

TZWorks® Modular Inspection Network Xfer (*minx*) Agent Users Guide



Abstract

minx is a standalone, command-line tool that acts as a client to exchange data from any client computer (Windows, Linux or macOS) to a forensic workstation running a *nx* server. Data that is transferred to the client is encrypted.

Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v0.29 of *minx*

Updated: Apr 25, 2025

Table of Contents

1	Introduction	3
2	How to use <i>minx</i> and <i>nx</i> to complement live forensics collection	4
3	Configuring <i>nx</i> as a server	5
4	Configuring <i>minx</i> (the client)	8
5	Sending data to the server	10
5.1	Sending the output of other tools with <i>minx</i>	10
5.2	Copying the contents of files	11
5.2.1	Copying one or more files	12
5.2.2	Using the <i>-pipe</i> option to copy many files	12
5.2.3	Using the <i>-copydir</i> option to copy whole directories or a set of subdirectories	13
5.2.4	Using the Combination Copy Options.....	14
5.2.5	Volume Shadow Copies.....	14
5.3	Imaging Disks and Volumes	15
5.3.1	Scan the drives on the system	16
5.3.2	Imaging an entire drive	16
5.3.3	Imaging an entire volume	16
5.3.4	Copying Interesting disk sections.....	17
5.4	Running with and without a license file.....	17
6	Annotating metadata to the data sent	17
6.1	Tagging a <i>minx</i> Client with Identifiers.....	17
7	Using Scripts.....	18
7.1	Script File Syntax	18
7.1.1	General Rules	19
7.1.2	Command Lines.....	19
7.1.3	Environment Variables.....	19
7.2	Using Built-in commands	20
7.3	Spawning third party tools.....	20

7.4	Enumerating files and copying them	21
7.4.1	Copying a collection of files (method 1 – just using built-in commands).....	21
7.4.2	Copying a collection of files (method 2 – spawning 3 rd party tools).....	21
7.5	Serving out command scripts.....	22
7.5.1	Example of the setup of using -queryfile <script name>	23
8	Example of a script file to collect various artifacts	24
9	Various Use-Case for Transferring Data.....	24
9.1	Normal Private Intranet for both <i>minx</i> and <i>nx</i>	25
9.2	Sending Data from a Private Intranet to a Public Internet Address	26
9.3	Using TCP/IP Redirection	26
10	Zlib Dependency.....	28
11	Available Options	28
12	Authentication and the License File.....	30
13	References	30

TZWorks® Modular Inspection Network Xfer (*minx*) Agent Users Guide

Copyright © TZWorks LLC

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=36 [05-MINX-111SM]

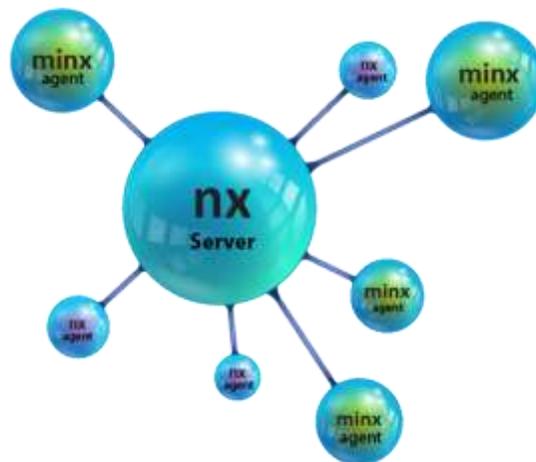
Contact Information: info@tzworks.com

1 Introduction

The *minx* utility is a command line tool that runs on an endpoint target machine, acting as an agent to collect data from the endpoint and send it a central forensics workstation running the *nx* tool.

Therefore, to understand *minx*, one needs to understand *nx*.

As background, the *nx* tool can act as both a client and server to transfer data from one computer to a central computer. From a terminology standpoint, the *server* is the forensic collection point gathering information during an incident response (not necessarily a Windows Microsoft Server). The *clients* are those endpoint computers that are under investigation that would be sending data to the central forensics workstation. What can be confusing at first, is that the *nx* tool can be both (a client and a server), depending on the application's configuration. Both *minx* and *nx* clients can co-exist and work with an *nx* server as shown below.



The network relationship between the *minx* client and the *nx* service uses peer-to-peer communication. This is defined to be communication between just two nodes, as opposed to multi-node or broadcast communication. For peer-to-peer communication, domain credentials are not required to be set up if deployed in an enterprise network. As long as *minx* can communicate to the *nx* service's IP address/port, without being impacted by firewalls or other network devices that can block IP traffic, the communications should be seamless.

When designing **minx**, and expanding the existing **nx** client capabilities, there were several requirements levied by users. These requirements were: (a) the ability to copy files that were locked down by the operating system, (b) to have the client run from an internal script such that a separate instance was not required for each command executed, and (c) image/copy raw bytes from a disk or volume.

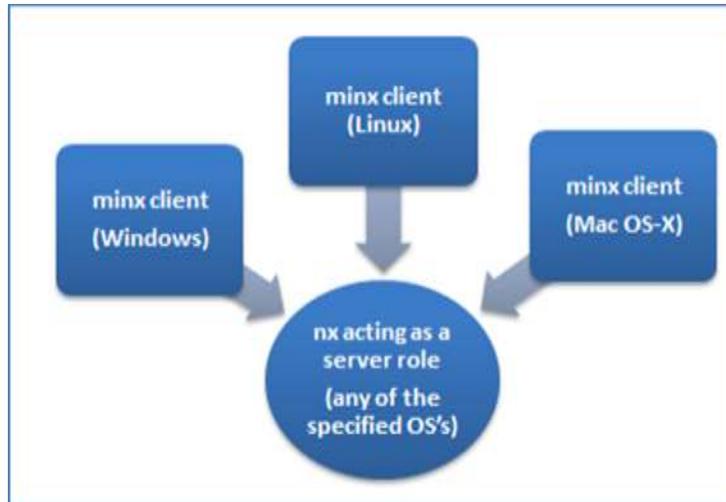
The enhancements put into **minx** include: (a) an integrated NTFS engine to allow **minx** to copy any file from a host Windows computer by accessing the file data via raw cluster reads, (b) an ability to scan all drives attached to a Windows computer, (c) an ability to copy any number of bytes from a specific drive, (d) an internal directory enumerator with a filtering ability to target specific files within one, or a group, of subdirectories, (e) the ability to spawn other applications and act on their output, (f) the ability to pull common artifacts from all the volume shadow copies, and (g) an internal scripting engine that allows **minx** to receive instructions from the **nx** service and act on them.

The communications protocol between **nx** and **minx** is the same as the previous **nx-client** to the **nx-server** communication, with some additional changes to accommodate the enhanced features of **minx**. The network transport used is TCP/IP with the data content encrypted in an RC4 stream cipher. To ensure data integrity from the client computer to the final archived file on the server, the data is computed into a hash at client side before transmission and during receipt by the server. A mismatch in hashes results in the archive file being labeled as having errors. Labels, comments, and filenames are allowed to be passed during each data transfer, which is read by the server and acted upon accordingly. Currently, **minx** and **nx** are restricted to IPv4.

There are compiled versions for Windows, Linux, and macOS, and the architecture is such that each one is designed to play well with another instance despite operating on a different OS (e.g. use **nx** on a Linux box as the server and use **minx** on a Windows client box to send data to the server). To use this tool, an enterprise license is required.

2 How to use **minx** and **nx** to complement live forensics collection

The terms 'client and server' are used here as 'roles' for the **minx** and **nx** tools, respectively. Any machine that **minx** or **nx** runs on does not require the operating system to run as a client or formal server. Any standard (non-server operating system) computer configuration will work. All that is needed is that there be some level of network connectivity between the computers.



The direction of data flow is initiated by the client to the server. One can think of the clients as those workstations you would like to extract forensics artifacts from, and the workstation you are sending the extracted artifacts to is the server. In the server role, *nx* can handle multiple clients at once. Since the *nx* server-mode is a multi-threaded application, *nx* spawns a separate thread per client connection. Therefore, simultaneous collection from a few clients should not be an issue under normal loading conditions. The maximum number of simultaneous client connections is really a function of (a) the computer and network resources of the machine acting in the server role, (b) the amount of data transferred from each client, and (c) the network bandwidth of the system.

3 Configuring *nx* as a server

To configure *nx* as a server to work with a *minx* client, both the *nx* and *minx* network protocols need to be in sync. This is done by looking at the version of either of the tools' protocol (which is different than the version of the tool). For example, version 0.27 is the first version of *nx* that is compatible with *minx*. The protocol used with version 0.27 of *nx* is version 0.08.

If you are unsure which version you have or need, then just run *nx* (or *minx*) at the command prompt; the tool will display the current version you have as well as what version of the protocol it is using. The protocol version matching allows for proper network handshaking between the *minx* client and the *nx* service. It is anticipated that the protocol will rarely be updated, so while the version number of the tools (*nx* and *minx*) will increment with each update, the protocol version typically will not.

Below is a screenshot of the *nx* server options. The highlighted area is where the protocol versioning is shown on the command prompt.

```
Command Prompt
nx - full ver: 0.27; [protocol ver: 0.08 (nx)]; Copyright (c) TZWorks LLC
Usage: Protocol ver 0.08 required for minx
nx options available for minx client
Forensics workstation (server) setup
nx -server -ip <ip addr> -port <port #> -dir <results dir> [options]
Options for server
-redirect <ip addr> = where client sends data to
-scripts <dir for scripts> = scripts available for minx clients
-key "password phrase" = set the key for the transmitted data
```

To run **nx** in server mode, these are the required parameters:

- **-server** option to tell **nx** to run in server mode and listen for client connections.
- **-ip <IPv4 address>** and **-port <number>** identifies which IP address and port number **nx** should listen to for client connections. This IP address should be the one assigned to the computer that is running the **nx** service.
- **-dir <folder>** identifies which directory to store the log file and any data sent from a client to the server.

The rest of the server parameters can be optional, depending on whether they are needed or not. For example, if you want to set up a session key, one can use the **-key <password phrase>**. If this is done, then the same **<password phrase>** needs to be used at the **minx** client end. This **<password phrase>** is used to assist in the generation of the RC4 encryption key that is used. Without using the **-key** option, **minx** and **nx** still encrypts the data transferred over the network but will use its own key generation algorithm.

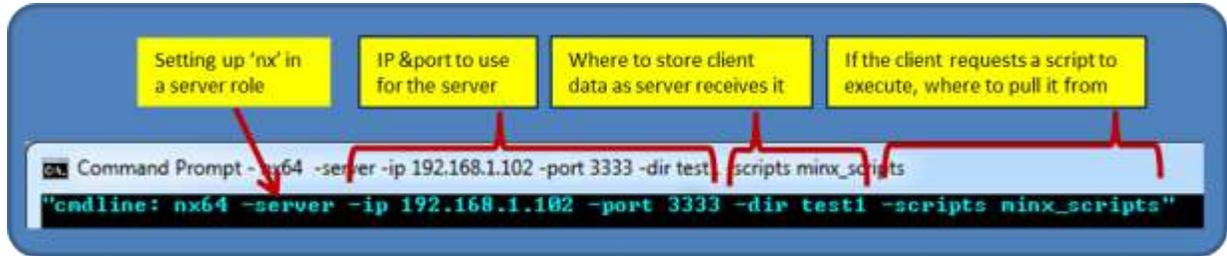
Still in the experimental phase: if you want to have the **nx** server offer up scripts for the **minx** client to run, then use the **-scripts <folder>** option. Then, when a request comes in from **minx** to retrieve a particular script, the above option tells the **nx** server where to look for them and then pass the script back to the **minx** client. Once **minx** receives the script, it will execute it and pass the results back to the **nx** server. This architecture allows the scripts to reside and be maintained on the server side. Thus, when a change is made to a script, all the **minx** clients will have access to the same modifications of the script when they run. More details about scripts and how to write them are discussed later.

In some situations, it may be required to make use of a redirector to send data from a private network to the host computer running the **nx** service. To handle this use-case, we have the **-redirect <ip addr>** option available. This tells the **nx** service to expect **minx** network packets to be directed to another IP address before reaching the **nx** service. Since network configuration with IP and port redirection can be a complicated topic, it is discussed in its own section later in this document.

The final set of server options is related to formatting the log file. Since the results log file is archived as a CSV type file, one can use the same **TZWorks** command options used in many of the other tools, such as: excluding whitespace between field delimiters (**-no_whitespace**), changing the field delimiter from a

pipe character to a comma character (*-csv_separator “,”*), changing the timestamp format (*-dateformat “yyyy/mm/dd”*) or grouping the date and time together (*-pair_datetime*).

Below is an example of setting up *nx* in server mode:



After invoking the above command, the operating system will request permission to open up a network connection at the port specified in the 'listen' mode. One should get a pop-up box similar to the screenshot below. To configure *nx* to work with *minx* clients (or *nx* clients) on various networks, select the “public networks” option.



4 Configuring *minx* (the client)

When configuring *minx* to talk to the *nx* service, one needs to specify the same IP address, and port number used when setting the *nx* server, and then append the desired command to whichever action is needed. The screenshot below shows the command line options that are currently available.

```
Administrator: Windows PowerShell

Usage:

Connect to server
  minx64.exe -ip <server ip addr> -port <server port #> [options]

Basic options
  -key <"password">           = set the encryption key
  -nx <lic num of nx>         = only access nx using license # specified
  -ping                       = ping server and displays time stats
  -quiet                      = don't echo what is being sent
  -cust_id <label>           = assoc transactions w/ this client label
  -host_id <label>           = assoc transactions w/ this host label
  -pipe                       = use stdin to pipe in cmds.
  -name <"tell server to use this filename to store results">
  -comment <"insert a comment in the server log">
  -subdir <folder>           = store results in this subdirectory
  -retain_path                = store results in orig subdir context

Drive options
  -scandrives                 = pull stats on all the drives
  -copydrive <drive #> [-offset <start>] [-size <#>] [-grip]
  -copyvolume <partition letter> [-offset <start>] [-size <#>] [-grip]
  -copy_diskgaps <disk #> [-grip] = pull sectors from disk# not tied to a vol
  -copy_all_diskgaps[-grip]      = pull sectors from all disks not tied to a vol
  -copy_bootfiles              = pull MBR, VBR (vol boot record) files

File copy options
  -copyfile <file>           = specific file
  -copydir <dir> [-level <#>] = entire folder and/or subdirectories

Combo copy options [for common collections of files]
  -pull_sysfiles              = pull $MFT, $UsnJrnl:$J files
  -pull_reghives              = pull system and user registry hives
  -pull_evtlogs               = pull system event logs
  -pull_lnk                    = pull user LNK files and Jumplists
  -pull_pfs                   = pull system prefetch files
  -pull_systrash              = pull recycle bin from system vol
  -pull_all                   = pull all of the above

Targeting Volume shadow copy's that can be used with above options
  -vssall <partition letter> = pull data from vol shadows

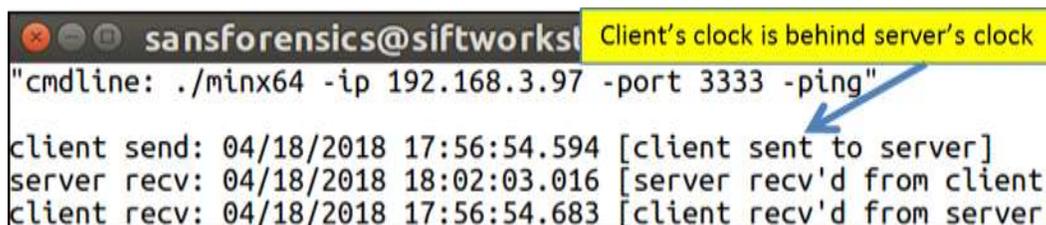
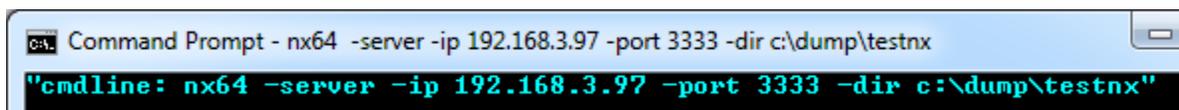
Enumerate directory examples
  dir <folder> /b /s | minx64.exe -pipe -filter <*"partial*"*.ext> -copyfile
  minx64.exe -copydir <dir> -level <#> -filter <*"partial*"*.ext>

Automation options [Experimental]
  -script <script file>      = run specified script
  -get_cmds [-queryfile <script>] = get script from server

Examples of running client via piping cmds via stdin or copying files
  ipconfig /all | minx64.exe -pipe -ip <serv> -port <#> -name ipconfig_all.txt
  netstat -anob | minx64.exe -pipe -ip <serv> -port <#> -name netstat_anob.txt
  wmic process list full | minx64.exe -pipe -ip <serv> -port <#> -name procs.txt
  dir c:\*.pdf /b /s | minx64.exe -ip <serv> -port <#> -copyfile -pipe
```

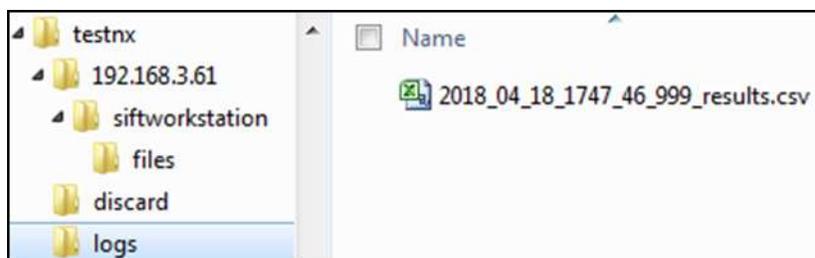
Continuing with the basic configuration, one should test out the network connectivity between the *minx* client and the *nx* service. To do this, one should try to 'ping' the server from the client computer using the [-ping] option. This will ensure the crypto is synchronized between *minx* and *nx*.

Below is an example of doing this. The *nx* service was configured to be listening on IP address 192.168.3.97, port 3333 (first screenshot). The *minx* client is running on an Ubuntu 64-bit computer, sending the ping command to the *nx* service (second screenshot).



Not only does the [-ping] verify the connection, but the timing statistics get archived in the server log. For the above example, if one notices closely, one can see that the client's clock is 'behind' that of the server's clock. How does one know this? One can see that the round-trip delay from the client -> server -> client was about 89 milliseconds (difference of 17:56:54.594 and 17:56:54.683). Using half of this difference should approximately equate to the transit time in one direction or 44.5 msec. However, the difference in timestamps from when the client sent the packet (using the client's clock) and when the server received the packet (using the server's clock) is approximately 5 minutes. This says the client and server are out of sync by approximately 5 minutes. Anything that significant in time difference should be noted since any artifacts extracted from a client computer is relative to that specific client's computer clock. While this is a contrived example where the clock was manipulated on one system, it makes the point that when taking artifacts from one computer and comparing them to another computer, one needs to be able to synchronize the time between artifacts from different machines.

On the *nx* server side, all metadata is archived in the log file; this includes the time statistics from the ping operation. This log file is named '<start_date_time>_results.csv'. Based on where the server set up the root directory for this session, multiple subfolders are set up. For now, the one of interest is the *logs* subdirectory which stores all the log files. The other subdirectories will be discussed later.



When looking in the log file for the specific entry, one should see the following type statistics for the ping operation.

host	orig src IP	type	data	comment
siftworkstation	192.168.3.61	ping		client send: 04/18/2018 17:56:54.594; server rcv: 04/18/2018 18:02:03.016; client rcv: 04/18/2018 17:56:54.683;

5 Sending data to the server

After the *minx* client and *nx* server connectivity have been verified, one can send artifacts from a client computer to the server in a couple of ways. These can be categorized into three groups: (a) running separate or built-in operating system tools and piping their output into *minx* to send to the server, (b) sending the contents of a file to the server with one of *minx*'s copy options, and (c) sending raw bytes from a drive or the drive's statistics to the server. Each of these categories is discussed below.

5.1 Sending the output of other tools with *minx*

The most basic command one can invoke is to take any console output from some other tool and relay it to the server. Below are some examples of doing this:

```
Examples of running client via piping cmds via stdin or copying files
ipconfig /all | minx64 -pipe -ip <serv> -port <#> -name ipconfig_all.txt
netstat -anob | minx64 -pipe -ip <serv> -port <#> -name netstat_anob.txt
wmic process list full | minx64 -pipe -ip <serv> -port <#> -name procs.txt
dir c:\*.pdf /b /s | minx64 -ip <serv> -port <#> -copyfile -pipe
```

If we take apart an example and annotate each field/option, it becomes clear what is going on. For this example, we look at the network configuration as described by *ifconfig* (on Linux) or *ipconfig* (on Windows). As part of the command, one can annotate a comment to help the examiner remember any context information at the time of the collection. One can also specify a name to be used as part of the filename that is saved on the server end. Below is the command that is used along with annotations of what each portion of the command does.



Below is a snapshot of the log file entry for this above command. From it, one can see the type of metadata recorded per transaction, such as status, date/time, source IP/port, data type, etc. For those transactions that created a file in the archive directory, an MD5 hash will be computed and documented here. Furthermore, the name of the archive file itself will incorporate: (a) date/time, (b) md5 hash, and

(c) any name that was requested to be used. While this makes for long filenames, it solves the problem of ensuring all names are unique.

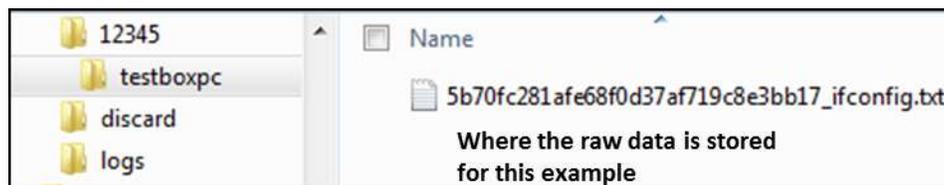
The comment, specified by the **-comment** option in the above command, was included in the log entry. The name, specified by the **-name** option, was appended to the end of the filename of the archived file. Finally, the customer ID was changed from the default of the client IP address to a user-defined name via the **-cust_id** option.

Log entry associated with **ifconfig-a** command

target	host	orig src IP	comment	file recvd
12345	testboxpc	192.168.154.180	ifconfig-a	\\?\c:\dump\testnx\12345\testboxpc\5b70fc281afe68f0d37af719c8e3bb17_ifconfig.txt

Customer ID MD5 hash

The location of the raw data that was transmitted is put in the customer ID's folder as shown below.



5.2 Copying the contents of files

There are a few ways of copying a file. The option that is best will depend on your requirement. Below demonstrates the various options:

```
File copy options
-copyfile <file> = specific file
-copydir <dir> [-level <#>] = entire folder and/or subdirectories
dir <folder> /b /s | minx64 -pipe -filter <*>partial*!*>.ext -copyfile
minx64 -copydir <dir> -level <#> -filter <*>partial*!*>.ext
```

For all the copy operations, the following rules are used in **minx**:

- For files that have sparse clusters like \$UsnJrnl:\$J, **minx** will only extract non-sparse clusters.
- Internally, **minx** will determine if the file is locked down by Windows, and if it is, then it will bypass the Windows security and restrictions and resort to raw disk cluster reads to copy a file. If the file can be read with normal file reads, then it will do that instead. Raw disk cluster reading requires **minx** to run with Administrator privileges.
- If the analyst wishes to save the resulting file that is copied in a set of subdirectories that mimic the source computer, then one needs to specify the **-retain_path** option. Without this option, the default behavior is to store the copied file in a common directory unique to that target computer. In either case, the original path will be stored in the results log.

- One can specify each target computer’s unique results subdirectory path, by using one or both of the options: **-host_id <user defined name>** and/or **-cust_id <user defined name >**. The default behavior uses a combination of the target computer’s IP address and hostname for the unique results subdirectory path.

5.2.1 Copying one or more files

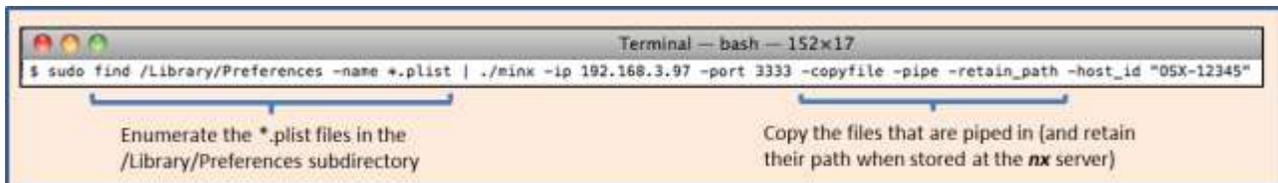
To copy one or more files, one can use the **-copyfile <file(s)>** option. Typical files you may want to copy individually are the *\$MFT* file, *\$UsnJrnl:\$J* file, and any other hidden file that is not easily enumerated by a directory command. One can copy more than one file by separating the argument passed in **-copyfile** by a pipe character, like “*file1 | file2 | ...* “. This variation is useful if the number of files is limited to a few files. For example, to copy the *\$MFT*, *\$Boot*, and *\$Bitmap* files, one could issue the following command:

minx -copyfile “c:\\$MFT | c:\\$Boot | c:\\$Bitmap” -ip <addr of the nx server> -port <# of the server>

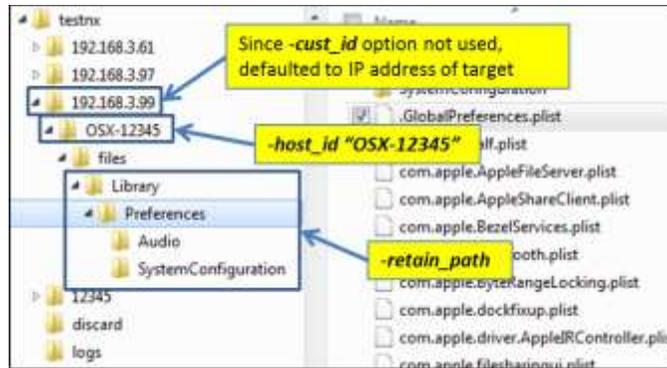
5.2.2 Using the **-pipe** option to copy many files

For copying a category of file types, you will most likely want to either pipe in the files in one session using the **-pipe** option or use a more scriptable option that makes use of the **-copydir** option. The **-pipe** option is used in conjunction with the **-copyfile** option to tell **minx** to use standard input to determine which files to copy. This is best shown via an example.

In the example below, the macOS command pipes in all the *plist* type files from a specified directory into **minx** to copy:



On the server end, when **nx** receives data from a file copy operation from **minx**, the **nx** service stores each file contents in the subfolder that matches the source **minx** target machine (which in this case is the IP address of the source machine). One can override this behavior by specifying a **-host_id**, **-cust_id** or both, which will cause the **nx** server to store the data in the subfolders specified by these other parameters. In the example above, two additional options are used: (a) **-retain_path**, and (b) **-host_id "OSX-12345"**. The first (**-retain_path**), tells the **nx** server when storing the results of the copy operation to store the file in the directory hierarchy from the original client machine. The second (**-host_id**) tells the **nx** server not to use the default client hostname to store the results, but to use the **-host_id** specified; it is another way to further categorize the client’s data.



Other implicit behavior on the *nx* side can be seen from the results log file. This includes: (a) the name of the file is used as the archived name, (b) the path of the file is included in the comments section of the server log, and (c) the MD5 hash of the file.

host	orig src IP	type data	comment	file MD5 ha
OSX-1234	192.168.3.99	copied file	/Library/Preferences/.GlobalPreferences.plist	b6b346e4d
OSX-1234	192.168.3.99	copied file	/Library/Preferences/Audio/com.apple.audio.DeviceSettings.plist	7245d4ddc5
OSX-1234	192.168.3.99	copied file	/Library/Preferences/Audio/com.apple.audio.SystemSettings.plist	0428ef00db
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.alf.plist	d9bdb8a154
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.AppleShareClient.plist	1d6ccf38aa
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.AppleShareClient.plist	cb4755a830
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.BezeServices.plist	b0b8bffa3f
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.Bluetooth.plist	e3a557dc6c
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.ByteRangeLocking.plist	c6a755508a
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.dockfixup.plist	560388868f
OSX-1234	192.168.3.99	copied file	/Library/Preferences/com.apple.driver.AppleIRController.plist	2219fa1dd1

One point to mention here is in regards to the client source IP address that gets archived in the log: this IP address is extracted from the client side and packaged as part of the data that gets sent to the *nx* server. This means that if a client is behind a NAT'd firewall, the actual IP address of the client machine still shows up in the log file as opposed to the firewall's IP address.

5.2.3 Using the *-copydir* option to copy whole directories or a set of subdirectories

Piping in file names from standard input using the *-pipe* and *-copyfile* options is a common way to copy files. However, there is another way which is more flexible and is the choice option when using scripts. The *-copydir* allows one to specify a starting folder, the number of subdirectories to traverse (via the *-level* option), and to discriminate files based on some filter criteria (via the *-filter* option).

Let's say one wanted to pull artifacts from the *c:\users* directories. These artifacts could be user hives, user documents, LNK files, jump lists or whatever. One could construct the following command where the filter would have the following syntax: *"ntuser.dat|usrclass.dat|*.lnk|*ions-ms|*.docx|*.txt"*, starting at the root folder *c:\users*. The syntax would look something like this:

minx -copydir c:\users -level 8 -filter "ntuser.dat | usrclass.dat | *.lnk | *ions-ms | *.docx | *.txt"
<rest of the command>

The <rest of the command> would include the ***-ip <address of the nx server>*** and ***-port <# of the server>***.

5.2.4 Using the Combination Copy Options

These are command shortcuts that make use of the various internal options to copy common collections of files. They include the following:

Group Option	Files Targeted
<i>-pull_sysfiles</i>	<i>\$MFT, \$Boot, \$LogFile, \$Bitmap, \$BadClus:\$Bad, &UsnJrnl:\$J</i> files
<i>-pull_reghives</i>	User and OS level (system, software, security, etc) registry hives
<i>-pull_evtlogs</i>	Event and <i>setupapi</i> logs
<i>-pull_lnk</i>	<i>LNK</i> and <i>JumpList</i> files
<i>-pull_pfs</i>	<i>prefetch</i> files
<i>-pull_systrash</i>	Recycle Bin directory on the system drive.
<i>-pull_all</i>	All pre-defined groups above

There are groups for registry hives, event logs, prefetch files, LNK/JumpList, trash entries, and system files. One can use one or more of the options listed above in one session, or just invoke the ***-pull_all*** parameter to tell ***minx*** to pull all pre-canned artifacts. ***minx*** will discern which version of the Windows operating system the tool is running on so that the proper default directories are targeted for the file groups selected. The other unique aspect about this option is that it will spawn multiple instances to go after the specific groups. For computers with multiple cores, this will result in a faster copy.

5.2.5 Volume Shadow Copies

To access Volume Shadow copies, one needs to be running with administrator privileges. Volume Shadow copies, as is discussed here, only applies to Windows Vista, Win7, Win8, Win10 and beyond. It does not apply to Windows XP.

To make it easier with the syntax, we've built in some shortcut syntax to access a specified Volume Shadow copy, via the ***%vss%*** keyword. This internally gets expanded into `\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy`. Thus, to access index 1 of the volume shadow copy, one would prepend the keyword and index, like so: ***%vss%1***, to the normal path of the file. For example, to access a user hive located in the *testuser* account from the *HarddiskVolumeShadowCopy1*, the following syntax can be used:

minx -copyfile %vss%3\Users\testuser\ntuser.dat -ip <nx server IP addr> -port <nx server port #>

To determine which indexes are available from the various Volume Shadows, one can use the Windows built-in utility *vssadmin*, as follows:

```
vssadmin list shadows
```

To filter some of the extraneous detail, type:

```
vssadmin list shadows | find /i "volume"
```

While the amount of data can be voluminous, the keywords one needs to look for are names that look like this:

```
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1  
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2  
...
```

From the above, notice the number after the word *HarddiskVolumeShadowCopy*. It is this number that is appended to the *%vss%* keyword.

5.2.5.1 Volume Shadow Copies applied to the Combination Copy Options

The second flavor of targeting artifacts in the volume shadow copies is to use the *-vssall <partition letter>*. This instructs *minx* to analyze all volume shadow copies on that volume and pull the files requested.

5.3 Imaging Disks and Volumes

Since *minx* can pull the file contents by reading raw clusters, this functionality is extended to perform imaging of volumes, drives and specific sectors. The following options are available in this regard.

```
Drive options  
-scandrives = pull stats on all the drives  
-copydrive <drive #> [-offset <start>] [-size <#>] [-gzip]  
-copyvolume <partition letter> [-offset <start>] [-size <#>] [-gzip]  
-copy_diskgaps <disk #> [-gzip] = pull sectors from disk# not tied to a vol  
-copy_all_diskgaps [-gzip] = pull sectors from all disks not tied to a vol  
-copy_bootfiles = pull MBR, UBR (vol boot record) files
```

One should keep in mind that *minx/nx* was *not designed* for entire disk imaging; it would be very slow compared to other imaging tools that image over the network. The reason is that *minx/nx* uses much more network overhead than other network tools and consequently imaging over the network. This option was only added so one could image small portions of the disk (such as the boot sector or sectors that are suspicious). With that in mind, this capability is provided, if it is needed. The imaging format available for *minx* does not allow for *E01* or *AFF* formats, but only in the raw 'dd' format. One, however, can compress the 'dd' image by using the *-gzip* option. The compression uses the *zlib* library and will

make use of multiple threads to process the raw data in parallel chunks to minimize the processing image of the compression.

When using the default option, the extension used for the resulting image file will be “.dd”. If using the compression option (-*gzip*), then the extension used will be “.gz”. The other naming convention is to use the following syntax to represent the drive image name.

<MD5 hash>_drive_<#>_offset_<starting offset>_<ending offset>.[dd | gz]

5.3.1 Scan the drives on the system

To copy specific raw sectors of a drive, one should understand the details of the drives on the system. Specifically, one needs to know: (a) the physical drive number of the device, (b) the device volume offsets, and (c) the number of bytes for each volume. There are a number of built-in Windows tools to get this information, but an easy way is to use the *-scandrives* option in *minx*. This option will enumerate all the drives attached on the target system and output the results to the console (as well as send the results to the *nx* server). Below is a snapshot of doing this on a system. From the data, one can see the starting/ending offset of a volume along with its signature. Based on this data, one can extract volume data using the starting offset and desired number of bytes one wishes to extract.

disk sig	type	start offset	end offset	num bytes	vol sig	type	description
5fe9-e51f	MBR	0x000000000000	0x00000000ffff	0x000000100000	0000-0000	0x00	empty; fixed
5fe9-e51f	MBR	0x000000100000	0x00000064ffdf	0x00000063ffe0	34b6-9a5e	0x07	ntfs; fixed
5fe9-e51f	MBR	0x00000064ffe0	0x00000064ffff	0x000000000200	0000-0000	0x00	empty; fixed
5fe9-e51f	MBR	0x000000650000	0x00f2979ffdf	0x00f2914ffe00	92b9-d459	0x07	ntfs; fixed
5fe9-e51f	MBR	0x00f2979ffdf	0x00f2979fffff	0x000000000200	0000-0000	0x00	empty; fixed
5fe9-e51f	MBR	0x00f297a00000	0x01dcee7ffdf	0x00ea56dffe00	728c-3951	0x07	ntfs; fixed
5fe9-e51f	MBR	0x01dcee7ffdf	0x01dcee55fff	0x000000256200	0000-0000	0x00	empty; fixed
7c69-c138	MBR	0x000000000000	0x00000000ffff	0x000001000000	0000-0000	0x00	empty; removable
7c69-c138	MBR	0x000001000000	0x000ee8ffff	0x000ee7ffe0	fcce-c14a	0x07	ntfs; removable
7c69-c138	MBR	0x000ee8ffff	0x000ee8ffff	0x000000000200	0000-0000	0x00	empty; removable
6673c82e-2b30-42bc-9b95-978978dd598c	GPT	0x000000000000	0x0000000043ff	0x000000004400	00000000-0000-0000-0000-000000000000	0x00	empty; unallocated
6673c82e-2b30-42bc-9b95-978978dd598c	GPT	0x000000004400	0x0000080043ff	0x000008000000	a37c21d0-a16b-47e6-8725-4989147f4de5	0xee	Microsoft re
6673c82e-2b30-42bc-9b95-978978dd598c	GPT	0x000008004400	0x0000080ffff	0x00000000fbc0	00000000-0000-0000-0000-000000000000	0x00	empty; unallocated
6673c82e-2b30-42bc-9b95-978978dd598c	GPT	0x000008100000	0x00f42c0ffff	0x00f424000000	cfe7714b-671e-4b81-bd38-24e0f9d0f31f	0x07	ntfs; Basic d
6673c82e-2b30-42bc-9b95-978978dd598c	GPT	0x00f42c100000	0x01dcee8ffff	0x00e8c2800000	8cca001f-a0d5-4365-a58f-c9b852080a88	0x07	ntfs; Basic d

If one wanted to pull the partition boot data from the 2nd volume in disk 0, one would start at offset 0x500000 (note: there are empty sectors between the 1st volume and 2nd volume and this set of empty sectors shows up as a separate line item in the output above). Then one needs to determine how many bytes are desired. For this example, we will pull the first 0x1000 bytes. Putting this data together into a command, would give the following:

minx -copydrive 0 -offset 0x500000 -size 1x000 < rest of the command>

5.3.2 Imaging an entire drive

If desiring to image an entire drive, one would use the *-copydrive <#>* option *without* specifying either the *-offset* or the *-size* options. To have the results compressed, use the *-gzip* option.

5.3.3 Imaging an entire volume

If desiring to image an entire volume, one would use the **-copyvolume <partition letter>** option *without* specifying either the **-offset** or the **-size** options. To have the results compressed, use the **-gzip** option.

5.3.4 Copying Interesting disk sections

If interested in boot sectors, whether it is from the master boot record or volume boot record, one can use the **-copy_bootfiles** option.

If interested in locations on the drive that may be used by malware for persistence storage, one can use the **-copy_diskgaps <#>** or **-copy_all_diskgaps** options. The first one targets a specific disk, whereas the second pulls the disk gaps from all the drives on the system.

5.4 Running with and without a license file

The **minx** tool can be run with or without the license file. Starting with v.0.27, if one knows the license number used with the **nx** server, one can use that same license number in the **minx** command line to have minx sync up with **nx**. One just needs to append the option **-nx <license number>** to any command used in **minx**.

6 Annotating metadata to the data sent

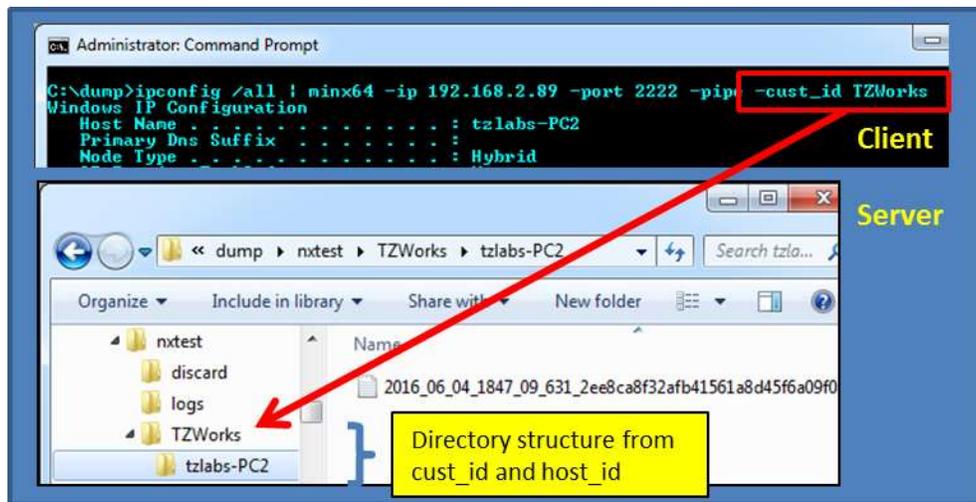
When sending any category of data, **minx** allows one to add extra metadata to each transmission that will then get archived in the log on the server side. This includes: (a) comments, (b) a name to be used as part of the filename of the data archive, (c) customer ID, and (d) host ID. The first option, (a) comments, has the syntax **-comment <data to annotate>** and gets recorded directly in the log on the server for that transaction. The second option, (b) filename, has the syntax **-name <filename>** and gets used as the name recorded when the file is recreated on the server. The last two options are for the customer and host ID. They are special, in the sense that they will create a separate directory structure for each customer and host identifier specified. This is one way to keep artifacts from one target box/customer separated from another target box/customer. The syntax for these are: **-cust_id <customer name or id>** and **-host_id <host name or id>**, respectively. This last pair of options is discussed in more detail below.

6.1 Tagging a **minx** Client with Identifiers

The default **minx** identifier that is sent to the **nx** server is derived from the **minx's** client source IP address and hostname. The source IP address and hostname are packaged by **minx** and sent to the server. To change this behavior, the user can specify that **minx** use some other set of identifiers. There are two optional switches available to do this: **-cust_id <name1>** and **-host_id <name2>**. The **-cust_id** affects the first-tier subdirectory at the **nx** server end and is meant to identify a specific customer. The **-host_id** affects the second-tier subdirectory and is meant to identify a specific host computer at the customer's facility. The only limitation with these options are that each name (or identifier) can only be up to 16 characters each. This flexibility was added along with the tiered directory structure so one

could easily associate the results at the **nx** server end to that of its source, and thus what **minx** client endpoint the data originated.

As an example, we can use **TZWorks** as the customer identifier and leave the host identifier as default (which will then cause **minx** to pull the endpoint's hostname directly from the box). For our test command, the example invokes a simple **ipconfig /all** command and pipes the output into **minx**. The screen snapshots below show the interaction between **minx** and **nx** server. One can see that the customer identifier, **TZWorks**, was created as a first-tier subdirectory and the endpoint hostname was used as the second tier.



7 Using Scripts

The scripting capability allows any or all the functionality of **minx** to be packaged into a script to be acted on in one session. These scripts can either be invoked directly from the command line on the **minx** side, or be stored on the **nx** server side, and when **minx** starts up on the remote computer, it can request any script to be forwarded to it. Once **minx** receives a script from the **nx** server, it will immediately execute it.

7.1 Script File Syntax

The script file is a text-based file that allows one to automate **minx**. The parsing engine used to read the command from a script file is reliable if the rules are followed. There are some nuances, however, that are caused by text editors used in Windows and those used in Unix based operating systems. Windows text editors, for example, will put both `[CRLF]` (two separate characters, 0x0d, 0x0a hexadecimal) at the end of a line, while Unix text editors like to put `[LF]` (one character, 0x0a hexadecimal) for an end of line. The older version of the Mac operating system used to put a `[CR]` (one character, 0x0d hex) for an end of line. Since the **minx** parsing engine tries to parse one line at a time it uses either `[CRLF]` or `[LF]` sequences to determine when the line ends and when a new line starts. Therefore, either the Windows OR Unix format is supported, but not both types of formats within one document. Thus, if a

configuration file is created in Windows with notepad, and then edited in Linux with *gedit* or some other Linux based editor, there can be a mixture of *[CRLF]* and *[LF]* combinations for the EOL (end of line) characters. The caution here is stick to the same editor, when editing a script file, that it was created with.

7.1.1 General Rules

The syntax rules for script files are as follows:

1. Each line is parsed separately.
2. A line that starts with two forward slashes (e.g. *//*) is ignored and used for comments.
3. A blank line is ignored.
4. Any line not satisfying the above Rules 2 and 3 is assumed to be a command.
5. All command lines are in CSV (comma separated value) format. The commas are used to separate the keywords and parameters associated with the keywords.

7.1.2 Command Lines

1. Commands must start with the text sequence: *!cmd*
2. Commands can contain the following keywords; comma delimited (in any order). If a keyword has parameters associated with it, they are also comma delimited. The exception to this is the *<spawn>... </spawn>* phrase. The main keywords are listed below:

- *-ping*
- *-copyfile*
- *-copydir*
- *-copydrive*
- *-copyvolume*
- *<spawn type="pipe_output"> cmdline syntax </spawn>*
- *<spawn type="send_output"> cmdline syntax </spawn>*
- *-name <name to output file>*
- *-comment <any comment you want associated>*
- *-cleanup*

7.1.3 Environment Variables

1. *minx* tries to resolve any environment variables that are passed as part of the command line parameters. Environment variables are surrounded by percent characters (e.g. *%*)
2. *minx* defines two internal, custom environment variables:
 - a. *%userbase%*, which gets resolved to:
 - i. *C:\users* [Vista, Win7 and higher]
 - ii. *C:\Documents and Settings* [pre Vista]
 - b. *%eventlogs%*, which gets resolved to:
 - i. *%systemroot%\System32\winevt\Logs* [Vista, Win7 and higher]

- ii. %systemroot%\System32\config [pre Vista]

Note: When using environment variables, one needs to account for the expansion of the variable to a name that may contain spaces. Therefore, it is recommended to always use quotes around the path/filename that includes an environment variable to avoid problems.

7.2 Using Built-in commands

Below are examples of using some of the **minx** built-in commands within a script file. *Action 1* is a simple custom command with no extra parameters. *Actions 2-4* are custom commands with additional parameters. Notice that each additional parameter is delimited by commas.

Desired Action	Example built-in commands used in script file
1 Ping the server	!cmd, -ping
2 Copy the Volume C: boot record	!cmd, -copy, c:\\$boot
3 Copy the first 0x1000 bytes from hard drive 0	!cmd, -copydrive, 0, -offset, 0, -size, 0x1000
4 Copy the change log journal on the C volume	!cmd, -copy, c:\\$extend\\$usnrnl:\$j
5 Cleanup (remove) the minx app	!cmd, -cleanup

If the above commands were put into a text file, and then invoked by the **-script** option, **minx** would proceed to execute commands 1-4 synchronously. Synchronously, in the sense, the previous command in the script must be completed before the next command in the sequence is executed.

7.3 Spawning third party tools

There are many times when you just want to use one of the operating system's built-in commands, or a 3rd party tool. From a script file perspective, the syntax to do this is **<spawn type="send_output"> [cmd to invoke] </spawn>**. The implicit behavior for **minx** when spawning another tool is to extract any console output from the tool and transport that data to the **nx** server.

Below are some examples of spawning another tool to perform some action:

Desired Action	Example spawn commands used in configuration file
1 Send network configuration data	!cmd, <spawn type="send_output"> ipconfig.exe /all </spawn>, -comment, ipconfig /all
2 Send all open network connections	!cmd, <spawn type="send_output"> netstat -anob </spawn>, -comment, netstat -anob
3 Send the process list	!cmd, <spawn type="send_output"> tasklist /svc </spawn>, -comment, tasklist /svc
4 Send the processed result of the prefetch files	!cmd, <spawn type="send_output"> dir "%systemroot%\prefetch*.pf" /b /s pf.exe -v -pipe </spawn>

5	Send the processed results of the change log journal.	<code>!cmd, <spawn type="send_output"> jp.exe -partition c -v </spawn></code>
6	Send the processed results of all the LNK files	<code>!cmd, <spawn type="send_output"> dir "%usersbase%*.lnk" /b /s lp.exe -csv </spawn></code>

Notice that the command that is *spawned* includes the name of the command and its related arguments. Notice also that that the spawned command is enclosed in the tags `<spawn type="send_output">` and `</spawn>` without using commas to separate the tags or commands, except before and after the `<spawn>` and `</spawn>`, respectively.

The complexity of the command to be spawned is up to the user. One can use environment variables, multiple tools on one command line, etc. The use of the `[<comment, any text can go here]` is strictly used to annotate notes in the *nx* log.

Examples 4 and 6 uses two commands each. Example 4 uses the built-in operating system directory listing, which then gets piped into a 3rd party *TZWorks* prefetch tool (called *pf.exe*). Example 6's directory listing gets piped into the *TZWorks* LNK parsing tool (called *lp.exe*). The result of the each parsing is sent to the server. In example 6, the `%usersbase%` is a *minx* custom environment variable for the Users directory. Note that double quotes were used around the variable and path/filename to avoid errors in processing the command on a Windows XP box. Windows XP would resolve the `%usersbase%` to be `"C:\Documents and Settings*.lnk"` vice in Windows 7 `"C:\users*.lnk"`.

7.4 Enumerating files and copying them

There are 2 methods to enumerate a directory of files for the purpose of passing these same files or a subset of them into a copy command. The first method is the recommended way, and the second is an older way that was left in as an alternative.

7.4.1 Copying a collection of files (method 1 – just using built-in commands)

This method is handled internally to *minx*, and thus does not rely on spawning another tool. It makes use of the `-copydir` command and the companion subcommands/arguments to specify a starting folder, how deep to traverse down, and any filters to discriminate specific files. Below are some examples:

	Desired Action	Example <code>-copydir</code> commands used in configuration file
1	Copy user hives	<code>!cmd, -copydir, %usersbase%, -level, 6, -filter, ntuser.dat* usrclass.dat*, -retain_path</code>
2	Copy LNK & JumpList files	<code>!cmd, -copydir, %usersbase%, -level, 9, -filter, *.lnk *ions-ms, -retain_path</code>
3	Copy event logs	<code>!cmd, -copydir, %eventlogs%, -level, 1, -filter, "*.evtx *.evt"</code>

7.4.2 Copying a collection of files (method 2 – spawning 3rd party tools)

This method uses both a `<spawn> ... </spawn>` command as well as the `-copyfile` keyword. The idea here would be to spawn a directory enumerator and put the results into a temporary buffer. After the enumeration is finished, replay the resulting buffer back performing the next command (in this case copying the file) one by one until the entire listing in the temporary buffer was exhausted.

To invoke this type of behavior, one uses the `<spawn type="pipe_output">` versus the previous example of `<spawn type="send_output">`. The "pipe_output" instruction will spawn the command enclosed in the `<spawn> ... </spawn>` tags and then collect the output of the command into a temporary buffer. The second part would be to specify an action keyword that operates on piped data; this would be `-copyfile`. Below are some examples:

Desired Action	Example <code>spawn</code> & <code>-copyfile</code> commands used in configuration file
1 Copy all ntuser.dat hives	<code>!cmd, <spawn type="pipe_output"> dir "%usersbase%*ntuser.dat" /b /s /a </spawn>, -copyfile, -comment, pull user registry hives</code>
2 Copy all LNK files	<code>!cmd, <spawn type="pipe_output"> dir "%usersbase%*.lnk" /b /s /a </spawn>, -copyfile</code>
3 Copy EVTX event logs	<code>!cmd, <spawn type="pipe_output"> dir "%eventlogs%*.evtx" /b /s /a </spawn>, -copyfile</code>

There are two major advantages of using the previous method compared to this method. The first is that another instance of a separate tool does not need to be spawned. The second is the `-copydir` option has more filtering options, and therefore, more generic statements can be made. Look at Action 3 for the two methods. The first method allows one to target *any* event log type with the script line, while the second needs to be more specific and can target either `evtx` (new eventlog format) or `evt` (older eventlog format) types, but not both.

7.5 Serving out command scripts

This area is still experimental in nature. The idea here is to have all the command scripts located at the `nx` server workstation and serve them out as requested by the `minx` clients. To enable this functionality, one needs to tell the `nx` server where the command scripts are located. This is done when starting up the `nx` server via the option `-scripts <folder containing the command scripts>`.

The second item one needs to account for is the command script names that get placed in the script folder. There are two options available: (a) The first case is straightforward in that the `minx` client explicitly requests a specific script via the `-get_cmds -queryfile <script name>`. In this mode, the `nx` service will look in the script folder and if it finds the name given, will send it back to the `minx` client for processing. The name of the script should follow the syntax: `<script name>.cmds.txt`. The nuance here is the `<script name>` without the `cmds.txt` extension should be used in the `-queryfile` argument. (b) The second case is where the `minx` client issues the `-get_cmds` option without specifying a specific script. In this case, the `nx` service will look in the script folder and search for the following script filename `<cust_id>.cmds.txt`. If the file is found, it will be sent back to the `minx` client for processing. This case

requires more upfront planning, but allows one the flexibility of targeting specific scripts for a class of customer via the `-cust_id <user defined name>` option.

7.5.1 Example of the setup of using `-queryfile <script name>`

Below is a screen shot showing the location of the script file `"test1.cmds.txt"`, which is in the folder `c:\dump\nx\scripts`. This location is passed into the server to identify it as the script repository, via the `-scripts c:\dump\nx\scripts` option. The minx client can then issue a command to invoke this script via the options: `-get_cmds -queryfile "test1"`. These sequences of events are shown below:

Script repository. All scripts should end with the extension `".cmds.txt"`. So in this case, the script is `test1`; what is stored in the script repository is `test1.cmds.txt`.

Setup of the server side to point to the script repository

```
Command Prompt
C:\Users\tzworks>nx64 -server -ip 127.0.0.1 -port 3333 -dir c:\dump\nx -scripts c:\dump\nx\scripts
```

Client side requesting a specific script: Note: script file is actually called `test1.cmds.txt`, but for the `minx` client we just pass in the name `test1` without the extension `'cmds.txt'`

```
Administrator: Command Prompt
C:\dump>minx64 -ip 127.0.0.1 -port 3333 -get_cmds -queryfile test1
```

Server: Actual test [using localhost]

```
Command Prompt - nx64 -server -ip 127.0.0.1 -port 3333 -dir c:\dump\nx -scripts c:\dump\nx\scripts
C:\Users\tzworks>nx64 -server -ip 127.0.0.1 -port 3333 -dir c:\dump\nx -scripts c:\dump\nx\scripts
nx - full ver: 0.29; [protocol ver: 0.08 (nx)]; Copyright (c) TZworks LLC
License #1d44da9ce74daF0 is authenticated for business use and registered to TZworks
run time: 03/15/2019 00:32:22 [UTC]; Host: DESKTOP-M8B00QP; MachineInfo: Windows;10.0.workstation;10.0.1:64
"cmdline: nx64 -server -ip 127.0.0.1 -port 3333 -dir c:\dump\nx -scripts c:\dump\nx\scripts"
server [protocol ver: 0.08 (nx)] listening on: 127.0.0.1, port: 3333, archive dir: c:\dump\nx [zlib]
num clients connected 1
recv'd query from client: 127.0.0.1:DESKTOP-M8B00QP : [1] cmds issued
```

Client: Actual test [using localhost]

```
Administrator: Command Prompt
C:\dump>minx64 -ip 127.0.0.1 -port 3333 -ping
minx - full ver: 0.14; [protocol ver: 0.08 (nx)]; Copyright (c) TZworks LLC
License #1d44da9ce74daF0 is authenticated for business use and registered to TZworks
run time: 03/15/2019 00:32:38 [UTC]; Host: DESKTOP-M8B00QP; MachineInfo: Windows;10.0.workstation;10.0.1:64
"cmdline: minx64 -ip 127.0.0.1 -port 3333 -ping"
client send: 03/15/2019 00:32:38.643 [client sent to server]
server recv: 03/15/2019 00:32:38.695 [server recv'd from client]
client recv: 03/15/2019 00:32:38.695 [client recv'd from server]
C:\dump>minx64 -ip 127.0.0.1 -port 3333 -get_cmds -queryfile test1
minx - full ver: 0.14; [protocol ver: 0.08 (nx)]; Copyright (c) TZworks LLC
License #1d44da9ce74daF0 is authenticated for business use and registered to TZworks
run time: 03/15/2019 00:32:51 [UTC]; Host: DESKTOP-M8B00QP; MachineInfo: Windows;10.0.workstation;10.0.1:64
"cmdline: minx64 -ip 127.0.0.1 -port 3333 -get_cmds -queryfile test1"
recv'd cmd(s) from server
success: C:\WINDOWS\system32\config\SAM
success: C:\WINDOWS\system32\config\SECURITY
success: C:\WINDOWS\system32\config\SOFTWARE
success: C:\WINDOWS\system32\config\SYSTEM
total time running server script: 20.733 secs
```

8 Example of a script file to collect various artifacts

```
// minx - ver: 0.10, [protocol ver: 0.07 (nx)], Copyright (c) TZWorks LLC
// sample script demo
//
!cmd, -comment, Test live collect using various techniques to gather data
!cmd, -ping

// copy the usnjrnl file for the 'C' volume
!cmd, -comment, change log journal
!cmd, -copy, c:\$extend\$usnjrnl:$j, -retain_path

// copy the prefetch files
!cmd, -comment, normal copy of prefetch files
!cmd, -copydir, %systemroot%\prefetch, -level, 1,-filter, *.pf, -retain_path

// copy the event logs
!cmd, -comment, event log raw copies
!cmd, -copydir, %eventlogs%, -level, 1,-filter, *.evt|*.evt, -retain_path

// copy the system hives
!cmd, -comment, system registry hive raw copies [ntfsraw required since this files are locked down]
!cmd, -copydir, %systemroot%\system32\config, -level, 0,-filter, sam|security|system|software, -retain_path

// copy the user hives [ntfsraw required since this files are locked down]
!cmd, -copydir, %usersbase%, -level, 9, -filter, ntuser.dat*|usrclass.dat*, -retain_path

// copy the LNK and JumpList files
!cmd, -comment, LNK files [normal copy]
!cmd, -copydir, %usersbase%, -level, 9, -filter, *.lnk|*ions-ms, -retain_path

// get the disk and volumes mounted to this machine
!cmd, -comment, drive stats
!cmd, -scandrives

// copy some $boot records. The first copies the $boot on a hidden partition
// if it exists. The second copies the $boot on the 'C' drive, which may
// be the second partition, if there is a hidden partition.
!cmd, -comment, $boot raw copies [require ntfsraw copy]
!cmd, -copyfile, c:\$boot, -retain_path

// copy MBR plus extra bytes
!cmd, -comment, MBR w/ extra bytes
!cmd, -copydrive, 0, -offset, 0, -size, 0x1000

// copy the $MFT file
!cmd, -copyfile, c:\$MFT, -retain_path
```

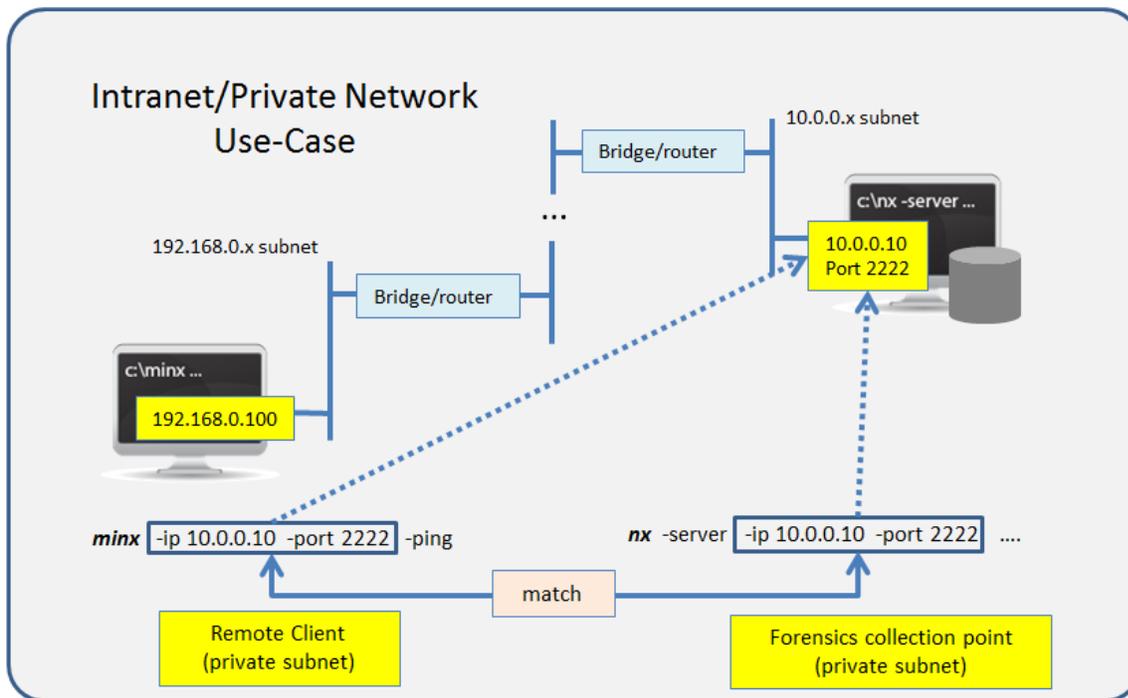
9 Various Use-Case for Transferring Data

As stated earlier, the communications between *minx* and *nx* is what is called peer-to-peer communications. This is defined to be communications between two nodes in contrast to multi-node or broadcast communication. In this case, *minx* must specify the IP address of the *nx* server and the communication is directly between them. One must assume routing will occur, but it should not impact the communication since the *minx* client explicitly specifies the *nx* IP address when running. For peer-to-peer communication, domain credentials are not required to be set up for enterprise networks. As long as *minx* can communicate to the *nx* IP address, without being impacted by firewalls or other network devices that can block IP traffic, the communication should be seamless.

There are many use-cases in setting up *minx/nx*, depending on the network architecture. This section goes over some of the common ones.

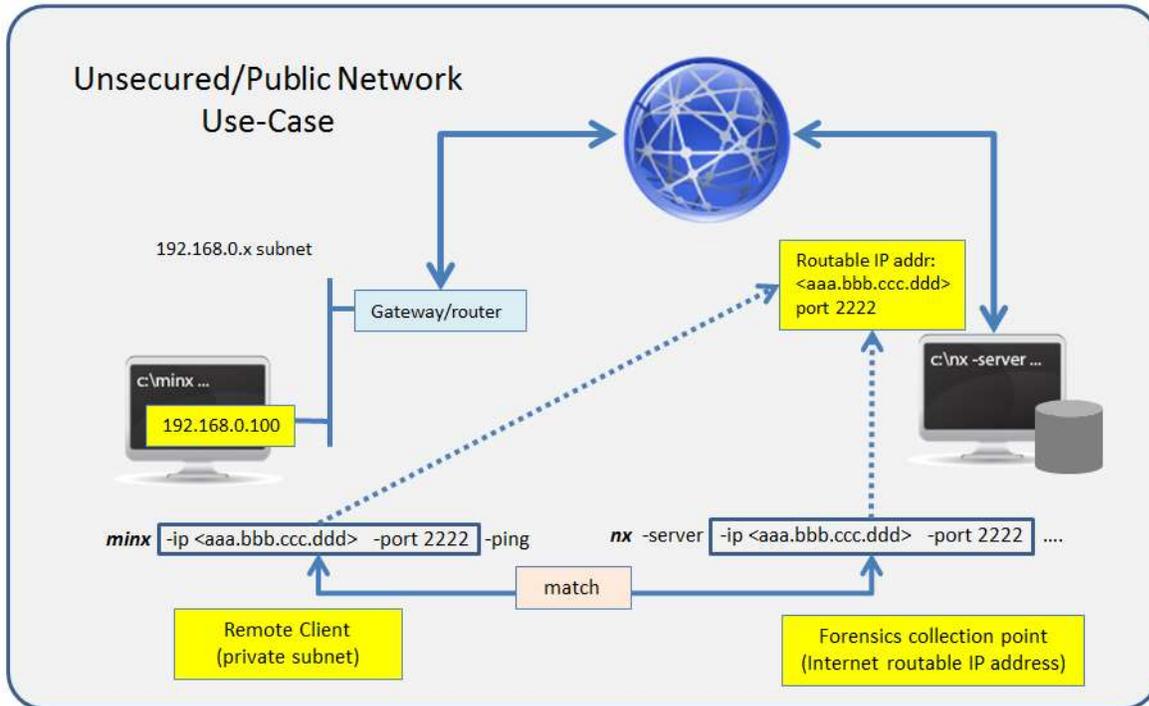
9.1 Normal Private Intranet for both *minx* and *nx*

For the case where the target endpoint and the forensics workstation are on the same private intranet, the IP and port specified on the command line should be equivalent on both *nx* and *minx* as shown below. In the example, the target endpoint (where the data is being collected from, e.g. *minx*) has an IP address of 192.168.0.100 and the forensics workstation (running *nx* in server mode) is at IP address 10.0.0.10. The *nx* service is using port 2222, in this example:



9.2 Sending Data from a Private Intranet to a Public Internet Address

If you have a situation where your forensics collection point is accessible from the Internet, then the setup is not any different than the case discussed above. Both the IP address and port specified in the *minx/nx* setup are the same and will match whatever the *nx* server has specified.



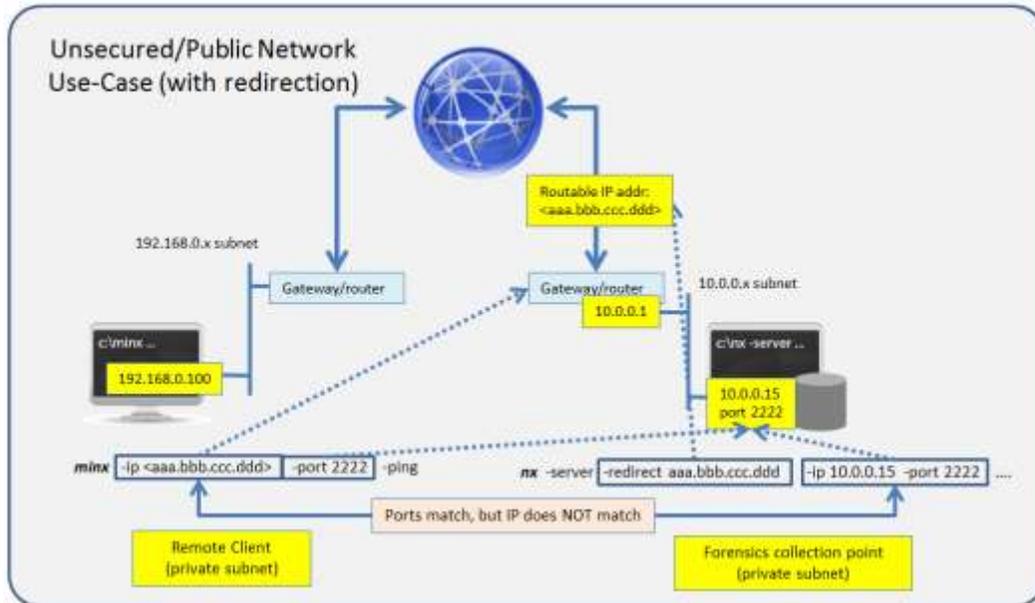
9.3 Using TCP/IP Redirection

These examples require more advanced knowledge and familiarity with setting up redirectors and VPNs.

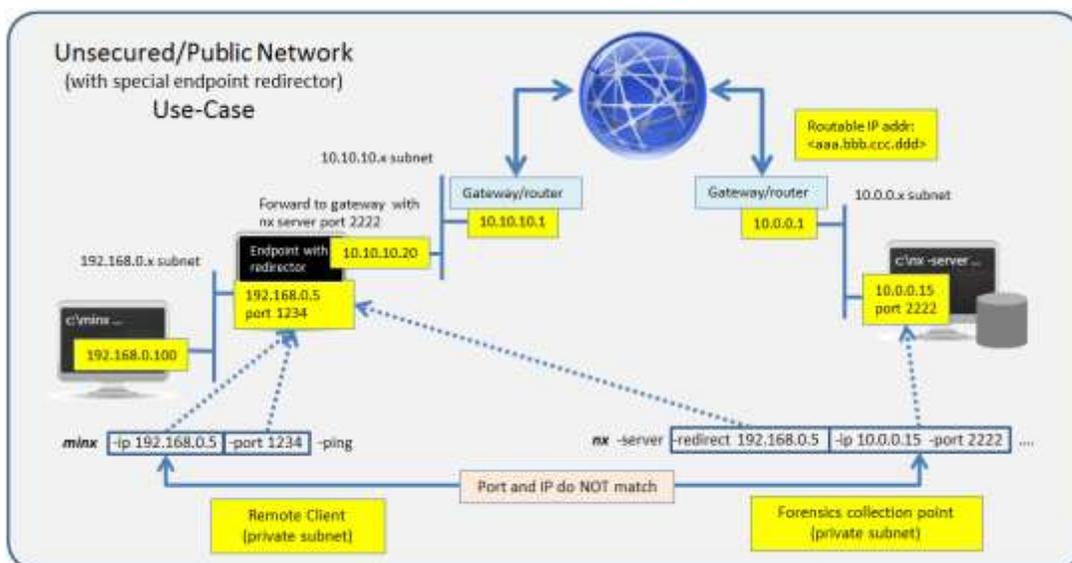
For more complex cases, where one needs to use a redirector to send the packets to the forensics collection point, two cases are shown below. The first one is similar to the previous case with a variation in that the forensics workstation is protected by a firewall/router. Since the IP address of the firewall/router (on the *nx* service side) is the only thing addressable from an *Internet* standpoint then the *minx* client must use this address. The firewall/router needs to be set up to redirect this traffic on the *nx* port (e.g. 2222 in this example) to the forensic workstation.

Normally, for peer-to-peer communications this should work without any other changes; however, for the *minx/nx* architecture, one must specify this redirection on the *nx* server side. To be clear, the redirection only needs to be specified from the *minx* perspective (as opposed to any other case that may happen unbeknownst to user). To do this, when one sets up the *nx* server, one specifies the additional option *-redirect <IP address that minx will be specifying to send data to the server>*. Below is a

diagram with command line arguments for both *minx* and *nx* that shows an example of how to set this up.



The second variation of the redirection use-case is when the target endpoint that is sending data is restricted from doing so because of a firewall or network blocking device. There are a number of solutions that one can implement. The one shown here is to set up an intermediary computer that is accessible to the target endpoint running *minx*. The intermediary computer will employ a redirection service for the internal private network to access the outside network. In this example, the internal private network is on subnet 192.168.0.x. The intermediary computer is connected to both this private network as well as to the 10.10.10.x subnet, so it can effectively route/redirect any packets received on port 1234 on IP address 192.168.0.5 to port 2222 on IP address aaa.bbb.ccc.ddd, as well as handle the reverse redirection. From *minx*'s perspective, it is sending data to IP address 192.168.0.5, port 1234. Doing so will allow the network packets to get to the forensics workstation running the *nx* service. For the *nx* service to accept *minx*'s packets, however, it must be told that a *-redirect* was occurring on 192.168.0.5. These command arguments are shown below:



10 Zlib Dependency

minx makes the use of the **zlib** library. If one is unfamiliar with **zlib**, the official **zlib** website is <https://zlib.net>. It has documentation and details on everything one would ever want to know. The **zlib** library is statically linked into **minx** binary. What this means is the tool is standalone and does not require any external **zlib** shared libraries.

11 Available Options

Option	Description
-ip	IP address of the server. The syntax is: -ip <server IP address> .
-port	Port address of the server. The syntax is: -port <server port address> .
-key	If network encryption is required, this is the symmetric key to use. If used, the same key is needed for both the minx client and nx server. The syntax is: -key <password phrase> .
-nx	If desiring to run minx without a license file, then one can use the license number of the nx server, as follows: -nx <license number>
-ping	Option for the client to check whether the server is up and has

	synchronized time statistics. These statistics are recorded in the server-side logs.
-quiet	Option for minx to not echo to the user what is being sent to the server.
-cust_id	Option for the client to identify itself to the server with a unique client label. This label is associated with all the files sent by this client. The syntax is: -cust_id <label> .
-host_id	Option for the client to identify itself to the server with a unique host label. This label is associated with all the files sent by this client. The syntax is: -host_id <label> .
-pipe	Used to pipe files or commands into the tool via STDIN (standard input). Each file passed in is parsed in sequence.
-copydir	Option to enumerate a directory or subdirectories without using the piping option. The -level sub-option specifies how deep to look in the subdirectories. A value of 0 (which is the default) means just the current directory. The -filter is discussed later as its own separate option. The syntax is: -copydir <dir> -level <#> -filter <*partial* *.ext> .
-name	Option for the client to request the server to use the specified filename to archive the results. The syntax is: -name <filename> .
-comment	Option for the client to cause the server to annotate a comment in the logs stored on the server session side. The syntax is: -comment <"phrase"> .
-filter	Filters data passed in via stdin via the -pipe option or via the -copydir option. The syntax is -filter <"*.ext *partialname* ..."> . The wildcard character '*' is restricted to either before the name or after the name.
-copyfile	Option for the client to copy a specified file and send it to the server. The syntax is: -copyfile <file to copy> . Can also be used to copy more than one file if delimiting the other files with a pipe character: -copyfile <file1 file2 ...> .
-retain_path	Option to store the copied file in the original subdirectory tree as the target. Default (without this option) is to copy all the files in a common

	subdirectory; and files are prepended with a MD5 hash to ensure files with like names are not overwritten.
-scandrives	Enumerates the drives on the host machine and pulls stats, including volume information.
-copydrive	Copies raw bytes from a specified drive. The syntax is: -copydrive <drive #> -offset <start> [-size <#>] .
-script <name>	If one wants to run minx with a script, use this option. The syntax is: -script <script file> .
-get_cmds	Experimental. This instructs minx to contact the nx server and ask if any scripts are available to run. If a script is present on the server side, it will be sent to the minx client and minx will execute it. The optional sub argument allows minx to specify which script file for the nx server to send to it. The syntax is: -get_cmds [-queryfile <script>] .

12 Authentication and the License File

This tool requires an *enterprise* license to run.

This tool has authentication built into the binary. The primary authentication mechanism is the digital X509 code signing certificate embedded into the binary (Windows and macOS).

The other mechanism is the runtime authentication, which applies to all the versions of the tools (Windows, Linux and macOS). The runtime authentication ensures that the tool has a valid license. The license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license.

13 References

1. **zlib** library version 1.2.8, April 28th, 2013, by Jean-loup Gailly and Mark Adler.
2. <http://tools.ietf.org/html/rfc1950> (*zlib* format), rfc1951 (deflate format) and rfc1952 (gzip format)
3. **ntfscopy** – TZWorks NTFS copy utility - https://tzworks.com/prototype_page.php?proto_id=9
4. **dup** – TZWorks Disk Utility & Packer utility - https://tzworks.com/prototype_page.php?proto_id=37
5. **nx** – TZWorks Network eXchange utility - https://tzworks.com/prototype_page.php?proto_id=18