DRDC | RDDC
technologysciencetechnologie

Network Asset Identification

Daniel O'Leary
TRM

Prepared by:
Daniel O'Leary
TRM
Task ID: 0004
Version 1.0.1
PSPC Contract Number: W7714-176208
Technical Authority: Jonathan Risto, Engineer
Contractor's date of publication: February 2019

Defence Research and Development Canada
Contract Report
DRDC-RDDC-2019-C267
November 2019

Canada

IMPORTANT INFORMATIVE STATEMENTS

This document was reviewed for Controlled Goods by Defence Research and Development Canada using the Schedule to the *Defence Production Act*.

Disclaimer: This document is not published by the Editorial Office of Defence Research and Development Canada, an agency of the Department of National Defence of Canada but is to be catalogued in the Canadian Defence Information System (CANDIS), the national repository for Defence S&T documents. Her Majesty the Queen in Right of Canada (Department of National Defence) makes no representations or warranties, expressed or implied, of any kind whatsoever, and assumes no liability for the accuracy, reliability, completeness, currency or usefulness of any information, product, process or material included in this document. Nothing in this document should be interpreted as an endorsement for the specific use of any tool, technique or process examined in it. Any reliance on, or use of, any information, product, process or material included in this document is at the sole risk of the person so using it or relying on it. Canada does not assume any liability in respect of any damages or losses arising out of or in connection with the use of, or reliance on, any information, product, process or material included in this document.

© Her Majesty the Queen in Right of Canada (Department of National Defence), 2019

© Sa Majesté la Reine en droit du Canada (Ministère de la Défense nationale), 2019

Network Asset Identification

Task ID: 0004

Daniel O'Leary

Feb 9, 2019

Version 1.0.1

REVISION HISTORY			
DATE	**VERSION**	**DESCRIPTION**	**AUTHOR**
Aug 2018	1.0		Daniel O'Leary
Jan 2019	1.0.1	Reconfigure platform methods.	Daniel O'Leary
Feb 2019	1.0.1f	Updated based on feedback from DRDC	Daniel O'Leary

Table of Contents

1 INTRODUCTION 3
1.1 Abstract/ Abstrait 3
1.2 PROJECT SCOPE 4
1.3 REFERENCES 4
1.4 ASSUMPTIONS 5
2 Overview 6
2.1 Fingerprinting Methods 6
2.1.1 Passive Fingerprinting 7
2.1.2 Active fingerprinting 7
2.2 Collection methods 7
2.3 Characteristics of a Strong Device Fingerprinting Solution 8
2.4 Clock Skew 8
3 Detailed Analysis 10
3.1 Collectible Parameters Recommended to be used 10
3.1.1 Windows 10
3.1.1.1 IP address 10
3.1.1.2 MAC address 13
3.1.1.3 Hostname/Computername 14
3.1.1.4 Open ports 15
3.1.1.5 Operating system unique identifiers 16
3.1.1.6 Hardware 20
3.1.2 MacOS 23
3.1.2.1 IP address 23
3.1.2.2 MAC address 23
3.1.2.3 Hostname/Computername 23
3.1.2.4 Open ports 24
3.1.2.5 Operating system unique identifiers 24
3.1.2.6 Hardware 26
3.1.3 Linux 28
3.1.3.1 IP address 28
3.1.3.2 MAC address 28
3.1.3.3 Hostname/Computername 29
3.1.3.4 Open ports 29
3.1.3.5 Operating system unique identifiers 30
3.1.3.6 Hardware 32
3.1.4 Accuracy/Assurance/Confidence Matrix 34
3.1.5 How to improve confidence level 35
3.1.6 Special Cases 35
3.1.7 3rd Party Tools 36
3.2 Code Examples 36
3.2.1 Example 1 36
3.2.2 Example 2 – Various command line tools per platform. 36
3.2.2.1 Windows 36
3.2.2.2 Linux 37
3.2.2.3 MacOS 38
4 Conclusion 40

1 INTRODUCTION

1.1 ABSTRACT/ ABSTRAIT

Background/Contexte - The Cyber Operations and Signals Warfare (COSW) section at Defence Research and Development Canada (DRDC) has been conducting significant work in the area of automated Computer Network Defence (CND). To accurately identify problems in the network, and to offer remediation techniques, requires that a computing system within the environment be uniquely identified by the systems in question. Current methods of identifying a system through parameters such as MAC address, IP address, or hostname have limitations due to the ability to modify these parameters. The goal is to identify a list of parameters that can be collected about a system, that when used together (either in full or in part), can ensure the unique identification of a system in question.

La section Guerre des transmissions et des cyber-opérations (COSW) à Recherche et développement pour la défense Canada (DRDC) a mené d'importants travaux dans le domaine de la défense automatisée de réseau informatique (DND). Pour identifier avec précision les problèmes du réseau et proposer des techniques de correction, un système informatique de l'environnement doit être identifié de manière unique par les systèmes en question. Les méthodes actuelles d'identification d'un système via des paramètres tels que l'adresse MAC, l'adresse IP ou le nom d'hôte ont des limites en raison de la possibilité de modifier ces paramètres. L'objectif est d'identifier une liste de paramètres pouvant être collectés sur un système, qui, utilisés ensemble (en tout ou en partie), peuvent assurer l'identification unique du système en question.

Results/Résultats – The results show that the combination of the various collectable parameters from Windows, Mac OS, and Linux platforms can be used to uniquely identify that device with a high level of confidence. The results indicate that while some are modifiable through regular device use and updates, that many only change when the physical hardware changes. It also shows that various methods exist to collect and store these parameters for continuous assurance of unique matches.

Les résultats montrent que la combinaison des divers paramètres collectables des plates-formes Windows, Mac OS et Linux peut être utilisée pour identifier de manière unique ce périphérique avec un niveau de confiance élevé. Les résultats indiquent que, même si certains sont modifiables par le biais de l'utilisation régulière de périphériques et de mises à jour, beaucoup ne changent que lorsque le matériel physique change. Il montre également qu'il existe différentes méthodes pour collecter et stocker ces paramètres afin de garantir en permanence des correspondances uniques.

Conclusion - By using the documented identifiers, you should be able to develop a system with a near 100% assurance level of uniqueness without requiring manual intervention.

En utilisant les identifiants documentés, vous devriez être capable de développer un système avec un niveau de garantie d'unicité proche de 100% sans nécessiter d'intervention manuelle.

1.2 PROJECT SCOPE

To provide resources that will perform analysis and provide recommendations based on the information available from various computing systems to accurately and uniquely identify the computing asset. This will include identifying the parameters collectible from the system type in question, limitations on collection methods and reliability of the parameter in question

1.3 REFERENCES

1. Criteria: unique identification of windows computers
    a. https://community.spiceworks.com/topic/1980675-how-to-uniquely-identify-windows-machine
    b. https://www.nextofwindows.com/the-best-way-to-uniquely-identify-a-windows-machine
2. Criteria: unique identification of cloned windows computers
    a. https://www.experts-exchange.com/questions/28696079/Cloned-PC's-and-Windows-GUIDS-How-to-change-Unique-Identifier.html
    b. https://support.microsoft.com/en-ca/help/314828/the-microsoft-policy-for-disk-duplication-of-windows-installations
3. Criteria: unique identification of mac os machines
    a. https://stackoverflow.com/questions/48116377/how-to-get-a-unique-id-of-a-mac-machine-in-2018
4. Criteria: Unique Identification of linux computers
    a. https://www.freedesktop.org/software/systemd/man/machine-id.html
5. Cross Platform Utilities:
    a. https://quicksetcloud.com/2018/06/29/device-fingerprinting-across-home-networks/
    b. https://www.npmjs.com/package/node-machine-id
    c. https://support.foundry.com/hc/en-us/articles/205762562-Q100002-What-is-the-System-ID-and-how-do-I-find-it-
    d. https://stackoverflow.com/questions/49488624/how-to-get-a-computer-specific-id-number-using-java
    e. https://www.networkworld.com/article/3287927/internet-of-things/identifying-the-internet-of-things-one-device-at-a-time.html
    f. https://www.centritechnology.com/product-iot-data-security/device-identity/
    g. https://oroboro.com/unique-machine-fingerprint/
6. https://en.wikipedia.org/wiki/Clock_skew
7. https://github.com/jackspirou/clientjs
8. http://clientjs.org/
9. https://nmap.org/book/osdetect-usage.html
10. https://nmap.org/book/man-os-detection.html
11. https://www.sans.org/reading-room/whitepapers/tools/os-application-fingerprinting-techniques-1891
12. http://www.scitepress.org/Papers/2017/63757/63757.pdf
13. https://github.com/jackspirou/clientjs
14. https://support.treasuredata.com/hc/en-us/articles/360000691587-Device-Fingerprint-with-JS-SDK
15. https://www.darkwavetech.com/index.php/device-fingerprint-blog
16. https://fingerbank.org/
17. https://clearcode.cc/blog/device-fingerprinting/

18. https://www.quora.com/What-makes-a-computer-unique-is-it-the-HDD-If-it-is-does-then-removing-it-and-putting-it-into-another-machine-make-that-machine-same-as-the-original-one
19. https://www.microsoft.com/Licensing/servicecenter/Help/FAQDetails.aspx?id=201#213
20. https://blogs.msdn.microsoft.com/oldnewthing/20180131-00/?p=97945
21. https://oroboro.com/unique-machine-fingerprint/
22. https://security.stackexchange.com/questions/157818/spoofing-a-guid
23. https://unix.stackexchange.com/questions/395331/is-machine-id-a-uuid

1.4 ASSUMPTIONS

- Access to the devices either directly or via the network.
- An administrator level account/access is available.
- Both managed and non-managed devices.
- Systems are not locked down to the point that collection is blocked.
- Firewalls will allow the required access to target systems.
- Obfuscating technology is not being used.
- Virtual instances and clones have all been properly prepared using steps such as ‘sysprep’ on Windows.

2 Overview

Device fingerprinting is the process of reading and measuring various data about a device, such as IP Address, MAC address, open ports, OS unique identifiers, hardware unique identifiers, screen size, installed fonts, and plug-ins, and calculating the degree to which the combination of them together are unique.

For example, the combination of a specific IP Address, MAC address, screen size, Flash plugin, and open ports, as well as the fonts that are installed on the computer might be enough to provide identification. Many other combinations can be used. It is a balance of access to the parameters as well as impact on the system and degree of uniqueness that will drive which parameters you will collect and maintain.

For this project, the following mandatory collectible parameters will used to uniquely identify a networked computing system:

- IP address
- MAC address
- Hostname
- Open ports
- Operating system unique identifiers
- Hardware unique identifiers, such as motherboard, bios, cpuid, and various peripherals(Each one is counted towards the twelve)

Along with potentially:

- Screensize
- Installed fonts
- Installed software

The following operating systems will be included:

- Windows 7 professional workstation
- Windows 10 professional workstation
- Ubuntu linux system
- MacOS system

We will test each parameter for all possible collection methods (e.g. command line, wmi, remote command, agent, OSS) and provide an accuracy score on each of the selected and tested parameters on a scale from 0 to 100.

Any of these parameters on their own will not ensure uniqueness for the purpose of identification, but a combination of various parameters will increase that assurance level. But what happens when one of the parameters change? How do you still ensure the uniqueness of the asset? The recommended approach is the first time you collect a target device`s parameters, is to store them and then create a GUID for the device and store it on that device as well. On subsequent identification passes, you would check both the GUID and the list of collected parameters to see if the parameters have changed. For that GUID, you would then make a decision based on which ones changed that the device is still the same. If you decide it is the same device, you would store the changed parameters against that GUID, or if not the same then generate a new GUID and store all as if it was a new device.

2.1 FINGERPRINTING METHODS

Fingerprinting methods range from passive to active.

2.1.1 Passive Fingerprinting

Passive fingerprinting refers to techniques which do not involve the obvious querying of the client machine. These methods rely upon precise classification of such factors as the client's TCP/IP configuration, OS fingerprint, IEEE 802.11 (wireless) settings and hardware clock skew.

In many cases, OSS managed systems can deploy software agents to perform more in-depth fingerprinting. This could cross the boundary into the active fingerprinting realm.

2.1.2 Active fingerprinting

Active fingerprinting assumes the client will tolerate some degree of invasive querying. The most active method is installation of executable code directly on the client machine. Such code may have access to attributes not typically available by other means, such as the MAC address, or other unique serial numbers assigned to the machine hardware. Such data is useful for fingerprinting by programs that employ digital rights management.

2.2 COLLECTION METHODS

Various techniques can be used to collect the information needed to accurately fingerprint a device. These include:

- Command Line – this uses simple built in commands to collect information such as ifconfig on Linux and OSX, system_profiler on OSX and ipconfig and systeminfo on Windows.
- Windows Management Instrumentation (WMI) – is Microsoft's implementation of Web Based Enterprise Management (WBEM) built on the industry standard Common Information Model (CIM). It can be accessed from PowerShell, remote management, custom code, OSS agents, and more. Information to collect can include:
  - Desktop profiles on the device
  - BIOS Information
  - Processor Information
  - Computer Manufacturer and Model
  - Installed Hotfixes
  - Operating System Version Information
  - Local Users and Owner
- Remote Command – This can include WMI as described above, or similar technologies on Mac and Linux. This assumes that the targeted systems allow for remote command invocation through configuration.
- Agent – If the environment is managed with deployed agents on the systems then all of the above, including Custom Code, can be used to collect device fingerprinting information. The agents themselves can also collect information but this varies by product. Agents do not always need an OSS to be used. Agents can also be used to collect network level information in the network segment in which they are deployed.
- OSS Systems – These use both network level collection and agents deployed to collect various amounts of information depending on the product. Many OSS systems can also deploy custom scripts or code to collect any information available on the target host.
- Custom Code – Be it a shell script or an application written in a language that can run natively on the targeted system. Non-native apps written in languages such as python would require the python runtime to also be installed.
- Network – This is using techniques such as packet capture and network management to collect identifiers available in IP packets transmitted over the network either passively or actively,

Each collection method on its own and/or the single unique items of information may not have enough information to uniquely identify the device with a high level of confidence. Combinations of identifiers

across collection methods will increase this confidence. This will be further explained below in the *Accuracy/Assurance/Confidence Matrix*.

2.3 CHARACTERISTICS OF A STRONG DEVICE FINGERPRINTING SOLUTION

The device measurement technologies invoked need to burden or complicate IT infrastructure as little as possible.

Uniqueness — is how accurately the technique can identify a device and differentiate it from other similar devices. The attributes retrieved should be based on information that gives a high level of uniqueness. It also needs to be collected from a reliable source on the devices.

Persistence — is how long an attribute can be depended upon to be a reliable component of the device's fingerprint. Since ideally the fingerprint should last as long as possible, the properties chosen should last the lifetime of the device or be slow to change.

Flexible fingerprint matching logic — is important as user's device configuration change over time, which results in fingerprint drift. Fingerprinting needs to accommodate the inevitable little changes in a device to ensure that even legitimate changes (e.g. company-wide operating system patch upgrade or time zone changes due to travel) do not result in the device to be considered "unidentified".

Readily obtainable — has to do with how fast fingerprint can be obtained from the device.

Performance impact — in critical applications, factors like the efficiency in the approach used for fingerprinting, play an important role. The fingerprint solution should not slow down the server's capabilities nor should it eat up resources.

Suitable metrics — is important because ideally device fingerprinting technology should get added in transparently.

Support of multiple devices per user — is fundamental because most users have multiple devices (e.g., laptop, desktop, tablet, mobile etc.). The fingerprinting solution implemented should be able to associate multiple number and types of devices to a particular user. If the solution doesn't support multiple devices, the ability to leverage fingerprinting to verify the user's identity will be limited. The result being that the user will be required to go through a stronger authentication procedure each time they change their device.

Risk based / adaptive authentication framework — should be used with fingerprinting to get the highest level of security and optimized user experience.

Supportable — the technique or technology chosen should be such that administrators can configure, update and effectively debug the fingerprinting methods. As part of the broader risk based authentication configuration, a supportable fingerprint will depend on how granular and customizable the configuration is, and how non-intrusive the mechanism is.

2.4 CLOCK SKEW

In their 2005 paper "Remote physical device fingerprinting", authors Tadayoshi Kohno, Andre Broido, and K.C. Claffy proposed a method of remote device fingerprinting using clock skew. From their abstract:

> We introduce the area of remote physical device fingerprinting, or fingerprinting a physical device, as opposed to an operating system or class of devices, remotely, and without the fingerprinted device's known cooperation. We accomplish this goal by exploiting small, microscopic deviations in device hardware: clock skews. Our techniques do not require any modification to the fingerprinted devices. Our techniques report consistent measurements when the measurer is thousands of miles, multiple hops, and tens of milliseconds away from the fingerprinted device, and when the fingerprinted device is connected to the Internet from different locations and via different access technologies. Further, one can apply our passive and semi-passive techniques when the fingerprinted device is behind a NAT or firewall, and also when the device's system time is maintained via NTP or SNTP. One can use our techniques to obtain information about whether two devices on the Internet, possibly shifted in time or IP addresses, are actually the same physical device. Example applications include: computer forensics; tracking, with some probability, a physical device as it connects to the Internet from different public access points; counting the number of devices behind a NAT even when the devices use constant or random IP IDs; remotely probing a block of addresses to determine if the addresses correspond to virtual hosts, e.g., as part of a virtual honeynet; and unanonymizing anonymized network traces.
>
> *https://homes.cs.washington.edu/~yoshi/papers/PDF/KoBrCl2005PDF-Extended-lowres.pdf*

In their study, they verified "...the ability and developed techniques for remote physical device fingerprinting that exploit the fact that modern computer chips have small yet non-trivial and remotely detectable clock skews."

3 Detailed Analysis

This section will provide details on the parameters, collection methods, quality rankings, and on the reassurance of identification when parameters change.

3.1 COLLECTIBLE PARAMETERS RECOMMENDED TO BE USED

This is a list of individual collectible parameters, pieces of information, which can be collected and used in formulating a device unique identifier. It will also provide each items chance of change in relation to confidence. In a following section, we will show how these can be used in combination to increase confidence levels. We will not attempt to cover the multitude of ways that these values can be changed, only showing you the most common. Any values that are modifiable by command line or editing of files, can also be changed programmatically.

We will not show collection methods from platform GUI. These are not used by automated systems to collect information.

WMI and PowerShell are available on Linux and MacOS. They would need to be installed and configured, but this will allow the querying of platform information, both local and remote, using either PowerShell scripts and commands, or wmic directly.

3.1.1 Windows

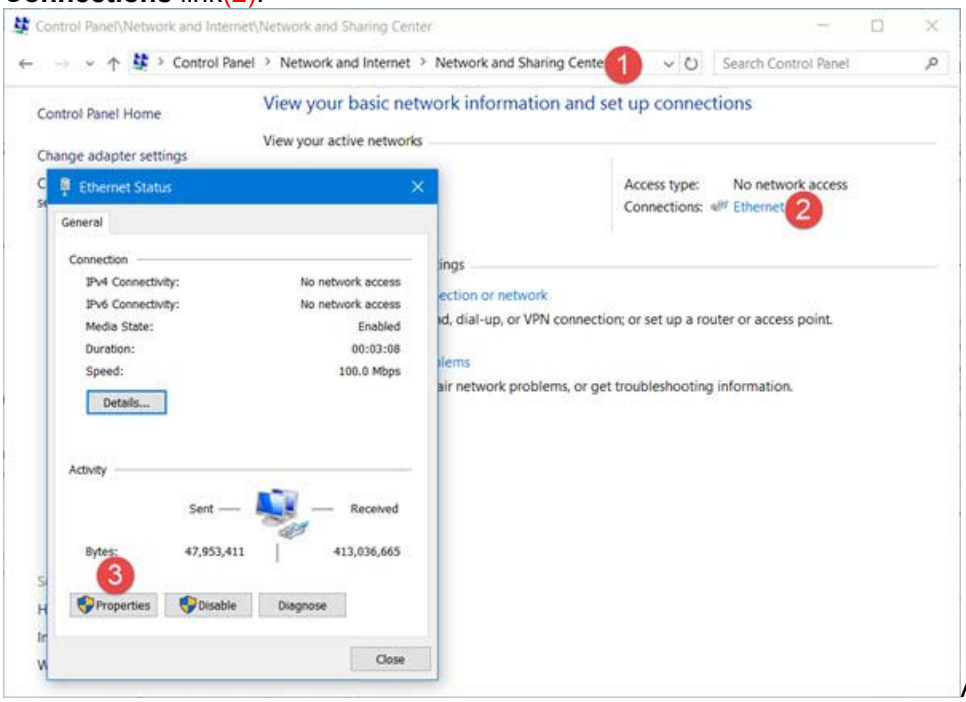
3.1.1.1 IP address

The IP address is either assigned by the network via DHCP or manually specified by a user or administrator for the subnet is which the device resides. This is a highly mutable value if the computer moves from network to network. If always on same network, this value can be changed manually as stated above, but even with DHCP, it can change periodically. Not a good value to use in identifying a device.

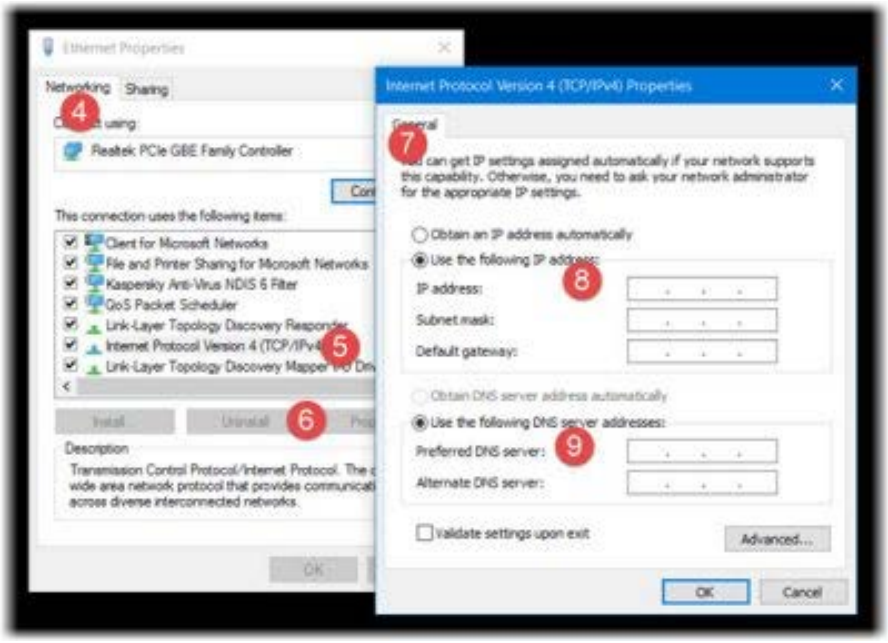
- **Methods of collection:**
  - From command prompt – “ipconfig” will return all the ip addresses of all interfaces on the system.
  - From PowerShell both local and remote – “gip” will return all the ip addresses of all interfaces on the system.
  - All OSS agents can collect IP information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - Through WMI – “Get-WmiObject -query "SELECT IPAddress FROM Win32_NetworkAdapterConfiguration WHERE IPEnabled = 'True'" or “Get-WmiObject win32_networkadapterconfiguration -filter "ipenabled = 'true'"”
  - Custom application code can use numerous methods of getting IP address information by either system calls of the above or calling available APIs to retrieve them such as “Dns.GetHostEntry(Dns.GetHostName()” in C#.

```
public static string GetLocalIPAddress()
{
    var host = Dns.GetHostEntry(Dns.GetHostName());
    foreach (var ip in host.AddressList)
    {
        if (ip.AddressFamily == AddressFamily.InterNetwork)
        {
            return ip.ToString();
        }
    }
    throw new Exception("No network adapters with an IPv4 address in the system!");
}
```

- **Methods of modification:**
  - The IP address can be change by anyone with assigned rights by **Control Panel**, **Network Connections** or **Network and Sharing Center(1)** then click on the **Connections** link(2).



A new window will open up showing the details about your internet connection. Click on the **Properties** button(3).Another window will open up showing the items under **Network** tab(4) used by your connection. Select **Internet Protocol Version 4 (TCP/ IP v4)(5).** Now select



Properties(6)

The default settings of a PC is to obtain the IP address automatically, but you can change it if required. Select **Use the following IP address(8)** and fill the required details (8 & 9 in above image) and click on OK, and you are done.

- You can also change the IP in PowerShell by running ***New-NetIPAddress –InterfaceAlias "Wired Ethernet Connection" –IPv4Address "192.168.0.1" –PrefixLength 24 -***

DefaultGateway 192.168.0.254. Making sure to use the correct Interface, IP address, Mask(PrefixLength), and gateway.

- Though WMI via PowerShell script SetStaticIP.ps1 –

```
$wmi = Get-WmiObject win32_networkadapterconfiguration -filter "ipenabled = 'true'"
$wmi.EnableStatic("10.0.0.15", "255.255.255.0")
$wmi.SetGateways("10.0.0.1", 1)
$wmi.SetDNSServerSearchOrder("10.0.0.100")
```

- Through C# -

Setting a static IP address

```
string myDesc = "Intel(R) Centrino(R) Advanced-N 6205";
string gateway = "192.168.0.1";
string subnetMask = "255.255.255.0";
string address = "192.168.0.10";

var adapterConfig = new ManagementClass("Win32_NetworkAdapterConfiguration");
var networkCollection = adapterConfig.GetInstances();

foreach (ManagementObject adapter in networkCollection)
{
   string description = adapter["Description"] as string;
   if (string.Compare(description,
      myDesc, StringComparison.InvariantCultureIgnoreCase) == 0)
   {
      try
      {
         // Set DefaultGateway
         var newGateway = adapter.GetMethodParameters("SetGateways");
         newGateway["DefaultIPGateway"] = new string[] { gateway };
         newGateway["GatewayCostMetric"] = new int[] { 1 };

         // Set IPAddress and Subnet Mask
         var newAddress = adapter.GetMethodParameters("EnableStatic");
         newAddress["IPAddress"] = new string[] { address };
         newAddress["SubnetMask"] = new string[] { subnetMask };

         adapter.InvokeMethod("EnableStatic", newAddress, null);
         adapter.InvokeMethod("SetGateways", newGateway, null);

         Console.WriteLine("Updated to static IP address!");
      }
      catch (Exception ex)
      {
         Console.WriteLine("Unable to Set IP : " + ex.Message);
      }
   }
}
```

Setting a dynamic IP address

```
string myDesc = "Intel(R) Centrino(R) Advanced-N 6205";
var adapterConfig = new ManagementClass("Win32_NetworkAdapterConfiguration");
var networkCollection = adapterConfig.GetInstances();

foreach (ManagementObject adapter in networkCollection)
{
   string description = adapter["Description"] as string;
   if (string.Compare(description,
      myDesc, StringComparison.InvariantCultureIgnoreCase) == 0)
   {
      try
      {
         adapter.InvokeMethod("EnableDHCP", null);

         Console.WriteLine("Updated Dynamic address!");
      }
      catch (Exception ex)
      {
         Console.WriteLine("Unable to Set IP : " + ex.Message);
      }
   }
}
```

3.1.1.2 MAC address

This is a hardware address that is read from each network interface on the device. This identifier can be modified by administrators or users, usually for hacking, but is rarely done. A good value to use in identifying the uniqueness of a device.

- **Methods of collection:**
  - From command prompt – "ipconfig /all" will return all the MAC addresses of all interfaces on the system.
  - From PowerShell both local and remote – "Get-WmiObject win32_networkadapterconfiguration | select description, macaddress" will return all the MAC addresses of all interfaces on the system.
  - From WMI both local and remote – "wmic nic get"
  - All OSS agents can collect MAC information.
  - Active and passive scanning tools such as Wireshark and Nmap(While on same subnet).
  - Custom application code can use numerous methods of getting MAC address information.

From WMI:

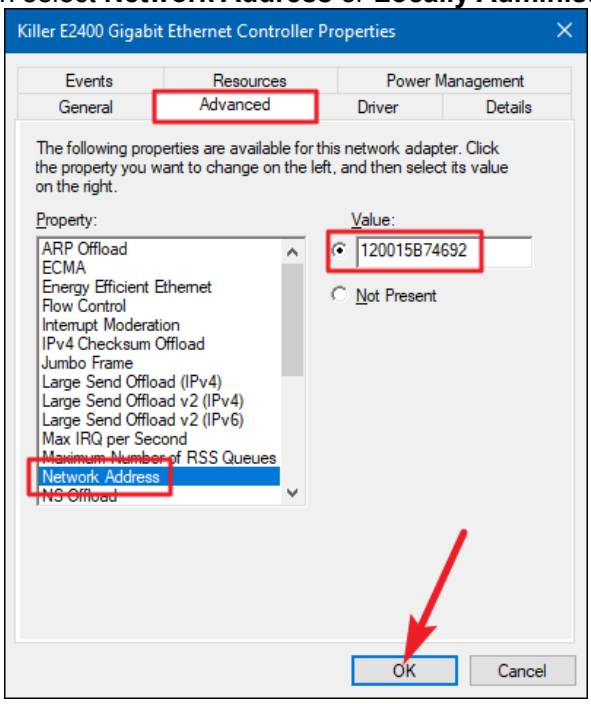
```
public static string GetMACAddress1()
{
   ManagementObjectSearcher objMOS = new ManagementObjectSearcher("Select * FROM
Win32_NetworkAdapterConfiguration");
   ManagementObjectCollection objMOC = objMOS.Get();
   string macAddress = String.Empty;
   foreach (ManagementObject objMO in objMOC)
   {
      object tempMacAddrObj = objMO["MacAddress"];

      if (tempMacAddrObj == null) //Skip objects without a MACAddress
      {
         continue;
      }
      if (macAddress == String.Empty) // only return MAC Address from first card that has a MAC
Address
      {
         macAddress = tempMacAddrObj.ToString();
      }
      objMO.Dispose();
   }
   macAddress = macAddress.Replace(":", "");
   return macAddress;
}
```

From System.Net namespace:

```
public static string GetMACAddress2()
{
   NetworkInterface[] nics = NetworkInterface.GetAllNetworkInterfaces();
   String sMacAddress = string.Empty;
   foreach (NetworkInterface adapter in nics)
   {
      if (sMacAddress == String.Empty)// only return MAC Address from first card
      {
         //IPInterfaceProperties properties = adapter.GetIPProperties(); Line is not required
         sMacAddress = adapter.GetPhysicalAddress().ToString();
      }
   } return sMacAddress;
}
```

- **Methods of modification:**

The MAC address can be change by anyone with assigned rights by A) **Control Panel**, **Network Connections;** or B) In Device Manager; then right click on the interface to change and select **Properties**, the **Configure**, then select **Network Address** or **Locally Administered Address**, and set the MAC to a valid Hex address.



- You can also change the MAC address from PowerShell by ***Set-NetAdapter -Name "Ethernet 1" -MacAddress " xx:xx:xx:xx:xx:xx"*** giving selected interface name and valid hex values for each xx pair.
- Through PowerShell both local and remote - "Get-NetAdapter -Name "Local Area Connection" | Set-NetAdapter -MACAddress "aa-bb-cc-dd-ee-ff" -Confirm:$false"
- Through WMI both local and remote – {wmic nic where "NetConnectionId='Local Area Connection'" set maccaddress='aa-bb-cc-dd-ee-ff'}
- Through C# by directly modifying the registry entry "HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class{4d36e972-e325-11ce-bfc1-08002be10318}"

3.1.1.3 Hostname/Computername

This is an OS or network management assigned value to uniquely identify the device by name. On the network, this will usually have a domain portion as well. It can be modified by administrators or users but must be unique within the subnet/domain in which it resides. Not a good value for device identification unless account restrictions/policies restrict regular users from modifying this value.

- **Methods of collection:**
  - Through command prompt – "hostname"
  - Through WMI both local and remote – "wmic computersystem get dnshostname,name"
  - Through PowerShell both local and remote – "hostname" or "(Get-WmiObject -Class Win32_ComputerSystem -Property Name).Name"
  - All OSS agents can collect this information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - In C#:

```
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
//  <-- Keep this information secure! -->
    Console.WriteLine("MachineName: {0}", Environment.MachineName);
```

```
    }
}
```

- **Methods of modification:**
  - Changing your PC's name involves paying a visit to the "System Properties" window. Starting with Windows 7, it's a little harder to get to, but here are several routes you can take: 1) Type "sysdm.cpl" into the Start menu search box or the Run box. 2) Head to ***Control Panel*** > ***System and Security*** > ***System***, and then click the ***Advanced system settings*** link. 3) In Windows 7, right-click on the ***Computer*** option on the Start menu, and then click the ***Advanced system settings*** link. From here, you click ***Change*** then enter the name and click ***OK***.
  - From CMD prompt run ***WMIC computersystem where caption='current_pc_name' rename new_pc_name.*** Replace current_pc_name with your current computer name, and new_pc_name with your desired new computer name.
  - From PowerShell run ***Rename-Computer -NewName CN1 -LocalCredential WS\Administrator -PassThru*** and press ***Enter*** to change the computer name. Replace **CN1** with the name of the computer and **WS** with the current name of the computer. Type the local **Administrator** password and click **OK** to complete the change.
  - In C#:

```
public static bool SetMachineName(string newName)
{
   RegistryKey key = Registry.LocalMachine;

   string activeComputerName = "SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ActiveComputerName";
   RegistryKey activeCmpName = key.CreateSubKey(activeComputerName);
   activeCmpName.SetValue("ComputerName", newName);
   activeCmpName.Close();
   string computerName = "SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ComputerName";
   RegistryKey cmpName = key.CreateSubKey(computerName);
   cmpName.SetValue("ComputerName", newName);
   cmpName.Close();
   string _hostName = "SYSTEM\\CurrentControlSet\\services\\Tcpip\\Parameters\\";
   RegistryKey hostName = key.CreateSubKey(_hostName);
   hostName.SetValue("Hostname",newName);
   hostName.SetValue("NV Hostname",newName);
   hostName.Close();
   return true;
}
```

3.1.1.4 Open ports

These are IP ports on the system which are listening for network connection requests. These will normally be the same on all systems within a controlled network environment. Not a good value to use for device identification. These change only when applications are installed or configured to listen on specific ports.

- **Methods of collection:**
  - In command prompt – "netstat -a -b"
  - In PowerShell – "Get-NetTCPConnection | ? {$_.State -eq "Listen"}"
  - All OSS agents can collect this information.
  - Active scanning tools such as Nmap. Note that firewalls can block these from being accessible from the network, even thought they are actively listening.
  - In C# methods such as GetActiveTcpConnections(), GetActiveTcpListeners(), and GetActiveUdpListeners() can be used.

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

namespace NetStatNet
{
   class Program
   {
      static void Main(string[] args)
      {
         var props = IPGlobalProperties.GetIPGlobalProperties();
```

```
            Console.WriteLine("  Proto  Local Address          Foreign Address        State");

            foreach (var conn in props.GetActiveTcpConnections())
               Console.WriteLine("  TCP    {0,-23}{1,-23}{2}",
                           conn.LocalEndPoint, conn.RemoteEndPoint, conn.State);

            foreach (var listener in props.GetActiveTcpListeners())
               Console.WriteLine("  TCP    {0,-23}{1,-23}{2}", listener, "", "Listening");

            foreach (var listener in props.GetActiveUdpListeners())
               Console.WriteLine("  UDP    {0,-23}{1,-23}{2}", listener, "", "Listening");

            Console.Read();
        }
    }
}
```

- **Methods of modification:**
  - These modified when applications or tools are installed/uninstalled or enabled/disabled, that are configured to use a port to actively listen for connection requests.
  - Users with correct privileges can enable or disable services in Windows that use ports through the “services” app.
  - Even if a service is enabled, firewalls can block them from being visible on the network.

3.1.1.5 Operating system unique identifiers

There are many of these and they should be grouped to become one value for higher assurance for the purpose of device identification. Each OS has a different set of values. Many potential values exist, but all can be easily changed both by users and by OS updates and patches. Values such as:

OS name

Not unique enough but could be useful in aggregation.

- **Methods of collection**
  - From the CMD prompt – “systeminfo | findstr /B /C:"OS Name"”
  - From PowerShell both local and remote – “$PSVersionTable”
  - From WMI both local and remote – “wmic os get Caption /value”
  - All OSS agents can collect this information.
  - Active and passive scanning tools, such as Nmap and Wireshark, can detect OS type and the version somewhat based on fingerprinting of the traffic and or ports. This is not 100% accurate but can be used to assist in platform detection.
  - In C#: See https://andrewensley.com/2009/06/c-detect-windows-os-part-1/ for details.

```
string getOSInfo()
{
   //Get Operating system information.
   OperatingSystem os = Environment.OSVersion;
   //Get version information about the os.
   Version vs = os.Version;

   //Variable to hold our return value
   string operatingSystem = "";

   if (os.Platform == PlatformID.Win32Windows)
   {
       //This is a pre-NT version of Windows
       switch (vs.Minor)
       {
           case 0:
               operatingSystem = "95";
               break;
           case 10:
               if (vs.Revision.ToString() == "2222A")
                   operatingSystem = "98SE";
               else
                   operatingSystem = "98";
               break;
           case 90:
               operatingSystem = "Me";
```

```
                    break;
                default:
                    break;
            }
        }
        else if (os.Platform == PlatformID.Win32NT)
        {
            switch (vs.Major)
            {
                case 3:
                    operatingSystem = "NT 3.51";
                    break;
                case 4:
                    operatingSystem = "NT 4.0";
                    break;
                case 5:
                    if (vs.Minor == 0)
                        operatingSystem = "2000";
                    else
                        operatingSystem = "XP";
                    break;
                case 6:
                    if (vs.Minor == 0)
                        operatingSystem = "Vista";
                    else if (vs.Minor == 1)
                        operatingSystem = "7";
                    else if (vs.Minor == 2)
                        operatingSystem = "8";
                    else
                        operatingSystem = "8.1";
                    break;
                case 10:
                    operatingSystem = "10";
                    break;
                default:
                    break;
            }
        }
        //Make sure we actually got something in our OS check
        //We don't want to just return " Service Pack 2" or " 32-bit"
        //That information is useless without the OS version.
        if (operatingSystem != "")
        {
            //Got something.  Let's prepend "Windows" and get more info.
            operatingSystem = "Windows " + operatingSystem;
            //See if there's a service pack installed.
            if (os.ServicePack != "")
            {
                //Append it to the OS name.  i.e. "Windows XP Service Pack 3"
                operatingSystem += " " + os.ServicePack;
            }
            //Append the OS architecture.  i.e. "Windows XP Service Pack 3 32-bit"
            //operatingSystem += " " + getOSArchitecture().ToString() + "-bit";
        }
        //Return the information we've gathered.
        return operatingSystem;
    }
```

- **Methods of modification:**
  - The name value can be modified by a user with the correct privileges by editing the registry value at HKEY_LOCAL_MACHINE\BCD00000000\Objects\{4dc2309a-c51f-11e0-817f-b2e2fca804f6}\Elements\12000004
  - Installation of a new major version of windows

OS version

Not unique enough but could be useful in aggregation.

- **Methods of collection**
  - From the CMD prompt – systeminfo | findstr /B /C:"OS Version"
  - From PowerShell both local and remote – "$PSVersionTable"
  - From WMI both local and remote – "wmic os get Version /value"
  - All OSS agents can collect this information.

- Active and passive scanning tools, such as Nmap and Wireshark, can detect OS type and the version somewhat based on fingerprinting of the traffic and or ports. This is not 100% accurate but can be used to assist in platform detection.
- In C#: See **OS Name** above for details.

▪ **Methods of modification:**

- The version value can be modified by a user with the correct privileges by editing the registry value at HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Windows NT\CurrentVersion
- Updating windows to a new version.

Device ID

This is a platform agnostic value such as “wmic csproduct get UUID” on Windows. It is not guaranteed to be unique.

▪ **Methods of collection:**

- From WMI both local and remote – “wmic csproduct get "UUID"
- From PowerShell both local and remote – “(get-wmiobject Win32_ComputerSystemProduct).UUID”
- All OSS agents can collect this information.
- C# from the “Win32_ComputerSystemProduct” class:

```
using System;
using System.Collections.Generic;
using System.Management;
using System.Text;

namespace GetWMI_Info
{
    class Program
    {

        static void Main(string[] args)
        {
            try
            {
                string ComputerName = "localhost";
                ManagementScope Scope;
                Scope = new ManagementScope(String.Format("\\\\{0}\\root\\CIMV2",
ComputerName), null);
                Scope.Connect();
                ObjectQuery Query = new ObjectQuery("SELECT UUID FROM
Win32_ComputerSystemProduct");
                ManagementObjectSearcher Searcher = new
ManagementObjectSearcher(Scope, Query);

                foreach (ManagementObject WmiObject in Searcher.Get())
                {
                    Console.WriteLine("{0,-35} {1,-40}","UUID",WmiObject["UUID"]);//
String
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(String.Format("Exception {0} Trace
{1}",e.Message,e.StackTrace));
            }
            Console.WriteLine("Press Enter to exit");
            Console.Read();
        }
    }
}
```

▪ **Methods of modification:**

- Replacement of motherboard.
- Modify SMBIOS contents by re-flashing BIOS.

Product ID

Windows Only. Not reliable or consistently set. Set upon installation of the OS. Clones and VM images need prep to clear this, such as sysprep on windows. This is generated at install and after sysprep system is booted, and is based on various BIOS and System information. This is not to be confused with Windows Product Key "wmic path softwarelicensingservice get OA3xOriginalProductKey" which under volume licensing will not be unique, and if the OS is upgraded, this will change.

- **Methods of collection:**
  - From WMI both local and remote – "wmic os get "SerialNumber""
  - All OSS agents can collect this information.
- **Methods of modification:**
  - Sysprep of the windows system will erase this value.

MachineGUID

Windows only

- **Methods of collection:**
  - From PowerShell - "(Get-ItemProperty registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\ -Name MachineGuid).MachineGUID " This is not guaranteed to be unique if Cloned or VM snapshots are used. You will need to change it, but some applications may need to be re-installed.
- **Methods of modification:**
  - This is modified by editing the value in the registry at **HKLM\Software\Microsoft\Cryptography\MachineGuid.**

Patches installed

Not unique enough but could be useful in aggregation. These are not modifiable, but will change with OS re-install, upgrade, or new patches installed. To get a complete list of patches regardless of application, see https://www.thewindowsclub.com/check-windows-update-history-using-PowerShell

- **Methods of collection:**
  - From WMI both local and remote – "wmic qfe list"
  - From PowerShell both Local and Remote – "get-wmiobject -class win32_quickfixengineering"
  - All OSS agents can collect this information.
  - In C# using "UpdateSession3::QueryHistory" method:

```
Set updateSearch = CreateObject("Microsoft.Update.Session").CreateUpdateSearcher
Set updateHistory = updateSearch.QueryHistory(1, updateSearch.GetTotalHistoryCount)

For Each updateEntry in updateHistory
  Wscript.Echo "Title: " & updateEntry.Title
  Wscript.Echo "application ID: " & updateEntry.ClientApplicationID
  Wscript.Echo " --"
Next
```

- **Methods of modification:**
  - Re-install windows
  - Apply new patches
  - Upgrade OS such as Windows 10 Release

Locale/Time zone

Not unique enough but could be useful in aggregation.

- **Methods of collection:**
  - From command prompt – "systeminfo | findstr /C:"Time Zone" and "systeminfo /FO LIST | find "System Locale""
  - From PowerShell both local and remote – "(Get-ItemProperty 'HKCU:\Control Panel\International').LocaleName"
  - All OSS agents can collect this information.
- **Methods of modification:**

- It is easier to change the time zone directly via the graphical interface by clicking the **clock** in the system tray and selecting **Change date and time settings**. Next you need to click **Change Time Zone** button, choose a suitable time zone from the list of available time zones, and then save the changes.
- From the cmd prompt, you can also run ***tzutil /s "Eastern Standard Time"*** changing ***Eastern Standard Time*** to the time zone of choice***.***
- Windows PowerShell has a similar command. Run Set-TimeZone -Name "US Eastern Standard Time". Changing ***Eastern Standard Time*** to the time zone of choice.

Username

Of logged in user. Not reliable or useful.

- **Methods of collection:**
  - From command prompt – "whoami"
  - From PowerShell both local and remote – "$env:UserName"
  - From WMI both local and remote – "wmic computersystem get username"
  - All OSS agents can collect this information.
  - In C# use the following in your code "string userName = System.Security.Principal.WindowsIdentity.GetCurrent().Name;"
- **Methods of modification:**
  - New account assigned by administrator.
  - Log in as a different user.
  - Create a local account (requires correct privileges) and log in with it.

Enabled services/daemons

Not unique enough but could be useful in aggregation. These are not great for device identification since they are modifiable and can change when updates are applied. These can also be enabled and disabled by users.

- **Methods of collection:**
  - From the command prompt – "net start"
  - From PowerShell both local and remote – "Get-Service | Where-Object {$_.Status -eq "Running"}"
  - From WMI both local and remote – "wmic service where started=true"
  - All OSS agents can collect this information.
  - In C# you can use the following in your code:

```
ServiceController[] services = ServiceController.GetServices();
var servicesStatus = services.ToDictionary(s => s.ServiceName, s => s.Status);
```

- **Methods of modification:**
  - Users with correct privileges can stop and start services.
  - Services installed or removed by administrator or user with correct privileges.

3.1.1.6 Hardware

These are values tied to specific hardware installed in the device. These are best for identifying unique devices and are the major part what Microsoft uses to generate their UUID to tie to the product key for Windows. These are not modifiable by anyone in any normal situation, but very advanced developers can create code to intercept the calls to collect such information and return bogus values. They change when hardware devices are changed.

Motherboard identifiers

Such as DMI Baseboard.

- **Methods of collection:**
  - From WMI both local and remote – "wmic baseboard get"
  - From PowerShell both local and remote – "Get-WmiObject Win32_baseboard"
  - All OSS agents can collect this information.

BIOS identifiers

All set bios parameters. Not all vendors populate.

- **Methods of collection:**
  - From WMI both local and remote – "wmic bios get"
  - From PowerShell both local and remote – "Get-WmiObject Win32_Bios"
  - All OSS agents can collect this information.

SMBIOS/DMI UUID identifiers

Best overall method of identifying a machine. But is not guaranteed.

- **Methods of collection:**
  - From WMI both local and remote – "wmic path win32_computersystemproduct get"
  - From PowerShell both local and remote – "Get-WmiObject win32_computersystemproduct"
  - All OSS agents can collect this information.

cpuid identifier

Not reliable or consistent.

- **Methods of collection:**
  - From WMI both local and remote – "wmic cpu get"
  - From PowerShell both local and remote – "Get-WmiObject Win32_processor" To get details, see https://docs.microsoft.com/en-us/windows/desktop/cimwin32prov/win32-processor

```
$cpu_info = Get-WmiObject -class Win32_Processor


foreach ($cpu_values in $cpu_info) {

"Name " +[char]9 +[char]9 +[char]9 +[char]9 +[char]9 +[char]9 +" : " +
$cpu_values.Name  +[char]10
"Manufacturer " +[char]9 +[char]9 +[char]9 +[char]9+" : " +
$cpu_values.Manufacturer+[char]10
"Caption "+[char]9 +[char]9 +[char]9 +[char]9 +[char]9+" : " +
$cpu_values.Caption+[char]10
"ProcessorId "+[char]9 +[char]9 +[char]9 +[char]9 +" : " +
$cpu_values.ProcessorId+[char]10
"CurrentClockSpeed "+[char]9 +[char]9+[char]9+" : " + $cpu_values.CurrentClockSpeed
+ "MHz"+[char]10

"NumberOfCores"+[char]9 +[char]9+[char]9+[char]9 +" : " +$cpu_values.NumberOfCores
+[char]10
"NumberOfEnabledCore"+[char]9 +[char]9+[char]9+" : "
+$cpu_values.NumberOfEnabledCore  +[char]10
"NumberOfLogicalProcessors "+[char]9 +" : " + $cpu_values.NumberOfLogicalProcessors
+[char]10

$xcpu = $cpu_values.Architecture

switch ($xcpu)
    {
0     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : " +"x86
(32bit)"+[char]10}
1     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : "
+"MIPS"+[char]10}
2     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : "
+"Alpha"+[char]10}
3     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : "
+"PowerPC"+[char]10}
5     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : " +"ARM"+[char]10}
6     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : " + "ia64 Itanium-
based systems"+[char]10}
9     {"Architecture" +[char]9 +[char]9  +[char]9 +[char]9 +  " : " + "x64
(64bit)"+[char]10}
    }

   $xcpu = $cpu_values.CpuStatus

switch ($xcpu)
    {
0     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : "
+"Unknown"+[char]10}
1     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"CPU
Enabled"+[char]10}
2     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"CPU
Disabled by User via BIOS Setup"+[char]10}
```

```
3     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"CPU
Disabled By BIOS (POST Error)"+[char]10}
4     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"CPU is
Idle"+[char]10}
5     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"Reserved
"+[char]10}
6     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"Reserved
"+[char]10}
7     {"CpuStatus" +[char]9 +[char]9  +[char]9 +[char]9 +[char]9 +  " : " +"Other
"+[char]10}

    }

$xcpu = $cpu_values.ProcessorType

switch ($xcpu)
    {
1     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : "
+"Other"+[char]10}
2     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : "
+"Unknown"+[char]10}
3     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : " +"Central
Processor"+[char]10}
4     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : " +"Math
Processor"+[char]10}
5     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : " +"DSP
Processor"+[char]10}
6     {"ProcessorType" +[char]9 +[char]9  +[char]9 +[char]9  +  " : " +"Video
Processor "+[char]10}
    }
}
```

- All OSS agents can collect this information.

Memory identifiers

These are all memory dims details.

- **Methods of collection:**
  - From WMI both local and remote – "wmic memorychip get"
  - From PowerShell both local and remote – "Get-WmiObject win32_physicalmemory | Format-Table Manufacturer,Banklabel,Configuredclockspeed,Devicelocator,Capacity,Serialnumber -autosize"
  - All OSS agents can collect this information.

Hard drive identifiers

Drive serial numbers are very accurate and only change if drive changes.

- **Methods of collection:**
  - From WMI both local and remote – "wmic diskdrive get"
  - From PowerShell both local and remote – "Get-WmiObject win32_diskdrive | select model,serialnumber"
  - All OSS agents can collect this information.

Video card identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From WMI both local and remote – "wmic path win32_VideoController get"
  - From PowerShell both local and remote – "Get-WmiObject Win32_VideoController"
  - All OSS agents can collect this information.

Display Identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From WMI both local and remote – "wmic desktopmonitor get"
  - From PowerShell both local and remote:

```
$Monitors = Get-WmiObject WmiMonitorID -Namespace root\wmi
$LogFile = "monitors.txt"
"Manufacturer,Serial" | Out-File $LogFile

ForEach ($Monitor in $Monitors)
{
```

```
        $Manufacturer = ($Monitor.ManufacturerName -notmatch 0 | ForEach{[char]$_}) -join ""
        $Serial = ($Monitor.SerialNumberID -notmatch 0 | ForEach{[char]$_}) -join ""

        "$Manufacturer,$Serial" | Out-File $LogFile -append
    }
    Cat $LogFile
```

- All OSS agents can collect this information.

3.1.2 MacOS

Apple systems also support Apple Remote Desktop (ARD), which is based on WBEM which is the basis of WMI on windows. It is more complex than WMI in that you configure it on one system and generate reports that can then be parsed/exported to other systems. All of the information we list here can be collected via ARD.

During research, it was found that although both Objective-C and Swift can be used to collect all this information, working examples are very difficult to find, and it would greatly delay this report to build out solutions in either language.

3.1.2.1 IP address

Since MacOS is very similar to Linux, many of the commands between the two will be similar.

- **Methods of collection:**
  - From the terminal – "ifconfig /all"
  - From PowerShell both local and remote – "gip" will return all the ip addresses of all interfaces on the system.
  - All OSS agents can collect IP information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - In Objective-C see: http://bluelove1968.pixnet.net/blog/post/222276784-get-ip%EF%BC%8Fmac-address-in-c-objective-c-on-mac-osx
- **Methods of modification:**
  - From the Apple menu, select **System Preferences**. Select **Network**. From the sidebar, select an active network interface. Click **Advanced**.Select **TCP/IP**. From the Configure IPv4 menu, select **Manually**. Enter a static IP address in the IPv4 Address field. Click **OK**. Click **Apply**.
  - You can also change from Terminal with ***sudo ipconfig set en1 INFORM 192.168.0.150***. changing the interface name and IP address to the one you want to use.

3.1.2.2 MAC address

- **Methods of collection:**
  - See all options listed in "IP address" above.
- **Methods of modification:**
  - Open a terminal, and run ***sudo ifconfig en0 ether xx:xx:xx:xx:xx:xx*** giving valid hex values for each xx pair.

3.1.2.3 Hostname/Computername

- **Methods of collection:**
  - From the terminal – "hostname"
  - From PowerShell both local and remote – "gip" will return all the ip addresses of all interfaces on the system.
  - All OSS agents can collect IP information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - In Objective-C use: "[[NSHost currentHost] localizedName];"

- **Methods of modification:**
  - Open a terminal. Type the following command to change the primary hostname of your Mac: “This is your fully qualified hostname, for example myMac.domain.com” ***sudo scutil --set HostName <new host name>.*** Then type the following command to change the Bonjour hostname of your Mac: “This is the name usable on the local network, for example myMac.local”. ***sudo scutil --set LocalHostName <new host name>***. Optional: If you also want to change the computer name, type the following command: “This is the user-friendly computer name you see in Finder, for example myMac”.***sudo scutil --set ComputerName <new name>***. Now flush the DNS cache by typing: ***dscacheutil -flushcache*** . Restart your Mac.
  - From the UI, open up **System Preferences**, go to **Sharing**, and change the **Computer Name**. Click anywhere outside the window to set it.

3.1.2.4 Open ports

- **Methods of collection:**
  - From the terminal, you can collect bot the ports that are listening and the process behind it using:
    netstat -ap tcp | grep -i "listen"
    sudo lsof -PiTCP -sTCP:LISTEN
  - From PowerShell you can write a function for powershell. See https://aarontheadmin.wordpress.com/2018/07/09/powershell-core-get-tcp-connections-on-macos/ for details.
  - All OSS agents can collect this information.
  - Active and passive scanning tools such as Wireshark and Nmap.
- **Methods of modification:**
  - These modified when applications or tools are installed/uninstalled or enabled/disabled, that are configured to use a port to actively listen for connection requests.
  - Users with correct privileges can enable or disable daemons in MacOS that use ports through the “launchctl” app.
  - Even if a daemon is enabled, firewalls can block them from being visible on the network.

3.1.2.5 Operating system unique identifiers

OS Name

- **Methods of collection:**
  - From the terminal – “sw_vers”
  - From PowerShell you can run the following :
    $IsMacOS
    $PSVersionTable
  - All OSS agents can collect this information.
  - Active and passive scanning tools, such as Nmap and Wireshark, can detect OS type and the version somewhat based on fingerprinting of the traffic and or ports. This is not 100% accurate but can be used to assist in platform detection.
- **Methods of modification:**
  - The name value can be modified by a user with the correct privileges by editing the contents of /System/Library/CoreServices/SystemVersion.plist. This is not guaranteed since the name value is also imbedded in various libraries.
  - Installation of a new major version of MacOS

OS version

Not unique enough but could be useful in aggregation.

- **Methods of collection**

- From the terminal – “sw_vers”
- From PowerShell you can run the following :
  $IsMacOS
  $PSVersionTable
- All OSS agents can collect this information.
- Active and passive scanning tools, such as Nmap and Wireshark, can detect OS type and the version somewhat based on fingerprinting of the traffic and or ports. This is not 100% accurate but can be used to assist in platform detection.
- **Methods of modification:**
  - The version values can be modified by a user with the correct privileges by editing the contents of /System/Library/CoreServices/SystemVersion.plist.
  - Updating MacOS to a new version.

Device ID

This is a platform agnostic value such as “wmic csproduct get UUID” on Windows. It is not guaranteed to be unique.

- **Methods of collection:**
  - From the terminal – “ioreg -d2 -c IOPlatformExpertDevice | awk -F\" '/IOPlatformUUID/{print $(NF-1)}'"
  - Using C/C++ you can query the hardware unique identifier:
    ```
    void get_platform_uuid(char * buf, int bufSize) {
        io_registry_entry_t ioRegistryRoot = IORegistryEntryFromPath(kIOMasterPortDefault, "IOService:/");
        CFStringRef uuidCf = (CFStringRef) IORegistryEntryCreateCFProperty(ioRegistryRoot, CFSTR(kIOPlatformUUIDKey), kCFAllocatorDefault, 0);
        IOObjectRelease(ioRegistryRoot);
        CFStringGetCString(uuidCf, buf, bufSize, kCFStringEncodingMacRoman);
        CFRelease(uuidCf);
    }
    ```
  - All OSS agents can collect this information.
- **Methods of modification:**
  - Replacement of motherboard, Ram, Video, Disk.
  - In a VM, Modify the *.vmx file in the machine folder by changing the uuid.bios, then delete the .nvram file.
  - The version values can be modified by a user with the correct privileges by editing the contents of /System/Library/CoreServices/SystemVersion.plist.

Patches installed

Not unique enough but could be useful in aggregation. These are not modifiable, but will change with OS re-install, upgrade, or new patches installed. To get a complete list of patches regardless of application, see https://www.thewindowsclub.com/check-windows-update-history-using-PowerShell

- **Methods of collection:**
  - From the terminal – “/usr/sbin/system_profiler SPInstallHistoryDataType”
  - All OSS agents can collect this information.
- **Methods of modification:**
  - Re-install MacOS
  - Apply new patches
  - Upgrade OS

Locale/Time zone

Not unique enough but could be useful in aggregation.

- **Methods of collection:**
  - From the terminal – “defaults read .GlobalPreferences AppleLanguages | tr -d [:space:] | cut -c2-3” and “ls /etc/timezone"
  - All OSS agents can collect this information.
  - In Swift3 Xcode 8, you can retrieve the current timezone with:
    ```
    var localTimeZoneAbbreviation: String { return TimeZone.current.abbreviation() ?? "" }
    ```

```
localTimeZoneAbbreviation   // "GMT-2"
var localTimeZoneName: String { return TimeZone.current.identifier }
localTimeZoneName // "America/Sao_Paulo"
```

- **Methods of modification:**
  - It is easier to change the time zone directly via the graphical interface by clicking the **System Preferences …** in the Apple menu and selecting **Date & Time**. Next you need to Select **Set date & time automatically:**, Select the **Time Zone** tab. If the system is unable to determine your location automatically, uncheck **Set time zone automatically using current location** so you can manually set your location. On the world map, select your time zone region. From the "Closest City:" drop-down list, choose Indianapolis for Eastern Time or Chicago for Central Time.
  - From the terminal, you can also run the following command, replacing *timezone* with a valid value :
    sudo systemsetup -settimezone timezone

Username

Of logged in user. Not reliable or useful.

- **Methods of collection:**
  - From the terminal – "id -un" or "whoami"
  - All OSS agents can collect this information.
  - In Swift:
    let userName = NSUserName()
    let fullUserName = NSFullUserName()
- **Methods of modification:**
  - New account assigned by administrator.
  - Log in as a different user.
  - Create a local account (requires correct privileges) and log in with it.

- **Enabled services/daemons** - Not unique enough but could be useful in aggregation. These are not great for device identification since they are modifiable and can change when updates are applied. These can also be enabled and disabled by users.
  - **Methods of collection:**
    - From the terminal – "sudo launchctl list"
    - All OSS agents can collect this information.
  - **Methods of modification:**
    - Users with correct privileges can stop and start daemons.
    - Daemons installed or removed by administrator or user with correct privileges.

3.1.2.6 Hardware

These are values tied to specific hardware installed in the device. These are best for identifying unique devices. These are not modifiable by anyone in any normal situation, but very advanced developers can create code to intercept the calls to collect such information and return bogus values. They change when hardware devices are changed. On MacOS, installing something like dmidecode to query everything from firmware to hardware identifiers may assist in the collection process.

Motherboard identifiers

On MacOS by using system_profiler.

- **Methods of collection:**
  - From a terminal/shell "system_profiler SPHardwareDataType | awk '/Serial/ {print $4}'"
  - All OSS agents can collect this information.

BIOS identifiers

Apple Mac devices do not use Bios, but use EFI, PRAM/NVRAM, and System Management controller (SMC). These usually store locale, keyboard language, boot device, boot order, drivers, driver load order, among others.

- **Methods of collection:**
  - From the terminal, use “nvram” command with either the “-xp” option for xml format, or “-p” for the unformatted output.
  - Other values stored in SMC are not well documented, and may or may not be of any value in identification
  - All OSS agents can collect this information.
- **Methods of Modification:**
  - Both SMC and NVRAM EFI values can be reset/modified using SMC reset method on the Mac in question, and the “nvram” terminal command to change EFI values.

SMBIOS/DMI UUID identifiers

Best overall method of identifying a machine. But is not guaranteed. On MacOS, us system_profiler.

- **Methods of collection:**
  - From a terminal/shell “system_profiler SPHardwareDataType | awk '/UUID/ {print $3}'”
  - All OSS agents can collect this information.

cpuid identifier

Not reliable or consistent.

- **Methods of collection:**
  - This is not available with basic MacOS installation. You will be required to install something like MacCPUID to query those values.
  - From a terminal/shell “system_profiler SPHardwareDataType”
  - All OSS agents can collect this information.

Memory identifiers

These are all memory dims details.

- **Methods of modification:**
  - From a terminal/shell “system_profiler SPMemoryDataType”
  - All OSS agents can collect this information.

Hard drive identifiers

Drive serial numbers are very accurate and only change if drive changes.

- **Methods of collection:**
  - From the terminal, use “diskutil info disk0”.
  - From the terminal/shell “system_profiler SPSerialATADataType”
  - All OSS agents can collect this information.

Video card identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From the terminal: system_profiler SPDisplaysDataType
  - All OSS agents can collect this information.

Display Identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From the terminal/shell: “system_profiler SPDisplaysDataType”
  - All OSS agents can collect this information.

3.1.3 Linux

For Linux/Unix systems, it is best to rely as much as possible on tools such as dmidecode to provide system unique identifiers.

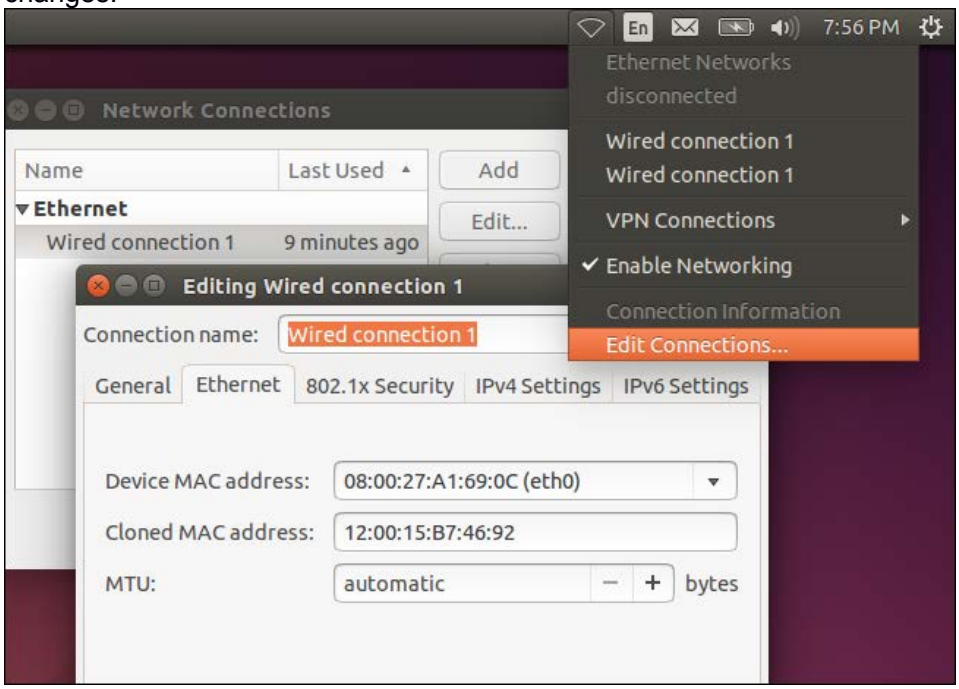
3.1.3.1 IP address

- **Methods of collection:**
  - From the terminal/shell: "ifconfig"
  - From the terminal/shell: "hostname -I"
  - From the terminal/shell: "ip a"
  - All OSS agents can collect this information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - Using c++ you can query IP information, but this approach relies on ifconfig or ip commands above.
- **Methods of modification:**
  - All Linux versions support changing the IP address by opening a terminal window and running ***ifconfig eth0 192.168.1.5 netmask 255.255.255.0 up*** Making sure to use the correct interface, IP address and Mask(netmask).
  - Most modern Linux distributions like Ubuntu typically use Network Manager, which provides a graphical way to change a IP address. For example, in Ubuntu you'd click the network icon on the top panel, click ***Edit Connections***, select the network connection you want to modify, and then click ***Edit***. Select the ***IPv4 Settings*** tab, Change ***Method*** to ***Manual***, Select ***Add***, you'd then enter a new IP address, netmask, and gateway, and then save your changes.
  - Ifconfig is being replaced by newer commands "ip" and "netplan".
  - You can also change the IP address by editing a configuration file:
    - Ubuntu - ***/etc/network/interfaces*** followed by ***sudo /etc/init.d/networking restart***
    - CentOS - ***/etc/sysconfig/network-scripts/ifcfg-eth0*** for eth0 followed by ***service network restart***
    - Similar methods exists for other Linux variants.

3.1.3.2 MAC address

- **Methods of collection:**
  - From the terminal/shell: "ifconfig"
  - From the terminal/shell: "ip a"
  - All OSS agents can collect this information.
  - Active and passive scanning tools such as Wireshark and Nmap(while on same subnet).
  - Using c++ you can query IP information, but this approach relies on ifconfig or ip commands above.
- **Methods of modification:**
  - All Linux versions support changing the MAC address by opening a terminal window and running ***sudo ifconfig eth0 hw ether xx:xx:xx:xx:xx:xx*** giving valid hex values for each xx pair.
  - Most modern Linux distributions like Ubuntu typically use Network Manager, which provides a graphical way to change a MAC address. For example, in Ubuntu you'd click the network icon on the top panel, click ***Edit Connections***, select the network connection you want to modify, and then click ***Edit***. Select the ***Ethernet*** tab, you'd enter a new MAC address in the ***Cloned MAC address*** field, and then save your

changes.



- Ifconfig is being replaced by newer commands “ip” and “netplan”.
- You can also change the IP address by editing a configuration file:
  Ubuntu - ***/etc/network/interfaces*** followed by ***sudo /etc/init.d/networking restart***
  CentOS - ***/etc/sysconfig/network-scripts/ifcfg-eth0*** for eth0 followed by ***service network restart***
  Similar methods exist for other Linux variants.

3.1.3.3 Hostname/Computername

- **Methods of collection:**
  - From the terminal/shell: “hostname”
  - All OSS agents can collect this information.
  - Active and passive scanning tools such as Wireshark and Nmap.
  - All current languages can query the hostname.
- **Methods of modification:**
  - All Linux versions support changing the hostname by opening a terminal window and running ***sudo hostname <new-server-name-here>*** . You should also edit **/etc/hosts** and change the old hostname entry to the new entry.
  - You can also edit **/etc/hostname** and **/etc/hosts** to change the hostname, but this will require a restart.
  - Various Linux distros also support the **hostnamectl** command. From a terminal window run ***hostnamectl set-hostname <new name>***.
  - Linux distros also support ways to change the hostname from the GUI. For example, on Ubuntu go to **System Settings** (search for it in Unity Dash or GNOME). In here, look for **Details.** Modify the **Device name.** You can close that window.

3.1.3.4 Open ports

- **Methods of collection:**
  - From the terminal/shell: “sudo lsof -i -P -n | grep LISTEN”

- From the terminal/shell: "netstat -tulpn | grep LISTEN"
- From the terminal/shell: "nmap -sTU -O IP-address-Here"
- All OSS agents can collect this information.
- Active scanning tools such as Nmap. Note that firewalls can block these from being accessible from the network, even thought they are actively listening.
- All current languages can query for listening ports.

- **Methods of modification:**
  - These are modified when applications or tools are installed/uninstalled or enabled/disabled, that are configured to use a port to actively listen for connection requests.
  - Users with correct privileges can enable or disable daemons and/or applications that use ports.
  - Even if a port is actively listening, firewalls can block them from being visible on the network.

3.1.3.5 Operating system unique identifiers

OS Name

- **Methods of collection:**
  - From the terminal/shell – "cat /etc/os-release"
  - From the terminal/shell – "lsb_release -a"
  - From the terminal/shell – "hostnamectl"
  - From the terminal/shell – "uname -r"
  - From the terminal/shell – "cat /proc/version"
  - All OSS agents collect this information.
  - Active and passive scanning tools, such as Nmap and Wireshark, can detect OS type and the version somewhat based on fingerprinting of the traffic and or ports. This is not 100% accurate but can be used to assist in platform detection.
  - In C++, use of GetSystemOutput() using the above and parsing the results.
- **Methods of modification:**
  - The name value can be modified by a user with the correct privileges by editing the "/etc/os-release", "/etc/issue", or "/etc/lsb-release" file.
  - Installation of a new major version of the various linux distros.
  - Many locations throughout each distro contain name and version information, that would have to be changed.

OS version

- **Methods of collection**
  - See above **OS Name** section.
- **Methods of modification:**
  - See above **OS Name** section.

Device ID

This is a platform agnostic value such as "wmic csproduct get UUID" on Windows. It is not guaranteed to be unique.

- **Methods of collection:**
  - From the terminal/shell – "cat /etc/machine-id"
  - Most OSS agents can collect this information.
  - "***cat /sys/class/dmi/id/product_uuid***"
  - Use of dmidecode application.
  - In C++, use of GetSystemOutput() using the above and parsing the results.
- **Methods of modification:**

- Directly editing /etc/machine-id file. Clearing the file will cause a subsequent reboot to generate a new value.

Patches installed

Not unique enough but could be useful in aggregation. These are not modifiable, but will change with OS re-install, upgrade, or new patches installed. The complexity of the various distributions across releases and versions make this very complex, but the base set of tools for the various major distributions are as follows. If patches and updates were installed outside of the distributions set of tools, then you would need to scan the complete system. For querying these through various computer languages, you use system calls to the command line utilities as described.

- **Methods of collection:**
  - From the terminal/shell on Ubuntu – "less /var/log/apt/history.log", "/var/log/dpkg.log", or "apt list –installed"
  - From the terminal/shell on RedHat/CentOS and variants – "rpm -qa" or "yum list installed"
  - From the terminal/shell on Debian – "dpkg-query -l" or "cat /var/lib/dpkg/status"
  - From the terminal/shell on Arch – "pacman -Qqe"
  - Most OSS agents can collect this information.
- **Methods of modification:**
  - Re-install distribution
  - Apply new updates
  - Upgrade OS
  - Manually delete databases of installed packages.

Locale/Time zone

Not unique enough but could be useful in aggregation.

- **Methods of collection:**
  - From the terminal/shell – For locale = "locale", for timezone = "date" or "timedatectl"
  - All OSS agents can collect this information.
- **Methods of modification:**
  - From the terminal/shell – "localectl set-locale XXX" or modifying "/etc/locale.conf" with a line XXX. Where XXX is a valid locale such as "LANG=en_US.UTF-8"
  - Through the distributions UI where supported.
  - There are as many ways to change the time zone from a terminal on Linux as there are distros and versions of those distros. It is easily found by searching the web for "DISTRO VERSION time zone". Changing DISTRO and VERSION as required.
  - Ways to change the time zone from the GUI also exist, such as Ubuntu for example: Select Settings, Details, then Date & Time. Change Automatic Time Zone to Off, then click on the current Time Zone. Select a new time zone.
  - Specific distribution options to change these various settings, see this site.

Username

Of logged in user. Not reliable or useful.

- **Methods of collection:**
  - From the terminal/shell – "w", "who", or "users"
  - All OSS agents can collect this information.
- **Methods of modification:**
  - New account assigned by administrator.
  - Log in as a different user.
  - Create a local account (requires correct privileges) and log in with it.

Enabled services/daemons

Not unique enough but could be useful in aggregation. These are not great for device identification since they are modifiable and can change when updates are applied. These can also be enabled and disabled by users.

- **Methods of collection:**
  - From the terminal/shell
- **Methods of modification:**
  - Users with correct privileges can stop and start daemons.
  - Applications installed or removed by administrator or user with correct privileges.

3.1.3.6 Hardware

These are values tied to specific hardware installed in the device. These are best for identifying unique devices. These are not modifiable by anyone in any normal situation, but very advanced developers can create code to intercept the calls to collect such information and return bogus values. They change when hardware devices are changed.

Motherboard identifiers

Such as DMI Baseboard.

- **Methods of collection:**
  - From the terminal/shell – "lshw -class system", using 3rd party free tool inxi, and "dmidecode -t 2"

BIOS identifiers

All set bios parameters. Not all vendors populate.

- **Methods of collection:**
  - From the terminal/shell – "lshw -class bios", "hwinfo --bios", "biosdecode", using 3rd party free tool inxi, and "dmidecode -t 0"

SMBIOS/DMI UUID identifiers

Best overall method of identifying a machine. But is not guaranteed.

- **Methods of collection:**
  - From the terminal/shell – "lshw -class system", using 3rd party free tool inxi, and "dmidecode -t 1"

cpuid identifier

Not reliable or consistent.

- **Methods of collection:**
  - From the terminal/shell – "lscpu", "lshw -class processor", "hwinfo --cpu", using 3rd party tool inxi, and "dmidecode -t 4"

Memory identifiers

These are all memory dims details.

- **Methods of collection:**
  - From the terminal/shell – "lshw -class memory", "hwinfo --memory", "lspci", using 3rd party free tool inxi, and "dmidecode -t 17"

Hard drive identifiers

Drive serial numbers are very accurate and only change if drive changes.

- **Methods of collection:**
  - From the terminal/shell – "lshw -class disk", "hwinfo --disk", "lspci", and using 3rd party free tool inxi

Video card identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From the terminal/shell – "lshw -class video", "hwinfo --gfxcard", "lspci | grep ' VGA ' | cut -d" " -f 1 | xargs -i lspci -v -s {}", using 3rd party free tool inxi, and "dmidecode"

Display Identifiers

These are the queryable identifiers such as serial number, model number, brand etc…

- **Methods of collection:**
  - From the terminal/shell – "get-edid | parse-edid"

Along with:

- **Installed fonts** – Some research has shown that using a list of installed fonts will also assist in identifying device uniqueness. This has not been proven, but it may be another grouped value to use with other identifiers to ensured device uniqueness. These change when fonts are installed or removed.
- **Installed software** – This is another set of values that can greatly assist in the unique identification of a device. This includes:
  - Product name
  - Product version
  - Product key/serial number – if it applies

  These can change when software is installed, removed, or updated, but if used with the other identifiers will greatly enhance device identification.

3.1.4 Accuracy/Assurance/Confidence Matrix

This table is to simplify the display of accuracy/assurance/confidence of each value using Low(L), Medium(M), High(H), 100%. We have the 100% to differentiate that high does not mean 100%.

VALUE	*L*	*M*	*H*	*100%*	*NOTES*
IP address	X			X[1]	Easily modifiable except in tightly controlled environments
MAC address	X			X[1]	Easily modifiable except in tightly controlled environments
Hostname/Computername	X		X	X[1]	Highly dependant on how tightly the environment is, such as Windows Domains
Open ports	X				Not unique enough but useful in aggregation
OS name	X				Not unique enough but useful in aggregation
OS Version	X				Not unique enough but useful in aggregation
OS UUID			X	X[1]	This is a good basis for creating a unique ID of high confidence.
Product ID	X				Not unique enough but useful in aggregation. Not unique enough but useful in aggregation
Patches installed	X				Not unique enough but useful in aggregation
Local/Time zone	X				Not unique enough but useful in aggregation
Enabled services/daemons	X				Not unique enough but useful in aggregation
Motherboard identifiers			X	X[1]	DMI baseboard values aggregated. Values “product,Manufacturer,version,serialnumber”. Not useful inVMs or containers.
BIOS identifiers	X				Not unique enough but useful in aggregation. Not useful inVMs or containers.
Other SMBIOS/DMI identifiers		X			Many exist, and while not always populated by vendors, those that do exist can be used in an aggregate fashion to help uniquely identify the system
cpuid identifier	X				Not unique enough but useful in aggregation
Memory identifiers			X		Very unique with Name, Revision, and Serial #. Not useful inVMs or containers.
Hard drive identifiers			X		Very unique with Manufacturer, Part #, and Serial #. Not useful inVMs or containers.
Video card identifiers	X				Not unique enough but useful in aggregation. Not useful inVMs or containers.
Display Identifiers	X				Not unique enough but useful in aggregation. Not useful inVMs or containers.
Other attached devices		X			Many devices/Peripherals have identifiers when attached to a system. This includes Mice, Keyboards, etc... This includes USB and PCI devices. Not useful inVMs or containers.
HKEY_LOCAL_MACHINE			X		Windows only. Will only change if the bootable hard drive is changed, or a re-install of the OS.

SOFTWARE\Microsoft\Cryptography					

[1] – Can be 100% only in completely controlled environment where only senior administrators can change this value or allow it to be falsely reported.

3.1.5 How to improve confidence level

Looking at the above matrix, you can quickly see that in uncontrolled environments that unique identification can nearly be impossible for any individual identifier.

This can be overcome to the greater extent by aggregating various identifiers and generating your own UUID to be stored on the identified system. Many code examples 3rd party tools exist to do this.

To properly do this, you would collect all identifiers specific to a platform (Windows, Linux, Osx) then store them in a central repository. You then generate a UUID and store it on the system and in the repository. On subsequent identification collection passes, you may find that the your generated UUID now exists on two machines, or that a lot of individual identifiers associated with that generated UUID have changed. You can then use an algorithm to review these other identifiers and make a decision on if this is NOT a new system, but is in fact a system that may only have had a number of components replaced, or that the system was cloned.

This is how Microsoft re-activates OS installs automatically in a majority of cases. Where intervention is required is in extreme hardware upgrades. Take for example a user who needs newer hardware to meet certain needs, and he decides to move to a completely new device and does not wish to move hardware over, but only his OS image. This now appears as all new identifiers except the original System UUID generated at OS install on old system. You may need to do a manual process to ensure that this should be treated as the same system. Don't forget that someone may re-use the old system as well. So what would you do? Microsoft invalidates that machine identifiers on their side and re-assign's new ones to the generated UUID. When the old system tries to be used, it will fail Microsoft's validation and will need a new license key to be able to generate a new System UUID and store it as net new in their systems. In enterprise deployments, the license key is not unique. In these cases, sysprep is highly important to resolve activation issues.When the new system is booted, it will generate a net new System UUID and store it and collected identifiers in their systems.

3.1.6 Special Cases

Special cases deal with environments that make it difficult, or nearly impossible to uniquely identify a system. These include virtual machines, containers, and clones. These issues can be overcome by ensuring that users cannot modify MAC addresses or IP addressing on these systems. All the platforms in this analysis provide mechanisms to ensure that the UUID is unique. In Windows, this is "sysprep", and on Linux/Mac computers is the removal of /etc/machine-id and /var/lib/dbus/machine-id on perspective systems. Once these systems are restarted, a new UUID will be generated. These files should be protected from user modification.

Clones will pick up new Hardware identifiers which greatly ensures uniqueness. Take for example a clone of the OS on a Dell laptop. If you apply this clone to another Dell laptop with the exact same model/configuration, the identifiers such as Service ID (Dell), motherboard Serial number, hard drive identifiers, and other peripheral devices will change.

Virtual machines and containers on the other hand will not see new hardware identifiers, so you need to rely more on proper image preparation as identified above.

3.1.7 3rd Party Tools

Various 3rd party tools exist that can assist in the unique identification of systems on your network. One such tool for Windows is CPU-Z (https://www.cpuid.com/softwares/cpu-z.html). Similar tools exist for Linux and Mac. These tools can be run within a GUI or from the command line, and information collected can be centrally stored and parsed.

Other tools such as many of the network management systems can also uniquely identify systems and use various approaches to accomplish this. One such is FusionInventory (http://fusioninventory.org/) which collects many of the above identifiers to generate a UUID for each system. See (http://fusioninventory.org/documentation/dev/spec/protocol/inventory.html) for more information.

Other fingerprinting applications exist. One such is QuickSet (https://quicksetcloud.com/2018/06/29/device-fingerprinting-across-home-networks/) which goes way beyond just Windows, Linux, or Mac.

3.2 CODE EXAMPLES

These are various code examples on how to collect information and then either create a UUID which is stored on the device, and/or the collected information is stored on a server and used with ranking and acceptable changes to identify the device.

3.2.1 Example 1

This example is from https://oroboro.com/unique-machine-fingerprint/ .

The c++ code provided by the author shows technique that collects individual id's and hashes them to create a fingerprint for the device. It also a technique to accommodates the changing of a few of them and allows the match to still be met.

The code also includes source for Windows, Mac OS and Linux.

The code can be expanded to collect more identifiers, but instead of doing the hash, send the collected data back to a central server to be stored. This could then be used to track the changes and to provide more flexible rules for deciding a match.

3.2.2 Example 2 – Various command line tools per platform.

Here are listed a number of command line utilities per platform that can be used to extract various pieces of information.

3.2.2.1 Windows

Windows has many commands available to query various data points about the OS and applications, as well as information about the hardware. Both the older cmd prompt as well as PowerShell can be used. Please note that some information can only be queried if the account has the privileges to do so. These can also be collected through various computer languages.

3.2.2.2 Linux

Linux provides shell commands to query the same information that Windows or MacOS can. These can also be queried by PowerShell for linux as well as using various computer languages. One of the most powerful is “dmidecode”. It returns everything from hardware to OS. The following is an extract:

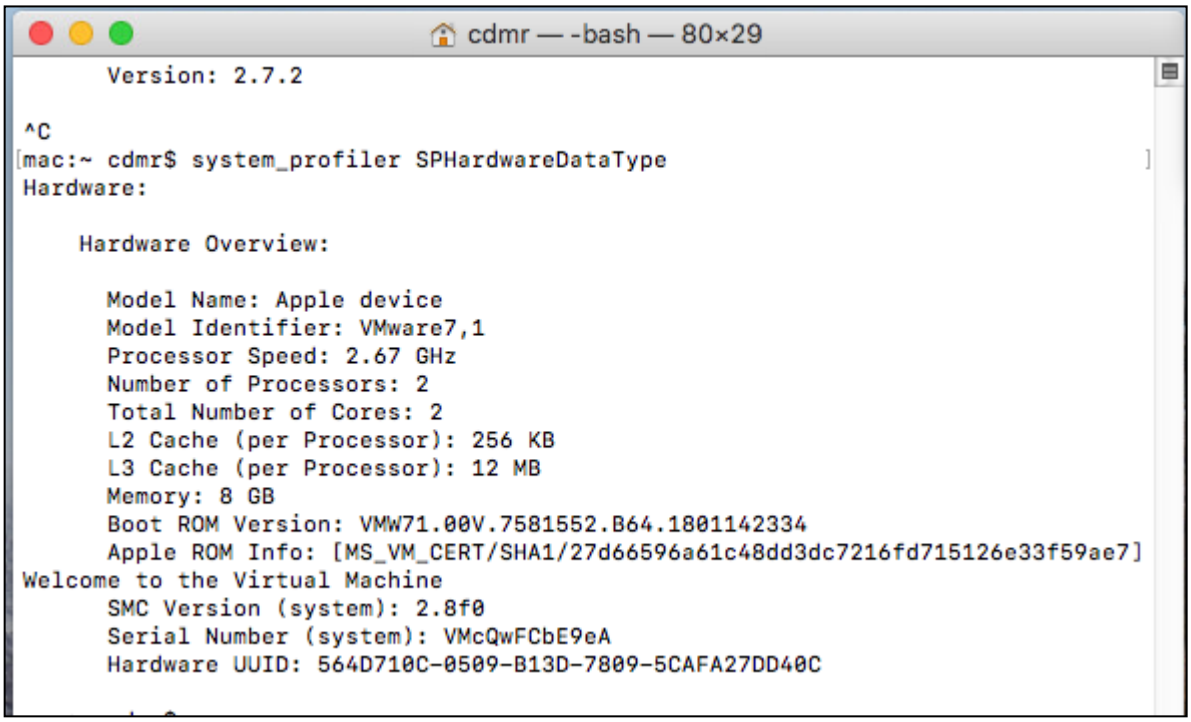
```
BIOS Information
        Vendor: Phoenix Technologies LTD
        Version: 6.00
        Release Date: 07/28/2017
        Address: 0xEA520
        Runtime Size: 88800 bytes
        ROM Size: 64 kB
        Characteristics:
                ISA is supported
                PCI is supported
                PC Card (PCMCIA) is supported
                PNP is supported
                APM is supported
                BIOS is upgradeable
                BIOS shadowing is allowed
                ESCD support is available
                Boot from CD is supported
                Selectable boot is supported
                EDD is supported
                Print screen service is supported (int 5h)
                8042 keyboard services are supported (int 9h)
                Serial services are supported (int 14h)
                Printer services are supported (int 17h)
                CGA/mono video services are supported (int 10h)
                ACPI is supported
                Smart battery is supported
                BIOS boot specification is supported
                Function key-initiated network boot is supported
                Targeted content distribution is supported
        BIOS Revision: 4.6
        Firmware Revision: 0.0

Handle 0x0001, DMI type 1, 27 bytes
System Information
        Manufacturer: VMware, Inc.
        Product Name: VMware Virtual Platform
        Version: None
        Serial Number: VMware-56 4d 46 3e c7 ca 22 d6-98 69 9d 81 60 65 89 d2
        UUID: 3E464D56-CAC7-D622-9869-9D81606589D2
        Wake-up Type: Power Switch
        SKU Number: Not Specified
        Family: Not Specified
```

One important piece of information on Linux is the machine-id. This value is either in “/etc/machine-id” or in “/var/lib/dbus/machine-id”. When cloning a Linux machine, this value needs to be cleared by removing the

file to force the OS to generate a new machine-id. On the same box that returned the above values, the /etc/machine-id contains “60eea0facf434ad7827f92e361d54d9f”.

3.2.2.3 MacOS

Like the other platforms, MacOS also provides many methods to collect information from a shell prompt, as well as through various computer languages.



Running “system_profiler SPHardwareDataType” will return:

As you can see, you can get quite a bit of information about the device and its OS. “system_profiler” has many options, you can collect even more information.

Another command that can also extract the information is “ioreg” this command is very powerful. Here is an example “ioreg -rd1 -c IOPlatformExpertDevice” which returns:

```
mac:~ cdmr$ ioreg -rd1 -c IOPlatformExpertDevice
+-o VMware7,1  <class IOPlatformExpertDevice, id 0x100000112, registered, matched, active, busy 0 (37162 ms), retain 23>
    {
      "compatible" = <"VMware7,1">
      "version" = <"None">
      "board-id" = <"440BX Desktop Reference Platform">
      "IOInterruptSpecifiers" = (<0900000007000000>)
      "IOPolledInterface" = "SMCPolledInterface is not serializable"
      "serial-number" = <453965410000000000000000000564d6351774643624539654100000000000000000000000000000000000000>
      "IOInterruptControllers" = ("io-apic-0")
      "IOPlatformUUID" = "564D710C-0509-B13D-7809-5CAFA27DD40C"
      "target-type" = <"Mac">
      "clock-frequency" = <00e1f505>
      "manufacturer" = <"VMware, Inc.">
      "IOPlatformSerialNumber" = "VMcQwFCbE9eA"
      "product-name" = <"VMware7,1">
      "IOBusyInterest" = "IOCommand is not serializable"
      "model" = <"VMware7,1">
      "name" = <"/">
    }
```

Here you can see quite a bit of information that can be used in fingerprinting the device.

4 Conclusion

In order to easily uniquely identify a system running Windows, Linux, or MacOS, you need three things.

1. Administrative rights to the targeted systems.
2. Centralized system to store collected information.
3. Process/algorithm to follow to scan for values process those values, to generate a unique identifier to be stored both in the centralized system and on the targeted systems.

Special consideration needs to be given to cloned systems, containers, and virtual machines. Each of these can cause issues but can be overcome by proper prepping of the clone to generate new OS specific unique identifiers.

By using the above listed identifiers, you should be able to develop a system with a near 100% assurance level of uniqueness without requiring manual intervention.

DOCUMENT CONTROL DATA

*Security markings for the title, authors, abstract and keywords must be entered when the document is sensitive

1. ORIGINATOR (Name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or tasking agency, is entered in Section 8.)

TRM

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

CAN UNCLASSIFIED

2b. CONTROLLED GOODS

NON-CONTROLLED GOODS
DMC A

3. TITLE (The document title and sub-title as indicated on the title page.)

Network Asset Identification

4. AUTHORS (Last name, followed by initials – ranks, titles, etc., not to be used)

O'Leary, D.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.)	6b. NO. OF REFS (Total references cited.)
February 2019	40	23

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter.)

Contract Report

8. SPONSORING CENTRE (The name and address of the department project office or laboratory sponsoring the research and development.)

DRDC – Ottawa Research Centre
Defence Research and Development Canada, Shirley's Bay
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4
Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
05ac - Cyber Decision Making and Response (CDMR)	W7714-176208

10a. DRDC PUBLICATION NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC-RDDC-2019-C267	Task ID: 0004; Version 1.0.1

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

Asset Inventory; Detection and Identification; Identification; Cyber Capabilities

13. ABSTRACT/RÉSUMÉ (When available in the document, the French version of the abstract must be included here.)

Background/Contexte - The Cyber Operations and Signals Warfare (COSW) section at Defence Research and Development Canada (DRDC) has been conducting significant work in the area of automated Computer Network Defence (CND). To accurately identify problems in the network, and to offer remediation techniques, requires that a computing system within the environment be uniquely identified by the systems in question. Current methods of identifying a system through parameters such as MAC address, IP address, or hostname have limitations due to the ability to modify these parameters. The goal is to identify a list of parameters that can be collected about a system, that when used together (either in full or in part), can ensure the unique identification of a system in question.

La section Guerre des transmissions et des cyber-opérations (COSW) à Recherche et développement pour la défense Canada (DRDC) a mené d'importants travaux dans le domaine de la défense automatisée de réseau informatique (DND). Pour identifier avec précision les problèmes du réseau et proposer des techniques de correction, un système informatique de l'environnement doit être identifié de manière unique par les systèmes en question. Les méthodes actuelles d'identification d'un système via des paramètres tels que l'adresse MAC, l'adresse IP ou le nom d'hôte ont des limites en raison de la possibilité de modifier ces paramètres. L'objectif est d'identifier une liste de paramètres pouvant être collectés sur un système, qui, utilisés ensemble (en tout ou en partie), peuvent assurer l'identification unique du système en question.

Results/Résultats – The results show that the combination of the various collectable parameters from Windows, Mac OS, and Linux platforms can be used to uniquely identify that device with a high level of confidence. The results indicate that while some are modifiable through regular device use and updates, that many only change when the physical hardware changes. It also shows that various methods exist to collect and store these parameters for continuous assurance of unique matches.

Les résultats montrent que la combinaison des divers paramètres collectables des plates-formes Windows, Mac OS et Linux peut être utilisée pour identifier de manière unique ce périphérique avec un niveau de confiance élevé. Les résultats indiquent que, même si certains sont modifiables par le biais de l'utilisation régulière de périphériques et de mises à jour, beaucoup ne changent que lorsque le matériel physique change. Il montre également qu'il existe différentes méthodes pour collecter et stocker ces paramètres afin de garantir en permanence des correspondances uniques.

Conclusion - By using the documented identifiers, you should be able to develop a system with a near 100% assurance level of uniqueness without requiring manual intervention.

En utilisant les identifiants documentés, vous devriez être capable de développer un système avec un niveau de garantie d'unicité proche de 100% sans nécessiter d'intervention manuelle.