Detecting NUSHU Covert Channels Using Neural Networks

Eugene Tumoian
Maxim Anikeev
Taganrog State University of Radio Engineering
Department of Information Security
Lab. I-409, ul. Chekhova, 2, Taganrog, Russia
{tumoyan, anikeev}@users.tsure.ru

Abstract

A method of NUSHU covert channel detection based on neural networks is described in the paper. The detection relies on ISN generation model of original OS. The neural network learns to detect statistical deviations of ISN network packet from the ISN model. We tested the method using experimental data generated by NUSHU covert channel creation tool, which are freely available on invisiblethings.org.

1 Introduction

The general idea of covert channels relies on the idea that information can be transferred in unused fields of network protocols or it can change any non-critical data in a network protocol. Many programs for creation of covert channels are developed.

Loki2 [1] for Linux hides information in ICMP-packets and in DNS requests and responses. Reverse WWW Shell tool, developed by van Hauser [2] uses HTTP protocol. Reverse WWW Shell server creates the reversed connection to a client; it periodically "pushes" command requests and "pulls" the commands. Then the result of each command is pushed back. This method transfers data in the packet data field and they can be discovered quite easily. The use of non-critical fields of packets provides higher secrecy. Such methods are described in [3] by Craig H. Rowland. He developed Covert_TCP tool which changes IP identification, sequence number (SEQ#), and acknowledgement number (ACK#) in IP and TCP service fields.

SEQ# field is a 32-bit field, which identifies packet location in the TCP session, and denotes belonging to a certain session. Whenever a TCP session is established, SEQ# is initialized with an Initial Sequence Number (ISN) pseudorandom value and SEQ# value is being incremented with a certain value during the session. ACK# value is used to support session uniqueness by another workstation and it equals SEQ# of packets of this workstation. It is also initialized with a random ISN value during connection setting. ACK# matches SEQ# of another workstation during data transmission and it is also being incremented with a certain value [4]. Initial values of SEQ# (ACK#) are not important. An attacker can use this fact to transfer his/her information in the ISN value.

2 NUSHU, a proof-of-concept tool for TCP/IP passive channel creation

NUSHU for Linux is one of the newest tools for covert channel implementation developed by Joanna Rutkowska. It is a proof-of-concept tool for Linux using 2.4 kernel. It provides a passive covert channel (PCC), i.e. it does not generate any additional traffic, but uses data of the existing one. NUSHU was introduced on the Chaos Communication Congress in 2004. The comprehensive NUSHU description can be found in [5], NUSHU source code is also available from http://invisiblethings.org. Here we shall fix on some points, which are important for the detection of a covert channel established by NUSHU.

NUSHU creates a covert channel in the following way. Whenever a connection is established, the kernel generates ISN_{ORIG} and puts it into SEQ# of the sent packet TCP header. NUSHU is a kernel module, that is why it can change any information is the packet including its header. NUSHU puts data, which are to be passed to ISN_{COVERT}, into SEQ# field. It must be noted that ACK# should be replaced so that the original stack could maintain connections changed by NUSHU. In order to do this properly, NUSHU saves the value of $\Delta = (ISN_{COVERT} - ISN_{ORIG})$ and the packet is sent to the network afterwards. Whenever a packet containing ACK# to the packets sent earlier with the covert channel comes, NUSHU calculates $ACK\#_{ORIG} = ACK\#_{COVERT} + \Delta$ and obtains original ACK#. Connection identification is performed using sender address field, receiver address field, sender port and receiver port. NUSHU implementation is shown on Fig. 1.

Data is encrypted before putting into a TCP packet. A simple block version of Vernam algorithm with one-time pad generated with DES is used.

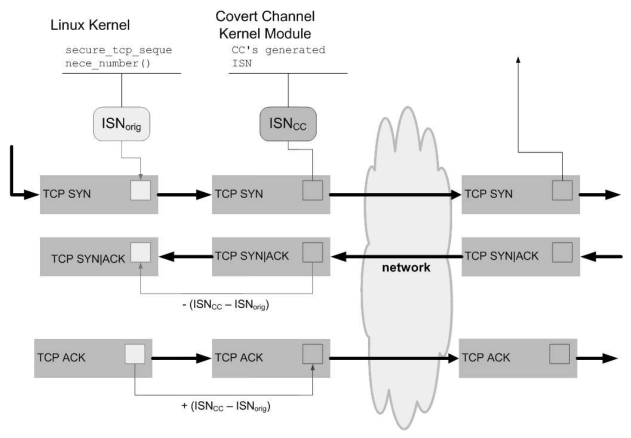


Figure 1: NUSHU implementation overview (according to [5])

Encryption is used only to make statistical characteristics of SEQ# close to a pseudo random process. This algorithm is too weak to provide secure transmission. The procedure of such encryption is shown on 2.

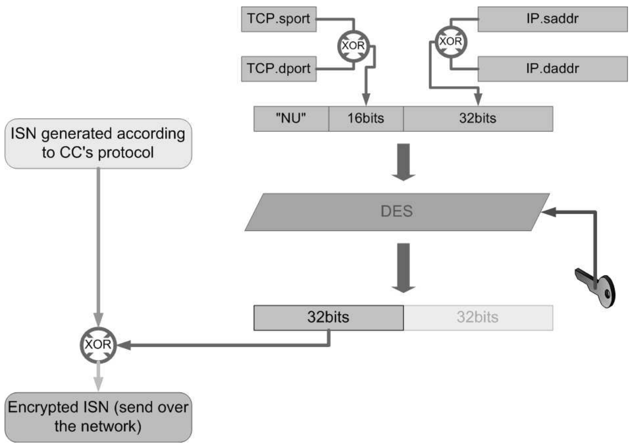

TCP.sport
TCP.dport
XOR
XOR
IP.saddr
IP.daddr
"NU"
16bits
32bits
ISN generated according to CC's protocol
DES
32bits
32bits
XOR
Encrypted ISN (send over the network)

Figure 2: Data coding before sending (according to [5])

On Fig. 2 TCP.sport and TCP.dport are the ports of sender and receiver respectively, while IP.saddr and IP.daddr are their IP addresses. The key icon indicates a private key shared by the given copy of NUSHU and its owner, who intends to read data, transferred by NUSHU. That is a new key is created each time a session is established.

In order to receive data sent by NUSHU, an intruder should have an opportunity to intercept all the traffic (or its major part) containing covert information. In [5] Joanna Rutkowska offers the following schemes of applying NUSHU:

1. A compromised workstation is connected to the Internet through a gateway controlled by the attacker. The compromised workstation initiates most of the connections (SYN-packets). In this case NUSHU puts data into SYN# of the first packet. This scheme is shown on Fig. 3.

2. A compromised server is connected to the Internet via a compromised gateway. Connections are initiated by the clients of this server. In this case NUSHU puts data into the SYN# of the second packet of the session (SYN-ACK packet). This scheme is shown on Fig. 4.

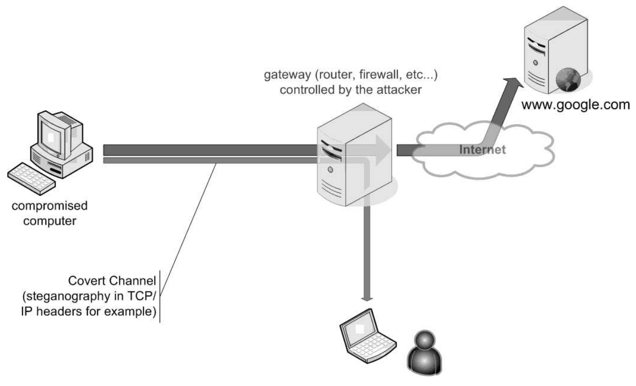

gateway (router, firewall, etc...) controlled by the attacker
www.google.com
Internet
compromised computer
Covert Channel (steganography in TCP/IP headers for example)

Figure 3: NUSHU working on a compromised workstation (according to [5])

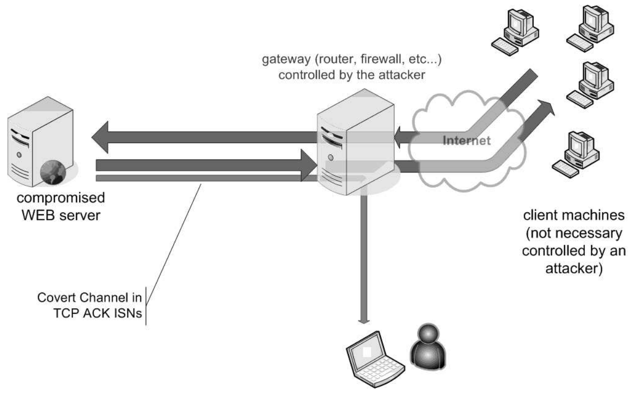

gateway (router, firewall, etc...) controlled by the attacker
Internet
compromised WEB server
client machines (not necessary controlled by an attacker)
Covert Channel in TCP ACK ISNs

Figure 4: NUSHU working on a compromised server (according to [5])

3 Covert channel detection

Joanna Rutkowska points out that there are some possibilities to discover a passive covert channel from host side, e.g. using Tripwire. Theoretically, the detection of a covert channel on the network layer is more complex. Fig. 5 depicts a typical situation of a passive covert channel detection. The sensor intercepts all the traffic of the current segment and determines the presence of a hidden channel. During the detection, it is necessary to build a classifier, which separates normal ISN from those generated by NUSHU.

Decision rule construction is possible only having the following facts taken into account. In most operating systems the kernel generates ISN using a complex function, which depends on the current time and the previous ISN value.

We offer the following way of covert channel detection. We construct a model of ISN generation using experimental ISN data of the original stack that allows prediction of the successive ISN value based on the preceding ones. There are many possibilities of building an ISN generation model, e.g. examination of the ISN generation algorithm of the operating system and an attempt to reproduce them. However we consider that the use of neural networks is the most promising approach.

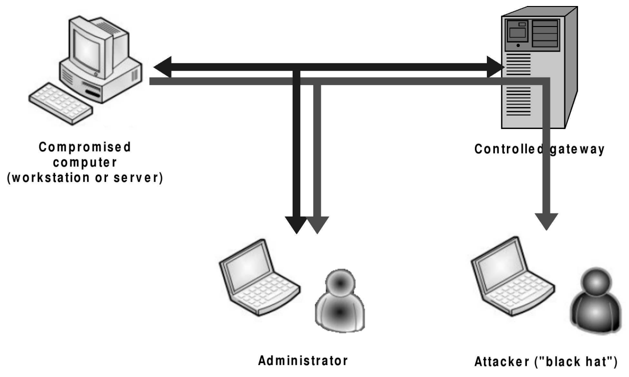


Figure 5: Detection scheme

A neural network is able to create a model using experimental data without any information about the data generation algorithm, which is a serious benefit. Neural network architectures and training methods are described in [6] for instance; we shall emphasize several basic ideas only:

- Neural networks can generalize (i.e. they can output the correct result using data which is similar to the ones used in the training set) and correct mistakes (i.e. they output the correct result if input data is corrupted or incomplete). The neural network obtains these properties during training.
- Neural network training is an iterative process with random initial parameters. The duration of this process depends on the training set and it can last quite long.

So the detection system should build an ISN generation model, in fact the system must be trained to recognize the presence of a hidden channel. In order to achieve this, ISNs generated by the standard stack are collected to form a training set.

Then the following table is constructed using collected data: $P_i = ISN_i, T_i = ISN_{i+1}, i = N - 1$, where N is the quantity of collected normal ISN. Then the neural network is trained to reproduce the following mapping: $F : P_i \overset{F}{\rightarrow} T_i, \forall i$. We used Elman neural network for our experiments. This network predicts the successive ISN using all the data received before. Neural network training is shown on Fig. 6. Besides that, we calculate similarity threshold during training. Whenever this threshold is exceeded, we consider that the tested packets do not match the normal stack model. Threshold calculation method is not detailed here.

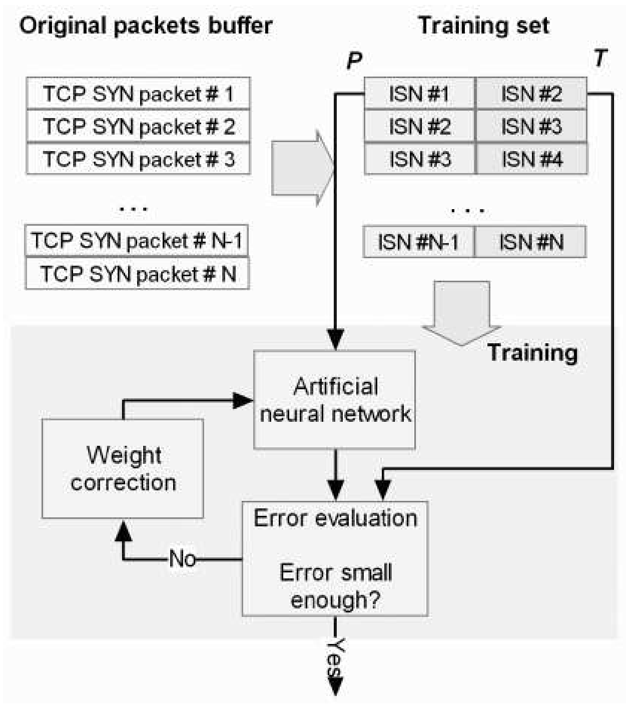


Figure 6: Neural network training

Whenever the training is completed, SYN-ACK-packets are being intercepted in the controlled network (other packets are not needed). The neural network tries to predict the successive ISN. As soon as the current ISN is also intercepted, it can be compared with the predicted value. Similarity measure between the two values characterizes how well the network data matches the constructed model.

If the difference is higher than the chosen threshold, we consider that the ISN was not generated by the original stack. Hamming distance between the two binary numbers is chosen as a similarity measure. This process is shown on Fig. 7.

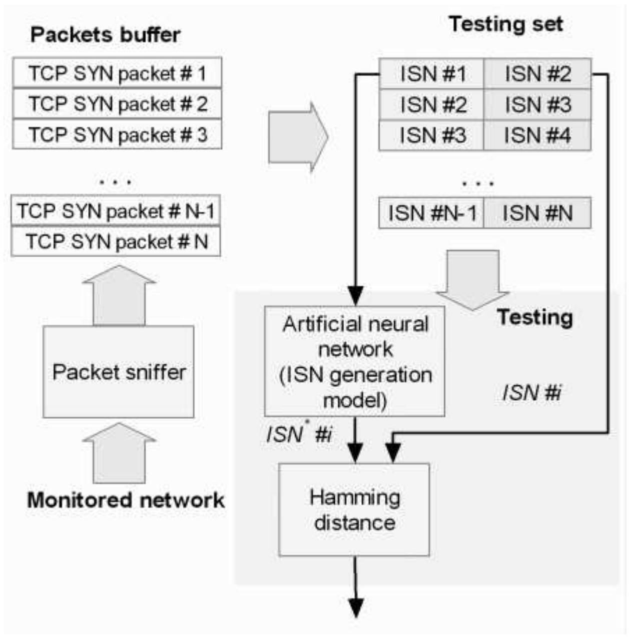


Figure 7: Similarity measure calculation

It must be noted that workstations with different operating systems can present in the same network. As soon as different parameter values are used for ISN generation by different TCP/IP stack implementations, we can state that ISN generation models should differ greatly from one operating system to another. In order to take these difference into consideration, ISN model for several (five in our experiments) operating systems is created (Fig. 8). Testing was done for all the stacks and the model is found for which the prediction error is minimal. Then we check if this error exceeds the threshold.

Unfortunately, it is impossible to detect a covert channel using a single SYN-ACK-packet. That is, the process shown on Fig. 8 is repeated several times for several sniffed packets. Similarity measure is calculated with each two consecutive packets. The more packets we try to predict, the more precise is the estimation of similarity measure of actual ISNs and their model.

4 Experimental results

In order to check the proposed concept we used the data introduced by Joanna Rutkowska as well as data collected from our computer network. The testing data are TCP-

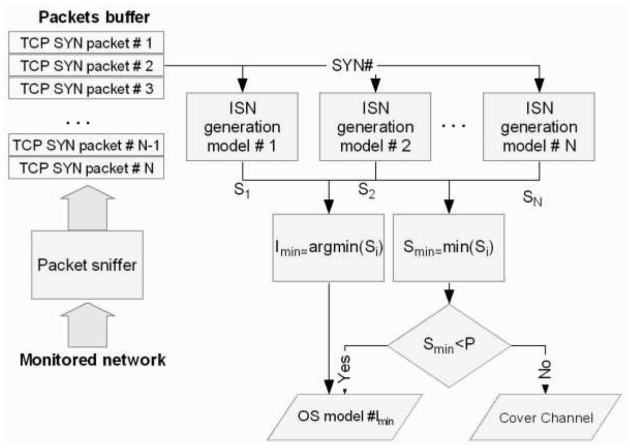


Figure 8: Passive covert channel detection

Dump logs containing SYN-ACK-packets. During experiments we use SYN-ACK-packets of the original Red Hat 9, original Fedora Core 2 and Red Hat 9 with installed NUSHU (taken from http://invisiblethings.org); the data collected by us includes Fedora Core 3, Windows 2000 SP4, Windows XP SP1, and Windows XP SP2 SYN-ACK-packets. Here is a record example:

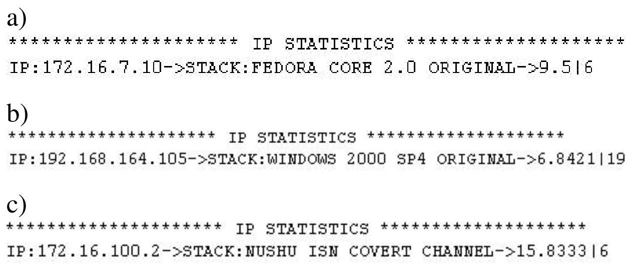
```
18:11:32.624303 172.16.7.10.1025
> local-172-16-0-1.ttn.16131: S
130110378 :130110378(0) win 5840 <mss
1460,sackOK,timestamp 5603914 0,nop,
wscale 0> (DF)
```

Only SEQ# value is needed for the experiments (marked bold). Logs contain 20,000 records of each type. We used only 1,500 records for neural network training; remaining data were used for testing.

During model generation stage we read all the 1,500 values for each operating system and then we use Mathworks Matlab 6.5 R13 script to create and train the neural network. Elman recurrent network was chosen because we suppose that in order to predict a successive ISN value, information about all the preceding ones is needed. Each neural network is saved into its own file after training.

On monitoring stage we use Matlab script which reads packets from a WinDump log (or any other specified file) one by one. It is needed for online network monitoring. Fig. 9a shows output monitoring-script for original Fedora 2, Fig. 9b shows output monitoring-script for original Windows 2000 SP4, and Fig. 9c shows output monitoring-script for Fedora with NUSHU.

On Fig. 9 the first column (IP) contains source IP-addresses of the monitored packets; the second one (STACK) is a predicted stack; the third one is a prediction error (0 .. 32); and the last one denotes the numbers of tested packets from the same IP-addresses. A short review of script creation is denoted in Table 1.



a)
```
********************* IP STATISTICS ********************
IP:172.16.7.10->STACK:FEDORA CORE 2.0 ORIGINAL->9.5|6
```
b)
```
******************** IP STATISTICS *******************
IP:192.168.164.105->STACK:WINDOWS 2000 SP4 ORIGINAL->6.8421|19
```
c)
```
******************** IP STATISTICS *******************
IP:172.16.100.2->STACK:NUSHU ISN COVERT CHANNEL->15.8333|6
```

Figure 9: Monitoring script output: a) original Fedora Core 2.0; b) original Windows 2000 SP4; c) NUSHU covert channel

Elman network size	Number of epochs	Creation time, s
30x32	1000	$8 - 10$
40x32	1000	$15 - 20$
50x32	1000	$20 - 30$
100x32	1000	$40 - 60$

Table 1: Model creation (using PC AMD Athlon XP 1800+, RAM 512)

Table 2 summarizes testing results.

Elman network size	False channel detection	False channel missing	OS detection error [1]
30x32	$< 0.1\%$	5%	$< 1\%$
40x32	$< 0.2\%$	7%	$< 2\%$
50x32	$< 0.2\%$	8%	$< 3\%$
100x32	$< 0.5\%$	10%	$< 4\%$

Table 2: Monitoring script accuracy

Monitoring script performance allows to implement online monitoring of 10 Mbit network with 50% bandwidth traffic. Table 3 summarizes testing results for the optimal Elman network size (30x32).

As mentioned above, if the quantity of ISN is small, we cannot guarantee accuracy of the decision. If ISN is smaller than 3, the system cannot detect the stack. According to our experiments, the probability of NUSHU detection for Windows XP SP2 reaches 60%. This fact occurs because the ISN generation mechanism of this OS is very close to uniform random distribution and thus it resembles NUSHU generation method. Monitoring script performance allows monitoring a 10Mbit network with 50% bandwidth traffic online.

[1]The results are shown for four operating systems only, namely Red Hat 9, Fedora Core 3, Windows 2000 SP4, Windows XP SP1. Windows XP SP2 is not taken into account.

Quantity of ISN	< 3	< 50	< 100	> 100
False channel detection	50%	5%	0.1%	$< 0.1\%$
False channel missing	$50\% url$	8%	6%	5%
OS detection error [1]	50%	5%	1%	$< 1\%$

Table 3: Dependence of recognition accuracy on the quantity of ISNs in the training set

5 Discussion

Experiments demonstrate that our method allows to detect a passive covert channel with high precision. Besides that, the method can automatically create ISN generation models for any operating system (it is not necessary to know how the OS creates ISNs). Another useful side effect of the model is the ability to determine an operating system working in a network.

The following strategy of applying our method is possible. One should install a sensor (e.g. a computer with WinDump running on it) so that it could have a chance to intercept all the connection of the controlled network. Our detector should receive logs of the sensor (or several sensors). The system can be trained when network load is minimal (e.g. at nighttime). Retraining is recommended to be done regularly to maintain the system up-to-date in case any new operating systems or patches are installed (if they change the ISN generation principle).

The most important problem for further research is the detection of covert channels with acyclic connection establishment. It is known that many operating systems generate ISN depending on the current time that is on the interval between the current connection and the previous one. At the moment we used data of both original and compromised machines, which had established connections regularly after a very short period each time. We do not know how the proposed method works with irregular connections.

Beside that, our further research will include the development of covert channel detection program. Now we use standard TCPDump and a Matlab script, which is less productive than a software product. The development of a program, which captures packets and detects covert channels itself should increase the overall performance.

6 Acknowledgments

The authors (and the rest of the I-409 Labs) would like to express their thanks to Joanna Rutkowska for helpful contributions.

References

[1] Daemon9. LOKI2 implementation. *Phrack Magazine*, 7, September 1997. Available: http://www.phrack.org/show.php?p=51&a=6.

[2] van Hauser. Placing backdoors through firewalls. Available: http://www.thc.org/papers/fw-backd.htm.

[3] Craig H. Rowland. Covert channels in the TCP/IP protocol suite. Available: http://firstmonday.org/issues/issue2_5/rowland/index.html.

[4] RFC793: Transmission control protocol. Available: http://www.rfc.net/rfc793.html.

[5] Joanna Rutkowska. The implementation of passive covert channels in the Linux kernel. Available: http://invisiblethings.org/papers.html.

[6] Simon Haykin. *Neural Networks. A Comprehensive Foundation*. Prentice Hall, New Jersey, USA, 2nd edition, 1999.