



Interested in learning more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Developing a Snort Dynamic Preprocessor

The goal of this paper is to demonstrate how to create a controlled environment for testing and writing a dynamic preprocessor.

Copyright SANS Institute
Author Retains Full Rights

AD



Developing a Snort Dynamic Preprocessor

GCIH Gold Certification

Author: Daryl Ashley, ashley@infosec.utexas.edu

Adviser: Joey Niem

Accepted: 2008-08-19

1. Introduction	3
2. Intended Audience	4
3. Netcat	5
4. SFSnortPacket Data Structure	6
5. DynamicPreprocessorData Data Structure	7
6. Example Dynamic Preprocessor Configuration	8
7. Example Dynamic Preprocessor Configuration and Testing	11
8. Example Dynamic Preprocessor Header File	14
9. Registering and Initializing the Example Dynamic Preprocessor 16	
10. Example Dynamic Preprocessor Preproc Function	17
11. The alertAdd Function	20
12. Example Application - Looking For a Credit Card Number	21
13. Determining the Purpose of Other DynamicPreprocessorData Functions 22	
14. Conclusion	23
15. References	24
16. Example Code Listing	26

1. Introduction

The goal of this paper is to demonstrate how to create a controlled environment for testing and writing a dynamic preprocessor. Why would it be necessary to write a dynamic preprocessor? Those familiar with Snort know that it is highly configurable. If you look through the Snort User Guide, there are many options that can be used when writing a Snort signature (Sturges, 2008). However, there are some situations where a preprocessor has some advantages.

When Snort receives a packet, it performs a series of operations before the packet is analyzed against signatures in the Snort configuration file. The packet is decoded and processed by preprocessors before being analyzed for rule hits (Scott, Wolfe, and Hayes, 2004). So, a preprocessor can be used to “normalize” traffic. For example, url-encoding is a technique that can be used to obscure HTTP traffic (Skoudis, 2007). Suppose you want to write a snort rule that will generate an alert if a packet contains the credit card number “4444 4444 4444 4444”. A content match rule can be written that looks specifically for this string of characters. But, if the credit card number is url-encoded, it will look like the listing shown below.

```
%34%34%34%34%20%34%34%34%34%20%34%34%34%34%20%34%34%34%34
```

To generate an alert for the url-encoded string, a separate content match rule must be written to detect this sequence of characters. The Http Inspect preprocessor can be used to “normalize” http traffic. It converts the url-encoded string above into the character string normally seen before snort rules are applied to generate alerts (Sturges, 2008). The preprocessing of the packet payload allows

the Snort administrator to write only one rule for a specific http exploit. The administrator no longer needs to worry about different methods an attacker can use to obfuscate the packet payload.

A preprocessor can be developed to perform complex analysis of the packet payload. A conventional snort rule to detect credit card numbers would use a regular expression to detect sequences of sixteen digit characters. This type of rule may produce many false positives. The Luhn algorithm is a simple mathematical algorithm which can be used to detect false positives (2008 July 5). However, a conventional Snort rule cannot be written to perform a mathematical analysis on the packet payload. To reduce false positives, a preprocessor can be written to use the Luhn algorithm to verify a credit card number before generating an alert.

2. Intended Audience

This paper is intended for experienced Snort administrators (those who have compiled, installed, configured, and used Snort). Some familiarity with the C programming language and netcat is also necessary. Netcat will be used to transmit the network packets used to test the preprocessor.

This paper was written while running Snort version 2.8.2.1 on a Gentoo Linux system. Snort was built on the system using the commands shown below.

```
gunzip snort-2.8.2.1.tar.gz
tar -xvf snort-2.8.2.1.tar
cd snort-2.8.2.1
./configure
make
```

```
make install
```

The list below shows the locations of important files and directories. The files will be edited and inspected throughout this paper.

```
snort-2.8.2.1/src - source files for main snort engine
snort-2.8.2.1/src/dynamic-examples/dynamic-preprocessor - source files for
the example dynamic preprocessor
/usr/local/snort/etc - location of snort configuration file
/usr/local/snort/log - directory where log output and alerts are generated
/usr/local/lib/snort_dynamicpreprocessor - directory where the dynamic
preprocessor will be installed
/usr/local/bin/snort - snort executable
```

If you built snort using different configuration options, you will need to determine where everything is located on your system. To verify the version of snort installed on your system, type the command shown below.

```
/usr/local/bin/snort -V
```

3. Netcat

Netcat is a program which is used to read and write data across network connections using the TCP/IP protocol (Giacobbi, 2006). Netcat will be used to set up a listening port on the machine running snort and to transmit data from a client machine to test the dynamic preprocessor. This paper will refer to the machine running snort as the snort host, and the machine used to transmit test data as the

client host.

The following command is used to create a netcat listener on port 61324 of the snort host.

```
nc -l -p 61324
```

The command will hang until it receives input from a client connecting to port 61324.

Data can then be sent from the client host using the command shown below.

```
echo "hello" | nc snort_host_ip 61324
```

“Hello” will be displayed and the command prompt will be returned on the snort host. The listener on the snort host will not be “persistent”. You will need to re-run the command each time you want the snort host to listen for data from the client host.

4. SFSnortPacket Data Structure

When Snort receives network packets, it decodes the various portions of the packet, such as layer 2, 3, and 4 headers and packet payload (Scott, Wolfe, and Hayes, 2004). The decoded information is used to assign values or pointers to a SFSnortPacket data structure (Sturges, 2008). A dynamic preprocessor will have access to the SFSnortPacket data structure (Sturges, 2008), allowing the dynamic preprocessor to inspect the protocol header information as well as the packet payload. A portion of the definition for the SFSnortPacket struct is shown below (Sturges, 2008). The definition includes members such as `payload`, `payload_size`, `src_port`, and `dst_port` as part of the data structure.

The full SFSnortPacket data structure and some related data structures are

available in the Snort User Guide (Sturges, 2008).

```
typedef struct _SFSnortPacket
{
    struct pcap_pkthdr *pcap_header;
    u_int8_t *pkt_data;
    ... Lots of layer 2 stuff
    IPV4Header *ip4_header, *orig_ip4_header;
    u_int32_t ip4_options_length;
    void *ip4_options_data;
    TCPHeader *tcp_header, *orig_tcp_header;
    u_int32_t tcp_options_length;
    void *tcp_options_data;
    UDPHeader *udp_header, *orig_udp_header;
    ICMPHeader *icmp_header, *orig_icmp_header;
    u_int8_t *payload;
    u_int16_t payload_size;
    u_int16_t normalized_payload_size;
    ... some IP stuff
    u_int16_t src_port;
    u_int16_t dst_port;
    ... more stuff
} SFSnortPacket;
```

5. DynamicPreprocessorData Data Structure

The DynamicPreprocessorData data structure can be thought of as the API used to communicate with Snort. Unlike the SFSnortPacket data structure, most of the members of this data structure are functions instead of variables. The functions

can be used to log messages, generate alerts, and perform other tasks. A portion of the data structure's definition is shown below (Sturges, 2008). The full definition for this data structure can be found in the Snort User Guide (Sturges, 2008).

```
typedef struct _DynamicPreprocessorData
{
    ... some variables
    LogMsgFunc logMsg;
    ... some more log/debug functions
    PreprocRegisterFunc registerPreproc;
    ... more functions
    AlertQueueAdd alertAdd;
    ... more stuff
} DynamicPreprocessorData;
```

6. Example Dynamic Preprocessor Configuration

Snort provides a source file for an example dynamic preprocessor. The example preprocessor will be modified throughout this paper. Before making changes to the source code, Snort must be configured to use the example preprocessor.

Change into the library directory (from section 2 above) and use the following command to verify the example dynamic preprocessor has been installed correctly.

```
ls -l *example*
```

If the example preprocessor has been installed correctly, output similar to the listing shown below should be displayed.

```
rw-r--r-- 1 root root 38594 Jun 30 15:46 lib_sfdynamic_preprocessor_example.a
-rwxr-xr-x 1 root root 1064 Jun 30 15:46 lib_sfdynamic_preprocessor_example.la
lrwxrwxrwx 1 root root 43 Jun 30 15:46 lib_sfdynamic_preprocessor_example.so ->
lib_sfdynamic_preprocessor_example.so.0.0.0
lrwxrwxrwx 1 root root 43 Jun 30 15:46 lib_sfdynamic_preprocessor_example.so.0 -
> lib_sfdynamic_preprocessor_example.so.0.0.0
-rwxr-xr-x 1 root root 29309 Jun 30 15:46
lib_sfdynamic_preprocessor_example.so.0.0.0
```

When you start modifying the example source code, you will need to check the timestamps of these libraries to verify that they have been updated after the preprocessor has been recompiled and reinstalled.

To use the example preprocessor, add the following line to the `snort.conf` file at the end of the section labeled “Step #2” .

```
preprocessor dynamic_example: port 61324
```

Use the command shown below to start snort. You will need a terminal that allows you to scroll through the snort output.

```
snort -c /usr/local/snort/etc/snort.conf -l /usr/local/snort/log
```

Snort should display output similar to the listing shown below.

```
--== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file /usr/local/snort/etc/snort.conf
PortVar 'HTTP_PORTS' defined : [ 80 ]
PortVar 'SHELLCODE_PORTS' defined : [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined : [ 1521 ]
Tagged Packet Limit: 256
Loading dynamic engine
/usr/local/lib/snort_dynamicengine/libsf_engine.so... done
    Loading all dynamic preprocessor libs from
/usr/local/lib/snort_dynamicpreprocessor/...
    Loading dynamic preprocessor library libsf_ftptelnet_preproc.so... done
    Loading dynamic preprocessor library libsf_smtp_preproc.so... done
    Loading dynamic preprocessor library libsf_dcerpc_preproc.so... done
    Loading dynamic preprocessor library libsf_dns_preproc.so... done
    Loading dynamic preprocessor library libsf_ssl_preproc.so... done
    Loading dynamic preprocessor library lib_sfdynamic_preprocessor_example.so...
done
    Finished Loading all dynamic preprocessor libs from ...
Example dynamic preprocessor configuration
    Port: 61324
```

The red output shown above was generated by the example preprocessor.

7. Example Dynamic Preprocessor Configuration and Testing

Change into the `snort-2.8.2.1/src/dynamic-examples/dynamic-preprocessor` directory. There should be a file named `spp_example.c`. Open this file with a text editor. The `spp_example.c` file should contain the following declaration.

```
extern DynamicPreprocessorData _dpd;
```

The variable `_dpd` is declared in the `sf_dynamic_preproc_lib.c` file. This is the interface used to communicate with Snort. For example, the `logMsg` function of the `DynamicPreprocessorData` struct can be used as shown below.

```
_dpd.logMsg("Daryl was here...\n");
```

This line of code tells the Snort to display the string "Daryl was here..." to standard output or to the syslog facility if snort is run in daemon mode. To demonstrate this, look for the function `Example_Setup` in the `spp_example.c` file. The `Example_Setup` function is shown below.

```
void ExampleSetup()
{
    _dpd.registerPreproc("dynamic_example", ExampleInit);
    DEBUG_WRAP(_dpd.debugMsg(DEBUG_PLUGIN, "Preprocessor: Example is setup\n"));
}
```

Modify the code block to include the `logMsg` function call.

```
void ExampleSetup()
{
    _dpd.registerPreproc("dynamic_example", ExampleInit);
    _dpd.logMsg("Daryl was here...\n");
}
```

```

_dpdpd.logMsg("Daryl was here...\n");

DEBUG_WRAP(_dpdpd.debugMsg(DEBUG_PLUGIN, "Preprocessor: Example is
setup\n"));
}

```

Recompile and reinstall the dynamic preprocessor by typing the command shown below.

```

make
make install

```

Verify there are no compile errors after running the "make" command. After successfully running the above commands, verify that the timestamps on the example preprocessor libraries have been updated and restart snort. The following should now be displayed.

```

---== Initializing Snort ===---
Initializing Output Plugins!
processor_example.so... done
...
...
Finished Loading all dynamic preprocessor libs from ...
Daryl was here...
Example dynamic preprocessor configuration
Port: 61324

```

The example preprocessor generates alerts when packets with a source or destination port of 61324 are processed by snort. To verify this, create a netcat listener on port 61324 on the snort host and use the client host to send some data.

```
snort_host> nc -l -p 61324
client_host> echo "hello" | nc snort.host 61324
```

Next, look at the snort alert file in the log directory. The alert file will contain text similar to the listing shown below.

```
[**] [256:2:1] example_preprocessor: dest port match [**]
[Priority: 3]
06/30-15:18:57.193524 146.6.97.140:33144 -> 146.6.193.164:61324
TCP TTL:61 TOS:0x0 ID:6034 IpLen:20 DgmLen:60 DF
*****S* Seq: 0x793E1469 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 268895227 0 NOP WS: 2

[**] [256:1:1] example_preprocessor: src port match [**]
[Priority: 3]
06/30-15:18:57.193547 146.6.193.164:61324 -> 146.6.97.140:33144
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0xA27FE395 Ack: 0x793E146A Win: 0x16A0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 581098500 268895227 NOP WS: 5

... more alerts ...
```

The top line of each alert includes "example_preprocessor", the name of the preprocessor. The top line contains [256:1:1] or [256:2:1]. The numbers in the brackets are the generator ID, SID, and SID version (Scott, Wolfe, and Hayes, 2004). There is a short text description describing the type of alert (Scott,

Wolfe, and Hayes, 2004), either "src port match" or "dest port match". The source code for the example preprocessor contains definitions used to generate these fields.

The flags section of each alert explains why so many alerts were generated. An alert was generated for each TCP packet that was transmitted - three alerts for the three way handshake that initializes the TCP connection, an alert for the packet containing the "hello" string, the ACK packet that acknowledges receipt of the packet containing the "hello" string, and several FIN packets associated with the tear down of the TCP connection.

8. Example Dynamic Preprocessor Header File

Use a text editor to open the sf_preproc_info.h header file. This file is included in the spp_example.c file. A portion of the header file is shown below.

```
#define MAJOR_VERSION 1
#define MINOR_VERSION 0
#define BUILD_VERSION 1
#define PREPROC_NAME "SF_Dynamic_Example_Preprocessor"

#define DYNAMIC_PREPROC_SETUP ExampleSetup
extern void ExampleSetup();
```

The major, minor, and build versions for the preprocessor are defined in this file. This information is displayed as part of the output when snort is started. Each preprocessor object loaded by snort is displayed after the initialization of snort is complete.

```

--== Initialization Complete == --

,,_    -*> Snort! <*-
o" )~  Version 2.8.2.1 (Build 16)
'''    By Martin Roesch & The Snort Team: http://www.snort.org/team.html
        (C) Copyright 1998-2008 Sourcefire Inc., et al.
        Using PCRE version: 7.6 2008-01-28

        Rules Engine: SF_SNORT_DETECTION_ENGINE Version 1.8 < Build 14>
        Preprocessor Object: SF_Dynamic_Example_Preprocessor Version 1.0 <Build 1>

```

The output includes major and minor version numbers along with a build number.

The `#define DYNAMIC_PREPROC_SETUP` statement must be set to the name of the function that will be used to register the dynamic preprocessor with snort. There should also be an external declaration of this function in the header file. The source file `spp_example.c` contains a function called `ExampleSetup` that includes the code shown below.

```
_dpd.registerPreproc("dynamic_example", ExampleInit);
```

This line of code registers the example preprocessor with Snort.

As an exercise, change the major version number in the `sf_preproc_info.h` header file to 2. You may need to run the command below before the preprocessor will be recompiled.

```
touch spp_example.c
```

After the example preprocessor has been recompiled and reinstalled, restart Snort.

The version number of the example preprocessor in the Snort output should now be 2.

9. Registering and Initializing the Example Dynamic Preprocessor

The following line of code is used to register the dynamic preprocessor.

```
_dpd.registerPreproc("dynamic_example", ExampleInit);
```

The registerPreproc function takes two arguments. The first argument, "dynamic_example", is a unique string used to identify the preprocessor. This string was entered in the snort.conf file when configuring snort to use the preprocessor. The line added to the snort.conf file is shown below.

```
preprocessor dynamic_example: port 61324
```

If "plain_old_example" was used as the first argument to the registerPreproc function, the following snort configuration line must be used.

```
preprocessor plain_old_example: port 61324
```

As an exercise, change the first argument of the registerPreproc function to "plain_old_example". Recompile and reinstall the example preprocessor and restart snort. What kind of error messages does Snort display? Next, change the configuration file to use the new preprocessor name and restart Snort. Does the error message go away?

The second argument to the registerPreproc function is a pointer to an initialization function. The spp_example.c file contains a function called ExampleInit. The address of the ExampleInit function is passed to the registerPreproc function as its second parameter.

The initialization function is used to process options in the configuration file that affect how the preprocessor analyzes packets. The function receives a pointer to a character string as a function argument. The option in the snort configuration file for the example preprocessor is “port 61324”. Snort passes the string “port 61324” as an argument to the Init function. The Init function then uses this configuration option to initialize variables which allow the preprocessor to generate alerts only for packets with a source or destination port of 61324.

The initialization function has one other purpose, to register a Preprocessor function. The preprocessor function is the function that will analyze the packet and decide whether or not to generate an alert. The following line of code registers ExampleProcess as the preprocessor function responsible for performing this task.

```
_dpd.addPreproc(ExampleProcess, PRIORITY_TRANSPORT, 10000);
```

This paper will not address the other two parameters that were passed to this function.

10. Example Dynamic Preprocessor Preproc Function

The spp_example.c file contains a function called ExampleProcess. The argument passed to this function is a void pointer. This pointer is actually a pointer to a SFSnortPacket object. The following line of code can be used to access the contents of this object.

```
SFSnortPacket *p = (SFSnortPacket *)pkt;
```

The definition of the SFSnortPacket data structure includes a member called payload. The packet payload can be accessed through this member as shown in the

following line of code.

```
p->payload
```

As an exercise, modify the ExampleProcess function as shown below (additions are shown in red). The code copies up to 11 bytes from the payload member of the SFSnortPacket data structure to a temporary buffer, and uses the `_dpd.logMsg` function to display the characters as part of the Snort output.

```
void ExampleProcess(void *pkt, void *context)
{
    SFSnortPacket *p = (SFSnortPacket *)pkt;

    char tmp[12];

    bzero(tmp, 12);

    int length = 11;

    if (length > p->payload_size)
        length = p->payload_size;

    if (!p->ip4_header || p->ip4_header->proto != IPPROTO_TCP || !p->tcp_header)
    {
        /* Not for me, return */

        return;
    }
}
```

```
if (p->src_port == portToCheck)
{
    if (length > 0) {
        _dpd.logMsg("Copying %i bytes of packet payload into buffer\n",
length);
        strncpy(tmp, (const char *) p->payload, length);
        _dpd.logMsg("Payload data: %s\n", tmp);
    }

    /* Source port matched, log alert */
    _dpd.alertAdd(GENERATOR_EXAMPLE, SRC_PORT_MATCH,
        1, 0, 3, SRC_PORT_MATCH_STR, 0);

    return;
}

if (p->dst_port == portToCheck)
{
    if (length > 0) {
        _dpd.logMsg("Copying %i bytes of packet payload into buffer\n",
length);
        strncpy(tmp, (const char *) p->payload, length);
```

```

        _dpd.logMsg("Payload data: %s\n", tmp);
    }

    /* Destination port matched, log alert */
    _dpd.alertAdd(GENERATOR_EXAMPLE, DST_PORT_MATCH,
                 1, 0, 3, DST_PORT_MATCH_STR, 0);

    return;
}
}

```

Recompile and reinstall the preprocessor. Restart Snort, and use netcat to send a packet containing “hello” to the snort host. After the packet is received by the Snort host, the packet data should be displayed as part of the output of the snort command.

11. The alertAdd Function

The example preprocessor looks at `p->src_port` and `p->dst_port` to determine if the packet’s source or destination port matches 61324. If so, it uses the `alertAdd` function to generate an alert. The `alertAdd` function takes a number of arguments that are used to generate the alert information in the alert log. The arguments are as follows: Generator ID, SID, Revision number for the rule, classification number, priority, a message, and rule info.

The generator ID, SID, and message in the example preprocessor are defined in `#define` statements toward the top of the `spp_example.c` file. The following

information was generated at the top of one of the alerts within the alert file.

```
[**] [256:2:1] example_preprocessor: dest port match [**]
[Priority: 3]
```

GENERATOR_EXAMPLE, defined at the top of the spp_example.c file as 256, is passed as the generator ID to the alertAdd function. This is the 256 in the brackets on the top line of the alert. DST_PORT_MATCH is defined as 2, and is passed as the SID to the alertLog function. This is the “2” in the brackets next to the 256. DST_PORT_MATCH_STR was defined as “example_preprocessor: dest port match” in spp_example.c. This is the string displayed in the first line as a description of the alert.

As an exercise, change the GENERATOR_EXAMPLE, DST_PORT_MATCH, and DST_PORT_MATCH_STR values and recompile and reinstall the example preprocessor. Restart Snort, and use netcat to transmit data to the snort host. The new values should be used to generate alerts in the alert log.

12. Example Application - Looking For a Credit Card Number

As an exercise, add code to the example preprocessor that looks for a specific credit card number, “4444 4444 4444 4444”. The strcmp function can be used to detect this string. Use a netcat listener on the snort host to listen for data on port 61324 and to send test data from the client host. Some of the test data should contain the above credit card number and some should not.

The example preprocessor should be modified so alerts are generated only if the credit card number above is transmitted in a TCP connection on port 61324. The SID and alert message should also be modified so there is an easily identifiable alert in the log file. Remember, the Preprocessor function is passed a pointer to

a SFSnortPacket data structure. The payload and payload_size members of this data structure can be used to examine the contents of the packet.

13. Determining the Purpose of Other DynamicPreprocessorData Functions

The definition of the DynamicPreprocessorData struct in the Snort User Guide contains a number of functions (Sturges, 2008) that were not described in this paper. Unfortunately, the Snort User Guide does not provide a lot of documentation for the functions. If you are interested in looking at source code, this section should help you start determining the purpose of each of the functions.

Change into the snort-2.8.2.1/src/dynamic-plugins and use a text editor to open the file sf_dynamic_plugins.c. Look for the InitDynamicPreprocessors function. The variable preprocData is a DynamicPreprocessorData struct, and the function members are assigned pointers to functions that will be executed when the _dpd functions are called in the preprocessor. For example, the following line assigns a pointer to the actual logMsg function.

```
preprocData.logMsg = &LogMessage;
```

The function that is executed when you make a call to _dpd.logMsg is LogMessage.

Change into the snort-2.8.2.1/src directory and use the following command to search for this text in the other source files:

```
grep LogMessage *.c | less
```

This command will display the lines of the source files in which this function is called. This command should also display what looks like a function declaration in

the output for the `util.c` file. The source file `util.c` contains a `LogMessage` function. The code block for this function outputs messages to standard output if snort is not run in daemon mode and outputs to `syslog` if run in daemon mode (as well as not outputting at all if snort is run in quiet mode with the `-q` command line flag). This is a difficult way to figure out what the different `DynamicPreprocessorData` functions do. But, it is an available option if you are willing to read source code.

14. Conclusion

Certain tasks are very difficult, if not impossible, to accomplish with snort signatures. Dynamic preprocessors can be written to perform very complex tasks because custom code is being written that will interface with snort. This paper has demonstrated how netcat and the example preprocessor provided by snort can be used to start writing a dynamic preprocessor. Once a stable development environment is in place, you should be able to start writing a preprocessor that will fit whatever needs you may have.

15. References

1. Giacobbi, G (2006, November 1). The GNU Netcat project. Retrieved July 7, 2008, from The GNU Netcat -- Official homepage Web site:
<http://netcat.sourceforge.net/>
2. Scott, C, Wolfe, P, & Hayes, B (2004). *Snort for Dummies*. Hoboken: Wiley Publishing Inc.
3. Skoudis, E (2007). *Hacker Techniques, Exploits, and Incident Handling*. Bethesda, MD: The SANS Institute.
4. Sturges, S (2008 May 28). SnortTMUsers Manual 2.8.2. Retrieved July 7, 2008, from Snort - the de facto standard for intrusion detection/prevention Web site: http://www.snort.org/docs/snort_htmanuals/htmanual_282/
5. (2008 July 5). Luhn algorithm - Wikipedia, the free encyclopedia. Retrieved July 7, 2008, from Wikipedia, the Free Encyclopedia Web site:
http://en.wikipedia.org/wiki/Luhn_algorithm

16. Example Code Listing

```
#define SRC_CCN_MATCH 3

#define SRC_CCN_MATCH_STR "example_preprocessor: source CCN match"

#define DST_CCN_MATCH 4

#define DST_CCN_MATCH_STR "example_preprocessor: dest CCN match"

#define SRCH_STRING "4444 4444 4444 4444"

void ExampleProcess(void *pkt, void *context)
{
    SFSnortPacket *p = (SFSnortPacket *)pkt;

    int i, result;

    if (!p->ip4_header || p->ip4_header->proto != IPPROTO_TCP || !p->tcp_header)
    {
        /* Not for me, return */
        return;
    }

    if (p->src_port == portToCheck)
```

```

{
    char *ptr = (char *) p->payload;

    for (i = 0; i < (p->payload_size - 19); i++) {
        result = strncmp(&ptr[i], SRCH_STRING, 19);

        if (result == 0) {
            _dpd.logMsg("CCN found in outgoing traffic");

            /* Source port matched, log alert */
            _dpd.alertAdd(GENERATOR_EXAMPLE, SRC_CCN_MATCH,
                1, 0, 3, SRC_CCN_MATCH_STR, 0);

            return;
        }
    }
}

if (p->dst_port == portToCheck)
{
    char *ptr = (char *) p->payload;

    for (i = 0; i < (p->payload_size - 19); i++) {
        result = strncmp(&ptr[i], SRCH_STRING, 19);

        if (result == 0) {

```

```
_dpd.logMsg("CCN found in incoming traffic");  
  
/* Destination port matched, log alert */  
  
_dpd.alertAdd(GENERATOR_EXAMPLE, DST_CCN_MATCH,  
              1, 0, 3, DST_CCN_MATCH_STR, 0);  
  
return;  
}  
  
}  
  
}  
  
}
```



Upcoming SANS Training

[Click Here for a full list of all Upcoming SANS Events by Location](#)

SANS Singapore 2009	Singapore, Singapore	Jul 06, 2009 - Jul 11, 2009	Live Event
SANS Rocky Mountain 2009	Denver, CO	Jul 07, 2009 - Jul 13, 2009	Live Event
SANS SOS London 2009	London, United Kingdom	Jul 13, 2009 - Jul 18, 2009	Live Event
SANS Future Visions 2009 Tokyo	Tokyo, Japan	Jul 15, 2009 - Jul 17, 2009	Live Event
SANS IMPACT 2009	Kuala Lumpur, Malaysia	Jul 27, 2009 - Aug 01, 2009	Live Event
SANS SEC563: Mobile Device Forensics Debut	Baltimore, MD	Jul 27, 2009 - Jul 31, 2009	Live Event
SANS Boston 2009	Boston, MA	Aug 02, 2009 - Aug 09, 2009	Live Event
SANS Atlanta 2009	Atlanta, GA	Aug 17, 2009 - Aug 28, 2009	Live Event
SANS WhatWorks in Virtualization and Cloud Computing Security Summit 2009	Washington, DC	Aug 17, 2009 - Aug 21, 2009	Live Event
SANS Virginia Beach 2009	Virginia Beach, VA	Aug 28, 2009 - Sep 04, 2009	Live Event
SANS SCDP SEC556: Comprehensive Packet Analysis - Sept. 2009	Ottawa, ON	Sep 09, 2009 - Sep 10, 2009	Live Event
SANS Critical Infrastructure Protection at Oceania CACS2009	Canberra, Australia	Sep 10, 2009 - Sep 11, 2009	Live Event
SANS Network Security 2009	San Diego, CA	Sep 14, 2009 - Sep 22, 2009	Live Event
SANS SCDP Cutting Edge Hacking Techniques - June 2009	Ottawa, ON	Sep 15, 2009 - Sep 15, 2009	Live Event
SANS WhatWorks Summit in Forensics and Incident Response	OnlineDC	Jul 06, 2009 - Jul 14, 2009	Live Event
SANS OnDemand	Books & MP3s Only	Anytime	Self Paced