Network Xfer Xtra (nxx) Client/Server Utility

Abstract

nxx enables simultaneous acquisition of raw forensic artifacts on many clients to one or more collection points. All artifacts are securely transported to ensure data integrity and confidentiality. Artifacts are recorded with critical metadata, signatured with unique hashes, and archived in segregated client based directories for ease in follow-on analysis. Each deployment of ***nxx*** on a client machine can be customized with its own script for automated, repeatable collection.

www.tzworks.net

Contact Info: jon@tzworks.net

*Document applies to v0.05 of **nxx***

Updated: March 29, 2012

Table of Contents

1 Introduction

nxx is a tool that has the ability to act as either a client or server for the purposes of transferring data from more than one computer (acting as clients) to a central computer (acting as the server). The server in this case, would be the forensic workstation gathering information during an incident response. The clients would be those computers that are under investigation. ***nxx*** is an enhanced version of the ***nx*** tool. The ***nx*** tool is available for download on the TZWorks LLC website.

There are other tools available that perform the client/server function such as *netcat* and *cryptcat*. ***nxx***, however, focuses on (a) secure collection, (b) archiving the file metadata (c) creating a complete log file of all the transactions that occurred and (d) incorporating a scripting ability to automate a series of commands that can be controlled via a timer. For extracted files, a MD5 hash, the original directory path, and timestamp information are logged.

The client version of ***nxx*** has the following capabilities:

- Windows specific:
  - Perform low level 'live' copying at the cluster and/or NTFS MFT level.
  - Take a screen snapshot and send the image to the server.
- All operating systems that ***nxx*** runs on
  - Copy user accessible files and forward to the server.
  - Spawn any executable and forward the resulting standard output generated by the executable to the ***nxx*** server.
  - Script a collection of commands into a group for automated sequence execution. This also includes the ability to replay a script based on a user defined interval.
  - Run in a mode where the server tells the client what instructions to run.

The ***nxx*** server has the following capabilities (applies to all operating systems that ***nxx*** runs on):

- Use a configuration file to identify which commands to send to clients. These commands can be grouped into collections of commands, where each group can service a specific set of client computers.
- Create a multi-tier subdirectory structure to store all artifacts collected. These tiers are uniquely defined by each of the clients. This allows similar clients to be grouped in the first tier directory. The second tier will have a subdirectory name unique to each client. In this way, all artifacts from a specific client will be contained in its own separate directory.

The transport used in ***nxx*** is TCP/IP with the data content encrypted in a RC4 stream cipher. While this requires a specific crypto key to be stored in the ***nxx*** binary, the key is cryptographically salted with temporal client metadata, so the final key generated is unique. The client temporal metadata is obfuscated in the packet that is sent to the server allowing the server to decrypt the data. For maximum security during network transport, the user is recommended to input a separate passphrase to generate a unique key. To ensure data integrity from the client computer to the final archived file on

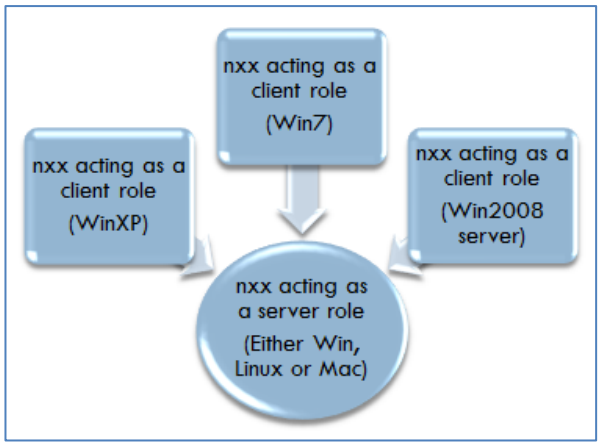
the server, an MD5 hash is computed on the client side before transmission, and again, by the server during its receipt of the data. A mismatch in hashes results in the archive file being labeled as having errors. Addressing ***nxx*** binary integrity, an authentication system checks whether or not the binary has been modified prior to execution.

Currently, ***nxx*** is restricted to IPv4. While there are compiled versions for Windows, Linux and Mac OS-X, only the full set of client/server specifications work on the Windows architecture. A subset of ***nxx*** client and most of the server specifications have been tested on Linux and Mac OS-X, but they are still prototype. The architecture is such that any ***nxx*** client can talk to any ***nxx*** server running on Linux, Windows or Mac OS-X.

2 How to use nxx to complement live forensics collection

The terms 'client and server' are used here as 'roles' for the ***nxx*** tool. Any machine that ***nxx*** runs on does *not* require the operating system to be configured as a client or server. Any normal (non-server operating system) computer configuration will work. All that is required is that there is some network connectivity between the computers.

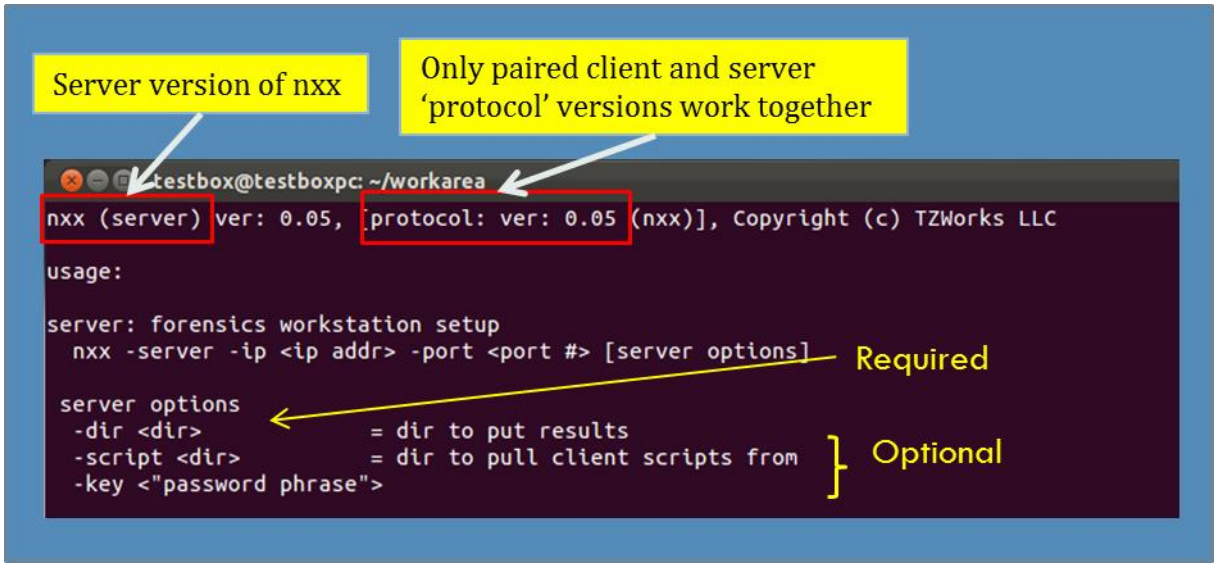
The direction of artifact data flow is always from the client to the server. Although the server can pass data back to the client, the network connection needs to originate from the client. One can think of the clients as those workstations you would like to extract forensics artifacts from, and the server as the single workstation you would like to store the extracted artifacts on. In the server role, ***nxx*** can handle multiple clients at once. Since it was designed as a multi-threaded application, ***nxx*** simply 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 resources of the machine acting in the server role, (b) the amount of data being transferred from each client, and (c) the network bandwidth of the system.



3 Configuring nxx as a server

To configure ***nxx*** in a server role, one uses the ***-server*** command option. Without specifying this option, ***nxx*** runs in the default, client mode. Since ***nxx*** can be configured as one of two roles (client or server), the menu options are broken out accordingly. Secondly, the protocol version number of a client is synchronized with the same protocol version of the server. A mismatch in client and server

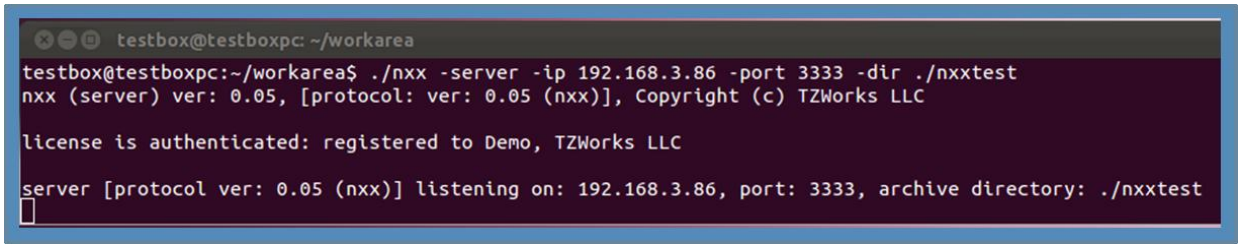
protocol versions will cause the connection between client/server to fail. See picture below where the version number is displayed.



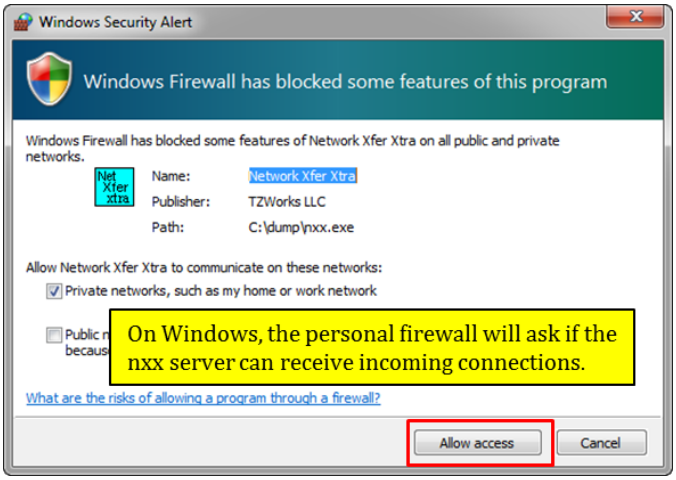
The *required* parameters for the server are: (a) IP address of the server, (b) unique port number to identify how to contact the ***nxx*** service, and (c) the directory to store all the collected artifacts to. The two *optional* parameters are the: ***-key*** parameter and the ***-script*** parameter. The ***-key*** option can be used to set the crypto key to some user defined passphrase. Without using this option, ***nxx*** still encrypts the data transferred over the network (for clients sending data to the server) using its own internal algorithm. As stated earlier, it is not recommended to rely on this encryption scheme by itself. To ensure maximum data protection during network transmission, it is strongly recommended to use the ***-key <user specified password>*** option.

The ***-script*** parameter is used to identify a directory containing configuration files. Each configuration file is associated with a unique client customer identifier. The sections on (a) *Tagging a Client with Identifiers* and (b) *Building a Configuration File* go into depth on this topic.

The screenshot below is an example of setting up the ***nxx*** server on an Ubuntu Linux box. If everything checks out, ***nxx*** will output the IP address and port it is listening on, as well as the directory that it will use to archive data.



Note: If the server was run on a Windows 7 box, the user should get a request from the operating system to allow incoming connections at the port that was specified for the server. Below is the popup when running an ***nxx*** server on Window 7. Without allowing access, the personal firewall will block all ***nxx*** client requests to connect to the ***nxx*** server.

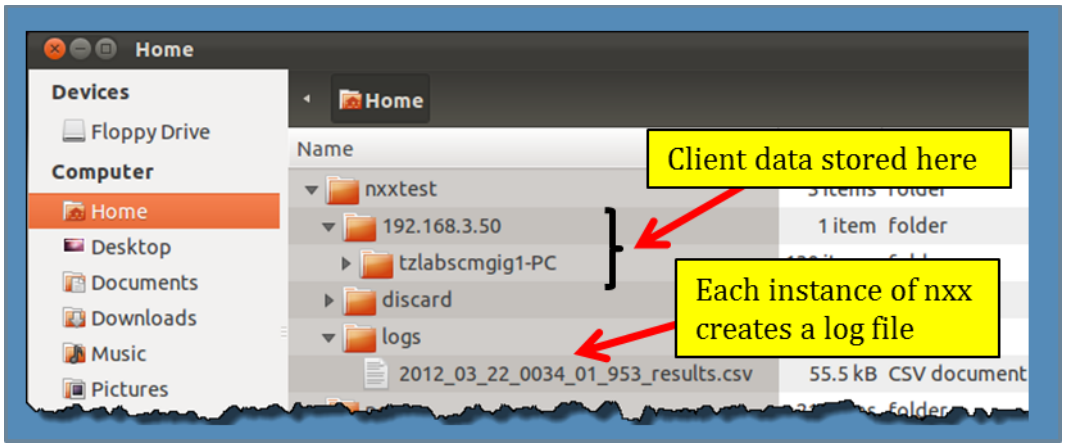


3.1 Server Archive Directory Hierarchy

The server archive directory structure is segregated into various subdirectories starting from the parent directory that was specified during the server setup. The first two subdirectories that get created are the: (a) *discard* and (b) *logs*. The *discard* subdirectory is used as a place where temporary files are created when data is received from clients. Once the complete file is constructed on the server side, the file is renamed, moved from this temporary directory, and placed in the proper client subdirectory. The *logs* subdirectory is the repository for the server logs. The server log is named based on the date and time the server instance started. The log file will continue to grow in size until the ***nxx*** server is stopped.

The second category of subdirectories that is created is strictly controlled by the clients that contact the server. The client identifies to the server what the two level subdirectories should be, and the server dynamically creates these subdirectories. The default mode, assuming the analyst has not given any specific client information (when starting the client), is to use the IP address of the client and the client's

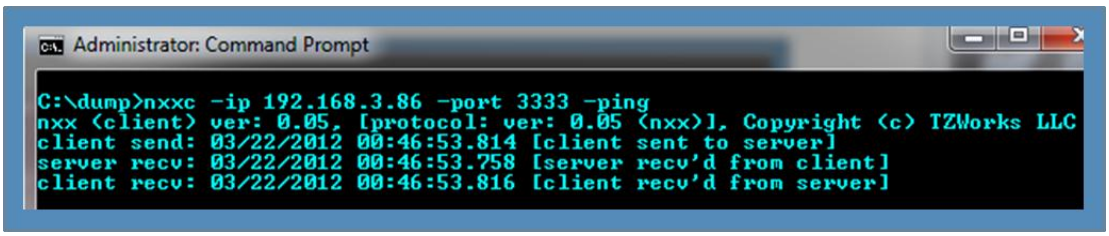
hostname. See the annotated snapshot below of an example directory tree created when one client connects to the server. More information on how to set the client with specific identifying information is located in the section, *Configuring nxx as a Client*.



All metadata information that is received by the server is archived within its log. This log is named '<*start_date_time*>*_results.csv*'. From the extension, one can see, it is formatted as a Comma Separated Value (csv) file. This format was chosen since it is ubiquitous across various platform spreadsheet applications. The only disadvantage with the csv format, however, is that some filenames contain commas, since the comma is a valid filename character on Windows. ***nxx*** handles this by converting every comma in the filename to a space when putting it into the log file. This is something to be aware of when matching names from the log file to that of the original filename.

4 Configuring nxx as a Client

Configuring ***nxx*** in the client role is simply a matter of specifying the same IP address and port number used when configuring the server, and then, issuing the desire command to transfer data. As a test, one should try to 'ping' the server from the client computer using the ***-ping*** option in ***nxx*** (vice the built-in operating system *ping* command). This will ensure the crypto is synchronized between the client and server as well as gather several important statistics. Below is an example of doing this with a Windows 7 computer (for the client) talking to an ***nxx*** server.



First off from the snapshot above, one can see that some timing statistics are displayed to the user. During the ping, the client will send its system time to the server. The server will then respond with the time it received the client's packet. Finally, the client takes the server's response and reports when it received the server's packet. These timing statistics are archived in the server log. With this information, one can later compute the round trip delay, as well as estimate the time mismatch between the server and client system clocks. System time of the client relative to the server is a good metric to archive. The client machine may have a system clock that has been altered, or is just out of sync, with other client computers on the network you are analyzing. For any timeline analysis, knowing what the system clock of the computer you are analyzing relative to a source of time you trust is a good practice.

The second thing that the ***nxx -ping*** command did is not visible in the client's display. Prior to the initial ping packet being sent by the client, the client computed its own MD5 hash of the ***nxx*** binary. This hash is sent along with the initial packet to the server. The server logs this hash value along with the timing statistics. This allows the analyst to check if the version of the ***nxx*** client used is what was expected, or if it was modified. This becomes more important when running the ***nxx*** client in an automated mode, spawned from the task scheduler or other method, on a repetitive basis for some duration of time without user intervention. Below shows how the server log documents these statistics.

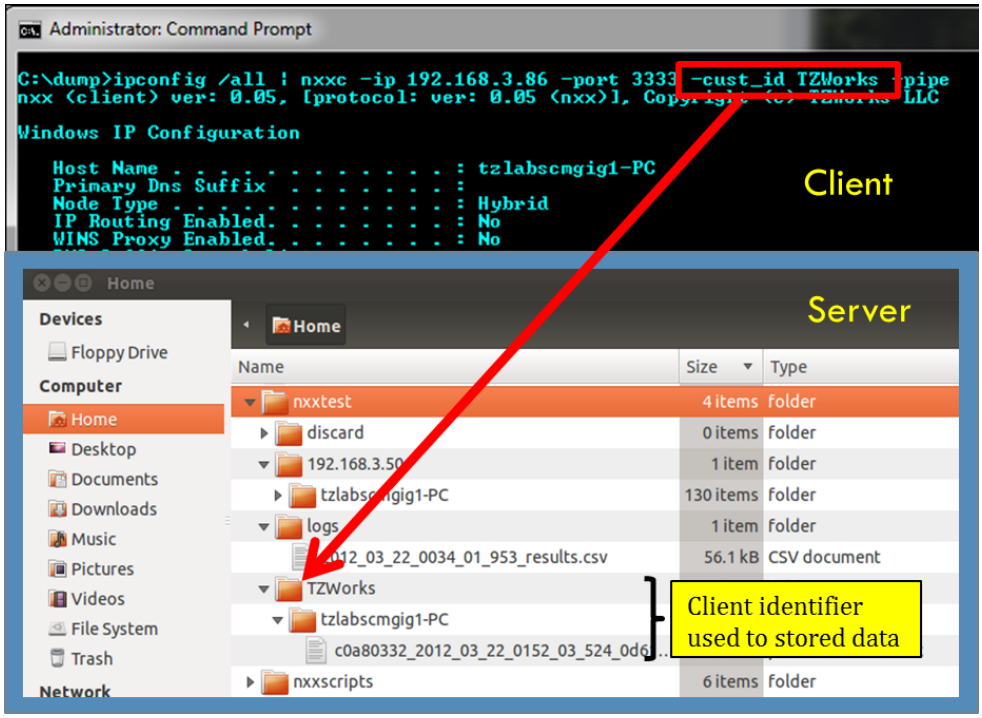
	A	B	C	D	E	H	I
1	Server: 192.168.3.86; Port: 3333						
2	Time started: 03/22/12 00:34:01						
3							
4	status	date	time	client	host	type data	comment
5	success	03/22/2012	00:46:53.819	192.168.3.50	tzlabscmgig1-PC	ping	client send: 03/22/2012 00:46:53.814; server recv: 03/22/2012 00:46:53.758; client recv: 03/22/2012 00:46:53.816; md5 2b35fb72564f1f664149f9b453e033fd

4.1 Tagging a Client with Identifiers

Previously, there was a discussion on how the server sets up the client directories based on the client's identification. This section discusses this in more detail.

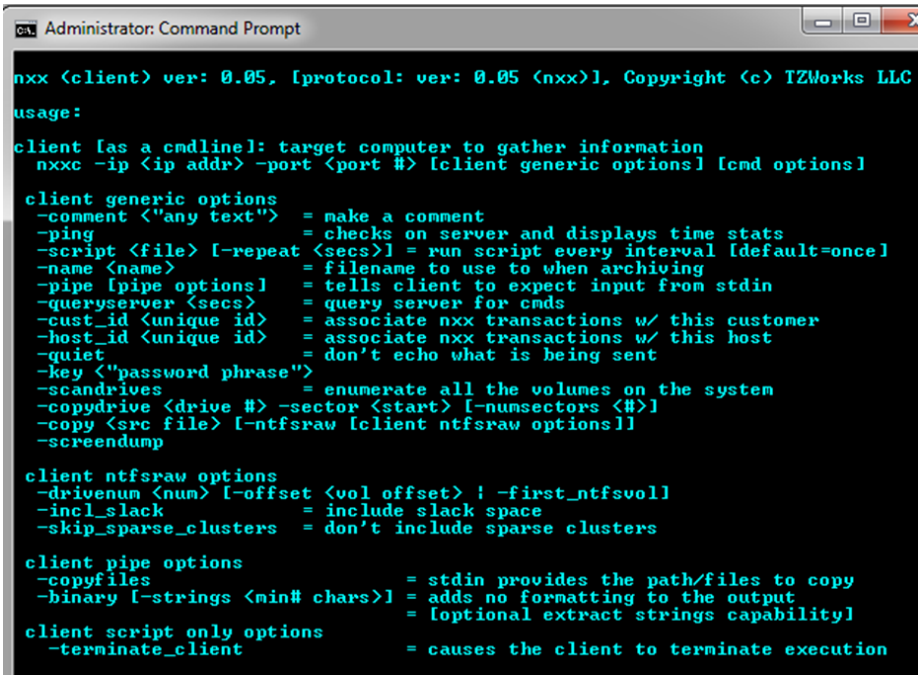
The default client identifier that the server uses is the client's source IP address and hostname. These artifacts are packaged by the client automatically and sent to the server. To change this behavior, the user can specify that the client use some other identifier. There are two optional switches available to do this: ***-cust_id <name1>*** and ***-host_id <name2>***. The ***-cust_id*** affects the first tier subdirectory at the 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 is 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 server end to that of its source.

As an example, we will use *TZWorks* as the customer identifier and leave the host identifier as default. For a 'test' command to send data to the server, the client invokes a simple ***ipconfig /all*** command and pipes the output into ***nxx***. The screen snapshots below show the interaction between the client and server. One can see that the customer identifier, *TZWorks*, was created as a first tier subdirectory and the client's host name was used as the second tier.



5 Sending data to the server

With ***nxx***, there are a number of ways to send artifacts from a client computer to the server. Most of the options are enumerated in the menu displayed from the command prompt, shown below:



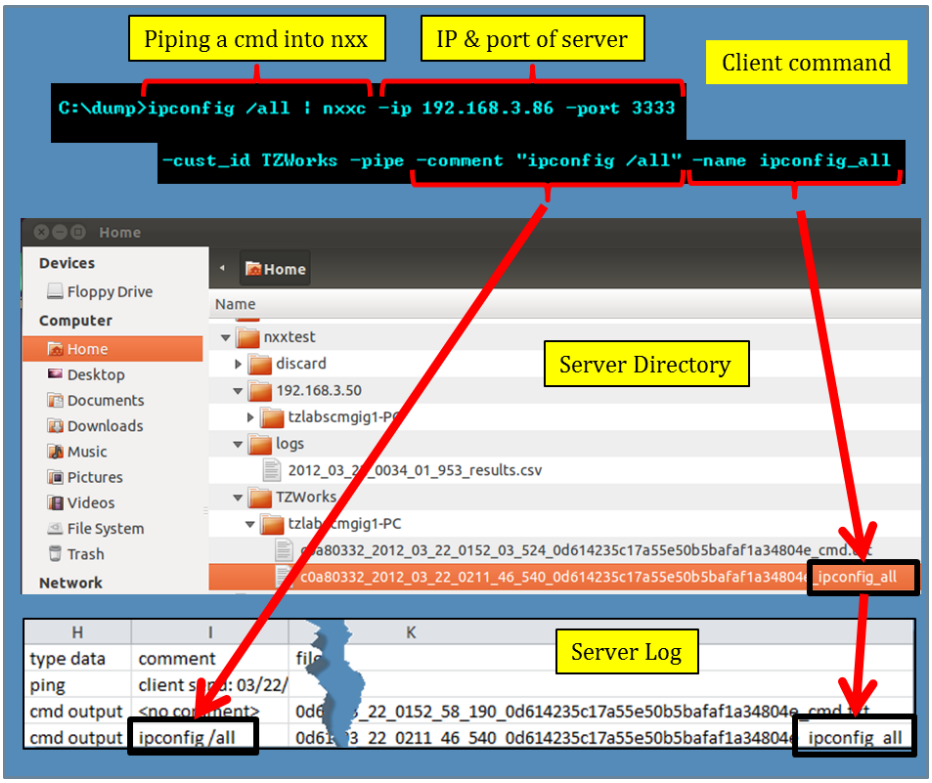
5.1 Running 3rd party commands with the –pipe switch

The most basic command, which has already been shown in a previous example, is the ability to take any console output and pipe it to ***nxx***, whereupon it will be relayed to the server. This is particularly handy if one wants to use a built-in tool from the operating system or some other 3rd party tool that outputs some result to the screen. The ***-pipe*** switch is used to tell ***nxx*** to expect data input from standard input.

5.2 Using –comment and -name switches

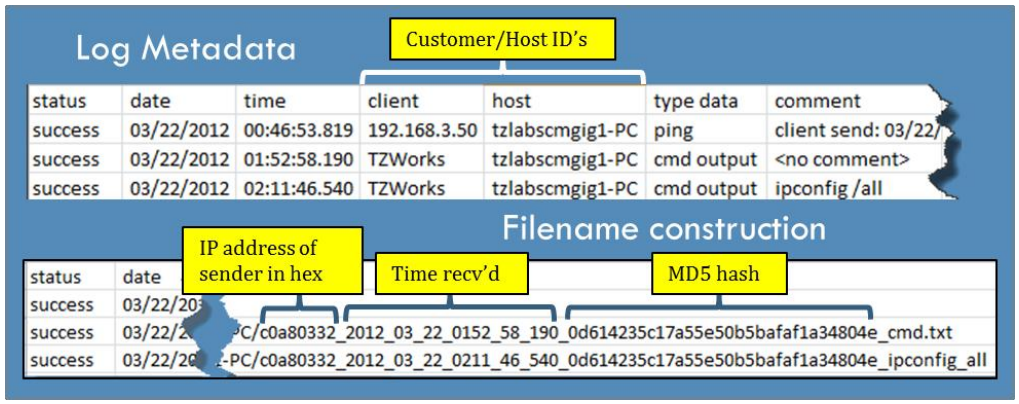
Taking the simple example used before, one can annotate the data passed to the server with a specific comment and name. This is done with the ***-comment <"any text">*** and ***-name <filename to use>*** switches.

From the annotated screen snapshots shown below, one can see the interaction of setting the ***-comment*** and ***-name*** switches on the client end, and how it affects the results on the server end. For the ***-comment***, the data associated with it just gets added to the server log. The ***-name*** parameter is used by the server in the construction of the archived file.



5.3 Name Construction and Server Log Metadata

Name construction of the archived file will consist of: (a) IP address, (b) date/time, (c) md5 hash, and (d) any name that was requested to be used (via the ***-name*** switch). While this makes for long filenames, it solves the problem of identifying where the data came from and what time it was collected. Finally, the name construction will ensure all names are unique, which is important when dumping many files into one subdirectory.



When looking at the other server metadata that gets archived in the log, one can see each transaction that occurred. Metadata such as: status, date, time, source IP, source port, data type, etc. are stored. For those transactions that created a file in the archive directory, an MD5 hash will be computed and documented as well.

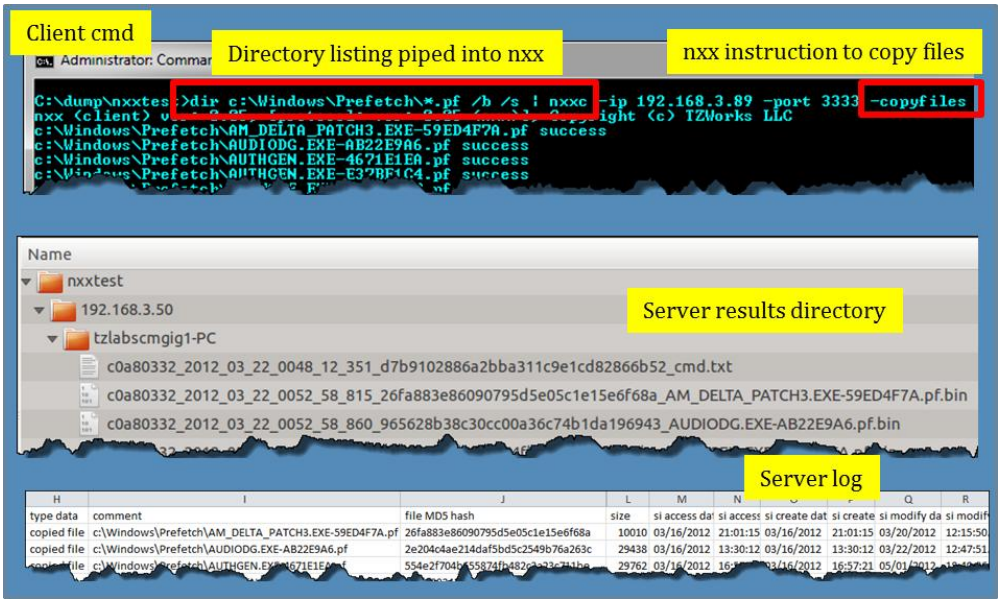
5.4 Carriage Return/Line Feed Issues

While it is not the case with the previous examples, if the situation were reversed, and ***nxx*** was running on a Linux client and the server was on Windows, one would see that the data would be archived with line feeds (LF) for 'end of line' characters. This behavior is normal to Linux/Unix operating systems. Windows, however, would like to see carriage return/line feed (CRLF) pairs. To address this, ***nxx*** has a ***-crlf*** switch which performs this conversion as part of the transmission. Since this is only affected by Linux and Mac OS-X clients, the menu option is only displayed with Linux (and Mac OS-x) ***nxx*** binaries.

5.5 Keying off Directory listings to Copy files

nxx has the ability to take information from standard input and use this information to copy files. In using the built-in operating system directory listing to copy files, one needs to configure the command in two ways.

The first nuance is based on the restriction that the directory listing needs to contain the exact path and filename of the file you wish to copy. For Windows, one can accomplish this via the command ***dir <path> /b /s***. The ***/b*** option tells the directory listing to use bare formatting with no headers. The



/s option tells the directory listing to traverse to lower level subdirectories. The combination of ***/b /s*** forces the directory listing to contain an absolute path with the filename.

The second configuration step tells ***nxx*** that the data received from standard input is really just a bunch of path/filenames that should be copied to the server. This is done via the ***-copyfiles*** switch, which implicitly invokes the ***-pipe*** switch. ***nxx*** will not only copy the files, but it will package each copied file with the appropriate metadata that contains the original source path of the file and the original MACB set of timestamps. Thus, when the server receives the data, it can be annotated in the server log so traceability is retained. The series of snapshots displayed above are an example of copying all the prefetch files from the Windows prefetch directory to the server. While the content of the data is shrunk to fit on the page, one can see that both original path and filename are stored in the server log along with the MACB date and timestamps of the file. Also, the original filename is used in the archive file that is stored the results directory.

5.6 Copying files at the cluster level (Windows version only)

In the previous example, ***nxx*** copied a file from the target computer using standard C/C++ calls to (a) open the file, and (b) read the contents of the file. This is acceptable for most situations. The advantage is that it is fast, is not file-system unique, and is cross platform. The disadvantage with this technique is that it is restricted to only those files that the operating system allows one to read. The second disadvantage is if the system under analysis has been suspected of being compromised, then the files read by traditional means may have suspect integrity when doing a data read. ***nxx***, therefore, has the ability to read any file on an NTFS filesystem at the cluster level.

Cluster level reading of files means that one does not go through the operating system to read a file, but instead traverses the volume at the raw disk level to find the file in question and read the appropriate raw clusters associated with that file. The biggest issues with raw cluster level copying are: (a) it is file system specific, and (b) it is much slower than the normal copy.

Taking into account the speed issues with cluster level file copying, one should use this option sparingly. It should be restricted to a handful of files, such as: registry hives, the pagefile, the $MFT file, the change log journal, etc.

To use the raw cluster reads in ***nxx***, one uses the ***-ntfsraw*** switch in conjunction with the ***-copyfiles*** or ***-copy <file>*** switches. The former is with standard input to identify the path/files to copy and the latter is to explicitly specify which file to copy. Finally, additional metadata is gathered with a raw cluster file copy, such as MFT entry number, standard information MACB timestamps and filename MACB timestamps.

5.6.1 Copying slack space during a file copy (Windows version only)

In forensic analysis, many are interested in the data contained in the slack space of a file. This is space allocated to a file, but has not been used yet. Any data in the slack space would be data that was left there by the previous owner of the cluster.

To perform a copy of a file, and also copy its slack space, one would use the ***-incl_slack*** switch. The resulting file archived at the server will have the slack space appended to the end of the normal file. Therefore, the size reported when using this option will include the slack space (which is different than when the operating system reports size).

5.6.2 Copying without including the sparse clusters (Windows version only)

Windows uses the concept of sparse clusters for reserving a size for a file without actually using clusters. Normally when copying a file, one could care less about the sparse clusters. This is because sparse data is not backed by real clusters, and thus the sparse data would be realized as just zeros

nxx allows one to skip these types of clusters via the ***-skip_sparse_clusters*** switch. This option is important since blindly copying a file, that is possibly sparse, may be much more data than you expected. A useful file in forensics that contains sparse data is the *change log journal*. The size of the change log journal can be significantly smaller when using this option.

5.7 Copying raw disk sectors (Windows version only)

nxx has the capability to pull any sectors off a disk and send it back to the server, via the three switches ***-copydrive <drive#> -sector <start> -numsectors <#>***. Presumably, one could image an entire drive this way, however, it was not intended for this purpose since the network bandwidth required would be staggering. Instead one could pull out the Master Boot Record of the drive, or any sector that was suspected of being used to hide data, if a rootkit was suspected.

5.8 Collecting a Screen Capture (Windows version only)

One of the more interesting items to look at when examining a computer is the screen. Therefore, ***nxx*** has the ability to collect a screen capture via the ***-screendump*** switch. If the computer you are examining has more than one monitor, this option only takes a screen capture of one of the monitors (not both). The file that is generated is a standard bitmap file.

6 Automating Collection through Configuration files

One of the biggest advantages of ***nxx*** over the free ***nx*** tool is the ability to script out a number of commands in the form of ***nxx*** configuration files. The configuration file is a series of steps you instruct the ***nxx*** client to perform. The rules and syntax of building configuration files are discussed in the Section on *Building a Configuration File*.

There are two 'use-cases' for using these configuration files. The first is to have the client run with its local version of the configuration file, and the second is to have the server deliver to the client a configuration file. The syntax for the configuration file is identical between both use-cases. The switches, however, for using these options vary depending on the use-case and are discussed in their respective sections below.

6.1 Push the desired Collection artifacts to the Server

In this scenario the configuration file is pre-built and installed with the ***nxx*** client on the target box. When the ***nxx*** client is invoked with the option, ***-script <config file> [-repeat <#seconds>]***, the client will read the configuration file specified and proceed to execute each command. The optional ***-repeat*** parameter will cause the client to wait for the specified number of seconds before re-running the commands in the configuration file.

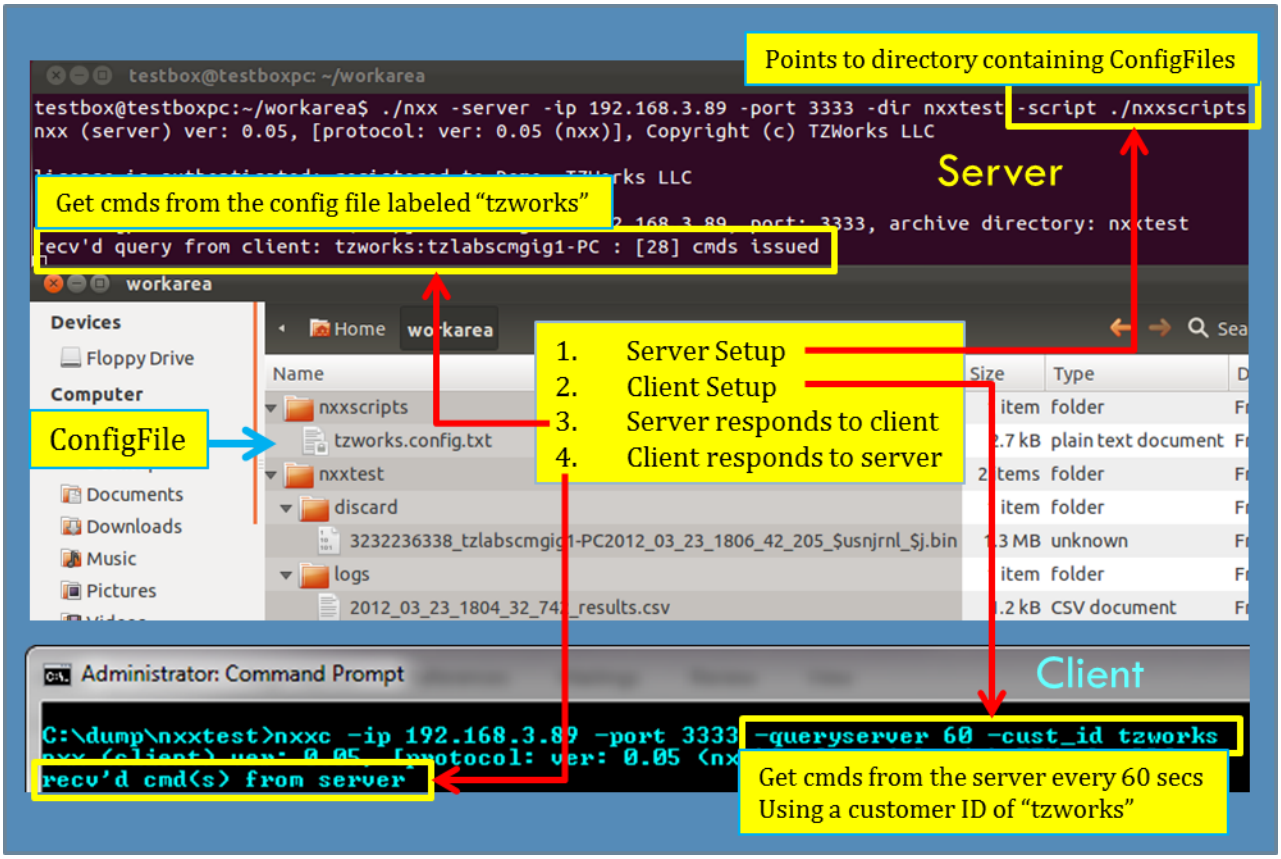
6.2 Pull the type of artifacts from the Server

In this scenario, the configuration file resides on the server side. The client will request from the server to issue a set of commands to be executed. The client syntax to invoke this behavior is ***–queryserver <#seconds>***. The ***nxx*** server needs to be set up as well via the ***-script <path for configuration files>***. On the server side, the configuration files use the following naming convention ***<cust_id>.config.txt***. This convention enables the server to match a specific client ***cust_id*** to a companion configuration file.

Since the steps involved in setting up the server and client can be confusing, the following steps are enumerated and relate to the annotated diagram provided below.

1. Starting with the server setup, one needs to put the configuration file(s) that will be used in a common directory. For this example, the directory is named *nxxscripts,* and the configuration file placed in that directory is called *tzworks.config.txt*. From the naming convention of the configuration files, the keyword is '*tzworks*' that a client needs to use for the server to select the proper configuration file. The server now can be launched with the additional option of ***-server ./nxxscripts***.

2. Now that the server is up and running, the client needs to be configured so that it continuously queries the server and identifies itself with a customer identifier of *tzworks*. This is done via the ***-queryserver <#secs> -cust_id tzworks***. The value of number of seconds selected, for this example, was 60. This means anytime the client is idle for 60 seconds, it will query the server for commands to run.
3. At this point the server has received a request from the client for commands. The server looks at the client's customer identifier to see if there is a companion configuration file in the ***-script*** directory. If it finds one, it is parsed and the commands are sent back to the client.
4. Now the client has received commands from the server and the client proceeds to execute them in the order they were received. Each command is executed synchronously. Therefore, if one command stalls, then the client will stall.



7 Building a configuration file

7.1 Configuration File Syntax

The configuration file is a text based script that allows one to automate ***nxx***. The parsing engine used to read the command from a configuration 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 <CR><LF> (two separate

characters, 0x0d, 0x0a hexadecimal) at the an 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 ***nxx*** parsing engine tries to parse one line at a time it uses either <CR><LF> or <LF> sequences to determine when the line ends and when a new lines starts. Therefore, either the Windows OR Unix format is supported, but not both in one document. Thus if a configuration file is created in Windows with notepad, and then edited in Linux with gedit, there can be a mixture of <CR><LF> and <LF> combinations for end of line characters. The caution here is stick to the same editor when editing a configuration file that it was created with.

The syntax rules for configuration files are as follows:

7.1.1 General Rules

1. Each line is parsed separately.
2. A line that starts with two forward slashes (eg. //) 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 listed in Section 7.1.2.
6. Custom type commands (eg. those built into ***nxx***), use a semicolon to separate custom command parameters.
7. Script files are called *config* files and use the naming convention ***<cust_id>.config.txt***

7.1.2 Command Lines

1. Must start with the sequence: ***!cmd***
2. Can contain the following options, CSV delimited (in any order). The main ones are listed below:
    a. ***!spawn-*** <cmd to execute>
    b. ***!custom-*** <custom cmd>
    c. ***!name-*** <name to output file>
    d. ***!repeat-*** <#secs to wait before executing the cmd again>
        - Note: only one ***!repeat-*** option is allowed per configuration file and it applies to the entire script contained in the configuration file.
    e. ***!comment-*** <any comment you want associated>

7.1.3 Environment Variables

1. ***nxx*** tries to resolve any environment variables that are passed as part of the command line parameters. Environment variables are surrounded by percent characters (eg. *%)*
2. ***nxx*** 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 include an environment variable to avoid problems.

7.2 Built in 'custom' commands (!custom-)

nxx built-in commands are referred to as *custom* commands in the configuration file. To invoke an ***nxx*** built-in command, one uses the ***!custom- <custom cmd>*** syntax. As stated in the Section above on *General Rules,* the custom commands use a semicolon to delimit any extra parameters.

Below are some examples of using ***nxx*** built-in commands to perform some action:

	Desired Action	Example *custom* command used in configuration file
1	Ping the server	*!cmd, !custom- -ping*
2	Take a screen snapshot	*!cmd, !custom- -screendump*
3	Copy the Volume C: boot record	*!cmd, !custom- -copy; c:\$boot; -ntfsraw*
4	Copy the Volume boot record from the 1st drive partition	*!cmd, !custom- -drivenum; 0; -first_ntfsvol; -copy; \$boot; -ntfsraw*
5	Copy the first 200 sectors from hard drive 0	*!cmd, !custom- -copydrive; 0; -sector; 0; -numsectors; 200*
6	Copy the change log journal on the C volume	*!cmd, !custom- -copy; c:\$extend\$usnjrnl:$j; -ntfsraw; -skip_sparse_clusters*
7	Terminate a script	*!cmd, !custom- -terminate_client*

Actions 1 and 2 are simple custom commands with no extra parameters. Actions 3-6 are custom commands with additional parameters. Notice that these additional parameters are delimited by a semicolon.

The last command is unique for a configuration file. If this command is used, then it is always the last command in the script. The ***-terminate_client*** tells the client to terminate its execution and go away. The 'use-case' for this option is for the following scenario: (a) instructions are provided by the server to the client, and (b) the client is spawned as a result of a *cron* or *task scheduler* job. When the client is launched from the *cron* job, the client will seek out instructions from the server and terminate on completion. This cycle is repeated every time the client is spawned. In summary, the ***-terminate_client*** forces a single execution of a script.

If the above commands were put into a text file and then invoked by the client using the ***-script*** option, the client would proceed to execute commands 1-7 in sequence.

7.3 Spawning third party tools to supplement collection (!spawn-)

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 configuration file perspective, the syntax to do this is ***!spawn- <cmd to spawn>***. The implicit behavior for ***nxx*** when spawning another tool is to extract any console output from the tool and transport the data to the server.

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

	Desired Action	Example *spawn* command used in configuration file
1	Send network configuration data	***!cmd, !spawn- ipconfig.exe /all , !comment- "ipconfig /all"***
2	Send all open network connections	***!cmd, !spawn- netstat -anob, !comment- "netstat –anob"***
3	Send the process list	***!cmd, !spawn- tasklist /svc, !comment- "tasklist /svc"***
4	Send the processed result of the prefetch files	***!cmd, !spawn- dir "%systemroot%\prefetch*.pf" / b /s | pf.exe -v***
5	Send the processed results of the change log journal.	***!cmd, !spawn- jp.exe –partition c -v***
6	Send the processed results of all the LNK files	***!cmd, !spawn- dir "%userbase%*.lnk" /b /s | lp.exe -csv***

Notice that the command that is 'spawned' includes the name of the command and its related arguments. Notice also that that there are NO commas except before the ***!spawn-*** keyword and after all the arguments relating to spawn keyword, or the end of line. This is important, since the ***nxx*** script interpreter uses commas to separate keywords, so it is essential that the command that is spawned and its arguments do not have commas in the sequence.

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">*** is used to document in the log which command was used to gather the data. While it is only shown on the first 3 examples, it is generally a good idea to use it for any ***!spawn-*** command that is not specifically tied to a file copy (see next section).

Examples 4 and 6 use two commands each. The first command uses the built-in operating system directory listing which then gets piped into a 3rd party *TZWorks* tool. Example 4's directory listing gets piped into the prefetch parsing tool (called *pf.exe*) and example 6's directory listing gets piped into the LNK parsing tool (called *lp.exe*). The results of the parsing are sent to the server. The ***%userbase%*** is an ***nxx*** custom environment variable that was discussed in the Section on *Environment Variables*. 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 above to be "C:\Documents and Settings*.lnk" vice in Windows 7 "C:\users*.lnk".

7.4 Using a Combination Spawn and Custom Invocation

There are many cases when one may want to use both the ***-spawn*** and the ***-custom*** options as part of one configuration command. The general use-case for using these two options together is (a) to get a directory listing of a particular type of file and (b) then copy the file. The syntax rules discussed in the Section on *General Rules* still apply.

Below are some examples

	Desired Action	Example *spawn & custom* commands used in configuration file
1	Copy all ntuser.dat hives	***!cmd, !spawn- dir "%userbase%*ntuser.dat" /b /s /a:h, !custom- -copyfiles -ntfsraw***
2	Copy all LNK files	***!cmd, !spawn- dir "%userbase%*.lnk" /b /s, !custom- copyfiles***
3	Copy Win7 event logs	***!cmd, !spawn- dir "%eventlogs%*.evtx / b /s, !custom- copyfiles***

7.5 Miscellaneous Configuration File Commands

7.5.1 !comment-

The ***!comment-*** option is the script version of the ***-comment*** option for when running ***nxx*** in command-line mode. It can be used as a standalone command or in combination with any other command. Below are examples.

	Desired Action	Example *spawn & custom* commands used in configuration file
1	Send just a comment to the server log	***!cmd, !comment- This is the start of a new sequence***
2	Use a comment to clarify what command was used	***!cmd, !spawn- netstat -anob, !comment- "netstat –anob"***

7.5.2 !name-

The ***!name-*** option is the script version of the ***-name*** option for when running ***nxx*** in command-line mode. Similar to the command-line, this option is usually used in combination with spawning a command and gives a hint to the server how to name the resultant archive file.

	Desired Action	Example *spawn & custom* commands used in configuration file
1	Use the name when creating the archive file on the server end	***!cmd, !spawn- netstat -anob, !name- "netstat"***

7.5.3 !repeat-

The ***!repeat- <#seconds>*** option can be used in a configuration file to identify how often to repeat the script sequence. The client or server, depending on the reader of the configuration file, uses the value of seconds to pace how often to invoke the script.

Changed from previous versions of ***nxx***, the ***!repeat-*** option, if used, can only have one invocation per configuration file.

8 Example of a collection configuration file

```
// nxx ver: 0.05, [protocol: ver: 0.05 (beta)], Copyright (c) TZWorks LLC
// sample config/script demo
//
!cmd, !comment- Test live collect using various techniques to gather data
!cmd, !custom- -ping

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

// copy the prefetch files
!cmd, !comment- normal copy of prefetch files
!cmd, !spawn- dir %systemroot%\prefetch\*.pf /b /s, !custom- -copyfiles

// dump the screen
!cmd, !custom- -screendump

// copy the event logs
!cmd, !comment- event log raw copies
```

```
!cmd, !spawn- dir %eventlogs%\*.evtx /b /s, !custom- -copyfiles

// copy the system32/config hives
!cmd, !comment- registry hive raw copies [ntfs raw required since this files are locked down]
!cmd, !custom- -copy; "%systemroot%\system32\config\sam"; -ntfsraw, !name- sam_hive.bin
!cmd, !custom- -copy; "%systemroot%\system32\config\security"; -ntfsraw, !name- security_hive.bin
!cmd, !custom- -copy; "%systemroot%\system32\config\system"; -ntfsraw, !name- system_hive.bin
!cmd, !custom- -copy; "%systemroot%\system32\config\software"; -ntfsraw, !name- software_hive.bin
!cmd, !custom- -copy; "%systemroot%\system32\config\components"; -ntfsraw, !name- software_hive.bin

// copy the user hives to retrieve shell bags and other metadata [ntfs raw required since this files are locked down]
!cmd, !spawn- dir "%usersbase%\*ntuser.dat" /b /s /a:h, !custom- -copyfiles; -ntfsraw,
!cmd, !spawn- dir "%usersbase%\*usrclass.dat" /b /s /a:h, !custom- -copyfiles; -ntfsraw,

// copy the index.dat files [normal copy - try both non-hidden and hidden files]
!cmd, !comment- index.dat raw copies
!cmd, !spawn- dir "%usersbase%\*index.dat" /b /s, !custom- -copyfiles;
!cmd, !spawn- dir "%usersbase%\*index.dat" /b /s /a:h, !custom- -copyfiles

// copy the lnk files
!cmd, !comment- lnk files [normal copy - try both non-hidden and hidden files]
!cmd, !spawn- dir "%usersbase%\*.lnk" /b /s, !custom- -copyfiles
!cmd, !spawn- dir "%usersbase%\*.lnk" /b /s /a:h, !custom- -copyfiles

// 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 ntfs raw copy]
!cmd, !custom- -drivenum; 0; -first_ntfsvol; -copy; \$boot; -ntfsraw
!cmd, !custom- -copy; c:\$boot; -ntfsraw

// copy MBR plus extra sectors
!cmd, !comment-MBR w/ extra sector [requires disk raw copy]
!cmd, !custom- -copydrive; 0; -sector; 0; -numsectors; 200
```

9 Authentication and the License File

nxx has authentication built into the binary. There are two authentication mechanisms: (a) the digital certificate embedded into the binary and (b) the runtime authentication. For the first method, only the Windows and Mac OS-X versions have been signed by an X-509 digital code signing certificate, which is

validated by Windows during operation. If the binary has been tampered with, the digital certificate will be invalidated.

For the second (runtime authentication) method, the authentication does two things: (i) validates that ***nxx*** has a current license present and (ii) validates the ***nxx*** binary has not been corrupted. The license needs to be in the same directory of the ***nxx*** binary. Furthermore any modification to the license, either with its name or contents, will invalidate the license. The runtime binary validation hashes the executable that is running and fails the authentication if it detects any modifications.

10 Various nxx Binaries and their Naming Convention

The ***nxx*** binary comes in fifteen different forms. Below are various combinations with their names:

	Architecture	Name	Comment
1	Win 32 bit	nxx.exe	Both client and server in one binary. This 32 bit binary works on Windows 64 bit architectures as well.
2	Win 64 bit	nxx64.exe	Both client and server in one binary. Only works on Windows 64 bit architectures.
3	Win 32 bit	nxxc.exe	Just the client binary. This 32 bit binary works on Windows 64 bit architectures as well.
4	Win 64 bit	nxxc64.exe	Just the client binary. Only works on Windows 64 bit architectures.
5	Win 32 bit	nxxs.exe	Just the server binary. This 32 bit binary works on Windows 64 bit architectures as well.
6	Win 64 bit	nxxs64.exe	Just the server binary. Only works on Windows 64 bit architectures.
7	Linux 32 bit	nxx	Both client and server in one binary.
8	Linux 64 bit	nxx64	Both client and server in one binary. Only works on Windows 64 bit architectures.
9	Linux 32 bit	nxxc	Just the client binary.
10	Linux 64	nxxc64	Just the client binary. Only works on Windows 64 bit architectures.
11	Linux 32	nxxs	Just the server binary.
12	Linux 64	nxxs64	Just the server binary. Only works on Windows 64 bit architectures.
13	Mac OS-X	nxx.mac	Both client and server in one binary. Works on either 32 or 64 bit architectures.
14	Mac OS-X	nxxc.mac	Just the client binary. Works on either 32 or 64 bit architectures.
15	Mac OS-X	nxxs.mac	Just the server binary. Works on either 32 or 64 bit architectures.

11 References

1. Fall 2011, CFRS 660, Network Forensics class at George Mason University
2. ***nx*** from TZWorks, LLC, ref: http://tzworks.net/prototype_page.php?proto_id=18
3. Jones, Bejtlich, Rose, Real Digital Forensics, Computer Security and Incident Response, 200