**Communication with Processes**

One of the best places to look for privilege escalation is the processes that are running on the system. Even if a process is not running as an administrator, it may lead to additional privileges. The most common example is discovering a web server like IIS or XAMPP running on the box, placing an aspx/php shell on the box, and gaining a shell as the user running the web server. Generally, this is not an administrator but will often have the SeImpersonate token, allowing for Rogue/Juicy/Lonely Potato to provide SYSTEM permissions.

**Access Tokens**

In Windows, [access tokens](https://docs.microsoft.com/en-us/windows/win32/secauthz/access-tokens) are used to describe the security context (security attributes or rules) of a process or thread. The token includes information about the user account's identity and privileges related to a specific process or thread. When a user authenticates to a system, their password is verified against a security database, and if properly authenticated, they will be assigned an access token. Every time a user interacts with a process, a copy of this token will be presented to determine their privilege level.

**Enumerating Network Services**

The most common way people interact with processes is through a network socket (DNS, HTTP, SMB, etc.). The [netstat](https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat) command will display active TCP and UDP connections which will give us a better idea of what services are listening on which port(s) both locally and accessible to the outside. We may find a vulnerable service only accessible to the localhost (when logged on to the host) that we can exploit to escalate privileges.

**Display Active Network Connections**

Display Active Network Connections

C:\htb> netstat -ano

Active Connections

Proto Local Address Foreign Address State PID

TCP 0.0.0.0:21 0.0.0.0:0 LISTENING 3812

TCP 0.0.0.0:80 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:135 0.0.0.0:0 LISTENING 836

TCP 0.0.0.0:445 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:3389 0.0.0.0:0 LISTENING 936

TCP 0.0.0.0:5985 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:8080 0.0.0.0:0 LISTENING 5044

TCP 0.0.0.0:47001 0.0.0.0:0 LISTENING 4

TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING 528

TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING 996

TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING 1260

TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING 2008

TCP 0.0.0.0:49669 0.0.0.0:0 LISTENING 600

TCP 0.0.0.0:49670 0.0.0.0:0 LISTENING 1888

TCP 0.0.0.0:49674 0.0.0.0:0 LISTENING 616

TCP 10.129.43.8:139 0.0.0.0:0 LISTENING 4

TCP 10.129.43.8:3389 10.10.14.3:63191 ESTABLISHED 936

TCP 10.129.43.8:49671 40.67.251.132:443 ESTABLISHED 1260

TCP 10.129.43.8:49773 52.37.190.150:443 ESTABLISHED 2608

TCP 10.129.43.8:51580 40.67.251.132:443 ESTABLISHED 3808

TCP 10.129.43.8:54267 40.67.254.36:443 ESTABLISHED 3808

TCP 10.129.43.8:54268 40.67.254.36:443 ESTABLISHED 1260

TCP 10.129.43.8:54269 64.233.184.189:443 ESTABLISHED 2608

TCP 10.129.43.8:54273 216.58.210.195:443 ESTABLISHED 2608

TCP 127.0.0.1:14147 0.0.0.0:0 LISTENING 3812

<SNIP>

TCP 192.168.20.56:139 0.0.0.0:0 LISTENING 4

TCP [::]:21 [::]:0 LISTENING 3812

TCP [::]:80 [::]:0 LISTENING 4

TCP [::]:135 [::]:0 LISTENING 836

TCP [::]:445 [::]:0 LISTENING 4

TCP [::]:3389 [::]:0 LISTENING 936

TCP [::]:5985 [::]:0 LISTENING 4

TCP [::]:8080 [::]:0 LISTENING 5044

TCP [::]:47001 [::]:0 LISTENING 4

TCP [::]:49664 [::]:0 LISTENING 528

TCP [::]:49665 [::]:0 LISTENING 996

TCP [::]:49666 [::]:0 LISTENING 1260

TCP [::]:49668 [::]:0 LISTENING 2008

TCP [::]:49669 [::]:0 LISTENING 600

TCP [::]:49670 [::]:0 LISTENING 1888

TCP [::]:49674 [::]:0 LISTENING 616

TCP [::1]:14147 [::]:0 LISTENING 3812

UDP 0.0.0.0:123 \*:\* 1104

UDP 0.0.0.0:500 \*:\* 1260

UDP 0.0.0.0:3389 \*:\* 936

<SNIP>

The main thing to look for with Active Network Connections are entries listening on loopback addresses (127.0.0.1 and ::1) that are not listening on the IP Address (10.129.43.8) or broadcast (0.0.0.0, ::/0). The reason for this is network sockets on localhost are often insecure due to the thought that "they aren't accessible to the network." The one that sticks out immediately will be port 14147, which is used for FileZilla's administrative interface. By connecting to this port, it may be possible to extract FTP passwords in addition to creating an FTP Share at c:\ as the FileZilla Server user (potentially Administrator).

**More Examples**

One of the best examples of this type of privilege escalation is the Splunk Universal Forwarder, installed on endpoints to send logs into Splunk. The default configuration of Splunk did not have any authentication on the software and allowed anyone to deploy applications, which could lead to code execution. Again, the default configuration of Splunk was to run it as SYSTEM$ and not a low privilege user. For more information, check out [Splunk Universal Forwarder Hijacking](https://airman604.medium.com/splunk-universal-forwarder-hijacking-5899c3e0e6b2) and [SplunkWhisperer2](https://clement.notin.org/blog/2019/02/25/Splunk-Universal-Forwarder-Hijacking-2-SplunkWhisperer2/).

Another overlooked but common local privilege escalation vector is the Erlang Port (25672). Erlang is a programming language designed around distributed computing and will have a network port that allows other Erlang nodes to join the cluster. The secret to join this cluster is called a cookie. Many applications that utilize Erlang will either use a weak cookie (RabbitMQ uses rabbit by default) or place the cookie in a configuration file that is not well protected. Some example Erlang applications are SolarWinds, RabbitMQ, and CouchDB. For more information check out the [Erlang-arce blogpost from Mubix](https://malicious.link/post/2018/erlang-arce/)

**Named Pipes**

The other way processes communicate with each other is through Named Pipes. Pipes are essentially files stored in memory that get cleared out after being read. Cobalt Strike uses Named Pipes for every command (excluding [BOF](https://www.cobaltstrike.com/help-beacon-object-files)). Essentially the workflow looks like this:

1. Beacon starts a named pipe of \.\pipe\msagent\_12
2. Beacon starts a new process and injects command into that process directing output to \.\pipe\msagent\_12
3. Server displays what was written into \.\pipe\msagent\_12

Cobalt Strike did this because if the command being ran got flagged by antivirus or crashed, it would not affect the beacon (process running the command). Often, Cobalt Strike users will change their named pipes to masquerade as another program. One of the most common examples is mojo instead of msagent. One of my favorite findings was finding a named pipe start with mojo, but the computer itself did not have Chrome installed. Thankfully, this turned out to be the company's internal red team. It speaks volumes when an external consultant finds the red team, but the internal blue team did not.

**More on Named Pipes**

Pipes are used for communication between two applications or processes using shared memory. There are two types of pipes, [named pipes](https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes) and anonymous pipes. An example of a named pipe is \\.\PipeName\\ExampleNamedPipeServer. Windows systems use a client-server implementation for pipe communication. In this type of implementation, the process that creates a named pipe is the server, and the process communicating with the named pipe is the client. Named pipes can communicate using half-duplex, or a one-way channel with the client only being able to write data to the server, or duplex, which is a two-way communication channel that allows the client to write data over the pipe, and the server to respond back with data over that pipe. Every active connection to a named pipe server results in the creation of a new named pipe. These all share the same pipe name but communicate using a different data buffer.

We can use the tool [PipeList](https://docs.microsoft.com/en-us/sysinternals/downloads/pipelist) from the Sysinternals Suite to enumerate instances of named pipes.

**Listing Named Pipes with Pipelist**

Listing Named Pipes with Pipelist

C:\htb> pipelist.exe /accepteula

PipeList v1.02 - Lists open named pipes

Copyright (C) 2005-2016 Mark Russinovich

Sysinternals - www.sysinternals.com

Pipe Name Instances Max Instances

--------- --------- -------------

InitShutdown 3 -1

lsass 4 -1

ntsvcs 3 -1

scerpc 3 -1

Winsock2\CatalogChangeListener-340-0 1 1

Winsock2\CatalogChangeListener-414-0 1 1

epmapper 3 -1

Winsock2\CatalogChangeListener-3ec-0 1 1

Winsock2\CatalogChangeListener-44c-0 1 1

LSM\_API\_service 3 -1

atsvc 3 -1

Winsock2\CatalogChangeListener-5e0-0 1 1

eventlog 3 -1

Winsock2\CatalogChangeListener-6a8-0 1 1

spoolss 3 -1

Winsock2\CatalogChangeListener-ec0-0 1 1

wkssvc 4 -1

trkwks 3 -1

vmware-usbarbpipe 5 -1

srvsvc 4 -1

ROUTER 3 -1

vmware-authdpipe 1 1

<SNIP>

Additionally, we can use PowerShell to list named pipes using gci (Get-ChildItem).

**Listing Named Pipes with PowerShell**

Listing Named Pipes with PowerShell

PS C:\htb> gci \\.\pipe\

Directory: \\.\pipe

Mode LastWriteTime Length Name

---- ------------- ------ ----

------ 12/31/1600 4:00 PM 3 InitShutdown

------ 12/31/1600 4:00 PM 4 lsass

------ 12/31/1600 4:00 PM 3 ntsvcs

------ 12/31/1600 4:00 PM 3 scerpc

Directory: \\.\pipe\Winsock2

Mode LastWriteTime Length Name

---- ------------- ------ ----

------ 12/31/1600 4:00 PM 1 Winsock2\CatalogChangeListener-34c-0

Directory: \\.\pipe

Mode LastWriteTime Length Name

---- ------------- ------ ----

------ 12/31/1600 4:00 PM 3 epmapper

<SNIP>

After obtaining a listing of named pipes, we can use [Accesschk](https://docs.microsoft.com/en-us/sysinternals/downloads/accesschk) to enumerate the permissions assigned to a specific named pipe by reviewing the Discretionary Access List (DACL), which shows us who has the permissions to modify, write, read, or execute a resource. Let's take a look at the LSASS process. We can also review the DACLs of all named pipes using the command .\accesschk.exe /accepteula \pipe\.

**Reviewing LSASS Named Pipe Permissions**

Reviewing LSASS Named Pipe Permissions

C:\htb> accesschk.exe /accepteula \\.\Pipe\lsass -v

Accesschk v6.12 - Reports effective permissions for securable objects

Copyright (C) 2006-2017 Mark Russinovich

Sysinternals - www.sysinternals.com

\\.\Pipe\lsass

Untrusted Mandatory Level [No-Write-Up]

RW Everyone

FILE\_READ\_ATTRIBUTES

FILE\_READ\_DATA

FILE\_READ\_EA

FILE\_WRITE\_ATTRIBUTES

FILE\_WRITE\_DATA

FILE\_WRITE\_EA

SYNCHRONIZE

READ\_CONTROL

RW NT AUTHORITY\ANONYMOUS LOGON

FILE\_READ\_ATTRIBUTES

FILE\_READ\_DATA

FILE\_READ\_EA

FILE\_WRITE\_ATTRIBUTES

FILE\_WRITE\_DATA

FILE\_WRITE\_EA

SYNCHRONIZE

READ\_CONTROL

RW APPLICATION PACKAGE AUTHORITY\Your Windows credentials

FILE\_READ\_ATTRIBUTES

FILE\_READ\_DATA

FILE\_READ\_EA

FILE\_WRITE\_ATTRIBUTES

FILE\_WRITE\_DATA

FILE\_WRITE\_EA

SYNCHRONIZE

READ\_CONTROL

RW BUILTIN\Administrators

FILE\_ALL\_ACCESS

From the output above, we can see that only administrators have full access to the LSASS process, as expected.

**Named Pipes Attack Example**

Let's walk through an example of taking advantage of an exposed named pipe to escalate privileges. This [WindscribeService Named Pipe Privilege Escalation](https://www.exploit-db.com/exploits/48021) is a great example. Using accesschk we can search for all named pipes that allow write access with a command such as accesschk.exe -w \pipe\\* -v and notice that the WindscribeService named pipe allows READ and WRITE access to the Everyone group, meaning all authenticated users.

**Checking WindscribeService Named Pipe Permissions**

Confirming with accesschk we see that the Everyone group does indeed have FILE\_ALL\_ACCESS (All possible access rights) over the pipe.

Checking WindscribeService Named Pipe Permissions

C:\htb> accesschk.exe -accepteula -w \pipe\WindscribeService -v

Accesschk v6.13 - Reports effective permissions for securable objects

Copyright ⌐ 2006-2020 Mark Russinovich

Sysinternals - www.sysinternals.com

\\.\Pipe\WindscribeService

Medium Mandatory Level (Default) [No-Write-Up]

RW Everyone

FILE\_ALL\_ACCESS

From here, we could leverage these lax permissions to escalate privileges on the host to SYSTEM.