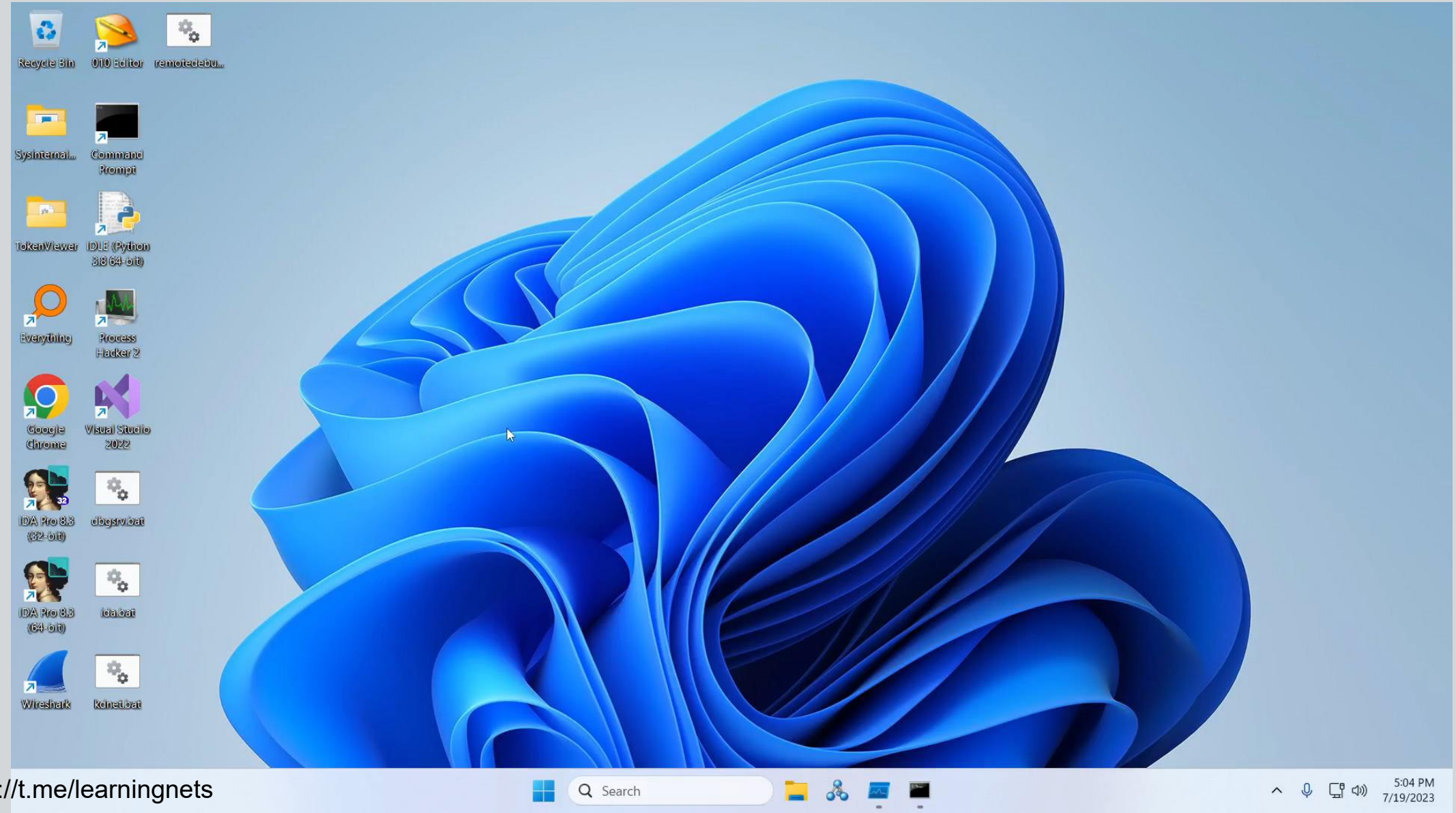
Attack #2 – Trigger IPSec Connection

The goal is to manipulate a service to create a socket that matches the IPSec policy:




```
DWORD RpcOpenPrinter(
[in, string, unique] STRING_HANDLE
pPrinterName,
[out] PRINTER_HANDLE* pHandle,
[in, string, unique] wchar_t* pDatatype,
[in] DEVMODE_CONTAINER* pDevModeContainer,
[in] DWORD AccessRequired
);
```

Attack #2 - Demo



Attack #2 - Conclusion

Advantages:

- Configuring an IPSec policy on a machine is a legitimate action
- No service communicating on the localhost is affected by the IPSec policy
- EDR solutions will most likely ignore localhost communication
- There is no need to call `WfpAleProcessTokenReference` directly
- The interaction with Spooler isn't suspicious

Can we manipulate more services to gain other tokens?

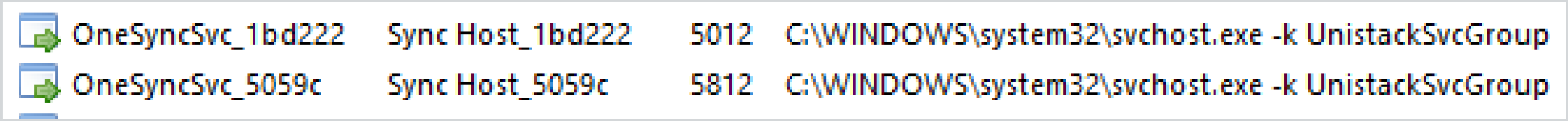
Attack #3 – Manipulate User Service

- The goal is to steal the token of another user logged on the machine
  - Can lead to lateral movement in the domain
- RPC servers running as logged on users were searched for

```
PS C:\WINDOWS\system32> $DomainAdminProcs = Get-Process -IncludeUserName |
where {$_.UserName -eq 'TEST\Administrator'}
$RpcServers = New-Object -TypeName System.Collections.ArrayList
foreach ($proc in $DomainAdminProcs) {
    foreach ($interface in Get-RpcServer -ProcessId $proc.id) {
        [void]$RpcServers.Add($interface.Name) } }
$RpcServers | sort -unique
aphostservice.dll
d3d10warp.dll
modernexecserver.dll
PlaySndSrv.dll
SyncController.dll
...
```

Attack #3 – OneSyncSvc

- Every session launches the OneSyncSvc service with the user's permissions



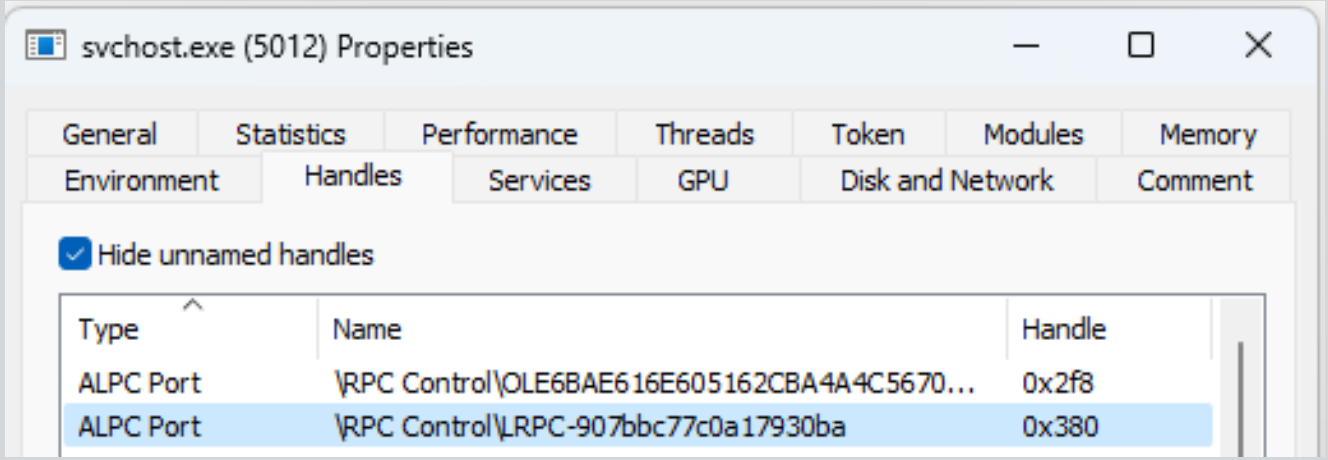
- This service loads SyncController.dll
- The DLL implements an undocumented RPC method

```
Int AccountsMgmtRpcDiscoverExchangeServerAuthType(
[in, string] const wchar_t* ServerAddress,
[out] int* IntOut);
```

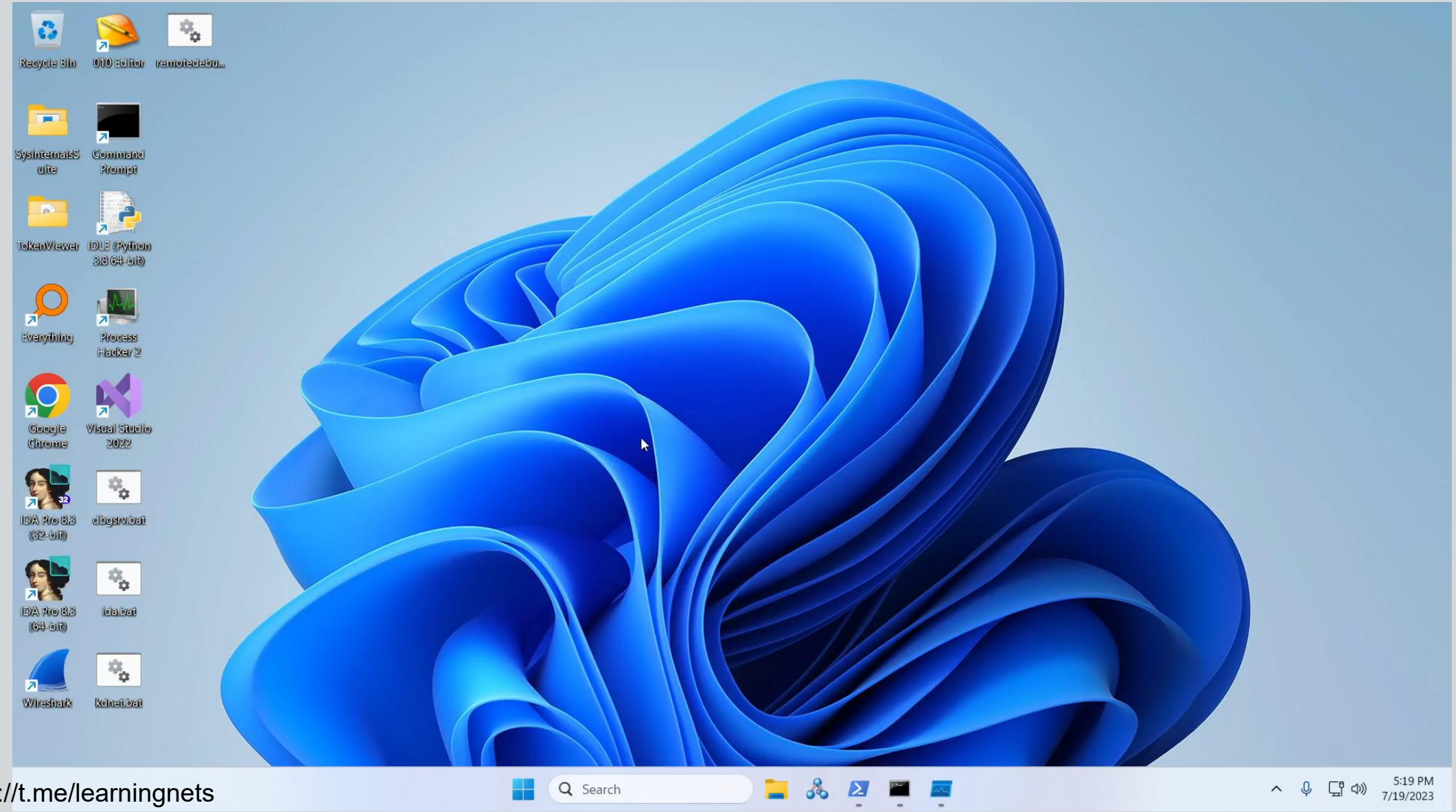
- This method receives an SSH-like string: *username@127.0.0.1*

Attack #3 - Manipulate User Service

1. Configure an IPSec policy for connections to the localhost with pre-shared key
2. Enum services and find the pid of OneSyncSvc running in the target session
3. Find the handle to the ALPC port
4. Create a listener thread for 127.0.0.1:443
5. Call `AccountsMgmtRpcDiscoverExchangeServerAuthType`
6. Bruteforce the LUID of the new token while OneSyncSvc is connected to the socket
7. Launch process with the new token



Attack #3 - Demo



Attack #3 - Conclusion

Advantages:

- No other tool abused OneSyncSvc before. Shouldn’t trigger EDR solutions
- Can achieve lateral movement

01 Intro
Technical background and known techniques

02 Reverse Engineering
Drivers and RPC components of WFP

03 Attacks
Duplicating tokens via Windows Filtering Platform

04 Conclusion
Detection methods and further research

Reporting Findings

- This research was reported to Microsoft Security Response Center
- The response of Microsoft is:

"We determined that this behavior is considered to be by design."

WinAPI Detection

- Configuring new IPSec policies
- RPC calls to Spooler / OneSyncSvc
- Brute force the LUID value
- **Device IO requests sent to WfpAle**

Logs Detection

- WFP generates logs for events
- Most logs are about packets drops, or failures during the key exchange process
- Logging allowed packets can be configured explicitly, but not recommended
- Logs about packets sent during the attack aren’t informative

Logs Detection

```
FilterId         : 67348
LayerId          : 48
ReauthReason     : 0
OriginalProfile  : None
CurrentProfile   : None
MsFwpDirection   : 0
IsLoopback       : False
Type             : ClassifyAllow
Flags            : IpProtocolSet, LocalAddrSet,
RemoteAddrSet, LocalPortSet, RemotePortSet,
AppIdSet, UserIdSet,
                   IpVersionSet, PackageIdSet
Timestamp        : 3/23/2023 10:39:59 AM
IPProtocol       : Tcp
LocalEndpoint    : 127.0.0.1:49841
RemoteEndpoint   : 127.0.0.1:135
ScopeId          : 0
AppId            :
\device\harddiskvolume3\windows\system32\spoolsv.exe
UserId           : S-1-5-21-3059025837-677458256-
3910875200-1001
AddressFamily    : Inet
PackageSid       : S-1-0-0
```

```
Name        : AppContainerLoopback
Action Type: Permit
Key         : f8d36c2f-87a1-4c73-8936-fdbe29c19ab2
Id          : 67348
Description: This filter allows outbound AppContainer loopback
traffic
Layer       : FWPM_LAYER_ALE_AUTH_CONNECT_V4
Sub Layer   : {ffe221c3-92a8-4564-a59f-dafb70756020}
Flags       : 4100
Weight      : 18446744073709551615
Conditions :
FieldKeyName          MatchType   Value
------------          ---------   -----
FWPM_CONDITION_FLAGS FlagsAllSet IsLoopback
```

Further Research – tcpip.sys

- The driver tcpip.sys creates several devices that expose several functionalities
- Sending device IO requests to them can uncover new attack surfaces

Device	Control Code	Tcpip Function
IPSECDOSP	0x124004	IdpProcessQueryStatsIoctl
	0x124002	IdpProcessEnumStateIoctl
NXTIPSEC	0x128028	IPsecSetS2STunnelInterfaceHndlr
	0x12801C	IPSecNotifyStatusHndlr
	0x128018	IPSecUpdateSaInfoHndlr
WFP	0x12803C	IoctlKfdResetState
	0x124050	IoctlKfdSetBfeEngineSd
	0x128004	IoctlKfdAddIndex

Further Research – tcpip.sys

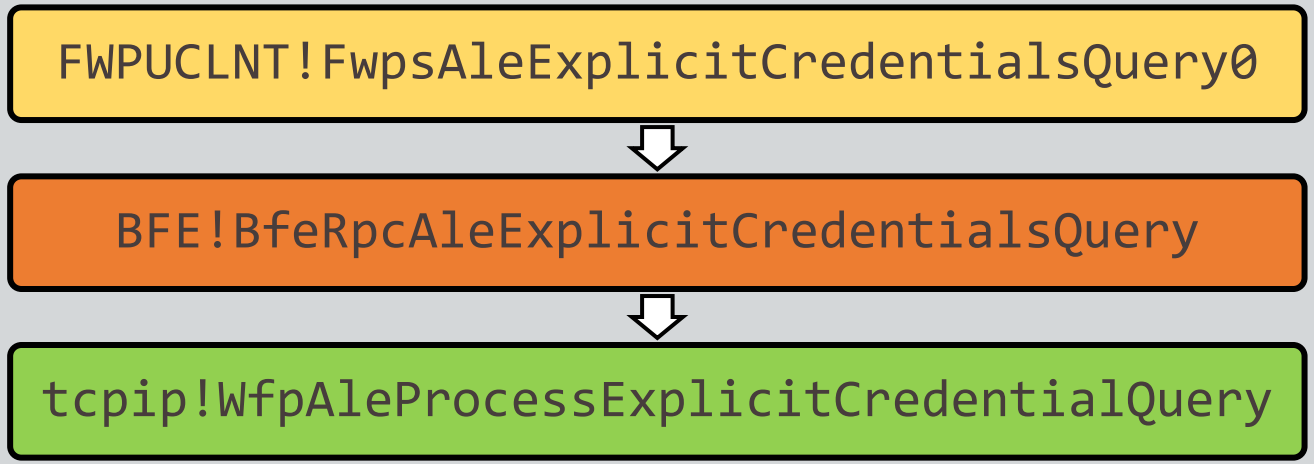
- gAleMasterHashTable is used for managing data about various operations
- Storing and retrieving entries from this table can be beneficial for attackers

xrefs to gAleMasterHashTable

Directio	Type	Address	Text
Up	o	WfpAleLookupProcessInformation+CD	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAlepLookupTokenInformation+38	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAleFindPeerInformationForPeerName+1DD	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAlepTokenInformationCleanup+90	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAlepTokenInformationCleanup+B1	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpUpdateConnectionContext+551	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAleInsertCredentialInformation+176	lea rcx, gAleMasterHashTable
Up	o	WfpAleInsertCredentialInformation+1B6	lea rcx, gAleMasterHashTable
Up	o	WfpAlepLookupCredentialInformation+41	lea rcx, gAleMasterHashTable; HashTable
Up	o	WfpAleInsertProcessInformation+DC	lea rcx, gAleMasterHashTable

Further Research – Explicit Credentials

- tcpip.sys holds data labeled as “Process Explicit Credentials”
- Retrieving this data is exposed through RPC:



Further Research – Explicit Credentials

- Access check is performed on clients by the BFE service
- “NT AUTHORITY\SYSTEM” is allowed to perform the RPC call
- It can be bypassed by sending the device IO request directly

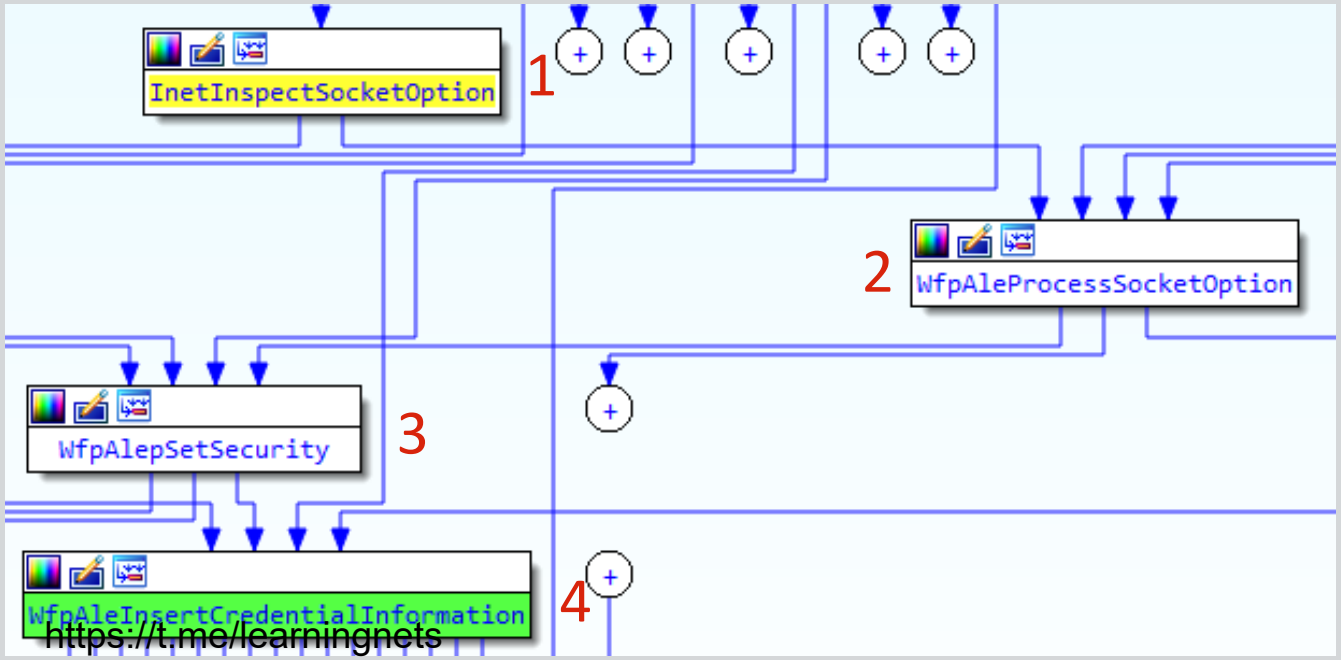
```
int BfeDriverQueryExplicitCredentials(int ValueToQuery, void* ExplicitCredentials)
{
    int value = ValueToQuery;
    return BfeDeviceIoControl(WfpAleHandle, 0x128010, 8i64, &value, 0x107FE, ExplicitCredentials, 0, 0);
}
```

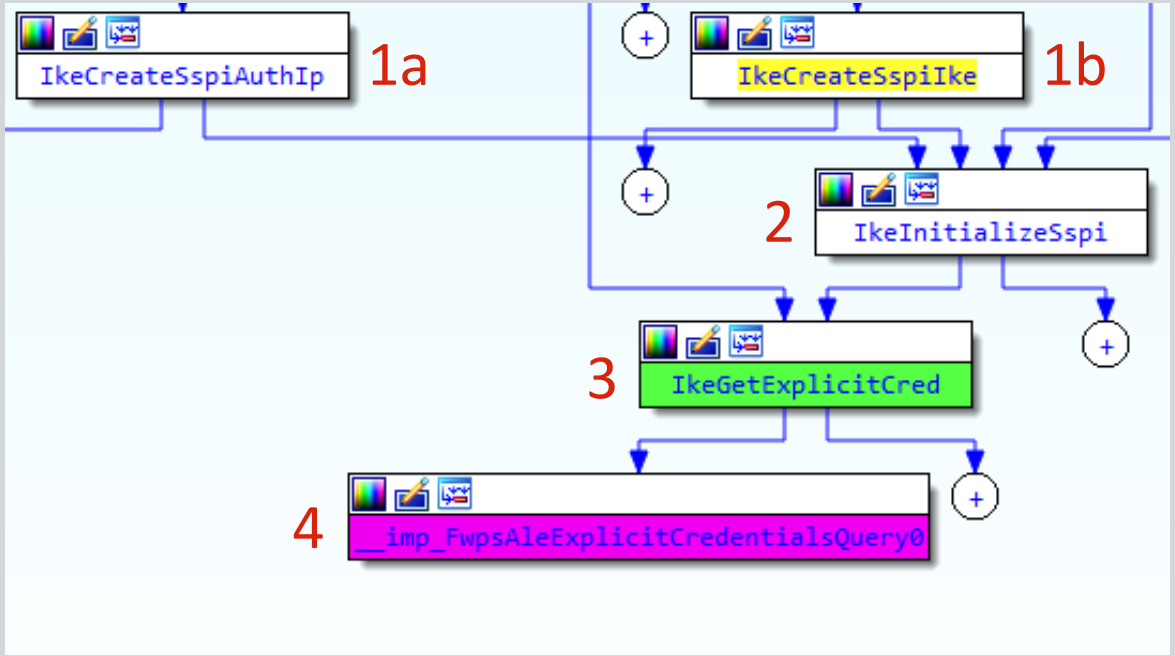
Further Research – Explicit Credentials

- `WfpAleInsertCredentialInformation` is never called
- Credentials aren't stored by default
- The configuration that inserts explicit credentials was not found
- Cross references might shed light on the purpose of explicit credentials

Further Research – Explicit Credentials

- The insert function is related to socket options
- Maybe `WSASetSocketSecurity` is relevant
- The query function is used by the IKE service
- It is related to SSPI





Conclusion

- Single RPC call was reverse engineered to achieve lateral movement and privilege escalation
- Various components of the Windows Filtering Platform were analyzed
- Security mechanisms protecting the platform were bypassed
- Leads were shared to further abuse this platform

Advantages of the techniques:

- ✓ Avoid WinAPI that are monitored by security products
- ✓ Execute programs as SYSTEM **and** other logged on users (most tools only elevate to SYSTEM)
- ✓ Stealthier than ever before - barely any evidence and logs
- ✓ Undetected by several security products

GitHub repo: https://github.com/deepinstinct/NoFilter

Thank You

Contact me:


@RonB_Y


https://www.linkedin.com/in/ron-by