Is anti virus installed

```
using System;
using System.Management;


class AntivirusCheck
{
   static void Main()
   {
      Console.WriteLine("1. Is Antivirus Installed?");
      try
      {
         string query = "SELECT * FROM AntiVirusProduct";
         ManagementObjectSearcher searcher = new
ManagementObjectSearcher("root\\SecurityCenter2", query);
         bool found = false;
         foreach (ManagementObject obj in searcher.Get())
         {
            found = true;
            Console.WriteLine("Yes - " + obj["displayName"]);
         }
         if (!found)
            Console.WriteLine("No");
      }
      catch
      {
         Console.WriteLine("Error: Cannot determine.");
      }
   }
}
```

Last update date of os

```
using System;
using System.Management;

class Program
{
   static void Main()
   {
      Console.WriteLine("5. Last OS Update Date:");
      GetLastOSUpdateDate();
   }

   static void GetLastOSUpdateDate()
   {
      try
      {
         ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT * FROM
Win32_QuickFixEngineering");
         DateTime latestUpdateDate = DateTime.MinValue;
         string latestKB = "";

         foreach (ManagementObject update in searcher.Get())
         {
            string installedOn = update["InstalledOn"]?.ToString();

            if (!string.IsNullOrEmpty(installedOn) && DateTime.TryParse(installedOn, out DateTime
updateDate))
            {
               if (updateDate > latestUpdateDate)
               {
                  latestUpdateDate = updateDate;
```

```
                latestKB = update["HotFixID"]?.ToString();
            }
        }
    }

    if (latestUpdateDate != DateTime.MinValue)
    {
        Console.WriteLine($"Latest OS Update: KB{latestKB} installed on
{latestUpdateDate.ToShortDateString()}");
    }
    else
    {
        Console.WriteLine("No update information found.");
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error retrieving update date: " + ex.Message);
  }
 }
}
```

Major feature updates

```
using System;
using Microsoft.Win32;

class Program
{
  static void Main()
  {
```

```
    Console.WriteLine("Checking Windows Major Feature Update Info:");
    CheckWindowsFeatureUpdate();
  }

  static void CheckWindowsFeatureUpdate()
  {
    try
    {
      using (RegistryKey key =
Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\Windows NT\CurrentVersion"))
      {
        if (key != null)
        {
          string productName = key.GetValue("ProductName")?.ToString();
          string releaseId = key.GetValue("ReleaseId")?.ToString(); // For older versions (till 2004)
          string displayVersion = key.GetValue("DisplayVersion")?.ToString(); // For newer
versions (21H1+)
          string buildNumber = key.GetValue("CurrentBuildNumber")?.ToString();
          string installDate = key.GetValue("InstallDate")?.ToString();

          Console.WriteLine($"Edition: {productName}");
          Console.WriteLine($"Build Number: {buildNumber}");

          if (!string.IsNullOrEmpty(displayVersion))
            Console.WriteLine($"Feature Update Version: {displayVersion}");  // Like "22H2"
          else if (!string.IsNullOrEmpty(releaseId))
            Console.WriteLine($"Feature Update Version: {releaseId}");  // Like "2004"

          if (!string.IsNullOrEmpty(installDate))
          {
            // Convert InstallDate (Unix timestamp) to readable format
            long unixTime = Convert.ToInt64(installDate);
```

```
                DateTime installDateTime =
DateTimeOffset.FromUnixTimeSeconds(unixTime).LocalDateTime;
                Console.WriteLine($"Feature Update Install Date: {installDateTime}");
            }
        }
        else
        {
            Console.WriteLine("Could not access registry path.");
        }
    }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error retrieving Windows feature update info: " + ex.Message);
    }
  }
}
```

User has root access

```
using System;
using System.Security.Principal;

class Program
{
    static void Main()
    {
        Console.WriteLine("6. Checking if user has root/admin access...");
        CheckAdmin();
    }
```

```
    static void CheckAdmin()
    {
        bool isAdmin = new WindowsPrincipal(WindowsIdentity.GetCurrent())
                    .IsInRole(WindowsBuiltInRole.Administrator);

        if (isAdmin)
        {
            Console.WriteLine("No (Root Access Given)");  // User has admin rights
        }
        else
        {
            Console.WriteLine("Yes (No Admin Rights)");   // User does NOT have admin rights
        }
    }
}
```

Licensed os found

```
using System;
using Microsoft.Win32;

class Program
{
    static void Main()
    {
        Console.WriteLine("7. Checking if Licensed OS is Installed...");
        CheckLicense();
    }
```

```
    static void CheckLicense()
    {
        try
        {
            using (RegistryKey key = Registry.LocalMachine.OpenSubKey(@"Software\Microsoft\Windows NT\CurrentVersion"))
            {
                if (key != null)
                {
                    object productId = key.GetValue("DigitalProductId");
                    if (productId != null)
                    {
                        Console.WriteLine("Yes (Digital License Found)");
                    }
                    else
                    {
                        Console.WriteLine("No (Digital License Not Found)");
                    }
                }
                else
                {
                    Console.WriteLine("Registry key not found.");
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking license: " + ex.Message);
        }
    }
}
```

Usb storage blocked or not

```
using System;
using Microsoft.Win32;


class Program
{
   static void Main()
   {
      Console.WriteLine("8. Checking if USB Storage Media is Blocked...");
      CheckUSBBlock();
   }


   static void CheckUSBBlock()
   {
      try
      {
         using (RegistryKey key =
Registry.LocalMachine.OpenSubKey(@"SYSTEM\CurrentControlSet\Services\USBSTOR"))
         {
            if (key != null)
            {
               int startValue = (int)key.GetValue("Start", 3); // Default is 3 (enabled), 4 = disabled
               string result = startValue == 4 ? "Yes (Blocked)" : "No (Not Blocked)";
               Console.WriteLine("USB Storage Media Blocked: " + result);
            }
            else
            {
               Console.WriteLine("USBSTOR registry key not found.");
            }
         }
      }
```

```
        catch (Exception ex)
        {
            Console.WriteLine("Error checking USB block: " + ex.Message);
        }
    }
}
```

Ip tables are used in system

```
using System;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        Console.WriteLine("9. Checking if IP Tables / Windows Firewall is Enabled...");
        CheckFirewallStatus();
    }

    static void CheckFirewallStatus()
    {
        try
        {
            Process process = new Process();
            process.StartInfo.FileName = "netsh";
            process.StartInfo.Arguments = "advfirewall show allprofiles";
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.CreateNoWindow = true;
            process.Start();
```

```
            string output = process.StandardOutput.ReadToEnd();
            bool isEnabled = output.Contains("State ON");

            Console.WriteLine("IP Tables / Windows Firewall Enabled: " + (isEnabled ? "Yes" : "No"));
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking firewall: " + ex.Message);
        }
    }
}
```

No of usb connected

```
using System;
using System.Management;

class Program
{
    static void Main()
    {
        Console.WriteLine("10. Checking number of connected USB Pen Drives...");
        CountUSBDrives();
    }

    static void CountUSBDrives()
    {
```

```
        try
        {
            int usbCount = 0;
            ManagementObjectSearcher searcher = new ManagementObjectSearcher(
                "SELECT * FROM Win32_DiskDrive WHERE InterfaceType='USB'"
            );

            foreach (ManagementObject mo in searcher.Get())
            {
                usbCount++;
            }

            Console.WriteLine($"Number of USB Pen Drives Connected: {usbCount}");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking USB devices: " + ex.Message);
        }
    }
}
```

traces of smartphone

```
using System;
using Microsoft.Win32;

class Program
{
    static void Main()
    {
        Console.WriteLine("12. Checking traces of previously connected smartphone/dongle...");
```

```
bool traceFound = false;


try
{
    using (RegistryKey usbRoot =
Registry.LocalMachine.OpenSubKey(@"SYSTEM\CurrentControlSet\Enum\USB"))
    {
        if (usbRoot != null)
        {
            foreach (string deviceKey in usbRoot.GetSubKeyNames())
            {
                if (deviceKey.ToLower().Contains("android") ||
                    deviceKey.ToLower().Contains("mtp") ||
                    deviceKey.ToLower().Contains("modem") ||
                    deviceKey.ToLower().Contains("iphone") ||
                    deviceKey.ToLower().Contains("samsung") ||
                    deviceKey.ToLower().Contains("huawei") ||
                    deviceKey.ToLower().Contains("vivo") ||
                    deviceKey.ToLower().Contains("mobile"))
                {
                    traceFound = true;
                    Console.WriteLine("Trace Found: " + deviceKey);
                    break;
                }

                // Also check inside subkeys
                using (RegistryKey subDevice = usbRoot.OpenSubKey(deviceKey))
                {
                    foreach (string sub in subDevice.GetSubKeyNames())
                    {
```

```
                    if (sub.ToLower().Contains("android") ||
                        sub.ToLower().Contains("mtp") ||
                        sub.ToLower().Contains("modem") ||
                        sub.ToLower().Contains("iphone") ||
                        sub.ToLower().Contains("mobile"))
                    {
                        traceFound = true;
                        Console.WriteLine($"Trace Found: {deviceKey}\\{sub}");
                        break;
                    }
                }
            }

            if (traceFound) break;
        }
    }

    Console.WriteLine("Traces of Smartphone/Dongle Connection: " + (traceFound ? "Yes (Found)"
: "No (Not Found)"));
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error checking registry for device traces: " + ex.Message);
  }
 }
}
```

if telnet installed

```
using System;
```

```
using System.IO;

class Program
{
   static void Main()
   {
      Console.WriteLine("13. Checking if Telnet is installed...");

      try
      {
         string telnetPath = Environment.SystemDirectory + @"\telnet.exe";
         bool exists = File.Exists(telnetPath);

         Console.WriteLine("Telnet Installed: " + (exists ? "No (Installed)" : "Yes (Not Installed)"));
      }
      catch (Exception ex)
      {
         Console.WriteLine("Error checking Telnet: " + ex.Message);
      }
   }
}
```

Auto play disabled

```
using System;
using Microsoft.Win32;

class Program
{
   static void Main()
   {
```

```
        Console.WriteLine("14. Checking if Autoplay is disabled...");


        try
        {
            using (RegistryKey key =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Policies\Explore
r"))
            {
                object value = key?.GetValue("NoDriveTypeAutoRun");


                if (value != null && (int)value == 255)
                    Console.WriteLine("Autoplay Disabled: Yes");
                else
                    Console.WriteLine("Autoplay Disabled: No");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking Autoplay: " + ex.Message);
        }
    }
}
```

Ntp running

```
using System;
using System.Diagnostics;


class Program
{
    static void Main()
```

```
    {
        Console.WriteLine("15. Checking if NTP (w32time) service is installed and running...");

        try
        {
            Process process = new Process();
            process.StartInfo.FileName = "sc";
            process.StartInfo.Arguments = "query w32time";
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.CreateNoWindow = true;
            process.Start();

            string output = process.StandardOutput.ReadToEnd();

            bool isRunning = output.Contains("RUNNING");
            bool isInstalled = output.Contains("SERVICE_NAME");

            if (isInstalled)
                Console.WriteLine("NTP (w32time) Installed and Running: " + (isRunning ? "Yes" : "No
(Installed but not running)"));
            else
                Console.WriteLine("NTP (w32time) Service Not Installed.");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking NTP: " + ex.Message);
        }
    }
}
```

Temp partition is configured

```
using System;
using System.IO;

class Program
{
   static void Main()
   {
      Console.WriteLine("16. Checking if Temp Partition is configured (i.e., on a different drive)...");

      try
      {
         // Get TEMP path
         string tempPath = Path.GetTempPath();

         // Get root of system drive (usually C:\)
         string systemDrive =
Path.GetPathRoot(Environment.GetFolderPath(Environment.SpecialFolder.System));

         // Get root of TEMP directory (could be D:\TEMP or something else)
         string tempDrive = Path.GetPathRoot(tempPath);

         bool isSeparate = !string.Equals(systemDrive, tempDrive,
StringComparison.OrdinalIgnoreCase);

         Console.WriteLine("Temp Partition Configured (Different Drive): " + (isSeparate ? "Yes" :
"No"));
      }
      catch (Exception ex)
      {
         Console.WriteLine("Error checking temp partition: " + ex.Message);
```

```
        }
    }
}
```

Remote acess disabled

```
using System;
using Microsoft.Win32;

class Program
{
    static void Main()
    {
        Console.WriteLine("17. Checking if Remote Access (Remote Desktop) is disabled...");

        try
        {
            using (RegistryKey key =
Registry.LocalMachine.OpenSubKey(@"SYSTEM\CurrentControlSet\Control\Terminal Server"))
            {
                int value = (int)(key?.GetValue("fDenyTSConnections", 1) ?? 1);
                bool isDisabled = value == 1;

                Console.WriteLine("Remote Access Disabled: " + (isDisabled ? "Yes" : "No"));
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error checking remote access: " + ex.Message);
        }
```

```
    }
}
```

Acess to system or network restricted

```
using System;
using System.Net.NetworkInformation;

namespace SecurityCheck
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("21. Access to System or Network is Restricted");

            try
            {
                Ping ping = new Ping();
                PingReply reply = ping.Send("8.8.8.8", 3000); // Google DNS

                if (reply.Status == IPStatus.Success)
                    Console.WriteLine("Answer: No (System has network access)");
                else
                    Console.WriteLine("Answer: Yes (System has no network access)");
            }
            catch
            {
                Console.WriteLine("Answer: Yes (Ping failed - network likely restricted)");
            }
```

```
        }
    }
}
```

admin rights with iso

```
using System;
using System.Security.Principal;

namespace SecurityCheck
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("22. Admin Rights is with ISO");

            WindowsIdentity identity = WindowsIdentity.GetCurrent();
            WindowsPrincipal principal = new WindowsPrincipal(identity);

            if (principal.IsInRole(WindowsBuiltInRole.Administrator))
                Console.WriteLine("Answer: Yes (Admin rights present)");
            else
                Console.WriteLine("Answer: No (Admin rights not present)");
        }
    }
}
```

default share is disabled

```
using System;
using System.Diagnostics;

namespace SecurityCheck
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("25. Default Share is Disabled");

            Process process = new Process();
            process.StartInfo.FileName = "net";
            process.StartInfo.Arguments = "share";
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.CreateNoWindow = true;
            process.Start();

            string output = process.StandardOutput.ReadToEnd();
            process.WaitForExit();

            if (output.Contains("C$") || output.Contains("ADMIN$"))
                Console.WriteLine("Answer: No (Default shares like C$ or ADMIN$ are enabled)");
            else
                Console.WriteLine("Answer: Yes (Default shares are disabled)");
        }
    }
}
```

usb access is blocked

```
using System;
using Microsoft.Win32;

namespace SecurityCheck
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("26. USB Storage Media Access Was Blocked");


            try
            {
                string keyPath = @"SYSTEM\CurrentControlSet\Services\USBSTOR";
                RegistryKey key = Registry.LocalMachine.OpenSubKey(keyPath);


                if (key != null)
                {
                    object value = key.GetValue("Start");
                    if (value != null && Convert.ToInt32(value) == 4)
                        Console.WriteLine("Answer: Yes (USB access is blocked)");
                    else
                        Console.WriteLine("Answer: No (USB access is not blocked)");
                }
                else
                {
                    Console.WriteLine("Answer: Not Found (Key doesn't exist)");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }
```

```
}
```

shared folder configured

```
using System;
using System.Diagnostics;

namespace SecurityCheck
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("27. Shared Folders Are Not Configured");

         Process process = new Process();
         process.StartInfo.FileName = "net";
         process.StartInfo.Arguments = "share";
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.CreateNoWindow = true;
         process.Start();

         string output = process.StandardOutput.ReadToEnd();
         process.WaitForExit();

         int count = 0;
         string[] lines = output.Split('\n');
         foreach (var line in lines)
         {
            if (!line.Contains("Share name") && !line.Contains("------") && line.Trim().Length > 0)
               count++;
```

```
        }

        // Usually default shares are 2-3 (ADMIN$, C$, etc.)
        if (count > 3)
            Console.WriteLine("Answer: No (Custom shared folders found)");
        else
            Console.WriteLine("Answer: Yes (Only default shares present)");
    }
  }
}
```

Prohibited software found or not

```
using System;
using Microsoft.Win32;

namespace SecurityCheck
{
  class Program
  {
    static void Main(string[] args)
    {
        Console.WriteLine("30. Prohibited Software Not Installed");

        string[] badApps = { "utorrent", "torrent", "cheat engine", "bluestacks", "vmware",
"virtualbox" };
        bool found = false;

        string[] registryKeys = {
            @"SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall",
```

```
            @"SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall"
        };

        foreach (string keyPath in registryKeys)
        {
            using (RegistryKey key = Registry.LocalMachine.OpenSubKey(keyPath))
            {
                if (key == null) continue;

                foreach (string subkeyName in key.GetSubKeyNames())
                {
                    using (RegistryKey subkey = key.OpenSubKey(subkeyName))
                    {
                        string displayName = subkey?.GetValue("DisplayName")?.ToString() ?? "";
                        if (badApps.Any(b => displayName.ToLower().Contains(b)))
                        {
                            Console.WriteLine("Answer: No (Prohibited app found: " + displayName + ")");
                            found = true;
                            break;
                        }
                    }
                    if (found) break;
                }
            }
            if (found) break;
        }

        if (!found)
            Console.WriteLine("Answer: Yes (No prohibited software found)");
    }
}
```

```
}
```

Wifi disabled

```
using System;
using System.Diagnostics;

namespace SecurityCheck
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("31. Wi-Fi Disabled");

            try
            {
                Process process = new Process();
                process.StartInfo.FileName = "netsh";
                process.StartInfo.Arguments = "interface show interface";
                process.StartInfo.RedirectStandardOutput = true;
                process.StartInfo.UseShellExecute = false;
                process.StartInfo.CreateNoWindow = true;
                process.Start();

                string output = process.StandardOutput.ReadToEnd();
                process.WaitForExit();

                bool wifiFound = false;
                bool wifiEnabled = false;

                string[] lines = output.Split('\n');
```

```
            foreach (string line in lines)
            {
                if (line.ToLower().Contains("wi-fi"))
                {
                    wifiFound = true;
                    if (line.ToLower().Contains("enabled") || line.ToLower().Contains("connected"))
                    {
                        wifiEnabled = true;
                        break;
                    }
                }
            }

            if (!wifiFound)
                Console.WriteLine("Answer: Yes (Wi-Fi not found, assumed disabled)");
            else if (wifiEnabled)
                Console.WriteLine("Answer: No (Wi-Fi is enabled)");
            else
                Console.WriteLine("Answer: Yes (Wi-Fi is disabled)");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
  }
}
```

Bluetooth enable disable

```
using System;
using System.Management;
```

```
namespace SecurityCheck
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("32. Bluetooth Disabled");

            try
            {
                bool foundBluetooth = false;
                bool anyEnabled = false;

                ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT * FROM
Win32_PnPEntity");
                foreach (ManagementObject obj in searcher.Get())
                {
                    string name = obj["Name"]?.ToString()?.ToLower() ?? "";
                    string status = obj["Status"]?.ToString() ?? "";

                    if (name.Contains("bluetooth"))
                    {
                        foundBluetooth = true;
                        if (status.ToLower() == "ok")
                        {
                            anyEnabled = true;
                            break;
                        }
                    }
                }
```

```
            if (!foundBluetooth)
                Console.WriteLine("Answer: Yes (Bluetooth not found)");
            else if (anyEnabled)
                Console.WriteLine("Answer: No (Bluetooth is enabled)");
            else
                Console.WriteLine("Answer: Yes (Bluetooth is disabled)");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
  }
}
```