Hiding the SSH port via smart Port Knocking

Antonios S. Andreatos

***Abstract*—**A good security practice in web servers is to de-activate services when not needed in order to reduce the so-called attack surface. For this purpose, port-knock software has been developed; port-knock software hides a specific port (in our case, the SSH port) until a specific port sequence happens. A port-knock software for Linux servers is knockd. A knockd server listens to all traffic on an Ethernet interface, looking for special knock sequences of port-hits. This paper proposes an advanced port-knocking mechanism based on a Pseudo-Random Number Generator and a Chaotic Random Number Generator which produces proper pseudo-random knock sequences. The proposed mechanism produces different sequences each time and is highly parameterizable.

***Keywords*—**Hénon map, knockd, knock sequences, pseudo-random number generator, port-knocking.

I. Introduction

A popular protocol for connecting to servers remotely is Secure Shell, or SSH. SSH is a cryptographic network protocol operating at the application layer to allow remote login and other network services to operate securely over an unsecured channel such as the one provided by the public Internet [1], [2]. A good security practice is to de-activate services when not needed in order to reduce the so-called attack surface. For this purpose, port-knock software has been developed; port-knock software hides a specific port (here, the SSH port) until a specific port sequence happens. A popular port-knock software for Linux servers is knockd [3]. A knockd server listens to all traffic on an Ethernet interface, looking for special "knock" sequences of port-hits until a specific sequence appears. A client produces these port-hits by sending TCP and/or UDP packets to specific ports on the server. These ports need not be open; since knockd listens at the link-layer level, it sees all traffic, even that destined for a closed port. When the server detects a specific sequence of port-hits, it runs a command defined in knockd configuration file, in order to open up the SSH port (typically, no. 22) in a firewall for quick access [4], [5]. In knockd.conf the user can also specify a timeout interval (after which the SSH port will automatically close), as well as, a sequence of port-hits for closing and hiding the port again. For detailed information about the iptables firewall configuration the reader is referred to the bibliography [4], [6]. Figure 1 demonstrates an example of a port-knocking sequence using TCP and UDP.

If the knock sequence is invariant, sniffing may uncover the secret sequence [6]; hence, the security of this scheme is low. A better way is to use a pseudo-random number generator (PRNG) to produce a varying port sequence [7]. However, the produced sequence will be the same each time the system re-starts; another possible weakness is that if the generator has a small period, the pseudo-random sequence may be revealed, hence the system is not safe enough. In order to solve this problem, a powerful PRNG could be used.

13821:tcp, 7803:udp, 19552:tcp, 35813:udp, 54926:udp

Fig. 1 Example port-knocking sequence

This paper proposes an advanced port-knocking mechanism based on two totally uncorrelated PRNGs, which produces different sequences each time. The proposed mechanism meets the following requirements:

a) It generates a series of pseudo-random port numbers and/or protocols (TCP or UDP) and writes them in a special file; the produced pseudo-random is different in each run because it uses a function of the system time as a seed;

b) The produced port numbers fall in the upper port range (e.g., 3000-65000) and the range limits may be configured by the user;

c) The same algorithm runs at both the client and the server;

d) It modifies the server's knockd configuration file to get the secret port-knocking sequence from the above special file;

e) It generates a new port-knocking sequence after each SSH connection at the client and the server.

The proposed mechanism has been simulated using a server installed on a virtual machine.

An additional security action would be the use a higher port number for the SSH port instead of the default one (22).

II. The PRN Generators

In order to produce the port-knocking sequence we need two things: a) a series of random but valid port numbers (i.e., from 1024 to 65535) and a random series of protocols (either TCP or UDP) of the same length.

For best results we should use a pseudo-random number generator (PRNG) with good characteristics. Special tests have been devised for assessing the quality of random and pseudo-random number sequences [8], [9]. The two random sequences are totally uncorrelated, thus increasing the level of security.

A. Python's PRNG

For the port numbers Python's PRNG function has been used. Python's PRNG functions produce satisfactory results as the test results indicate (e.g. entropy = 0.99962). Figure 2 presents a histogram of a PRN sequence consisting of 65536 numbers.

A. Andreatos is with the Div. of Computer Engineering & Information Science, Hellenic Air Force Academy, Dekeleia Air Force Base, Dekeleia, Attica, Greece (e-mail: aandreatos@hafa.haf.gr, aandreatos@gmail.com).

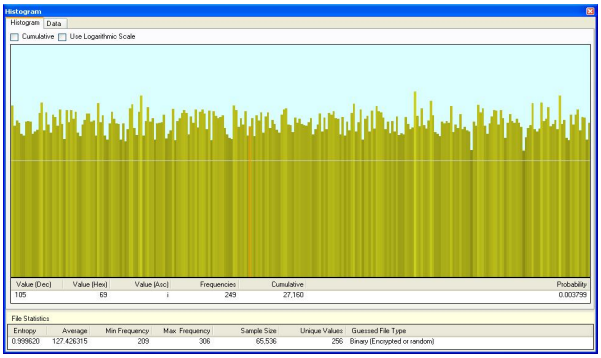

Fig. 2 A histogram of 65536 pseudo-random numbers

Python code includes a function of current system time as a seed in the PRNG so that each time it produces a different sequence. This function takes into account the least significant digits as well as the decimal digits which change continuously.

B. Hénon chaotic RNG

For the protocols (TCP or UDP) a chaotic RNG based on Hénon's chaotic map has been used [10], [11].

The Hénon map is produced by the solution of two coupled first-order differential equations:

$$x' = 1-ax^2+y \quad (1)$$
$$y' = bx \quad (2)$$

where a, b are constant parameters. Typical values are: a = 1.4 and b = 0.3.

For numerical solution, the following set of difference equations is used:

$$x(i+1) = y(i) + 1 - ax(i)^2 \quad (3)$$
$$y(i+1) = bx(i) \quad (4)$$

with initial conditions [x(0), y(0)] = [0, 0].

The set of these two equations was solved using a Python script. For a=1.4 and b=0.3 the produced (x,y) pairs when plotted form a characteristic shape shown in Figure 3:

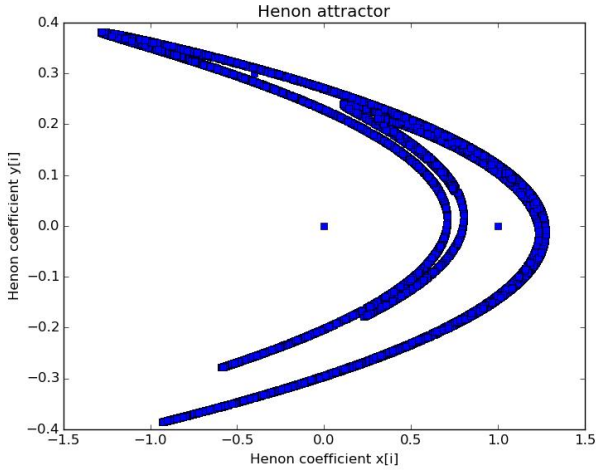

Fig. 3 Hénon's attractor for 20000 pairs

Figure 4 demonstrates the Hénon's variables versus time produced by the same Python script.

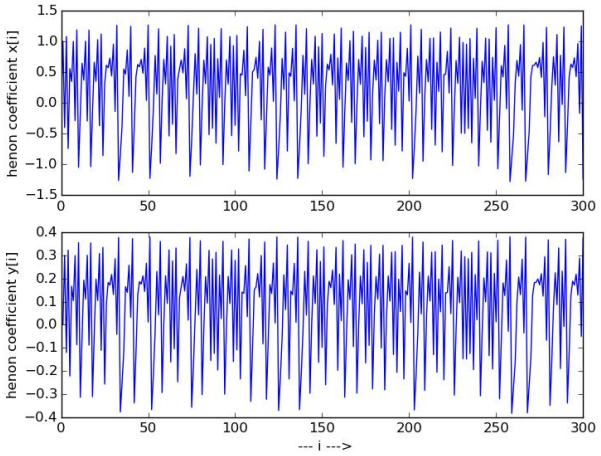

Fig. 4 Hénon's variables for 300 time steps

By sampling the x waveform using a proper threshold we get a binary vector; this vector is then used to produce the series of protocols that will be used in the port-knocking sequence. This binary waveform is quasi uniform, and is used to define the protocol that will be used in combination with a port number. Thus, that the probability of a port being TCP or UDP is about 50%.

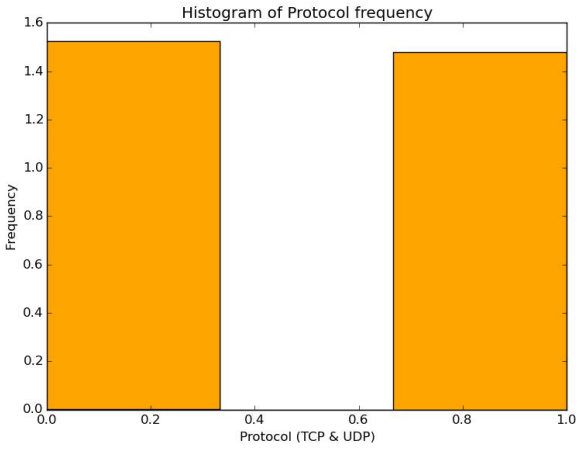

Fig. 5 A histogram of the protocol frequency

C. Combining the two sequences

A third Python script reads the two random sequences and produces a series of port-knocking sequences, containing a random but reasonable number of (port, protocol) pairs (e.g., from three to six). The result is stored in a sequence file.

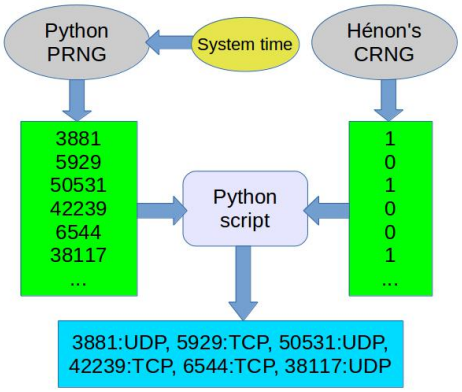

Fig. 6 Combining the two sequences

An example sequence produced by that script is shown below (Fig. 7).

```
11045:tcp,61405:udp,39673:tcp
20583:udp,8382:tcp,1353:udp,1717:tcp,21917:udp
12574:tcp,38441:tcp,7583:udp
42271:udp,49594:tcp,35510:udp,58555:tcp,10900:udp
14954:tcp,53270:tcp,49274:udp,4133:udp,61935:udp
29362:udp,45912:udp,16999:tcp,4574:udp
```

Fig. 7 Example knockd output of the proposed scheme

D. *Transferring the sequence file*

How does the server get the knock sequence file? At least three possible solutions exist.

a) One alternative is to use secure copy (scp) – channel using cryptography [7].

b) A second alternative is to transfer the file using a cryptographic protocol such as secure ftp (sftp) [2] or SSHFS [12], [13].

c) Another alternative is to use the same algorithm (code) in both the server and the client. In this approach we must ensure that the two sequences are synchronized. This might not be the case if a different seed is used in each machine.

III. Simulating the Mechanism

We have used virtualisation in order to simulate the use of the proposed mechanism. The client is implemented on a PC whereas the server is implemented as a virtual machine (VM) running on that PC using VirtualBox.

The real and the virtual machines are connected on a virtual LAN. The virtual machine runs Ubuntu 12.04 Server software. The procedure is as follows (see Fig. 8):

a) The nmap software has been used by the client to discover the server's open ports and services. Initially, only the HTTP port (80) is open while the SSH port (22) is closed.

b) knockd runs on the client and produces a secret sequence of port numbers and protocols.

c) nmap runs again and now we notice that two server ports are open.

```
File  Edit  View  Search  Terminal  Help
antony@N5110:~$ nmap 192.168.1.7

Starting Nmap 6.47 ( http://nmap.org ) at 2017-04-07 12:33 EEST
Nmap scan report for ubuntu (192.168.1.7)
Host is up (0.00060s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 5.28 seconds
antony@N5110:~$ knock 192.168.1.7 11045:tcp 61405:udp 39673:tcp 20583:udp 8382:tcp 1353:udp
antony@N5110:~$ nmap 192.168.1.7

Starting Nmap 6.47 ( http://nmap.org ) at 2017-04-07 12:33 EEST
Nmap scan report for ubuntu (192.168.1.7)
Host is up (0.00069s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 5.78 seconds
antony@N5110:~$
```

Fig. 8 A secret random knockd sequence opens the SSH port

Now the client can connect to the server using SSH and perform any administrative actions.

d) After that, the client runs knockd again, making the server's ssh port to close (Fig. 9).

```
File  Edit  View  Search  Terminal  Help
antony@N5110:~$ nmap 192.168.1.7

Starting Nmap 6.47 ( http://nmap.org ) at 2017-04-07 12:03 EEST
Nmap scan report for ubuntu (192.168.1.7)
Host is up (0.00093s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 5.79 seconds
antony@N5110:~$ knock 192.168.1.7 9123 8451 7048
antony@N5110:~$ nmap 192.168.1.7

Starting Nmap 6.47 ( http://nmap.org ) at 2017-04-07 12:03 EEST
Nmap scan report for ubuntu (192.168.1.7)
Host is up (0.00069s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 5.48 seconds
antony@N5110:~$
```

Fig. 9 A secret knockd sequence closes the SSH port

IV. Conclusions

In this paper a smart port-knocking application for Linux servers has been presented. Using knockd in combination with a smart mechanism consisting of a PRNG and a CRNG we can increase server security, first because the attack surface is reduced and second because the port-knocking sequence is difficult to guess.

The application has been simulated using virtualisation. Python programming language has been used to produce the port-knocking sequences.

Two different and totally independent RNGs have been used: a PRNG based on a Python function produces the port numbers and a Chaotic RNG based on Hénon's map produces the port sequence (TCP or UDP). Another Python script combines the series of random port number and protocol pairs into a file containing port-knocking sequences of variable length. For increased security a custom function has been implemented

which takes system time into account and produces a different seed each time. In this way we avoid playback attacks.

The proposed system is highly parameterizable for increased security:

a) The custom function producing the seed of the PRNG is parameterizable;

b) The port range limits may be configured by the user;

c) By changing the coefficients a and b of the CRNG as well as the threshold we get different results;

d) The number of knocks is also variable.

Future work plans include the use of powerful PRNGs (such as Mersenne Twister [14]), True RNGs or different CRNGs such as those based on Chua's circuit [15], [16].

REFERENCES

[1] RFC 4251, "The Secure Shell (SSH) Protocol Architecture", Network Working Group of the IETF, January 2006.

[2] A. S. Andreatos, "SSH Tutorial", Oct. 2011. Available: https://www.merlot.org/merlot/viewMaterial.htm?id=593816.

[3] knockd, Linux man page. Available: http://linux.die.net/man/1/knockd.

[4] "How To Use Port Knocking to Hide your SSH Daemon from Attackers on Ubuntu". Available: https://www.digitalocean.com/community/tutorials/how-to-use-port-knocking-to-hide-your-ssh-daemon-from-attackers-on-ubuntu.

[5] knockd - a port-knocking server. Available: http://www.zeroflux.org/projects/knock.

[6] P. Varelas, Port knocking part 1 - "Listen to your own door". Delta Hacker magazine, no. 32, pp. 24-32, May 2014 (in Greek).

[7] P. Varelas, Port knocking part 2 - "Let them eavesdrop". Delta Hacker magazine, no. 32, pp. 70-78, May 2014 (in Greek).

[8] A. S. Andreatos and A. P. Leros, "A comparison of random number sequences for image encryption", in Proc. MMCTSE, Mathematical Methods & Computational Techniques in Science & Engineering, Athens, Greece, November 28-30, 2014, pp. 146-151. ISBN: 978-1-61804-256-9.

[9] A. S. Andreatos and A. P. Leros, "Random number sequences assessment for image encryption", International Journal of Applied Mathematics and Informatics (http://naun.org/cms.action?id=10193). ISSN: 2074-1278. Volume 9, 2015, pp. 14-22. Available: http://www.naun.org/main/ Upress/ami/2015/a062013-125.pdf.

[10] M. Hénon, "A Two-dimensional Mapping with a Strange Attractor". Commun. Mathematical Physics 50, pp. 69-77 (1976), Springer-Verlag.

[11] W. F. H. Al-Shameri, "Dynamical Properties of the Hénon Mapping", *Int. Journal of Math. Analysis*, vol. 6, 2012, no. 49, pp. 2419-2430.

[12] https://en.wikipedia.org/wiki/SSHFS.

[13] https://github.com/libfuse/sshfs.

[14] Mersenne twister. Available: https://en.wikipedia.org/wiki/Mersenne_Twister.

[15] A. Leros and A. Andreatos, "A Steganography Telecom System Based on a Chua Circuit Chaotic Noise Generator", *Chaotic Modeling and Simulation Journal*, January 2013, pp. 199-208.

[16] C. K. Volos and A. S. Andreatos, "Secure Text Encryption Based on Hardware Chaotic Noise Generator", *Journal of Applied Mathematics and Bioiformatics*, vol. 5, issue 3, 2014, pp. 15-35. http://www.scienpress.com/Upload/JAMB/Vol%205_3_2.pdf.