Netmapper User Guide

Release 2.1.1722

Distributed Analytics and Security Institute (DASI)
Mississippi State University

Oct 02, 2017

CONTENTS

CHAPTER

ONE

EXECUTIVE SUMMARY

The *Netmapper* tool from the Distributed Analytics and Security Institute (DASI) at Mississippi State University is a network mapping tool with the following features:

- Host discovery using the *Nmap* program and SNMP (v1/v2c/v3 support, SNMP functions are within *Netmapper*, not *Nmap*).
- Layer 3 topology discovery using SNMP and data retrieved via remote login.
- Layer 2 topology discovery using SNMP.
- Host role inference (such as Domain Controller, DNS, DHCP, WSUS, Yum server, etc.) for both Windows and Linux nodes from data retrieved via remote login. Supported OSes for remote login are Windows Server 2008/2012, Windows 7 and above, CentOS 6/7, and Ubuntu 12/14.
- Remote CLI login query for Cisco devices. Supported OSes are IOS, NX-OS, IOS-XR, IOS-XE, and ASA.
- Network Difference tool for comparing networks produced from different scans.
- Network Merge tool for merging networks from different scans to produce a larger network view.
- Encrypted credential storage.
- XML output format.
- Dynamic visualization (zoom/pan) of the network.
- Static visualization using PDF or Visio as the output file format, different options for controlling graph output.
- Graphical difference between a reference Visio file of the network and a scanned network.
- Command-line capability that supports automated regression testing of VMware Vapps.

The *Netmapper* tool runs under Windows 7, 64-bit. Refer to the *Language/Platform* section for other requirements.

CHAPTER

TWO

REQUIREMENTS

This section documents the computer requirements for the local host running *Netmapper* and for remote machines and devices that are to be queried by *Netmapper* for information extraction.

2.1 Host Requirements

Netmapper has been tested under Windows 7, 64-bit, SP1. While it is conceivable that it would run under other Windows OS versions without any problems, no other Windows OS versions have been tested.

Netmapper must be installed in a directory path that has no spaces. The default installation directory is *C:\DasiNetmapper*.

Netmapper uses *Nmap* to discover targets for query. *Netmapper* should be run with admin privilege so that *Nmap* discovery will be able to all of its available discovery mechanisms for best operation.

There are some special requirements for both the local host and for remote hosts in order for data retrieval via remote login to be successful.

2.1.1 Netmapper host requirements for Windows target query

Netmapper uses a combination of methods to query remote Windows hosts for information. Remote data retrieval from Windows hosts is performed by PowerShell scripts located in the *Netmapper* installation directory. The primary methods used are WMI, ADSI retrieval via LDAP queries, and Remote PowerShell Execution (only used for some role data retrieval). For this to succeed, the follow are requirements for the *Netmapper* host machine:

- PowerShell 4.0 or higher installed.
- The login credentials for the remote machine must belong to the administrator group or some queries for remote data will fail.
- *Netmapper* uses WMI and remote registry query to retrieve most information from Windows targets. However, *Netmapper* also uses powershell remote execution to retrieve some role configuration information from Windows targets when there is no WMI alternative. If the *Netmapper* host is in a different domain from the windows target, then powershell execution will fail unless the target is in the trusted hosts of the Netmapper host. Two convenience scripts are in the *Netmapper* installation directory: `get_trusted_hosts.ps1` and `set_trusted_hosts.ps1`. The first script displays information that lists the trusted hosts (look for the *TrustedHosts* key, will typically be empty). The second script will set the trusted hosts from a file that has one host per line. To display or modify the trusted hosts, the Windows Remote Management (WinRM) service must be started on the *Netmapper* host. The service start mode should be set to automatic (configuring/starting the *winrm* service is a *Netmapper* installation option). To set mode to automatic, execute the following command as an administrator from a command prompt:

```
sc config winrm start= auto
```

To start the service, execute the following command as an administrator from a command prompt:

```
net start winrm
```

- For retrieval of WSUS server role configuration information, the *Netmapper* machine must have the WSUS console and the Microsoft Report Viewer 2008 Redistributable installed. Installation of the Microsoft Report Viewer 2008 Redistributable is a *Netmapper* installation option.

2.1.2 Netmapper host requirements for Linux target query

There are no special requirements on the *Netmapper* host for querying Linux hosts.

2.1.3 Netmapper host requirements for Cisco target query

There are no special requirements on the *Netmapper* host for querying Cisco devices.

2.2 Remote Target Requirements

2.2.1 Windows remote hosts

This section lists the requirements for remote Windows hosts to be successfully queried by *Netmapper*.

- Remote hosts must allow WMI and remote registry queries at a minimum for all default information to be retrieved. In general, servers like Windows 2008 and above already have this enabled, while workstations do not. Please refer to online references for enabling this capability as it varies by OS type. In general, the command:

```
winrm quickconfig
```

can be used on a machine to enable WMI queries to it.

- The network on the remote host for incoming WMI queries must be classified as a 'work' or 'home' network, not a 'public' network. This classification can be viewed in the *Network and Sharing Center* accessible from the *Control Panel*.

To test if the remote machine allows WMI connections, open a PowerShell window on the local machine and execute the command:

```
Test-WSman hostname
```

where *hostname* is the name of the remote machine. If successful, key/values pairs are returned for keys of *skid*, *ProtocolVersion*, *ProductVendor*, and *ProductVersion*.

- Some role data from servers is retrieved by remote PowerShell execution. On Windows Server 2012, this is already enabled. On Windows Server 2008, it is not enabled. To enable remote PowerShell execution, open a PowerShell window as an administrator and execute:

```
Enable-PsRemoting -Force
```

If *Test-WSman hostname* succeeds, but *Netmapper* login fails, then it may be necessary to modify the firewall rules on the remote target machine by executing the following on the remote target machine:

```
netsh advfirewall firewall set rule group="Windows Management Instrumentation (WMI)"␣
↪new enable=yes
```

If you are querying Windows machines that are part of a domain, then Windows credentials should have the domain name included in the user name (i.e. domain\user).

If querying Windows machines that are part of a domain, and the *Netmapper* machine is not part of the domain, then retrieval of role configuration information for some roles will fail (domain controller configuration, Exchange server configuration).

2.2.2 Linux remote hosts

The remote machine must allow SSH login via user/password authentication (Ubuntu and Kali nodes seem to have this disabled by default).

The login credentials for the remote machine must have admin read privilege or some queries for remote data will fail.

2.2.3 Cisco devices

The device must allow SSH login via user/password authentication.

The login credential privilege level will determine the amount of information returned. The output of various *show* commands are parsed for interface and routing configuration information. the *show running-config* command is also executed and its output is saved in an archinve object within the XML. The output of the *show running-config* command is not parsed as this requires the highest privilege level in order to successfully execute, so the information returned in the non-archive XML structures does not depend on this command's successful execution.

CHAPTER

THREE

INSTALLATION

The *Netmapper* self-installer allows the user to choose an installation directory. *Netmapper* must be installed in a directory path that has no spaces in its path. The default installation directory is *C:\DasiNetmapper*.

During installation, *Netmapper* installs Microsoft Visual C++ 2008 and 2010 redistributables.

In addition to its own binary, *Netmapper* installs the *Nmap* tool under its own installation directory and uses this *Nmap* version for host discovery and port scanning. *Netmapper* also installs the *Graphviz* tools under its installation directory; this is needed for the network visualization options in *Netmapper*.

Other installation choices are:

- Nmap *performance enhancements* - these are registry modifications to improve *Nmap* performance. This is an optional installation choice.
- Nmap Winpcap - a version of *Winpcap* for *Nmap* - this will not be installed if a later version of *Winpcap* is already installed. This is an optional installation choice, but *Nmap* will fail if *Winpcap* is not already installed on the system.
- Add *Graphviz* binaries to path - this will ad the *Graphviz* binaries under the *Nmap* installation directory to the system search path. This is an optional installation choice, but *Netmapper* visualization will fail if the *Graphviz* binaries are not on the system search path.
- *WinRm* service started and configured as auto - this is if you wish to modify the trusted hosts list (needed for remote powershell execution to Windows hosts not in the same domain as the *Netmapper* host)
- *Win2008 Report Viewer* redistributable installation - this is needed for retrieval of WSUS configuration information to succeed.
- Start Menu shortcuts - these can be optionally installed; a folder named *DasiNetmapper* is created in the start menu. The *Netmapper* executable is available from the *DasiNetmapper* choice within this folder.

CHAPTER

FOUR

USAGE OVERVIEW

The figure below shows the *Netmapper* main screen. The *Mapping* tab is the primary tab for starting *Netmapper* actions. The *Discover/Gather* button is a do-all button that kicks off the following sequence when pressed:

- Discovery/TCP port scan - The target IPs/networks/hostnames specified on the first type-in line ('Enter target IPs...') are passed to *Nmap* which begins a discovery and TCP port scanning for these target IPs/networks/hostnames.
- UDP port scan - Any devices discovered by *Nmap* in step one are then port scanned for UDP port status by a second *Nmap* invocation.
- SNMP data retrieval - All discovered devices are then queried by SNMP for its system description to determine if a device responds to an SNMP query. Any devices that respond to the SNMP system description are then further SNMP queried for network information such as interfaces, routing tables, bridge tables, VLANs, etc. During the SNMP information gathering phase, all devices are probed with an SNMP v1/v2c credential with community string equal to *public* if no other SNMP credential is found for a node (this default query can be disabled via a configuration option).
- Rlogin data retrieval - Any devices that match login credentials loaded from the *Credentials* tab are queried by remote login. Remote login has two phases: a) Default data retrieval and b) role data retrieval. Default data retrieval is performed first, and returns data that is common to all hosts such as running services, disk configuration, firewall status/rules, etc. The default data is examined, and roles are assigned to a host such as DNS server, DHCP server, Domain Controller etc. Based on the role determination, a second query may made to a host to retrieve role-specific configuration data. Devices that match login credentials are also queried to determine if they respond to a HyperVisor API call (vSphere API currently supported). If a device responds, then role-specific data is retrieved from that node (the HyperVisor API call can be disabled by via configuration option).
- Topology extraction - based on data gathered by SNMP and Remote Login, L2 (layer 2) and L3 (layer 3) topology links are created for nodes.
- Custom tag - this allows user-written Python code to be dynamically imported and executed after the Topology Extraction step. The intended use for this is to allow custom XML attributes to be added to an object.

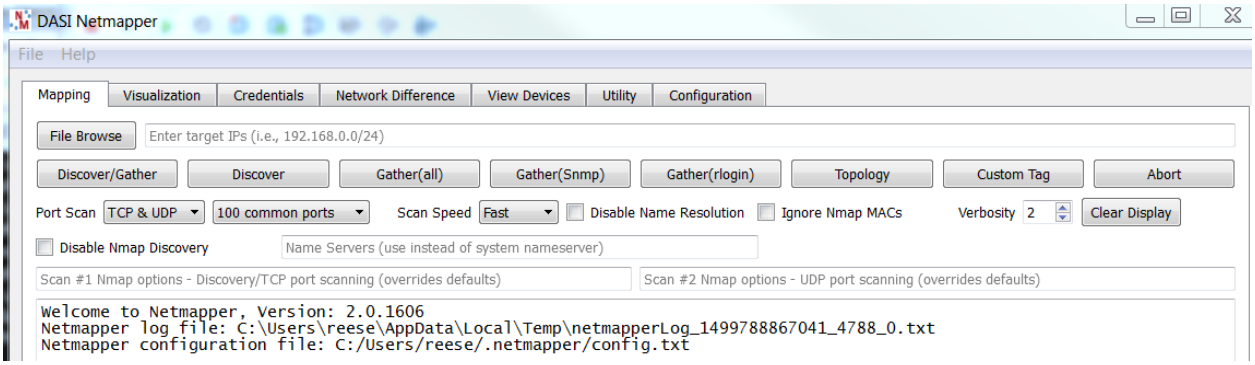


Fig. 4.1: *DASI Netmapper* main screen.

Once the Discovery/Information Gathering/Topology extraction is complete, the data can be stored to an external file in XML format using the *File|Save Network* menu choice.

4.1 General Usage

4.1.1 Startup actions (Initialization/Log files)

On startup, *Netmapper* loads an initialization file named *config.txt* from the *.netmapper* directory in the user's home directory (the *–homedir <path>* command line option can be used to change the directory for the location of the *.netmapper* folder). Both the *.netmapper* directory and *config.txt* file are created if they do not exist. The configuration file stores both dynamic and static configuration options. Static configuration options are editable by the user and appear in the file before the dynamic options. The dynamic options are updated each time *Netmapper* is closed and contain the current state of GUI options so that the same GUI options appear when *Netmapper* is reopened.

A log file is created in the users home *appdata\local\temp* directory. If you cannot see this directory then, from an Explorer folder window, click on the *Organize* menu choice, then select *Folder and search options*. On the *Folder Options* window, click on the *View* tab, and ensure that *Show hidden files, folders, and drives* is enabled (see the figures below).

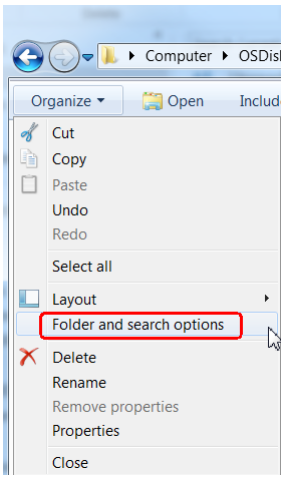


Fig. 4.2: Right-click folder options menu.

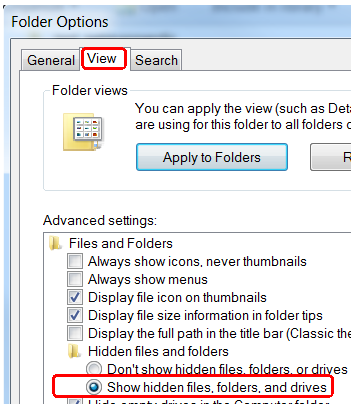


Fig. 4.3: Enabling the *Show hidden files, folders, and drives* option.

All log messages are marked with one of three levels: INFO, WARNING, and ERROR.

4.1.2 Controlling verbosity

The verbosity control is used for limiting the number of log messages that appear in the *Mapping* tab output window (all log messages always appear in the log file regardless of the verbosity setting).

Fig. 4.4: Screenshot of the verbosity selection box.

The verbosity control has three levels:

- 0 – only messages for gross actions are reported (Discovery, SNMP, Rlogin, etc.).
- 1 – messages for steps within a phase such as SNMP are printed, but not for each host/device.
- 2 – all log messages are echoed to the *Mapping* tab output window.

It is recommended that new users set verbosity to the maximum level so that there is ample feedback on *Netmapper* actions.

4.1.3 File menu choices

The File Menu along the top ribbon has the following choices:

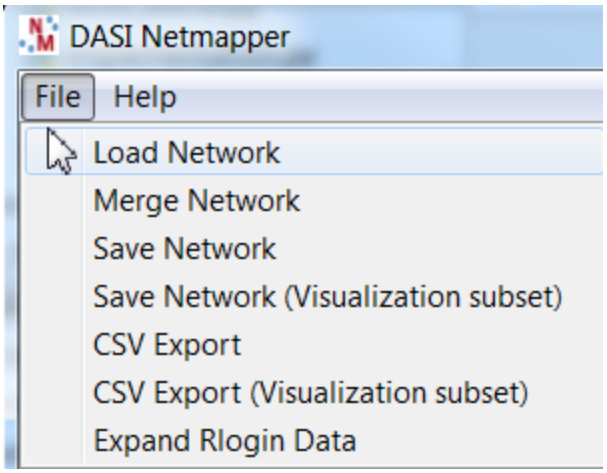


Fig. 4.5: Screenshot of *File* menu choices.

- *Load Network* – loads a previous saved XML network file into memory. Any network objects currently in memory are lost.
- *Merge Network* – loads a previous saved XML network file into memory but merges the new network objects with objects currently in memory. Duplicate nodes are merged. This allows scans of different networks to be combined into one file.
- *Save Network* – saves the current network objects in memory to an XML file.
- *Save Network (Visualization subset)* - saves the current network objects listed in the *Visualization* tab to an XML file. The *Visualization* tab allows an object subset to be selected based on configurable filters, such as device type (router, switch, etc.), by network connection, etc.
- *CSV Export* - exports the current network objects in memory as a CSV file. Only a small subset of the node data is exported due to the limits of the CSV format (this functions more as an executive summary of the node data). The CSV export code is in *INSTALLDIR/plugins/csvexport/implementation.py* and can be modified by the user.
- *CSV Export (Visualization subset)* - exports the current network objects listed in the *Visualization* tab to a CSV file.
- *Expand Rlogin Data* – Some retrieved data from remote login are stored as files in compressed ZIP archives within the XML. This expands the compressed data for all nodes in memory onto disk in the same folder that the file was loaded from, under a folder named *RloginNodeConfigData*. A sub-folder named by the archive timestamp is created, and then all nodes that have compressed data that share the same timestamp are expanded here. Each node is identified by its hostname plus a portion of its UUID, with sub-folders for each role. Each role sub-folder has one or more files/directories containing configuration data for that role (see the figure below).

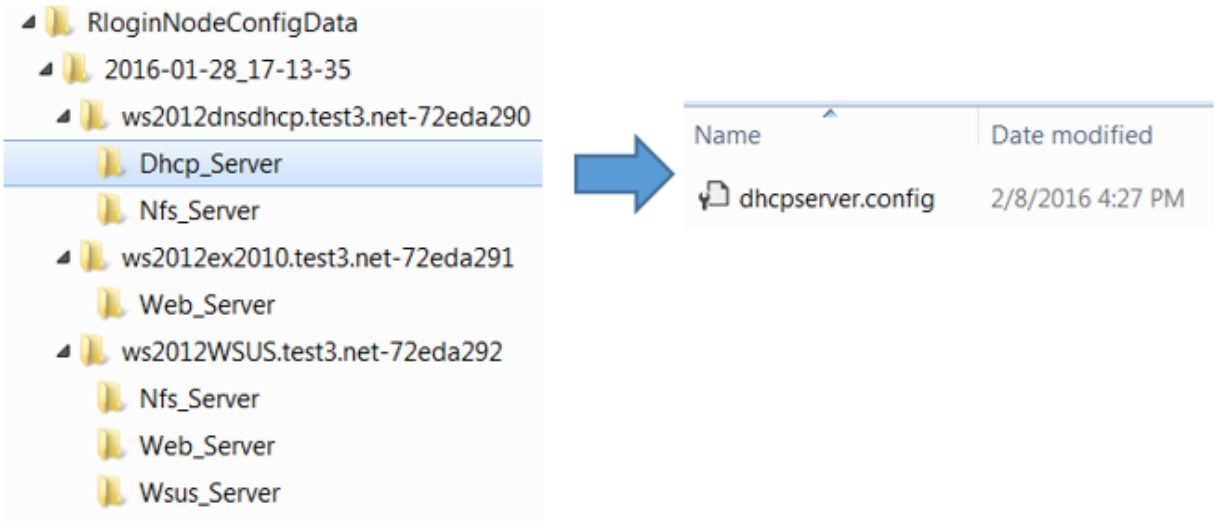


Fig. 4.6: Rlogin data folder expanded.

4.2 *Mapping* Tab Details

The *Mapping* tab is the primary tab in *Netmapper* for starting network discovery/information gathering and display of log messages.

4.2.1 *File Browse* button, *Enter target* text edit

The *Enter target* text edit is for entry of target hosts or networks for discovery and information gathering.

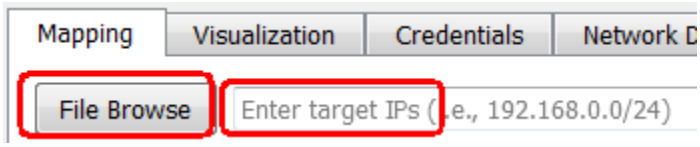


Fig. 4.7: Screenshot of the *File Browse* button and the *Enter target* text box.

Entries must be white-space delimited. Hosts can be specified either by name (either FQDN or simple hostname) or as an IP address (IPv4 or IPv6). Networks are specified in CIDR format (i.e., 192.168.1.0/24). The *File Browse* button is used to browse to a text file containing a list of targets, one per line, with the last line being blank.

- If IPv6 addresses are passed, then the -6 flag is passed automatically to Nmap (required by Nmap).
- If a mixture of IPv4, IPv6 addresses are passed, then the targets are parsed into two lists (IPv4 and IPv6), and Nmap scans are called once for each list. Nmap allows ranges to be specified for IPv4 addresses, and interfaces for link-local IPv6 addresses, and these are allowed.
- If hostnames are passed, these are resolved using system DNS before Nmap is called so that it can be determined if they are IPv4 or IPv6 addresses. If a hostname resolves into both IPv4 and IPv6 addresses, then Nmap is called for each address.

4.2.2 Action buttons

The action buttons initiate discovery/information/topology actions in *Netmapper*.

Fig. 4.8: Screenshot of the action buttons bar.

When an action button is used, the other buttons, except for the *Abort* button, are disabled until the current action completes.

- The *Discover/Gather* button is the do-everything button that performs discovery/port scanning, information gathering, and topology inference for the targets specified in the *Enter target* text edit. The internal list of devices is cleared on *Discover/Gather* button activation.
- The *Discover* button only performs discovery/port scanning for the targets specified in the *Enter target* text edit. The internal list of devices is cleared on *Discover* button activation.
- The *Gather(all)* button performs SNMP and Remote login information gathering for network objects currently in memory (either from a previous *Discover* action or by loading an XML file that was previously generated by *Netmapper*.
- The *Gather(Snmp)* button performs only SNMP information gathering for network objects currently in memory.
- The *Gather(rlogin)* button performs only Remote login information gathering for network objects currently in memory.
- The *Topology* button regenerates L3/L2 topology links for network objects currently in memory.
- The *Custom Tag* button dynamically imports and executes user-written Python code. The intended use of this functionality is for adding custom XML attributes to an object. This capability is disabled by default via a configuration option.
- The *Abort* button will abort any current *Netmapper* actions. It may take some time for the abort to complete as *Netmapper* waits for any currently executing threads or processes to complete before halting the specified action on remaining network objects.

4.2.3 Controlling *Nmap* discovery/port scanning

By default, *Nmap* is run twice during the discovery/port scanning phase:

- Scan #1 performs device discovery on the targets specified in the *Enter target* text edit. Any devices discovered are then scanned for their TCP port status.
- Scan #2 performs UDP port scanning on the devices discovered by Scan #1.

The number of ports scanned (TCP & UDP) is controlled by the second combo box in the *Port Scan* control.

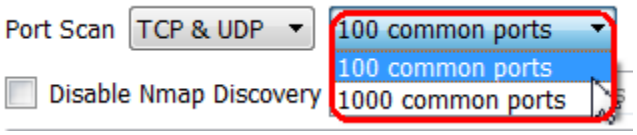


Fig. 4.9: Screenshot of the port scan number combo box.

The first combo box in the *Port Scan* control is used to enable TCP & UDP port scanning, enable TCP port scan only, or disable port scanning completely.

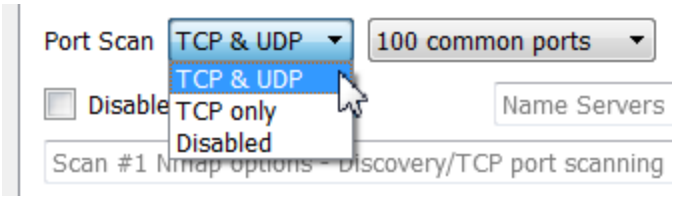


Fig. 4.10: Screenshot of the port scan selection.

The *Scan Speed* combo box sets the speed at which port scans are generated, with *Normal*, *Fast*, *Slow* corresponding to *Nmap* speed templates of `-T3`, `-T4`, and `-T2`.

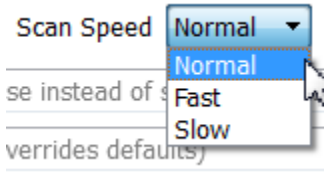


Fig. 4.11: Screenshot of the scan speed selection.

The *Name Servers* typein field can be used to specify alternate name servers instead of the system name server (this field is passed to Nmap via the `--dns-servers` command line option).

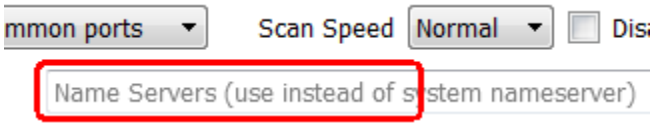


Fig. 4.12: Screenshot of the nameserver typein field.

The *Disable Name Resolution* checkbox disables *Nmap* name resolution (useful for reducing discovery time).

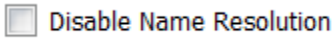


Fig. 4.13: Screenshot of the *Disable Name Resolution* checkbox.

The *Ignore* Nmap *MACs* checkbox disables use of MAC addresses returned by *Nmap* (useful when querying over a VPN – in this case, the MACs returned by *Nmap* are incorrect).

The two text edit controls under the *Port Scan* control can be used to completely customize the *Nmap* actions for Scan #1 and Scan #2. Do not specify any input or output options or the `-d` option (increases debug information including more port information) as *Netmapper* automatically adds these options. These text edit control values are stored as dynamic parameters in the *config.txt* file, so if a complex set of options is entered, these are recalled the next time that the *Netmapper* GUI is started.

Fig. 4.14: Screenshot of the *Nmap* options text boxes.

The *Disable Nmap Discovery* checkbox disables the Nmap discovery phase via the `-Pn` option. This means that Nmap assumes that all hosts are available, and procedes directly to the port scanning phase. If port scanning is disabled, then no packets are sent to any hosts. Caution - use of this with a network specification means that all addresses in the network are expanded out to objects in the XML.

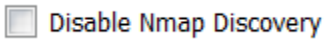


Fig. 4.15: Screenshot of the *Disable Nmap Discovery* checkbox.

The *Disable Nmap OS Scan* checkbox disables the Nmap OS scan during the TCP port scan phase (the Nmap OS scan populates the OSGuess field in the XML). By default, Netmapper includes the Nmap OS Scan option (-O) during the TCP port scan phase. Disabling this slightly improves scan speed and reduces traffic.

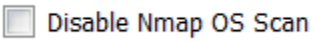


Fig. 4.16: Screenshot of the *Disable Nmap OS Scan* checkbox.

4.3 *Configuration* Tab Details

The *Configuration* tab provides easy access to different configuration options.

4.3.1 Loading/Saving config files

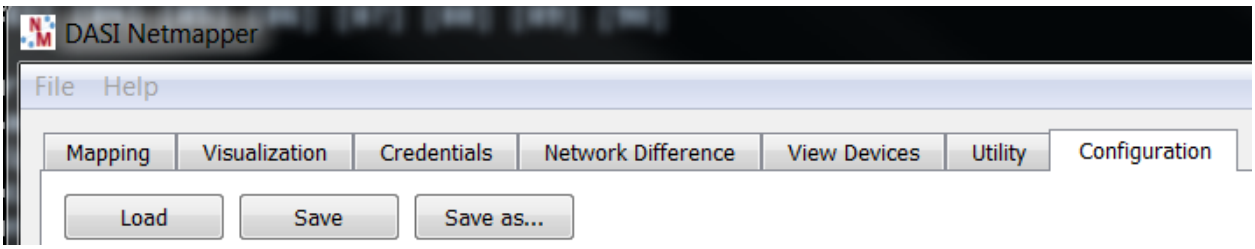


Fig. 4.17: Screenshot of the *Configuration* tab.

Netmapper loads a configuration text file named *config.txt* from the *.netmapper* directory in the user's home directory. The *Load* button will load a configuration file from a user-specified location. This location is saved in a file named *lastconfig.txt* in the *.netmapper* directory in the user's home directory. This same configuration file will then be loaded the next time that *Netmapper* is started; this allows the user to change what configuration file is loaded on startup.

The *Save* button saves all configuration information to the last opened configuration file. The *Save As* button saves configuration information to a user-specified file.

4.3.2 Seed-based discovery

Under the *Configuration* tab is a group labeled *Seed Based Discovery*.

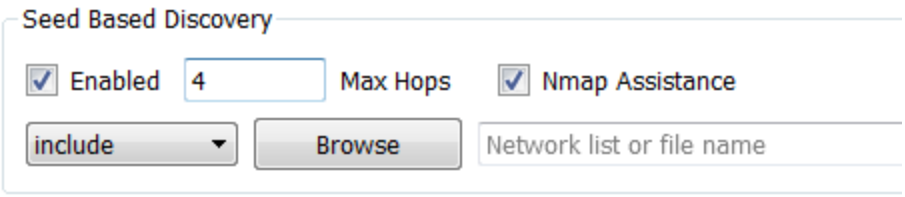


Fig. 4.18: Screenshot of the *Seed Based Discovery* section.

This allows the user to specify a single router as the discovery starting point. Entries in the ARP table from this router are then used as targets in the next discovery iteration. The discovery iteration loop stops whenever no new devices are discovered or the maximum hops in the L3 graph becomes equal or greater to the *Max Hops* parameter.

After seeded discovery is finished, the *Nmap Assistance* checkbox allows an optional additional discovery round by doing an *Nmap* scan of all networks discovered to this point. This is intended to find any devices are in these networks but not in the ARP tables used during seeded discovery. If a network is larger than /24, then a peephole of /24 is used around an existing discovered device within that network instead of scanning all of the larger network. This is intended to limit *Nmap* execution time during this step.

The typein field next to the *Browse* button can be used to specify either a list of networks or a file name that contains one network per line. These networks are used limit the devices kept during seeded discovery. If the pulldown menu is set to *include*, then only devices in the specified networks are kept. If the pulldown menu is set to *exclude*, then discovered devices in the specified networks are discarded.

4.3.3 Query options

The *Query* group has options that control Rlogin/SNMP data retrieval.

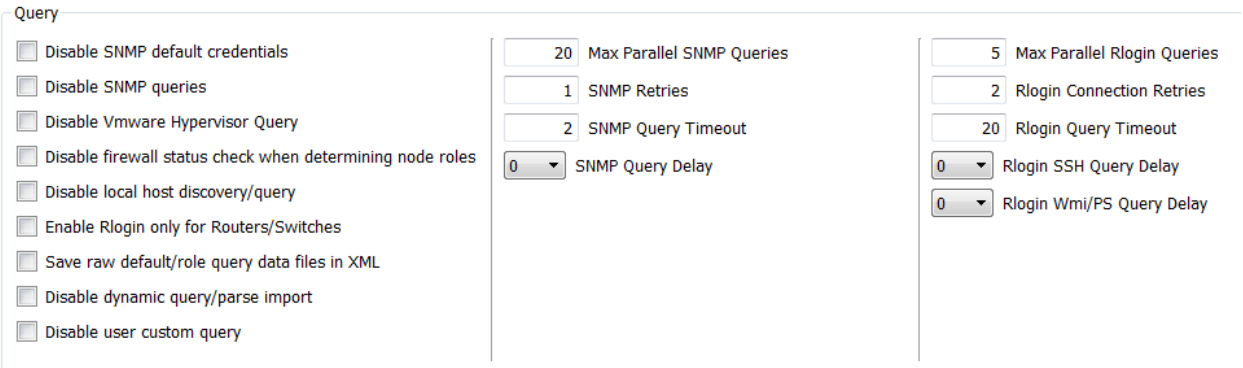


Fig. 4.19: Screenshot of the *Query* group options.

These options are:

- *Disable SNMP default credentials* – disable SNMP default query of discovered device using V2/Community string public.
- *Disable SNMP queries* – disable all SNMP queries
- *Disable Vmware Hypervisor Query* – if a device matches a login credential, the first interrogation determines if the device responds a Vmware Hyervisor query. Disabling this default query can reduce query time.

- *Disable firewall status check when determining node roles* – when determining a role (i.e, NFS server) one of the qualifiers is if the firewall allows communication over the port(s) associated with that role. This option disables the firewall check.
- *Disable local host discovery/query* – the local host (netmapper host) will not be included in discovery or query (if discovered by Nmap, it will be deleted from the device list, preventing further queries to it).
- *Enable Rlogin only for Routers/Switches* – this enables remote login to be attempted only if a device has been determined to be a router or switch. For this to work, SNMP query has to be enabled and the target device responsive to SNMP query. This is intended to reduce *Netmapper*-generated network traffic.
- *Save raw default/role query data files in XML* – this saves the default query data and role query data files as archive file objects, under rolename *hasRawQueryDataFiles* (XML tag name) and directory name *Raw_Query_Data_Files*, with filenames of *default_query_data.txt* and *role_query_data.txt* (role query file will not exist if node was not queried for role information). These are the raw data files that are parsed to generate XML data. These are useful if one is curious as to what information is being queried and how it is being query (also useful for debugging parsing problems). Using this option will significantly increase XML file size.
- *Disable dynamic query/parse import* – this disables importing of dynamic/query parse code from the *<installdir>/plugins/queryparse* directory.
- *Disable user custom query* – this disables execution of user custom query commands.
- *Max Parallel SNMP Queries*, *Max Parallel Rlogin Queries* – limit the maximum number of parallel SNMP/Rlogin queries to these values. This allows user control of network bandwidth used by *Netmapper* queries.
- *SNMP/Rlogin SSH/Rlogin Wmi/PS Query Delay* – time in seconds between individual queries during an SNMP, Rlogin SSH, Rlogin Wmi/PS interrogation. This option allows user control of network bandwidth used by *Netmapper* queries.
- *SNMP/Rlogin Connection Retries* – number of retries for failed SNMP/Rlogin connection attempt.
- *SNMP/Rlogin Query Timeout* – timeout in seconds for a non-responsive SNMP/Rlogin query.

4.3.4 Miscellaneous configuration options

The *Misc* group contains miscellaneous configuration options.

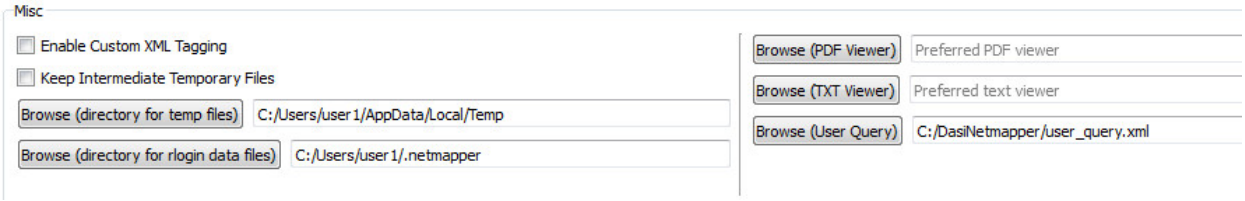


Fig. 4.20: Screenshot of the *Misc* group options.

- *Enable Custom XML Tagging* – enable dynamic import and execution of user Python code after the update topology step of a discover/gather action. Seed the documentation section Custom XML Tagging for more details.
- *Keep Intermediate Temporary Files* – keep all *Netmapper* temporary files instead of deleting them, useful for debugging.
- *Browse (directory for temp files)* – Use this browse button to change the directory used for temporary files.
- *Browse (directory for rlogin data files)* – Use this browse button to change the directory used for storing data files retrieved during remote login. The files are stored in a directory tree named *RLoginNodeConfigData*.

- *Browse (PDF Viewer)* – This sets the viewer used for displaying static PDF graphs. You are prompted to browse to a PDF viewer the first time a static PDF graph is generated if a PDF viewer has not been set yet. Use this button to change the PDF viewer at any time.
- *Browse (TXT Viewer)* – This sets the viewer used for displaying text files. You are prompted to browse to a text file viewer the first time an archive file is viewed in the View Devices tab using the View File button. Use this button to change the text file viewer at any time.
- *Browse (User Query)* – This sets the XML file path for user custom query.

4.4 *Visualization* Tab Details

The *Visualization* tab is used to display Layer 3, Layer 2, and Virtual network plots.

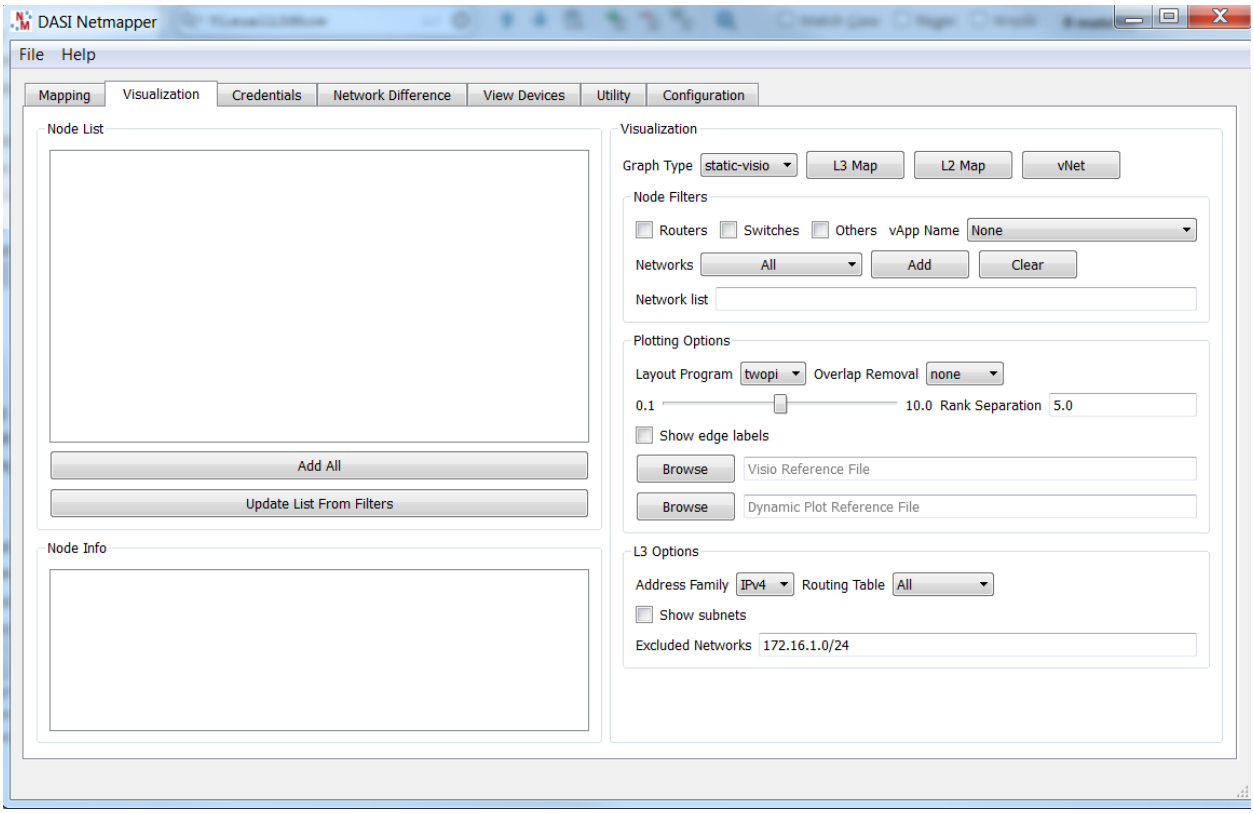


Fig. 4.21: Screenshot of the *Visualization* tab.

4.4.1 *Update List From Filters* button

Any *netobjects* current in memory will be displayed in the *Node List* when the *Visualization* tab is selected. The *Update List From Filters* button is used to update the *Node List* based on the *Node Filters* controls. As an example, the figure below shows updating the *Node List* with only nodes marked as routers.

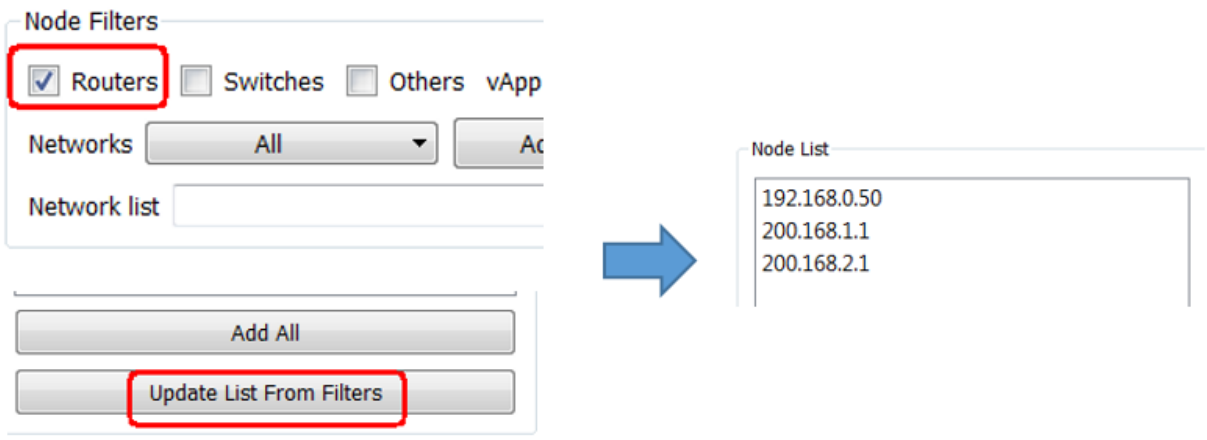


Fig. 4.22: Screenshot of the *Node Filters* options.

The *Networks* combo box lists the known networks of the nodes currently in memory. Selecting a network, then clicking on *Add* will add the network to the *Network list* control. This can be repeated multiple times to add more than one network to the *Network list* control.

Note: There will only be networks listed if a device like a router has been queried either via SNMP or remote login and routing table information retrieved.

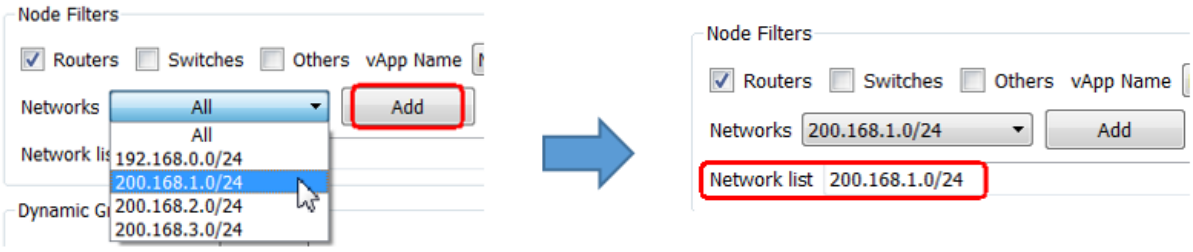


Fig. 4.23: Screenshot of using the *Networks* control options.

Clicking on *Update List From Filters* updates the *Node List* with all nodes from the *Network list* networks as shown in this figure.

Fig. 4.24: Updating the *Node List*.

If *Netmapper* has retrieved information via remote login from a Vmware Vcenter server, then it may have found VirtualApp managed objects (a vApp contains a collection of virtual machines). These vApps will be available in the *vApp Name* combo box. Selecting a vApp from the combo box, then clicking on the *Update List From Filters* button will put all of the nodes in the vApp into the *Node List* control. These actions are shown in the next figures.

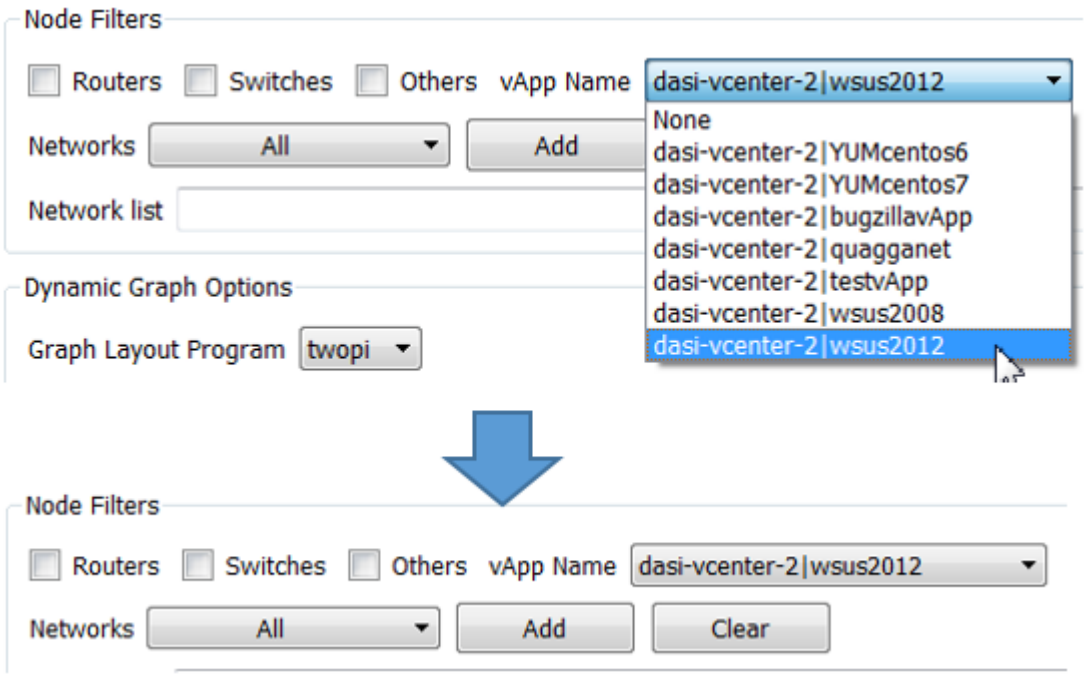


Fig. 4.25: Screenshot of selecting a vApp.

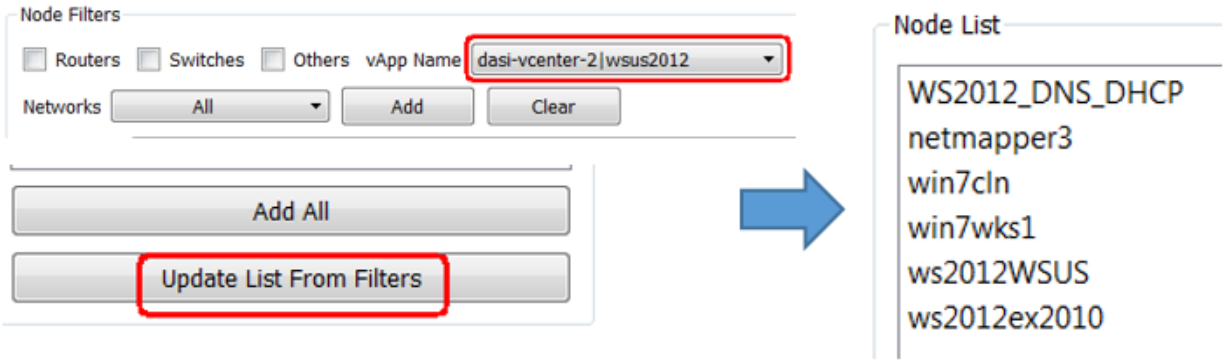


Fig. 4.26: Screenshot of updating a vApp.

4.4.2 *Add All* button

The *Add All* button adds all nodes currently in memory to the *Node List*.

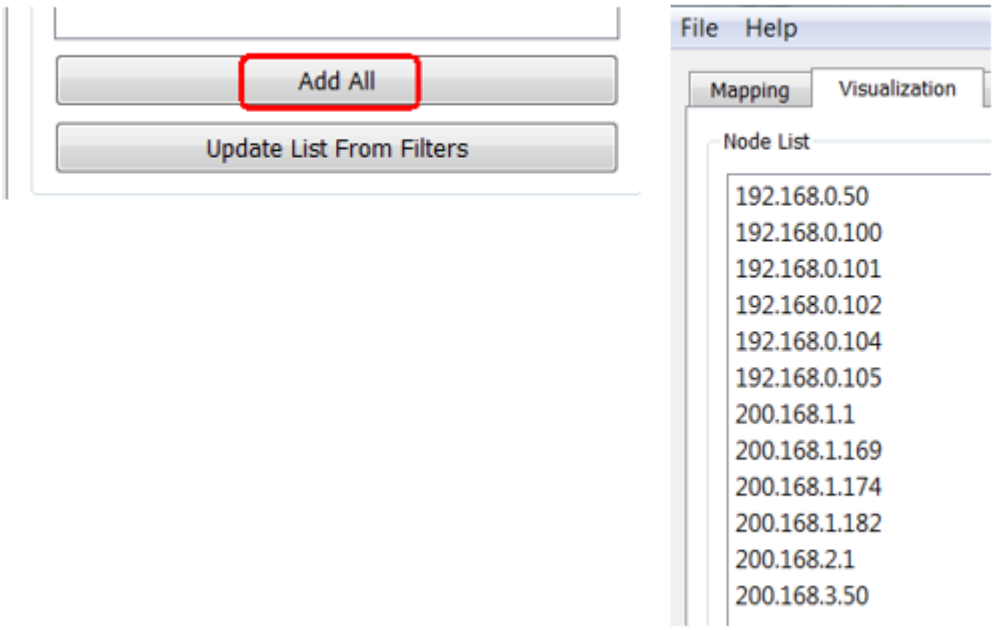


Fig. 4.27: Using the *Add All* button.

4.4.3 Dynamic Graphs

L3/L2/vNet plotting

Once nodes have been added to the *Node List* control, a layer 3 (L3), layer 2 (L2), or virtual network (vNet) plot can be created. An L3 plot can be created only if nodes have L3 links, which are created from retrieved routing table

information. An L2 plot can be created only if nodes have L2 links, which are created from retrieved bridge table information. A vNet plot can be created only if nodes are connected to virtual networks (e.g. virtual machines nodes were retrieved by API query from an ESXi or vCenter server).

An L3/L2/vNet plot can be displayed either as a dynamic pan/zoom window, static PDF, or static Visio drawing via the *Graph Type* combo box.

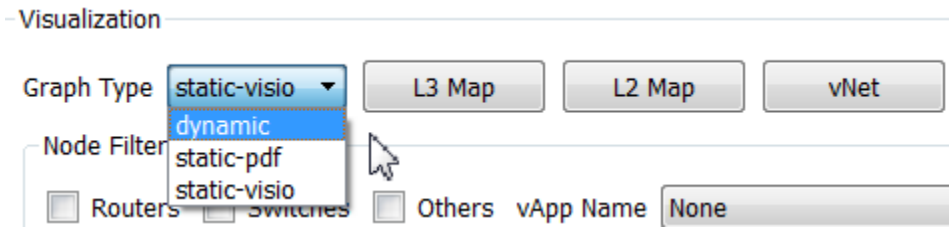


Fig. 4.28: Screenshot of the *Graph Type* box.

For example, selecting the dynamic option in the *Graph Type* control and clicking the L3 Map button produces an L3 dynamic plot of the nodes in the *Node List* control.

The type of L3 links plotted depends on the options selected in the *L3 Options* section.

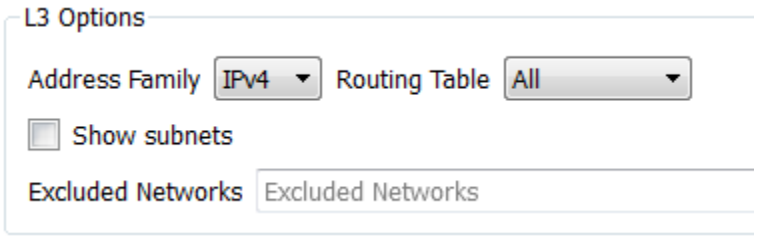


Fig. 4.29: Screenshot of the *L3 Options*.

The *Address Family* combo box allows plotting of either IPv4 links or IPv6 links.

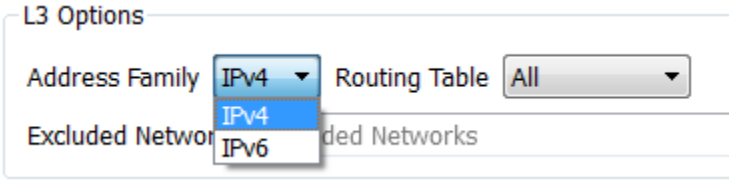


Fig. 4.30: Screenshot of the *Address Family* box.

The *Routing Table* combo box allows plotting of links that belong to a particular VRF instance.

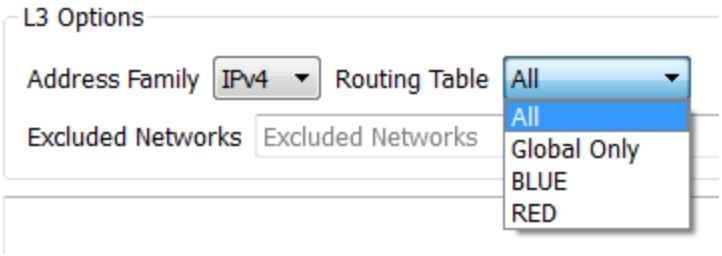


Fig. 4.31: Screenshot of the *Routing Table* box.

The *Excluded Networks* type-in field allows the user to specify networks whose connections should be excluded from the graph. This is useful to remove connections corresponding to a global management network.

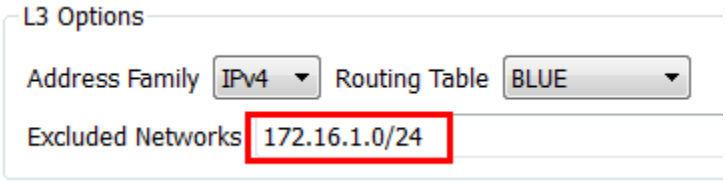


Fig. 4.32: Screenshot of the *Excluded Networks* field.

The *Show subnets* checkbox will show subnets as plot objects in the graph, directly exposing subnet connections.

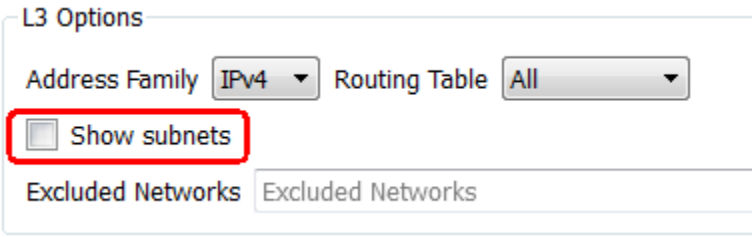


Fig. 4.33: Screenshot of the *Show subnets* field.

For L3 graphs, edge labels can be turned on via the *Show edge labels* in the *Plotting Options* section. Edge labels display the last two octets of the IP address destination. Currently, edge labels do not exist for other graph types but this may change in in the future.

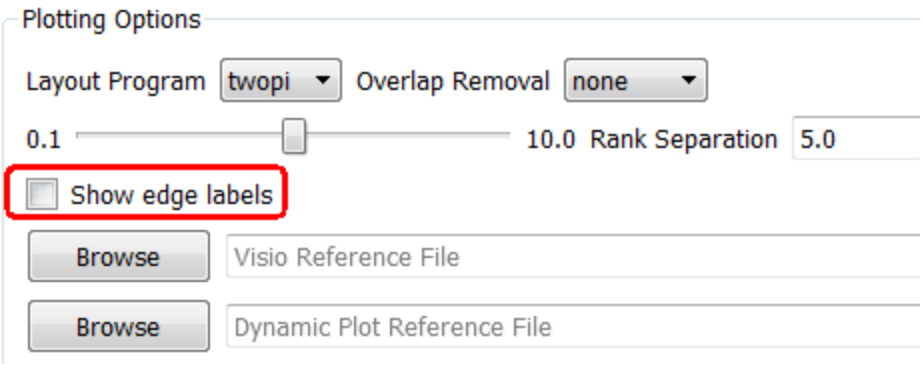


Fig. 4.34: Screenshot of the *Show edge labels* field.

An example of a dynamic L3 graph is shown in the following figure. The node size and zoom level are controlled by the controls along the bottom of the window. This graph does not have subnets or edge labels turned on.

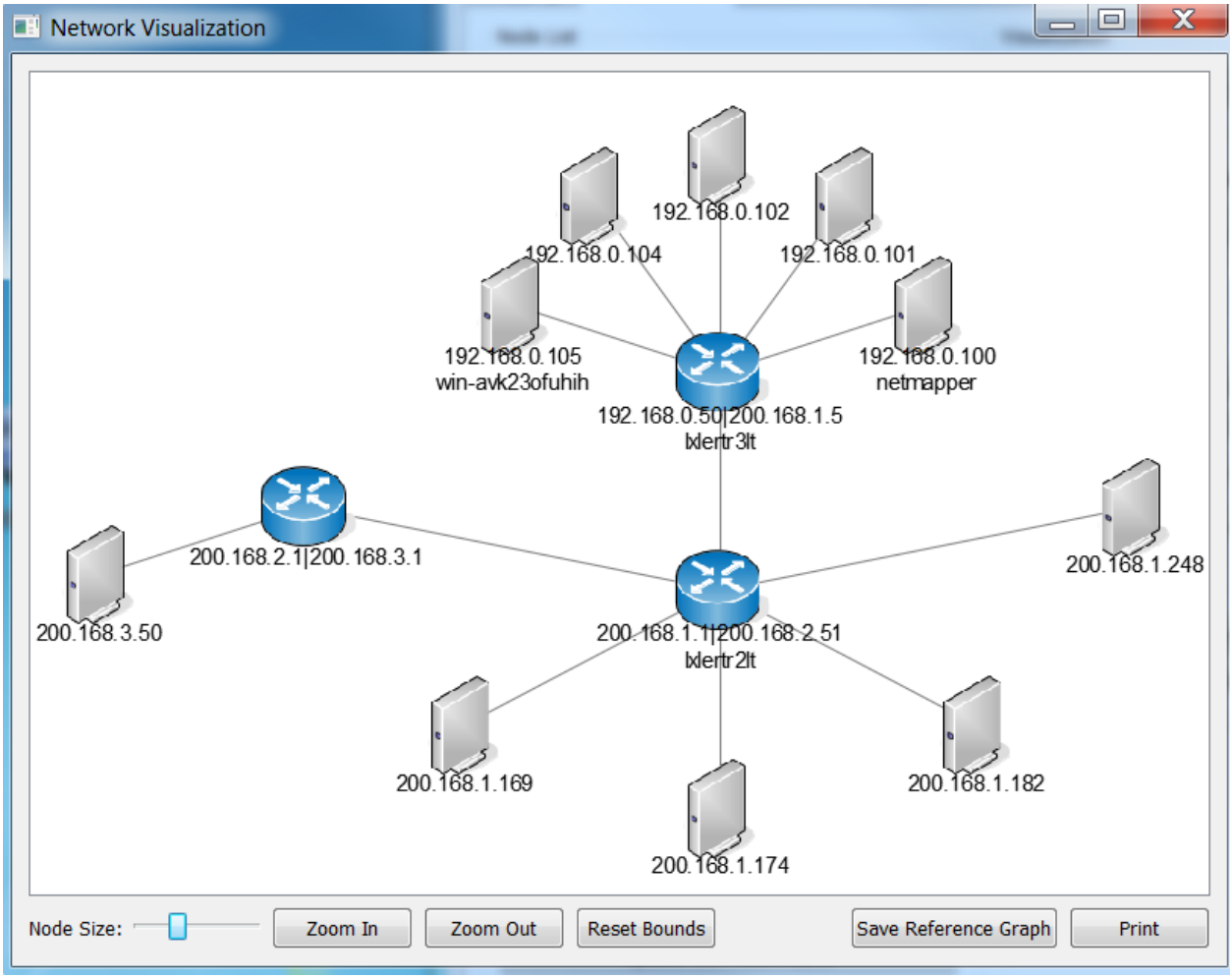


Fig. 4.35: An example of a dynamic L3 graph.

The same graph is shown below with edge labels and subnets enabled.. This graph has had its nodes re-arranged from

the default layout. Both of these graphs are created from the same L3 connection objects present in the XML.

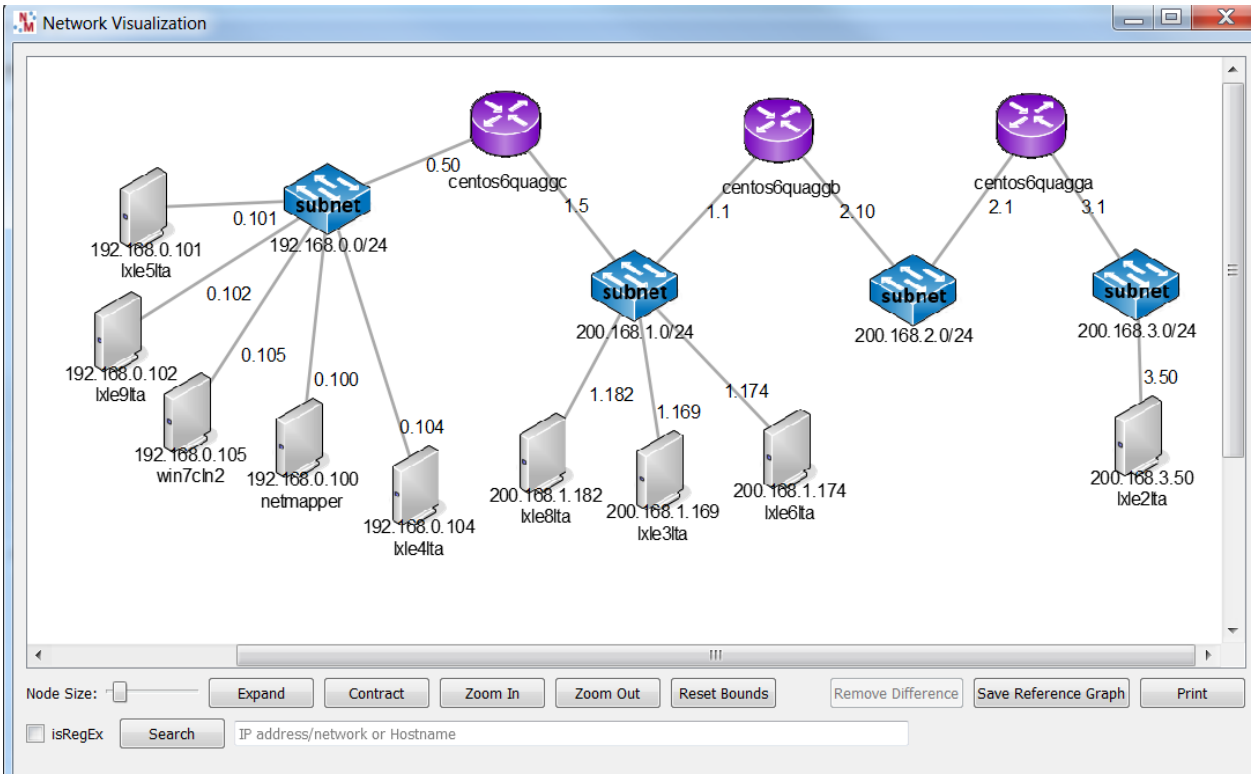


Fig. 4.36: An example of a dynamic L3 graph with subnets shown.

The plot in the previous figure is an L3 graph using IPv4. Multiple IP addresses assigned to a node means that the node has multiple interfaces.

PDF versions (as well as XPS) of dynamic graphs can be generated by using the *Print* button on the dynamic graph window.

Initially, nodes are automatically arranged and spaced on the plot. The *Plotting Options* control section has a combo box that allows the user to choose either the *twopi* or *dot* programs for node arrangement (*twopi and *dot* are from the *Graphviz* distribution).

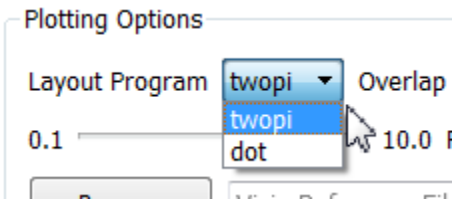


Fig. 4.37: Screenshot of the *Plotting Options* section.

The *twopi* program gives a radial node distribution while the *dot* program gives a tree structure.

The *Overlap Removal* combo box for chooses a program from the *Graphviz* distribution for handling node overlap. The *Rank Separation* slider control value is passed as a parameter to the chosen program. The *voronoi* program gives good results with large graphs, but the user is encouraged to experiment.

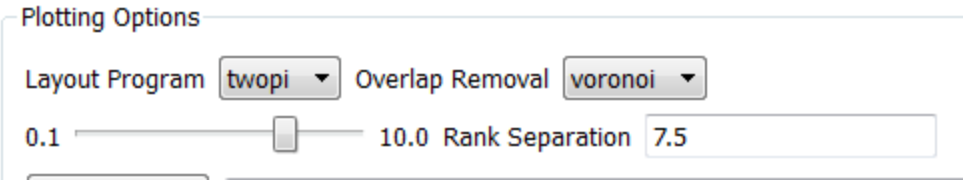


Fig. 4.38: Selecting the *voronoi* program for use in plotting large graphs is suggested.

Double-clicking on a link between two nodes in a dynamic graph displays a pop-up window that shows the interfaces that connect both side of the link.

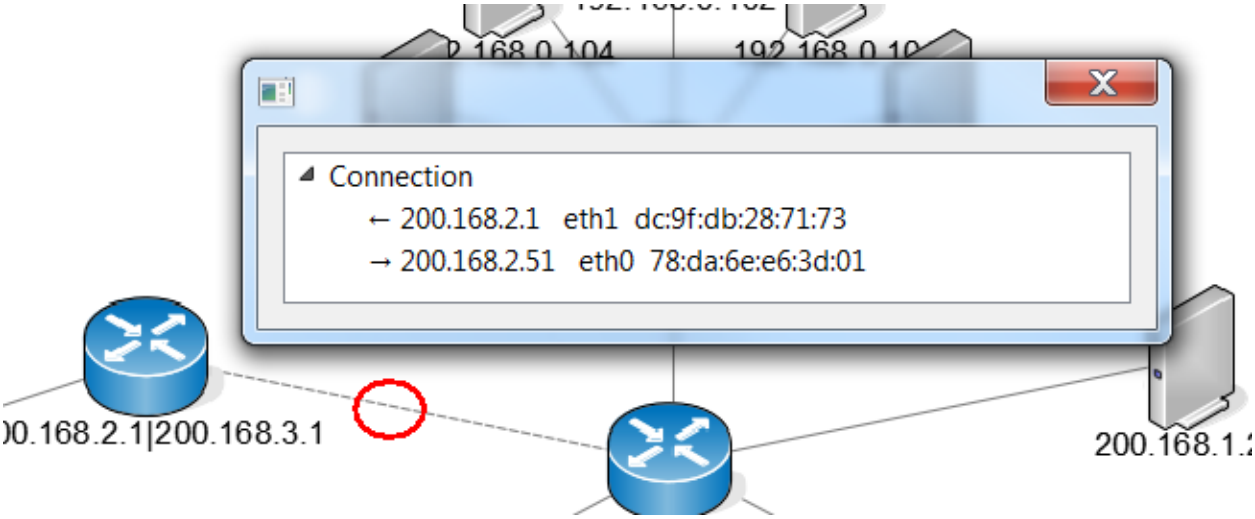


Fig. 4.39: Screenshot of the interfaces pop-up while viewing a dynamic graph.

Double-clicking on a node displays a pop-up window with the node description.

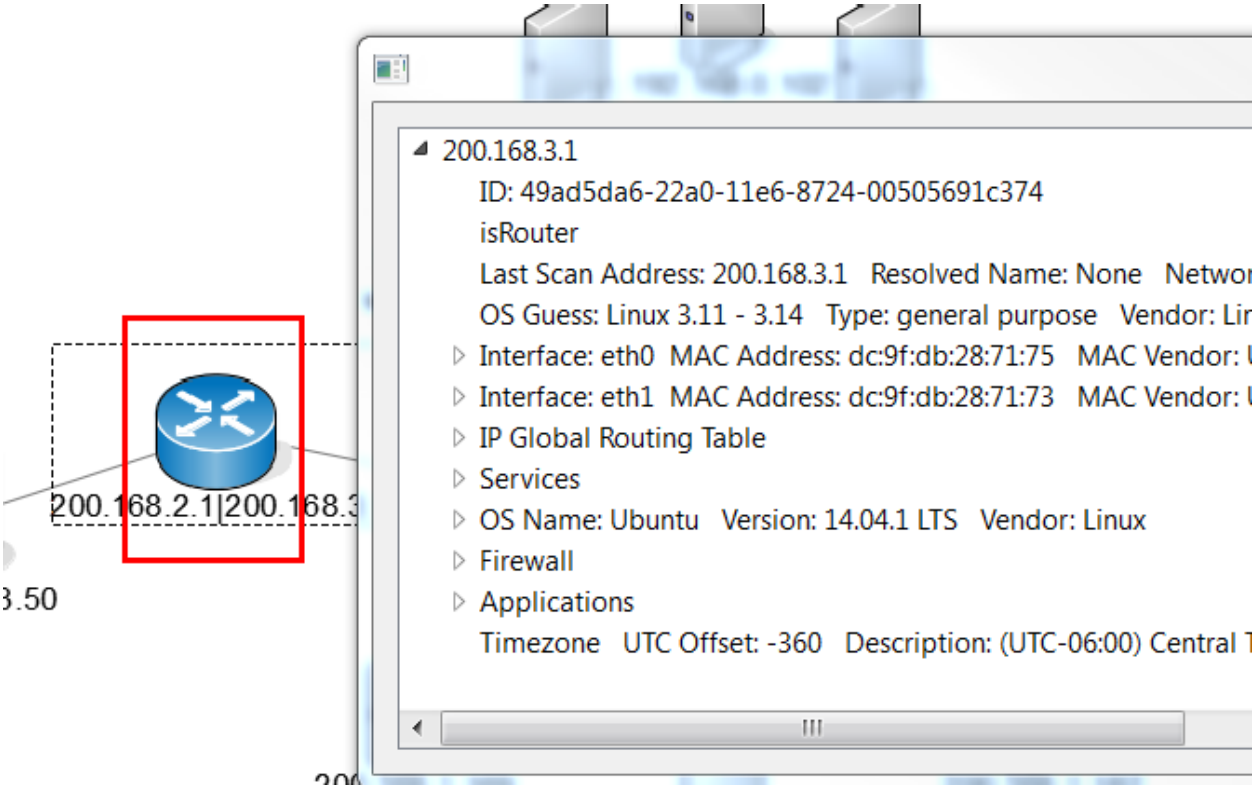


Fig. 4.40: Screenshot of the node pop-up while viewing a dynamic graph.

Dynamic graph customization

Once a dynamic graph has been plotted, the user can drag nodes around to position them as desired, and then use the *Save Reference Graph* button in the dynamic plot window to save this arrangement.

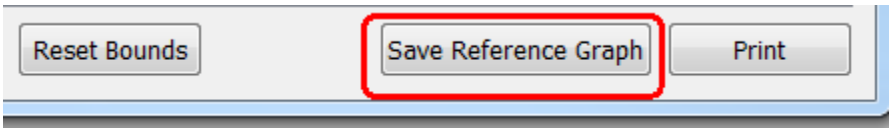


Fig. 4.41: Screenshot of the *Save Reference Graph* button.

To use this reference graph during dynamic graph plotting, use the *Browse* button for the dynamic plot reference file to browse to the saved reference file, and then plot the graph again.

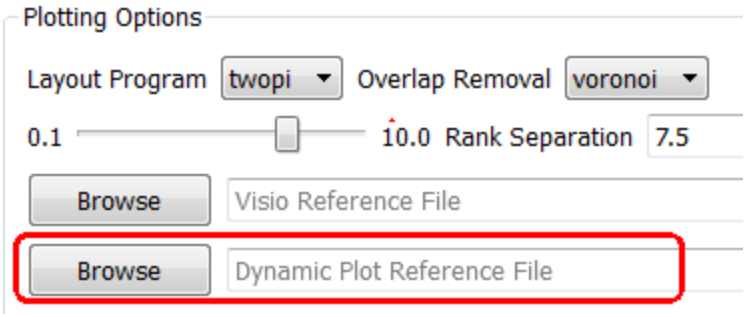


Fig. 4.42: Browsing for the saved reference file.

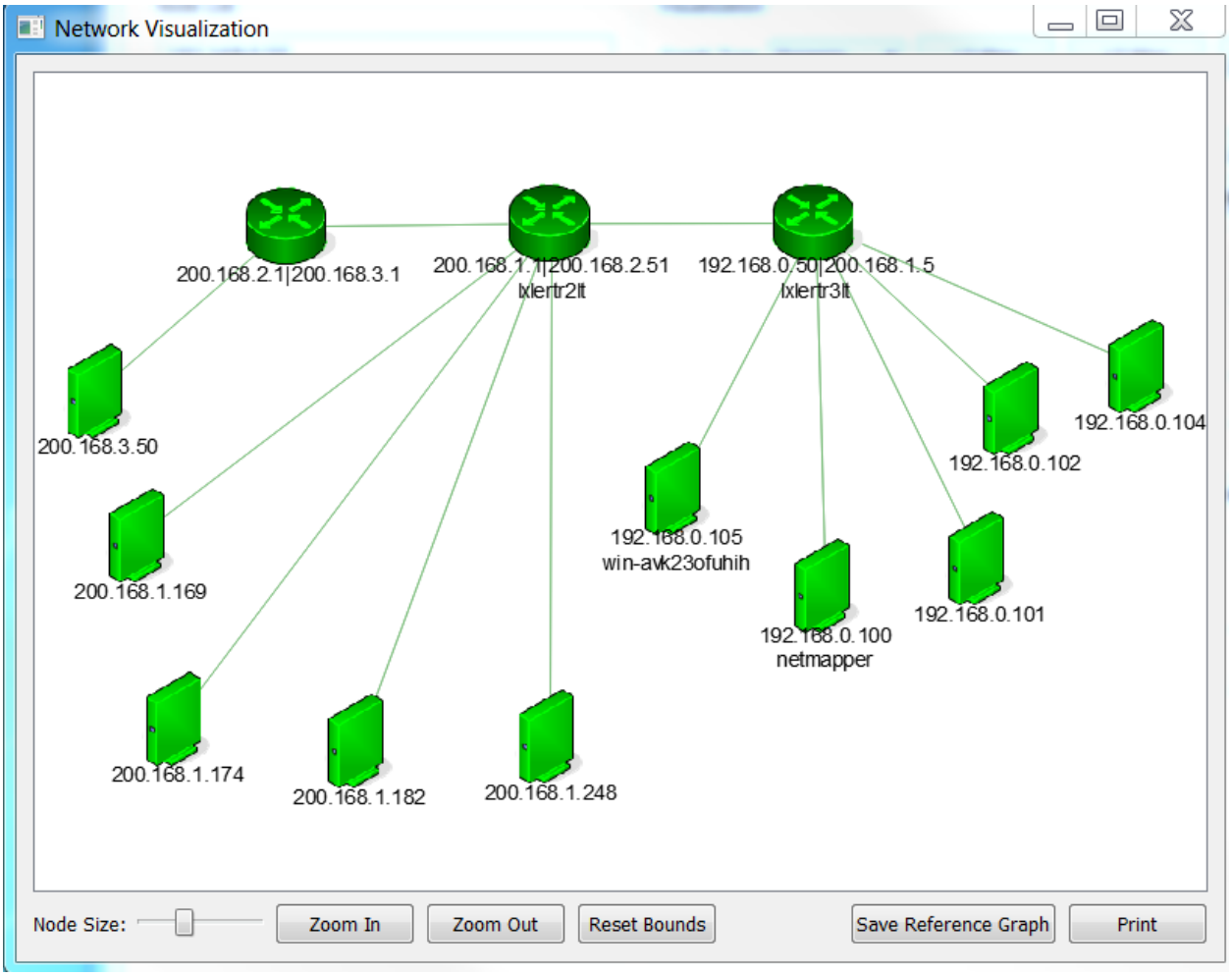


Fig. 4.43: A dynamic L3 graph created from a reference file.

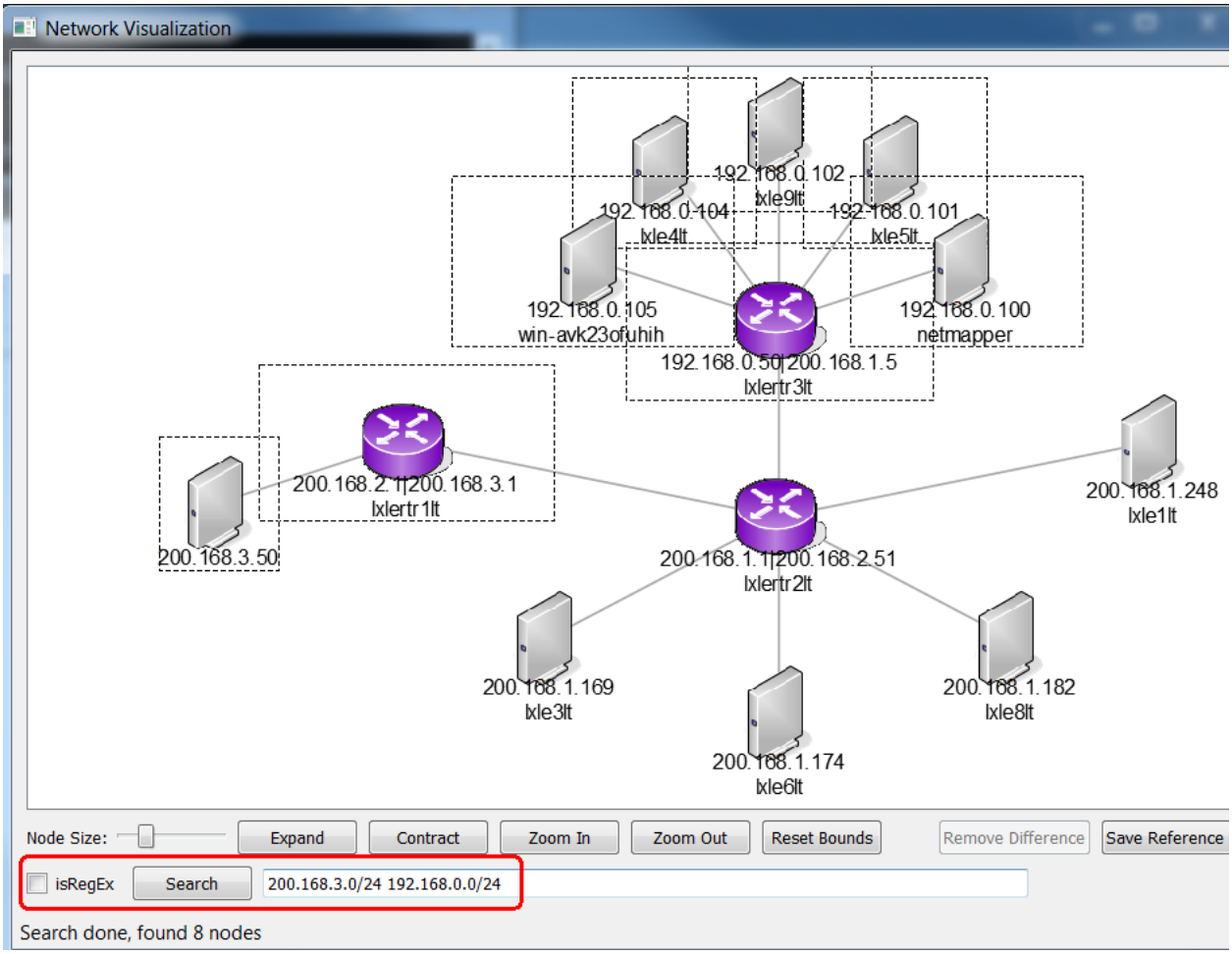
This does a graph comparison of the reference graph against the nodes in-memory. Links/nodes that match are highlighted in green; missing links/nodes are highlighted in red; and new links/nodes are highlighted in yellow.

The Visio plotting section that follows gives more examples of graph comparisons as its graph comparison works in the same manner.

Dynamic graph search

A dynamic graph can be searched by ip address, ip network, or hostname/resolved ip address name by entering these in the search type-in line at the bottom of the graph window and then clicking the *Search* button. Any nodes that match the search criteria will be selected. The *isRegEx* checkbox indicates if the search string is a regular expression, used to match hostname/resolved ip address names. If regular expressions are not used, then multiple ip addresses, ip networks, hostnames, resolved ip address names can be entered on the search type-in line, separated by whitespace.

In the example below, nodes belonging to networks 200.168.3.0/24, 192.168.0.0/24 are searched for. The search returned 8 nodes, which are selected.



4.4.4 Visio plotting and graphical comparison

If *Netmapper* detects that Visio is installed, then the *static-visio* option is available via the *Graph Type* combo box. This opens the Visio application and plots the graph in a Visio drawing using stencils from the *netmapper.vss* stencil file in the *Netmapper* installation directory.

An example of a Visio output plot is shown here. Each stencil has two custom properties named *nm_Hostname* and *nm_IPAddresses*. The *nm_Hostname* property is filled with either the FQDN or hostname, if available, from the corresponding network device. The *nm_IPAddresses* property is a list of all IP addresses associated with the interfaces of this network device. These property values are populated when the drawing is generated.

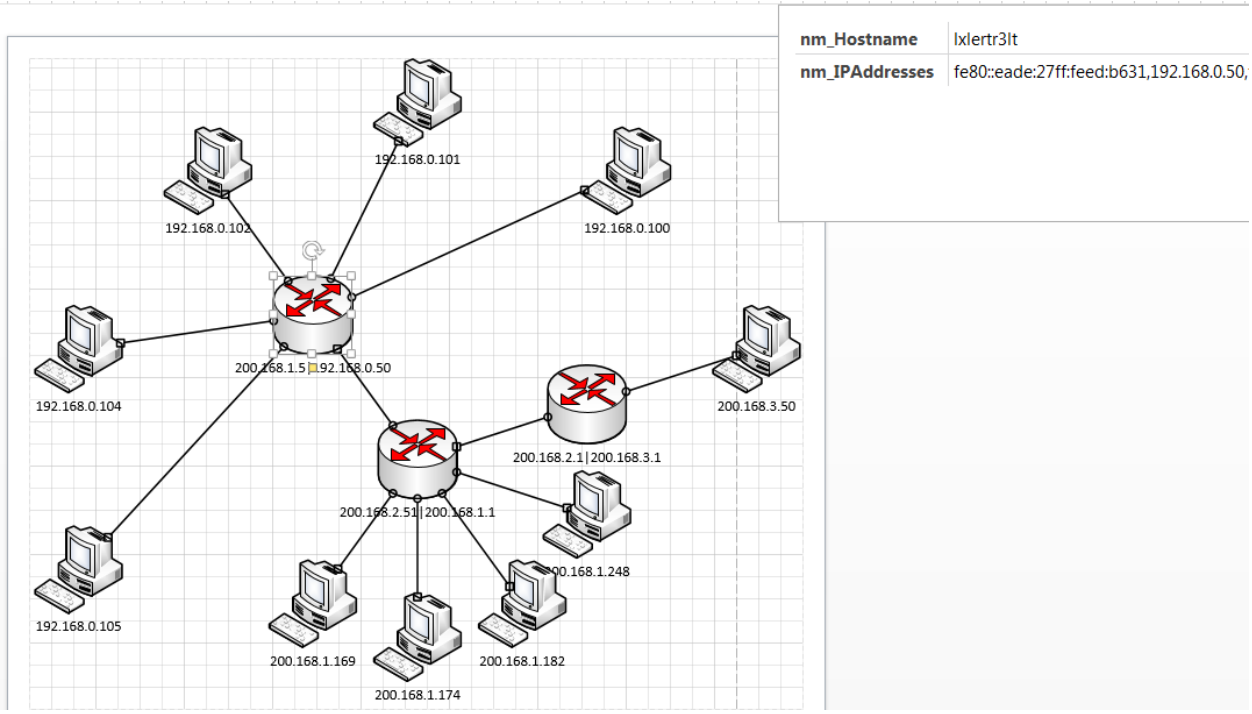


Fig. 4.44: Example Visio plot.

An alternative to simply plotting the graph in a Visio drawing is to graphically compare the *Netmapper* network against another network that is specified by a Visio reference file. The reference file is selected by the *Browse* button in the *Plotting Options* section before using one of the *L2*/*L3*/*vNet* buttons.

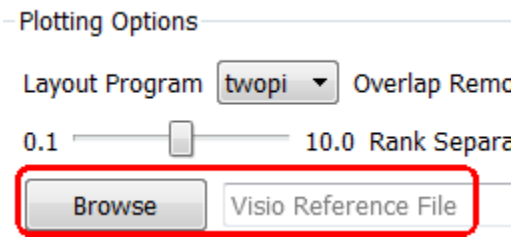


Fig. 4.45: Screenshot of the selecting a Visio Reference Graph.

The Visio stencil file with master shapes used to produce a Visio reference graph has the following restrictions:

- The stencil file is named *netmapper.vss* and is located in the installation directory.
- The master shapes (except for the *Connector* shape) must have the two custom properties *nm_Hostname* and *nm_IPAddresses* on them as string properties.
- The master shapes (except for the *Connector* shape) must also have a property named *nm_Type* of type string, and this property must have one of the following values: Router, Host, Switch, Subnet, Vnet, or Printer. The Host master shape is used for any object that is not a router, switch, subnet, vnet, or printer. The stencil must have a master shape for each of these types. The *nm_Type* property is used to identify which master shape in the stencil to use for representing a Netmapper object.
- The stencil must have a connector master shape named *Connector*. It does not require any other Netmapper-related properties.

The Visio reference file has the following restrictions:

- The drawing to be compared must be on the first page.
- Any stencils representing network objects to be compared must have the two custom properties *nm_Hostname* and *nm_IPAddresses* on them; shape name does not matter (except for the *Connector* master shape). One of these two fields should have a value in them that is to be compared against the network objects identified by *Netmapper*. The *nm_IPAddresses* property can be a comma separated list of IP addresses. The *nm_Hostname* value can either be a FQDN or a simple hostname, and the comparison is case insensitive. The *nm_Hostname* value, if present, is matched first. If this fails, then the *nm_IPAddresses* property is matched.
- Do not use grouping in the drawing that includes any Netmapper host objects or connectors; the host objects and connections should be all on the same object level. You can add any other objects to the drawing that you wish, as long they are not grouped with Netmapper host objects or connectors.

When comparing network objects, a hostname comparison is done first.

- If the FQDN and/or hostname is available (these are retrieved by remote login), then these are compared against the *nm_Hostname* property (if both FQDN and hostname are available, then both are compared to see if either match).
- If neither FQDN or hostname on a *Netmapper* object is available, then a resolved IP address name is used if one is available. It is assumed that all IP addresses on a object resolve to the same name, so the first IP address with a resolved named is used.

If the hostname comparison fails, then IP addresses are compared. If any IP address of the discovered *Netmapper* object matches an IP address from a *nm_IPAddresses* property, then the device matches.

Nodes and edges in the reference graph are highlighted in green. This figure shows an example of everything matching in the reference graph.

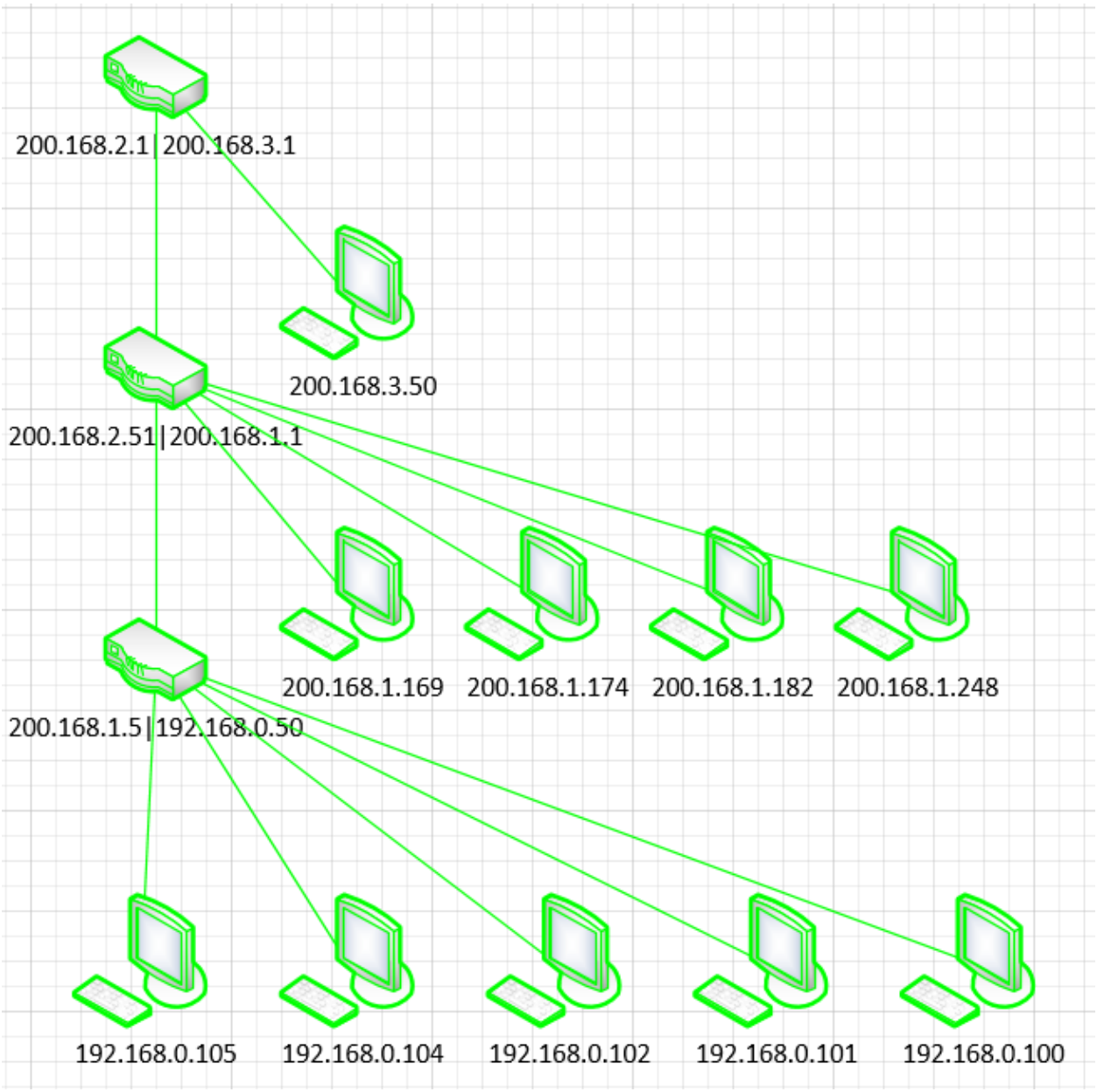


Fig. 4.46: Screenshot of matching Visio graphs.

A graph mismatch occurs if:

- The *Netmapper* network has extra node(s) or edges(s) over what is in the reference graph. These extra nodes are added to the reference drawing and highlighted in yellow.
- The *Netmapper* network is missing node(s) or edges from the reference graph. These missing nodes are highlighted in red in the reference drawing.

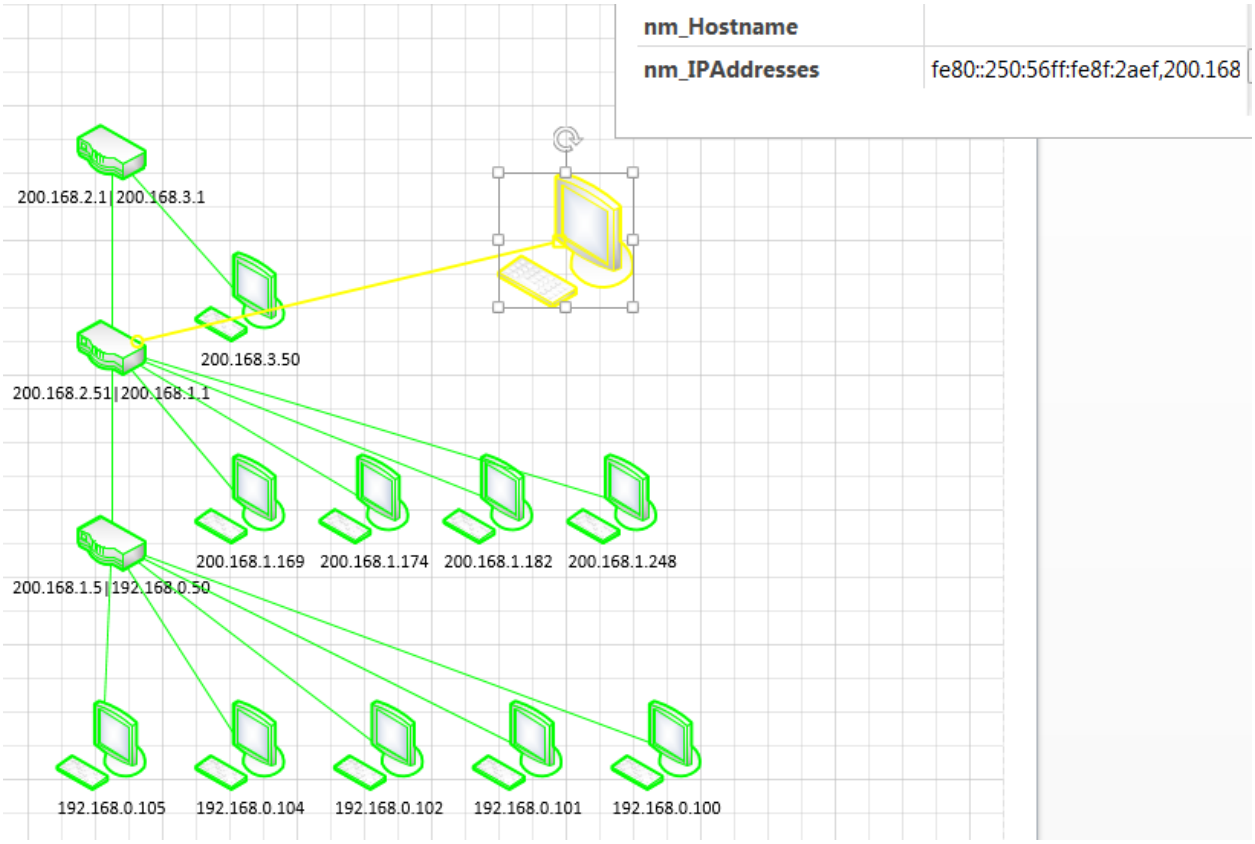


Fig. 4.47: Screenshot of a Visio graph with extra nodes.

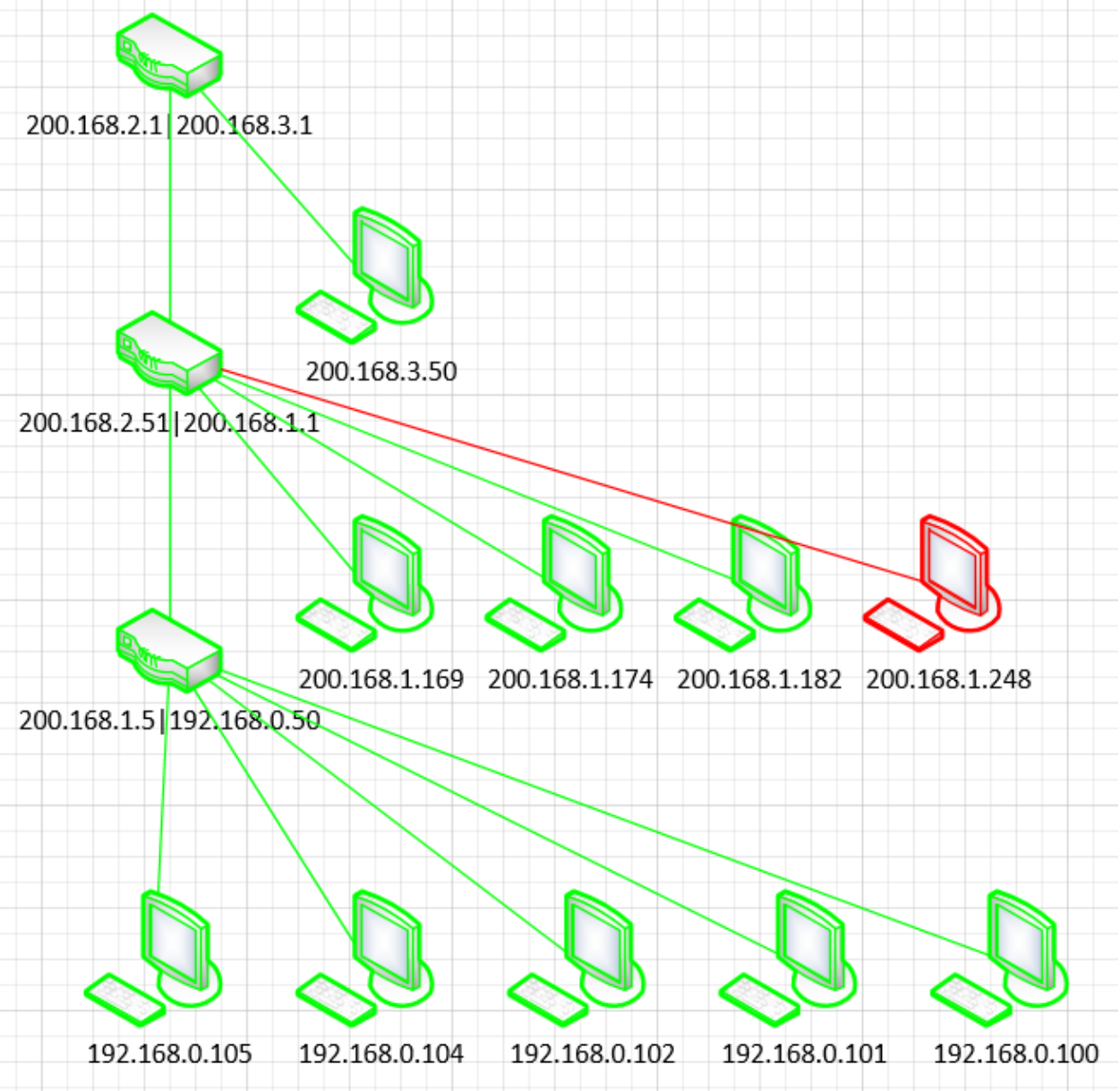


Fig. 4.48: Screenshot of a Visio graph with missing nodes.

The following example shows a Visio reference diagram comparison that has missing edges, missing nodes, and extra edges.

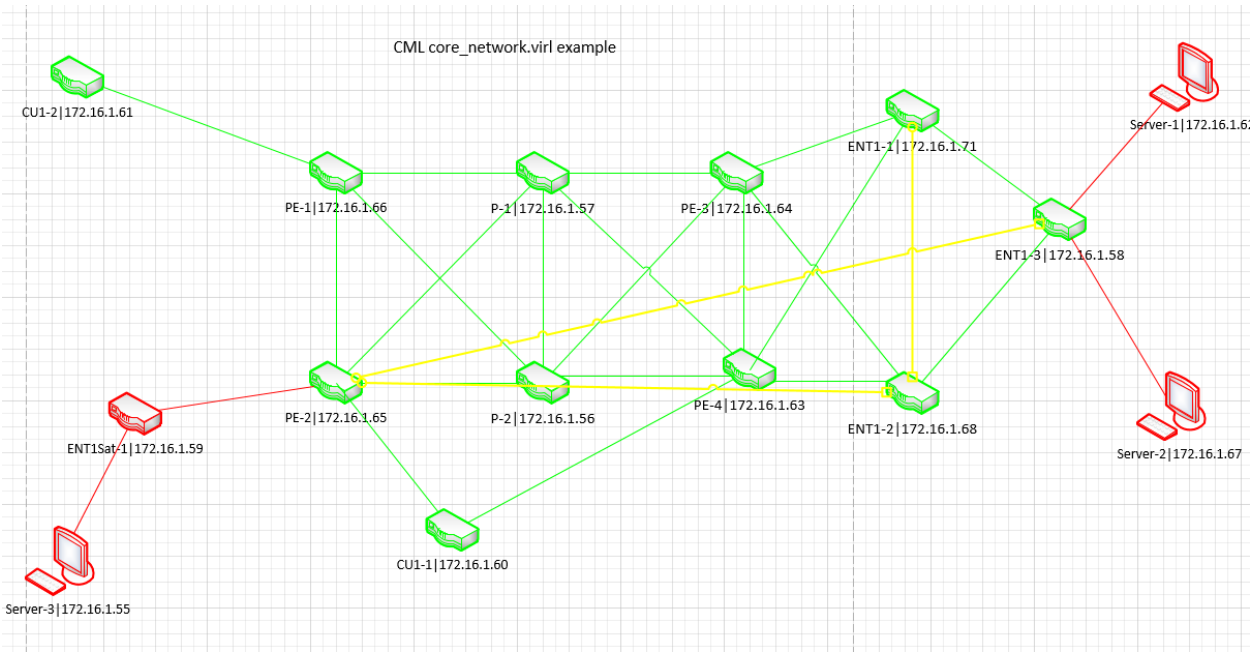


Fig. 4.49: Screenshot of a Visio graph that is very different.

The usage model for the Visio export/diff capability is that Netmapper is used to export an initial graph, that is then annotated by the user as desired to be later used as a comparison graph.

For example, the graph below is a Visio export with nodes minimally re-arranged.

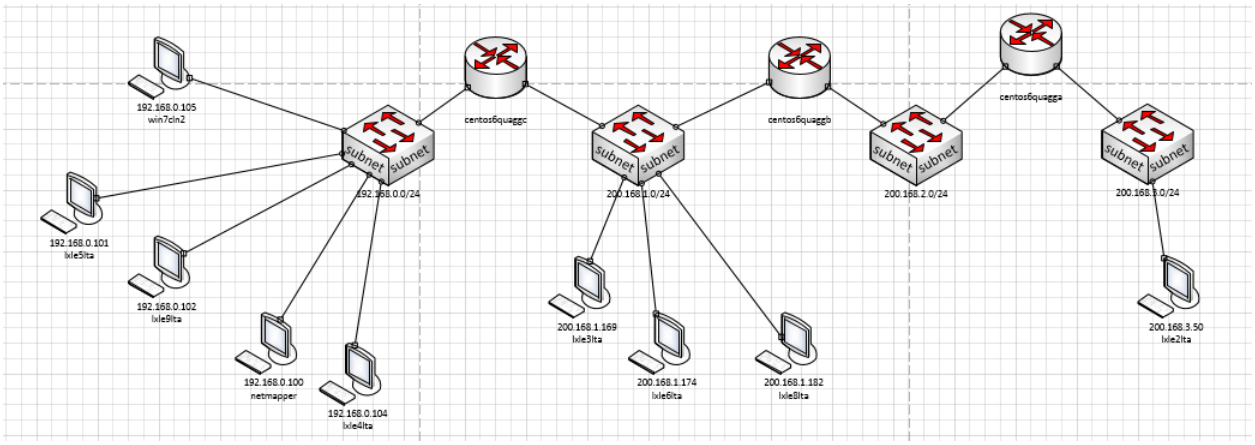


Fig. 4.50: Initial Visio graph with no annotation.

This graph is then annotated by the user with additional graphics and font changes to make it look nicer.

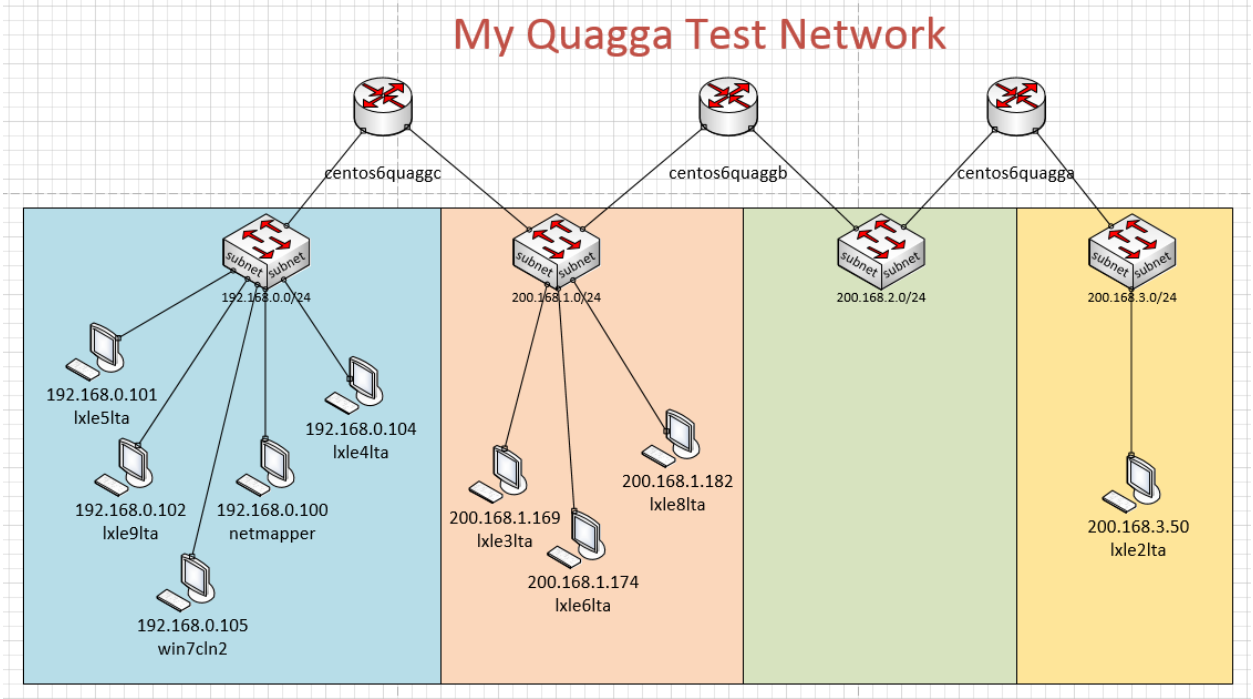


Fig. 4.51: Visio annotated graph.

The annotated graph can then be used for comparison purposes if the node objects and connections that were originally exported from Netmapper are preserved. The next image shows this graph used as a reference graph; note the green highlights mean that all nodes and connections matched.

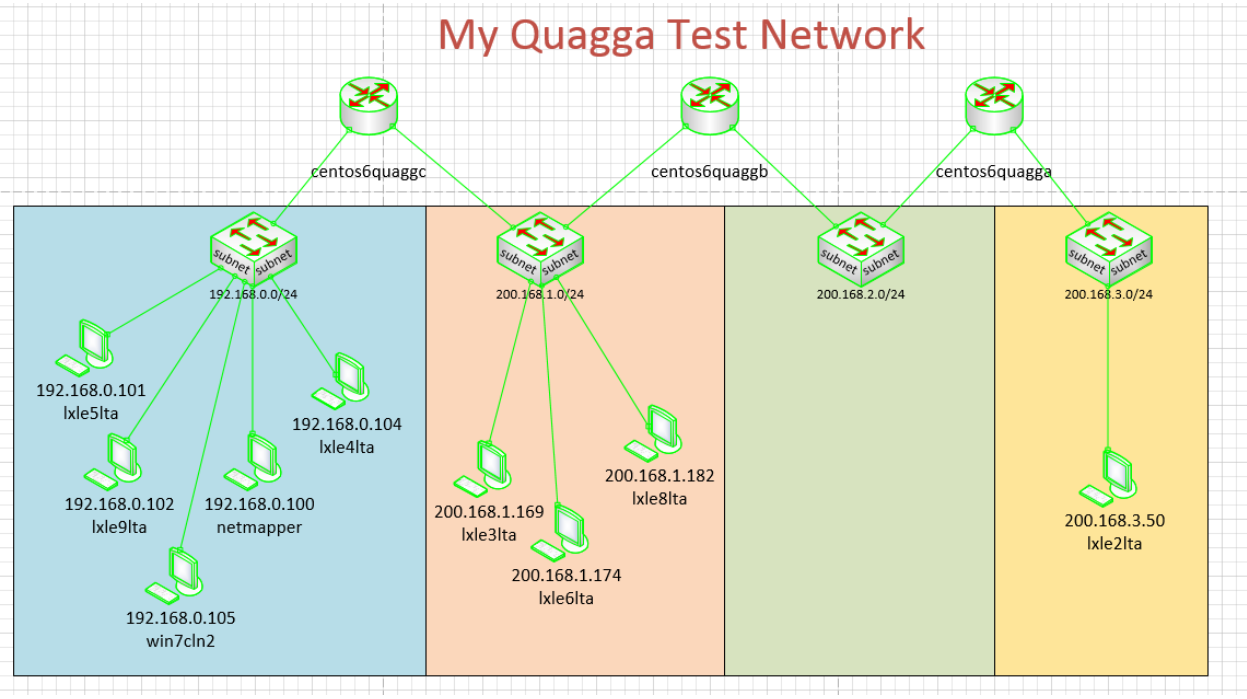


Fig. 4.52: Visio annotated graph used as a reference graph.

4.4.5 Static PDF Graph Type

The *static-pdf* choice from the *Graph Type* combo box in the Visualization group box displays a graph as a static PDF. The first time this option is chosen, the user is prompted to browse to a desired PDF viewer executable (typically, Acrobat.exe). This choice is then stored as a configuration option so that it does not have to be repeated.

Note: The *static-pdf* choice is a holdover from older versions of Netmapper and may be removed in the future; PDFs can be now generated from the *dynamic* and *static Visio* options.

4.5 *Credentials* Tab Details

The *Credentials* tab is used to enter/display credentials for SNMP and Remote login.

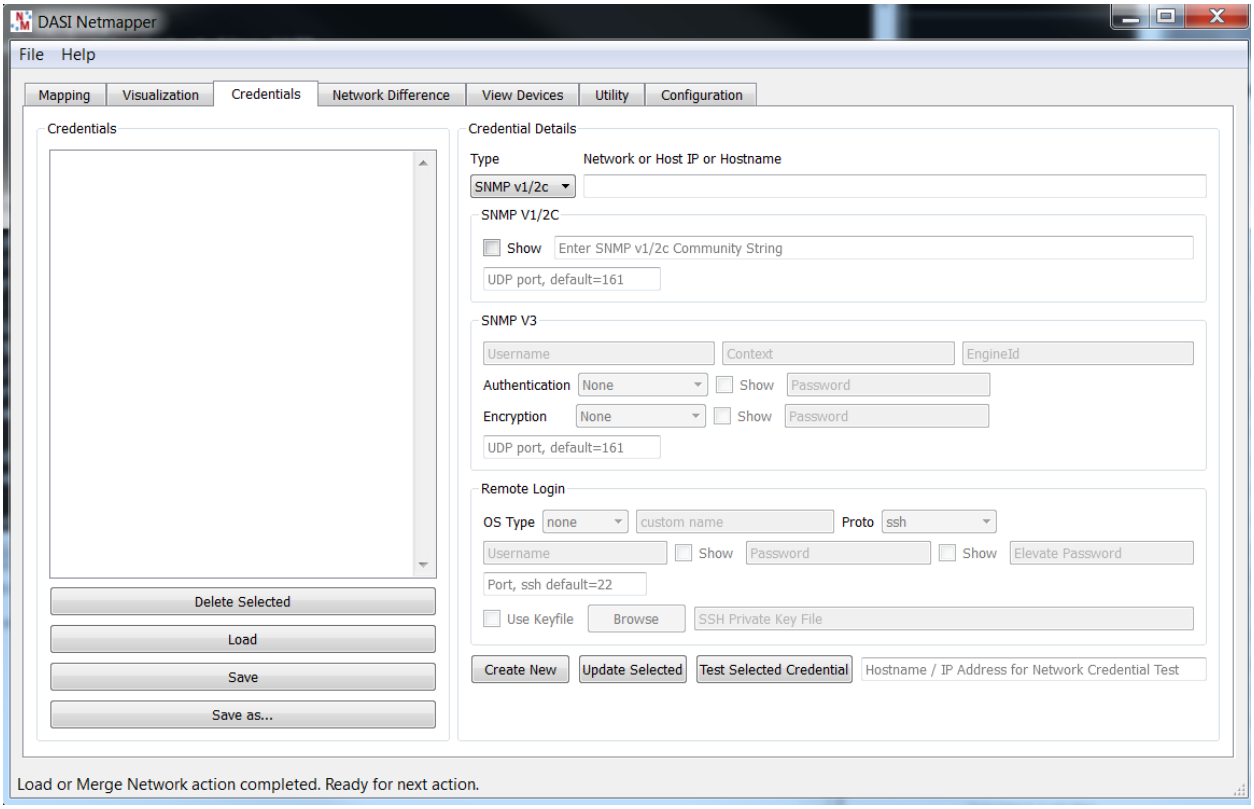


Fig. 4.53: Screenshot of the *Credentials* tab.

4.5.1 General controls

Three types of credentials are supported through the *Type* combo box: SNMP v1/2c, SNMP v3, and Login.

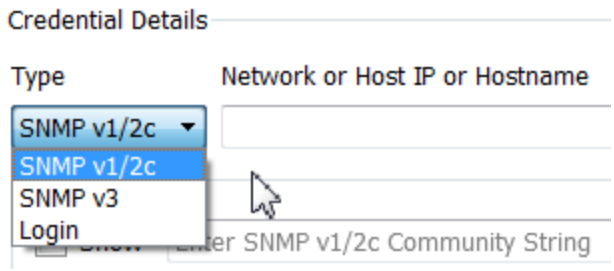


Fig. 4.54: Screenshot of the credential login type combo box.

The text edit control next to the *Type* control is used to specify the target for this credential. The target can be specified as a single host target (either IP address or hostname) or as a network in CIDR format. When *Netmapper* searches for a credential match to a node, it will always use a single host credential if present before using a network credential. An IPv4 network credential of 0.0.0.0/0 will match any IP address.

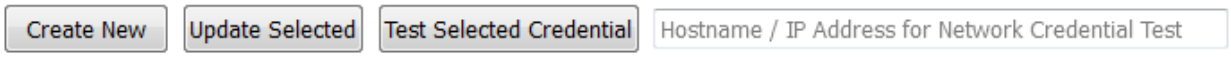


Fig. 4.55: Screenshot of the credentials entry interface.

When a credential is entered, the *Create New* button must be clicked for the credential to be created and entered into the *Credentials* list on the left-hand side.

Clicking on a displayed credential in the *Credentials* list will display the credential details in the appropriate controls. If these details are modified via the credential controls, then clicking the *Update Selected* button will make propagate these changes to the credential.

The *Test Selected Credential* button is used to test a selected credential. For a SNMP credential, this means making a request for the node's system description. For a login credential, this means achieving a successful login connection to the node. If the selected login credential specifies a network, then the text edit field below the *Test Selected Credential* button must be filled in with either an IP address or hostname in the target network.

The buttons underneath the *Credentials* list can be used to delete the selected credential (*Delete Selected*), load credentials from a file (*Load*), save credentials to the file they were loaded from (*Save*), or save the credentials to a new file (*Save as...*).

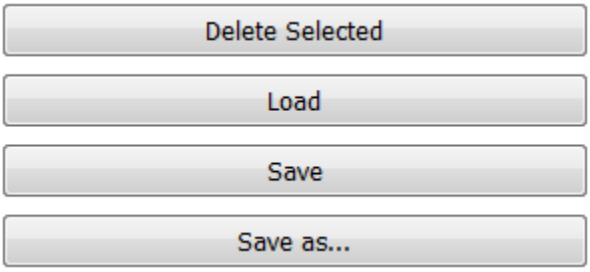


Fig. 4.56: Screenshot of the credentials editing interface.

When saving or loading credentials, the user is prompted for a passphrase. This passphrase is used to encrypt sensitive portions of the credential. Credentials are saved in an XML format, so specify an *.xml* extension when saving.

4.5.2 SNMP v1/2c credential

SNMP v1/2c credential entry uses a Text Edit control for entering a community string. The *Show* checkbox control, when checked, will display the true contents of this field.

During the SNMP information gathering phase, all devices are probed with a SNMP v1/2c credential with community string equal to public if no other SNMP credential is found for this node (this can be disabled by a configuration option). A UDP port number other than the 161 default can be entered for the credential using the type-in box.

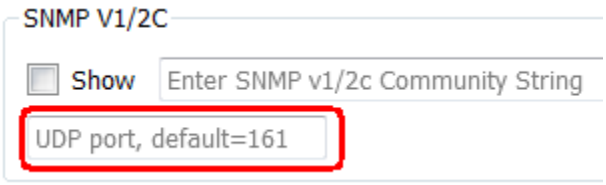


Fig. 4.57: Entering an SNMP v1/2c credential.

4.5.3 SNMP v3 credential

The SNMP v3 credential control is shown below. SNMP v3 credentials have many optional parts, and it depends on how the target devices is configured as to what parts are needed in an SNMP v3 credential.

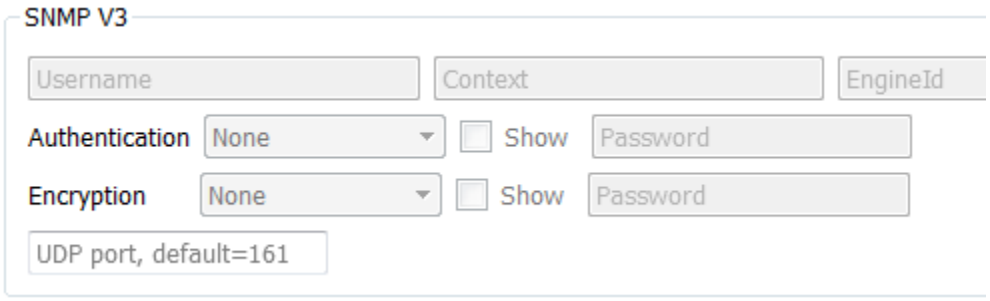


Fig. 4.58: Entering an SNMP v3 credential.

The username is non-optional and must be filled in for the credential. The *Context* and *EngineId* are optional fields that can be specified. The *Context* field is simply a string that can be used to disambiguate a query to a device that has multiple copies of a MIB. The *EngineId*, if required, must be specified as a hex number without a leading 0x (ie. 80000009010A0101C1). An *EngineId* can be up to 32 bytes in length (so up to 64 hex digits) and is the *EngineId* of the remote SNMP device. Typically, this does not have to be specified.

The *Authentication* combo box has three choices: *None*, *MD5*, and *SHA-1*. If the *Authentication* choice is not *None*, then the associated password text edit must be filled in.

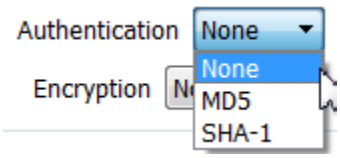


Fig. 4.59: Entering the authentication for an SNMP v3 credential.

The *Encryption* combo box has five choices: *None*, *DES*, *AES-128*, *AES-192* (Cisco), *AES-256* (Cisco) and *3DES* (triple-DES). If the *Encryption* combo box choice is not *None*, then the associated password text edit must be filled in. The encryption choices of *AES-192* (Cisco) and *AES-256* (Cisco) implement the Cisco method of AES-192/256 and may not work with other vendors that support SNMPV3 AES192/256 encryption.

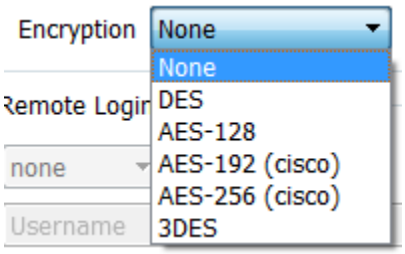


Fig. 4.60: Entering the encryption method for an SNMP v3 credential.

An authentication/encryption combination of *none*/*none* is typically referred to as the *noAuth*/*noPriv* selection. Valid combinations of authentication/encryption are: none/none (noAuth/noPriv), non-none/none (Auth/noPriv), and non-none/non-none (Auth/Priv). If Encryption is not *none*, then authentication must be something other than none as well (a choice of noAuth/Priv is illegal). Authentication and Privacy passwords must be 8 characters or greater.

SNMP v3 examples

It can be difficult to configure SNMP v3 for correct authentication due its many options. If problems are encountered with configuring *Netmapper* SNMP v3 credentials, then use a third-party tool to independently verify the SNMP v3 credentials. An excellent tool is SnmpGet by *SnmpSoft*.

The following three images show a *Netmapper* SNMP v3 credential test to a TP-LINK managed switch; one of the images show the same access using the *SnmpSoft SnmpGet* tool.

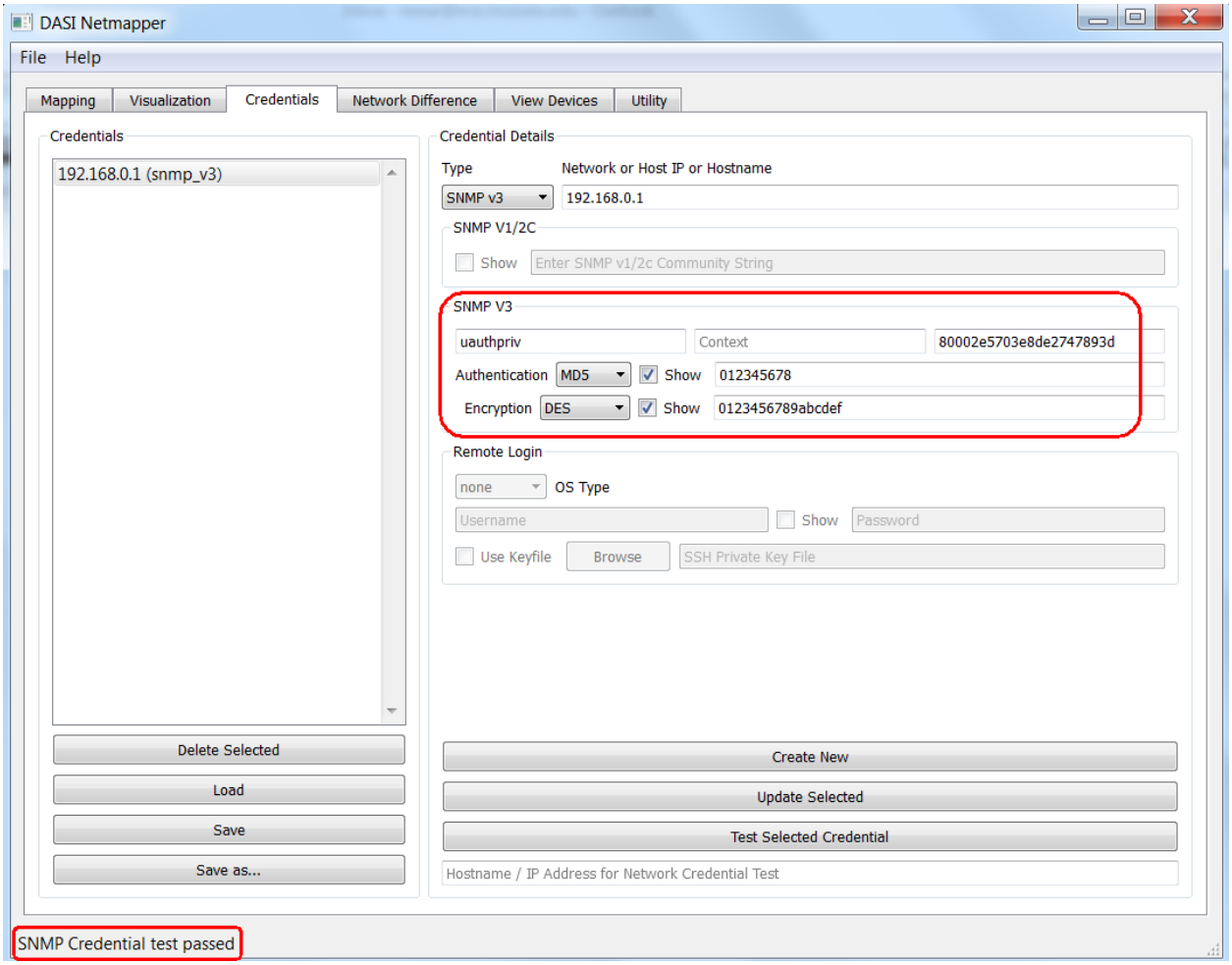


Fig. 4.61: SNMP v3 credential testing for a TP-LINK managed switch.

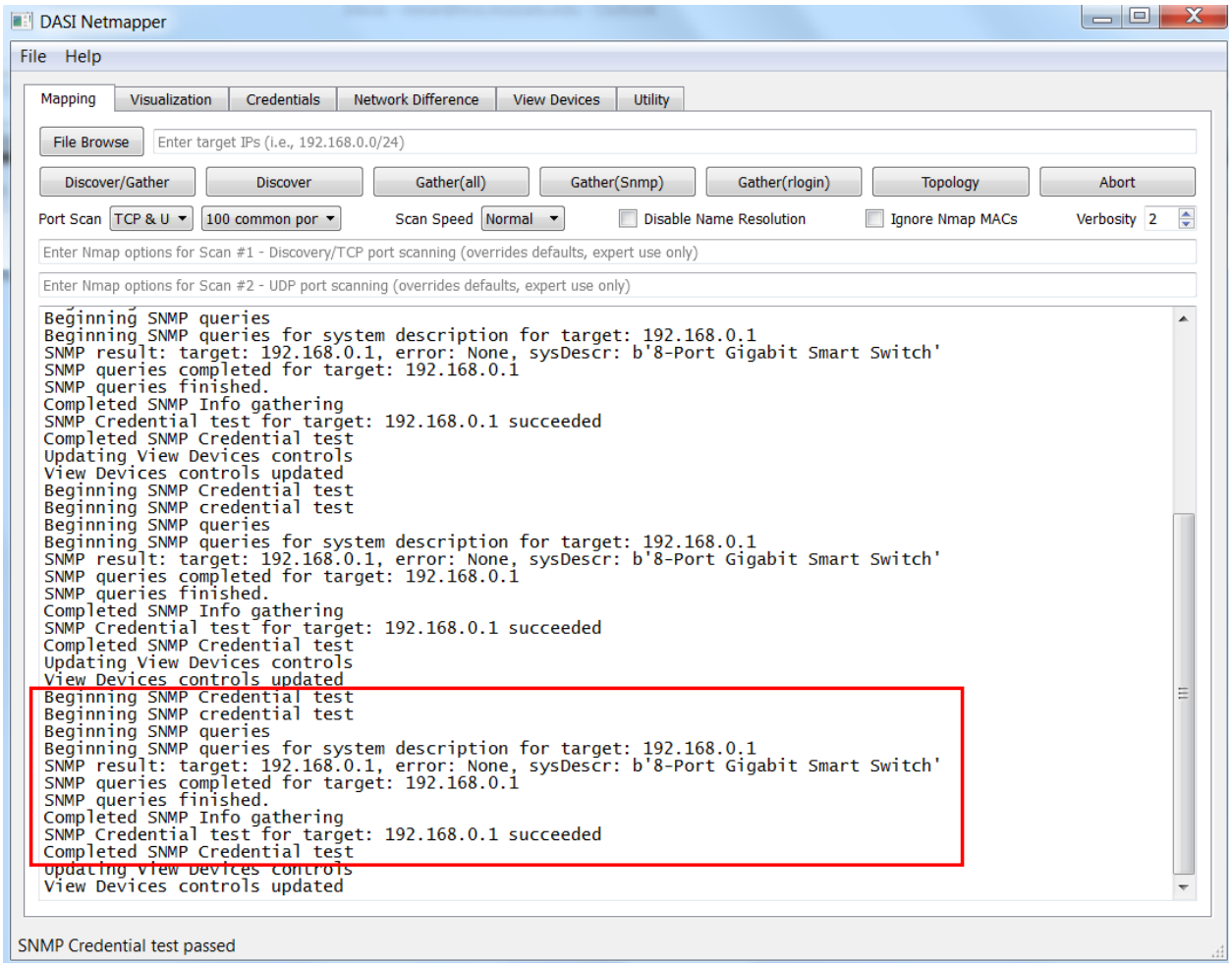


Fig. 4.62: SNMP v3 credential test output for a TP-LINK managed switch.

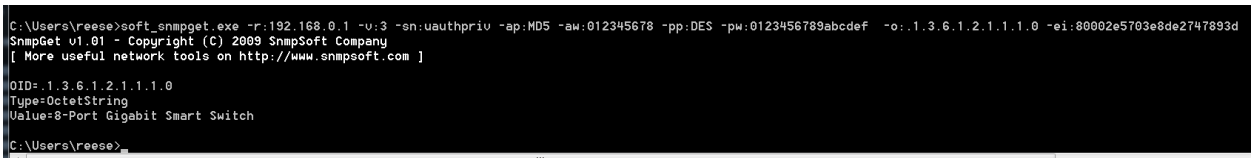


Fig. 4.63: *SnmpSoft SnmpGet* test output for a TP-LINK managed switch.

The following three images show a *Netmapper* SNMP v3 credential test to a Cisco Small Business router; one of the images shows the same access using the SnmpSoft SnmpGet tool.

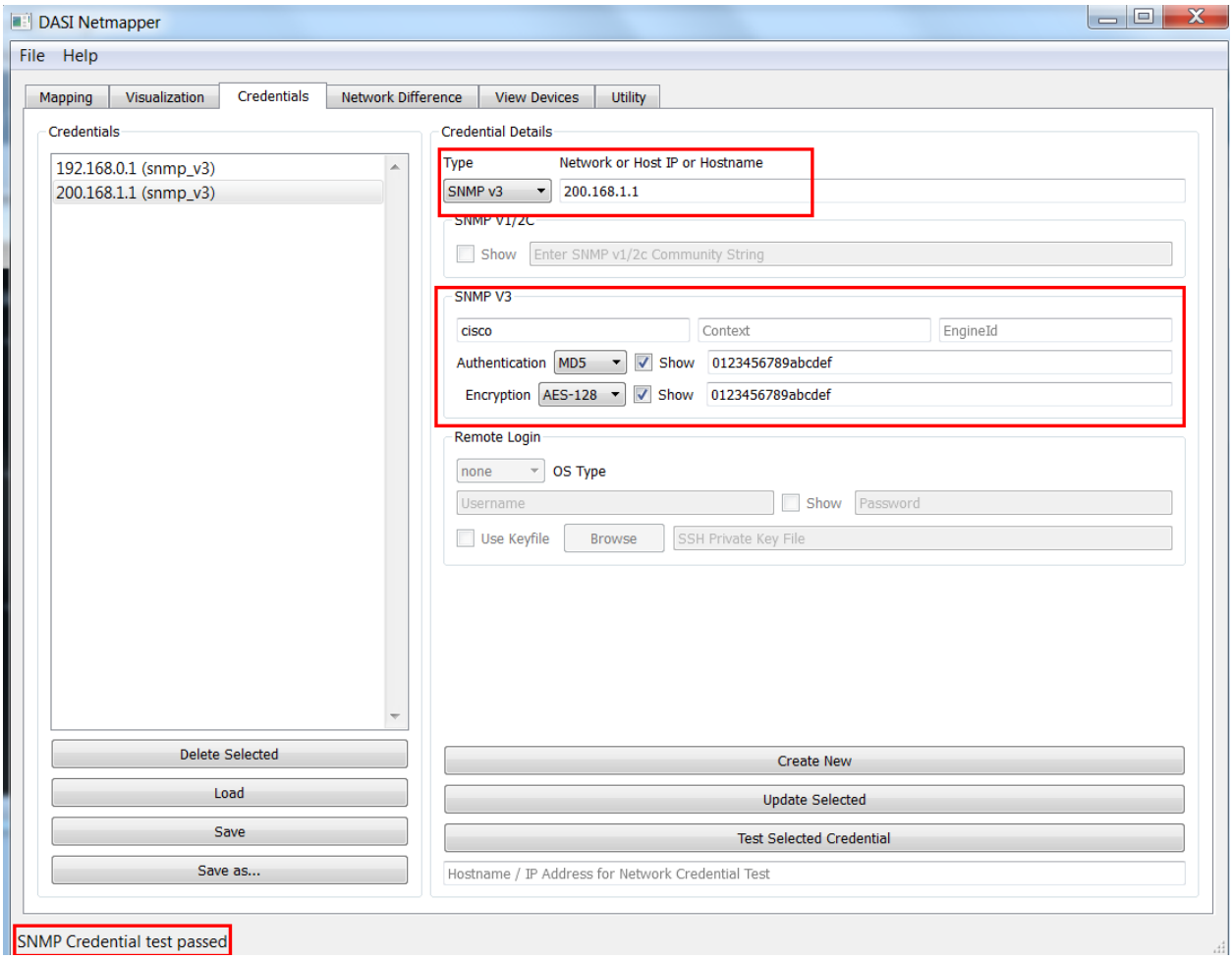


Fig. 4.64: SNMP v3 credential testing for a Cisco Small Business router.

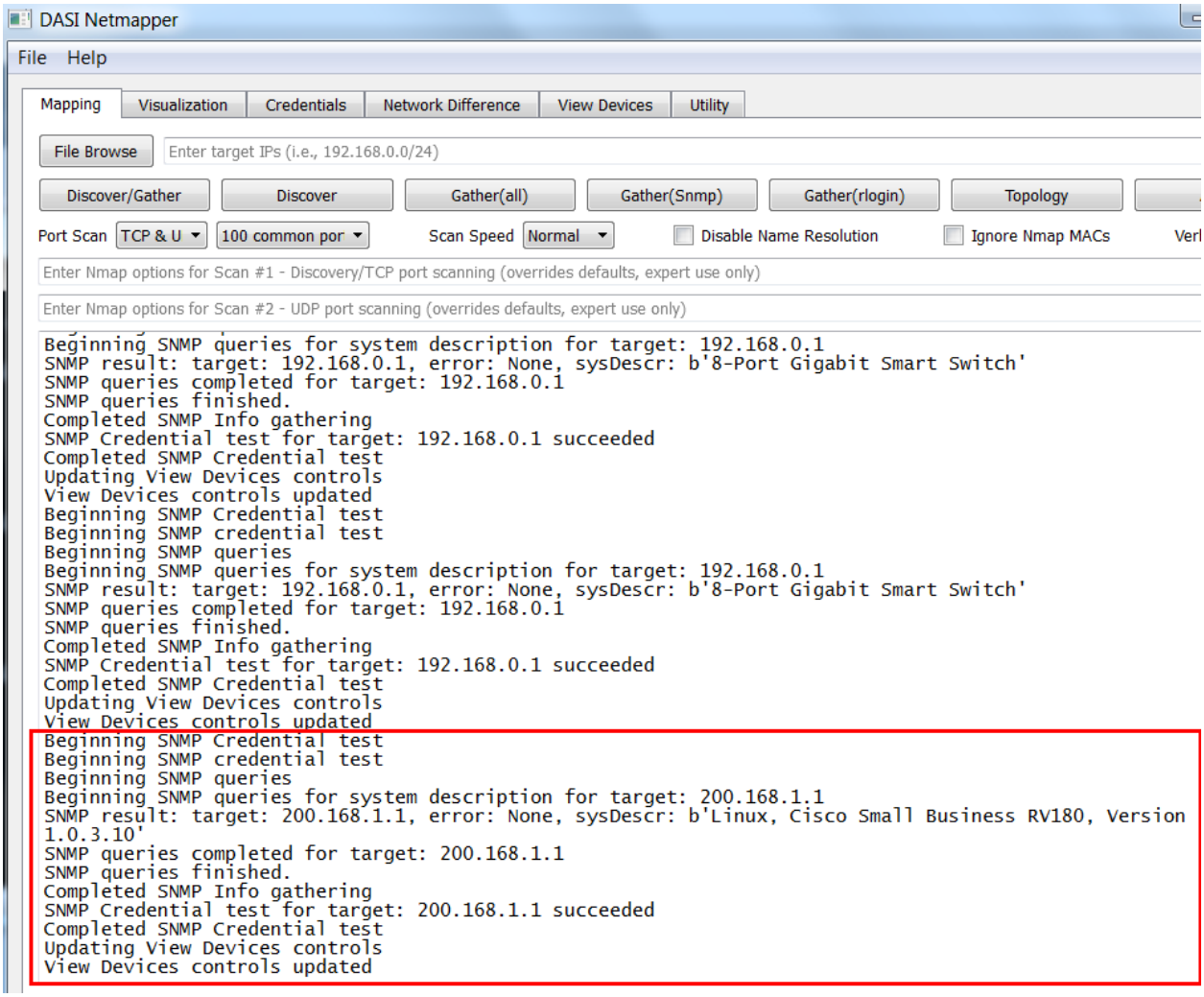


Fig. 4.65: SNMP v3 credential test output for a Cisco Small Business router.

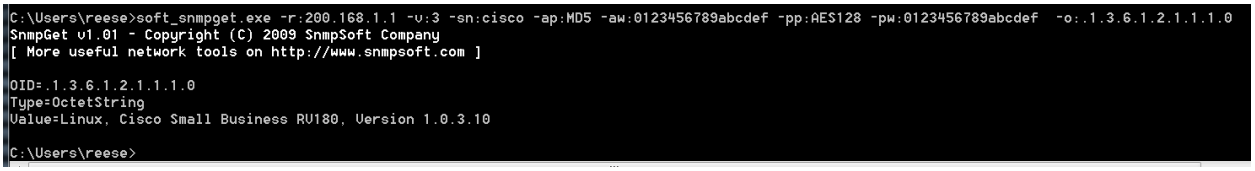


Fig. 4.66: *SnmpSoft SnmpGet* test output for a Cisco Small Business router.

4.5.4 Login credential

The login credential controls are shown below.

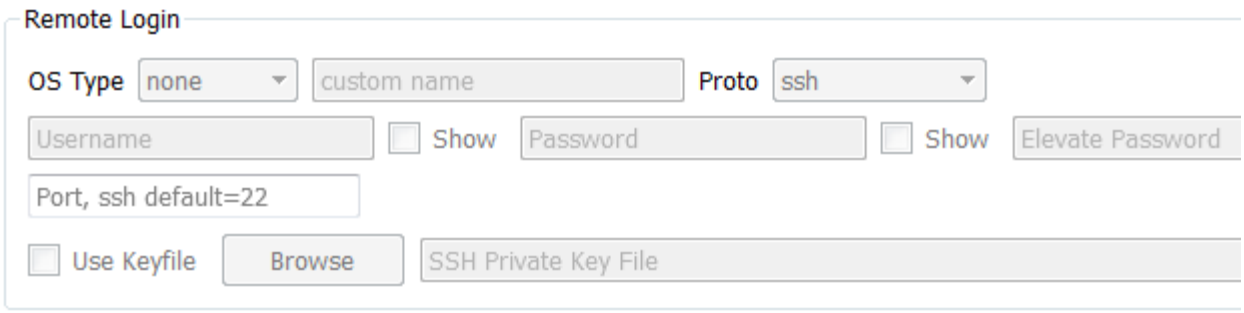


Fig. 4.67: Screenshot of the login credential controls.

Remote login interrogation is done either by SSH or by WMI/Powershell. The *OS Type* combo box has the following choices: *none*, *linux*, *windows*, or *custom*. These choices are tags that decide whether the SSH or WMI/Powershell method is used on the remote device.

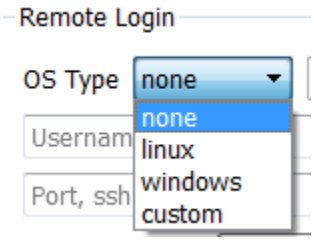


Fig. 4.68: Screenshot of the *OS Type* combo box.

The *linux* choice means that an SSH connection will be made. When an SSH connection is established, commands are executed to determine the OS type/version (Centos, Ubuntu, Cisco appliance, etc). After the OS type/version is determined, commands specific to that OS type/version are executed to return information.

The *windows* choice means that the WMI/Powershell interrogation method is used. A *none* OS type means that both SSH and WMI/Powershell connection attempts are made, with the order dependent upon the OS guess returned by the *Nmap* discovery scan. If the OS guess returned by *Nmap* is neither Linux or Windows, or if no OS guess is present, then an SSH connection is tried first, followed by a WMI/Powershell connection attempt.

The *custom* choice allows the user to specify a custom OS tag for this credential, and specify the connection type. In the example below, a custom tag of *cisco* is used with the SSH protocol. It is convenient to think of the *linux* and *windows* choices as short cuts for custom tags that use the SSH and WMI/Powershell protocols respectively.

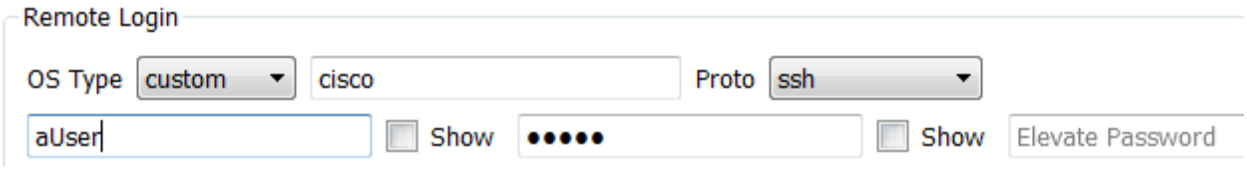


Fig. 4.69: Options for remote login.

The *Username*/*Password* text edits must be filled in. The *Elevate Password* text edit is optional. For Cisco devices, this is the password needed for elevation to the highest privilege level via the enable command if the user does not have this privilege on initial login. This password is only used if it is 1) supplied, and 2) if it is detected after login to a

Cisco device that the user does not have access to the highest privilege. For a Cisco ASA device, the *Elevate Password* should always be supplied as the highest privilege is needed for executing even basic show commands.

If the *Use Keyfile* checkbox is selected, then the password field is the password for the private key file specified in the SSH Private Key File text edit control (this field can be left blank if the SSH file does not have a password). The *Use Keyfile* option is only allowed for credentials with OS type of *linux* or *custom* with proto of SSH. If an SSH Key file is used for a Linux file login, then the *Elevate Password* field should be the password of the user, as this is needed for *sudo* operations. For Linux systems, if the user is root and the SSH key file is for the root user, then the *Elevate Password* field can be left empty as *sudo* commands are not used for the root user (warning, the root login is generally not available by default on Ubuntu systems).

To target the same host (or network) with different users, the credentials must be differentiated by the *custom* tag (Credentials are indexed by credential type, target name, and OS type (if OS type is *custom*, then also by custom tag). This can cause a *netobject* to be successfully matched against more than one credential. In this case, the matching credentials are tried in an unspecified order with the first one that achieves a successful connection used to return data for that *netobject*.

4.6 *Network Difference* Tab Details

This figure shows the *Network Difference* tab which is used to intelligently compare the XML files of two networks.

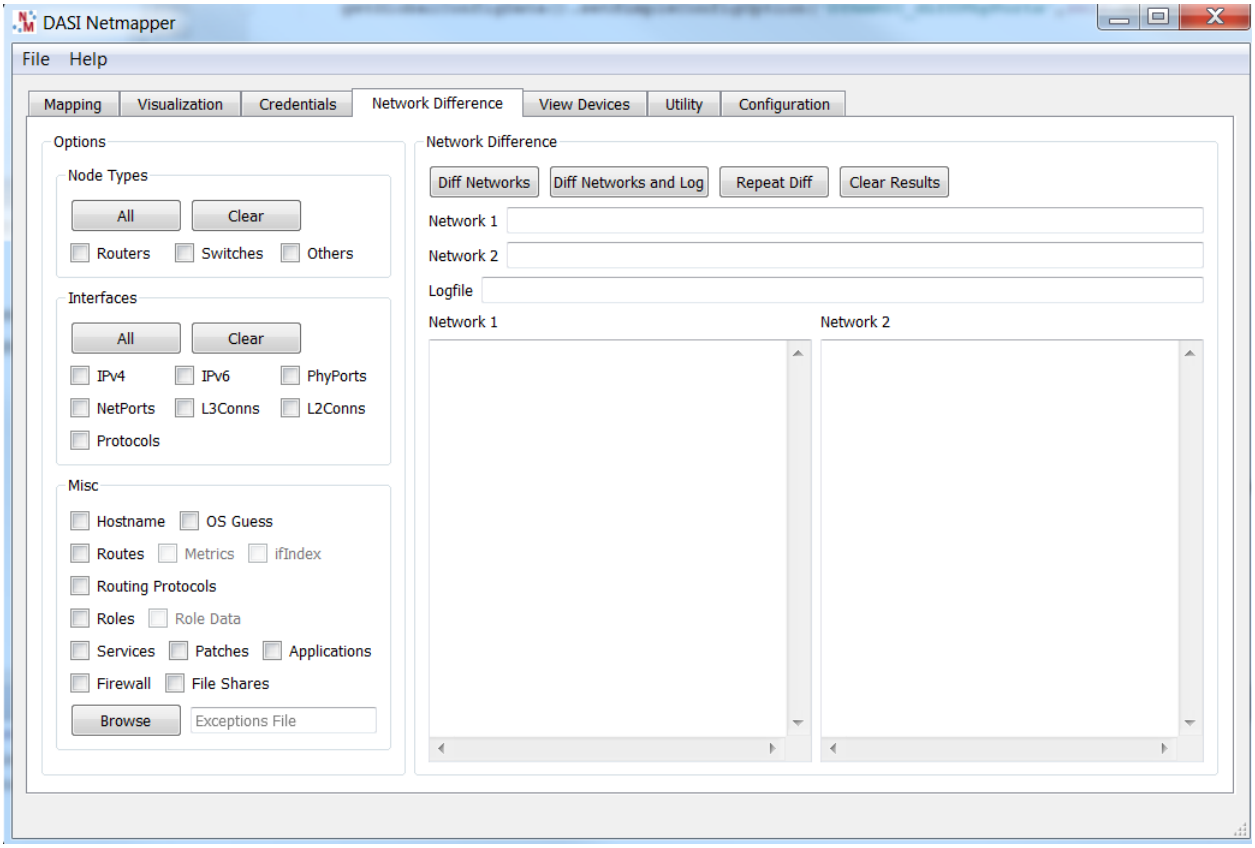


Fig. 4.70: Screenshot of the *Network Difference* tab.

The *Diff Networks* button prompts the user to browse to two different XML files. Once the files are specified, the XML files are loaded, and then an intelligent in-memory difference of the resulting network objects is performed and the

results are displayed in the two output panes (*Network 1* and *Network 2*).

The *Diff Networks* and *Log* buttons prompt the user for a third file to which any differences are saved.

The *Repeat Diff* button repeats the network difference on the currently specified files – this is useful if any options have been changed in the *Options* control.

The *Clear Results* button clears the result output panes.

4.6.1 Options Control

The Options control provides the user with control over the fidelity of the network comparison.

Node Type control

The *Node Type* controls specify what nodes are to be compared. The three checkboxes specify nodes by type (*Routers*, *Switches*, *Others*). The *All* button causes all checkboxes to be selected while the *Clear* button clears all checkboxes.

Interfaces control

The Interfaces control specifies the fidelity of the comparison of node information found in interfaces.

- IPv4 – compare IPv4 addresses
- IPv6 – compare IPv6 addresses
- PhyPorts – compare physical ports (returned by SNMP/Rlogin query)
- NetPorts – compare scanned ports (returned by *Nmap* scanning). It is suggested that this be disabled if using the network difference for regression testing as *Nmap* port scanning can be unreliable.
- L3Conns – compare L3 topology connections
- L2Conns – compare L2 topology connections
- Protocols – compare routing protocols identified for this interface

Miscellaneous control

The Misc control specifies the fidelity of the comparison of node information outside of MAC interfaces. Most of this data is retrieved via remote login and not by SNMP.

The majority of these options are self-explanatory. The OS guess is returned by *Nmap* scanning and is based upon the port signature found. It is suggested that this be disabled if using the network difference for regression testing as *Nmap* port scanning can be unreliable.

The *Role Data* comparison is performed on files unpacked from the XML *rloginDataObjects* which are retrieved by remote login. The files are simply text-diffed and the line numbers that differ are reported in output panes (only the line numbers are reported, not the actual differences themselves).

Exceptions file

The exceptions file is a method for doing fine grain exceptions during network difference that works in addition to the previous methods for including or excluding portions of an object to difference. Currently, exclusion file checking is only implemented a subset of the XML data.

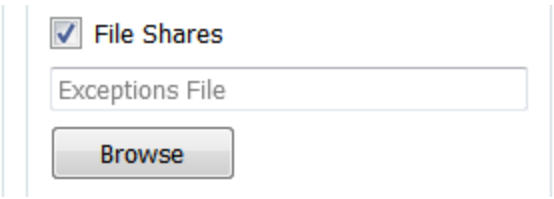


Fig. 4.71: Entering an exceptions file.

An exclusion file is an XML file with the schema shown here.:

```
<exceptionTargets>
 <exceptionTarget target=ipaddress/ipnetwork/hostNamePattern os=osStringPattern>
  <schemaSection name=schemaSectionName>
   <exception attrName1=pattern attrName2=pattern .. attrNameN=pattern />
   <exception attrName1=pattern attrName2=pattern .. attrNameN=pattern />
   ... multiple exceptions ...
   <exception attrName1=pattern attrName2=pattern .. attrNameN=pattern />
  </schemaSection>
 </exceptionTarget>
</exceptionTargets>
```

The exceptionTarget specifies a node(s) for the attached exceptions. It has two attributes:

- *target* : either an ip4/ip6 address, ip4/ip6 network, or a hostname regular expression pattern (the pattern '.*' matches any hostname. If an ip6/ip6 network is specified, then the host will match this target if it has any IP address in that network. This attribute must be specified.
- *os* : specifies a regular expression that is matched against the OS name retrieved by remote login; this attribute is optional. The special value '*' says to ignore OS in terms of matching (this is the same as not specifying the *os* attribute in the *exceptionTarget*).

The *schemaSection* specifies a schema section for the attached exceptions. The *name* attribute specifies the schema section; currently only the following are supported:

- *fileshares*
- *services*
- *iproutes*
- *archive*

Each exception specifies an exception for this schema section. One or more attribute names can be specified in the exception, these names match the attribute names found in the XML for this schema section. The value of the attribute name is either a regular expression pattern that is matched against the value of this attribute, or it is the special pattern "*" which means to ignore this attribute all together when performing a network difference (note that "*" is not a wild card regular expression pattern, ".*" is the wildcard regular expression pattern that will match any attribute value). If an exception match is found, then this is excluded from network difference comparison.

The following will ignore the *startmode* of any service named BITS for a target that has a Windows OS. So, if one target has the BITS service with *startmode* manual, and the other target has a BITS service with *startmode* automatic, this difference is not reported as the *startmode* attribute values are not compared in this case. However, if the BIT service does not exist on one of the two nodes, then this difference is reported.:

```
<exceptionTargets>
 <exceptionTarget target="*" os=".*Windows*.">
  <schemaSection name="services">
   <exception name="BITS" startmode="*"/>
```

```
  </schemaSection>
 </exceptionTarget>
</exceptionTargets>
```

The following will cause the BITS service to be ignored during comparison.:

```
<exceptionTargets>
 <exceptionTarget target="*" os=".*Windows*.">
  <schemaSection name="services">
   <exception name="BITS"/>
  </schemaSection>
 </exceptionTarget>
</exceptionTargets >
```

The following will ignore any *fileshare* named "E$" on windows targets.:

```
<exceptionTargets>
 <exceptionTarget target=* os='.*Windows*.'>
  <schemaSection name='fileshares'>
   <exception name="E$"/>
  </schemaSection>
 </exceptionTarget>
</exceptionTargets>
```

The following ignores several attributes during network difference of *iproutes* for all hosts.:

```
<exceptionTargets>
 <exceptionTarget target=*>
  <schemaSection name='iproutes'>
   <exception metric1="*" metric2="*" metric3="*" metric4="*" metric5='*' ifindex="*"/
↪>
  </schemaSection>
 </exceptionTarget>
</exceptionTargets>
```

The following combines all of these exceptions together.:

```
<exceptionTargets>
 <exceptionTarget target=*>
  <schemaSection name='iproutes'>
   <exception metric1="*" metric2="*" metric3="*" metric4="*" metric5='*' ifindex="*"/
↪>
  </schemaSection>
 </exceptionTarget>
 <exceptionTarget target=* os='.*Windows*.'>
  <schemaSection name='fileshares'>
   <exception name="E$"/>
  </schemaSection>
  <schemaSection name="services">
   <exception name="BITS"/>
  </schemaSection>
 </exceptionTarget>
</exceptionTargets>
```

The following XML snippet shows two exceptions that cause file file comparison to be skipped for retrieved role data files for particular roles.:

```
<schemaSection name="archive">
 <!-- This is neeeded as some elements in this file have timestamps, other data that␣
↪changes run-to-run -->
 <exception rolename="HyperV_Hypervisor" archiveFile="vmdata.config"/>
 <exception rolename="Windows_Server" archiveFile="windowsfeatures.xml"/>
</schemaSection>
```

The *vmdata.config* file has some timestamps that cause simple file comparison to fail. The *windowsfeatures.xml* file has some integer hash values that change each time the file is created, causing simple file comparison to fail.

4.7 *View Devices* Tab Details

This figure shows the *View Devices* tab which is used to quickly peruse node data collected through discovery, port scanning, SNMP, remote login, and topology inference.

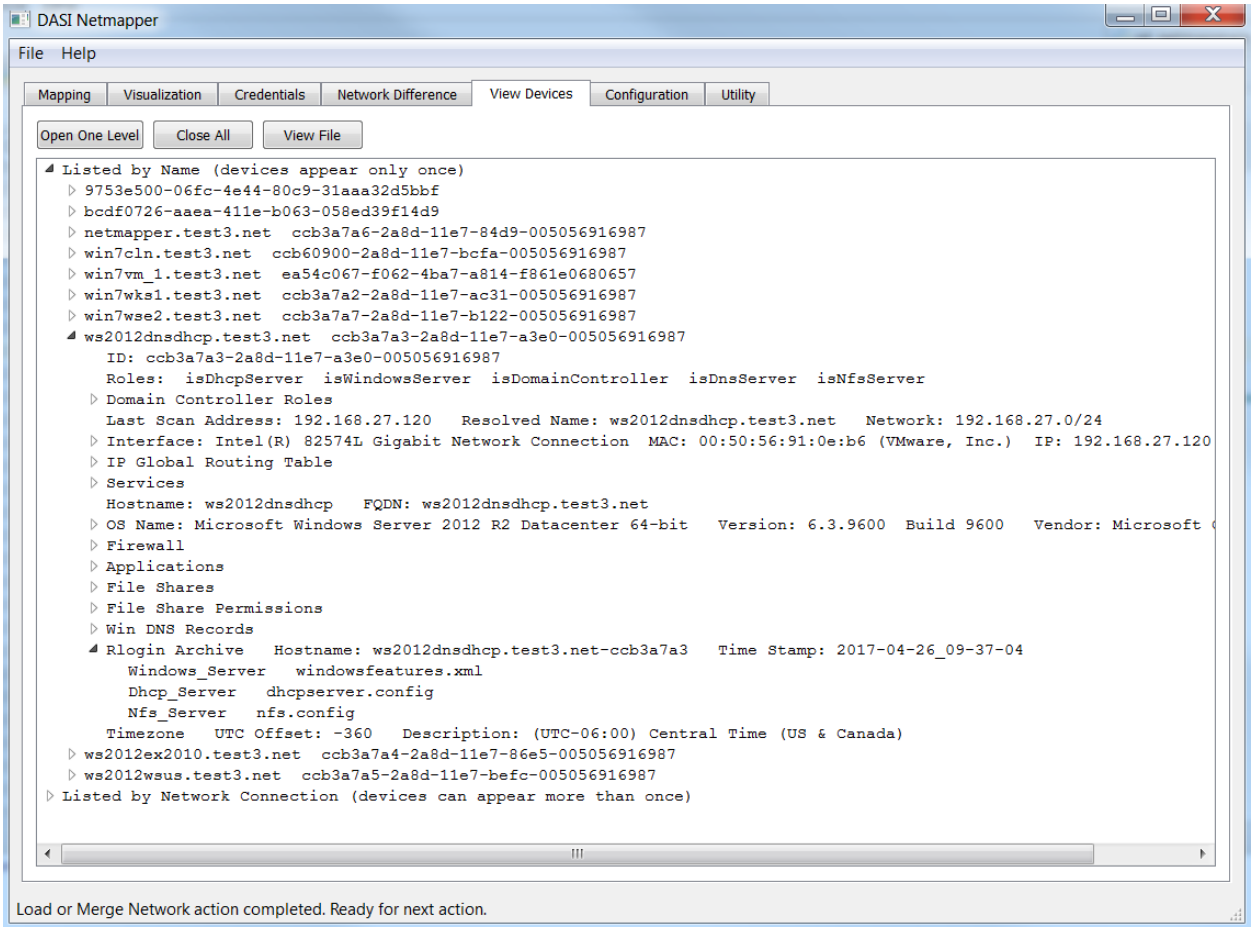


Fig. 4.72: Screenshot of the *View Devices* tab.

Netobjects are listed by name and by network. A *netobject* appears only once in the *Listed By Name* section. The name will be the host name if it is available, else the UUID for the *netobject* is used.

A *netobject* can appear more than once in the *Listed by Network Connection* section if it is attached to more than one network. The network section lists the discovered networks, with nodes attached to that network under each network.

Any nodes for which network membership cannot be determined are grouped in an *Unknown Network* classification.

The *Open One Level* button will open all sub-items under the currently selected item by one level. Multiple clicks on this button will quickly open all sub-items.

The *Close All* button will close all open sub-items under the currently selected item.

The *View File* button opens the currently selected Rlogin Archive file in a text editor. The first time this button is used, the user is prompted to browse to a text editor used for displaying the file. This choice is saved in a configuration option so the user is not prompted for this again.

As an example, clicking on the *View File* button while the *windowsfeatures.xml* item is selected opens the *windowsfeatures.xml* file in a text editor.

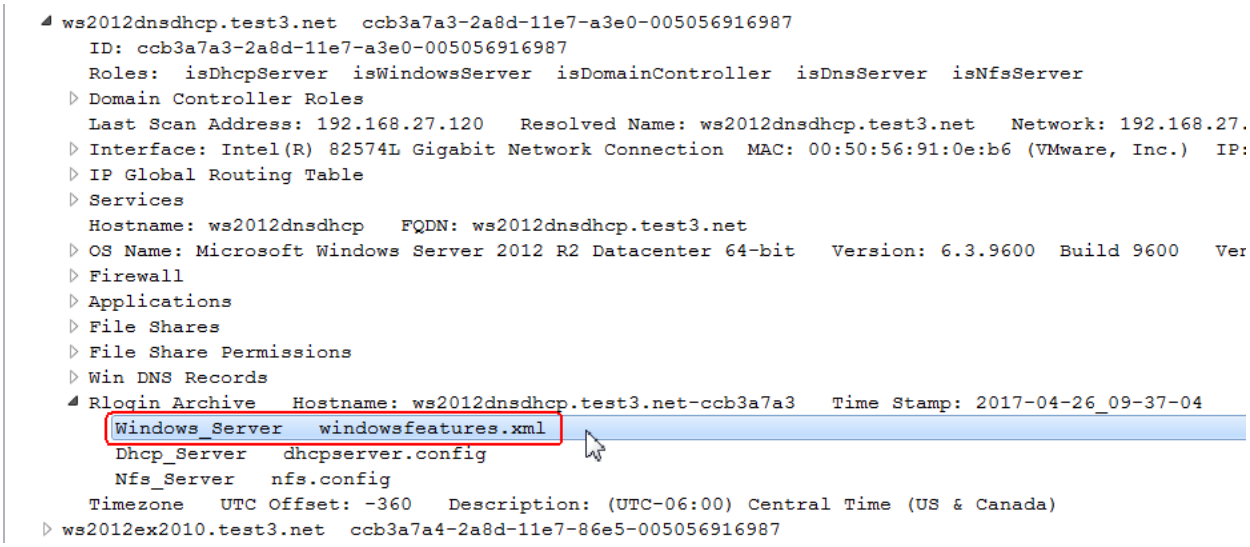


Fig. 4.73: Viewing the *windowsfeatures.xml* file.

If these *netobjects* were loaded from an XML file, then the archive file is in a compressed format in memory. Clicking on the *View File* button expands the archive data for this *netobject* to the same directory that the XML file was loaded from, under a directory named *RloginNodeConfigData*. If this file is edited and saved, and then the *netobjects* are saved back to XML, the XML will contain the changed data. Closing *Netmapper* leaves the expanded data in the *RloginNodeConfigData* directory on disk; this directory must be manually deleted if cleanup is desired.

If these *netobjects* are from a scan and the *netobjects* have not yet been saved to XML, then the archive data is already expanded and is in a temporary directory.

CHAPTER

FIVE

CUSTOM XML TAGGING

During a discover/gather sequence, custom Python code can be dynamically imported/executed after the topology update step if the *Enable Custom XML Tagging* check box is checked on the *Configuration* tab. The intended use of this capability is to add custom XML tags to *netobjects* based on user criteria.

This code is imported from the *<installDir>/plugins/customtag/implementation.py* file. The function *doCustomTagging(database)* is called, where the database parameter is the Python object used to access all *Netmapper* objects/methods. The Python interpreter used internally by *Netmapper* is version 3.4.

There are some examples of this capability provided in this directory. Copy an example to *implementation.py* and enable the custom tagging option in the *Configuration* tab. After *Netmapper* is started, clicking on the *Custom Tag* button will import and execute this code. In the example below, the *hello_world.py* example was copied to the *implementation.py* file.

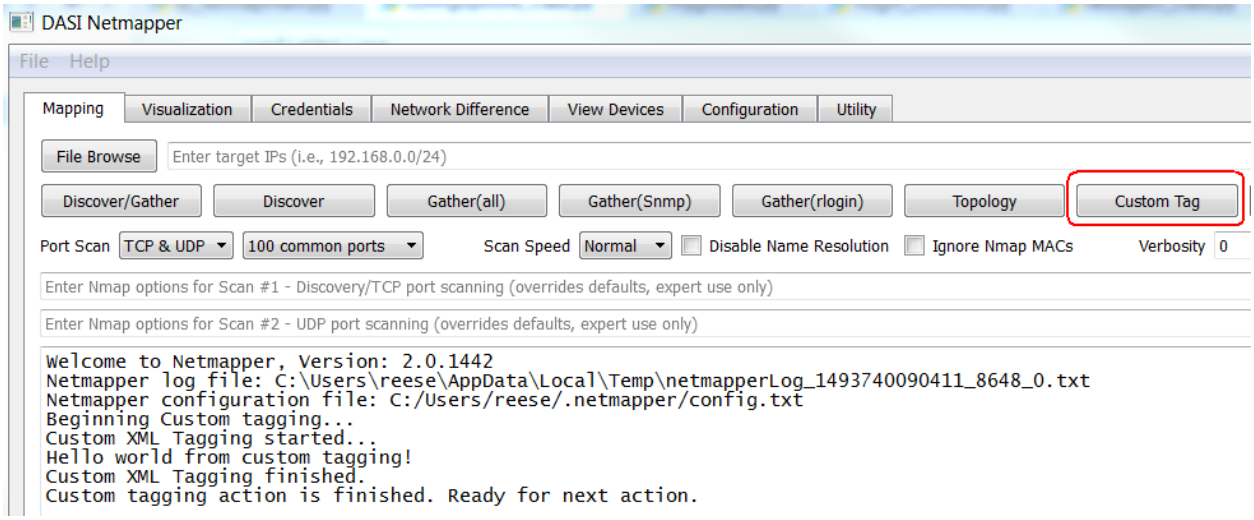


Fig. 5.1: An example of custom tagging.

Other examples require an XML file to be loaded so that there are objects present to be manipulated. See the comments at the top of each example file for more explanation. The *Netmapper* Python API is documented in the appendix.

CHAPTER

SIX

USER CUSTOM QUERY

Custom user queries for a query Engine (i.e., Centos6, Centos7, Ubuntu, etc) can be executed during a node query. These queries are loaded from an XML file with a default path of *<installDir>/user_query.xml*. The default file is shown below:

```
<queryEngines>
 <queryEngine name="centos6">
        <userQuery name="netstat -lnp" useSudo="true"/>
 </queryEngine>
 <queryEngine name="centos7">
        <userQuery name="ss -lnp" useSudo="true"/>
 </queryEngine>
 <queryEngine name="kali">
        <userQuery name="netstat -lnp" useSudo="true"/>
 </queryEngine>
 <queryEngine name="vyos">
        <userQuery name="netstat -lnp" useSudo="true"/>
 </queryEngine>
 <queryEngine name="ubuntu">
        <userQuery name="netstat -lnp" useSudo="true"/>
 </queryEngine>
</queryEngines>
```

The *user_query.xml* file contains multiple *queryEngine* sections (see the section on dynamic query/parse import for a list of Netmapper query engines). The *name* attribute is the id for the query engine, which must match the value returned by the *getId()* method of that query engine. Each *queryEngine* section can contain multiple *userQuery* entries. The *name* attribute is the command to execute, and the *useSudo* attribute must be *true* if sudo is to be used for command execution.

The command output is stored in the *commandoutput* section of the XML, and is displayed in the *Command Output* section of the device in the View Devices tab. The output of user queries is not parsed in any way except to display it in the View Devices tab.

The *Configuration* tab has a browse button that allows the user to browse to a custom query file that is different from the default. User custom query can also be disabled via the *Configuration* tab.

Custom user queries are only supported for stream queries (i.e., SSH); windows wmi/powershell query does not support custom user quries.

CHAPTER

SEVEN

REMOTE LOGIN AND NODE IDENTIFICATION

This section details role assignments and data retrieved during node remote login.

7.1 Windows OS Default Data Retrieval

This table gives a summary of the default data retrieved from each Windows host, method used to retrieve the data, and where the data is stored (XML means the data is stored in XML tags, Archive data means the data is stored in a file in an *rloginDataObject*).

Windows OSes that have been tested for data retrieval are 2008 server, 2012 Server, and Windows 7.

Data Name	Query Method	XML Data or Archive Data?
hostname	WMI	XML (hostname)
OS type, version	WMI	XML (os)
CPU info	WMI	XML (physicalserver, cpus)
Network interfaces	WMI	XML (interface)
Disk information (local and mapped)	WMI and Remote Registry	XML (physicalserver, disks)
Routing table	WMI	XML (iproute)
Applications	WMI	XML (applications)
Services	WMI	XML (services)
Firewall Profiles	Remote Registry	XML (firewall)
Firewall Rules	Remote Registry	XML (firewall)
Routing Enabled	Remote Registry	XML (node attribute)
OS patches	WMI	XML (os, patches)
SMB File shares	WMI	XML (fileshares)
Routing Enabled Determination	WMI	XML (isRouter=True) attribute

All of these queries are done by the Powershell script *get_all_host_info.ps1* in the *code/netmapper* directory.

7.2 Windows OS Role Data Retrieval

The following table gives a summary of the roles and role data retrieved from each Windows host. The second column gives the XML attribute in the *NetObject* for this host that identifies this role. The third column gives the directory name that contains the retrieved role data when the compressed role data is expanded to disk (this value is N/A if the role data is stored directly in the XML). The fourth column describes the role data. If it is a filename, then this is the filename that contains the data when the compressed archive for the role data is expanded to disk. See the XML description for *rloginDataObjects* for how the compressed archive data is expanded to disk.

Windows OSes that have been tested for data retrieval are 2008 server, 2012 Server, and Windows 7.

Role	XML Attribute	Rolename tag in compressed archive	Config Data
Domain Controller	isDomainController	N/A	Domain controller sub-roles stored directly in XML under *domainControllerRole*
Exchange Server*	isExchangeServer	N/A	Exchange server sub-roles stored directly in XML under *exchangeServerRoles*
DNS Server	isDnsServer	N/A	DNS A records stored directly in XML under *windnsrecords*
DHCP Server**	isDhcpServer	Dhcp_Server	File named *dhcpserver.config*
Web Server**	isWebServer	Web_Server	File named *applicationHost.config*
WSUS Server	isWsusServer	Wsus_Server	File named *wsus.config*
NFS Server***	isNfsServer	Nfs_Server	File named *nfs.config*
Windows Server** ****	isWindowsServer	Windows_Server	File named *windowsfeatures.xml*. Result of Powershell command *get-windowsFeature*
Tomcat Server* **	isTomcatServer	Tomcat_Server	Various config files: *web.xml*, *tomcat-users.xml*, *context.xml*, *server.xml*
Snort Server**	isSnortServer	Snort_Server	Various config files: *classification.config*, *snort.config*, *reference.config*, *threshold.conf*

- * - Query will not succeed unless the IP address of this host can be resolved with NSlookup.
- ** - Data retrieval method relies on remote Powershell command execution.
- *** - Different commands used for data retrieval method for Windows 2012 server versus Windows 2008 server. Windows 2008 method requires remote command execution.
- **** - Only works on Windows 2012 Server and above.

All of these queries are done by the Powershell script *get_all_role_info.ps1* in the *code/netmapper* directory.

7.3 Linux OS Default Data Retrieval

The following table summarizes the default data retrieved for Linux OSes. The Linux OSes/distributions tested are CentOS 6, CentOS 7, Ubuntu Workstation, and Kali.

Data Name	OS	XML Data or Archive Data?
hostname	All variants	XML (hostname)
OS type, version	All variants	XML (os)
Arp cache	All variants	XML (interface)
CPU info	All variants	XML (physicalserver, cpus)
Network interfaces	All variants	XML (interface)
Disk information (local and mapped)	All variants	XML (physicalserver, disks)
Routing table	All variants	XML (iproute)
Applications	All variants	XML (applications)
Services	All variants	XML (services)
Processes, linkage of processes to ports	All variants	Not saved to XML, used internally for role determination
Firewall Rules	CentOS 7, Ubuntu	XML (firewall)
Firewall Rules	CentOS 6	*/etc/sysconfig/iptables* file stored as compressed archive
Firewall Rules	Kali-Linux	Output of *iptables-save* command captured and saved as compressed archive
OS patches	All variants	XML (os, patches)
Routing Enabled Determination	All variants	XML (isRouter=True) attribute
Yum client config	CentOS 6, CentOS 7	*/etc/yum.conf* file, and all files/directories under */etc/yum.repos.d*. Data stored in compressed archive under the *'Yum_Configuration'* tag.

On Ubuntu, if an SSH Private Key file is used and the username is not recognized as root, then some data will not be retrieved as the *sudo* command will not be able to be issued without a password.

7.4 Linux OS Role Data Retrieval

The next table gives a summary of the roles and role data retrieved from each Linux host. The second column gives the XML attribute in the *NetObject* for this host that identifies this role. The third column gives the directory name that contains the retrieved role data when the compressed role data is expanded to disk (this value is N/A if the role data is stored directly in the XML). The fourth column describes the role data. If it is a *filename*, then this is the *filename* that contains the data when the compressed archive for the role data is expanded to disk. See the XML description for *rloginDataObjects* for how the compressed archive data is expanded to disk.

Windows OSes that have been tested for role data retrieval are CentOS 6 and CentOS 7 (except for Quagga router, which has only been tested under CentOS 6).

Role	XML Attribute	Rolename tag in compressed archive	Config Data
DNS Server	isDnsServer	Dns_Server	Forward and reverse DNS configuration files are retrieved and stored (*forward.config*, *reverse.config*)
DHCP Server	isDhcpServer	Dhcp_Server	File named *dhcpd.conf*
Web Server	isWebServer	Web_Server	File named *httpd.conf*
Yum Server	isYumServer	Yum_Server	File named *yum.config* and any repo files found in */etc/yum.repos.d*
Samba Server	isSambaServer	Samba_Server	File named *smb.conf*
Quagga Router	isQuaggaRouter	Quagga_Router	Various files under */etc/quagga*, such as */etc/quagga/ripd.conf* - these files are only retrieved if there is an active daemon for the protocol on the node.
Tomcat Server	isTomcatServer	Tomcat_Server	See description for Windows OS.
Snort Server	isSnortServer	Snort_Server	See description for Windows OS.
FTP Server	isFtpServer	Ftp_Server	Only implemented for vsftp, retrieves config file */etc/vsftpd/vsftpd.conf*
Vyos Router	isVyosRouter	Vyos_Router	Config files for Vyos

7.5 Hypervisor Role Data Retrieval

The next table summarizes the role data collected for Hypervisors. VMware Hypervisor information is collected using vSphere API query.

Role	XML Attribute	Rolename tag in compressed archive	Config Data
VMware Hypervisor	isVmwareHypervisor	Vmware_Hypervisor	All *VirtualMachine*, *HostSystem* managed objects pretty printed to a file named *vmware.config*. Some information is parsed from the VirtualMachine managed objects and is used to create a NetObject XML object for each virtual machine. *VirtualApp* managed objects are also retrieved and stored in vmwareVapp XML objects (Vapps are only available from a Vcenter server)
ESXI Server	isEsxiServer	Esxi_Server	The output of the *esxcfg-info* command executed on the ESXI server is stored in a file named *esxi.config*
Vcenter Server	isVcenterServer	N/A	No Vcenter specific configuration retrieved.
Hyper-V Hypervisor* **	isHypervHypervisor	HyperV_Hypervisor	Information on virtual machines hosted are placed in *vmdata.config* file. (Vm data retrieved by remote Powershell execution of *get-vm* command). Configuration data for the host machine stored in *host-system.config*

- * - Query will not succeed unless the IP address of this host can be resolved with NSlookup.
- ** - Data retrieval method relies on remote Powershell command execution.

7.6 Cisco Network Appliance Data Retrieval

For Cisco network appliances, the *netobject* is given a top level XML attribute of isCiscoAppliance in the XML. Most of the data retrieved from the Cisco device is parsed into the various XML sections of a *netobject*. However, some data is not parsed as is saved in compressed archive files. When this data is expanded, it is expanded under a role directory named *Cisco_Appliance* (the DHCP server configuration files are stored in the standard DHCP role directory of Dhcp_Server).

Data	OS	Command	Filename
Running config	IOS	*show running-config*	*running-config.txt*
Running config (all VDCs)	NX-OS	*show running-config vdc-all* (only present in the net object corresponding to the default VDC)	*running-config-all-txt*
Running config (one VDC)	NX-OS	*show running-config* (executed within a VDC)	*running-config.txt*
Spanning Tree	IOS/NX-OS	*show spanning-tree*	*spanning_tree.txt*
Access Lists	IOS/NX-OS	*show access-lists*	*access_lists.txt*
IPv4 interface options*	IOS	*show ip interface*	*ipv4_interface_options.txt*
IPv4 interface options*	NX-OS	*show ip interface vrf all*	*ipv4_interface_options.txt*
IPv6 interface options*	IOS	*show ipv6 interface*	*ipv6_interface_options.txt*
IPv6 interface options*	NX-OS	*show ipv6 interface vrf all*	*ipv6_interface_options.txt*
Firewall config	IOS	*show policy-firewall config*	*firewall_config.txt*
IP inspection	IOS	*show ip inspect all*	*ip_inspect_config_VRF_default.txt*
IP inspection	IOS	*show ip inspect all vrf* vrf-name	*ip_inspect_config_VRF_vrfname.txt*
DHCP Server config**	IOS	*show ip dhcp pool*	*ip_dhcp_pool.txt*
DHCP Server config**	IOS	*show ipv6 dhcp pool*	*ipv6_dhcp_pool.txt*

- * - Some information is parsed, but the entire output of this command is also saved in an archive file.
- ** - Stored in role directory *Dhcp_Server*, *netobject* has *isDhcpServer* XML tag.

CHAPTER

EIGHT

ADDING NEW QUERY/PARSE ENGINES

Netmapper has the capablity to import user code to extend the stream query and parsing of returned data for a network device. This can be used to extend Netmapper query/parse to OSes that are not currently supported. This capability is supported only for stream query (i.e., SSH query).

8.1 Query/Parse Overview

Each supported OS for remote login data retrieval has associated query and parse classes (referred to as query and parse engines). The query engine executes commands that retrieve data. This data is stored in a temporary file, which is handed to the parse engine after all queries have been completed. Each query/parse engine has a static method called *getId()* that returns a string. Query/Parse engines that are to be matched return the same *getId()* string. The *getId()* string is written to the data output file produced by a query engine, and this *getId()* string is used to choose the parse engine for that file.

The currently supported query/parse engines in Netmapper have the following *getId()* strings: *base*, *linuxbase*, *centos6*, *centos7*, *ubuntu*, *vyos*, *kali*, *cisco_ios*, *cisco_ios-xe*, *cisco_ios-xr*, *cisco_nxos*, *cisco_asa*, *windows*, *vmware*, *esxi*, *vcenter*. The *base* query/parse engines are the base query/parse classes for all query/parse engines. The *linuxbase* query/parse engines are the base query/parse classes for all Linux query/parse engines. In this document, query/parse classes are referred to by their *getId()* string, and not by their python class name.

8.2 Dynamic Import

New query/parse classes can be dynamically imported at Netmapper startup. This code is imported from *<installDir>/plugins/queryparse*. Each directory under this path is assumed to be a new query/parse engine. Each query/parse engine directory must contain a file named *implementation.py*. This file must contain a function named *getQueryParseClasses(queryClassList, parseClassList)* that is called at import time, and this function should return *query_class_object*, *parse_class_object*. This function is passed a list of known query classes and parse classes so the function can determine which query/parse class to inherit from (query/parse classes should be chosen based on their returned getId() string). At a minimum, a new query class and parse class must inherit from the *base* query/parse classes, which provide some common methods required for query/parse.

Two example dynamic query/parse engines are provided in the Netmapper installation, one for ESXI query/parse and one for Vcenter application query/parse (this one does not do much). The example ESXI query/parse classes inherits from *linuxbase* query/parse classes as they provide some default query/parse capability for Linux OSes. The example Vcenter application query/parse classes inherit from the *base* query/parse classes.

8.3 Query action (default query):

The *default* query is the first query done of a network object. Based on parsed results, a second query may be done (decided by the parse engine) to retrieve information specfic to roles hosted by the network object.

A *base* query object is created for each network node that is to be queried via SSH. This base query object is passed a list of known query classes (this includes classes that are dynamically imported).

The first action taken by the base query object is to open an SSH connection to the network object in order to verify that the credentials for the network object works (no command is executed). If the connection is successfully opened, the connection is closed, and then the list of known query classes is used to classify the network object.

Each query class must provide a method named *getClassifyCmdOutput* which must open a connection, execute a classification command (i.e, *uname*), close the connection, and then return the command and its accompanying output. After the *getClassifyCmdOutput* method is excuted, an inner loop through all query classes passes the classification command and its output to each query class via a method named *classify*. The *classify* method checks if the command is supported, and if it is, parses the output to determine if the device matches what is expected. If the *classify* method returns *True*, then a query class match has been determined. An object of this class is created, then the *doQuery* method is called to accomplish the default query. The query class list is processed in an undetermined order, so a network object may have several connections made to it during the classification process. It is assumed that a classification command for a particular OS, when executed on a foreign OS, has no side effects on the foreign OS if it is not supported.

8.4 Query action (role query):

Role query is optional, and is decided by the parse engine after the default query data is parsed. If role query is needed, the base query object is called again with a list of roles and the query class to be used (the query class is known because the default query classified the network object). The method *doRoleQueryInit* is called first, which must open the connection and do any other needed initialization. Then the method *doRoleQuery(role)* is called for each role that must be queried.

8.5 Other Comments:

The type of connection to open to an object is decided by the query engine. By default, the base class provides a method that will open a paramiko connection and that uses the paramiko *exec_command* method to execute query commands. However, a query engine may need to use a different query method, such as the paramiko *invoke_shell* method (the base class also supports this query method through Netmiko). The base query class has no specific knowledge about the OSes that it is to query. It simply calls the classification method for each query class until one succeeds. If no classification method is successful, then a message is printed that the connection was opened, but the device could not be classified so no query was performed.

CHAPTER

NINE

XML OUTPUT SCHEMA

This section gives highlights of the XML output schema (a more detailed description is in the appendix). The top-level XML contains three lists of primary objects:

- *netobjects* – this contains the information for the discovered/scanned network devices
- *vmwareVapps* – contains VMware Vapp information retrieved from a Vcenter server
- *rloginDataObjects* – contains configuration information in the form of base-64 encoding zip archives of retrieved files from a host. The retrieved files can be for default information that does not fit well into an XML format or for role-specific configuration information.

The XML schema for *netobjects* and *vmwareVapps* is readable and self-explanatory. While most XML objects are consistent across the different OS types, some have OS-dependent tags/attributes, most notably the firewall rules XML schema.

9.1 *rloginDataObjects*

The *rloginDataObjects* schema requires special mention since it is very different from the other object schema:

```
<rloginDataObjects>
 <!---Rlogin data archives -->
</ rloginDataObjects>
```

The *rloginDataObjects* section is used to store retrieved files from the host in a base-64 encoded ZIP archive format.

Each *rloginDataObject* under the *rloginDataObjects* section has the form:

```
<rloginDataObject hostname=hostnameUniquified timestamp=tsString id=uuidForNetObject>
 <archive rolename=rolename>
  <archiveDataLines>
   <archiveDataLine value=base64string/>
   ...many archiveDataLines...
  </archiveDataLines>
  <archiveFiles>
   <archiveFile name=localFilename orgpath=originalPathOnHost/>
  </archiveFiles>
 </archive>
</rloginDataObject>
```

The attributes for the *rloginDataObject* are:

- *hostname=*hostnameUniquified – The host name that this data was retrieved from; uniquified by adding part of the *netobject*'s* uuid to the end of the hostname.

- *timestamp=tsString* – A timestamp string indicating when this data was retrieved from the remote host.
- *id=uuidForNetObject* – The uuid of the NetObject that owns this data.

Each *rloginDataObject* can have multiple archive objects, each archive object contains the file(s) associated with some role data or default data retrieved from that hose. Each archive object is a base-64 encoded ZIP archive containing one or more files for the role or default data.

The rolename attribute of the archive object is the identifying role tag or default data tag for this data. Under an archive object are multiple *archiveDataLine* objects, each containing a value attribute that is the base64 encoded line for this entry. Reconstructing the ZIP archive for this data requires all value attributes of archiveDataLine objects to be written to a file in the same order as found in the archive object, and then decoding this base64 file back to its binary form – this will be the original ZIP archive.

Following the archive object is an *archiveFiles* object containing one or more *archiveFile* objects that gives the files in the archive.

The attributes for the *archiveFile* object are:

- *name=localFilename* - The name of the file in the archive.
- *orgpath=originalPathOnHost* – The name of the file on the remote host.

Example:

```
<rloginDataObject hostname="centos6ns.test2.net-b0d50d61" timestamp="2015-12-16_16-45-
↪21" id="b0d50d61-a372-11e5-b708-00505691a02e">
 <archive rolename="DHCP_Server">
  <archiveDataLines>
   <archiveDataLine value=
↪"UEsDBBQAAAAAALqFkEfMyeEdygwAAMoMAAALAAAAZGhjcC5jb25maWcjDQojIFNhbXBsZSBjb25m"/>
    ...other lines not included...
  </archiveDataLines>
  <archiveFiles>
   <archiveFile orgpath="/etc/dhcp/dhcpd.conf" name="dhcp.config"/>
  </archiveFiles>
 </archive>
</rloginDataObject>
```

When an XML file with *rloginDataObject* is read into memory, the compressed archives are kept as is. To examine the data in uncompressed form, the *File* > *Expand Rlogin Data* menu choice is used.

This expands the data to the current working directory in the following directory structure:

```
RloginNodeConfigData\tsString\hostnameUniquified1\rolename1\filename1
                             |                    |         \filename2
                             |                    |         \filenameN
                             |                    \rolename2\filename1
                             |                    |         \filename2
                             |                    |         \filenameN
                             |                    \rolenameN\filename1
                             |                    |         \filename2
                             |                    |         \filenameN
                             \hostnameUniquified2\rolename1\filename1
                                                  |         \filename2
                                                  |         \filenameN
                                                  \rolename2\filename1
                                                  |         \filename2
                                                  |         \filenameN
                                                  \rolenameN\filename1
```

```
                                                    \filename2
                                                    \filenameN
```

etc. The above example shows only files retrieved for each role, but the retrieved data can have directory structure as well.

CHAPTER

TEN

AUTOMATED REGRESSION TESTING OF VMWARE VIRTUAL NETWORKS

Netmapper has a command line scripting interface that is currently used by the developers for regression testing of VMware virtual networks. The regression testing feature has been integrated into *Netmapper* for user benefit and is documented here. The regression testing feature allows *Netmapper* to be run from the command line using a script that scans a user-specified Vmware Virtual Application, perform a network difference against a user-provided golden file, and report the results.

The *cli_examples/regression* directory in the *Netmapper* installation directory contains the following files:

- *generic_host_regress.netmap* – a *Netmapper* script that is used on the remote machine to run a regression test.
- *run_regression_test.netmap* – a *Netmapper* script that is used on the local machine to kick off the remote regression tests.
- *regessionTestDataExample.xml* – an example data file that specifies the information needed for a regression test.

A regression test has the following steps:

- The Vcenter host is contacted, and the remote Vapp located. The Vapp is started if necessary.
- The target machine in the Vapp that will run *Netmapper* is identified.
- A user-specified temporary directory on the target VM is created (if it already exists, it is deleted, then recreated so that it will be empty).
- The *Netmapper* installer executable is copied to the target VM and executed. This step is optional if a path to the *Netmapper* installer executable is not provided (in this case, it is assumed that *Netmapper* is already installed on the target VM). The installation directory for *Netmapper* on the remote machine is the default directory. UAC must be turned off on the target VM that is to run the *Netmapper* installer so it does not query the user for permission.
- Data files necessary for the regression test are copied over to the remote directory.
- *Netmapper* is run in command line mode with a script that performs a scan of user-specified networks/hosts. The resulting network data is saved, and a network difference is performed against a provided golden network file.
- Result files (*Netmapper* log, the output network file, golden file, and network difference exceptions file) are copied back to the local machine and placed in a results directory.
- The network difference file is parsed and a result file named *PASS.txt* (no differences) or *FAIL.txt* (differences exist) is created.
- The Vapp is shutdown (this is optional, specified by user).

Multiple regression tests (multiple virtual networks) can be specified in the regression data XML file; these tests are run sequentially.

10.1 Regression Test Data File

The regression test data XML file has the following format:

```
<regressionTestData>
 <crypto attributes for encryption, optional section/>
 <regressionTests common attributes for all tests under this section, this section␣
↪can repeat />
  <regressionTest attributes for one regression test, this section can repeat />
 </regressionTests>
</regressionTestData>
```

The *regressionTests* section contains one or more *regressionTest* sections; the *regressionTests* attributes are applied to all of the child *regressionTest* sections. The attributes on a *regressionTest* section specifies information for one regression test. There can be multiple *regressTest* sections within a *regressionTests* section, and there can be multiple *regressionTests* sections within a regress test data XML file. The following two tables give these attributes.

10.1.1 *regressionTests* XML Attributes

Attribute Name	Description
localWorkingDirectory	*Netmapper* will change to this directory before accessing any local files. Any relative local file pathnames are relative to this directory.
localNetmapperInstallerExe	(Optional attribute) This is the path to the *Netmapper* installer executable on the local machine. If specified, then this is copied to the remote machine and executed before performing the regression test.
localResultDirectory	The local directory that will hold regression test results. This directory will be created if it does not already exist, and individual tests are created as subdirectories within this directory.
localRegressionScript	The path on the local machine to the *Netmapper* script run on the remote machine that performs the scan and network difference. See the example script named *generic_host_regress.netmap*.
remoteWorkingDirectory	The working directory on the remote machine for any remote command execution that copies files or creates directories.
remoteNetmapperExe	The path on the remote machine to the *Netmapper* executable. This must be an absolute path to the *Netmapper* executable, the remote working directory will be set to the directory containing this executable.
remoteTempDirectory	The directory on the remote machine for holding files copied from the local host and for holding results. This directory will be recursively deleted if it exists, and then recreated. It is not deleted at the end of the test.
remoteWinCmdExePath	The full path to the Windows *cmd.exe* executable, typically *"c:\Windows\System32\cmd.exe"*

10.1.2 *regressionTest* XML Attributes

Attribute Name	Description
name	Name of this regression test, must be unique within *regressionTests* - this name is also used as part of the result directory name.
vapp	Name of the VMware vApp.
doc	Doc string for this test.
host	Name of the VM host that will run *Netmapper*.
user	User for *host*. This is clear text, the *userEncrypted* attribute can be used in place of this (see section on encrypted attributes).
password	Password for *user*. This is clear text, the *passwordEncrypted* attribute can be used in place of this.
targetsFileInput	The path to a local file that contains the target IPs, hostname, networks to be scanned by *Netmapper*.
credentialFileInput	The path to a local file that specifies the credentials used for remote login. *This is an optional attribute.*
credentialFilePassword	The passphrase for the credentials file. This is clear text; the *credentialFilePasswordEncrypted* attribute can be used in place of this. *This is an optional attribute*; it is only needed if a *credentialFileInput* parameter is specified.
goldenFileInput	The path to a local file that is the golden network XML to compare against.
exceptionFileInput	The path to a local file that is the exception file to be used in the network diff comparison. *This is an optional attribute.*
resultFileOutput	The name of the output network file for the scan. This path should not have a directory component.
diffFileOutput	The name of the output diffs file to be used for the network difference. This path should not have a directory component.
logFileOutput	The name for the *Netmapper* log file. This path should not have a directory component.

10.2 Running a Regression Test

An example regression data file is given below (this is the *regessionTestDataExample.xml* file found in the directory). This file runs one regression test named *wsus2008* on a VMware Vapp that is also named *wsus2008*:

```
<regressionTestData>
 <!-- Crypto section is optional, only needed if use encrypted attributes -->
 <crypto salt="ce:fc:8a:e0:a2:a6:7b:44:ff:4a:fa:a6:4e:47:c3:20" digest=
↪"d3:90:b1:5a:e0:0d:f2:dc:1d:6f:9f:fb:12:8d:96:8b:22:7b:1f:5a:68:2a:ab:c6:fd:3e:0d:1f:65:9a:08:68
↪" />
 <regressionTests remoteWorkingDirectory="C:\"
  remoteNetmapperExe="c:\DasiNetmapper\netmapper.exe"
  remoteTempDirectory="c:\NetmapperRegressData"
  remoteWinCmdExePath="c:\Windows\System32\cmd.exe"
  localWorkingDirectory=" c:\DasiNetmapper\"
  localNetmapperInstallerExe=".\installer\netmapper.exe"
  localResultDirectory="c:\regression_results"
  localRegressionScript="cli_examples\regression\generic_host_regress.netmap">
  <!-- Encrypted atttributes supported: userEncrypted, passwordEncrypted,␣
↪credentialFilePasswordEncrypted -->
  <!-- Cleartext versions supported: user, password, credentialFilePassword -->
  <regressionTest
   name="wsus2008"
   vapp="wsus2008"
```

```
   doc="Test on wsus2008 network"
   host="win7cln4"
   user="test1.net\administrator"
   passwordEncrypted="bc:82:5b:4b:6c:8d:97:a1:b5:54:1f:0a:0d:3e:b1:f5"
   targetsFileInput="scripts\wsus2008_esxi_hostlist.txt"
   credentialFileInput="scripts\wsus2008_esxi_credentials.xml"
   credentialFilePasswordEncrypted="8a:89:9c:b8:48:73:8b:10:d9:38:ce:c0:64:88:bd:ce"
   goldenFileInput="scans\wsus2008_esxi_network_golden.xml"
   exceptionFileInput="scripts\wsus2008_diff_exceptions.xml"
   resultFileOutput="wsus2008_esxi_network.xml"
   diffFileOutput="wsus2008_esxi_network_diffs.xml"
   logFileOutput="wsus2008_esxi.log"
  />
 </regressionTests>
</regressionTestData>
```

The script *cli_examples/regression/run_regression_test.netmap* is used on the local machine to run the test. This script requires the following parameters:

- DF:<regression data file>
- PP:<passphrase for encrypted attributes in the regression data file>
- HOST:<vcenter host to has the Vapps>
- USER:<vcenter username>
- PW:<vcenter user password>
- TEST: <all|<test name>>
- STOP: <false/no | true/yes> – use true or yes to stop the Vapp after the run
- SKIPINSTALL: <false/no | true/yes> – use true or yes to skip netmapper installation step (useful if running multiple tests on same VM)

Assuming that Powershell window is open, and the current working directory is *C:\DasiNetmapper*, then the following command executes this script:

```
.\netmapper.exe -c cli_examples/regression/run_regression_test.netmap DF:cli_examples/
↪regression regressionTestDataExample.xml PP:msu#1bulldogs HOST:<a vcenter host>␣
↪USER:<a Vcenter username> PW:<vcenter password> TEST:<all | a particular vapp, i.e,␣
↪wsus2008> STOP:no SKIPINSTALL:no
```

Parameters are passed to a *Netmapper* script in the form of *PARM_NAME:PARM_VALUE.*

The *cli_examples/regression/run_regression_test.netmap* script contains the following lines:

```
##parameters
# DF:<regression data file>
# PP:<passphrase>
# HOST:<vcenter host>
# USER:<vcenter username>
# PW:<center user password>
# TEST: <all|<test name>>
# STOP: <false/no | true/yes>
# SKIPINSTALL: <false/no | true/yes>

run-vmware-regression  DF PP HOST USER PW TEST STOP SKIPINSTALL
```

It has a single command, *run-vmware-regression*, that accepts the DF, PP, HOST, USER, PW, TEST, STOP, SKIPINSTALL parameters.

Once the regression test is finished, the results are placed in the *localResultDirectory* (*C:\regression_results*), in a subdirectory named *wsus2008* appended to a timestamp. In this directory will either be a file named PASSED.txt (empty), or FAILED.txt (contains the network differences produced by the *Netmapper* network difference after the scan). This directory also includes a downloads subdirectory that contains:

- the log file of the *Netmapper* run on the remote Vapp
- the network XML produced by the *Netmapper* run and golden XML file that this was compared against
- a file containing the network differences

10.3 Encrypted Attributes

Some attributes in a *regressionTest* section can be encrypted (i.e., use *passwordEncrypted* instead of *password*). The *Utility* tab, *Crypto* controls support generation of these encrypted attributes.

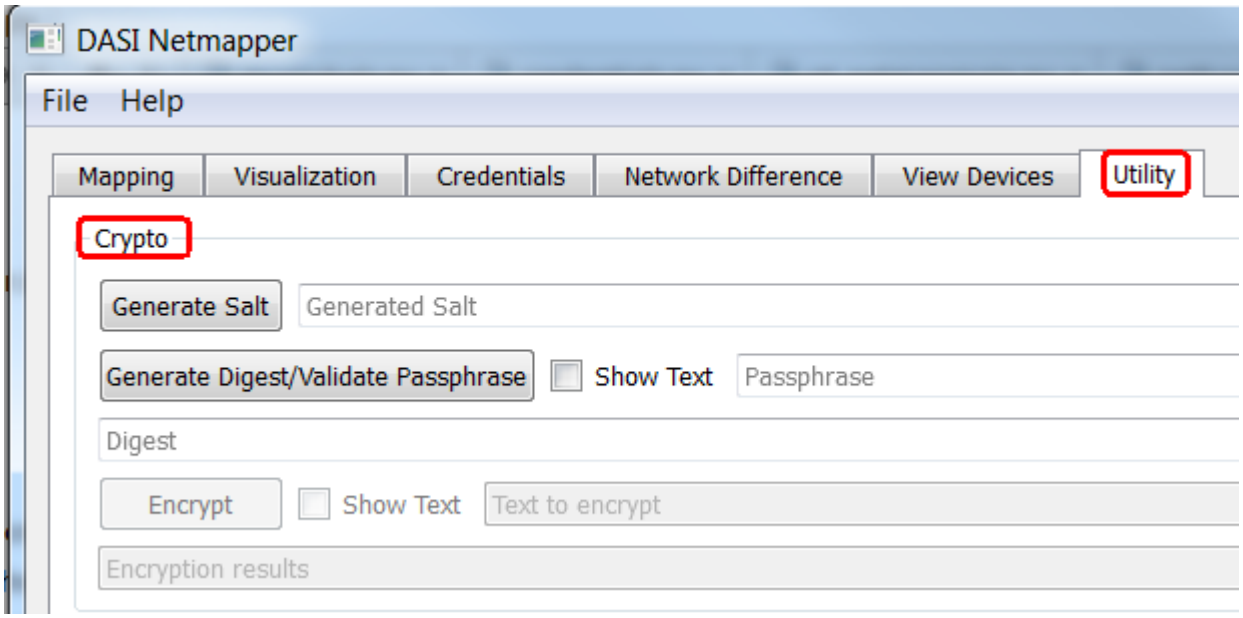


Fig. 10.1: Screenshot of the *Crypto* controls on the *Utility* tab.

10.3.1 Creating a new regression data XML file

If a new regression data XML file is being created that requires encrypted attributes, follow these steps:

- Click on the *Generate Salt* button (generates random salt value).
- Type in a passphrase in the *Passphrase* line edit.
- Click on the *Generate Digest* button (generates digest from the salt and passphrase).

After this, copy the generated salt and digest hex strings and put them into the new regression data XML file as shown below.

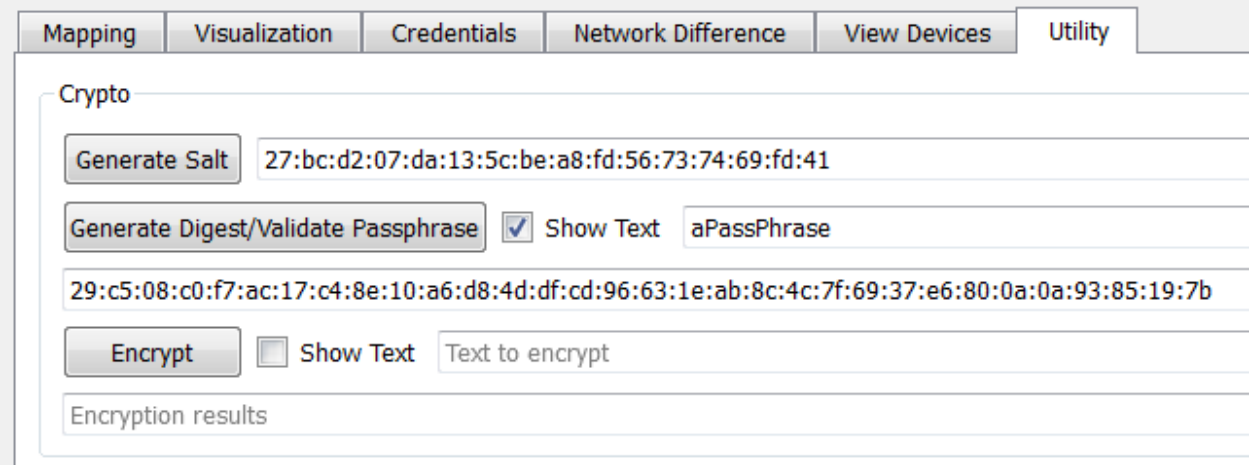


Fig. 10.2: Entering the generated salt and digest hex strings into the new regression data XML.

```
<regressionTestData>
 <!-- Crypto section is optional, only needed if use encrypted attributes -->
 <crypto salt="27:bc:d2:07:da:13:5c:be:a8:fd:56:73:74:69:fd:41"
 digest="29:c5:08:c0:f7:ac:17:c4:8e:10:a6:d8:4d:df:cd:96:63:1e:ab:8c:4c:7f:69:37:e6:80:0a:0a:93:85:19:7b"
 />
```

Fig. 10.3: The XML is now updated with the new salt and digest hex strings.

The salt is a randomly generated value that is mixed with the provided passphrase to produce the digest that is stored in the XML. When encryption is required, the salt is mixed with the passphrase in a different manner to create a digest used for encryption. The digest in the XML file is only used for validating the passphrase, which is done when the file is loaded. Validation creates a new digest using the salt value in the XML file along with a provided passphrase. If the new digest matches the digest in the XML file, then the passphrase is validated.

Once a salt and digest has been created, the *Encrypt* button will be active. Arbitrary text can be placed in the *Text* to encrypt line edit control, which is then encrypted by the *Encrypt* button. The hex string representing the encrypted text is generated in the *Encryption Results* line edit. This hex string is then used as the value of the encrypted attribute (copy this hex string into your XML file).

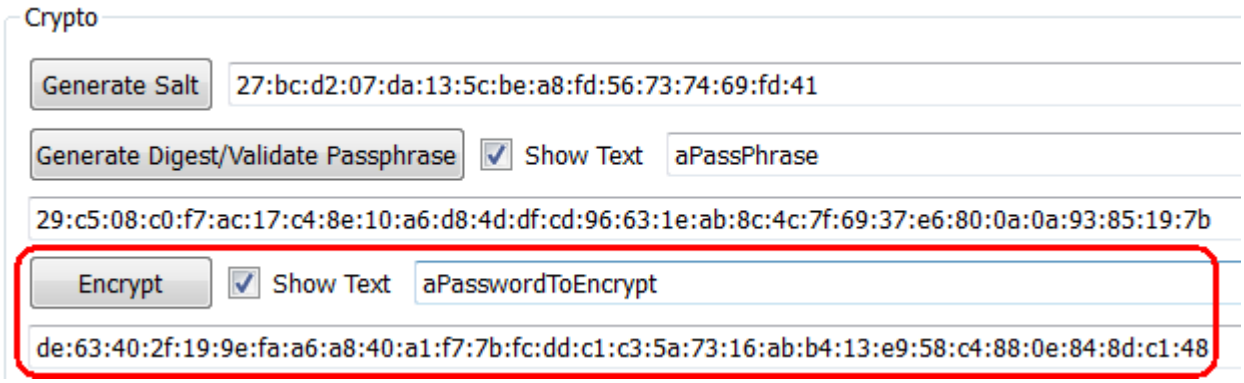


Fig. 10.4: Encrypting an attribute.

10.3.2 Editing a previously created regression data XML file

To add more encrypted attributes to a previously created regression data XML file, perform the following steps:

- Start *Netmapper*, and navigate to the *Utility* tab.
- Open the regression data XML file in a text editor.
- Copy the salt, digest values in the XML file to their appropriate fields in the *Crypto* controls.
- Enter the passphrase originally used for this XML file into the *Passphrase* line edit control.
- Click on the *Generate Digest/Validate Passphrase* button. A new digest is created from this passphrase and the salt, and then checked against the entered digest. If the two digests match, then the passphrase is validated, and the *Encrypt* controls are enabled.
- Use the *Encrypt* controls to generate new encrypted attributes for the regression data XML file.

CHAPTER

ELEVEN

APPENDIX

11.1 Initialization File Key Words and File Format

The default initialization file read by *Netmapper* on startup resides in *~/.netmapper/config.txt*. Configuration options are formatted as *keyword|option* (the | symbol is the separator). A # symbol marks a comment line.

The first part of the file contains static configuration options that are editable by the user. The table below shows the currently supported static configuration options.

Keyword|Default	Description
disableAllDataRetrieval|hostname	Disables all data retrieval for host (default and role data)
disableRoleDataRetrieval|hostname|rolename	Disables Role Data Retrieval for host, rolename
disableRoleDataStorage|hostname|rolename	Disables Role Data Storage for host, rolename

The second part of the file contains configuration options set by the GUI. These options are updated when the GUI exits, so any user edits to these options are lost. All of these options start with *DYNAMIC_*.

11.1.1 Controlling role data retrieval/storage

The *disableAllDataRetrieval*, *disableRoleDataRetrieval*, *disableRoleDataStorage*, options are intended to give the user full control over role data retrieval and storage. This may be needed if the role data that is retrieved is large, and the user only wishes to disable retrieving or storing the data.

Using *disableRoleDataRetrieval* can speed execution time by skipping role data retrieval. Using *disableAllDataRetrieval* will skip all data retrieval for a host – this is useful if using a network login credential and it is desired to skip one or more particular hosts.

When role data storage is disabled, role data is still retrieved but is not stored in the XML file as a compressed archived. As an example, a VmWare EXSI server can return a large configuration file, parts of which are parsed to populate specific fields in the XML, and the entire file is saved as a compressed archive in the XML. The user may want disable storage of the compressed data, but still enable retrieval of the data for the parts that are parsed into XML.

The Host field can be '*', which disables for all hosts. The rolename field can be '*', which disables for all roles. The Host field can be an IP address or hostname.

Examples

The following disables data retrieval from all hosts that respond to VMware API calls (ESXI or Vcenter):

```
disableRoleDataRetrieval|*|Vmware_Hypervisor
```

The following disables data storage from all hosts that respond to VMware API calls - VMs will be populated in XML, but config data not saved in XML:

```
disableRoleDataRetrieval|*|Vmware_Hypervisor
```

The following disables data retrieval for WSUS roles from one host:

```
disableRoleDataRetrieval|myWsusHost|Wsus_Server
```

Valid role names are:

- Esxi_Server - (doc: ESXI Server)
- Vmware_Hypervisor - (doc: Responds to VMWare API Calls)
- Yum_Server - (doc: Yum Server)
- Wsus_Server - (doc: Wsus Server)
- Nfs_Server - (doc: NFS Server)
- Web_Server - (doc: Web Server)
- Domain_Controller - (doc: Domain Controller)
- Dns_Server - (doc: DNS Server)
- Dhcp_Server - (doc: DHCP Server)
- Samba_Server - (doc: Samba Server)
- Vmware_Hypervisor - (doc: VCenter Server)
- Exchange_Server - (doc: Exchange Server)
- HyperV_Hypervisor - (doc: HyperV Server, responds to HyperV API Calls)

11.2 Detailed XML Output Schema

The XML output format for network information has been iterated on by MSU and Circadence.

This is the current output format of the DASI *Netmapper* Tool as of January 2016:

```
<Netmapper>
 <netobjects>
  <netobject various attributes  id=unique_id >
       <!-- netobject content -->
  </netobject>
 </netobjects>
 <vmwareVapps>
  <!--Vmware Vapp contents -->
 </vmwareVapps>
 <rloginDataObjects>
  <!--Rlogin data archives -->
 </ rloginDataObjects>
</Netmapper>
```

The top-level XML contains three lists of primary objects:

- netobjects - this contains the information for the discovered/scanned network devices
- vmwareVapps - contains VMware Vapp information retrieved from a Vcenter server

- rloginDataObjects – contains configuration information in the form of base-64 encoding zip archives of retrieved files from a host. The retrieved files can be for default information that does not fit well into an XML format or for role-specific configuration information.

11.2.1 *netobject*

Each *netobject* is a discovered network object by an NMAP scan and has the following general format (ordering of the fields/attributes should not be assumed):

```
<netobject various attributes  id=objectId >
 <lastscanaddress network=networkspec iptype=iptype ipaddr=ipaddr />
 <interfaces>
  <interface macaddress=macaddress_spec macvendor=vendorIdentifierString
   <!-- interface content -- >
  </interface>
 </interfaces>
 <routingConfigs>
  <!-- routingConfig objects -->
 </routingConfigs>
 <iproutes>
  <!-- iproute objects -- >
 </iproutes>
 <osguess various attributes >
 <transient>
  <addresstable>
   <!--addrtblentry objects -- >
  </addresstable>
  <bridgetable macaddress=macaddress>
   <!--bridgetblentry objects -- >
  </bridgetable >
 </transient>
 <domainControllerRoles variousAttributesForSubroles>
 <exchangeServerRoles variousAttributesForSubroles>
 <os name="" version="" manufacturer="">
  <patches>
   <!--patch objects -- >
   <patch id="1" name="value"/>
  </patches>
 </os>
 <physicalserver>
  <memory ram=size vmem=size>
  <cpus>
   <!--cpu objects -- >
   <cpu name="">
  </cpus>
  <disks>
   <!--disk objects -- >
   <disk name=name size=size DriveType=drivetype ProviderName=remotepath >
  </disks>
 </physicalserver>
 <applications>
  <!--application objects -- >
  <application name=name vendor=vendor version=version >
 </applications>
 <services>
  <!--service objects -- >
  <service name="" status="stopped|running" startmode="auto|manual">
```

```
 </services>
 <hostname fqdn=fqdn_name name=hostname/>
 <firewall>
  <profiles>
   <!--profile objects for Windows OS>
  </profiles>
  <rules>
   <!--rule objects - Tags are OS-specific other than name>
  </rules>
 <firewall/>
 <windnsrecords>
  <!--arecord objects for Windows DNS server>
 </windnsrecords >
 <fileshares>
  <!--fileshare objects for shared disks>
 </fileshares>
</netobject>
```

The attributes attached to a *netobject* are as follows. Only the id attribute is guaranteed to be present.

- id=objectId - an internally generated UUID string uniquely identifying this object. This string can be used a reference pointer by other *netobjects*. For virtual machines retrieved from VMware API calls, this UUID is the UUID generated by VMware.

Network device attributes (if present, will have a value of True):

- isRouter=”True” - indicates the device is forwarding packets.
- isBridge=”True” - indicates the device returned SNMP information that it is performing bridge operations (i.e, it is a switch).
- isPrinter=”True” - indicates the device performs print functions

Role specific attributes (if present, will have a value of True):

- isDnsServer=”True” - this device is a DNS server
- isDomainController=”True” - this device is a Windows Domain Controller
- isExchangeServer =”True” – this device is a Windows Exchange Server
- isDhcpServer =”True” – this device is a DHCP Server
- isWebServer =”True” – this device is a Web Server
- isWsusServer =”True” – this device is a WSUS Server
- isNfsServer =”True” – this device is a NFS Server
- isSambaServer =”True” – this device is a Samba Server
- isYumServer =”True” – this device is a Samba Server
- isVmwareHypervisor =”True” – this device responds to VMware API calls
- isEsxiServer =”True” – this device is a VMware ESXI server
- isVcenterServer =”True” – this device is a VMware Vcenter server
- isHypervHypervisor =”True” – this device is a Windows Hyper_V Hypervisor (unimplemented, reserved)

Miscellaneous attributes:

- isVirtualMachine =”True” – this device is a virtual machine
- vmName=name – name of this virtual machine as identified by API call (not the hostname)

- scanType= ”vmwareAPI”| ”network” – identifies how this information was retrieved. ‘vmwareAPI’ means data was retrieved by VMware API calls, ‘network’ means by some sort of network query (SNMP or Nmap). This is used when merging nodes produced by two different methods.
- loginSuccess =”True” – indicates remote login attempt to this node was successful.

11.2.2 *lastscanaddress*

```
<lastscanaddress network=networkspec iptype=iptype ipaddr=ipaddr />
```

Example:

```
<lastscanaddress network="192.168.27.0/24" iptype="ipv4" resolvedname="win7ser08base"␣
↪ipaddress="192.168.27.130"/>
```

The *lastscanaddress* is the last IP address that the *netobject* responded to during a network scan. The network attribute is present if *Netmapper* has discovered what network this *netobject* belongs to and is formatted in CIDR notation. The *iptype* attribute is either ipv4 or ipv6. The *ipaddress* attribute is the IP address discovered by the scan. The *resolvedname* is the resolved name of the IP address.

11.2.3 *customTags*

```
<customTags user-defined attributes >
```

This section is populated by user-defined attributes created by dynamically imported/executed user code. See the section on *Custom Tagging* for more details.

11.2.4 *interface*

The *interfaces* contains a list of *interface* objects for this *netobject*. A *netobject* will have at least one *interface* object, with a value of ‘0:0:0:0:0:0’ if the actual MAC address could not be discovered. A *netobject* may have multiple interface objects.

Each *interface* object has the following general format:

```
<interface macaddress=macaddress_spec macvendor=vendorIdentifierString isVnic=True␣
↪vnet=name vffName=name linkLocalAddress=ip6LLaddress>
 <ipaddresses >
  <!-- list of ipaddress objects, may not be present -- >
 </ipaddresses>
 <ports iptype=iptype ipaddr=addr >
  <!-- list of port objects, may not be present -- >
 </ports>
 <phyports>
  <!-- list of phyport objects, may not be present -- >
 </phyports>
 <routingProtocols>
  <!-- list of routingProtocol objects, may not be present -- >
 </routingProtocols>
 <dnsservers>
  <dnsserver iptype=iptype ipaddr=addr resolvedname=name dnstype=primary|secondary>
 </dnsserver>
 <l3connections>
  <l3connection various attributes />
```

```
 </l3connections>
 <l2connections>
  <!-- list of l2connection objects, may not be present -- >
 </l2connections>
</interface>
```

Attributes for the interface object are:

- macaddress=macaddress – mac address of as ascii hex bytes separated by ':'
- macvendor=vendorName – vendor assigned to this mac address
- isVnic='True' – if present, value is True and indicates that this is a virtual NIC
- vnet=vnetName – if present, indicates this interface is connected to a virtual net with name vnetName.
- vrfName=name - if present, indicates this interface is associated with a VRF instance.
- linkLocalAddress=ip6LLaddress - if present, is the IPv6 link local address assigned to this interface

An *interface* object can contain one or more *ipaddress*, *routingProtocols*, *phyport*, *port*, *dnsserver*, *l3connection*, and *l2connection* objects. It can also contain a *dhcpserver* object.

11.2.5 *ipaddress*

An *ipaddress* object has the following format:

```
<ipaddress network=networkspec iptype=iptype ipaddr=ipaddr ifindex=integer␣
↪dhcpServer=ipaddr/>
```

Example:

```
<ipaddress network="192.168.27.0/24" iptype="ipv4" resolvedname="win7ser08base"␣
↪ipaddress="192.168.27.130" ifindex="16" />
```

An *interface* may have multiple *ipaddress* objects or it may have none. An *ipaddress* object is present only if has been verified to map to this mac address, either via SNMP query or by extraction from a local network scan. The *ifindex* field ties this address to a physical port, and it is only present if this information has been discovered. If there are no *ipaddress* objects, the *lastscanaddress* is used as the address for this *netobject*. The *dhcpServer* field is present if this address was assigned by a DHCP server and is the ip address of the server.

11.2.6 *port*

```
<port protocol=protocol reason=reason portnum=portnum state=state name=name/>
```

Example:

```
<ports iptype="ipv4" ipaddress="192.168.27.133">
 <port protocol="udp" reason="no-response" portnum="7" state="open|filtered" name=
↪"echo"/>
 <port protocol="tcp" reason="syn-ack" portnum="53" state="open" name="domain"/>
</ports>
```

The ports revealed by the NMAP TCP and UDP scans are specified in the *ports* section, which contains a list of *port* objects. The attributes attached to *ports* indicate the IP address used for this scan. The *port* object contains the port scan information for a *port* as returned by NMAP. Only ports with state of *open* or *open|filtered* are returned (the latter

state means that NMAP cannot determine if the port is open or if packets intended for the port are being filtered). The other attributes are documented in the NMAP DTD. Currently, *Netmapper* only scans for UDP and TCP ports.

11.2.7 *phyport*

```
<phyports>
 <phyport  various attributes>
</phyports>
```

Example:

```
<phyports>
 <phyport vlanForUntagged="1" vlanIngressFiltering="False" vlans="1" portnum="60"␣
↪name="lag 7" vlansXmitUntagged="1,2" ifindex="60" vlanAcceptableFrameTypes="admitall
↪"/>
 <phyport vlanForUntagged="1" vlanIngressFiltering="False" vlans="1" portnum="61"␣
↪name="lag 8" vlansXmitUntagged="1,2" ifindex="61" vlanAcceptableFrameTypes="admitall
↪"/>
</phyports>
```

The *phyport* objects are physical ports discovered via SNMP/Remote Login for this *netobject*. The attributes for a *phyport* are as follows. Not all attributes are guaranteed to be present, it depends on what SNMP tables the switch/router supports or what is discovered during remote login.

- portnum=portnum – The port number for this port
- name – port name
- ifindex=interface index – The interface index for this port – often the same as the port number but does not have to be. This *ifindex* can referenced by other objects (*ipaddress*, *l3connection*, and *l2connection* objects).
- vlans=vlan list – A list of decimal numbers separated by commas that are the VLANs egressed on this port.
- vlansBlocked=vlan list – A list of decimal numbers separated by commas that are the VLANs forbidden to egress on this port.
- vlansXmitUntagged=vlan list – A list of decimal numbers separated by commas that are the VLANs to transmit as untagged on this port.
- vlanForUntagged=vlan list – Vlan Id that is assigned to untagged packets on this port.
- vlanAcceptableFrameTypes=" admitAll"|" admitOnlyVlanTagged" – Vlan policy for this port.
- vlanIngressFiltering="True"|"False"– if set, will be boolean – 'False' means to accept all frames.

11.2.8 *routingProtocols*

```
<routingProtocols>
 <!-- list of routingProtocol objects, may not be present -- >
</routingProtocols>
```

Example:

```
<routingProtocols>
 <routingProtocol iptype="ipv6" protocol="eigrp" neighbor="FE80::F816:3EFF:FE00:3"␣
↪autonomousSystem="600" name="eigrp_ipv6"/>
```

```
 <routingProtocol iptype="ipv4" protocol="is-is" neighbor="10.0.2.2" name="is-is_ipv4
↪"/>
</routingProtocols>
```

This identifies the routing protocol(s) found to be associated with this interface. Each *routingProtocol* object has attributes identifying the protocol and its aspects. Common attributes are:

- name – a unique key for this protocol object
- protocol – generic name of the protocol
- neighbor – an ip address for a neighbor of the same protocol
- iptype – type of ip address for the neighbor

The *routingConfigs* section found in *netobject* gives more details about the configuration of the particular protocol.

11.2.9 *dnsservers*

```
<dnsservers>
 <dnsserver iptype=iptype ipaddr=addr resolvedname=name dnstype=primary|secondary>
</dnsserver>
```

Example:

```
<dnsservers>
 <dnsserver iptype="ipv4" ipaddress="192.168.27.120" dnstype="primary"/>
</dnsservers>
```

The *dnsserver* object gives the ip address of the DNS servers for this interface. The *dnstype* attribute has values of either 'primary' or 'secondary'.

11.2.10 *dhcpserver*

```
<dhcpserver iptype=iptype ipaddr=addr resolvedname=name >
```

The *dhcpserver* object gives the dhcp server for this interface object.

11.2.11 *l3connection*

```
<l3connection dstnetobject=objectId iptype=iptype srcaddress=srcIpaddress␣
↪srcifindex=integer dstaddress=dstIpaddress dstifindex=integer network=networkspec />
```

Example:

```
<l3connections>
 <l3connection network="192.168.0.0/24" iptype="ipv4" dstifindex="2" dstaddress="192.
↪168.0.50" srcaddress="192.168.0.104" dstnetobject="906583cf-8e09-11e5-942c-
↪005056c0000a"/>
</l3connections>
```

An *interface* object may have multiple *l3connection* objects, contained in an *l3connections* list.

The *l3connection* object connects this *netobject* at the L3 level to another object. If object A connects to object B, only one of the objects (A or B) will have an *l3connection* object (this is to reduce the amount of redundancy in the XML file, and the link is obviously bi-directional).

The *dstnetobject* attribute represents the destination connected object, specified by the unique string contained in the id attribute of the *netobject*. The *srcaddress* and *dstaddress* are the source and destination IP addresses of the connected nodes. The *iptype* attribute specifies the type of these addresses while the network attribute specifies the shared network of this connection. The *srcifindex* and *dstifindex* are the *ifindex* fields (interface index) of the source *phyport* and destination *phyport* (one or both of these may be missing depending upon discovered information).

11.2.12 *l2connection*

```
<l2connection srcphyport=integer srcphyportname=string srcaddress=srcIpaddress␣
↪dstnetobject=objectId dstmacaddress=macaddress_spec dstphyport=integer␣
↪dstphyportname=string dstaddress=dstIpaddress dstnetobject=objectId />
```

Example:

```
<l2connections>
 <l2connection srcphyport="10148 macaddress="e8:4:62:5a:15:af" dstaddress="10.92.3.85
↪" netobject="b57bcd40-4819-11e5-8051-005056c0000a" dstphyport="10147" srcaddress=
↪"10.92.3.87"/>
</l2connections>
```

An *interface* object may have multiple *l2connection* objects (if the same mac address is mapped to multiple ports, like on a switch), specified in the *l2connections* list.

This information is returned by SNMP/remote login. The *srcphyport/dstphyportname* and *dstphyport/dstphyportname* fields are the port number/port name values from the source and destination physical ports. Either the source port number or port name is guaranteed to be present. The destination port number and destination port name may be both absent, or only one may be present. The *dstmacaddress* is the destination mac address and will always be present. The L2 connection object is attached to the interface object that is the source mac. The *srcaddress* and *dstaddress* fields are presented for readability purposes and represent IP addresses contained at the source and destination nodes (these will be the addresses that the nodes were scanned or discovered at). The L2 connection is defined by the source macaddress/source port number to destination macaddress/destination port number. The *dstnetobject* attribute represents the destination connected object, specified by the unique string contained in the id attribute of the *netobject*.

11.2.13 *routingConfigs*

```
<routingConfigs vrfName=name >
 <!-- routingConfig objects -->
</routingConfigs>

<routingConfig   various attributes >
 <neighbors>
  <!-- neighbor objects -->
 </neighbors>
</routingConfig>
```

Example:

```
<routingConfigs>
 <routingConfig proto="ospf" routerId="192.168.0.5" area="10" name="ospf_10_ipv4">
  <neighbors>
   <neighbor iptype="ipv4" routerId="192.168.0.6" ifname="GigabitEthernet0/2"␣
↪ipaddress="10.0.3.1" area="10"/>
```

```
  </neighbors>
 </routingConfig>
 <routingConfig proto="ospf" routerId="192.168.0.5" area="10" name="ospf_10_ipv6">
  <neighbors>
   <neighbor iptype="ipv6" routerId="192.168.0.6" ifname="GigabitEthernet0/2"␣
↪ipaddress="FE80::F816:3EFF:FE1F:4838" area="10"/>
  </neighbors>
 </routingConfig>
 <routingConfig proto="bgp" routerId="192.168.0.5" autonomousSystem="1" name="bgp">
  <neighbors>
   <neighbor iptype="ipv4" routerId="192.168.0.3" ipaddress="10.0.2.2"␣
↪autonomousSystem="2"/>
  </neighbors>
 </routingConfig>
</routingConfigs>
```

The *routingConfigs* section contains multiple *routingConfig* objects. There can be more than one *routingConfigs* section in a *netobject*; they are distinguished by a *vrfName* attribute (VRF instance name). If no *vrfName* attribute is present, then this is the routing config for the default or global instance.

Each *routingConfig* option contains multiple attributes for this configuration. It may also contain a *neighbors* section that lists *neighbors* or peers of the same protocol for this *netobject*.

Routing configuration information is discovered during direct login of a network device. The following protocols are supported:

- ospf – attributes are area, routerId. May have an optional neighbors section.
- rip – no attributes
- eigrp – attributes are autonomousSystem, routerId. May have an optional neighbors section.
- bgp - attributes are autonomousSystem, routerId. May have an optional neighbors section.
- is-is – attributes are area, systemId. May have an optional neighbors section.

11.2.14 *iproutes*

```
<iproutes vrfName=name>
 <!-- iproute objects -- >
 <iproute iptype=iptype dest=ipaddr nexthop=ipaddr ifindex=integer metric1=integer␣
↪metric2=integer..metric5=integer proto=protocol />
</iproutes>
```

Example:

```
<iproutes>
 <iproute iptype="ipv4" dest="172.16.1.0/24" proto="local" metric1="0" nexthop="0.0.0.
↪0" ifindex="1"/>
 <iproute iptype="ipv4" dest="172.16.1.3/32" proto="local" metric1="0" nexthop="0.0.0.
↪0" ifindex="1"/>
 <iproute iptype="ipv4" dest="192.168.0.2/32" proto="local" metric1="0" nexthop="0.0.
↪0.0" ifindex="3"/>
</iproutes>
<iproutes vrfName="BLUE">
 <iproute iptype="ipv6" dest="::3:1:1:1:0/120" proto="local" metric1="0" nexthop="::"␣
↪ifindex="2"/>
 <iproute iptype="ipv4" dest="10.0.2.2/32" proto="local" metric1="0" nexthop="0.0.0.0
↪" ifindex="2"/>
```

```
 <iproute iptype="ipv4" dest="10.0.3.0/24" proto="local" metric1="0" nexthop="0.0.0.0
↪" ifindex="6"/>
 <iproute iptype="ipv4" dest="10.0.3.2/32" proto="local" metric1="0" nexthop="0.0.0.0
↪" ifindex="6"/>
 <iproute iptype="ipv4" dest="10.0.2.0/24" proto="local" metric1="0" nexthop="0.0.0.0
↪" ifindex="2"/>
 <iproute iptype="ipv6" dest="::4:1:1:1:0/120" proto="local" metric1="0" nexthop="::"␣
↪ifindex="6"/>
</iproutes>
```

The *iproutes* section contains multiple *iproute* objects. There can be more than one *iproutes* section in a *netobject*; they are distinguished by a *vrfName* attribute (VRF instance name). If no *vrfName* attribute is present, then this is the global routing table (default routing table). SNMP will only discover global routing tables. Cisco remote login can discover routing tables associated with VRFs.

The *iproute* objects within an *iproutes* section contains information from the routing table of a *netobject* obtained by SNMP or by direct login. The network destination is specified by the dest attribute in *destaddress*(no host bits)/*cidr* format. The *nexthop* attribute specifies the next hop for this route. There can be up to five routing metric attributes (metric1 through metric5) included. A value of -1 means that metric is not used. Metric1 is the primary routing metric, the other metrics are alternates. The *ifindex* attribute specifies the physical port that this route uses.

The *proto* attribute returns a string that indicates the routing protocol (these strings are defined in the SNMP IANA-RTPROTO MIB and/or Cisco output) and is included for informational purposes.

11.2.15 *osguess*

```
<osguess various attributes >
```

Example:

```
<osguess accuracy="100" family="Linux" vendor="Linux" type="general purpose" name=
↪"Linux 2.6.32 - 2.6.33"/>
```

The *osguess* object contains the OS guess from *Nmap*. The attribute fields are documented in the *Nmap* DTD for *Nmap's* XML output. *Netmapper* only returns the accuracy, family, vendor, type, and name attributes produced by *Nmap* and only the *osguess* with the highest accuracy is saved in the XML.

11.2.16 *snmp*

```
<snmp sysDescription=”system description returned by SNMP query” >
```

Example:

```
<snmp sysDescription="Linux lxle2lta 3.13.0-39-generic #66-Ubuntu SMP Tue Oct 28␣
↪13:30:27 UTC 2014 x86_64"/>
```

Contains the system description (OID 1.3.6.1.2.1.1.1) returned by a successful SNMP query.

11.2.17 *domainControllerRoles*

```
<domainControllerRoles various attributes >
```

The *domainControllerRoles* section is present if this node has been identified as a domain controller. The attributes are various subroles for the domain controller and have a value of "True" if present. The subrole attributes are:

- IsRoleInfrastructure ="True"
- IsRoleRID="True"
- IsRoleDomainNaming="True"
- IsRolePDC="True"
- IsRoleSchema="True"

11.2.18 *exchangeServerRoles*

```
<exchangeServerRoles various attributes>
```

The *exchangeServerRoles* section is present if this node has been identified as an exchange controller. The attributes are various subroles for the exchanges controller. All attributes will always be present, and will either have a value of "True" or "False".

- IsEdgeServer ="True"|"False"
- IsUnifiedMessagingServer="True"|"False"
- IsMailboxServer="True"|"False"

IsClientAccessServer="True"|"False"

- IsHubTransportServer="True"|"False"
- IsProvisionedServer="True"|"False"

11.2.19 *services*

```
<services>
 <service various attributes>
</services>
```

Example:

```
<services>
 <service status="stopped" startmode="manual" name="ALG"/>
</services>
```

This information is retrieved by remote login and contains the list of running services on the remote machine. The attributes are:

- status="running"|" stopped"– service status.
- startmode=" manual"|"auto"– service start mode.

11.2.20 *hostname*

```
<hostname fqdn=fqdn_name name=hostname/>
```

Example:

```
<hostname fqdn="ws2012ex2010.test3.net" name="ws2012ex2010"/>
```

This information is retrieved by remote login and contains the list of running services on the remote machine. The attributes are:

- fqdn=fqdn_name– fully qualified domain name
- hostname=hostname – host name

11.2.21 *os*

```
<os version=version name=name manufacturer=manufacturer />
 <patches>
  <patch objects>
 </patches>
</os>
```

Example (Windows):

```
<os version="6.3.9600  Build 9600" name="Microsoft Windows Server 2012 R2 Datacenter␣
↪64-bit" manufacturer="Microsoft Corporation">
 <patches>
  <patch name="KB2862152"/>
  <patch name="KB2868626"/>
 </patches>
</os>
```

Example (CentOS):

```
<os version="6.6" name="CentOS" manufacturer="Linux">
 <patches>
  <patch value="2.6.32-504.el6.x86_64" name="kernel version"/>
 </patches>
</os>
```

This information is retrieved by remote login and is the OS and patch information for the host. The OS attributes are self-explanatory. For Windows OSes, each patch object contains a single name attribute as shown. For CentOS OSes, there is only one patch object with attributes name=”kernel version” and a value attribute that is the kernel version.

11.2.22 *physicalserver, cpus, disks, memory*

```
<physicalserver>
 <memory ram=size vmem=size>
 <cpus>
  <!--cpu objects -- >
  <cpu name="">
 </cpus>
 <disks>
  <!--disk objects -- >
  <disk name=name size=size  DriveType=drivetype ProviderName=remotepath>
 </disks>
</physicalserver>
```

Example (Windows):

```
<physicalserver>
 <memory ram="2047 MiB" vmem="3839 MiB"/>
 <cpus>
  <cpu name="Intel(R) Core(TM) i7-4800MQ CPU @ 2.70GHz"/>
 </cpus>
 <disks>
  <disk size="61438 MiB" name="C:"/>
 </disks>
</physicalserver>
```

Example (CentOS):

```
<physicalserver>
 <memory ram="1922372 kiB" vmem="142992 kiB"/>
 <cpus>
  <cpu name="Intel(R) Xeon(R) CPU E7-4830 v2 @ 2.20GHz"/>
 </cpus>
 <disks>
  <disk size="2147 MB" name="/dev/mapper/vg_centos6base-lv_swap"/>
  <disk size="18.8 GB" name="/dev/mapper/vg_centos6base-lv_root"/>
  <disk size="21.5 GB" name="/dev/sda"/>
 </disks>
</physicalserver>
```

This information is retrieved by remote login and contains CPU, memory, and local disk information attributes of the host.

The memory object has attributes of *ram* (physical memory size) and *vmem* (virtual memory size).

The *cpus* object has one *cpu* object for each CPU on the host. The *cpu* object has only a name attribute.

The *disks* object has one *disk* object for each local disk on the host. The *disk* object has *size*, *name*, *ProviderNam*e, *VolumeName*, *DriveType* attributes. The *ProviderName* attribute is set if the disk is remote mounted and its value is the remote path. The *VolumeName* attribute is the local name for a remote mounted disk. The *DriveType* attribute is only used by Windows OSes and indicates the type of drive (LocalDisk, RemoveableDisk, CompactDisk, NetworkDisk).

11.2.23 *applications*

```
<applications>
 <!--application objects -- >
 <application name=name vendor=vendor version=version >
</applications>
```

Example (Windows):

```
<applications>
 <application Vendor="Microsoft Corporation" name="Microsoft Visual C++ 2008␣
↪Redistributable - x86" Version="9.0.30729.4148"/>
</applications>
```

Example (CentOS):

```
<applications>
 <application Vendor="CentOS" name="busybox" Version="1.15.1"/>
</applications>
```

This information is retrieved by remote login and the list of applications installed on the host. The application object has *name*, *Vendor*, and *Version* attributes.

11.2.24 *fileshares*

```
<fileshares>
 <!--fileshare objects for shared disks>
 <!--permissions objects for each shared disk>
</fileshares
```

Example (Windows):

```
<fileshares>
 <fileshare AllowMaximum="True" Path="C:\Program Files\Update␣
↪Services\UpdateServicesPackages" MaximumAllowed="" name="UpdateServicesPackages">
  <permissions>
   <permission IdentityReference="WS2012WSUS\WSUS Administrators" AccessControlType=
↪"Allow" InheritanceFlags="None" PropagationFlags="None" IsInherited="False"␣
↪FileSystemRights="FullControl"/>
  </permissions>
 </fileshare>
</fileshares>
```

The *fileshare* object represents disks available for sharing via SMB or NFS. The SMB information is retrieved by WMI via: *Get-WmiObject -Class Win32_share*. The NFS information is retrieved by WMI via: *Get-WMIObject -Class MSFT_NfsSharePermission -NameSpace RootMicrosoftWindowsNFS*

The *fileshare* object has a *sharetype* attribute that is "nfs" for an NFS share and "smb" for an SMB share. The other attributes on the *fileshare* object are from the WMI query and are Windows specific.

The *fileshare* object has a list of permissions objects whose attributes are set by the returned WMI query and are windows specific.

11.2.25 *firewall*

```
<firewall>
 <profiles>
  <!--profile objects for Windows OS>
 </profiles>
 <rules>
  <!--rule objects -- Tags are OS-specific other than name>
 </rules>
<firewall/>
```

Example (Windows):

```
<firewall>
 <profiles>
  <profile name="Domain" state="on"/>
  <profile name="Public" state="on"/>
  <profile name="Private" state="on"/>
 </profiles>
 <rules>
  <rule EmbedCtxt="@FirewallAPI.dll,-25000" Desc="@FirewallAPI.dll,-25358" Action=
↪"Allow" Active="TRUE" regvalue="v2.
↪10|Action=Allow|Active=TRUE|Dir=Out|Protocol=41|App=System|Name=@FirewallAPI.dll,-
↪25352|Desc=@FirewallAPI.dll,-25358|EmbedCtxt=@FirewallAPI.dll,-25000|" Dir="Out"␣
↪Name="@FirewallAPI.dll,-25352" Protocol="41" App="System"/>
 </rules>
</firewall>
```

Firewall information is retrieved via remote login. For Windows OSes, there are profile and rule objects which are read from the registry.

Each profile object has a name and state attributes as shown. The name attributes are standard and have the values shown. The state attribute value is either "on" or "off". The state value is stored in the registry under the key *HKLMprofilenameEnableFirewall* as a DWORD value of 0 (off) or non-zero (on).

The firewall rules are read from the registry location: *HKLMSYSTEMCurrentControlSetservicesSharedAccessParametersFirewallPolicyFirewallRules*

The *regvalue* attribute in the *firewall* object is the raw registry value string for the rule. The other attributes are parsed out of this rule by using the 'l' separator.

Example (CentOS 7):

```
<firewall>
 <rules>
  <rule sources="" forward-ports="" regvalue="work" icmp-blocks="" services="dhcpv6-
↪client ipp-client ssh" ports="" interfaces="" masquerade="no"/>
 </rules>
</firewall>
```

The firewall section for CentOS 7 does not the profiles section. The firewall attributes are parsed from the column fields are returned by the command "firewall-cmd –list-all-zones".

Example (CentOS 6):

Centos6 hosts do not have a firewall section since their firewall rules are stored in a configuration file named */etc/sysconfig/iptables* – this file is retrieved and is saved as an *rloginDataObject* and its archive *rolename* is *Centos6_firewallrules*. CentOS 7 does not use iptables.

11.2.26 *transient*

A transient section contains information that either changes on each scan or is temporary information cached in the XML for development purposes. It should not be necessary for external parsers to parse information in a transient section – it is documented here for informational purposes.

transient section for *netobject*

The *transient* section in the *netobject* section can contain two different tables with an example shown below:

```
<addresstable>
 <addrtblentry macaddress="b8:27:eb:30:cd:ec" ipaddress="200.168.3.50" ifindex="4"/>
</addresstable>
<bridgetable macaddress=macaddress>
 <bridgetblentry macaddress="78:da:6e:e6:3d:0" portnum="1" vlanid="1"/>
</bridgetable >
```

The *addresstable* section contains *addrtblentry* entries that map mac addresses to IP addresses. Once this table is read, these mac address to IP address mappings are propagated out to appropriate *ipaddress* sections contained in *interface* sections.

The *bridgetable* section contains *bridgetblentry* entries that map a mac address to a port number that this mac address was seen on. This information is used to build *l2connection* sections.

11.2.27 *vmwareVapp*

```
<vmwareVapps>
 <!--Vmware Vapp contents ?
</ vmwareVapps>
```

Example:

```
<vmwareVapps>
 <host name="dasi-vcenter-2">
  <vapp name="wsus2008">
   <vm name="5011904c-6e71-779f-a686-d120248a1566"/>
   <vm name="5011d8f0-827e-05cd-2911-e025797bb9bb"/>
   <vm name="5011d5b7-38f5-88a1-cde8-d9c9bd00c064"/>
   <vm name="50117b29-35c1-77e5-ab4e-dca4e143b401"/>
   <vm name="5011d59c-8926-8fe0-5006-9424b316b1f1"/>
   <vm name="5011cd27-81f5-4894-4141-d4b3f93bc63f"/>
  </vapp>
 </host>
</ vmwareVapps>
```

The *vmwareVapps* section gives the VMware Vapps retrieved by VMware API host query.

The *host* object contains the vapps found on that host; the name attribute is the name of the host.

The *vapp* object lists the virtual machines in the Vapp; the name attribute is the name of the Vapp.

Each *vm* object in a *vapp* object has a name attribute that is the unique instance identifier of this virtual machine that resides in the vapp.

11.2.28 *rloginDataObjects*

```
<rloginDataObjects>
 <!--Rlogin data archives ?
</ rloginDataObjects>
```

The *rloginDataObjects* section is used to store retrieved files from the host in a base-64 encoded ZIP archive format.

Each *rloginDataObject* under the *rloginDataObjects* section has the form:

```
<rloginDataObject hostname=hostnameUniquified timestamp=tsString id=uuidForNetObject>
 <archive rolename=rolename>
  <archiveDataLines>
   <archiveDataLine value=base64string/>
   ...many archiveDataLines...
  </archiveDataLines>
  <archiveFiles>
   <archiveFile name=localFilename orgpath=originalPathOnHost/>
  </archiveFiles>
 </archive>
</rloginDataObject>
```

The attributes for the *rloginDataObject* are:

- hostname=hostnameUniquified – The host name that this data was retrieved from; uniquified by adding part of the netobject's uuid to the end of the hostname.
- timestamp=tsString – A timestamp string indicating when this data was retrieved from the remote host.
- id=uuidForNetObject – The uuid of the NetObject that owns this data.

Each *rloginDataObject* can have multiple archive objects, each archive object contains the file(s) associated with some role data or default data retrieved from that hose. Each archive object is a base-64 encoded ZIP archive containing one or more files for the role or default data.

The *rolename* attribute of the archive object is the identifying role tag or default data tag for this data. Under an archive object are multiple *archiveDataLine* objects, each containing a value attribute that is the base64 encoded line for this entry. Reconstructing the ZIP archive for this data requires all value attributes of *archiveDataLine* objects to be written to a file in the same order as found in the archive object, and then decoding this base64 file back to its binary form – this will be the original ZIP archive.

Following the archive object is an *archiveFiles* object containing one or more *archiveFile* objects that gives the files in the archive.

The attributes for the *archiveFile* object are:

- name=localFilename – The name of the file in the archive.
- orgpath=originalPathOnHost – The name of the file on the remote host.

Example:

```
<rloginDataObject hostname="centos6ns.test2.net-b0d50d61" timestamp="2015-12-16_16-45-
↪21" id="b0d50d61-a372-11e5-b708-00505691a02e">
 <archive rolename="DHCP_Server">
  <archiveDataLines>
   <archiveDataLine value=
↪"UEsDBBQAAAAAALqFkEfMyeEdygwAAMoMAAALAAAAZGhjcC5jb25maWcjDQojIFNhbXBsZSBjb25m"/>
   ...other lines not included...
  </archiveDataLines>
  <archiveFiles>
   <archiveFile orgpath="/etc/dhcp/dhcpd.conf" name="dhcp.config"/>
  </archiveFiles>
 </archive>
</rloginDataObject>
```

When an XML file with *rloginDataObject* is read into memory, the compressed archives are kept as is. To examine the data in uncompressed form, the *File* > *Expand Rlogin Data* menu choice is used.

This expands the data to the current working directory in the following directory structure:

```
RloginNodeConfigData\tsString\hostnameUniquified1\rolename1\filename1
                             |                    |         \filename2
                             |                    |         \filenameN
                             |                    \rolename2\filename1
                             |                    |         \filename2
                             |                    |         \filenameN
                             |                    \rolenameN\filename1
                             |                              \filename2
                             |                              \filenameN
                             \hostnameUniquified2\rolename1\filename1
                                                  |         \filename2
                                                  |         \filenameN
                                                  \rolename2\filename1
                                                  |         \filename2
                                                  |         \filenameN
                                                  \rolenameN\filename1
                                                            \filename2
                                                            \filenameN
```

etc.

11.3 Python API Interface

This section is intentionally left blank at this time.

CHAPTER

TWELVE

GLOSSARY

ADSI Active Directory Service Interfaces

CIDR Classless Inter-Domain Routing

DASI Distributed Analytics and Security Institute

DNS Domain Name Server

FQDN Fully-Qualified Domain Name

LDAP Lightweight Directory Access Protocol

SNMP Simple Network Management Protocol

WMI Windows Management Instrumentation

WSUS Windows Server Update Services

VRF Virtual Routing Framework

VDC Virtual Device Context