Networking

- MAC Addresses
- Microsoft Network Monitor
- Netplan, Bonding, and VLANs
- OSPF
- Ruckus / Brocade
- The Most Common OpenSSL Commands
- Wireshark
- Finding listening ports on any operating system
- Microsoft netsh trace
- Port Mirror / Monitor
- tcpdump
- IPv6
- pktmon - tcpdump for Windows
- Using a SOCKS proxy with OpenSSH
- Monitoring
    - Smokeping
- HPE Aruba

MAC Addresses

MAC OUI

Reserved OUIs

Addresses	Usage	Reference
00-00-00 to 00-00-FF	Reserved	[RFC7042]
00-01-00 to 00-01-FF	VRRP (Virtual Router Redundancy Protocol)	[RFC5798]
00-02-00 to 00-02-FF	VRRP IPv6 (Virtual Router Redundancy Protocol IPv6)	[RFC5798]
00-03-00 to 00-51-FF	Unassigned	
00-52-00	PacketPWEthA	[RFC6658]
00-52-01	PacketPWEthB	[RFC6658]
00-52-02	BFD for VXLAN	[RFC8971]
00-52-03 to 00-52-12	Unassigned (small allocations)	
00-52-13	Proxy Mobile IPv6	[RFC6543]
00-52-14 to 00-52-FF	Unassigned (small allocations)	
00-53-00 to 00-53-FF	Documentation	[RFC7042]
00-54-00 to 90-00-FF	Unassigned	
90-01-00	TRILL OAM	[RFC7455]
90-01-01 to 90-01-FF	Unassigned (small allocations requiring both unicast and multicast)	
90-02-00 to FF-FF-FF	Unassigned	

Source

OUIs of virtualization platforms

Company and Products	MAC unique identifier (s)
VMware ESX 3, Server, Workstation, Player	00-50-56, 00-0C-29, 00-05-69
Microsoft Hyper-V, Virtual Server, Virtual PC	00-03-FF 00-15-5D
Parallells Desktop, Workstation, Server, Virtuozzo	00-1C-42
Virtual Iron 4	00-0F-4B
Red Hat Xen	00-16-3E
Oracle VM	00-16-3E
XenSource	00-16-3E
Novell Xen	00-16-3E
Sun xVM VirtualBox	08-00-27

Source

End

Microsoft Network Monitor

I had never heard of this tool until today... I've always used Wireshark. Today I needed to view traffic broken out by application (PID/ProcessName). I went hunting and found the Microsoft Network Monitor. Surprisingly it's very feature rich, easy to use, and did exactly what I needed it to do... and sooo much more. Check it out!

Microsoft Links

- Information about Network Monitor 3
- Download Microsoft Network Monitor 3.4 (archive)

Example Filters

Filter out the noise (no RDP or VNC)

```
!(tcp.port==3389) and !(udp.port==3389) and !(tcp.port==5900)
```

Capture only DNS:

```
DNS
```

Filter Source or Destination IPv4 Address:

```
IPv4.Address == 1.1.1.1
```

Filter Source IPv4 Address:

```
IPv4.SourceAddress == 1.1.1.1
```

Filter IPV4 Source and Destination:

```
IPv4.Address==1.1.1.1 and IPv4.Address==2.2.2.2
```

Filter IPv4 Source or Destination to subnet:

```
((ipv4.Address & 255.0.0.0) == 10.0.0.0)
```

Filter IPv4 traffic to private only traffic (source and destination in RFC-1918 private subnets):

```
(((IPv4.SourceAddress & 255.0.0.0) == 10.0.0.0) || ((IPv4.SourceAddress & 255.240.0.0) == 172.16.0.0) || ((IPv4.SourceAddress & 255.255.0.0) == 192.168.0.0))
&&
(((IPv4.DestinationAddress & 255.0.0.0) == 10.0.0.0) || ((IPv4.DestinationAddress & 255.240.0.0) == 172.16.0.0) || ((IPv4.DestinationAddress & 255.255.0.0) == 192.168.0.0))
```

Filter CDP traffic

```
Ethernet.Address == 01-00-0c-cc-cc-cc
```

Filter LLDP traffic

```
LLDP
```

Filter Mikrotik MNDP traffic

Microsoft netmon has no protocol disassembler for the MNDP protocol. All you will see is a UDPPayloadData Binary Large Object, however, you can see data in the Hex Details view and can extract the data you need from there fairly easily.

```
(udp.DstPort==5678 AND udp.SrcPort==5678)
```

Filter CDP + LLDP + MNDP

```
Ethernet.Address == 01-00-0c-cc-cc-cc
OR
LLDP
OR
(udp.DstPort==5678 AND udp.SrcPort==5678)
```

Filter traffic by ProcessName

The filter below allows you to see if a process is communicating with any other IP address besides the one you listed:

```
ProcessName.Contains("WindTerm.exe") && IPv4.Address!= 9.9.9.9
```

Filtering NPS + Azure MFA

The Azure MFA NPS Extension uses HTTPS to communicate with login.microsoftonline.com and credentials.azure.com. The filters below enable capturing related traffic.

Suggested capture filter:

```
// Suggested capture filter
tcp.port == 443         // HTTPS
OR udp.port == 1812     // RADIUS
OR DNS.Qrecord.QuestionName.contains("login.microsoftonline.com")
OR DNS.Qrecord.QuestionName.contains("credentials.azure.com")
```

Suggested display filter:

```
// Suggested display filter
udp.port==1812 // RADIUS packets
OR DNS.Qrecord.QuestionName.contains("login.microsoftonline.com")
OR DNS.Qrecord.QuestionName.contains("credentials.azure.com")
OR ContainsBin(FrameData, ASCII, "login.microsoftonline.com") // Will show HTTPS certificate
negotiation packets
OR ContainsBin(FrameData, ASCII, "credentials.azure.com")     // Will show HTTPS certificate
negotiation packets
OR ((ipv4.SourceAddress & 255.255.0.0) == 20.190.0.0) || ((ipv4.DestinationAddress &
255.255.0.0) == 20.190.0.0)
OR ((ipv4.SourceAddress & 255.255.0.0) == 40.126.0.0) || ((ipv4.DestinationAddress &
255.255.0.0) == 40.126.0.0)
```

Example on other sites:

- Network Monitor Filter Examples

Running nmcap from the cmd prompt

Run the following command from anywhere to view the command line usage.

```
"C:\Program Files\Microsoft Network Monitor 3\nmcap.exe" /usage
```

Run the following command from anywhere to view the network adapters.

```
"C:\Program Files\Microsoft Network Monitor 3\nmcap.exe" /displaynetworks
```

After determining the network adapter you would like to perform the capture on, grab the command below, update the capture network interface, the capture filter, and the stop / start conditions.

```
"C:\Program Files\Microsoft Network Monitor 3\nmcap.exe" /network 4 /capture "dns || icmp || ((ipv4.Address & 255.255.248.0) == 104.244.40.0)" /CaptureProcesses /file C:\tmp\nmcap-capture-testing.cap  /TerminateWhen /KeyPress x /StopWhen /TimeAfter 30 min
```

Running nmcap from PowerShell

Run the following command from anywhere to view the command line usage.

```
& 'C:\Program Files\Microsoft Network Monitor 3\nmcap.exe' /usage
```

Run the following command from anywhere to view the network adapters.

```
& 'C:\Program Files\Microsoft Network Monitor 3\nmcap.exe' /displaynetworks
```

After determining the network adapter you would like to perform the capture on, grab the command below, update the capture network interface, the capture filter, and the stop / start conditions.

```
& 'C:\Program Files\Microsoft Network Monitor 3\nmcap.exe' /network 4 /capture "dns || icmp || ((ipv4.Address & 255.255.248.0) == 104.244.40.0)" /CaptureProcesses /file C:\tmp\nmcap-capture-testing.cap  /TerminateWhen /KeyPress x /StopWhen /TimeAfter 30 min
```

nmcap full command line usage

```
PS C:\> & 'C:\Program Files\Microsoft Network Monitor 3\nmcap.exe' /usage
Network Monitor Command Line Capture (nmcap) 3.4.2350.0

Options:
 /Help /? /Usage
      Displays this message.
```

```
     Example Usage: nmcap /Usage

/Example(s)
     Displays a list of examples.
     Example Usage: nmcap /Example
                    nmcap /examples

/TimeFormat
     Displays the list of date and time formats available for the /Time switch.
     Example Usage: nmcap /TimeFormat

/DisplayNetwork(s)
     Displays the network adapters that Network Monitor can capture from.
     Example Usage: nmcap /DisplayNetwork
                    nmcap /displaynetworks

/SetNplPath <path>[;<path>;...]
     Builds a profile titled "NMCap Last Path" and attempts to compile.
     If compilation is successful, the new profile is set as active.
     If the provided NPL cannot compile, the Active Profile is unchanged.
     On success, any existing "NMCap Last Path" profile is updated.

/DisplayNplPath
     Displays the NPL path.
     Example Usage: nmcap /DisplayNplPath
     Note:
     Nmcap gets the NPL files in the following order:
     1) Files in the current directory
     2) Files in the path set by /SetNplPath
     3) Default NMCap installation directory

/DisplayProfiles
     Display the installed profiles.

/DisplayProfileInfo <Profile Key>
     Display detailed information for the indicated profile.
     A Profile Key can be either the GUID or the Index displayed when using
     the /DisplayProfiles argument.

/UseProfile <Profile Key>
```

```
     Use an alternate profile for the capture session. This setting must be
     before any /Capture arguments. Otherwise, the active profile is used.
     A Profile Key can be either the GUID or the Index displayed when using
     the /DisplayProfiles argument.

/SetActiveProfile <Profile Key>
     Set the active profile for to the indicated profile. This setting changes
     the default profile for other Network Monitor applications, as well.
     A Profile Key can be either the GUID or the index displayed when using
     the /DisplayProfiles argument.

/DeleteProfile <Profile Key>
     Delete the indicated profile.
     Only user-defined profiles can be deleted.
     A Profile Key can be either the GUID or the index displayed when using
     the /DisplayProfiles argument.

/SetDefaultParser [NplFileName.npl]
     Specifies the default NPL parser
     Example Usage: nmcap /SetDefaultParser sparser.npl

/DisplayDefaultParser
     Displays the default NPL parser.
     Example Usage: nmcap /DisplayDefaultParser
     Note: If you do not explicitly set the default parser using
     /SetDefaultParser, /DisplayDefaultParser does not display anything.

/MaxFrameLength <Number of Bytes>
     Specifying the Max Frame Length limits the number of bytes captured per
     frame to the specified value.  If you enter a value of 68, every
     frame is truncated to the first 68 bytes.  This has an impact
     on filtering because the filter may require elements defined in the frame
     which are no longer present as a result of the truncation.
     Example Usage:  The following captures all traffic on all adapters
     and limits the captured data to the first 68 bytes.
     nmcap /network * /MaxFrameLength 68
     Note: This does not apply to frames retrieved from /inputcapture files.

The following options are used together and have no meaning independently.
Refer to the examples to better understand how they can be used together.
```

```
Specifying Input Sources:

   /Network <Network Adapter> [<network Adapter> ...]
      Selects one or more space-delimited network adapters to capture from.
      Adapters may be specified using their index, partial name with wildcard (*), or
      quoted friendly name. (find using /DisplayNetwork above).
      Example Usage: /Network inte* 2 'Local Area Connection 1'

   /InputCapture <CaptureFile> [<CaptureFile> ...]
      Selects one or more space-delimited capture files to capture from.
      The frames in the capture files are replayed through NMCap.
      Example Usage: /InputCapture dns.cap tcp.cap c:\temp\test.cap

   /DisableConversations
      Disables conversations. This enhances the performance of
      NMCap. Some protocols such as MSRPC require conversation to be enabled.
      Example Usage: /DisableConversations

   /CaptureProcesses
      Enables process tracking. This is incompatible with the
      /DisableConversations switch as process tracking requires conversations.
      Example Usage: /CaptureProcesses

   /DisableLocalOnly
      Disables local-only capture. This enables the capture in p-mode. All
      frames seen by this computer are captured.
      Example Usage: /DisableLocalOnly

   /RecordFilters
      Records the capture filters in the capture file.
      Example Usage: /RecordFilters

   /RecordConfig
      Records the network configuration in the capture file.
      Example Usage: /RecordConfig

   /MinDiskQuotaPercentage <Minimal Disk Size Percentage>
      Specifies the minimal disk size percentage allowed.
      Note: Default minimal disk size percentage is 2 percent.
```

```
      Example Usage: /MinDiskQuotaPercentage 20
      (Sets the minimal disk size percentage to 20 percent.)

   /MinDiskQuota:<Minimal disk Size>
      Specifies the minimal disk size allowed.
      Example Usage: /MinDiskQuota 20M
      (Sets the minimal disk size to 20 MB.)

  /Frame <Filter>
      Frame filter. The frame filter is constructed from the NPL files.
      You can use all the filter expressions that you can in the Netmon UI.
      For more sample filters, type nmcap /Examples or see Standards Filter
      in the Filter Toolbar of the UI.
      This switch must be part of /StartWhen, /StopWhen, and /TerminateWhen.
      Example Usage: /Frame Dns.Flags.Stats == 3


Capture File output:

  /Capture [FrameFilter] /File <CaptureFile> [/File <CaptureFile>...]
      Saves frames that pass the frame filter to the specified capture files.
      Example Usage: /Capture dns.flags.status == 3 /File t.cap /File t2.cap

  /ReassembleCapture [FrameFilter] /File <CaptureFile> [/File <CaptureFile>...]
      An alternate to Capture (above) Saves frames that pass the frame filter
      to the specified capture files, along with reassembled payloads.
      Example Usage: /ReassembleCapture tcp /File tcp.cap /File tcp2.cap

  /File <Capture File>[:<File Size Limit>]
      Name of capture file to save frames to. Extensions are used to determine
      the behavior of NMCap.
       .cap -- Netmon 2 capture file
       .chn -- Series of Netmon 2 capture files: t.cap, t(1).cap, t(2).cap...
      <File Size Limit> is optional. It limits the file size of each capture
      file generated. Default single capture file size limit is 20 MB. The
      upper bound of the file size limit is 500 MB. The lower bound of the file
      size limit depends on the frame size captured. (Note that the maximal size
      of Ethernet frames is 1500 bytes)
      The files are circular, so once the size limit is reached, new data
      overwrites older data.
```

```
      Example Usage: /File t.cap:50M

Starting, stopping, and events:

  /StartWhen <Command Line Switch>
      List of conditions that specify when to start capturing network frames.
      If it is ignored, NMCap starts capturing immediately. When input
      from capture files by /InputCapture, this switch is generally ignored.
      Example Usage: /StartWhen /Time 7:02:00 AM 9/10/2005

  /StopWhen <Command Line Switch>
      List of conditions that specify when to stop capturing network frames.
      Example Usage: /StopWhen /TimeAfter 20 min
      This switch becomes active only after a preceding /StartWhen evaluates
      to TRUE.
      Example: /StartWhen /TimeAfter 10 min ... /StopWhen /TimeAfter 20 min
      Nmcap starts after 10 minutes and stops after 10+20=30 minutes.
      /StopWhen never terminates the program while the /StartWhen option
      is set to FALSE.  /TerminateWhen should be used to safely terminate immediately.

  /TimeAfter <number>[units]
      Indicates a time period. This switch can be part of /StartWhen, /StopWhen,
      and /TerminateWhen. The user specifies the time units. By default it
      is in seconds.
      Example Usage: /TimeAfter 20 seconds

  /Time <Time/Date>
      Indicates a time of day. This switch can be part of /StartWhen, /StopWhen,
      and /TerminateWhen. The time and date format depends on the settings in
      the 'Region and Language' Control Panel.
      Example Usage: /Time 10:30:00 AM 9/10/2005

  /TerminateWhen <Command Line Switch>
      This switch terminates NMCap immediately once it evaluates to TRUE.
      Example Usage: /TerminateWhen /KeyPress x
      Note:
      The default relationship among the conditions following /Startwhen,
      /Stopwhen, and /TerminateWhen is AND and the order is sensitive.
      For example: /TerminateWhen /Timeafter 10 min /Keypress x
      Nmcap terminates after 10 minutes have passed and the user presses the 'x' key.
```

```
        And:         /TerminateWhen /Keypress x /Timeafter 10 min
        Nmcap terminates 10 minutes after the user press 'x' key.

    /KeyPress <character>
        Specifies which key to press. This switch can be part of /StartWhen,
        /StopWhen, and /TerminateWhen.
        Example Usage: /KeyPress z
```

-end

Netplan, Bonding, and VLANs

Creating a bond interface with Netplan

The example below shows two Ethernet interfaces bonded using the active-passive mode.

The following packages are required: ifenslave and vlan

```
network:
  version: 2
  ethernets:
    enp1s0:
      dhcp4: no
      optional: true
    enp2s0:
      dhcp4: no
      optional: true
    # match all other ports if desired
    ethernetPorts:
      dhcp4: no
      optional: true
      match:
        name: eth*|em*|en*
  bonds:
    bond0:
      interfaces: [ enp1s0, enp2s0, ethernetPorts ]
      addresses: [ 192.168.168.115/24 ]
      gateway4: 192.168.168.1
      nameservers:
        search: [ mydomain.com ]
        addresses: [ 192.168.168.1, 8.8.8.8, 8.8.4.4 ]
      parameters:
        mode: active-backup
        primary: enp1s0
        primary-reselect-policy: always
```

```
        mii-monitor-interval: 100
        up-delay: 3s # must be a multiple of mii-monitor-interval, make sure its longer than
STP/RSTP/MSTP learning interval also
#      routes:
#        - to: 10.0.0.0/8
#          via: 192.168.168.1
#        - to: 172.16.0.0/12
#          via: 192.168.168.1
#        - to: 192.168.0.0/16
#          via: 192.168.168.1
```

Notes:

- "***optional: true***" instructs Netplan to boot the operating system even if the network interface is unavailable or not connected or unavailable. Without this option, the system will not fully boot until all network cables are connected.
- "***mii-monitor-interval: 100***" must be set to some value or link up / down events will not actually be detected. A value of zero, which is the default, will disable the detection of interface changes, which seems rather counter-intuitive when we're configuring the mode as active-backup.
- "***up-delay: 10000***" prevents packet loss when connecting an interface. It must be a multiple of the mii-monitor-interval value.

Using a VLAN on a bond interface

This is a configuration using from Ubuntu 18.04 LTS. Two Ethernet interfaces are bonded using the active-passive mode. The untagged bond0 interface is for private traffic, while a public IP address is being delivered to a tagged VLAN sub interface using VLAN 262.

The following packages are required: ifenslave and vlan

```
network:
  version: 2
  ethernets:
    enp1s0:
      dhcp4: no
      optional: true
    enp2s0:
      dhcp4: no
      optional: true
    # match all other ports if desired
    ethernetPorts:
```

```
      dhcp4: no
      optional: true
      match:
        name: eth*|em*|en*
  bonds:
    bond0:
      interfaces: [ enp1s0, enp2s0, ethernetPorts ]
      addresses: [ 192.168.168.115/24 ]
      parameters:
        mode: active-backup
        primary: enp1s0
        primary-reselect-policy: always
        mii-monitor-interval: 100
        up-delay: 3s # must be a multiple of mii-monitor-interval, make sure its longer than
STP/RSTP/MSTP learning interval also
#      routes:
#        - to: 10.0.0.0/8
#          via: 192.168.168.1
#        - to: 172.16.0.0/12
#          via: 192.168.168.1
#        - to: 192.168.0.0/16
#          via: 192.168.168.1

  vlans:
    bond0.262:
        id: 262
        link: bond0
        addresses: [ 1.1.1.123/28 ]
        gateway4: 1.1.1.113
        nameservers:
            search: [ servers.domain.com ]
            addresses: [ 1.1.1.113, 8.8.8.8, 8.8.4.4 ]
```

systemd-networkd-wait-online.service

Ubuntu 22.04

In Ubuntu 22.04, even though the netplan configuration is correct, the service systemd-networkd-wait-online.service will wait for 120 seconds if one of the network interfaces is not connected.

In order to get around this, one solution is to add the "--any" option to the ExecStart line as shown below. You can also reduce the default timeout from 120 seconds to 10 seconds by adding the "--timeout=10" option.

On Ubuntu, the file is located at /etc/systemd/system/
On Debian, the file is located at /var/lib/systemd/system/

```
[Unit]
Description=Wait for Network to be Configured
Documentation=man:systemd-networkd-wait-online.service(8)
DefaultDependencies=no
Conflicts=shutdown.target
Requires=systemd-networkd.service
After=systemd-networkd.service
Before=network-online.target shutdown.target

[Service]
Type=oneshot
ExecStart=/lib/systemd/systemd-networkd-wait-online --any --timeout=10
RemainAfterExit=yes

[Install]
WantedBy=network-online.target
```

Alternatively, you could just disable and mask the service altogether as it actually isn't needed. If you are going to disable the service, I would strongly recommend adding the two options shown above in addition just in case the service gets re-enabled in the future.

Debian 11 (Raspberry Pi)

On a Raspberry Pi running Debian 11.5, we had a similar issue. We never were able to get rid of the message, but making the above configuration changes made sure the system booted properly regardless of how many Ethernet interfaces were physically connected to switches at boot time.

Error during boot:

```
[FAILED] Failed to start Wait for Network to be Configured.
See 'systemctl status systemd-networkd-wait-online.service' for details.
```

Output from systemctl status commands:

```
# OUTPUT FROM systemctl status systemd-networkd-wait-online.service
    systemd-networkd-wait-online.service - Wait for Network to be Configured
```

```
     Loaded: loaded (/lib/systemd/system/systemd-networkd-wait-online.service; enabled-
runtime; vendor preset: disabled)
     Active: failed (Result: exit-code) since Thu 2023-05-18 17:10:59 BST; 2min 44s ago
       Docs: man:systemd-networkd-wait-online.service(8)
    Process: 189 ExecStart=/lib/systemd/systemd-networkd-wait-online (code=exited,
status=1/FAILURE)
   Main PID: 189 (code=exited, status=1/FAILURE)
        CPU: 148ms

Aug 07 14:25:36 6230dea99f07e90f52b5f68b systemd[1]: Starting Wait for Network to be
Configured...
May 18 17:10:59 6230dea99f07e90f52b5f68b systemd-networkd-wait-online[189]: Event loop failed:
Connection timed out
May 18 17:10:59 6230dea99f07e90f52b5f68b systemd[1]: systemd-networkd-wait-online.service:
Main process exited, code=exited, status=1/FAILURE
May 18 17:10:59 6230dea99f07e90f52b5f68b systemd[1]: systemd-networkd-wait-online.service:
Failed with result 'exit-code'.
May 18 17:10:59 6230dea99f07e90f52b5f68b systemd[1]: Failed to start Wait for Network to be
Configured.
```

References

https://netplan.io/examples

https://www.freedesktop.org/software/systemd/man/systemd-networkd-wait-online.service.html

OSPF

OSPF Cost

```
cost = reference bandwidth / configured bandwidth of interface in kbps


reference bandwidth = 100,000 kbps
```

Interface speed	OSPF Cost
100 Mbps	1
10 Mbps	10
6 Mbps	17
5 Mbps	20
4 Mbps	25
3 Mbps	33
2 Mbps	50
1.5 Mbps	67
1 Mbps	100
768 Kbps	130
512 Kbps	195
384 Kbps	260
256 Kbps	391
128 Kbps	781
64 Kbps	1563

Ruckus / Brocade

Links

https://robrobstation.com/2017/07/17/ruckus-icx7150-c12p-initial-configuration/

The Most Common OpenSSL Commands

General OpenSSL Commands

These commands allow you to generate CSRs, Certificates, Private Keys and do other miscellaneous tasks.

Generate a new private key and Certificate Signing Request

```
openssl req -out CSR.csr -new -newkey rsa:2048 -nodes -keyout privateKey.key
```

Generate a self-signed certificate (see How to Create and Install an Apache Self Signed Certificate for more info)

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt
```

Generate a certificate signing request (CSR) for an existing private key

```
openssl req -out CSR.csr -key privateKey.key -new
```

Generate a certificate signing request based on an existing certificate

```
openssl x509 -x509toreq -in certificate.crt -out CSR.csr -signkey privateKey.key
```

Remove a passphrase from a private key

```
openssl rsa -in privateKey.pem -out newPrivateKey.pem
```

Checking Using OpenSSL

If you need to check the information within a Certificate, CSR or Private Key, use these commands. You can also check CSRs and check certificates using our online tools.

Check a Certificate Signing Request (CSR)

```
openssl req -text -noout -verify -in CSR.csr
```

Check a private key

```
openssl rsa -in privateKey.key -check
```

Check a certificate

```
openssl x509 -in certificate.crt -text -noout
```

Check a PKCS#12 file (.pfx or .p12)

```
openssl pkcs12 -info -in keyStore.p12
```

Debugging Using OpenSSL

If you are receiving an error that the private doesn't match the certificate or that a certificate that you installed to a site is not trusted, try one of these commands. If you are trying to verify that an SSL certificate is installed correctly, be sure to check out the SSL Checker.

Check an MD5 hash of the public key to ensure that it matches with what is in a CSR or private key

```
openssl x509 -noout -modulus -in certificate.crt | openssl md5
openssl rsa -noout -modulus -in privateKey.key | openssl md5
openssl req -noout -modulus -in CSR.csr | openssl md5
```

Check an SSL connection. All the certificates (including Intermediates) should be displayed

```
openssl s_client -connect www.paypal.com:443
```

Converting Using OpenSSL

These commands allow you to convert certificates and keys to different formats to make them compatible with specific types of servers or software. For example, you can convert a normal PEM file that would work with Apache to a PFX (PKCS#12) file and use it with Tomcat or IIS. Use our SSL Converter to convert certificates without messing with OpenSSL.

Convert a DER file (.crt .cer .der) to PEM

```
openssl x509 -inform der -in certificate.cer -out certificate.pem
```

Convert a PEM file to DER

```
openssl x509 -outform der -in certificate.pem -out certificate.der
```

Convert a PKCS#12 file (.pfx .p12) containing a private key and certificates to PEM

```
openssl pkcs12 -in keyStore.pfx -out keyStore.pem -nodes
```

You can add -nocerts to only output the private key or add -nokeys to only output the certificates.

Convert a PEM certificate file and a private key to PKCS#12 (.pfx .p12)

```
openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key -in certificate.crt -
certfile CACert.crt
```

Source

View CRL

```
openssl crl -inform DER -text -noout -in crl.crl
```

```
openssl crl -inform PEM -text -noout -in crl.crl
```

Generate a request for Windows CA Services

The process below shows an "easy" way to add the certificateTemplateName to the certificate request that will be submitted to Windows CA Services.

1. Create an openssl configuration file for your request based on the template shown below.
    1. Make sure you update all the subject alternative names entries.
2. Use the script below to create the certificate request.

```
# FILE: server.mydomain.com.openssl.cnf
#
# run the following to generate the request:
# openssl req -newkey rsa:2048 -keyout server.mydomain.com -out server.mydomain.com.req -nodes
-config server.mydomain.com.openssl.cnf

oid_section = OIDs

[OIDs]
certificateTemplateName = 1.3.6.1.4.1.311.20.2

[req]
default_bits = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = US
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = My State
localityName = Locality Name (eg, city)
localityName_default = My City
organizationalUnitName    = Organizational Unit Name (eg, section)
organizationalUnitName_default    = My Org
commonName = Common Name
commonName_default = server.mydomain.com
commonName_max    = 64

[req_ext]
subjectAltName = @alt_names

# Replace PKI-WebServer with your certificate template name, NOT the display name
certificateTemplateName = ASN1:PRINTABLESTRING:PKI-WebServer

[alt_names]
```

```
DNS.1 = server.mydomain.com
DNS.2 = server1.mydomain.com
DNS.3 = server2.mydomain.com
IP.1 = 10.1.2.3
```

```
#!/bin/bash

# change this to match your certificate primary common name
CERTBASE="server.mydomain.com"

# generate timestamp
printf -v TIMESTAMP '%(%Y%m%d-%H%M%S)T' -1

# generate new private key and csr
openssl req -newkey rsa:2048 -keyout "$CERTBASE.$TIMESTAMP.key" -out
"$CERTBASE.$TIMESTAMP.req" -nodes -config "$CERTBASE.openssl.cnf"
```

Wireshark

If you need to do a quick tcpdump like capture from the command line in Windows, don't forget pktmon.

Capture Filters

Download and install

Silently install wireshark and npcap

```
# wireshark silent installer will not install npacp - tested
cd $env:TEMP
Invoke-WebRequest -URI https://1.na.dl.wireshark.org/win64/Wireshark-latest-x64.exe -Out
Wireshark-latest-x64.exe
Start-Process Wireshark-latest-x64.exe -Wait -ArgumentList
@("/D","/S","/desktopicon=no","/quicklaunchicon=no", "/EXTRACOMPONENTS=sshdump,udpdump")

get-process | Sort-Object -Property ProcessName | Where-Object {$_.ProcessName -Like
'Wireshark*'}



# npcap download and install
# only npcap oem supports silent installation
cd $env:TEMP
Invoke-WebRequest -URI https://npcap.com/dist/npcap-1.79.exe -Out npcap-1.79.exe
Start-Process npcap-1.79.exe -Wait -ArgumentList @("/force","/admin_only=yes")

get-process | Sort-Object -Property ProcessName | Where-Object {$_.ProcessName -Like 'npcap*'}
```

MAC address OUI

Source

```
# haven't figured this capture filter out yet... display filter is easy...
```

bootp and dhcp

Source

```
port 67 or port 68
```

Name resolution protocols

DNS

Cisco Discovery Protocol

```
udp port 53
```

mDNS

multicast DNS

```
udp port 5353
```

LLMNR

Link-local multicast name resolution

```
udp port 5355
```

All together now

```
udp port 53 or udp port 5353 or udp port 5355
```

Network discovery protocols

An easy way to view discovery protocol traffic from a laptop is by using Wireshark and the capture filters below for CDP, LLDP and MNDP. Use the appropriate capture filter for the type of device

you're trying to gather information about, or use all three of them in the same capture filter.

CDP

Cisco Discovery Protocol

```
ether host 01:00:0c:cc:cc:cc and ether[16:4] = 0x0300000C and ether[20:2] == 0x2000
```

LLDP

Link Layer Discovery Protocol

```
ether proto 0x88cc
```

MNDP

Mikrotik Discovery Protocol

```
udp dst port 5678 and udp src port 5678
```

CDP/LLDP/MNDP

All three of the above capture filters in one:

```
(ether host 01:00:0c:cc:cc:cc and ether[16:4] = 0x0300000C and ether[20:2] == 0x2000) or
(ether proto 0x88cc) or (udp dst port 5678 and udp src port 5678)
```

Capturing on an interval in Linux

The command below will capture all traffic to/from 8.8.8.8. A new capture file will be created every 600 seconds (10 minutes).

```
dumpcap -b duration:600 -f "host 8.8.8.8" -w capture-google
```

Mikrotik Packet Capture Streaming

To accept only TZSP traffic, Capture Filter like this can be used:

```
udp port 37008
```

Note that TZSP can be sent on any UDP port you set it to, so adjust the above capture as needed.

Using tshark

Interface List

This is typically needed when running tshark on Windows.

```
tshark -D
thsark -i <interface_id>
```

Capture Filter

```
# capture only udp dns packets
tshark -f "udp port 53"
```

Saving Packets

```
# save packets (doesn't display packets)
tsharp -f "udp port 37008" -w captured.pcap

# save and display packets
tsharp -f "udp port 37008" -w captured.pcap -P

# save and display packets with LOTS of detail
tsharp -f "udp port 37008" -w captured.pcap -P -O dns -V
```

Automatic stop

Options are duration:[seconds], filesize:[KB], and files:[n].

```
tshark -a duration:60
tshark -a filesize:1000
```

Ring Buffer Capture

```
tshark -b duration:3600 -b filesize:1000 -b files:24 -w ring_buffer.pcap
tshark -b duration:86400 -b filesize:1000 -b files:30 -w ring_buffer.pcap
```

Practical examples

```
# TZSP stream capture on specific interface
tshark -f "udp port 37008" -i 5

# TZSP stream capture on alternate udp port, uses decode as feature
tshark -f "udp port 37091" -d udp.port==37091,tzsp
```

DNS examples

```
# DNS queries
tshark -n -T fields -e ip.src -e ip.dst -e dns.qry.name -e dns.resp.name -f 'udp port 53'

# DNS query contains specific string
tshark -n -T fields -e dns.qry.name -f 'src port 53' -Y 'dns.qry.name contains "foo"'

# detailed DNS queries and responses
sudo tshark -nn -T fields -e frame.time -e ip.src -e ip.dst -e dns.count.queries -e
dns.count.answers -e dns.qry.name -e dns.qry.type -e dns.resp.name -e dns.resp.type -e
dns.resp.ttl -Y 'dns.flags.rcode==0 && dns.flags.response==1'
```

Display Filters

IP Address in Subnet

```
ip.address == 10.248.1.0/24
```

Radius AVP User-Name contains string

The username could also be the hostname of the device authenticating.

```
radius.User_Name ~ "NIT"
```

Wireless SSID via Radius AVP Called-Station-Id

```
radius.Called_Station_Id ~ "SSID"
```

-end

Finding listening ports on any operating system

ss - the new netstat

```
# show number listening process
ss -nlp | grep -e ^tcp -e ^tcp6 -e ^udp -e ^udp6
```

Output

```
udp   UNCONN 0      0                                    127.0.0.53%lo:53
0.0.0.0:*    users:(("systemd-resolve",pid=1985,fd=13))
udp   UNCONN 0      0                                     10.32.57.255:137
0.0.0.0:*    users:(("nmbd",pid=2033,fd=16))
udp   UNCONN 0      0                                       10.32.57.3:137
0.0.0.0:*    users:(("nmbd",pid=2033,fd=15))
udp   UNCONN 0      0                                          0.0.0.0:137
0.0.0.0:*    users:(("nmbd",pid=2033,fd=13))
udp   UNCONN 0      0                                     10.32.57.255:138
0.0.0.0:*    users:(("nmbd",pid=2033,fd=18))
udp   UNCONN 0      0                                       10.32.57.3:138
0.0.0.0:*    users:(("nmbd",pid=2033,fd=17))
udp   UNCONN 0      0                                          0.0.0.0:138
0.0.0.0:*    users:(("nmbd",pid=2033,fd=14))
udp   UNCONN 0      0                                          0.0.0.0:44900
0.0.0.0:*    users:(("avahi-daemon",pid=2020,fd=14))
udp   UNCONN 0      0                                          0.0.0.0:5353
0.0.0.0:*    users:(("avahi-daemon",pid=2020,fd=12))
udp   UNCONN 0      0                                             [::]:33933
[::]:*    users:(("avahi-daemon",pid=2020,fd=15))
```

```
udp   UNCONN 0      0                                          [::]:5353
[::]:*    users:(("avahi-daemon",pid=2020,fd=13))
tcp   LISTEN 0      4096                                  127.0.0.53%lo:53
0.0.0.0:*    users:(("systemd-resolve",pid=1985,fd=14))
tcp   LISTEN 0      128                                     0.0.0.0:22
0.0.0.0:*    users:(("sshd",pid=2231,fd=3))
tcp   LISTEN 0      10                                    127.0.0.1:25
0.0.0.0:*    users:(("sendmail-mta",pid=2872,fd=4))
tcp   LISTEN 0      50                                      0.0.0.0:445
0.0.0.0:*    users:(("smbd",pid=2438,fd=46))
tcp   LISTEN 0      16                                    127.0.0.1:3493
0.0.0.0:*    users:(("upsd",pid=2410,fd=4))
tcp   LISTEN 0      80                                    127.0.0.1:3306
0.0.0.0:*    users:(("mariadbd",pid=2367,fd=28))
tcp   LISTEN 0      10                                    127.0.0.1:587
0.0.0.0:*    users:(("sendmail-mta",pid=2872,fd=5))
tcp   LISTEN 0      50                                      0.0.0.0:139
0.0.0.0:*    users:(("smbd",pid=2438,fd=47))
tcp   LISTEN 0      128                                        [::]:22
[::]:*    users:(("sshd",pid=2231,fd=4))
tcp   LISTEN 0      50                                         [::]:445
[::]:*    users:(("smbd",pid=2438,fd=44))
tcp   LISTEN 0      16                                        [::1]:3493
[::]:*    users:(("upsd",pid=2410,fd=5))
tcp   LISTEN 0      50                                         [::]:139
[::]:*    users:(("smbd",pid=2438,fd=45))
```

#end

Microsoft netsh trace

Basic

Start the trace:

```
netsh trace start capture=yes Ethernet.Type=IPv4
```

Stop the trace:

```
netsh trace stop
```

Files will be created in %AppData%\Local\Temp\NetTraces

View ETL in Microsoft Network Monitor 3.4

If you load the ETL in Microsoft Network Monitor 3.4, you will see the following error in the packet display:

> MicrosoftWindowsNDISPacketCapture: Windows stub parser: Requires full Common parsers. See the "How Do I Change Parser Set Options(Version 3.3 or before) or Configure Parser Profile (Version 3.4)" help topic for tips on loading these parser sets.

To fix this you need to change the active Parser Profile. Click the Parser Profiles button on the top right of the screen, click Network Monitor Profiles, and then click Windows to set it as the active profile.

Details

```
C:\>netsh trace start help

start
  Starts tracing.

  Usage: trace start [sessionname=<sessionname>]
         [[scenario=]<scenario1,scenario2>]
         [[globalKeywords=]keywords] [[globalLevel=]level]
         [[capture=]yes|no] [[capturetype=]physical|vmswitch|both]
         [[report=]yes|no|disabled] [[persistent=]yes|no]
         [[traceFile=]path\filename] [[maxSize=]filemaxsize]
         [[fileMode=]single|circular|append] [[overwrite=]yes|no]
         [[correlation=]yes|no|disabled] [capturefilters]
         [[provider=]providerIdOrName] [[keywords=]keywordMaskOrSet]
         [[level=]level] [bufferSize=<bufferSize>]
         [[[provider=]provider2IdOrName] [[providerFilter=]yes|no]]
         [[keywords=]keyword2MaskOrSet] [[perfMerge=]yes|no]
         [[level=]level2] ...

Defaults:
        capture=no (specifies whether packet capture is enabled
                in addition to trace events)
        capturetype=physical (specifies whether packet capture needs to be
                enabled for physical network adapters only, virtual switch
                only, or both physical network adapters and virtual switch)
        report=no (specifies whether a complementing report will be generated
                along with the trace file)
        persistent=no (specifies whether the tracing session continues
                across reboots, and is on until netsh trace stop is issued)
        maxSize=250 MB (specifies the maximum trace file size, 0=no maximum)
        bufferSize=512 (specifies trace buffer size in KB, min 4, max 16384)
        fileMode=circular
        overwrite=yes (specifies whether an existing trace output file will
                be overwritten)
        correlation=disabled (specifies whether related events will be
```

```
                correlated and grouped together)
        perfMerge=yes (specifies whether performance metadata is merged
                into trace)
        traceFile=%LOCALAPPDATA%\Temp\NetTraces\[sessionname]NetTrace.etl
                (specifies location of the output file)
        providerFilter=no (specifies whether provider filter is enabled)
        sessionname='' (specifies a name for the trace session so that
                simultaneous traces can be collected.

Provider keywords default to all and level to 255 unless otherwise specified.

For example:

netsh trace start scenario=InternetClient capture=yes

        Starts tracing for the InternetClient scenario and dependent providers
                with packet capture enabled for physical network adapters only.
        Tracing will stop when the "netsh trace stop" command is issued
                or when the system reboots.
        Default location and name will be used for the output file. If an old
                file exists, it will be overwritten.

netsh trace start provider=microsoft-windows-wlan-autoconfig
        keywords=state,ut:authentication

        Starts tracing for the microsoft-windows-wlan-autoconfig provider
        Tracing will stop when the "netsh trace stop" command is issued
                or when the system reboots.
        Default location and name will be used for the output file. If an old
                file exists, it will be overwritten.
        Only events with keyword 'state' or 'ut:authentication' will be logged.

        netsh trace show provider command can be used to display
                supported keywords and levels.

Capture Filters:
        Capture filters are only supported when capture is explicitly
        enabled with capture=yes. Use 'netsh trace show CaptureFilterHelp'
        to display a list of supported capture filters and their usage.
```

```
Provider Filters:
        Provider filters are supported by multiple providers and are enabled
        with providerFilter=Yes after every provider.
        Use 'netsh trace show ProviderFilterHelp' to display a list of
        supported provider filters for each provider and their usage.
```

capturefilterhelp

```
C:\>netsh trace show capturefilterhelp

  Capture Filters:
        Capture filters are only supported when capture is explicitly
        enabled with capture=yes. Supported capture filters are:

        CaptureInterface=<interface name or GUID>
         Enables packet capture for the specified interface name or GUID. Use
         'netsh trace show interfaces' to list available interfaces.
        e.g. CaptureInterface={716A7812-4AEE-4545-9D00-C10EFD223551}
        e.g. CaptureInterface=!{716A7812-4AEE-4545-9D00-C10EFD223551}
        e.g. CaptureInterface="Local Area Connection"

        Ethernet.Address=<MAC address>
         Matches the specified filter against both source and destination
         MAC addresses.
        e.g. Ethernet.Address=00-0D-56-1F-73-64

        Ethernet.SourceAddress=<MAC address>
         Matches the specified filter against source MAC addresses.
        e.g. Ethernet.SourceAddress=00-0D-56-1F-73-64

        Ethernet.DestinationAddress=<MAC address>
         Matches the specified filter against destination MAC addresses.
        e.g. Ethernet.DestinationAddress=00-0D-56-1F-73-64

        Ethernet.Type=<ethertype>
         Matches the specified filter against the MAC ethertype.
```

```
e.g. Ethernet.Type=IPv4
e.g. Ethernet.Type=NOT(0x86DD)
e.g. Ethernet.Type=(IPv4,IPv6)

Wifi.Type=<Management|Data>
 Matches the specified filter against the Wifi type. Allowed values
 are 'Management' and 'Data'. If not specified, the Wifi.Type filter
 is not applied.
 Note: This capture filter does not support ranges, lists or negation.
e.g. Wifi.Type=Management

Protocol=<protocol>
 Matches the specified filter against the IP protocol.
e.g. Protocol=6
e.g. Protocol=!(TCP,UDP)
e.g. Protocol=(4-10)

IPv4.Address=<IPv4 address>
 Matches the specified filter against both source and destination
 IPv4 addresses.
e.g. IPv4.Address=157.59.136.1
e.g. IPv4.Address=!(157.59.136.1)
e.g. IPv4.Address=(157.59.136.1,157.59.136.11)

IPv4.SourceAddress=<IPv4 address>
 Matches the specified filter against source IPv4 addresses.
e.g. IPv4.SourceAddress=157.59.136.1

IPv4.DestinationAddress=<IPv4 address>
 Matches the specified filter against destination IPv4 addresses.
e.g. IPv4.DestinationAddress=157.59.136.1

IPv6.Address=<IPv6 address>
 Matches the specified filter against both source and destination
 IPv6 addresses.
e.g. IPv6.Address=fe80::5038:3c4:35de:f4c3\%8
e.g. IPv6.Address=!(fe80::5038:3c4:35de:f4c3\%8)

IPv6.SourceAddress=<IPv6 address>
 Matches the specified filter against source IPv6 addresses.
```

```
        e.g. IPv6.SourceAddress=fe80::5038:3c4:35de:f4c3\%8

        IPv6.DestinationAddress=<IPv6 address>
         Matches the specified filter against destination IPv6 addresses.
        e.g. IPv6.DestinationAddress=fe80::5038:3c4:35de:f4c3\%8

        CustomMac=<type(offset,value)>
         Matches the specified filter against the value at the specified
         offset starting with the MAC header.
         Note: This capture filter does not support ranges, lists or negation.
        e.g. CustomMac=UINT8(0x1,0x23)
        e.g. CustomMac=ASCIISTRING(3,test)
        e.g. CustomMac=UNICODESTRING(2,test)

        CustomIp=<type(offset,value)>
         Matches the specified filter against the value at the specified
         offset starting with the IP header.
         Note: This capture filter does not support ranges, lists or negation.
        e.g. CustomIp=UINT16(4,0x3201)
        e.g. CustomIp=UINT32(0x2,18932)

        CaptureMultiLayer=<yes|no>
         Enables multi-layer packet capture.
         Note: This capture filter does not support ranges, lists or negation.

        PacketTruncateBytes=<value>
         Captures only the the specified number of bytes of each packet.
         Note: This capture filter does not support ranges, lists or negation.
        e.g. PacketTruncateBytes=40

Note:
        Multiple filters may be used together. However the same filter may
        not be repeated.
        e.g. 'netsh trace start capture=yes Ethernet.Type=IPv4
              IPv4.Address=157.59.136.1'

        Filters need to be explicitly stated when required. If a filter is
        not specified, it is treated as "don't-care".
         e.g. 'netsh trace start capture=yes IPv4.SourceAddress=157.59.136.1'
              This will capture IPv4 packets only from 157.59.136.1, and it
```

```
              will also capture packets with non-IPv4 Ethernet Types, since
              the Ethernet.Type filter is not explicitly specified.
          e.g. 'netsh trace start capture=yes IPv4.SourceAddress=157.59.136.1
                Ethernet.Type=IPv4'
              This will capture IPv4 packets only from 157.59.136.1. Packets
              with other Ethernet Types will be discarded since an explicit
              filter has been specified.

        Capture filters support ranges, lists and negation (unless stated
        otherwise).
          e.g. Range: 'netsh trace start capture=yes Ethernet.Type=IPv4
                       Protocol=(4-10)'
              This will capture IPv4 packets with protocols between 4 and 10
              inclusive.
          e.g. List: 'netsh trace start capture=yes Ethernet.Type=(IPv4,IPv6)'
              This will capture only IPv4 and IPv6 packets.
          e.g. Negation: 'netsh trace start capture=yes Ethernet.Type=!IPv4'
              This will capture all non-IPv4 packets.

        Negation may be combined with lists in some cases.
          e.g. 'netsh trace start capture=yes Ethernet.Type=!(IPv4,IPv6)'
                This will capture all non-IPv4 and non-IPv6 packets.

        'NOT' can be used instead of '!' to indicate negation. This requires
        parentheses to be present around the values to be negated.
          e.g. 'netsh trace start capture=yes Ethernet.Type=NOT(IPv4)'
```

Scenarios

```
C:\>netsh trace show scenarios

Available scenarios (24):
---------------------------------------------------------------
AddressAcquisition       : Troubleshoot address acquisition related issues
AddressAcquisitionServer : Troubleshoot address acquisition server related issues
DirectAccess             : Troubleshoot DirectAccess related issues
DirectAccessServer       :
FileSharing              : Troubleshoot common file and printer sharing problems
ICS                      : Troubleshoot internet connection sharing related issues
InternetClient           : Troubleshoot web connectivity issues
```

```
InternetServer          : Troubleshoot server-side web connectivity issues
L2SEC                   : Troubleshoot layer 2 authentication related issues
LAN                     : Troubleshoot wired LAN related issues
Layer2                  : Troubleshoot layer 2 connectivity related issues
MBN                     : Troubleshoot mobile broadband related issues
NDIS                    : Troubleshoot network adapter related issues
NetConnection           : Troubleshoot network connection related issues
NetworkSnapshot         : Collect the current network state of the system
P2P-Grouping            : Troubleshoot Peer-to-Peer Grouping related issues
P2P-PNRP                : Troubleshoot Peer Name Resolution Protocol (PNRP) related issues
RemoteAssistance        : Troubleshoot Windows Remote Assistance related issues
Virtualization          :
VPNServer               : Troubleshoot VPN related issues
WCN                     : Troubleshoot Windows Connect Now related issues
WFP-IPsec               : Troubleshoot Windows Filtering Platform and IPsec related issues
WLAN                    : Troubleshoot wireless LAN related issues
XboxMultiplayer         : Troubleshoot Xbox Live Multiplayer connectivity-related issues
```

-end

Port Mirror / Monitor

Ubiquiti EdgeSwitch

```
monitor session 1 destination interface 0/10
monitor session 1 source interface 0/1
monitor session 1 source interface 0/2
monitor session 1 source interface 0/3
monitor session 1 source interface 0/4
monitor session 1 source interface 0/5
monitor session 1 source interface 0/6
monitor session 1 source interface 0/7
monitor session 1 source interface 0/8
monitor session 1 source interface 0/11
monitor session 1 source interface 0/12
monitor session 1 source interface 0/13
monitor session 1 source interface 0/14
monitor session 1 source interface 0/15
monitor session 1 source interface 0/16
monitor session 1 mode
```

tcpdump

Dealing with Ethernet headers and VLANs

```
# include ethernet header
tcpdump -n -e

# show only non-vlan traffic
tcpdump -n -e not vlan

# show only vlan traffic
tcpdump -n -e vlan

# show only vlan 1000 traffic
tcpdump -n -e '(vlan and (ether[14:2] & 0xfff == 1000))'

# show only vlan 1000 and 1001 traffic - needs testing
tcpdump -n -e '(vlan and (ether[14:2] & 0xfff == 1000 or ether[14:2] & 0xfff == 1001))'
```

#end

IPv6

pktmon - tcpdump for Windows

Here's a great series of articles by Rickard Nobel on using PKTMON.

Filter basics

```
# Layer-3
pktmon filter add "Some IP Address" -i 192.168.88.1
pktmon filter add "Some IP Subnet"  -i 192.168.88.0/24

pktmon filter add "All ICMP" -t ICMP
pktmon filter add "All TCP"  -t TCP
pktmon filter add "All UDP"  -t UDP

pktmon filter add "All DNS"  -p 53
pktmon filter add "DNS-UDP"  -t UDP -p 53
pktmon filter add "DNS-TCP"  -t TCP -p 53
pktmon filter add "LDAP"     -t TCP -p 389

pktmon filter add "Google DNS1" -i 8.8.8.8 -t UDP -p 53
pktmon filter add "Google DNS2" -i 8.8.4.4 -t UDP -p 53

pktmon filter add "All HTTP connections"  -p 80  -t TCP SYN RST FIN
pktmon filter add "All HTTPS connections" -p 443 -t TCP SYN RST FIN

pktmon filter add "IGMP"      -t 2
pktmon filter add "IPIP"      -t 4
pktmon filter add "GRE"       -t 47
pktmon filter add "IPSEC ESP" -t 50
pktmon filter add "OSPF"      -t 89
pktmon filter add "VRRP"      -t 112
```

```
# Layer-2
pktmon filter add "MAC Address" -m 00-11-22-33-44-55
pktmon filter add "MAC Address" -m 00:11:22:33:44:55

pktmon filter add "Cisco Discovery Protocol - CDP" -m 01:00:0C:CC:CC:CC

pktmon filter add "ARP"         -d ARP
pktmon filter add "Wake-on-Lan" -d 0x0842
pktmon filter add "LACP"        -d 0x8809
pktmon filter add "QinQ"        -d 0x88A8
pktmon filter add "LLDP"        -d 0x88cc

pktmon filter add "VLAN 101"    -v 101
```

Example: tcpdump like cli for Mikrotik MNDP packets

```
# clear filters
pktmon filter remove

# add MNDP filter
pktmon filter add Mikrotik_MNDP -t UDP -p 5678

# list interfaces available to capture on
pktmon list

# assign the capture interface number to a variable for use below
$captureInterface = x

# replace <interface> below with the interface number you wish to run the capture on
pktmon start -c -m rt -s 16 --comp $captureInterface

# same as above but removing the lines containing PktGroup if desired
pktmon start -c -m rt -s 16 --comp $captureInterface | Select-String -Pattern "PktGroup" -
NotMatch
```

Example: tcpdump like cli for LLDP packets

```
# clear filters
pktmon filter remove

# add MNDP filter
pktmon filter add "LLDP" -d 0x88cc

# list interfaces available to capture on
pktmon comp list

# assign the capture interface number to a variable for use below
$captureInterface = x

# replace <interface> below with the interface number you wish to run the capture on
pktmon start -c -m rt -s 16 --comp $captureInterface

# same as above but removing the lines containing PktGroup if desired
pktmon start -c -m rt -s 16 --comp $captureInterface | Select-String -Pattern "PktGroup" -
NotMatch
```

-end

Using a SOCKS proxy with OpenSSH

Setting up a SOCKS5 proxy using OpenSSH client command line

In order for this to work, the remote SSH server you're connecting to must have

```
ssh -D 1337 -q -C -N user@domain.com
```

Here's the options breakdown:

1. `-D 1337` opens a SOCKS proxy on local port 1337. You can specify any port number you would like.
2. `-c` enabled data compression in the tunnel, saving bandwidth
3. `-q` enables quiet mode
4. `-N` is to just forward ports, don't execute any remote commands

Once you've tested and verified that everything is working properly, you can add the -f option to fork the process to a background command.

```
# -f = fork to background
ssh -D 1337 -q -C -N -f user@domain.com
```

Using the SOCKS5

Setting up a SOCKS5 proxy using OpenSSH client config file

The configuration options `DynamicForward` is synonymous with the `-D` option and can be added to your `~/.ssh/config` file:

```
Host example.com
    User username
    DynamicForward 8080
```

-end

Monitoring

Monitoring

Smokeping

```
*** Database ***

# MSHARP 20250106
# - 20 pings every two minutes, one ping every 10 seconds
# - raw data (2 minutes) for 31 days (1 month)
#       - seconds in a month: 31 * 24 * 60 * 60 = 2678400
#       - steps per month: 2678400 / 120 = 22320
# - 30 minute data for 182 days (~6 months)
#       - number of steps in 30 minutes: 30 * 60 / 120 = 15
# - 1 hour data for 730 days (~2 years)

step     = 120
pings    = 12

# consfn mrhb steps total

# 31 days of raw data: 31 * 24 * 60 * 60 / 120
AVERAGE  0.5   1  22320

# 182 days of 30 minute data: 182 * 24 * 60 * 60 = 15724800 seconds / 120 = 131040 seconds /
15 = 8736
# 15 steps in a 30 minute window
AVERAGE  0.5  15   8736
    MIN  0.5  15   8736
    MAX  0.5  15   8736

# 730 days of 1 hour data: 730 * 24 * 60 * 60 = 63072000 seconds / 120 = 525600 seconds / 30 =
17520
# 30 steps in a 1 hour window
AVERAGE  0.5  30   17520
```

```
    MAX  0.5  30   17520
    MIN  0.5  30   17520
```

```
*** Probes ***

+ FPing

binary = /usr/bin/fping
hostinterval = 10
# These are set in the Database second:
# step = 120
# pings = 12
```

-end

HPE Aruba

HPE Aruba specific configuration snippets

Aruba 5400r

RFC-1918 subnet matching for route distribution

```
ip prefix-list "PRIVATE-IP-SUBNETS" seq 10 permit 10.0.0.0 255.0.0.0 ge 8 le 32
ip prefix-list "PRIVATE-IP-SUBNETS" seq 20 permit 172.16.0.0 255.240.0.0 ge 12 le 32
ip prefix-list "PRIVATE-IP-SUBNETS" seq 30 permit 192.168.0.0 255.255.0.0 ge 16 le 32

route-map "PRIVATE-IP-SUBNETS" permit seq 10
   match ip address prefix-list "PRIVATE-IP-SUBNETS"
   exit
```

Limit which Static subnets are allowed to redistribute

```
no ip prefix-list "OSPF-REDIST-STATIC-ALLOWED" seq 10
ip prefix-list "OSPF-REDIST-STATIC-ALLOWED" seq 10 permit 10.254.75.11 255.255.255.255

route-map "OSPF-REDIST-STATIC-ALLOWED" permit seq 10
   match ip address prefix-list "OSPF-REDIST-STATIC-ALLOWED"
   set metric-type external type-1
   exit

router ospf
   area backbone
   redistribute connected
   redistribute static route-map "OSPF-REDIST-STATIC-ALLOWED"
   enable
   exit
```

Relevant logs messages

Log messages to generate alerts on:

```
chassis: AM1: Insufficient power supplies to power

chassis: SM2: Internal power supply 2 removed.
chassis: AM1: Internal power supply 2 inserted.
chassis: SM2: Internal power supply 2 inserted.
chassis: AM1: Internal power supply 2 is OK.

chassis: AM1: Internal power supply 2 is faulted
chassis: AM1: Internal power supply 2 is faulted.

chassis: AM1: Internal power supply 2 main PoE power OK.
chassis: AM1: Internal power supply 2 main PoE power faulted.

chassis: AM1: Internal power supply 2 removed.

chassis: AM1: Internal power supply 2: Main inlet OK.
chassis: AM1: Internal power supply 2: Main inlet faulted.
```

-end