PsExec Talks the Talk.
Can You Walk the Walk?

Presented by:

Brandon Scullion - @nc3pt0r

November 20, 2020

SO-CON 2020

Brandon Scullion - @nc3pt0r

- Senior Detection Consultant at SpecterOps
- Former
  - Security Architecture Analyst
  - Security Engineer
  - Threat Hunt Team / Hunter of Security Threats
    - Assisted standing up an internal Security Operations Center (SOC)
    - Threat analyst, detection engineering, and adversarial simulation to validate detection
- Love
  - Family and miniature homesteading
  - Automation, building things, woodworking, new gadgets

Agenda

A. Understand the underlying technology - PsExec Lateral Movement

B. Identify defender blindspots and our strategy for detection

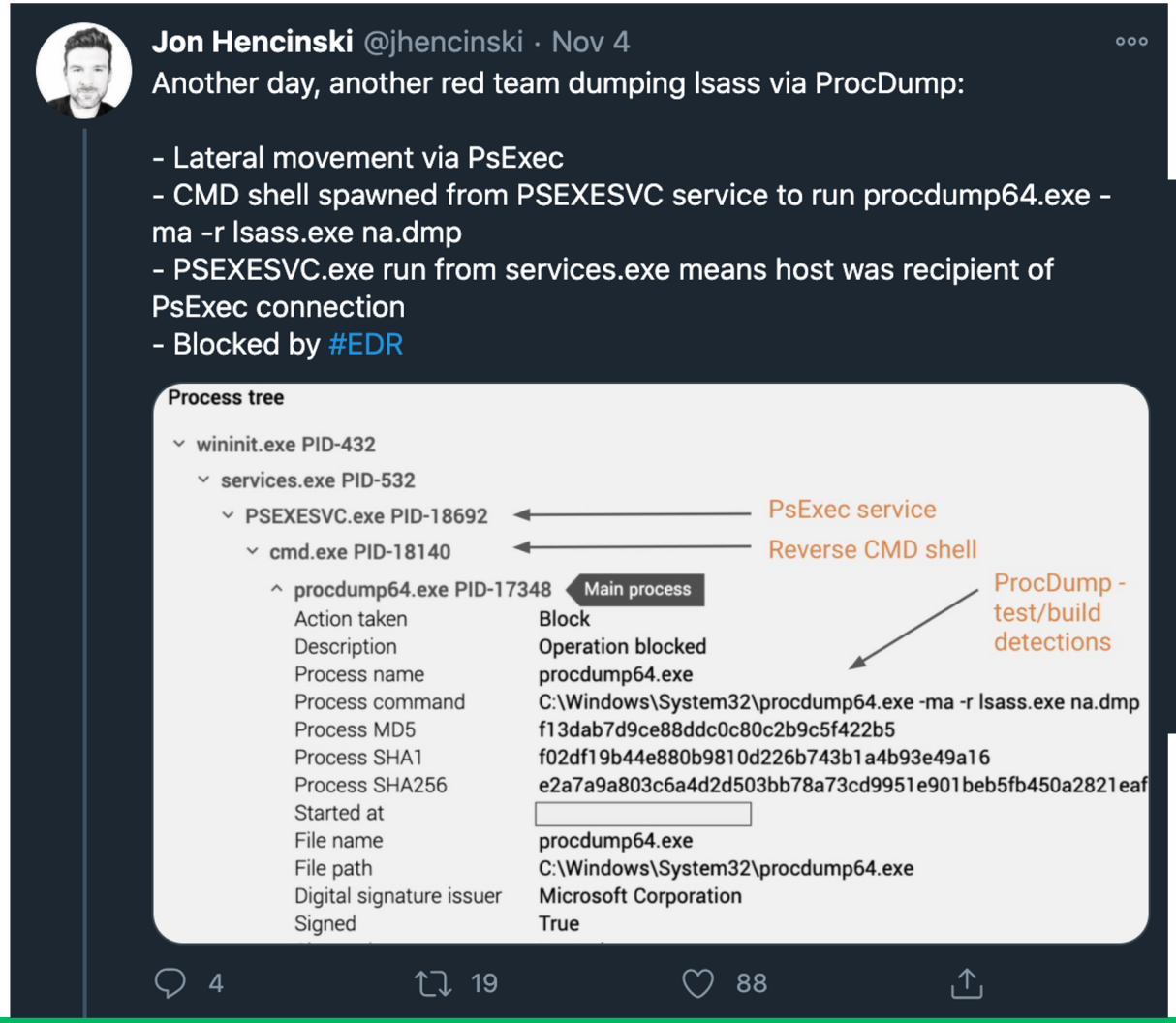
C. Prevention Considerations

https://medium.com/palantir/alerting-and-detection-strategy-framework-52dc33722df2

GOALS

- What and how does PSEXEC work?
- What data sources can we use to monitor this technique?
- What are the blindspots?
- What behavior does the adversary control?
- What behavior cannot be changed by the adversary?
- Is there opportunity for high fidelity detection with the data sources available?

Current Events - 11/4/2020 - Twitter Post by @jhencinski



Jon Hencinski @jhencinski · Nov 4

Another day, another red team dumping lsass via ProcDump:

- Lateral movement via PsExec
- CMD shell spawned from PSEXESVC service to run procdump64.exe -ma -r lsass.exe na.dmp
- PSEXESVC.exe run from services.exe means host was recipient of PsExec connection
- Blocked by #EDR

Process tree

- wininit.exe PID-432
  - services.exe PID-532
    - PSEXESVC.exe PID-18692 ← PsExec service
      - cmd.exe PID-18140 ← Reverse CMD shell
        - procdump64.exe PID-17348 **Main process** ← ProcDump - test/build detections

Action taken	Block
Description	Operation blocked
Process name	procdump64.exe
Process command	C:\Windows\System32\procdump64.exe -ma -r lsass.exe na.dmp
Process MD5	f13dab7d9ce88ddc0c80c2b9c5f422b5
Process SHA1	f02df19b44e880b9810d226b743b1a4b93e49a16
Process SHA256	e2a7a9a803c6a4d2d503bb78a73cd9951e901beb5fb450a2821eaf
Started at	
File name	procdump64.exe
File path	C:\Windows\System32\procdump64.exe
Digital signature issuer	Microsoft Corporation
Signed	True

4 19 88

Jon Hencinski @jhencinski · Nov 4

You can infer:
- Red team has admin rights
- Able to move to hosts within domain
- Run as SYSTEM

Next steps:
- Find source of PSEXEC connection (netconn to host or Windows Event Logs)
- Timeline host
- Establish new leads, pursue them
- Scope
- Test/build new detections

ATT&CK Categorization

Remote Services: SMB/Windows Admin Shares - T1021.002

System Services: Service Execution - T1569.002

Signed Binary Proxy Execution - T1218

Remote Services: SMB/Windows Admin Shares

Other sub-techniques of Remote Services (6)

Adversaries may use Valid Accounts to interact with a remote network share using Server Message Block (SMB). The adversary may then perform actions as the logged-on user.

SMB is a file, printer, and serial port sharing protocol for Windows machines on the same network or domain. Adversaries may use SMB to interact with file shares, allowing them to move laterally throughout a network. Linux and macOS implementations of SMB typically use Samba.

Windows systems have hidden network shares that are accessible only to administrators and provide the ability for remote file copy and other administrative functions. Example network shares include `C$`, `ADMIN$`, and `IPC$`. Adversaries may use this technique in conjunction with administrator-level Valid Accounts to remotely access a networked system over SMB,[1] to interact with systems using remote procedure calls (RPCs),[2] transfer files, and run transferred binaries through remote Execution. Example execution techniques that rely on authenticated sessions over SMB/RPC are Scheduled Task/Job, Service Execution, and Windows Management Instrumentation. Adversaries can also use NTLM hashes to access administrator shares on systems with Pass the Hash and certain configuration and patch levels.[3]

ID: T1021.002

Sub-technique of: T1021

Tactic: Lateral Movement

Platforms: Windows

System Requirements: SMB enabled; Host/network firewalls not blocking SMB ports between source and destination; Use of domain account in administrator group on remote system or default system admin account.

Permissions Required: Administrator, User

Data Sources: Authentication logs, Process command-line parameters, Process monitoring, Process use of network

CAPEC ID: CAPEC-561

Version: 1.0

Created: 11 February 2020

Last Modified: 23 March 2020

System Services: Service Execution

Other sub-techniques of System Services (2)

Adversaries may abuse the Windows service control manager to execute malicious commands or payloads. The Windows service control manager (`services.exe`) is an interface to manage and manipulate services.[1] The service control manager is accessible to users via GUI components as well as system utilities such as `sc.exe` and Net.

PsExec can also be used to execute commands or payloads via a temporary Windows service created through the service control manager API.[2]

Adversaries may leverage these mechanisms to execute malicious content. This can be done by either executing a new or modified service. This technique is the execution used in conjunction with Windows Service during service persistence or privilege escalation.

ID: T1569.002

Sub-technique of: T1569

Tactic: Execution

Platforms: Windows

Permissions Required: Administrator, SYSTEM

Data Sources: Process command-line parameters, Process monitoring, Windows Registry

Supports Remote: Yes

Version: 1.0

Created: 10 March 2020

Last Modified: 28 March 2020

Version Permalink

Signed Binary Proxy Execution

Sub-techniques (11)

Adversaries may bypass process and/or signature-based defenses by proxying execution of malicious content with signed binaries. Binaries signed with trusted digital certificates can execute on Windows systems protected by digital signature validation. Several Microsoft signed binaries that are default on Windows installations can be used to proxy execution of other files.

ID: T1218

Sub-techniques: T1218.001, T1218.002, T1218.003, T1218.004, T1218.005, T1218.007, T1218.008, T1218.009, T1218.010, T1218.011, T1218.012

Tactic: Defense Evasion

Platforms: Windows

Permissions Required: Administrator, User

Data Sources: API monitoring, Binary file metadata, DLL monitoring, File monitoring, Loaded DLLs, Process command-line parameters, Process monitoring, Process use of network, Windows Registry

Defense Bypassed: Anti-virus, Application control, Digital Certificate Validation

Contributors: Hans Christoffer Gaardløs; Nishan Maharjan, @loki248; Praetorian

Version: 2.1

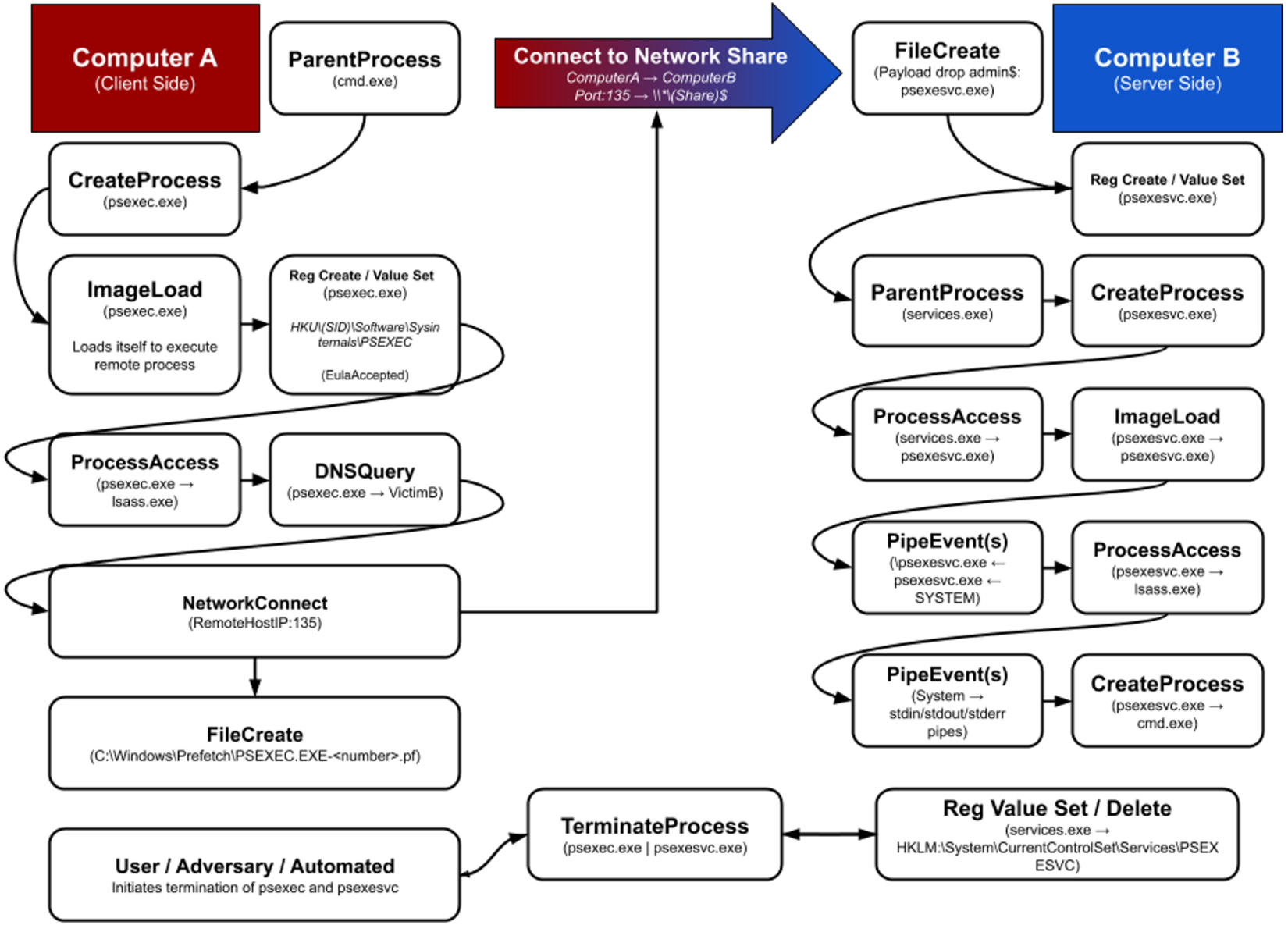
Created: 18 April 2018

Last Modified: 21 October 2020

Notes before testing

- Administrator or System permissions required
- SMB enabled and not blocked from source to destination
- Data Sources
  - Process Monitoring
  - Command-Line Parameters
  - Network Connections
  - Registry Monitoring
  - Module loads / DLL Monitoring?
  - RPC?

CLIENT SIDE



SERVER SIDE

Client Side (Computer A)

PSEXEC Flow Chart: ClientSide (Computer A)

ParentProcess
(User/Attacker Interaction)

User initiates PSEXEC from a ParentProcess

- Cmd.exe → with arguments
- PowerShell (or other shell) → with arguments

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

PSEXEC Flow Chart: ClientSide (Computer A)

ParentProcess
(User/Attacker Interaction)

CreateProcess
(PSEXEC.exe)

User initiates PSEXEC from a ParentProcess

- Cmd.exe → with arguments
- PowerShell (or other shell) → with arguments

Process creation occurs of the binary used to generate the RPC lateral movement

- In this test scenario, PSEXEC.exe

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

PSEXEC Flow Chart: ClientSide (Computer A)

User initiates PSEXEC from a ParentProcess

- Cmd.exe → with arguments
- PowerShell (or other shell) → with arguments

Process creation occurs of the binary used to generate the RPC lateral movement

- In this test scenario, PSEXEC.exe

Interesting event, attacker controlled

- PSEXEC.exe loads itself defined: to be able to execute remote process

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 7 - Image Load Event

WinEvent:Win32_ModuleLoadTrace class

DATP: DevcieImageLoadEvents

PSEXEC Flow Chart: ClientSide (Computer A)

User initiates PSEXEC from a ParentProcess

- Cmd.exe → with arguments
- PowerShell (or other shell) → with arguments

Process creation occurs of the binary used to generate the RPC lateral movement

- In this test scenario, PSEXEC.exe

Interesting event, attacker controlled

- PSEXEC.exe loads itself defined: to be able to execute remote process

Interesting event, attacker controlled:

- EULAAccepted Regkey Creation for PSEXEC
- Attacker could change with own tool

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 7 - Image Load Event

WinEvent:Win32_ModuleLoadTrace class

DATP: DevcieImageLoadEvents

SysmonID: 12,13,14 - Registry Events

WinEventID: 4657 - Registry Event

DATP: DeviceRegistryEvents

PSEXEC Flow Chart: ClientSide (Computer A)

User initiates PSEXEC from a ParentProcess

- Cmd.exe → with arguments
- PowerShell (or other shell) → with arguments

Process creation occurs of the binary used to generate the RPC lateral movement

- In this test scenario, PSEXEC.exe

Interesting event, attacker controlled

- PSEXEC.exe loads itself defined: to be able to execute remote process

Interesting event, attacker controlled:

- EULAAccepted Regkey Creation for PSEXEC
- Attacker could change with own tool

LSASS access is required for:

- Remote Connection
- Elevation (Local or Remote)

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 1 - Process Create

WinEventID: 4688 - Process Create

DATP: DeviceProcessEvents

SysmonID: 7 - Image Load Event

WinEvent:Win32_ModuleLoadTrace class

DATP: DevcieImageLoadEvents

SysmonID: 12,13,14 - Registry Events

WinEventID: 4657 - Registry Event

DATP: DeviceRegistryEvents

SysmonID: 10 - Process Access

*WinEventID: 4661 - A handle to an object was requested

PSEXEC Flow Chart: ClientSide (Computer A)

DNSQuery
(PSEXEC.exe → Victim B)

Process verifies remote host is remote host

SysmonID: 22 - DNS Event

DATP: DeviceNetworkEvents

PSEXEC Flow Chart: ClientSide (Computer A)

Process verifies remote host is remote host

Establishes network connection to remote host over port 135

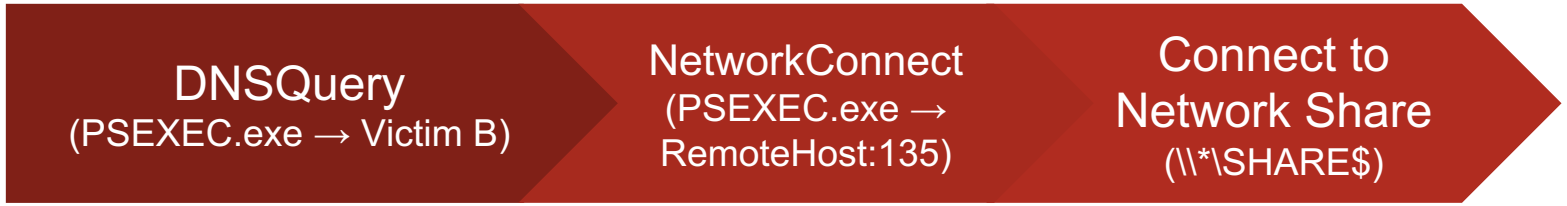
SysmonID: 22 - DNS Event

DATP: DeviceNetworkEvents

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S)

PSEXEC Flow Chart: ClientSide (Computer A)



Process verifies remote host is remote host

Establishes network connection to remote host over port 135

Drops a binary into the admin share on remote host and begins to create the service for RPC communication

SysmonID: 22 - DNS Event

DATP: DeviceNetworkEvents

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S)

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S) - RPC Event

DATP: DeviceNetworkEvent, DeviceFileEvents

PSEXEC Flow Chart: ClientSide (Computer A)

Process verifies remote host is remote host

Establishes network connection to remote host over port 135

Drops a binary into the admin share on remote host and begins to create the service for RPC communication

ProcMon utility identifies a file creation of the service binary on host A before copying it over to host B

SysmonID: 22 - DNS Event

DATP: DeviceNetworkEvents

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S)

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S) - RPC Event

DATP: DeviceNetworkEvent, DeviceFileEvents

SysmonID: 11 - File Create

ProcMon Capture

DATP: DeviceFileEvents

PSEXEC Flow Chart: ClientSide (Computer A)

Process verifies remote host is remote host

Establishes network connection to remote host over port 135

Drops a binary into the admin share on remote host and begins to create the service for RPC communication

ProcMon utility identifies a file creation of the service binary on host A before copying it over to host B

A prefetch file is created on host A which is used by the application/PSEXEC and contains information about the application

SysmonID: 22 - DNS Event

DATP: DeviceNetworkEvents

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S)

SysmonID: 3 - Network Connect

*WinEventID:: 5712(S) - RPC Event

DATP: DeviceNetworkEvent, DeviceFileEvents

SysmonID: 11 - File Create

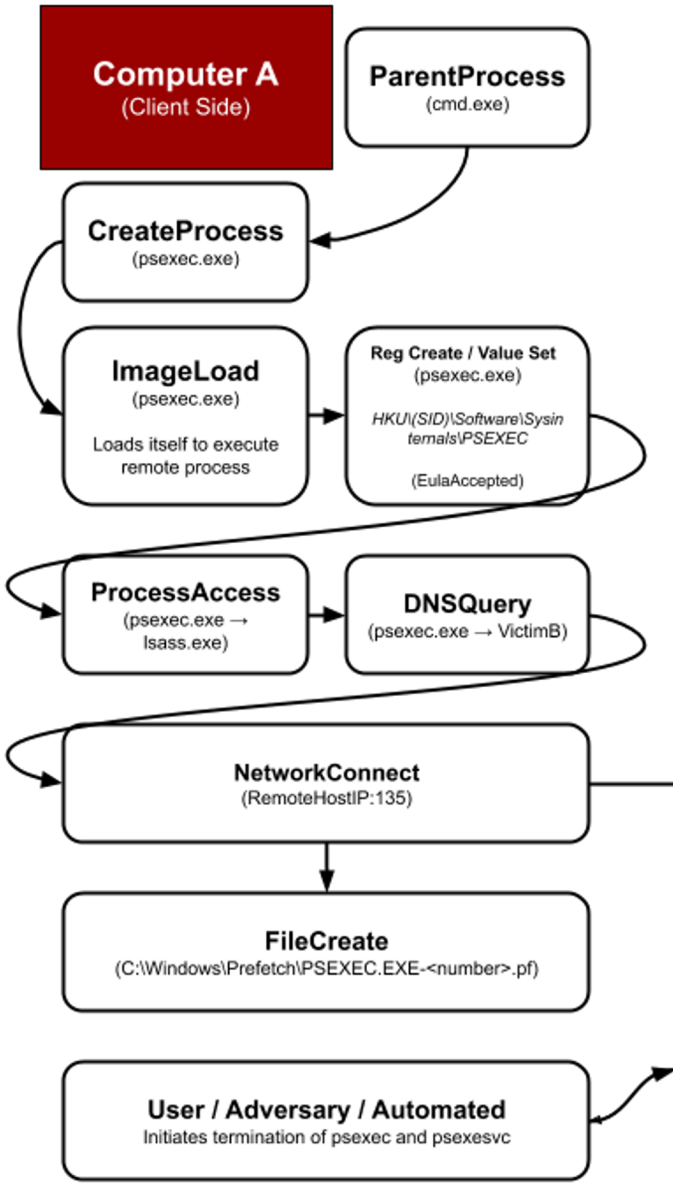
ProcMon Capture

DATP: DeviceFileEvents

SysmonID: 11 - File Create

DATP: DeviceFileEvents

CLIENT SIDE


Computer A
(Client Side)
ParentProcess
(cmd.exe)
CreateProcess
(psexec.exe)
ImageLoad
(psexec.exe)
Loads itself to execute remote process
Reg Create / Value Set
(psexec.exe)
HKU\(SID)\Software\Sysinternals\PSEXEC
(EulaAccepted)
ProcessAccess
(psexec.exe → lsass.exe)
DNSQuery
(psexec.exe → VictimB)
NetworkConnect
(RemoteHostIP:135)
FileCreate
(C:\Windows\Prefetch\PSEXEC.EXE-<number>.pf)
User / Adversary / Automated
Initiates termination of psexec and psexesvc

Server Side (Computer B)

PSEXEC Flow Chart: ServerSide (Computer B)

FileCreate
ServiceBinary in ADMIN$

File creation for dropping the service binary

Currently logs resolve the full directory path, example ADMIN$ == C:\windows\PSEXESVC.exe

SysmonID: 11 - FileCreate

DATP: DeviceFileEvents

PSEXEC Flow Chart: ServerSide (Computer B)

File creation for dropping the service binary

Currently logs resolve the full directory path, example ADMIN$ == C:\windows\PSEXESVC.exe

If a service is newly created, there will be a correlated Reg Key created

SACL could be used to detect service creation on the registry key

SysmonID: 11 - FileCreate

DATP: DeviceFileEvents

SysmonID: Registry 12,13, 14

WinEventID: 4663 when SACL enabled for specific object

DATP: DeviceRegistryEvents

PSEXEC Flow Chart: ServerSide (Computer B)

File creation for dropping the service binary

Currently logs resolve the full directory path, example ADMIN$ == C:\windows\PSEXESVC.exe

If a service is newly created, there will be a correlated Reg Key created

SACL could be used to detect service creation on the registry key

Service.exe will create the process/service using PSEXESVC.exe

ProcessCreate was logged here

SysmonID: 11 - FileCreate

DATP: DeviceFileEvents

SysmonID: Registry 12,13, 14

WinEventID: 4663 when SACL enabled for specific object

DATP: DeviceRegistryEvents

SysmonID: 1 - Process Create

WinEventID: 4697 - Service Install

DATP: DeviceEvents*, DeviceProcessEvents

PSEXEC Flow Chart: ServerSide (Computer B)

File creation for dropping the service binary

Currently logs resolve the full directory path, example ADMIN$ == C:\windows\PSEXESVC.exe

If a service is newly created, there will be a correlated Reg Key created

SACL could be used to detect service creation on the registry key

Service.exe will create the process/service using PSEXESVC.exe

ProcessCreate was logged here

Services.exe then access the newly created process

Not fully sure why here -- Task: Binary Triage PSEXEC/PSEXESVC

SysmonID: 11 - FileCreate

DATP: DeviceFileEvents

SysmonID: Registry 12,13, 14

WinEventID: 4663 when SACL enabled for specific object

DATP: DeviceRegistryEvents

SysmonID: 1 - Process Create

WinEventID: 4697 - Service Install

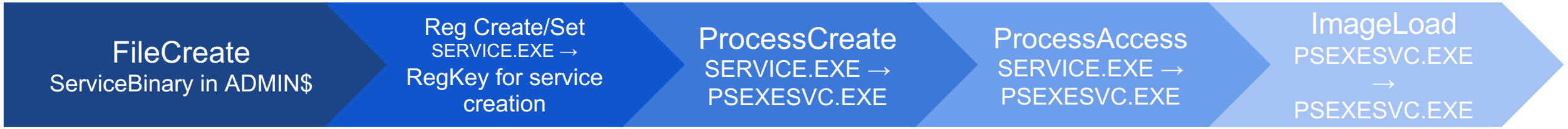
DATP: DeviceEvents*, DeviceProcessEvents

Sysmon: 10 - Process Access

WinEventID: 4658 - Handle to Process was Closed

DATP: DeviceEvents*

PSEXEC Flow Chart: ServerSide (Computer B)



File creation for dropping the service binary

Currently logs resolve the full directory path, example ADMIN$ == C:\windows\PSEXESVC.exe

If a service is newly created, there will be a correlated Reg Key created

SACL could be used to detect service creation on the registry key

Service.exe will create the process/service using PSEXESVC.exe

ProcessCreate was logged here

Services.exe then access the newly created process

Not fully sure why here -- Task: Binary Triage PSEXEC/PSEXESVC

Services loading the image binary again into itself for remote execution

SysmonID: 11 - FileCreate

DATP: DeviceFileEvents

SysmonID: Registry 12,13, 14

WinEventID: 4663 when SACL enabled for specific object

DATP: DeviceRegistryEvents

SysmonID: 1 - Process Create

WinEventID: 4697 - Service Install

DATP: DeviceEvents*, DeviceProcessEvents

Sysmon: 10 - Process Access

WinEventID: 4658 - Handle to Process was Closed

DATP: DeviceEvents*

Sysmon: 7 - Image Load Event

DATP: DeviceImageLoadEvents

PSEXEC Flow Chart: ServerSide (Computer B)

PipeEvent(s)
\psexesvc.exe ←
psexesvc.exe ← SYSTEM

Named pipe is created here

Pipe name matches binary name

NOTE: Pipe name could be attacker controlled

SysmonID: 17,18 - Pipe Event

Triage:
```
Invoke-Command -Session $session -ScriptBlock {if([System.IO.Directory]::GetFiles("\\.\pipe\\", "*"))}
```

PSEXEC Flow Chart: ServerSide (Computer B)

Named pipe is created here

Pipe name matches binary name

NOTE: Pipe name could be attacker controlled

PSEXESVC requires access to lsass in order to spawn the commands that ClientSide initiated with PSEXEC.exe

Especially if elevating to System

SysmonID: 17,18 - Pipe Event

Triage:
```
Invoke-Command -Session $session -ScriptBlock {if([System.IO.Directory]::GetFiles("\\.\pipe\\", "*"))}
```

SysmonID: 10 - Process Access

*WinEventID: 4661 - A handle to an object was requested

PSEXEC Flow Chart: ServerSide (Computer B)

Named pipe is created here

Pipe name matches binary name

NOTE: Pipe name could be attacker controlled

PSEXESVC requires access to lsass in order to spawn the commands that ClientSide initiated with PSEXEC.exe

Especially if elevating to System

Interesting but probably attacker controlled:

Pipe events created for stdin/stdout/stderr

SysmonID: 17,18 - Pipe Event

Triage: `Invoke-Command -Session $session -ScriptBlock {if ([System.IO.Directory]::GetFiles("\\.\pipe\\", "*"))}`

SysmonID: 10 - Process Access

*WinEventID: 4661 - A handle to an object was requested

SysmonID: 17,18 - Pipe Event

DATP: DeviceEvents*, DeviceFileEvents

PSEXEC Flow Chart: ServerSide (Computer B)

Named pipe is created here

Pipe name matches binary name

NOTE: Pipe name could be attacker controlled

SysmonID: 17,18 - Pipe Event

Triage: `Invoke-Command -Session $session -ScriptBlock {if ([System.IO.Directory]::GetFiles("\\.\pipe\\", "*"))}`

PSEXESVC requires access to lsass in order to spawn the commands that ClientSide initiated with PSEXEC.exe

Especially if elevating to System

SysmonID: 10 - Process Access

*WinEventID: 4661 - A handle to an object was requested

Interesting but probably attacker controlled:

Pipe events created for stdin/stdout/stderr

SysmonID: 17,18 - Pipe Event

DATP: DeviceEvents*, DeviceFileEvents

The final step from the psexec.exe RPC communication, in this event cmd.exe interactive shell was launched

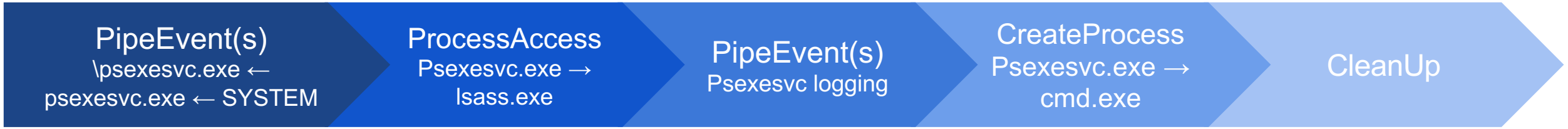
This could be commands or bat file

SysmonID: 1 - Process Create

SysmonID: 10 - Process Access

WinEventID: 4688 - Process Create

PSEXEC Flow Chart: ServerSide (Computer B)



Named pipe is created here

Pipe name matches binary name

NOTE: Pipe name could be attacker controlled

SysmonID: 17,18 - Pipe Event

Triage: `Invoke-Command -Session $session -ScriptBlock {if ([System.IO.Directory]::GetFiles("\\.\pipe\\", "*"))}`

PSEXESVC requires access to lsass in order to spawn the commands that ClientSide initiated with PSEXEC.exe

Especially if elevating to System

SysmonID: 10 - Process Access

*WinEventID: 4661 - A handle to an object was requested

Interesting but probably attacker controlled:

Pipe events created for stdin/stdout/stderr

SysmonID: 17,18 - Pipe Event

DATP: DeviceEvents*, DeviceFileEvents

The final step from the psexec.exe RPC communication, in this event cmd.exe interactive shell was launched

This could be commands or bat file

SysmonID: 1 - Process Create

SysmonID: 10 - Process Access

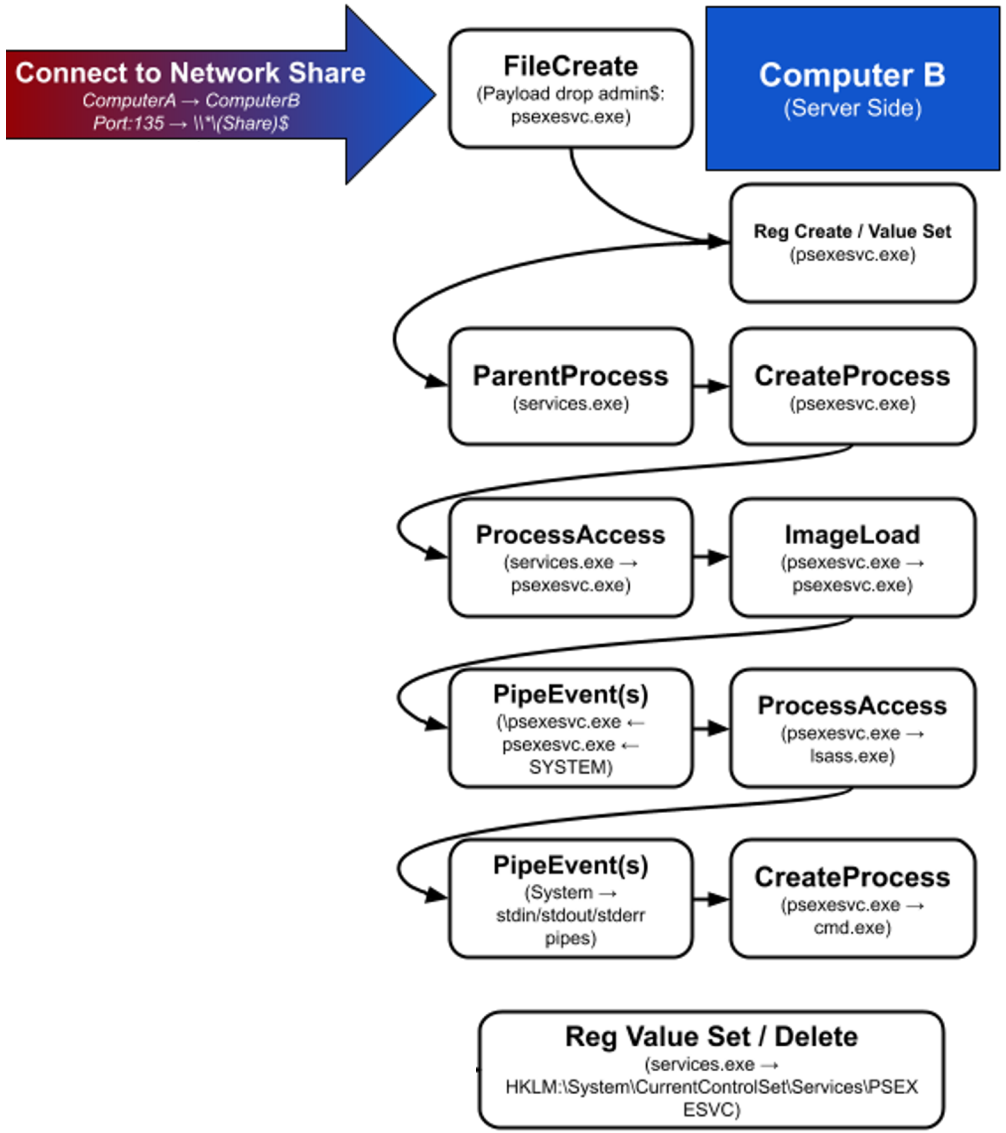
WinEventID: 4688 - Process Create

Upon termination of remote shell, cleanup actions occur:

- Delete service and registry keys
- Process termination

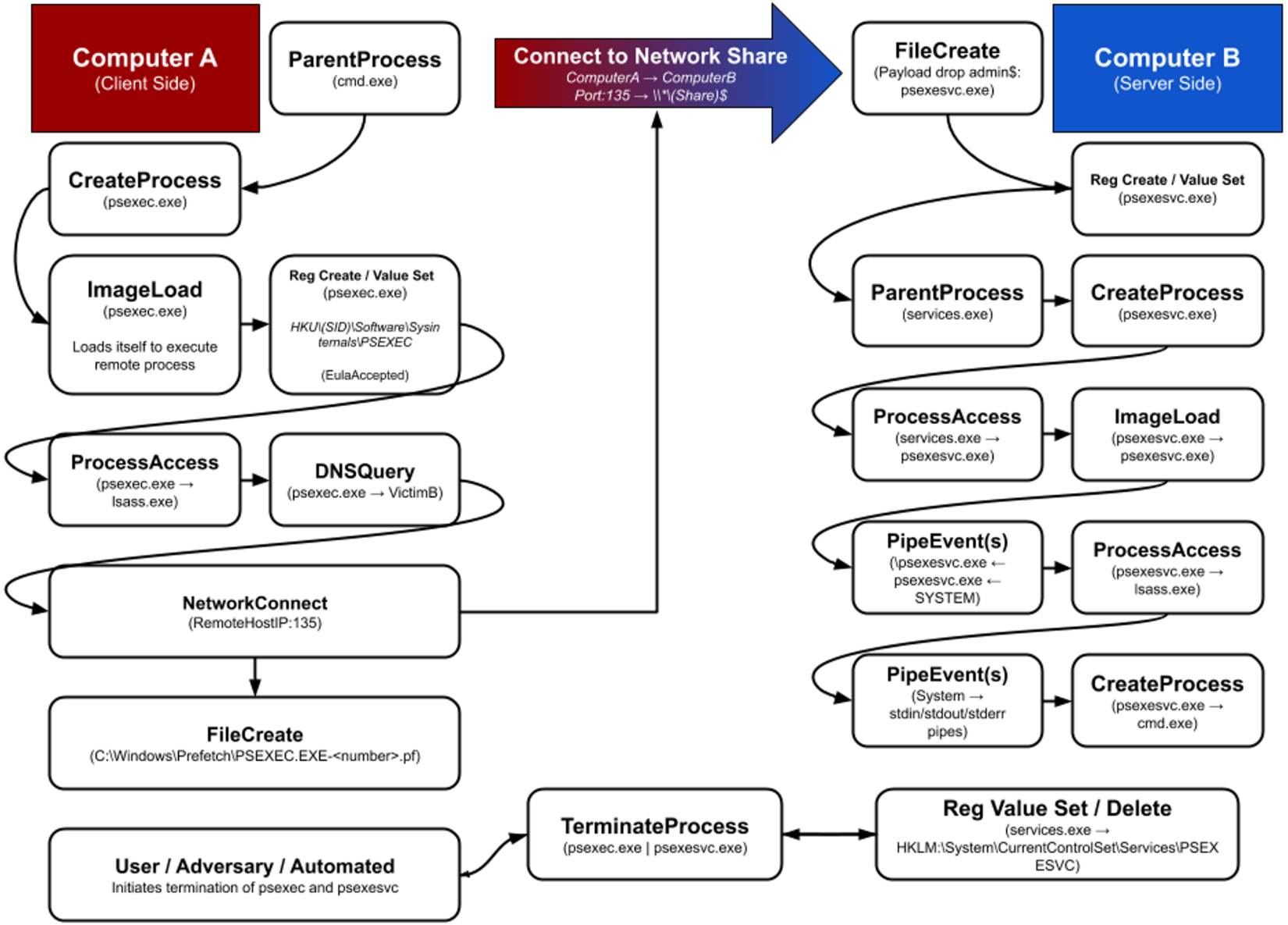
SysmonID: 5 - Process Terminate

SysmonID: 12, 13 - Registry object added/deleted, Value Set


Connect to Network Share
ComputerA → ComputerB
Port:135 → *\(Share)$
FileCreate
(Payload drop admin$: psexesvc.exe)
Computer B
(Server Side)
Reg Create / Value Set
(psexesvc.exe)
ParentProcess
(services.exe)
CreateProcess
(psexesvc.exe)
ProcessAccess
(services.exe → psexesvc.exe)
ImageLoad
(psexesvc.exe → psexesvc.exe)
PipeEvent(s)
(\psexesvc.exe ← psexesvc.exe ← SYSTEM)
ProcessAccess
(psexesvc.exe → lsass.exe)
PipeEvent(s)
(System → stdin/stdout/stderr pipes)
CreateProcess
(psexesvc.exe → cmd.exe)
Reg Value Set / Delete
(services.exe → HKLM:\System\CurrentControlSet\Services\PSEXESVC)

SERVER SIDE

CLIENT SIDE



SERVER SIDE

GOALS

- What and how does PSEXEC work?
- What data sources can we use to monitor this technique?
- What are the blindspots?
- What behavior does the adversary control?
- What behavior cannot be changed by the adversary?
- Is there opportunity for high fidelity detection with the data sources available?

PROBLEMS

- Lots of moving parts
- There is psexec specific behavior / Attacker controlled behavior
  - AcceptEULA
  - Name and Service Name
  - Default Admin Shares
- Lots of module loads!
- RPC and SMB communication between hosts -- Is this needed?

Invoke-PsExec

Script Description:

This function is a rough port of Metasploit's PsExec functionality. It utilizes Windows API calls to open up the service manager on a remote machine, creates/run a service with an associated binary path or command, and then cleans everything up.

```
PS C:\> Invoke-PsExec -ComputerName 192.168.50.200 -
Command "net user backdoor password123 /add" -
ServiceName Updater32
```

References:

- https://docs.microsoft.com/en-us/windows/win32/services/services-and-rpc-tcp
- https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-PsExec.ps1

Under the Hood

- PsExec.exe and Invoke-PsExec.ps1 both communicating using Remote Procedure Calls (RPC) over port 135
- Modules are loaded in order to create the service on remote host
- Services is interacted with via Windows API calls made to OpenSCManagerW and OpenSCManagerA

```
PS C:\> $advapi.Imports | Where-Object -Property FunctionName -like "*SCManager*"
```

Returns

ModuleName	FunctionName	Ordinal	VirtualAddress
api-ms-win-service-management-l1-1-0.dll	OpenSCManagerW		0x000000000009499C
api-ms-win-service-winsvc-l1-2-0.dll	OpenSCManagerA		0x0000000000094CB2

References:

- Luke Paine (SpecterOps) and https://posts.specterops.io/utilizing-rpc-telemetry-7af9ea08a1d5 (Jared Atkinson, Luke Paine, and Jonathan Johnson)
- https://docs.microsoft.com/en-us/windows/win32/services/services-and-rpc-tcp
- https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-PsExec.ps1

Under the Hood -- Windows Documentation

- Starting with Windows Vista, the service control manager (SCM) supports remote procedure calls over both Transmission Control Protocol (RPC/TCP) and named pipes (RPC/NP).
  - Client-side SCM functions use RPC/TCP by default.
- RPC/TCP is appropriate for most applications that use SCM functions remotely, such as remote administration or monitoring tools.
  - However, for compatibility and performance, some applications might need to disable RPC/TCP by setting the registry values described in this topic.
- When a service calls a remote SCM function, the client-side SCM first attempts to use RPC/TCP to communicate with the server-side SCM.
  - If the server is running a version of Windows that supports RPC/TCP and allows RPC/TCP traffic, the RPC/TCPP connection will succeed.
    - If the server is running a version of Windows that does not support RPC/TCP, or supports RPC/TCP but is operating behind a firewall which allows only named pipe traffic, the RPC/TCP connection times out and the SCM retries the connection with RPC/NP.
    - This will succeed eventually but can take some time (typically more than 20 seconds), causing the OpenSCManager function to appear blocked.
- TCP does not carry user credentials specified with a net use command.
  - Therefore, if RPC/TCP is enabled and sc.exe is used to attempt to access the specified service, the command could fail with access denied.
  - Disabling RPC/TCP on the client side causes the sc.exe command to use a named pipe that does carry user credentials, so the command will succeed.
    - For information about sc.exe, see Controlling a Service Using SC.

https://docs.microsoft.com/en-us/windows/win32/services/services-and-rpc-tcp
https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-PsExec.ps1

GOALS

- What and how does PSEXEC work?
- What data sources can we use to monitor this technique?
- What are the blindspots?
- What behavior does the adversary control?
- What behavior cannot be changed by the adversary?
- Is there opportunity for high fidelity detection with the data sources available?

PROBLEMS

- Lots of moving parts
- There is psexec specific behavior / Attacker controlled behavior
  - AcceptEULA
  - Name and Service Name
  - Default Admin Shares
- Lots of module loads!

DIAGNOSIS

What we know:

Client Side - Computer A

- Create Process
- CreateFile: PSEXEC.exe
- Network Connect: Port 135
- ProcMon FileCreate
- //RemoteHost/Admin$/PSEXESVC.exe

Server Side - Computer B

- FileCreate → Admin$/PSEXESVC.exe
- Service Creation
- ProcessCreate PSEXESVC.exe
- PipeEventCreate
- Process Create → cmd.exe

Blindspots

Is the adversary in control?

- We want to broaden the analytic but why?
  - PSEXEC is part of Microsoft Sysinternals Suite developed by Mark Russinovich
  - Adversaries can use this utility, but what if they build their own binary?
  - By casting a wider net, we can identify same behaviors outside of PSEXEC that mirror the same behaviors
- Identify broad indicators that the adversary can NOT control or change
- Are there logging blindspots in our environment for this technique?
  - The gap here is knowing where to look
  - Not all machines collect the same data
  - *** Not a blocker

Strategy

- Start with FileCreate events within Admin Shares (Admin$, C$, etc...)
  - This result provides the service binary name that will get dropped on ServerSide
  - Events identify both ClientSide and ServerSide machines
  - Events identify the initiating process on ClientSide machine
- Use ServerSide machine as the unique field to pivot into process events from that device
  - Expectation: Service creation / Process creation events on ServerSide machine
- Add Network Events over Port 135
- ClientSide Process Creates join based on ServerSide host within 1 min time window
  - Start time of the ServerSide FileCreate event
  - End time of the ClientSide process events

GOALS

- What and how does PSEXEC work?
- What data sources can we use to monitor this technique?
- What are the blindspots?
- What behavior does the adversary control?
- What behavior cannot be changed by the adversary?
- Is there opportunity for high fidelity detection with the data sources available?

PROBLEMS

- Lots of moving parts
- There is psexec specific behavior / Attacker controlled behavior
  - AcceptEULA
  - Name and Service Name
  - Default Admin Shares
- Lots of module loads!

DIAGNOSIS

What we know:
Client Side - Computer A

- Create Process
- CreateFile: PSEXEC.exe
- Network Connect: Port 135
- ProcMon FileCreate
- //RemoteHost/Admin$/PSEXESVC.exe

Server Side - Computer B

- FileCreate → Admin$/PSEXESVC.exe
- Service Creation
- ProcessCreate PSEXESVC.exe
- PipeEventCreate
- Process Create → cmd.exe

STRATEGY

Correlation Opportunity:

- Monitor for Admin Share File Creation Events (SysmonID 11)
- Using Primary Key Join with Process Creation Events (SysmonID 1)
- Network Connection Events - Port 135 RPC (SysmonID 3)

Prevention Considerations

- Does the organization really need SMB enabled host to host?
  - Communication between user endpoints over port 135 or 445 is not required for Windows to function
  - This may still need to be enabled between host and server (File shares, network shares, etc…)
  - Test, test and test gold images - blocking this inter-host protocol is effective in defending against this technique
  - WinRM, WMI, and PSRemoting are much more secure for system administration!
- Ensure proper admin share access policies
  - What shares must be available on the user endpoints? Admin$? C$?
    - Who has access to these shares? Localadmin? Specified domain user? Everyone?

GOALS

- What and how does PSEXEC work?
- What data sources can we use to monitor this technique?
- What are the blindspots?
- What behavior does the adversary control?
- What behavior cannot be changed by the adversary?
- Is there opportunity for high fidelity detection with the data sources available?

PROBLEMS

- Lots of moving parts
- There is psexec specific behavior / Attacker controlled behavior
  - AcceptEULA
  - Name and Service Name
  - Default Admin Shares
- Lots of module loads!

DIAGNOSIS

What we know:

Client Side - Computer A

- Create Process
- CreateFile: PSEXEC.exe
- Network Connect: Port 135
- ProcMon FileCreate
- //RemoteHost/Admin$/PSEXESVC.exe

Server Side - Computer B

- FileCreate → Admin$/PSEXESVC.exe
- Service Creation
- ProcessCreate PSEXESVC.exe
- PipeEventCreate
- Process Create → cmd.exe

STRATEGY

Correlation 1:

- Monitor for Admin Share File Creation Events (SysmonID 11)
- Using Primary Key Join with Process Creation Events (SysmonID 1)
- Network Connection Events - Port 135 RPC,445 SMB (SysmonID 3)

Prevention:

- Validate business needs do not require SMB over Port 135 or 445 between user endpoints
- System administration should not be required using the PSEXEC Utility
- WinRM, WMI and PSRemoting should be used for remote system administration
- Ensure proper Admin Share Permissions

Resources and References

Mitre

- https://attack.mitre.org/techniques/T1218
- https://attack.mitre.org/techniques/T1218/011
- https://attack.mitre.org/techniques/T1569/002
- https://attack.mitre.org/techniques/T1021/002
- https://attack.mitre.org/software/S0029
- https://attack.mitre.org/techniques/T1078/003
- https://attack.mitre.org/techniques/T1570

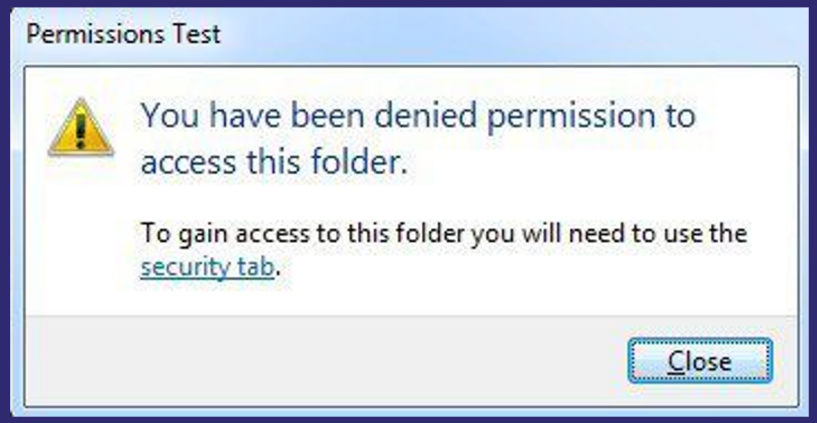
Blogs and Articles

- https://www.praetorian.com/blog/signed-binaries-proxy-execution-t1218
- https://docs.microsoft.com/en-us/sysinternals/downloads/psexec
- https://posts.specterops.io/utilizing-rpc-telemetry-7af9ea08a1d5

Tools or Projects

- https://github.com/Cyb3rWard0g/HELK
- https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
- https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
- https://docs.microsoft.com/en-us/sysinternals/downloads/psexec

END


Permissions Test
You have been denied permission to access this folder.
To gain access to this folder you will need to use the security tab.
Close