An introduction to NSM-Console

Matthew Lee Hinman

March 8, 2008

1 Abstract

With the proliferation of dozens of different packet analysis tools, a network traffic analyst has a dizzying amount of tools to choose from to analyze network data. As the number of tools will only increase, a framework to unite and manage each of these tools is necessary. This framework should provide a central and unified way to change the options for each of the tools. My solution to this problem is NSM-Console, or the Network Security Monitoring Console.

2 Introduction

NSM-Console was born from a discussion on IRC when Geek00l and I were discussing the best way to provide an automate-able, simple, and powerful way to combine different network analysis tools into a framework that would appeal to both the inexperienced and experienced analysts. After a large amount of discussion, we decided that a modular framework written in portable code would be the best solution to this problem.

NSM-Console is written entirely in Ruby, which allows for the framework to be as portable as possible, while still being extremely powerful and easy to develop. Since there is no performance requirement, a scripting language works perfectly (Ruby is one of the slower languages compared to alternatives like C, Python or Perl).

Rather than rewriting tools that already exist and work extremely well, NSM-Console is designed to encapsulate different tools (called modules) into a framework, allowing for easy configuration and automation. A module can be created around any executable installed on a system, some examples of modules that have already been created are: tcpdstat, tshark, aimsnarf, chaosreader, foremost, ngrep, tcpflow, argus, clamscan, p0f, bro-ids, pads, tcptrace, capinfos, honeysnap, snort and tcpxtract. NSM-Console was later extended with built-in tools and commands to aid in further analysis of network traffic.

3 NSM-Console modules

In earlier versions, NSM-Console's functionality relied solely on modules. A module is a wrapper around a previously-installed executable that handles command-line configuration as well as execution options. These modules are loaded when the console is started (although a different directory for modules can be specified with the '`modload`' command). When each module is run, the commands specified in the module are performed on the packet capture file. In this way a large amount of tools can be run against a packet capture file in an easily configurable and automated fashion.

A module consists of a number of plain-text files (found in the 'modules' directory in the distribution tarball) inside the following folder hierarchy:

3.1 NSM-Console module files

For a module with the name 'banana', the folder structure is as follows:

```
> modules/
---> banana.module/
------> banana
------> description
------> info
------> defaults
------> <support files>
```

A module is named whatever the directory is called (without the '.module'), so in this example, this would be the 'banana' module. Inside the banana.module directory, NSM-Console expects to find at least 3 files. In addition to these files, any other files (such as configuration files or rules for snort) are included in the directory to make each module as self-contained as possible.

The `banana` file contains newline-separated commands that will be executed when the module is run. This file is expected to be named the same as the module name (if the name of the module was 'rhubarb', NSM-Console would expect to find the 'rhubarb' file in the directory). This file is called the command file.

The next file NSM-Console expects is the `description` file, this file contains a single line describing what the module does. The console uses this to describe the module when all modules are listed. It should be succinct and descriptive. An example for a module using aimsnarf would be: "Extracts AIM conversations from the packet capture file".

Since the description file does not provide a very detailed summary of what the module means and all its options, the `info` file is required. The `info` file is a multi-line plaintext file that normally includes a more indepth description, website address and additional information for how to use a module. For modules that use variables in their command file, it also provides a description of what each variable means (more about this in the variables section).

The `defaults` file, which is optional, provides default values for any of the variables used in the command file. For example, if the variable was `OUTPUT_FILE`, the line in the `defaults` file would looks something like `OUTPUT_FILE=myoutput.txt`. If the `OUTPUT_FILE` variable was used in the command file, it would then be replaced by 'myoutput.txt'. This leads to our discussion of variables.

3.2 Variables

Variables allow an analyst to have easily configurable modules that can have their options set on a per-module basis, as well as inheriting some global options from NSM-Console itself. Each variables is referenced in a bash-like syntax such as `${VARIABLE_NAME}`.

3.2.1 NSM-Console base variables

NSM-Console provides each module the same global options:

```
PCAP_FILE
PCAP_BASE
MODULE_DIR
MODULE_NAME
OUTPUT_DIR
```

The `PCAP_FILE` variable is the full path and filename of the packet capture file that a user specifies to NSM-Console. This variable is used by most modules to set which file to read for analysis. In the case of a directory, NSM-Console will recurse through the directory and attempt to run each module on each file inside.

The `PCAP_BASE` variable is the basename of the packet capture file. This variable is mostly used by modules to determine the name of the output file. For example, if the full path to the packet capture file was /home/analyzt/pcap/data.pcap, the `PCAP_BASE` variable would be data.pcap. This variable is calculated for each individual file at runtime in the case that `PCAP_FILE` is a directory.

The `MODULE_DIR` variable is the name of the currently loaded module directory. Most modules use this to reference module-specific files. The default value of this variable is 'modules'.

The `MODULE_NAME` variable is the name of the currently executing module. Again, most modules use this to reference module-specific files and name their output files. This variable changes for each module.

The `OUTPUT_DIR` variable is the directory that all NSM-Console module output is directed to. The default output directory is `${PCAP_BASE}-output`.

3.2.2 Module variables

Modules may declare their own variables, which are declared in the command file. NSM-Console knows about the variables for each module from the `defaults` file in the module directory. Once, a variable for a module has been declared, it can be set by the user in the console. Variables will be replaced when each module is run.

Variables also support embedding other variables, although only 1 level of lookups are performed at the time of this writing. For example, a module might declare the default variable: `OUTPUT_FILE=${PCAP_BASE}.out` so that the output filename will retain the name of the packet capture file.

4 NSM-Console commands

NSM-Console modules are designed to be managed by a small set of simple commands. While NSM-Console itself offers a large number of commands, this section covers only the most basic usage and most common commands of NSM-Console.

4.1 file

The *file* command is used to tell NSM-Console which packet capture file (or directory of files if a directory is specified) the analysis is going to be performed on. If the file does not exist, the console will warn the user.

4.2 output

The output directory can be specified by using the *output* command. By default the output directory is set to `${PCAP_BASE}-output`.

4.3 info

The *info* command prints additional information about a particular module to the screen (from the module's 'info' file). Normally this would include a detailed description, website, and information about the module's variables.

4.4 list

The *list* command lists all the modules that are currently loaded, as well as their enabled or disabled state. A `[-]` means that a module is disabled, while a `[+]` means that a module is enabled. The description of each module (from the module's description file) is displayed next to the name of each module.

4.5 toggle

The *toggle* command is used to toggle modules on and off, in addition 'all' can be specified to turn all modules on or 'none' can be specified to turn all modules off. Multiple modules can be toggled by using a space-separated list. For example: '`toggle aimsnarf tcpxtract iploc`' will toggle the aimsnarf, tcpxtract and iploc modules.

4.6 options

The *options* command is used to list the global and module-specific variables as well as the commands that will be executed for each module. If no module is specified, the options command shows only the global variables. When a module is specified as an argument, in addition to the global variables, the variables and commands for that module are displayed.

4.7 set

The *set* command is used to set a module variable. The name of the module must be specified as well as the variable name and the new value. For example, setting the Honeysnap's HOST_LIST variable is accomplished by the following command: `set honeysnap HOST_LIST 10.0.0.1,10.0.0.2`. Although there are a large number of variables for all the modules in NSM-Console, the modules are designed to be able to run using the default settings.

4.8 run

Finally, the *run* command tells NSM-Console to perform analysis on the packet capture file using all of the enabled modules. Any module not enabled will be skipped. The output directory as well as directories for each module will be created for any program output if it does not already exist.

5 Additional features of NSM-Console

In addition to the basic commands used to manage the modules in NSM-Console, there are a number of built-in utility and tool commands that are available to make analysis of the packet capture file much easier for the analyst.

5.1 Categories

As the number of modules grows, toggling multiple modules becomes a longer task than I originally hoped for. To solve this problem the categories feature was introduced. Inside the ‘modules’ directory is another directory named ‘categories’. In this directory are a number of text files named after a category of network analysis tools, for example, the `IDS` category file might include the snort and bro-ids modules. This allows the user to toggle the category on and off, which in turn enables and disables all the modules contained inside the category. Categories are designed to be extremely easy to create and modify.

5.2 Shell commands

In order to provide full shell access to an analyst using NSM-Console, the `exec` command was added in order to perform a shell command. For example, ‘`exec bash`’ would start a bash shell, which would return to NSM-Console when exited.

5.3 Ruby evaluation

In addition to providing shell commands, using the `eval` command allows a user to directly evaluate any Ruby statement within NSM-Console. Note that none of the internal variables or functions are hidden from this function, making this an extremely power tool for experienced users. For example, ‘`eval $tabstrings.each { |t| t.puts; }`’ will display all of the strings in the tabstrings array (which is used for tab-completion).

5.4 Packet printing

While the packet printing feature is not as robust as viewing network traffic in wireshark or even tcpdump, the `print` command can serve as a good way to quickly view specific packets when a quick view of packet data is required. The print command supports printing connection information only, as well as displaying the payload data in ascii or hex. For example:

```
nsm> print -x 11
Filename:  /Users/hinmanm/data.pcap
full from 11 to 11
11 1195193092.92338 209.177.146.34 -> 192.168.1.136 TCP 6667 > 50344
Len=128
0010 3a 63 6c 61 72 6b 65 2e 66 72 65 65 6e 6f 64 65 :clarke.freenode
0020 2e 6e 65 74 20 50 4f 4e 47 20 63 6c 61 72 6b 65 .net.PONG.clarke
0030 2e 66 72 65 65 6e 6f 64 65 2e 6e 65 74 20 3a 4c .freenode.net.:L
```

The print command also supports ranges as well as comma-separated values. Some examples of valid ranges are: `1` , `*` , `2-10` and `1,3-10,15-*`

5.5 Payload dumping

While the print command allows a console user to print packet payload information to stdout, the `dump` command allows a user to dump binary payload data into a file. In the event that cryptographic or encoded data is captured, the analyst can perform analysis on the exported payload data with other tools.

The `dump` command supports the same ranges as the `print` command, allowing the user to select which ranges of payloads to dump into a file.

5.6 Encoding and decoding

As a wide variety of network data is not transmitted in plain-text, the `encode` and `decode` commands were introduced to help an analyst quickly translate between different encodings. Here is an example of how these commands are used:

```
nsm> encode base64 NSM-Console is awesome!
```
produces:
```
TlNNLUNvbnNvbGUgaXMgYXdlc29tZSE=
nsm> decode base64 TlNNLUNvbnNvbGUgaXMgYXdlc29tZSE=
```
produces:
```
NSM-Console is awesome!
```

There are a variety of encodings and decodings available currently, with more planned for future releases.

5.7 Aliases

NSM-Console is also designed to allow an analyst to customize their tools as well as tailor their environment to suit them better. Part of the way this is accomplished is through the `alias` command. The `alias` command allows a console user to alias any command to be another command. For example: '`alias ls=list`' will alias the `list` command to '`ls`'. A more complex example would be: '`alias serv = exec cat /etc/services | grep`', which would allow an analyst to use '`serv 1024`' to search for what service runs on port 1024.

5.8 The .nsmcrc file

In order to alleviate typing the same alias and set commands every time NSM-Console is started, NSM-Console will read the `.nsmcrc` file in a user's home directory upon startup, allowed an analyst to specify common aliases and module options to be set when NSM-Console starts.

6 The future of NSM-Console

As NSM-Console continues to grow and mature, it is hoped that it will continue to be a valuable tool to any network security analyst looking for an extensible framework to perform analysis with. Since the project is released under an open-source license, any community feedback or patches are encouraged and greatly appreciated.

7 Additional Information

NSM-Console home page
http://writequit.org/projects/nsm-console

NSM-Console trac page
https://trac.security.org.my/hex/wiki/nsm-console

The HeX LiveCD Project
http://rawpacket.org/projects/hex

HeX LiveCD Trac page
https://trac.security.org.my/hex/wiki