

Relay Your Heart Away

An OPSEC-Conscious Approach to 445 Takeover



Introduction

Nick Powers

- Adversary Simulation, Service Architect at SpecterOps
- Focused on red team and pentest engagements
- Interests:
 - Initial access techniques
 - Windows internals
 - Authentication relay attacks
- **@zyn3rgy**

Agenda

What will we be covering?

1. Relevance of 445/tcp control
2. Existing solutions to this problem
3. Debugging drivers for new solutions
4. Tooling for automation of abuse
5. Demonstration of practical abuse

Assumptions

- Intermediate-level knowledge of common tunneling tradecraft
 - SOCKS and reverse port forwards over C2 channel
- Entry-level knowledge of NTLM relay primitives
 - NTLM relay protections and coercion mechanisms
- Entry-level knowledge of reverse engineering
 - Nothing in-depth required here

Problem and Solution Statements



Problem

Conducting NTLM relays from command-and-control (C2) infrastructure involves several hurdles to overcome, such as the Windows kernel binding to 445/tcp by default. Existing solutions to this problem require taking noteworthy OPSEC risks.



Solution

Identify and implement a technique which results in control over port 445/tcp that is practical to leverage while operating from C2 and doesn't include OPSEC concerns of existing solutions.

Relevance of 445/tcp control

Relevance of 445/tcp control

NTLM Relay Effectiveness

- Targeted NTLM relay [still] incredibly effective in even “mature” orgs
 - HTTP → LDAP(S) for shadows creds or RBCD
 - SMB/HTTP → HTTP for AD CS ESC8, SCCM TAKEOVER-4.2,
 - SMB/HTTP → SMB for several SCCM TAKEOVER primitives
 - SMB/HTTP → MSSQL for SCCM TAKEOVER-1
 - Many many more...
- Depending on your perspective of access in the target network, relay of inbound SMB traffic could be more involved

Relevance of 445/tcp control

Hurdles while operating over C2

- Operating from a dedicated attacker machine bridged on the target network requires less preparation
 - When coercing SMB-based NTLM authentication:
 - Windows – disable 'LanmanServer' and reboot
 - Linux – simply bind to 445/tcp
- Operating from a compromised Windows host over C2 has additional challenges
 - We want some type of “reverse port forward” type functionality
 - By default, the Windows kernel is bound to 445/tcp on all network interfaces for both IPv4 and IPv6

```
C:\Users\default> netstat -ano | findstr :445
```

| | | | | |
|-----|-------------|-----------|-----------|---|
| TCP | 0.0.0.0:445 | 0.0.0.0:0 | LISTENING | 4 |
| TCP | :::445 | :::0 | LISTENING | 4 |

Existing Solutions

Existing Solutions to 445/tcp “Ownership”

- WinDivert driver interaction for traffic redirection
 - Load the signed WinDivert driver
 - “user-mode packet interception library”
 - PortBender, SharpRelay, StreamDivert, DivertTCPconn, hfwfbypass
- Custom LSA authentication provider
 - @CCob’s “lsarelayx”
 - Hook NTLM and Negotiate packages to redirect authentication requests over named pipes
- Disable ‘LanmanServer’ service and reboot
 - Change the start type of the ‘LanmanServer’ service to ‘disabled’ and reboot the host

Existing Solutions to 445/tcp “Ownership”

- OPSEC considerations for existing approaches
 - Leveraging drivers for post-exploitation
 - Potential BSOD is not an option in a lot of situations (critical infrastructure)
 - Potential single point of failure regarding detection/prevention
 - Interfacing with the driver will have additional considerations
 - Reflective DLL loading, .NET PE, unmanaged PE, PIC shellcode, BOF
 - Loading custom LSA authentication provider
 - Due to limitations of how LSA plugins work, current implementation’s DLL cannot be unloaded from LSASS until reboot occurs
 - Could affect stability of LSASS process resulting in forced reboot
 - Reboot after service start type modification
 - Triggering (or waiting for) reboots are unfortunately not an option in many red team scenarios

Reversing Drivers for New Solutions

Reversing Drivers for New Solutions

Prerequisite Notes

- Tools primarily used for analysis
 - System Informer, IDA free, WinDBG
- What is our goal?
 - Do *something* to *release* the target port without requiring a reboot, loading a driver, or loading a module into LSASS
- Where can we start?
 - `LanmanServer` can be disabled after reboot
 - Simply resetting the start type of this service will trigger reloading all necessary resources
 - Starting point for reproducing and debugging associated behavior

Reversing Drivers for New Solutions

Identifying Items of Interest

- Identify process bound to the target port
- Triage relevant loaded modules (drivers)
- Narrow down initial drivers for inspection
- Ensure consistency across modules being disassembled and debugged
 - e.g. Winindex

Reversing Drivers for New Solutions

Identifying Items of Interest

```
PS C:\> Get-NetTCPConnection -LocalPort 445 | ForEach-Object { Get-Process -Id $_.OwningProcess }
```

| Handles | NPM(K) | PM(K) | WS(K) | CPU(s) | Id | SI | ProcessName |
|---------|--------|-------|-------|----------|----|----|-------------|
| 6600 | 0 | 224 | 7424 | 9,175.72 | 4 | 0 | System |



System (4) Properties

| Name | Base address | Size | Description |
|--------------|----------------|--------|---------------------------------|
| ACPI.sys | 0xfffff8033... | 816 kB | ACPI Driver for NT |
| acpiex.sys | 0xfffff8033... | 152 kB | ACPIEx Driver |
| afd.sys | 0xfffff8033... | 660 kB | Ancillary Function Driver for . |
| afunix.sys | 0xfffff8033... | 80 kB | AF_UNIX socket provider |
| AgileVpn.sys | 0xfffff8034... | 160 kB | RAS Agile Vpn Miniport Call ... |
| ahcache.sys | 0xfffff8033... | 312 kB | Application Compatibility Ca.. |
| atapi.sys | 0xfffff8033... | 52 kB | ATAPI IDE Miniport Driver |
| ataport.SYS | 0xfffff8033... | 240 kB | ATAPI Driver Extension |
| bam.sys | 0xfffff8033... | 92 kB | BAM Kernel Driver |



ChatGPT

Absolutely, here's what is most likely to be relevant to your items of interest: `afd.sys`, `tcpip.sys`, and `netbt.sys` to network operations, specifically related to SMB/NetBIOS, followed by a few additional drivers:

```
rust
```

- `afd.sys` - Ancillary Function Driver for WinSock
Description: Crucial for Windows Socket operations, pivotal for network communication
- `tcpip.sys` - TCP/IP Protocol Driver
Description: Central to handling TCP/IP protocol operations, vital for all network
- `netbt.sys` - MBT Transport Driver
Description: Implements NetBIOS over TCP/IP necessary for legacy network operation

Reversing Drivers for New Solutions

Understanding the Binding Process

- IDA free used to do manual triage of relevant drivers in attempt to find functions associated with binding process
 - Thanks to Microsoft's symbols, several interesting functions identified by searching for "port", "socket", "bind", etc.
 - afd!WskProAPIBind
 - afd!Bind
 - afd!WskProAPISocket
 - tcpip!InspectBindEndpoint
 - tcpip!InetAcquirePort
 - (many... many more)

Reversing Drivers for New Solutions

Understanding the Binding Process

- Target VM configured to enable kernel debugging
 - Snapshotted in state of 445/tcp being unbound
 - PowerShell one-liner + hotkey to reenabale / revert efficiently

```
Set-Service -Name "lanmanserver" -StartType Automatic; Start-Service -Name "lanmanserver"
```

- Breakpoints set for driver functions of interest to inspect parameters
- Eventually led to the inspection of **tcpip!InetAcquirePort**
 - Reliably hit after reenabling 'LanmanServer' and rebinding to 445/tcp
 - Let's ensure this is associated with *our rebinding of 445/tcp...*

Reversing Drivers for New Solutions

Understanding the Binding Process

tcpip!InetAcquirePort pseudocode

```
ExAcquireResourceExclusiveLite(a1, v16);  
v68 = (unsigned __int16)__ROR2__(*a6, 8);  
v69 = IsPortInExclusion(*(_QWORD*)(a1 + 136), v68);  
if (v69 && (*(_BYTE*)(v69 + 16) & 0x12) == 2)
```

- subsequent call to **tcpip!IsPortInExclusion**
- second input parameter is of type **__int16**
 - could likely represent a TCP port number of 0-65535

WinDBG output

```
1: kd> p  
tcpip!InetAcquirePort+0xbae:  
ffff806`3e93c646 e8f9bd0100 call tcpip!IsPortInExclusion (ffff806`3e958444)  
  
1: kd> ? rdx  
Evaluate expression: 445 = 00000000`000001bd
```

- trigger enabling the 445/tcp bind by reverting + changing 'LanmanServer' start type
- step through **tcpip!IsPortInExclusion** breaks
- based on *fastcall* calling convention and target function's *prototype*, inspect RDX register value

Reversing Drivers for New Solutions

How can this help us understand the *unbinding* process?

```
[0x0] tcpip!InetAcquirePort+0xbae
[0x1] tcpip!TcpBindEndpointRequestInspectComplete+0x2cc
[0x2] tcpip!TcplControlEndpoint+0x2e9
[0x3] tcpip!TcpTIEndpointIoControlEndpointCalloutRoutine+0x74
[0x4] nt!KeExpandKernelStackAndCalloutInternal+0x78
[0x5] nt!KeExpandKernelStackAndCalloutEx+0x1d
[0x6] tcpip!TcpTIEndpointIoControlEndpoint+0x6e
[0x7] afd!WskProIRPBind+0x11e
[0x8] afd!AfdWskDispatchInternalDeviceControl+0x3c
[0x9] nt!IoCallDriver+0x55
[0xa] afd!WskProAPIBind+0x47
[0xb] srvnet!SrvNetWskOpenListenSocket+0x3ef
[0xc] srvnet!SrvNetAllocateEndpointCommon+0x34a
[0xd] srvnet!SrvNetAllocateEndpoint+0x3e02
[0xe] srvnet!SrvNetAddServedName+0x564
[0xf] srvnet!SvcXportAdd+0x14e
```

Reversing Drivers for New Solutions

Understanding the *Unbinding* Process

- What functionality is exposed related to the unbinding process?
 - Identify IOCTLs that maybe expose relevant function(s) to privileged users?
- Starting from **srvnet.sys** within the previously mentioned callstack
 - Symbols allow for easily associating similar unbinding behavior with what was seen during the binding process
- Beginning near the bottom of the call stack...

srvnet!SrvNetWskOpenListenSocket → srvnet!SrvNetWskCloseListenSocket

srvnet!SrvNetAllocateEndpoint → srvnet!SrvNetCloseEndpoint

srvnet!SrvNetAddServedName → srvnet!SrvNetDeleteServedName

```

1 __int64 __fastcall SrvNetWskCloseListenSocket(__int64 a1)
2 {
3     __int64
4     __int64
5     __int64
6     _DWORD
7     __int64
8     __int12
9     __int64

```

xrefs to SrvNetWskCloseListenSocket

| Direction | Type | Address | Text |
|-----------|------|----------------------------|-----------------------------------------------------|
| Up | p | SrvNetCloseEndpoint+F0C3 | call SrvNetWskCloseListenSocket |
| Up | o | .pdata:FFFFFF80069252A00 | RUNTIME_FUNCTION <rva SrvNetWskCloseListenSocket, \ |
| D... | p | SrvNetWskOpenListenSock... | call SrvNetWskCloseListenSocket |

```

1 __int64 __fastcall SrvNetCloseEndpoint(__int64 a1)
2 {
3     __int64 v1
4     __int64 v2
5     __int64 v3
6     bool v4;
7     __int64 v5
8     __int64 v6
9     int v7;
10    int v8;
11    __int64 v9

```

xrefs to SrvNetCloseEndpoint

| Direction | Type | Address | Text |
|-----------|------|--------------------------------------------------------|----------------------------|
| D... | o | .pdata:FFFFFF800692501D4 | RUNTIME_FUNCTION <rva SrvN |
| D... | o | .rdata:FFFFFF8006924BE94 | RUNTIME_FUNCTION <rva SrvN |
| D... | p | SrvNetAddServedName +369 | call SrvNetCloseEndpoint |
| D... | p | SrvNetAddServedName +56F | call SrvNetCloseEndpoint |
| D... | p | SrvNetCleanupDeviceExtensionPreScavengerTermination... | call SrvNetCloseEndpoint |

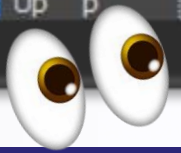
```

1 __int64 SrvNetCleanupDeviceExtensionPreScavengerTermination()
2 {
3     __int64
4     _QWORD
5     __int64
6     _QWORD
7     __int64
8     __int64
9     __int64

```

xrefs to SrvNetCleanupDeviceExtensionPreScavengerTermination

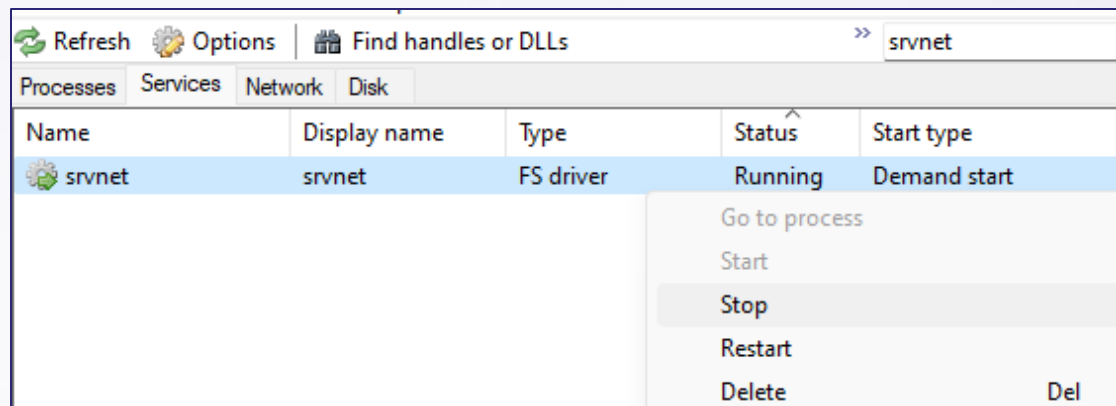
| Direction | Type | Address | Text |
|-----------|------|--------------------------|----------------------------------------------------------|
| Up | o | .pdata:FFFFFF80069252850 | RUNTIME_FUNCTION <rva SrvNetCleanupDeviceExtensionPreSc |
| Up | p | DriverUnload +6C | call SrvNetCleanupDeviceExtensionPreScavengerTermination |



Reversing Drivers for New Solutions

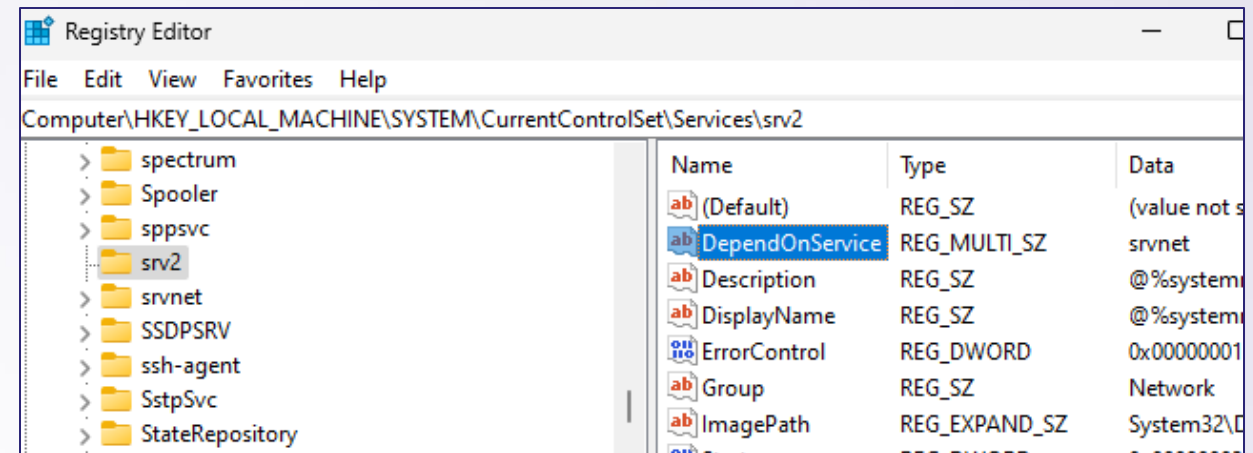
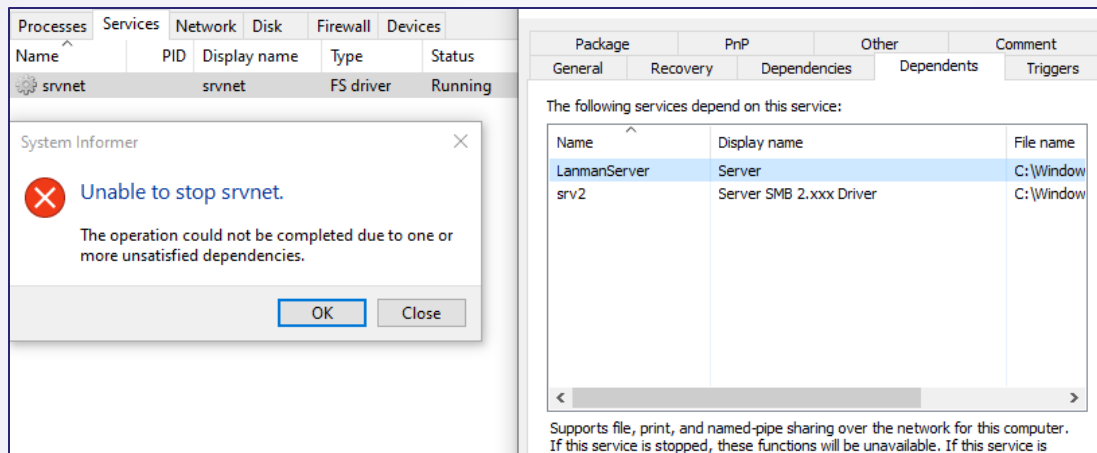
Understanding the Unbinding Process

- How can we determine if unloading this driver will reach the appropriate code path?
 1. Be a good reverse engineer and step through the disassembly until we have a thorough understanding of expected behavior
 2. Let Jesus take the wheel and start blindly disabling services



Service Dependents

- Why are we trying to stop this service?
 - Stopping a driver's service *should* call its respective unload function
- Why are service dependencies relevant?
 - MSDN - “*specifies the names of services or groups that must start before this service*”
 - In (most) default build of Windows, **srvnet** is a *dependent* of **srv2** which is a *dependent* of **LanmanServer**
 - Connecting some dots from initial interactions with LanmanServer...





Reversing Drivers for New Solutions

Understanding the Unbinding Process

- Reconfigure target services in a specific order
 1. Change start type of **LanmanServer** from *Auto Start (Trigger)* to *Disabled*
 - Triggers for this service occur often, changing this will be important
 2. Stop **LanmanServer** service
 3. Stop **srv2** service
 4. Stop **srvnet** service
 5. (optional) Hope our prayers are answered

NOTE: Potential variation in dependents listed here

Reversing Drivers for New Solutions

Understanding the Unbinding Process

tcpip! InetAcquirePort → tcpip!InetReleasePort

```
__int64 __fastcall InetReleasePort(__int64 a1, __int64 a2, __int64 a3, __int64 a4) {
    unsigned __int16 v4; // r14
    ...
    __int128 v21; // [rsp+20h] [rbp-48h] BYREF
    __int64 v22; // [rsp+30h] [rbp-38h]
    v4 = __ROR2__(a2, 8);
    v21 = 0i64;
    ...
    v13 = IsPortInExclusion((__int64 **)(a1 + 136), v4);
    if ( (unsigned __int8)IsEmptyAssignment(v12, v13) )
```

Reversing Drivers for New Solutions

Understanding the Unbinding Process

```
0: kd> g
Breakpoint 2 hit
tcpip!InetReleasePort:
ffff807`7d92a3fc 4c8bdc      mov     r11,rsp

1: kd> r
rax=ffffcf8d773ed190 rbx=ffffcf8d7a1eacb0 rcx=ffffcf8d77475000 rdx=000000000000bd01 rsi=ffffcf8d7a782770
rdi=0000000000000000 rip=ffff8077d92a3fc rsp=ffffe8bfb1ea0b8 rbp=ffffe8bfb1ea3a0 r8=ffffcf8d7a1ead28
r9=0000000000000000 r10=ffff80779cd2250 r11=ffffe8bfb1ea178 r12=0000000000000001 r13=0000000000000000
r14=ffffcf8d7a80ad98 r15=ffff807901ee040
```

```
0: kd> g
Breakpoint 1 hit
tcpip!IsPortInExclusion:
ffff807`7d918444 6690      nop

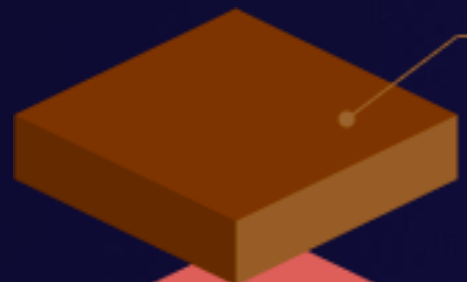
0: kd> ? rdx Evaluate expression: 445 = 00000000`000001bd
```

Reversing Drivers for New Solutions

Understanding the Unbinding Process

```
PS C:\Windows\system32> Get-NetTCPConnection -LocalPort 445
```

```
Get-NetTCPConnection : No MSFT_NetTCPConnection objects found with property 'LocalPort' equal to '445'. Verify the value of the property and retry. At line:1 char:1 + Get-NetTCPConnection -LocalPort 445 + ~~~~~ + CategoryInfo          : ObjectNotFound: (445:UInt16) [Get-NetTCPConnection], CimJobException + FullyQualifiedErrorId : CmdletizationQuery_NotFound_LocalPort,Get-NetTCPConnection
```



tcpip!InetReleasePort

Similar tcpip.sys function observed in call stack when binding to 445



...

A few afd.sys virtual function calls and tcpip.sys function calls



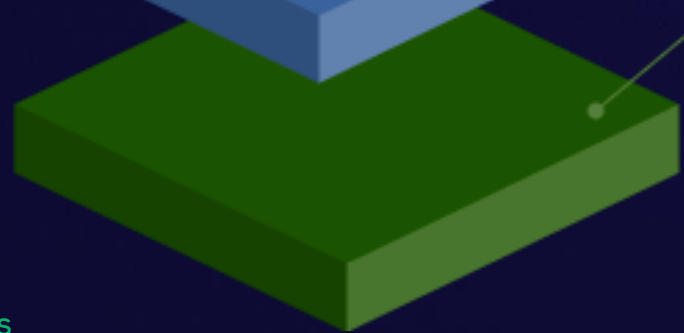
srvnet!SrvNetWskCloseListenSocket

Similar srvnet.sys function observed in call stack when binding to 445



...

A few more srvnet.sys function calls



srvnet!DriverUnload

Function called when srvnet.sys driver is unloaded

Tooling for Automation of Abuse

Tools for Automation of Abuse

- Important to remember
 - We are disabling services associated with facilitating communication via SMB
 - Tools that leverage RPC over named pipes (*ncacn_np*) will no longer work
 - If you're doing this remoting, ensure you're leveraging RPC over TCP (*ncacn_ip_tcp*)
- Bonus
 - Simply reconfiguring **LanmanServer** start type to *Auto Start (Trigger)* will result in all necessary services being reenabled for SMB to resume normal functionality
- Two implementations created to automate SCM interaction
 - Python and BOF

```
proxychains4 -q python3 smbtakeover.py atlas.lab/josh:password1@10.0.0.21 stop
```

```
[*] LanmanServer
```

```
    |--- action: starttype=Disabled
```

```
[*] LanmanServer
```

```
    |--- action: Stopped
```

```
[*] srv2
```

```
    |--- action: Stopped
```

```
[*] srvnet
```

```
    |--- action: Stopped
```

```
proxychains4 -q python3 smbtakeover.py atlas.lab/josh:password1@10.0.0.21 check
```

```
[*] LanmanServer
```

```
    |----- state:  Stopped
```

```
    |----- starttype: Disabled
```

```
    |----- path:   C:\Windows\system32\svchost.exe -k netsvcs -p
```

```
[*] srv2
```

```
    |----- state:  Stopped
```

```
    |----- starttype: Manual
```

```
    |----- path:   System32\DRIVERS\srv2.sys
```

```
[*] srvnet
```

```
    |----- state:  Stopped
```

```
    |----- starttype: Manual
```

```
    |----- path:   System32\DRIVERS\srvnet.sys
```

```
[+] 445/tcp bound: FALSE
```



```
beacon> bof_smbtakeover localhost stop
[*]
[*] ~Executing smbtakeover BOF by @zyn3rgy~
[*]
[+] host called home, sent: 15698 bytes
[+] received output:

-----STOPPING SMB FUNCTIONALITY-----
[*] LanmanServer
  |--- action: starttype=Disabled
[*] LanmanServer
  |--- action: Stopped
[*] srv2
  |--- action: Stopped
[*] srvnet
  |--- action: Stopped
-----

[+] 445/tcp bound - FALSE
```

```
beacon> bof_smbtakeover localhost start
[*]
[*] ~Executing smbtakeover BOF by @zyn3rgy~
[*]
[+] host called home, sent: 15699 bytes
[+] received output:

-----RESUME SMB FUNCTIONALITY-----
[*] LanmanServer
  |--- action: starttype=Auto
[*] LanmanServer
  |--- action: Started
-----

[+] 445/tcp bound - TRUE
```

```
beacon> bof_smbtakeover localhost check
[*]
[*] ~Executing smbtakeover BOF by @zyn3rgy~
[*]
[+] host called home, sent: 15699 bytes
[+] received output:

-----CHECKING SERVICES-----

[*] LanmanServer
  |----- state: Running
  |----- starttype: AUTO
  |----- path: System32\DRIVERS\svrnet.sys
[*] srv2
  |----- state: Running
  |----- starttype: MANUAL
  |----- path: System32\DRIVERS\svrnet.sys
[*] srvnet
  |----- state: Running
  |----- starttype: MANUAL
  |----- path: System32\DRIVERS\svrnet.sys
-----

[+] 445/tcp bound - TRUE
```

[Existing] Tools for Automation of Abuse

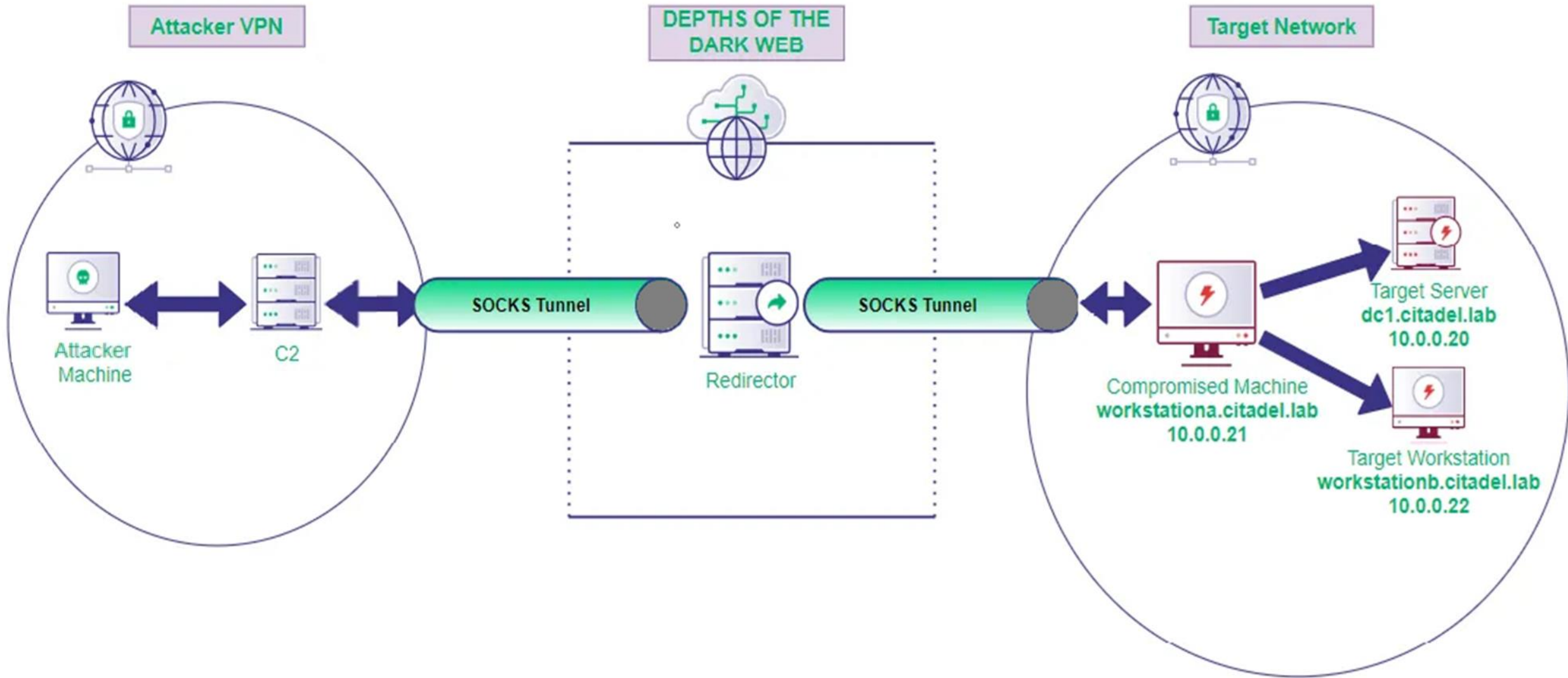


- `sc.exe`
 - Stop
 1. `sc config LanmanServer start= disabled`
 2. `sc stop LanmanServer`
 3. `sc stop srv2`
 4. `sc stop srvnet`
- `wmiexec-pro.py`
 - Stop
 1. `wmiexec-pro.py lab.local/admin@target.lab.local service -action disable -service-name "LanmanServer"`
 2. `wmiexec-pro.py lab.local/admin@target.lab.local service -action stop -service-name "LanmanServer"`
 3. `wmiexec-pro.py lab.local/admin@target.lab.local service -action stop -service-name "srv2"`
 4. `wmiexec-pro.py lab.local/admin@target.lab.local service -action disable -service-name "srvnet"`
 - Check
 1. `wmiexec-pro.py lab.local/admin@target.lab.local service -action getinfo -service-name "srvnet"`

Demonstration of Practical Abuse

(shoutout to @garrfoster and @_Mayyhem)

First, some review of tunneling...



root@WORKSTATION:~/github/sccmhunter

defaultuser@WORKSTATION:~/github/py-smbtakeover

root@WORKSTATION:~/github/sccmhunter

(venv) [2024-04-23 6:15:59] root ~/github/sccmhunter

#

Cobalt Strike

Cobalt Strike View Payloads Attacks Site Management Reporting Help S

| ex... | lis... | us... | co... | no... | pr... | pid | arch | last | sleep | |
|-------|--------|-------|-------|-------|-------|-------|--------|------|-------|-------------|
| 20... | 19... | H... | la... | CL... | co... | ba... | 5980 | x64 | 87ms | Interactive |
| 20... | 19... | H... | la... | CL... | co... | ba... | 102... | x64 | 9ms | 4 secon... |

Event Log X Proxy Pivots X Beacon 192.168.57.5@5980 X

| user | com... | pid | type | sock... | port | client | fhost | fport |
|---------|---------|-------|--------|---------|------|--------|-------|-------|
| laba... | CLIE... | 10232 | SOC... | NoA... | 9050 | | | |

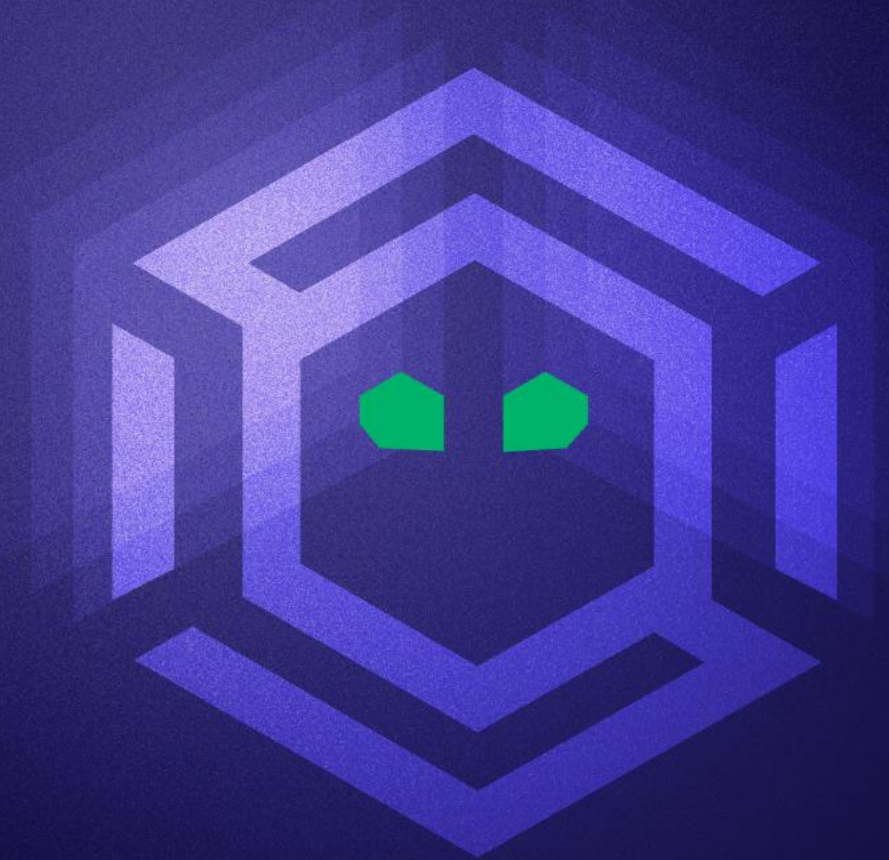
Stop Tunnel Help

Conclusion

- Simple interactions with SCM can result in 445/tcp being unbound by Windows kernel
 - Remotely conducting these actions using RCP over TCP is beneficial (connectivity)
- BOF and Python automation of abuse to be released
 - Existing tools to interact with SCM should do the trick though
- Provides “lower touch” solution to controlling inbound 445/tcp traffic for NTLM relay and other offensive techniques



Thank you



Nick Powers | [@zyn3rgy](#)